DOI: 10.1111/cgf.70235
Pacific Graphics 2025
M. Christie, N. Pietroni, and Y.-S. Wang
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 7

Projective Displacement Mapping for Ray Traced Editable Surfaces

Rama Hoetzlein
Quanta Sciences, Ithaca, New York

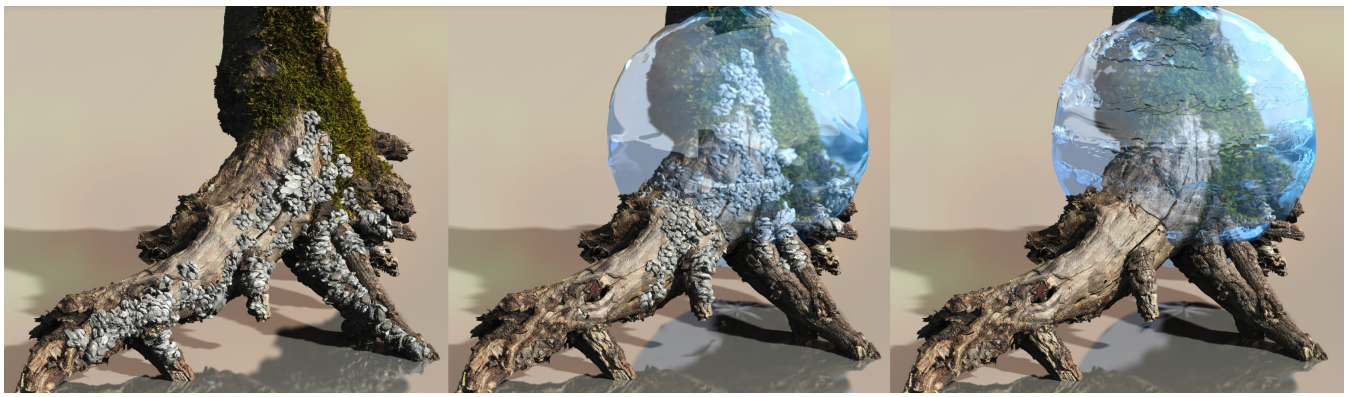

Figure 1: *Detailed geometry is interactively edited and ray traced with our technique. A base mesh of 7710 triangles with a 16-bit,* 4096^2 *displacement map is ray traced at 3460x1024 in 28 milliseconds for one sample including primary rays, path tracing, reflection, refraction and shadows. Interactive sculpting of the displacement map while ray tracing supports a) sculpting rocks over a tree stump, b) sculpting while looking through a refractive object, and c) modeling the surface of the refractive object itself.*

Abstract

Displacement mapping is an important tool for modeling detailed geometric features. We explore the problem of authoring complex surfaces while ray tracing interactively. Current techniques for ray tracing displaced surfaces rely on acceleration structures that require dynamic rebuilding when edited. These techniques are typically used for massive static scenes or the compression of detailed source assets. Our interest lies in modeling and look development of artistic features with real-time ray tracing. We introduce projective displacement mapping as a direct sampling method combined with a hardware BVH. Quality and performance are improved over existing methods with smoothed displaced normals, thin feature sampling, tight prism bounds and ray bi-linear patch intersections.

CCS Concepts

• ***Computing methodologies*** → ***Ray tracing; Parametric curve and surface models; Mesh models;***

1. Introduction

Displacement maps enable artists to add rich surface detail to 3D models without requiring high-resolution geometry. This approach is especially useful when working with limited resources or compressed assets. We focus on real-time look development of displacement-mapped surfaces over arbitrary base meshes, with ray traced reflections, refractions and global illumination as import visual cues during editing.

Early interactive systems such as 3D Paint offered editing without real-time updates [Wil90]. Volumetric distance functions [PKZ04] [RMD11], terrain editing [Yus12], surface collisions [NMMC14], or subdivision surfaces [BMBZ02], could be interactively editing in limited contexts or with specialized non-polygonal rendering. Our goal is to enable interactive editing of fine details over arbitrary base meshes.

We target high-resolution authoring of surface detail on low-poly meshes with real-time ray tracing during editing. Many existing techniques, such as parallax, relief or cone mapping [Bli78]

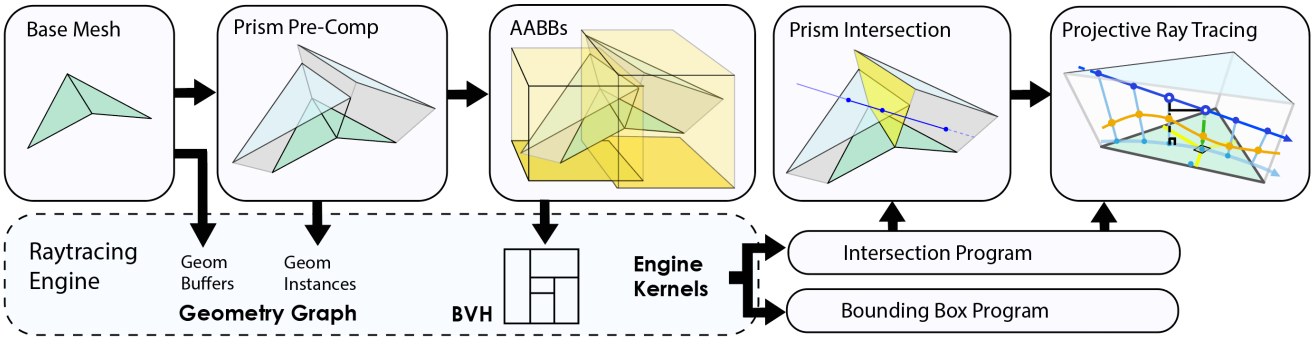

Figure 2: *Method overview. Our technique starts with a low-poly base mesh, from which we pre-compute parallel offset prisms with positive and negative extents. Prisms and their AABBs are used to build the geometry graph and hardware-accelerated BVH. A custom program implements prism intersection and the Projective Displacement algorithm described in text.*

[KTI*01] [OBM00], interval-based tracing [LG90] [WSC02] [TBS*21], and tessellation [SB87] [JH13] [NL13] require pre-processing or costly BVH updates which we seek to avoid. Instead, we consider a tessellation-free, direct sampling approach that balances quality, simplicity and real-time performance.

2. Related Work

Parallax mapping renders detail by offsetting texture samples based on parallax (shifted occlusion) [KTI*01]. *Parallax occlusion mapping* iteratively samples a height field to ensure correct depth with occlusion shadows [Tat05]. *Prism parallax occlusion* mapping enhances this with correct silhouettes [DT07]. *Relief maps* encode the displaced surface as an additional depth channel. Image-warping relief maps [OBM00] traverse arbitrary base meshes [PO07]. Quadric surfaces use ray-quadric intersection before relief stepping [OP05]. Intended for raster pipelines, these methods do not utilize BVHs or support general ray tracing.

Cone step mapping precomputes a cone map to skip features based on occlusion angle [DW05] [Dum06], employing binary search [PO07], or using a min-max texture [LTT09]. These methods are view-dependent, making them unsuitable for ray tracing arbitrary base meshes.

Curved rays map the 3D volume of the displaced surface to nonlinear texture space [Kaj83]. Tangent-space methods compute the curved mapping between world and texture space, approximating curved rays with piecewise segments [CC08]. Ogaki develops nonlinear ray tracing by solving cubic equations [Oga23]. Rays are transformed at the leaf nodes (prisms) of the broad-phase BVH, where they become nonlinear in rectified canonical space (see Fig. 3, top right). The pre-computation of data structures and solving cubic equations preclude our goals for interactive editing.

Parametric surfaces may be exactly evaluated through root-finding with Newton's method [Tot85] [JB86]. The intervals must be chosen to locally bound the surface at minima and maxima. Lischinski et al. construct a tree structure to efficiently traverse the surface [LG90]. Cached traversal trees enable fast ray tracing by sharing information with neighboring rays [LEF91].

Affine intervals provide better local bounds than min-max intervals [KHK*09]. Thonat et al. construct a D-BVH as a min-max mipmap over affine intervals [TBS*21]. Displaced bounds are generated dynamically from quad-tree nodes, requiring significant computation for intersection. Recent work improves performance with RMIP, a hierarchical bounding structure [TGBB23]. While rectangular regions in RMIP allow for acceleration of anisotropic ray sampling, traversal of D-BVH and RMIP can be costly per pixel.

