RMIP: Displacement ray-tracing via inversion and oblong bounding Supplemental document

THÉO THONAT, Adobe, France
ILIYAN GEORGIEV, Adobe, UK
FRANÇOIS BEAUNE, Adobe, France
TAMY BOUBEKEUR, Adobe, France

In this supplemental document we provide additional details on several components of our method, namely about the RMIP data structure and the per-triangle bounding prisms.

1 RMIP

In the main document, we introduce the RMIP data structure as an array of 2D grids that answers minmax queries over rectangle ranges of an input discrete grid. We detail here how to extend it so it provides bounds for an input texture equipped with interpolation, tiling, and level of details. We also detail how to reduce the RMIP memory footprint so it is similar to a traditional mipmap, and show how to precompute the structure.

1.1 Extension to textures

The RMIP structure described so far only deals with discrete grids of values. We present here how to extend it to compute bounds in a displacement mapping context, with continuous interpolation, tiling, and multi-scale filtering.

Displacement interpolation. Going from a continuous displacement signal to a discrete 2D grid in a conservative fashion is straightforward by computing a conservative minmax over each pixel region. For example with bilinear interpolation, we find the four bilinear patches that overlap one pixel of the output minmax, and aggregate bounds from each patch.

Tiling. We address queries that wrap around the input texture by pre-computing additional queries. Assuming an $N\times N$ resolution for the RMIP, the $H^{p,q}_{x,y}$ values with $N-2^p < x < N$ or $N-2^q < y < N$, which were unused so far as the corresponding query rectangle was crossing the grid boundaries, can now be computed assuming a repeating texture.

Level of detail. As noted by Thonat et al. [2021], minmax mipmaps do not directly provide bounds for pre-filtered versions of its input. They addressed this issue by making each texel of the minmax mipmap storing bounds over multiple input mip levels, thus degrading a bit the bounds tightness. For our RMIP, we take a different approach by computing a RMIP independently for each of the input mipmap levels. However, since the number of array layers is $(1+\log_2 N)^2$, each mip level has a different layer count, which is not suitable for sampling fractional LOD using the hardware. Therefore, we fill the whole mip hierarchy by carefully copying layers within the same mip level, in a way that the minmax region associated to an uv at a LoD k is always a subset of the minmax region for the same uv at the LoD $k+1$ (see the main document for the resulting layout).

Algorithm 1. RMIP precomputation for one mip level of an input displacement texture h. Each output texture array layer $H^{p,q}$ has an $N\times N$ resolution. The RMQ function is the one defined in Equation 5. in the main document.

function ComputeRMIP(h)
  $H^{0,0}$ = blockwise minmax(h)
  for $0 \le q \le \log_2 N$ **do**
    for $0 \le p < \log_2 N, 0 \le y \le N-2^q, 0 \le x \le N-2^{p+1}$ **do**
      $H^{p+1,q}_{x,y} = \text{minmax}\left(H^{p,q}_{x,y}, H^{p,q}_{x+2^p,y}\right)$ ← Merge bounds along x
    if $q < \log_2 N$ **then**
      for $0 \le y \le N-2^{q+1}, 0 \le x \le N-1$ **do**
        $H^{0,q+1}_{x,y} = \text{minmax}\left(H^{0,q}_{x,y}, H^{0,q}_{x,y+2^q}\right)$ ← Merge bounds along y
  for $0 \le q \le \log_2 N, 0 \le p \le \log_2 N$ **do**
    for $0 \le y_0 < N, 0 \le x_0 < N$ **do**
      if $x_0 \le N-2^p$ **and** $y_0 \le N-2^q$ **then**
        continue ← Those dont wrap around the texture
      if $x_0+2^p > N$ **then** ← Split the x range into two ranges
        $w_0 = N-x_0,\ x_1 = 0,\ w_1 = x_0+2^p-N$
      else ← Otherwise just duplicate the x range
        $x_1 = x_0,\ w_1 = w_0 = 2^p$
      if $y_0+2^q > N$ **then** ← Split the y range into two ranges
        $h_0 = N-y_0,\ y_1 = 0,\ h_1 = y_0+2^q-N$
      else ← Otherwise just duplicate the y range
        $y_1 = y_0,\ h_1 = h_0 = 2^q$
      $b_1 = \text{RMQ}(x_0, y_0, w_0, h_0)$
      $b_2 = \text{RMQ}(x_1, y_0, w_1, h_0)$
      $b_3 = \text{RMQ}(x_0, y_1, w_0, h_1)$
      $b_4 = \text{RMQ}(x_1, y_1, w_1, h_1)$
      $H^{p,q}_{x,y} = \text{minmax}(b_1, b_2, b_3, b_4)$
  return H

