

An Adaptive Acceleration Structure for Screen-space Ray Tracing

Supplemental Material

S. Widmer^{1,2} D. Pająk^{1,3} A. Schulz⁴ K. Pulli^{1,3} J. Kautz¹ M. Goesele² D. Luebke¹
¹NVIDIA ²Graduate School of Computational Engineering, TU Darmstadt ³Light ⁴TU Darmstadt

1 Pseudo code

Algorithm 1: GLSL pseudo-code for generating the bottom level of our traversal acceleration structure. λ_h and λ_d are set to 10^{-3} in our implementation.

```

input : depth ; /* depth buffer data */
output: out ; /* node texture at level 0 */

1  $D_{0,0\dots 2,2} \leftarrow \text{depth}$  ; /* read 3x3 depth neighborhood */
2 /* discontinuity hint computed via Laplacian thresholding */
3  $O \leftarrow \text{step}(\lambda_d, \text{getLaplacian}(D))$ 
4 /* compute forward and backward differentials */
5  $df_{xy} \leftarrow \text{vec2}(D_{2,1} - D_{1,1}, D_{1,2} - D_{1,1})$ 
6  $db_{xy} \leftarrow \text{vec2}(D_{1,1} - D_{0,1}, D_{1,1} - D_{1,0})$ 
7 /* enforce smoothness by picking the smallest derivative */
8  $d_{xy} \leftarrow \text{mix}(df_{xy}, db_{xy}, \text{abs}(df_{xy}) < \text{abs}(db_{xy}))$ 
9 /* zero large derivatives that connect different surfaces */
10  $d_{xy} \leftarrow \text{step}(\lambda_h, \text{abs}(d_{xy})) \cdot d_{xy}$ 
11 /* compute normal */
12  $\vec{N} \leftarrow \text{normalize}(\text{cross}(\text{vec3}(P_{\text{size}.x}, 0, d_x), \text{vec3}(0, P_{\text{size}.y}, d_y)))$ 
13 /* compute plane's top-left corner z-coordinate */
14  $P \leftarrow D_{1,1} - \text{dot}(\vec{N}_{xy}/\vec{N}_z, -0.5 \cdot P_{\text{size}})$ 
15 out  $\leftarrow \text{outputPlane}(\vec{N}, P, O)$  ; /* output a plane node */

```

We compare in Fig. 1 the depth reconstruction quality of our method against a gold-standard GPU ray tracer—NVIDIA OptiX [Parker et al. 2010]. Even though we only use two depth layers in this example, our approach correctly evaluates the depth at all pixels (except where three layers would be required), while being $3\times$ faster than general-purpose ray tracer. Note that our approach allows us to inpaint the remaining holes, so that no artifacts appear. Alternatively, one can simply use more layers.

References

PARKER, S. G., BIGLER, J., DIETRICH, A., FRIEDRICH, H., HOBEROCK, J., LUEBKE, D., MCALLISTER, D., MCGUIRE, M., MORLEY, K., ROBISON, A., AND STICH, M. 2010. OptiX: A general purpose ray tracing engine. *ACM Trans. Graph.* 29, 4.

Algorithm 2: GLSL pseudo-code for construction and compression of a single level of the quad-tree.

```

input : in ; /* node texture at level i (i > 0) */
output: out ; /* node texture at level i-1 */

1  $Q_{0\dots 3} \leftarrow \text{in}$  ; /* read 2x2 neighboring nodes */
2 if containOnlyPlanes ( $Q_{0\dots 3}$ ) then
3   /* get plane normal vector, origin and "discontinuity" flag */
4    $(\vec{N}, P, O)_{0\dots 3} \leftarrow \text{getPlaneData}(Q_{0\dots 3})$ 
5   /* set proxy plane normal to mean of children normals */
6    $\vec{N}_{\text{proxy}} \leftarrow \text{normalize}(\text{mean}(\vec{N}_{0\dots 3}))$ 
7   /* compute angle differences via dot-product */
8    $\text{float } d_{0\dots 3} \leftarrow 1 - \text{dot}(\vec{N}_{\text{proxy}}, \vec{N}_{0\dots 3})$ 
9   /* proxy plane origin  $P_{\text{proxy}}$  is least-square fit to child planes with fit errors stored in  $p_{0\dots 3}$  */
10   $(P_{\text{proxy}}, p_{0\dots 3}) \leftarrow \text{getPlaneOrigin}(\vec{N}_{\text{proxy}}, \vec{N}_{0\dots 3}, P_{0\dots 3})$ 
11  /* output a plane node if the proxy is close enough to children in terms of orientation and position */
12  if  $\max(d_{0\dots 3}) < \gamma_{\text{norm}}$  and  $\max(p_{0\dots 3}) < \gamma_{\text{dist}}$  then
13    out  $\leftarrow \text{outputPlane}(\vec{N}_{\text{proxy}}, P_{\text{proxy}}, \text{any}(O_{0\dots 3}))$ 
14    return
15  end
16 end
17 /* output AABB node that encompasses all children */
18  $\text{vec2 } Z_{0\dots 3} \leftarrow \text{getMinMaxZ}(Q_{0\dots 3})$ 
19  $\text{float } \min_z \leftarrow \min(Z_{0\dots 3}.x)$ 
20  $\text{float } \max_z \leftarrow \max(Z_{0\dots 3}.y)$ 
21 out  $\leftarrow \text{outputAABB}(\min_z, \max_z)$ 