Shell mapping constructs a bijective map between texture and shell space - the region between two offset surfaces - for arbitrary meso-geometry [PBFJ05]. Real-time rendering is achieved by Ristche et al. utilizing a distance map to accelerate ray tracing within the shell space [Rit06]. *Curved shell mapping* removes discontinuities by modeling Coons patches within each prism at the cost of solving a cubic equation per ray step [JMW07]. Other methods use novel encodings for intersections such as singular value decomposition [WWT*03] or neural networks [KMX*21]. These methods require considerable pre-processing.

Tessellation is an intuitive approach to displacement mapping [Coo84]. Snyder et al. accelerate fully tessellated surfaces with uniform spatial grids [SB87]. Local micro-tessellation is more efficient with geometry caching of sub-triangles [PH96]. *Adaptive hardware tessellation* was introduced as a shader pipeline stage with DirectX 11 [Mic09] enabling real-time adaptive displacement and higher-order surfaces [NKF*16]. Tessellation hardware can adaptively render displacement-mapped surfaces [JH12] [JH13]. However, non-view facing rays are not easily handled with raster-based tessellation.

Higher order surfaces are commonly tessellated, such as Bézier patches [MHTAM10] [CABD11] [TFM15], Catmull-Clark surfaces [NL13], Gregory patches [LSNCn09], and Loop subdivision surfaces [SL03] [LMH00] [AFF12]. These surfaces provide smoothness guarantees with C^1 or C^2 continuity. We explore smoothness without using an intermediate C^1 surface.

Locally adaptive tessellation is a class of methods whereby tessellation is applied to selectively detail a mesh. Nanite introduced virtual geometry to compress source meshes into triangle clusters

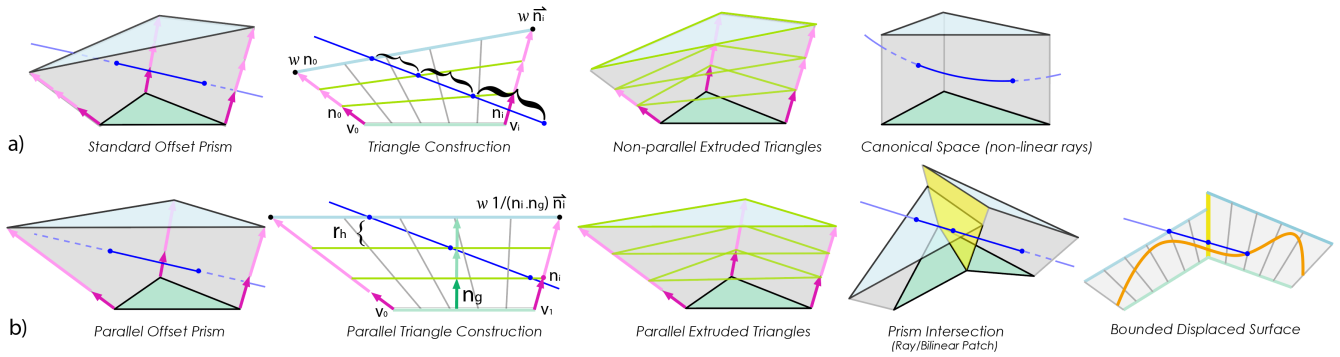

Figure 3: *Prism construction. a) Typical offset construction is based on multiples of the vertex normals which results in non-parallel extruded triangles and a non-linear sample space (brackets). One approach to handle this is by converting to canonical space in which rays are non-linear. b) Our method constructs a prism volume with parallel offsets by adjusting the normal length relative to the geometric normal resulting in parallel extruded triangles and a linear sample space. Rays intersect a prism at either triangles or bilinear patches.*

which are rendered as regional levels of detail on-the-fly [KSG21]. Triangle clusters may be inserted into a GPU-accelerated BVH per frame for ray tracing [BP23]. Haydel et al. refines each triangle through an adaptive one-to-four triangle subdivision [HYS23]. Micro-Meshes [Nvi23] is an API for locally adaptive subdivision using barycentric maps, which can be generated from displacement maps. We examine the rendering quality and performance of Micro-Meshes in comparison to our approach.

3. Overview

3.1. Motivation

Our goal is the interactive editing of complex details over meshes while performing real-time ray tracing. We deliberately avoid approaches that require compression or streaming of highly detailed source assets or frequent data structure rebuilding. Methods are considered for the authoring of displacement-mapped surfaces with a minimal memory footprint, relying on a low-poly mesh and a high-resolution depth texture, while addressing the performance constraints of real-time editing.

3.2. Algorithm Design

Efficient ray marching is challenging since equally spaced world samples are non-linear in texture space (Fig. 3a). Among the previous direct sampling methods only Prism Parallax Occlusion mapping (PPOM) produces correct silhouettes - and achieves this with offset prisms - yet has difficulty with ambient occlusion in the raster pipeline [DT07]. We construct parallel offset prisms such that world-space samples can be *linearly projected* to texture space (Fig. 3b). Our contributions include:

- Projective displacement mapping for efficient local intersections
- Parallel offset prisms for linearly projected sampling
- Ray-bilinear patch intersections to accelerate prism hits
- Optimized entry/exit sampling loop conditions
- Smoothed displaced normals for C^1 smoothness.
- Thin feature sampling to retain fine details.

4. Prism Construction

Prisms typically used in displacement mapping are constructed by offsetting base triangles along vertex normals. Given a base triangle in barycentric coordinates $P(u,v) \in \mathbf{R}^3$, with $0 \leq u,v \leq 1$, a classical offset prism $\mathbf{R}_{std}$ is defined by:

$$\mathbf{P}(u,v) = uV_0 + vV_1 + (1-u-v)V_2, \tag{1}$$

$$\mathbf{N}'(u,v) = uN_0 + vN_1 + (1-u-v)N_2, \tag{2}$$

$$\mathbf{R}_{std}(u,v,w) = \mathbf{P}(u,v) + w\mathbf{N}'(u,v) \tag{3}$$

The maximum extent is w_{max} at the offset triangle $\mathbf{P}(u,v,w_{max})$. In Figure 3a, offset triangles in classical prisms may not be parallel to the base triangle since the unit normals point in different directions. The projection of a ray in this volume to texture space is non-linear. Some ray tracing methods transform this irregular prism to canonical space where the rays are nonlinear [CC08] [Oga23], Figure 3a (top right).

We introduce *parallel offset prisms*, $\mathbf{R}_{pop}$, whose normal directions are identical, yet whose offset triangles are all parallel to the base triangle, Figure 3b.

$$n_f(u,v) = u\frac{1}{N_0 \cdot N_g} + v\frac{1}{N_1 \cdot N_g} + (1-u-v)\frac{1}{N_2 \cdot N_g} \tag{4}$$

$$\mathbf{R}_{pop}(u,v,w) = \mathbf{P}(u,v) + wn_f(u,v)\mathbf{N}'(u,v) \tag{5}$$

The normal factor $n_f(u,v)$ transforms the space to an orthogonal one where w can now be interpreted as distance along the geometric normal N_g. These triangles are parallel but not similar triangles, since the arbitrary normal directions still shift the vertices laterally. Maximum prism extents e_i are computed at the vertices as:

$$e_i = v_i + w_{max}\frac{1}{N_i \cdot N_g}N_i \tag{6}$$

A displaced surface $\mathbf{S}$ is defined by a depth function $D(u,v)$ over the base triangle as:

$$\mathbf{S}(u,v) = \mathbf{P}(u,v) + D(u,v)\mathbf{N}'(u,v) \tag{7}$$

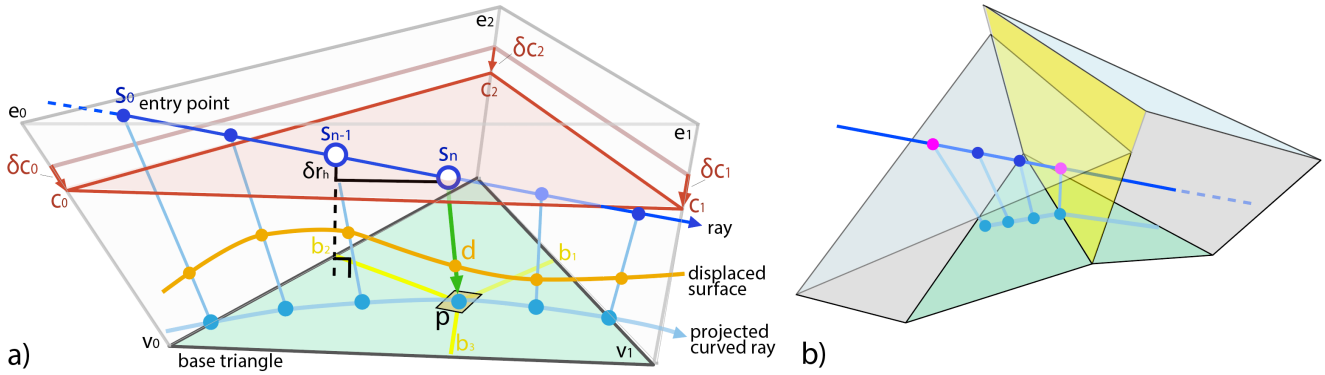

Figure 4: *Projective displacement mapping. a) Surface intersection proceeds along a ray while simultaneously tracking a parallel offset triangle (scanning triangle), c_i, where the barycentric projection of a sample s_n gives the displacement texture location d. See text for details. b) When a ray enters or exits a prism it intersects at either a triangle top/bottom or a bilinear patch. Prism watertightness is achieved by ensuring sample continuity across these boundaries.*

Displacement is commonly formulated by constructing $\mathbf{S}$ as an offset along $\mathbf{N}'$, the interpolated (shading) normal at each point [Coo84] [LMH00] [NL13]. We observe that $D(u,v)$ replaces the parametric scalars in Eqn. 3 and 4. This implies that any bounding prism constructed as a scalar multiple along the same normals will result in identical displaced surfaces, so long as $D(u,v) < w_{max} n_f(u,v)$.