1.2 Linear memory footprint

While the RMIP described so far gives tight and efficient bounds for displacement RMQs, its $N^2(1+\log_2^2 N)$ memory footprint is too prohibitive to be practical for high resolution displacement maps. However, since we only need conservative displacement bounds, we can sacrifice some tightness to reduce the RMIP resolution so that its

SA Conference Papers '23, December 12–15, 2023, Sydney, NSW, Australia

The definitive Version of Record was published in *SIGGRAPH Asia 2023 Conference Papers (SA Conference Papers '23), December 12–15, 2023, Sydney, NSW, Australia*, https://doi.org/10.1145/3610548.3618182.

memory footprint becomes similar to minmax mipmaps. As we cannot easily compress the RMIP, the only parameter we can act upon is the resolution of the input displacement map. More precisely, we look for a down scale resolution R such that its associated RMIP contains only N^2 values. This leads to the equation $R \cdot (1 + \log_2 R) = N$, whose solution is given by $R = \frac{1}{2}\exp(W_0(N \log 4)) \sim N / \log_2 N$, where W_0 is Lambert's W function's principal branch. The downscale of the input displacement is done conservatively, computing minmax over pixel blocks of size $B = N / R \sim \log_2 N$.

1.3 Precomputation

We present in this section the RMIP data structure precomputation, with pseudo code shown in Alg. 1. The computation is done in three passes. First the input texture is conservatively scaled down to the RMIP resolution by computing minmax over pixel blocks. The non-wrapping queries are then computed iteratively by combining bounds along one dimension at a time. This relies on the fact that a rectangle with a power-of-two size can be decomposed into two smaller non-overlapping sub rectangles with also a power-of-two size. Finally, the wrapping queries are computed by decomposing them into up to four non-wrapping sub queries, splitting the query at the texture boundary. As these sub queries do not have a power-of-two size, they need to be computed on the fly using already precomputed RMIP entries from the second pass.

For hardware LoD support, the above pre-computation can be independently applied to each mip level of the input texture to fill each mip level of the RMIP structure. Then, since the number of texture array layers varies for each mip level, several layers have to be copied to fill the remaining empty array layers (see Figure 6 in the main document). More specifically, for any mip level i with $1 \le i \le \log_2 N$, every array layer $H^{p,q}$ with $p > \log_2 N - i$ or $q > \log_2 N - i$ gets filled using $H^{\min(p, \log_2 N - i), \min(q, \log_2 N - i)}$ from the same mip level.

2 BOUNDING PRISM

Our bounding prism is made of two triangles and three bilinear patches. The triangles are computed by offsetting the base triangle along the vertex normals as we detail below. To intersect the prism we use the routines of Möller and Trumbore [2005] and Reshetov [2019], respectively. Intersections, including the ray starting point if inside the prism, are first sorted front to back, then grouped by pairs to form non overlapping intervals in 3D, that are finally projected into texture space to form the initial 2D bounds stack.

Following the displacement mapping equation, bounding prisms can be constructed considering the set

$$\{P(u,v) + sN(u,v), (u,v) \in \mathcal{T}, s \in [s_{\min}, s_{\max}]\}$$

which fully contains the displace surface when

$$s_{\min} = \min_{\mathcal{T}} \frac{h(u,v)}{|N(u,v)|} \ge \min\left(\frac{\min_{\mathcal{T}} h(u,v)}{\min_{\mathcal{T}} |N(u,v)|}, \frac{\min_{\mathcal{T}} h(u,v)}{\max_{\mathcal{T}} |N(u,v)|} \right)$$

and

$$s_{\max} = \max_{\mathcal{T}} \frac{h(u,v)}{|N(u,v)|} \le \max\left(\frac{\max_{\mathcal{T}} h(u,v)}{\min_{\mathcal{T}} |N(u,v)|}, \frac{\max_{\mathcal{T}} h(u,v)}{\max_{\mathcal{T}} |N(u,v)|} \right)$$