```

Algorithm 3: GLSL pseudo-code for ray traversal through a single depth layer. For brevity, we assume the quad-tree MIPMAP has power-of-two size.

```

input :  $T_0 \dots n-1$  ; /* texture MIPMAP storing depth quad-tree */
input :  $R$  ; /* ray structure storing direction and origin */
output: bool rayHit ; /* trace result */
output: bool occlusionHit ; /* did we hit an occlusion volume? */
output: float d ; /* hit-point distance along the ray */
output: vec4 plane ; /* hit-plane data */

1 int  $Q_{level} \leftarrow n - 1$  ; /* current quad-tree level, start at the root */
2 /* current node position in the quad-tree */
3 ivec2  $Q_{xy} \leftarrow \lfloor R.origin.xy * sizeof(T_0) / 2^{Q_{level}} \rfloor$ 

4 while insideBounds( $pos, T_{Q_{level}}$ ) do
5     vec2  $Q_{data} \leftarrow T_{Q_{level}}(Q_{xy})$  ; /* read the node data */
6     if nodeStoresPlane( $Q_{data}$ ) then
7         plane  $\leftarrow$  getPlaneData( $Q_{data}$ )
8         /* Get far and near Z of node */
9         FandN  $\leftarrow$  getFarNearOfNode( $R, Q_{xy}, Q_{level}$ )
10         $\vec{N} \leftarrow plane.xyz$  ; /* plane normal */
11         $P_0 \leftarrow vec3(Q_{xy} / 2^{Q_{level}}, plane.w)$  ; /* and origin */
12        /* compute ray-plane intersection */
13         $d \leftarrow dot(P_0 - R.origin, \vec{N}) / dot(R.dir, \vec{N})$ 
14        if  $dot(R.dir, \vec{N}) > 0$  then /* plane is front-facing the ray */
15            if  $d < FandN.near$  then occlusionHit  $\leftarrow$  true ;
16            if  $d \geq FandN.near$  and  $d < FandN.far$  then
17                rayHit  $\leftarrow$  true
18                plane  $\leftarrow$  vec4( $\vec{N}, dot(P_0, \vec{N})$ )
19                return
20            end
21        else /* plane is back-facing the ray */
22            if  $d \geq FandN.near$  then occlusionHit  $\leftarrow$  true ;
23            end
24        else
25            /* compute intersection with both node bounding box and occlusion volume */
26            ( $hitAABB, hitOV$ )  $\leftarrow$  rayIntersectAABB( $R, Q_{data}$ )
27            if  $hitAABB$  then
28                if  $hitAABB.near = hitOV.far$  then
29                    occlusionHit  $\leftarrow$  true
30                end
31                /* progress down to the next child */
32                 $ip \leftarrow R.dir.xy * hitAABB.near + R.origin.xy$ 
33                 $Q_{xy} \leftarrow Q_{xy} * 2 + step(0, ip - (Q_{xy} + 0.5) / 2^{Q_{level}})$ 
34                 $Q_{level} \leftarrow Q_{level} - 1$ 
35                continue
36            else
37                if  $hitOV$  then occlusionHit  $\leftarrow$  true ;
38            end
39        end
40        /* plane or AABB miss, progress to the next node */
41        /* current node successor position at  $Q_{level}$  */
42         $Q_{xy}^* \leftarrow$  getNextNode( $R, Q_{xy}, Q_{level}$ )
43        /* compute how many levels up we need to go */
44        int  $levelShift \leftarrow findMSB((Q_x \oplus Q_x^*) | (Q_y \oplus Q_y^*))$ 
45        /* prevent the traversal from going above the quad-tree root */
46         $Q_{level}^* \leftarrow \min(Q_{level} + levelShift, n - 1)$ 
47        /* update the node location and level to new values */
48         $Q_{xy} \leftarrow \lfloor Q_{xy}^* / 2^{(Q_{level}^* - Q_{level})} \rfloor$ 
49         $Q_{level} \leftarrow Q_{level}^*$ 
50    end
51 rayHit  $\leftarrow$  false

```

Algorithm 4: GLSL pseudo-code for finding coordinates of the next node that intersects with the ray in screen-space.

```

Function getNextNode(Ray  $R$ , ivec2  $Q_{xy}$ , int  $Q_{level}$ )
/* get node exit corner coordinate */
1 vec2  $B \leftarrow (Q_{xy} + step(0, R.dir.xy)) / 2^{Q_{level}}$ 
2 /* get distances to node edges that intersect at  $B_{xy}$  */
3 vec2  $D \leftarrow (B - R.origin.xy) / R.dir.xy$ 
4 /* compute position shift */
5 ivec2  $S_{xy} \leftarrow sign(R.dir.xy) * step(D.xy, D.yx)$ 
6 return  $Q_{xy} + S_{xy}$  ; /* return new position */

```



Figure 1: A comparison of depth reconstruction quality. (a) A new view synthesized with our method (using two depth-layers). The full-image took 6.5ms to render. The synthesized depth for (b) two-layer and a (c) single-layer configuration. (d) The reference was generated with NVIDIA OptiX in about 20ms. In both cases we report combined construction and tracing times.