More importantly, within a parallel prism, changes in ray height r_h, in Fig. 4a, can be evaluated *linearly* along N_g. This enables our projective displacement mapping (PDM) technique. The PDM algorithm and its optimizations are described in section 4.3.

4.1. Prism Intersection

The interface between two prisms is a bilinear patch since adjacent vertex normals v_i may not be co-planar [RPH04] [Res19]. The use of interpolated normals permits C^1 continuity internal to prisms and C^0 continuity between prisms [JMW07]. Prism-prism interfaces may be deconstructed into two triangles, where a consistent diagonal is selected to avoid artifacts [HEGD04] [DT07]. We take a different approach and directly ray-trace the prism sides as bilinear patches.

Direct ray tracing of bilinear patches has several benefits. First, the interface between two sides can support C^1 continuity. Second, diagonal selection is not needed. Third, rather than tracing the six exterior faces we can trace three bilinear patch primitives directly. Finally, efficient raytracing of bilinear patches on GPU hardware is now possible [Res19]. The loop condition of our primary algorithm requires both entry and exit t-values which can be computed more quickly together. Given n_p as the normal at the patch hit point:

$$\begin{aligned} t_{max} &= max(t, t_{max}) \textbf{ iff } n_p \cdot r_{dir} \geq 0 \\ t_{min} &= min(t, t_{min}) \textbf{ iff } n_p \cdot r_{dir} < 0 \end{aligned} \tag{8}$$

Rays may enter a bilinear patch side, through the top triangle of a prism, or start inside the prism. Top faces are handled with an extra ray/triangle test. We can support rays which start inside the prism by observing when t_{max} is set but t_{min} is not. These values are updated similarly for the three bilinear patch sides and the top and bottom triangles.

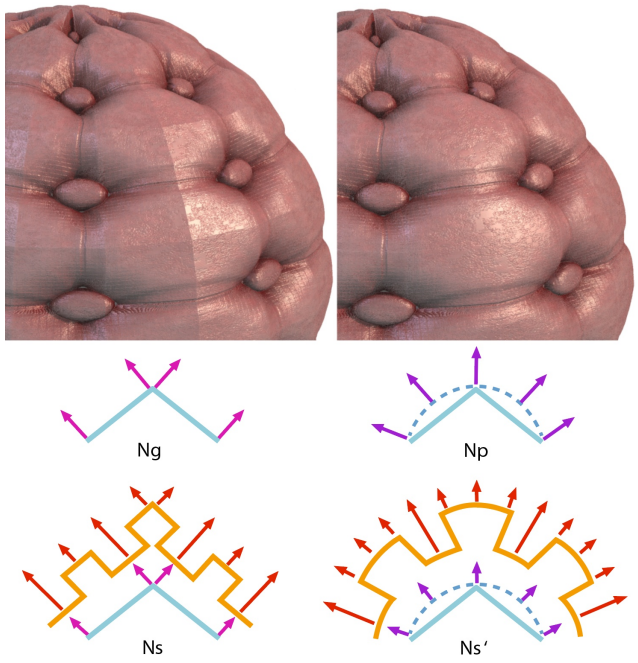

Figure 5: *Shading normal correction in low polygon meshes. When the angle between base triangles is high and no intermediate C^1 surface is used, the geometric normal N_g can be noticeably carried through to the displaced surface N_s. This can be corrected to give smoothed displaced normals N_s' by subtracting the geometric normal and adding in the interpolated normal N', as discussed in Section 5.1.*

ALGORITHM 1: Projective Displacement Mapping Program

Input: Prism: negative extents v_i, positive extents e_i, vertex normals n_i, Ray: r_0,r_d, and Displacement Map: f
Output: Hit t-value: t, Hit point: s, and Normal n

if *intersectPrism*(v_i, e_i, t_{min}, t_{max}) **then**
- $t = t_{min}$
- $s = r_0 + r_{dir}t$ — initial hit point
- $h_{ray} = pointToPlane(s, v_i)$ — initial sample height
- $c_i = v_i + (e_i - v_i)h_{ray}$ — initial scanning triangle

for $h_{ray} > h_{surf}$ *and* $t < t_{max}$ **do**
- $\delta h = n_s \cdot (c_0 - s)$ — point-to-plane distance
- $c_i = c_i - n_i \delta h$ — advance scanning triangle
- $b_i = triangleBarycentric(s, c_i)$ — barycentric coords
- $p = \vec{v}_0 b_0 + \vec{v}_1 b_1 + \vec{v}_2 b_2$ — projected point
- $h_{ray} = |s - p|$ — ray sample height
- $uv = \vec{uv}_0 b_0 + \vec{uv}_1 b_1 + \vec{uv}_2 b_2$ — uv coords
- $h_{surf} = textureSample(f, uv)$ — surface height
- $t = t + dt$
- $s = s + r_{dir}dt$

end

if $h_{ray} < h_{surf}$ **then**
- $t = interpolateCrossing(t, dt, h_{surf})$ — final hit
- $p_{hit} = r_0 + r_{dir}t$
- $b_i = triangleBarycentric(p_{hit}, c_i)$ — final barycentric coords
- $N' = \vec{n}_0 b_0 + \vec{n}_1 b_1 + \vec{n}_2 b_2$ — interpolated base normal
- $uv_a = \vec{uv}_0 b_0 + \vec{uv}_1 b_1 + \vec{uv}_2 b_2$ — finite difference uvs
- $uv_b = \vec{uv}_0 (b_0 + \delta b_x) + \vec{uv}_1 b_1 + \vec{uv}_2 (b_2 - \delta b_x)$
- $uv_c = \vec{uv}_0 b_0 + \vec{uv}_1 (b_1 + \delta b_y) + \vec{uv}_2 (b_2 - \delta b_y)$
- $p_a = \vec{v}_0 b_0 + \vec{v}_1 b_1 + \vec{n}_2 b_2$ — base points
- $p_b = \vec{v}_0 (b_0 + \delta b_x) + \vec{v}_1 b_1 + \vec{n}_2 (b_2 - \delta b_x)$
- $p_c = \vec{v}_0 b_0 + \vec{v}_1 (b_1 + \delta b_y) + \vec{n}_2 (b_2 - \delta b_y)$
- $s_a = p_a + N' textureSample(f, uv_a)$
- $s_b = p_b + N' textureSample(f, uv_b)$ — surface tangent
- $s_c = p_c + N' textureSample(f, uv_c)$ — and bi-tangent
- $N_s = (s_c - s_a) \times (s_b - s_a)$ — displaced surface normal
- $N'_s = N_s - N_g + N'$ — corrected normal (Sec 5.1)
- **return** t, p_{hit}, N'_s

end

end

4.2. Raytracing with Projective Displacement

The key idea of projective displacement mapping (PDM) is to sample the ray linearly in world space and then map this to non-linear barycentric coordinates while avoiding costly tangent-space matrix transforms. Each base mesh triangle is *parallel offset* to construct a prism defined by negative v_i and positive extents e_i. These prisms are inserted into a hardware BVH acceleration structure. A given ray $< r_0, r_d >$ then traverses the BVH to enter a specific prism at a bi-linear patch or at a top/bottom triangle face at entry point s_0 as in Figure 4a.