A tight bounding prism can therefore be obtained by computing displacement bounds and interpolated normal norm bounds over the base triangle. Conservative displacement bounds can be obtained for example using the RMIP data structure with a 2D bounding box of the base triangle in texture space, or using a minmax mipmap using the routine described by Thonat et al. [2021]. For the interpolated normal, assuming the three vertex normals $\mathbf{n}_i$ are normalized, the optimal upper bound is $\max_{\mathcal{T}} |N(u,v)| = 1$. For the lower bound, we need to minimize the following function:

$$\min_{u,v \in \mathcal{T}} |N(u,v)|^2 = \min_{u,v \in \mathcal{T}} |u\mathbf{n}_1 + v\mathbf{n}_2 + (1-u-v)\mathbf{n}_3|^2 \tag{1}$$

Since $\mathcal{T}$ is compact and $|N|^2$ is C^1, $|N|^2$ has a minimum on $\mathcal{T}$ that is either reached on the interior of $\mathcal{T}$ where $J_{|N|^2} = 0$, or on the boundary of $\mathcal{T}$. Computing the Jacobian gives:

$$J_{|N|^2}(u_0, v_0) = 0 \Leftrightarrow$$
$$\text{Gramian}(\mathbf{n}_1 - \mathbf{n}_3, \mathbf{n}_2 - \mathbf{n}_3) \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = - \begin{bmatrix} \mathbf{n}_3 \cdot (\mathbf{n}_1 - \mathbf{n}_3) \\ \mathbf{n}_3 \cdot (\mathbf{n}_2 - \mathbf{n}_3) \end{bmatrix} \tag{2}$$

If $|(\mathbf{n}_1 - \mathbf{n}_3) \times (\mathbf{n}_2 - \mathbf{n}_3)| \ne 0$, Gramian$(\mathbf{n}_1 - \mathbf{n}_3, \mathbf{n}_2 - \mathbf{n}_3)$ is positive-definite. Eq. (2) has a solution (u_0, v_0), and if it is inside $\mathcal{T}$, our lower bound is $\min_{\mathcal{T}} |N(u,v)| = |N(u_0, v_0)|$. Otherwise, the minimum is on the triangle boundary and we have:

$$\begin{aligned} \min_{u,v \in \mathcal{T}} |N(u,v)|^2 &= \min_{i \ne j} \min_{0 \le \lambda \le 1} |(1-\lambda)\mathbf{n_i} + \lambda \mathbf{n_j}|^2 \\ &= \min_{i \ne j} \left| \frac{1}{2} \left(\mathbf{n_i} + \mathbf{n_j} \right) \right|^2 \\ &= \frac{1}{2} \left(1 + \min\left(\mathbf{n}_1 \cdot \mathbf{n}_2, \mathbf{n}_1 \cdot \mathbf{n}_3, \mathbf{n}_2 \cdot \mathbf{n}_3 \right) \right) \end{aligned}$$

3 IMPLEMENTATION DETAILS

Inverse displacement. Inverting displacement for a 3D point $\mathbf{X}$ mean finding texture parameters u, v such that:

$$\mathbf{f}(u,v) = (\mathbf{P}(u,v) - \mathbf{X}) \times \mathbf{N}(u,v) = \mathbf{0}, \tag{3}$$

where $\mathbf{P}$ and $\mathbf{N}$ are respectively the linearly interpolated base position and normal. We solve the above equation using Newton's method, as the Jacobian of $\mathbf{f}$ has a simple expression:

$$J_{\mathbf{f}}(u,v) = \left[\frac{\partial \mathbf{f}}{\partial u}, \frac{\partial \mathbf{f}}{\partial v} \right] = \begin{bmatrix} \frac{\partial \mathbf{P}}{\partial u} \times \mathbf{N} + (\mathbf{P} - \mathbf{X}) \times \frac{\partial \mathbf{N}}{\partial u} \\ \frac{\partial \mathbf{P}}{\partial v} \times \mathbf{N} + (\mathbf{P} - \mathbf{X}) \times \frac{\partial \mathbf{N}}{\partial v} \end{bmatrix}^T, \tag{4}$$

where partial derivatives for base position and normal are constant per base triangle. We stop the iterative process when the update value has a norm less than 10^{-5}, with a maximum of 10 iterations. Regarding the initialization, there are two scenarios for choosing the starting point. For points on the bounding prism, we use the base triangle center in texture space. For points on the bounding boxes that are computed during traversal, we use the center of the current 2D bound. Since 2D bounds get tighter during the traversal, the starting point gets closer to the solution, making the inversion computational cost diminish with traversal depth.

Traversal stack. Our traversal relies on a stack of 2D bounds of the ray in texture space, as shown in Algorithm 1 in the main document. We use a maximum stack size of 17. As at most two bounds are pushed per loop iteration, with the last bound inserted being always popped at the next iteration, this stack size corresponds to a maximum traversal depth of 16. Our splitting halves the 2D bounds in their largest dimension at each split, so this stack size allows to cover a region of at least $2^{16} \cdot m^2$ pixels, where m is the marching scale. Note that this is a very conservative estimate, because most bounds get discarded right after the 3D bounds intersection test. In practice, we never reach the stack limit even with a marching scale of 1 and $4k$ maps tiled multiple times over the same base triangle.

REFERENCES

Tomas Möller and Ben Trumbore. 2005. Fast, minimum storage ray/triangle intersection. In *ACM SIGGRAPH 2005 Courses*. 7–es.

Alexander Reshetov. 2019. Cool patches: A geometric approach to ray/bilinear patch intersections. *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs* (2019), 95–109.

Theo Thonat, Francois Beaune, Xin Sun, Nathan Carr, and Tamy Boubekeur. 2021. Tessellation-Free Displacement Mapping for Ray Tracing. 40, 6, Article 282 (dec 2021), 16 pages. https://doi.org/10.1145/3478513.3480535