The scanning phase begins by constructing a parallel offset triangle, c_i, at the height of the sample point s_0, Figure 4a (red triangle). From this scanning triangle and the ray sample, known to be co-planar, we can compute barycentric coordinates b_i. These coordinates represent the projection of s_n along the interpolated normal to the base triangle point p, a proof of which is provided in Appendix A. The barycentric coordinates are used to derive UV coordinates to fetch a displacement texel d to compute the surface height $h_{surf} = |d - p|$, whose comparison to the ray sample height $h_{ray} = |s - p|$ determines if the surface has been hit.

If no surface is hit the scan proceeds by advancing the scanning triangle along the delta vectors δc_i according to the perpendicular height δr_h of the previous sample s_{n-1}. Note that we cannot simply update the ray height h_{ray} with δr_h because the sampling heights h_{ray} and h_{surf} are measured along the interpolated normal whereas δr_h is a change in height measured along the geometric normal n_g of the base triangle.

The main loop is optimized by avoiding boundary checks across each prism face. Instead, we leverage the earlier prism intersections which provide t_{min} and t_{max} as simplified loop bounds. Additionally, scanning is accelerated by updating the triangle incrementally by linear vectors δc_i that move the previous triangle to the next sample height.

Projective displacement is illustrated in Figure 4, with the algorithm provided in Listing 1. The function *triangleBarycentric* computes the barycentric coordinates of the sample point [Eri04]. The *textureSample* function evaluates the displacement map to retrieve the world space surface height. After an an initial hit is found, a more accurate hit is computed by interpolating across the surface boundary.

5. Smoothness, Quality and Details

5.1. Shading Normal Continuity

In several works on displacement mapping, the authors make use of intermediate surfaces such as Catmull-Clark or Bezier surfaces to achieve C^1 continuity across base triangle boundaries [SSS00] [LMH00] [JMW07] [NL13]. Discontinuities may be visible at base mesh boundaries if this is not done. We quantify and correct for these artifacts for the sake of efficiency without resorting to C^1 surfaces.

Shading discontinuities are observed when the angle between base triangles is high and the displacement map is relatively smooth. An example is provided in Figure 5. We prove in Appendix B that the displaced surface normal N_s, evaluated numerically at the hit point p with finite differences, contains the flat shading normal of a base triangle N_g, even though the surface $\mathbf{S}$ is computed from the interpolated normal $\mathbf{N}'$ in Eqn. 7. That is:

$$N_s = N_g + \frac{\partial D}{\partial v}\frac{\partial \mathbf{P}}{\partial v} + \frac{\partial D}{\partial u}\frac{\partial \mathbf{P}}{\partial u} \tag{9}$$

The term N_g introduces boundary artifacts across triangles in the base mesh, shown in Fig. 5 (left) and Fig. 8 (MicroMesh and RMIP). Similar to Phong shading [Pho73], we wish to replace the geometric normal N_g with the interpolated base normal N'. This can be done directly as:

$$N'_s = N_s - N_g + N' \tag{10}$$

where N_s is the displaced normal computed in Algorithm 1 and N'_s is the corrected normal. This retains the surface features of displacement while improving shading continuity. The resulting surface still has micro-bumps, displacement shadows and correct silhouettes.

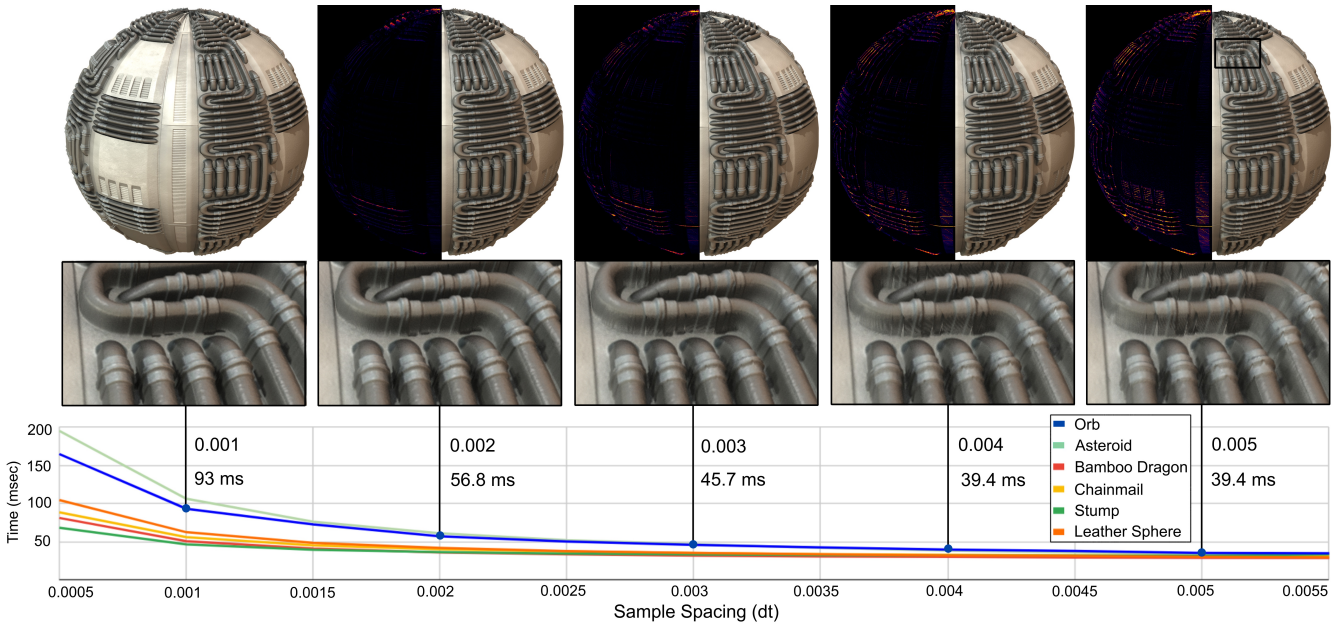

Figure 6: *Quality and performance versus sample rate. Visual and performance results are compare to sample spacing dt. Performance on the Orb model is <100 milliseconds when ray tracing high quality* 2048^2 *images with* $dt \leq 0.001$. *Errors are observed primarily at grazing angles when dt is large. We use* $dt = 0.002$ *for visual results in Fig. 8.*

5.2. Watertightness

The boundaries between prisms in our method are bilinear patches. Porumbescu et al. noted that piecewise approximations of this interface with tetrahedra result in "buckling" artifacts [PBFJ05]. Since we do not use tessellation, watertightness at prism boundaries is as good as the ray/bilinear patch algorithm "cool patches" we use from [Res19]. We can guarantee watertightness of samples across the base triangle so long as the minimum displacement height is adjusted to be larger than the sample spacing dt, that is $min(D(u,v)+\epsilon) > dt$.

5.3. Thin Features

Ray marching algorithms are known to miss features thinner than dt. We address this by stochastically shifting samples along t so that such features are integrated over multiple sample frames. Typically used to distribute rays across pixels, light sources, or surface BRDFs, the ray samples in a prism are shifted by $t_0 = t_{hit} + Rdt$ with a random number $R \sim \mathcal{U}(0,1)$. This ensures that a sample sequence covers thin features over many sample frames. The first ray sample s_0 must be jittered *after* prism intersection and entry. See Figure 12 for comparisons with and without thin feature sampling.

5.4. Offset Distance

Care must be taken to ensure that the maximum offset distance of prisms will bound the displaced surface. If two triangles meet at a sharp crease edge (Fig 11a), the denominator $N_i \cdot N_g$ in Equation 6 causes a singularity and the normal extension goes to infinity. A solution is to insert two new triangles at any such edges whose interior angle is less than some threshold (eg. $\theta < 5^\circ$). This issue occurred in fewer than 10 triangles in two of the models we used (Stanford Dragon and Tree Stump).

The offset distance, $w_{max}\frac{1}{N_i \cdot N_g}$, determines the maximum displacement from the base mesh. So long as this is everywhere greater than D(u,v) the displaced surface will be fully enclosed by the shell region. We can ensure tighter bounds and faster performance, by assigning w_{max} per prism at the cost of pre-computing $max(D(u,v))$ for each triangle over the depth texture. Although this can result in a misalignment in the top faces between two prisms, as in Figure 4b, the height of the displaced surface does not depend on the bounding prism extents and the surface is still found below both of these.

6. Implementation

Our algorithm is designed to integrate with standard hardware-based ray tracing engines that support bounding box, intersection and shading programs. We first developed a scanline CPU implementation for primary rays to verify algorithm design and resolve corner cases. For the GPU implementation we insert prisms and AABBs into a BVH acceleration structure using Nvidia OptiX for efficient hardware-based TRBVH trees [PBD*10] [KA13], then develop custom OptiX hit programs for intersections.

The sample spacing dt is a tradeoff between performance and quality. For our test models this was selected empirically and studied further in our results (Figure 6). Texture-based displacement methods require high quality 16-bit depth with most models using 4096^2 depth maps, while the Orb uses 8192^2. Final testing was

		Prism Intersect		Primary Rays (msec)			Beauty Image (msec)				Efficiency (Mrays/sec)		
Model	# Tri.	R_{isct}	R_{isct} %	RMIP 2023	Ours	Speed up	Micro Mesh	RMIP 2023	Ours	Speed up	RMIP 2023	Ours	Speed up
Asteroid	480	11.7	19	16.4	16.3	1.0	48.3	145.6	60.8	2.4	40.7	253.0	6.2
Chainmail	1423	11.8	57	29.4	12.6	2.3	28.9	289.0	20.7	13.9	26.3	388.3	14.8
Leather Sphere	3968	13.2	41	21.9	16.8	1.3	33.7	230.6	31.9	7.2	39.3	338.8	8.6
Orb	3968	13.2	23	32.5	18.2	1.8	68.2	329.1	56.8	5.8	28.2	188.2	6.7
Tree Stump	7710	10.4	57	18.9	13.3	1.4	19.5	166.2	18.1	9.2	34.7	369.5	10.6
Bamboo Dragon	22308	13.0	36	14.2	16.6	0.9	37.7	282.9	35.7	7.9	27.9	257.3	9.2

Table 1: *Performance. Comparisons of our method to RMIP [TGBB23] and Micro-Meshes [Nvi23] for models of varying complexity on a GeForce RTX 4090. Displacement maps are 16-bit* 4096^2 *pixels (Orb is* 8192^2*). Beauty images consist of 64 samples at 4 rays/pixel: primary, path trace, reflection and shadow. Measurements are for ray tracing* 2048^2 *output images at 1 sample/pixel and dt=0.002. All times are given in milliseconds.*

	Common				Algorithm Specific (MB)			Algorithm Overhead %		
Model	Displace Texture	Base Mesh	BVH	Total (MB)	MicroMesh Bary.Map	RMIP Dmap	PDM Prisms	MicroMesh	RMIP	PDM (ours)
Asteroid	33	0.008	0.08	33.1	6.9	27.19	0.049	21%	82%	0.1%
Chainmail	33	0.024	0.24	33.3	10.8	27.56	0.147	32%	83%	0.4%
Stump	33	0.132	1.32	34.7	13.6	30.05	0.794	39%	87%	2.3%
Bamboo Dragon	33	0.383	3.83	37.2	7.0	35.85	2.298	19%	96%	6.2%
Orb	134	0.068	0.68	134.7	23.2	28.57	0.409	17%	21%	0.3%

Table 2: *Memory usage. Comparison of our method to RMIP and Micro-Meshes on memory usage. Micro-Meshes are reported for 5 level micromesh maps (maximum allowed) with 5 level pre-tesselation. Projective Displacement achieves a low fixed overhead of 108 bytes per base triangle. The common costs are required by all algorithms. All memory units are in megabytes (MB).*

done on a GeForce RTX 4090 with 24 GB and on a RTX 3060 with an Intel Core i7-9700k, see 10b.

7. Results

Results are presented in several ways. Comparisons on the visual quality of projective displacement mapping to RMIP [TGBB23] and Micro-Meshes [Nvi23] are shown in Figures 8 and 10. Performance and memory usage are measured and reported in Tables 1 and 2 respectively. Final images were rendered at 4096^2 pixels with 64 samples consisting of 4 rays/pixel per sample: primary, path trace, reflection and shadow. For Micro-Meshes we modified the MicroMesh Toolkit source to output surface normal images, while Thonat [TGBB23] provided surface normal images for RMIP comparisons, Figure 8.

Thin features are well resolved by stochastic thin feature sampling. A fur torus was rendered with a high frequency 4096^2 displacement map in Figure 12 as discussed in Section 5.3. There is no overhead for this technique since the same number of rays and samples are used.

Although projective displacement is intended for authoring single objects, detailed scenes are possible with our method. Figure 9 contains nine displaced objects with different materials, rendered at 4096x1280 in 30 seconds with 462 ms/sample, 64 samples and 53.4 Mrays/sec while exhibiting four light sources, diffuse reflections, soft shadows and path tracing.

As a direct sampling technique, the sample spacing dt is an adjustable parameter that gives a tradeoff between quality and performance which are compared in Figure 6. Quality degrades primarily at grazing angles while we still achieve interactive rates under 100 ms on 2048^2 images with $dt = 0.002$ for all models.

7.1. Performance

Performance results are provided in Table 1. To quantify performance of algorithm stages we measured the average time for prism intersection R_{isct}, for just primary rays, and for ray tracing beauty images (4 rays/pix/sample). Our method ray traces on average between 20-60 milliseconds for 2048^2 pixel images.

Comparative performance to RMIP on our models was generously provided by Thonat [TGBB23]. Our method performed on average 40-60% faster for primary rays and 2x-13x faster for beauty images in comparison to RMIP using identical hardware. Although Micro-Meshes relies on tessellation, with goals and techniques different than ours, we still achieve faster ray tracing performance on most models tested. With the caveat that our method is not intended for massive scenes or high polygon meshes we achieve an overall performance between 200-400 Mrays/sec.

7.2. Memory Usage

Memory usage is provided in Table 2 in megabytes. We identify common costs for the displacement texture, base mesh triangles

and BVH. Since all algorithms require a BVH when ray tracing we report the maximum cost of 180 bytes/triangle from Domingues et al. as an upper bound since we were unable to measure the BVH directly [DP15]. Algorithm-specific costs were measured per method, such as Micro-Meshes barycentric map, RMIP D-map, and our PDM prisms. The algorithm overhead is given as the additional cost beyond the common memory used. Projective Displacement achieves high quality with a fixed memory overhead of 108 bytes per base triangle.

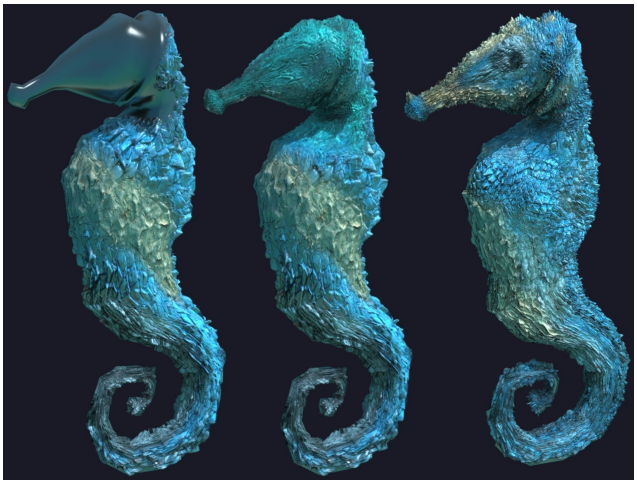

Figure 7: *Modeling example. Detailed seahorse modeled interactively with displacement brushes while ray tracing.*

Method	Tex [1]	Accel [2]	Raytrace [3]	Total (msec)	Total (FPS)
RMIP	3.2	3.4	329.1	335.7	2.9
PDM (full)	3.2	0	56.8	60.0	16.7
PDM (256^2)	3.2	0	11.0	14.2	70.4

Table 3: *Editing performance. Analysis of interative editing performance on RMIP and PDM (ours) for the Orb model. The table shows: [1] texture modifications to* 8192^2 *displacement and color maps, [2] acceleration rebuild for a* 1024^2 *RMIP in [TGBB23]; none for ours, and [3] re-rendering by ray tracing. PDM* 256^2 *only re-renders a screen sub-region around the cursor. All times in milliseconds.*

7.3. Interactive Editing

Artists desire interactive feedback when modeling displaced surfaces. We developed a brush-like editor to demonstrate surface sculpting with real-time effects such as global illumination, see Figures 1 and 7. A ray is traced at the mouse cursor to find the barycentric and u, v coordinates on a base triangle. Displacement brushes of 256^2 pixels are then blended into the target displacement map with color and depth. During sculpting a region is re-rendered around the mouse cursor at 14 milliseconds (70 fps). Figure 7 shows an example of authoring a 6400^2 displacement map (see Supplemental video). Upon mouse release or camera motion the full resolution image converges over multiple samples. Artists can interactively edit fine details with ray traced quality feedback.

Editing performance is analyzed in Table 2 for the 8192^2 displaced Orb model. Each frame requires texture modifications to the displacement and color maps over the 256^2 brush regions and ray tracing at one sample per pixel. PDM (full) is our technique with full screen resampling, whereas PDM (256^2) is ours while updating a region of pixels around the cursor.