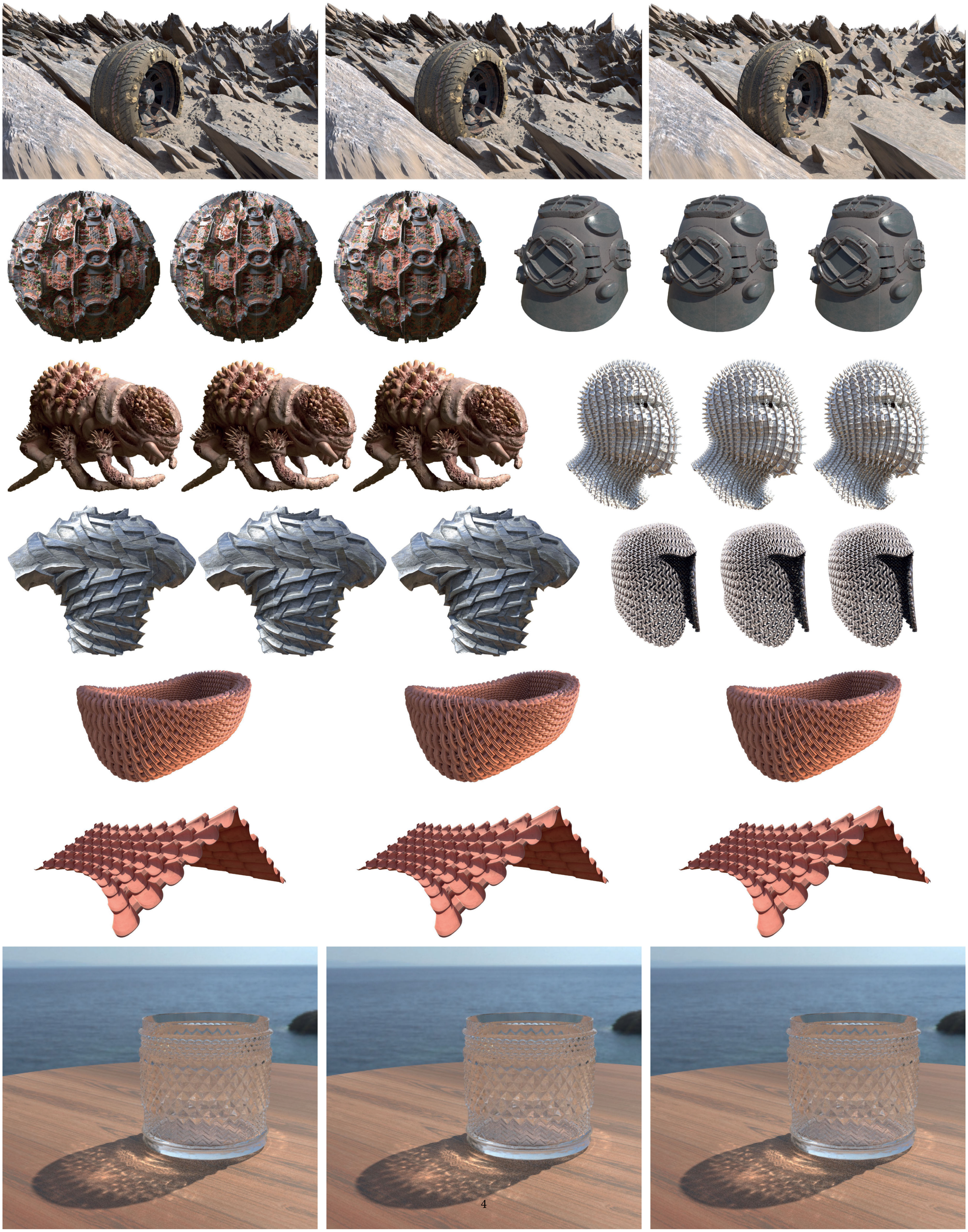

Fig. 1. Visual comparison between our method (left images), TFDM [Thonat et al. 2021] (middle images), and uniform pre-tessellation (right images).

Table 1. Performance and memory-footprint comparison between TFDM [Thonat et al. 2021] (serving as a 1× baseline), uniform tessellation, and our method with three different resolutions for our RMIP structure. For each configurations, the right column compares the data structure memory consumption (lower is better), and the right column reports render-time speed-ups on CPU (higher is better).

Scene	**#Tri**	**Disp.**	**Tiling**	**TFDM** (1×) [Thonat et al. 2021]		**Pre-tessellation** Uniform		**Our traversal** Low-res. RMIP		RMIP		High-res. RMIP	
				Mem.	CPU	Mem.↓	CPU↑	Mem.↓	CPU↑	Mem.↓	CPU↑	Mem.↓	CPU↑
Alien Sphere	0.9k	2k	6×8	21Mb	123ms	×124	×16	×0.048	×1.5	×1.3	×2.0	×30	×1.6
Basket	4.8k	2k	5×5	21Mb	155ms	×157	×18	×0.048	×1.9	×1.3	×2.5	×30	×2.0
Creature	53k	4k	1×1	85Mb	131ms	×27	×21	×0.012	×1.9	×1.6	×3.5	×7.6	×3.8
Diving Helmet	2.5k	4k	1×1	85Mb	43ms	×21	×11	×0.012	×1.3	×1.6	×1.6	×7.6	×1.7
Medieval Helmet	2.3k	2k	5×5	21Mb	45ms	×74	×11	×0.048	×1.2	×1.3	×1.6	×30	×1.3
Elven Armor	0.8k	4k	2×2	85Mb	79ms	×25	×15	×0.012	×1.1	×1.6	×1.4	×7.6	×1.5
Ninja Head	8.7k	2k	5×5	21Mb	226ms	×70	×39	×0.048	×2.2	×1.3	×3.6	×30	×2.7
Terracotta Roof	128	2k	2×2	21Mb	46ms	×66	×10	×0.048	×1.1	×1.3	×1.6	×30	×1.2
Desert Tire	9k	4k × 3	25^2, 6×1	256Mb	2948ms	×6.5	×284	×0.012	×5.4	×1.6	×8.4	×7.6	×8.8