8. Conclusions

We present projective displacement mapping as a novel direct sampling technique for interactive ray tracing and editing of displacement mapped surfaces. Our method integrates easily with hardware BVHs to accelerate per-ray sampling of displaced surfaces. We make available a reference CPU implementation, open source, to facilitate future research.

Ray tracing by our method is faster than the existing techniques for low poly meshes with detailed displacement maps. Visually we improve on the appearance of displaced surfaces by eliminating faceting with a new smoothed displaced normal, by eliminating buckling with ray/bilinear patch prism interfaces, by stochastically sampling thin features, and by improving watertightness with prism extent analysis. We demonstrate the real-time sculpting of detailed displaced surfaces as seen through other reflective and refractive objects.

Limitations of our technique are related to ray marching and sampling. This method is not intended for massive scenes, terrain, or compression of high geometry source meshes. We focus on the interactive sculpting of objects in look development work flows where memory resources may be limited. No acceleration rebuild is needed per frame if the displacement edits are less than a fixed maximum offset distance, whereas larger changes to the base mesh would require BVH reconstruction.

In the future we hope to examine the trade-offs between direct sampling and the rebuilding of BVH acceleration structures while attempting to maintaining interactivity. We would also like to explore C^1 intermediate surfaces to further study aspects of continuity. The geometric detail and natural authoring of displacement maps present compelling reasons for further exploration. We hope this work expands future opportunities for complex geometric modeling and interactive ray tracing.

9. Acknowledgements

Special thanks to Theo Thonat and Adobe Research for visual results and performance measurements on RMIP. Thanks to Pyralel Knowles and Nvidia for discussions on Micro-Meshes. Tree Stump model by Quixel Megascans. Displacement textures for orb, fish scales and creature skins by CGAxis.com. All other textures are public domain, CC0, or by the author.

References

[AFF12] AMRESH A., FEMIANI J. C., FÜNFZIG C.: Methods for approximating loop subdivision using tessellation enabled gpus. In *International Symposium on Visual Computing* (2012).

[Bli78] BLINN J. F.: Simulation of wrinkled surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1978), SIGGRAPH '78, Association for Computing Machinery, p. 286–292.

[BMBZ02] BIERMANN H., MARTIN I., BERNARDINI F., ZORIN D.: Cut-and-paste editing of multiresolution surfaces. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2002), SIGGRAPH '02, Association for Computing Machinery, p. 312–321.

[BP23] BENTHIN C., PETERS C.: Real-time ray tracing of micro-poly geometry with hierarchical level of detail. *Computer Graphics Forum 42*, 8 (2023), e14868.

[CABD11] CONCHEIRO R., AMOR M., BÓO M., DOGGETT M. C.: Dynamic and adaptive tessellation of bézier surfaces. In *International Conference on Computer Graphics Theory and Applications* (2011).

[CC08] CHEN Y.-C., CHANG C.-F.: A prism-free method for silhouette rendering in inverse displacement mapping. *Computer Graphics Forum 27*, 7 (2008), 1929–1936.

[Coo84] COOK R. L.: Shade trees. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1984), SIGGRAPH '84, Association for Computing Machinery, p. 223–231.

[DP15] DOMINGUES L. R., PEDRINI H.: Bounding volume hierarchy optimization through agglomerative treelet restructuring. In *Proceedings of the 7th Conference on High-Performance Graphics* (New York, NY, USA, 2015), HPG '15, Association for Computing Machinery, p. 13–20.

[DT07] DACHSBACHER C., TATARCHUK N.: Prism Parallax Occlusion Mapping with Accurate Silhouette Generation, 2007.

[Dum06] DUMMER J.: Cone step mapping: An iterative ray-heightfield intersection algorithm, 2006.

[DW05] DONNELLY W., WLOKA M. M.: *Per-Pixel Displacement Mapping with Distance Functions*. Addison-Wesley Professional, Glenview, IL, 2005, pp. 123–136.

[Eri04] ERICSON C.: *Real-time Collision Detection*. CRC Press, Boca Raton, FL, 2004.

[HEGD04] HIRCHE J., EHLERT A., GUTHE S., DOGGETT M.: Hardware accelerated per-pixel displacement mapping. In *Proceedings of Graphics Interface 2004* (Waterloo, CAN, 2004), GI '04, Canadian Human-Computer Communications Society, p. 153–158.

[HYS23] HAYDEL J., YUKSEL C., SEILER L.: Locally-adaptive level-of-detail for hardware-accelerated ray tracing. *ACM Trans. Graph. 42*, 6 (Dec. 2023).

[JB86] JOY K. I., BHETANABHOTLA M. N.: Ray tracing parametric surface patches utilizing numerical techniques and ray coherence. *SIGGRAPH Comput. Graph. 20*, 4 (aug 1986), 279–285.

[JH12] JANG H., HAN J.: Feature-preserving displacement mapping with graphics processing unit (gpu) tessellation. *Computer Graphics Forum 31*, 6 (2012), 1880–1894.

[JH13] JANG H., HAN J.: Gpu-optimized indirect scalar displacement mapping. *CAD Computer Aided Design 45*, 2 (Feb. 2013), 517–522. Funding Information: This research was supported by a National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2009-0086684).

[JMW07] JESCHKE S., MANTLER S., WIMMER M.: Interactive Smooth and Curved Shell Mapping. In *Rendering Techniques* (Prague, 2007), Kautz J., Pattanaik S., (Eds.), The Eurographics Association, pp. 351–360.

[KA13] KARRAS T., AILA T.: Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference* (New York, NY, USA, 2013), HPG '13, Association for Computing Machinery, p. 89–99.

[Kaj83] KAJIYA J. T.: New techniques for ray tracing procedurally defined objects. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1983), SIGGRAPH '83, Association for Computing Machinery, p. 91–102.

[KHK*09] KNOLL A., HIJAZI Y., KENSLER A., SCHOTT M., HANSEN C., HAGEN H.: Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. *Computer Graphics Forum 28*, 1 (2009), 26–40.

[KMX*21] KUZNETSOV A., MULLIA K., XU Z., HAŠAN M., RAMAMOORTHI R.: Neumip: multi-resolution neural materials. *ACM Trans. Graph. 40*, 4 (July 2021).

[KSG21] KARIS B., STUBBE R., G. W.: A deep dive into nanite virtualized geometry. ACM SIGGRAPH, Course Slides.

[KTI*01] KANEKO T., TAKAHEI T., INAMI M., KAWAKAMI N., YANAGIDA Y., MAEDA T., TACHI S.: Detailed shape representation with parallax mapping. *In Proceedings of the ICAT 2001* (01 2001).

[LEF91] LAMOTTE W., ELENS K., FLERACKERS E.: Surface tree caching for rendering patches in a parallel ray tracing system. In *Scientific Visualization of Physical Phenomena* (Tokyo, 1991), Patrikalakis N. M., (Ed.), Springer Japan, pp. 189–207.

[LG90] LISCHINSKI D., GONCZAROWSKI J.: Improved techniques for ray tracing parametric surfaces. *The Visual Computer 6*, 3 (1990), 134–152.

[LMH00] LEE A., MORETON H., HOPPE H.: Displaced subdivision surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., p. 85–94.

[LSNCn09] LOOP C., SCHAEFER S., NI T., CASTAÑO I.: Approximating subdivision surfaces with gregory patches for hardware tessellation. *ACM Trans. Graph. 28*, 5 (Dec. 2009), 1–9.

[LTT09] LEE L., TSENG S.-W., TAI W.: Improved relief texture mapping using minmax texture. Fifth International Conference on Image and Graphics, p. 547–552.

[MHTAM10] MUNKBERG J., HASSELGREN J., TOTH R., AKENINE-MÖLLER T.: Efficient bounding of displaced bézier patches. In *Proceedings of the Conference on High Performance Graphics* (Goslar, DEU, 2010), HPG '10, Eurographics Association, p. 153–162.

[Mic09] MICROSOFT: DirectX 11 api, 2009. Accessed: 2025-01-10.

[NKF*16] NIESSNER M., KEINERT B., FISHER M., STAMMINGER M., LOOP C., SCHÄFER H.: Real-time rendering techniques with hardware tessellation. *Computer Graphics Forum 35*, 1 (2016), 113–137.

[NL13] NIESSNER M., LOOP C.: Analytic displacement mapping using hardware tessellation. *ACM Trans. Graph. 32*, 3 (jul 2013).

[NMMC14] NYKL S., MOURNING C., M. CHELBERG D.: Interactive mesostructures with volumetric collisions. *IEEE Transactions on Visualization and Computer Graphics 20*, 7 (2014), 970–982.

[Nvi23] NVIDIA: Micro-mesh api, 2023. Accessed: 2025-01-10.

[OBM00] OLIVEIRA M. M., BISHOP G., MCALLISTER D.: Relief texture mapping. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 2000), SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., p. 359–368.

[Oga23] OGAKI S.: Nonlinear ray tracing for displacement and shell mapping. In *SIGGRAPH Asia 2023 Conference Papers* (New York, NY, USA, 2023), SA '23, Association for Computing Machinery.

[OP05] OLIVEIRA M., POLICARPO F.: An efficient representation for surface details. *UFRGS Technical Report RP-351* (01 2005).

[PBD*10] PARKER S. G., BIGLER J., DIETRICH A., FRIEDRICH H.,

HOBEROCK J., LUEBKE D., MCALLISTER D., MCGUIRE M., MORLEY K., ROBISON A., STICH M.: Optix: A general purpose ray tracing engine. *ACM Trans. Graph. 29*, 4 (jul 2010).

[PBFJ05] PORUMBESCU S. D., BUDGE B., FENG L., JOY K. I.: Shell maps. *ACM Trans. Graph. 24*, 3 (jul 2005), 626–633.

[PH96] PHARR M., HANRAHAN P.: Geometry caching for ray-tracing displacement maps. In *Rendering Techniques '96* (Vienna, 1996), Pueyo X., Schröder P., (Eds.), Springer Vienna, pp. 31–40.

[Pho73] PHONG B.: *lllumination for ComputerGenerated Images*. Ph.d. dissertation, University of Utah, Salt Lake City, Department of Computer Science, July 1973.

[PKZ04] PENG J., KRISTJANSSON D., ZORIN D.: Interactive modeling of topologically complex geometric detail. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 635–643.

[PO07] POLICAROP F., OLIVEIRA M.: *Relaxed Cone Stepping for Relief Mapping*. Addison-Wesley Professional, 2007.

[Res19] RESHETOV A.: *Cool Patches: A Geometric Approach to Ray/Bilinear Patch Intersections*. Apress, Berkeley, CA, 2019, pp. 95–109.

[Rit06] RITSCHE N.: Real-time shell space rendering of volumetric geometry. In *Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia* (New York, NY, USA, 2006), GRAPHITE '06, Association for Computing Machinery, p. 265–274.

[RMD11] REINER T., MÜCKL G., DACHSBACHER C.: Interactive modeling of implicit surfaces using a direct visualization approach with signed distance functions. *Computers and Graphics 35*, 3 (2011), 596–603. Shape Modeling International (SMI) Conference 2011.

[RPH04] RAMSEY S. D., POTTER K. C., HANSEN C. D.: Ray bilinear patch intersections. *Journal of Graphics Tools 9* (2004), 41 – 47.

[SB87] SNYDER J. M., BARR A. H.: Ray tracing complex models containing surface tessellations. *SIGGRAPH Comput. Graph. 21*, 4 (aug 1987), 119–128.

[SL03] STAM J., LOOP C.: Quad/triangle subdivision. *Computer Graphics Forum 22*, 1 (2003), 79–85.

[SSS00] SMITS B., SHIRLEY P., STARK M.: Direct ray tracing of displacement mapped triangles. In *Rendering Techniques, Proc. 11th Eurographics Workshop on Rendering* (Brno, Czech Republic, 07 2000), Springer, pp. 307–318.

[Tat05] TATARCHUK N.: Practical dynamic parallax occlusion mapping. In *ACM SIGGRAPH 2005 Sketches* (New York, NY, USA, 2005), SIGGRAPH '05, Association for Computing Machinery, p. 106–es.

[TBS*21] THONAT T., BEAUNE F., SUN X., CARR N., BOUBEKEUR T.: Tessellation-free displacement mapping for ray tracing. *ACM Transactions on Graphics 40*, 6 (dec 2021).

[TFM15] TEJIMA T., FUJITA M., MATSUOKA T.: Direct ray tracing of full-featured subdivision surfaces with bezier clipping. *Journal of Computer Graphics Techniques (JCGT) 4*, 1 (March 2015), 69–83.

[TGBB23] THONAT T., GEORGIEV I., BEAUNE F., BOUBEKEUR T.: Rmip: Displacement ray tracing via inversion and oblong bounding. In *SIGGRAPH Asia 2023 Conference Papers* (New York, NY, USA, 2023), SA '23, Association for Computing Machinery.

[Tot85] TOTH D. L.: On ray tracing parametric surfaces. *SIGGRAPH Comput. Graph. 19*, 3 (jul 1985), 171–179.

[Wil90] WILLIAMS L.: 3d paint. *SIGGRAPH Comput. Graph. 24*, 2 (Feb. 1990), 225–233.

[WSC02] WANG S.-W., SHIH Z.-C., CHANG R.-C.: An efficient and stable ray tracing algorithm for parametric surfaces. *J. Inf. Sci. Eng. 18* (07 2002), 541–561.

[WWT*03] WANG L., WANG X., TONG X., LIN S., HU S., GUO B., SHUM H.-Y.: View-dependent displacement mapping. *ACM Trans. Graph. 22*, 3 (July 2003), 334–339.

[Yus12] YUSOV E.: *Real-Time Deformable Terrain Rendering with DirectX 11*. A.K. Peters / CRC Press, 2012, p. 28.

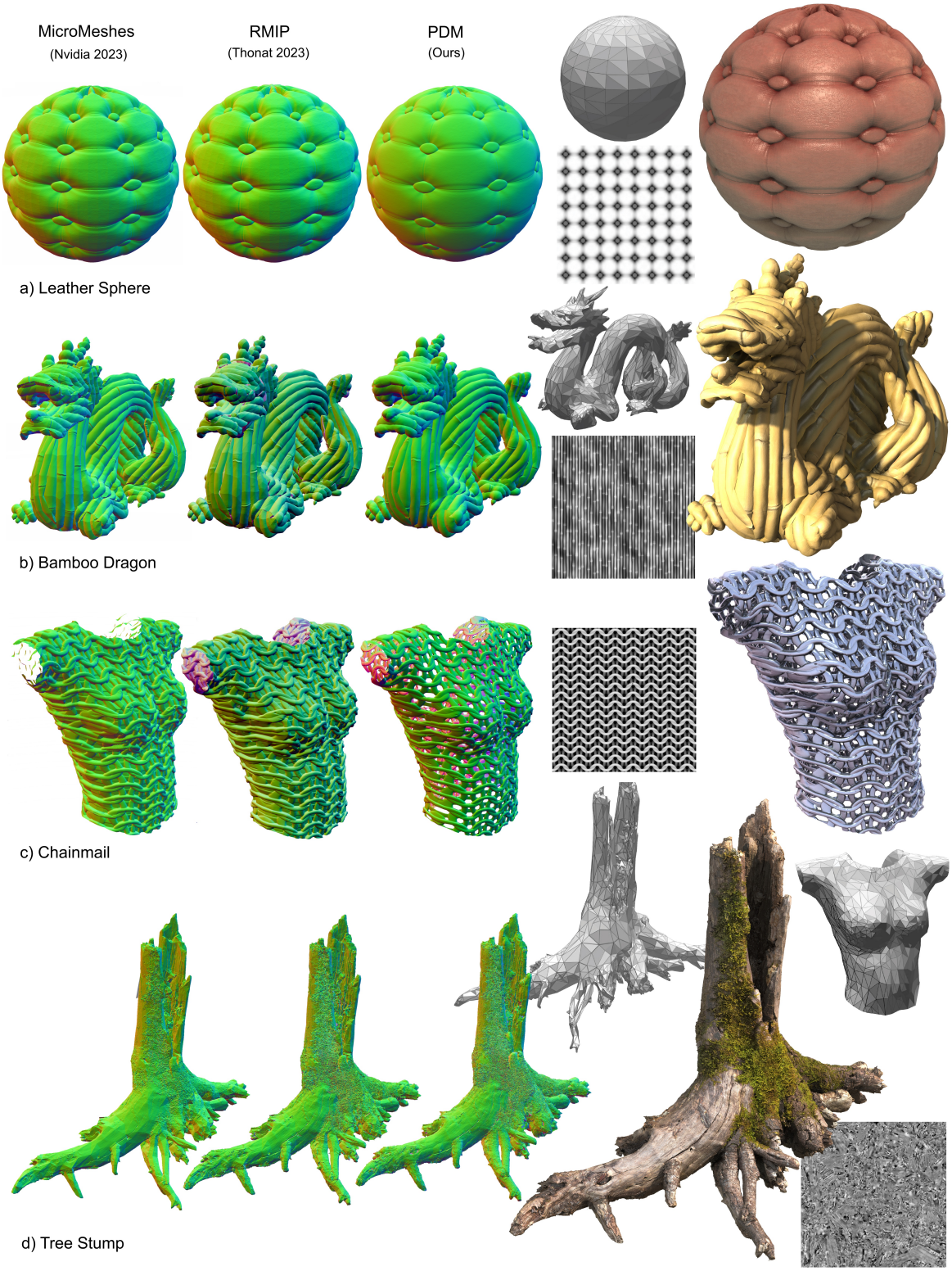

Figure 8: *Ray tracing results. Displaced surface normals are compared between Micro-Meshes [Nvi23], RMIP [TGBB23] and our Projective Displacement method with* 2048^2 *pixel at 64 spp (left). The base mesh, displacement map, and beauty images are shown for each model (right). The smooth surfaces in our leather sphere, dragon and tree stump are due to surface normal correction, see Section 5.1.*

Figure 9: *Detailed scene. A detailed scene ray traced with only 15k triangles and 1.8 MB GPU memory in 30 seconds at 4096x1280. All objects use our technique except for the ground surface.*

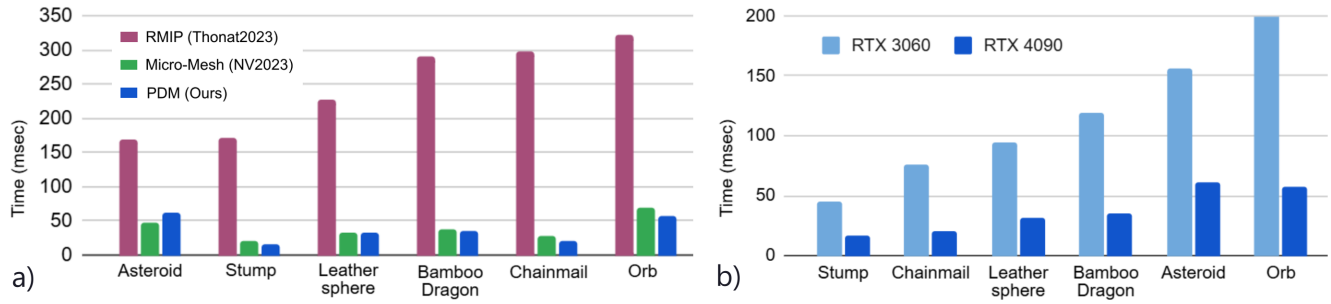

Figure 10: *a) Algorithm comparisons for projective displacement mapping (ours), Micro-Meshes [Nvi23] and RMIP [TGBB23] for different source models, and b) GPU Hardware comparsions for RTX 3060 and RTX 4090 versus for different source models.*

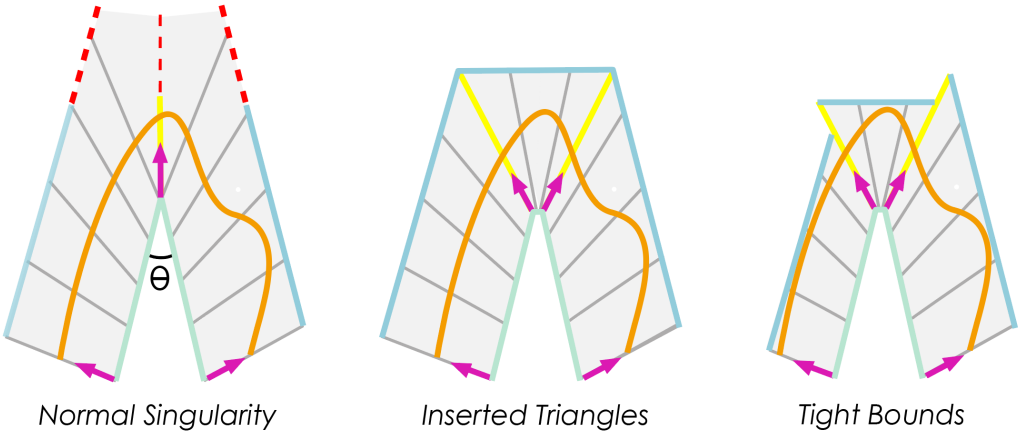

Figure 11: *Offset distance. Sharp crease edges can cause singularities (left), which are corrected by inserting base triangles at these edges. Tighter bounds can be achieved by setting maximum offset extents per prism (right).*

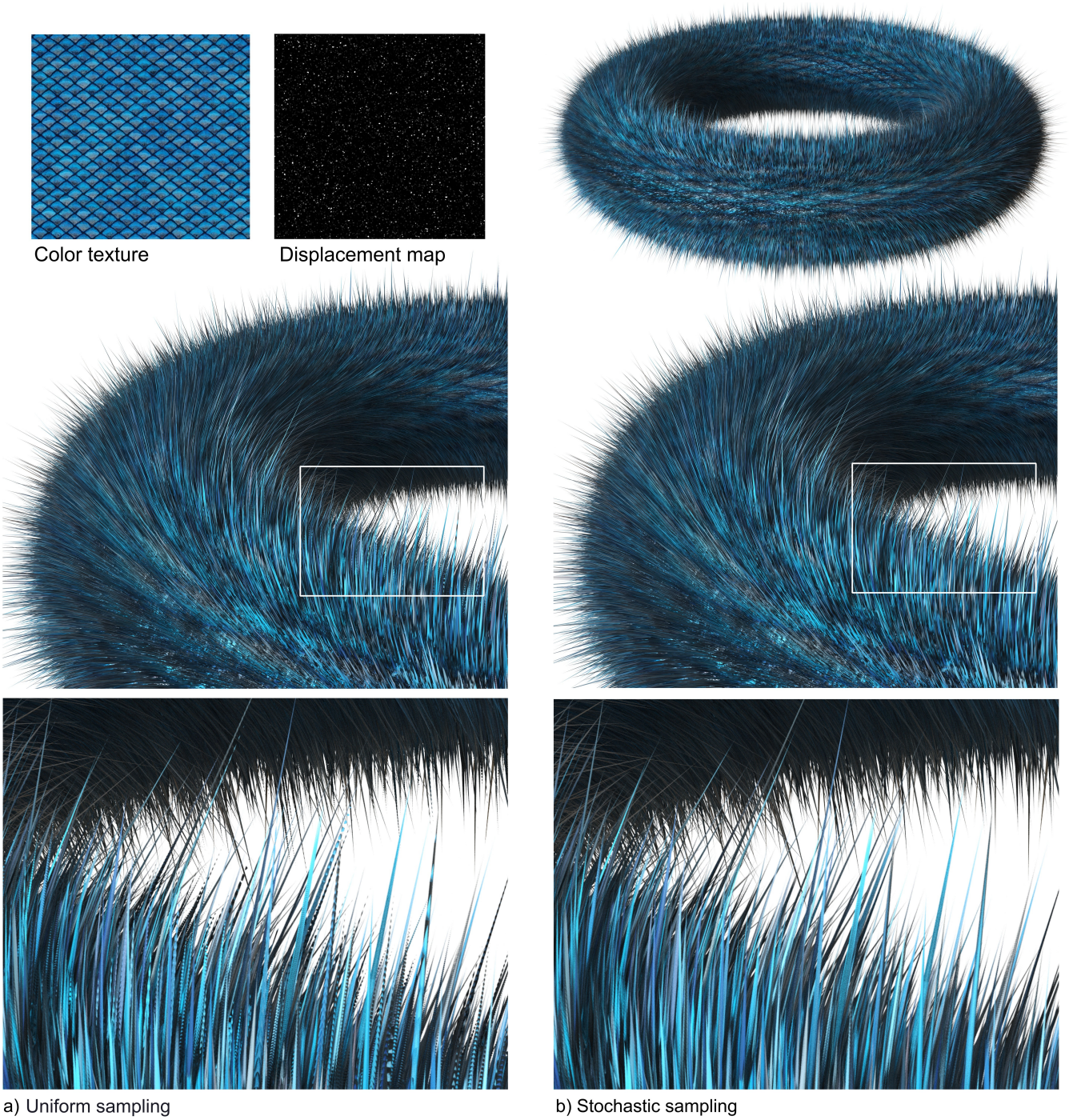

Figure 12: *Thin feature sampling. Thin features are ray traced with stochastic sampling, see Section 5.3. The fur torus is rendered at* 4096^2*, 64 spp, and* $dt = 0.001$*. a) Uniform sampling causes thin features to be periodically missed, whereas b) Stochastic sampling uses distributed first samples along t after prism intersection to integrate over these features. This method incurs no overhead since the same number of rays and samples are used.*