# Rendering Decals and Many Lights with Ray Tracing Acceleration Structures

Sidney Hansen
sidney.hansen@student.kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Christoph Peters
christoph.peters@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

**Figure 1: Left: The bistro with decals. Right: Emerald Square with 15k lights. The regions of influence are axis-aligned bounding boxes (AABBs) in an acceleration structure (right half). By tracing zero-length rays, we enumerate relevant decals and lights.**

## ABSTRACT

Methods for rendering decals and local light sources often rely on rasterising bounding volumes. While beeing efficient, this only works in deferred shading. We propose a new approach, that makes use of ray-tracing acceleration structures. The core problem is the enumeration of decals and lights with overlapping regions of influence per shading point. Thus, we build an acceleration structure with one axis-aligned bounding box per decal or light. We trace a zero-length ray from the shading point such that any hit yields a relevant decal or light. Our approach is more costly but also more widely applicable than classic deferred shading techniques.

## CCS CONCEPTS

• **Computing methodologies** → **Texturing**; **Ray tracing**.

## KEYWORDS

Decals, many lights, real-time rendering, ray tracing, ray query, nearest neighbors, deferred decals, visibility buffers


*i3D 2021 posters, 20-22 April 2021, online*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/none

**ACM Reference Format:**
Sidney Hansen and Christoph Peters. 2021. Rendering Decals and Many Lights with Ray Tracing Acceleration Structures. In *Proceedings of 25th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (i3D 2021 posters).* ACM, New York, NY, USA, 2 pages. https://doi.org/none

## 1 INTRODUCTION

Decals and local light sources are widely used in real-time rendering. Their use is particularly efficient in deferred rendering. A simple way to apply them is to rasterize bounding volumes for their regions of influence, e.g. a box for decals or a low-poly sphere for point lights. The fragment shader for decals then samples decal textures and blends them with the g-buffer [Kim, 2012]. For lights, it samples the g-buffer, computes shading and adds it to the framebuffer.

This approach is fast but not particularly flexible. For decals, it requires a complete g-buffer. Alternatives like visibility buffers [Burns and Hunt, 2013] are not supported. Transparent surfaces and surfaces in ray-traced reflections are also problematic. For forward rendering with depth prepass, there are alternatives such as tiled and clustered shading where a 2D or 3D grid holds lists of decals or lights per cell [Olsson et al., 2012]. However, these grids fail for ray-traced reflections because they usually only cover the view frustum.

Our approach can apply decals or shading with many lights directly to a given shading point anywhere in the scene and in any shader stage. Instead of regular grids, we leverage hardware-acceleration for ray tracing. The method is more costly than deferred decals but also far more flexible.

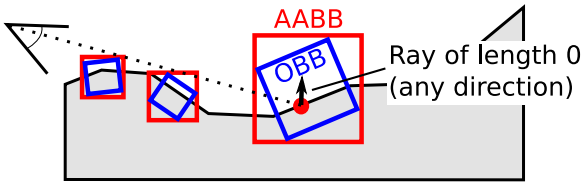


**Figure 2: Decals project onto surfaces within oriented bounding boxes (OBBs). We create acceleration structures with AABBs and trace zero-length rays to enumerate all relevant decals for a given shading point.**

## 2 ENUMERATION AND BLENDING OF DECALS AND MANY LIGHTS

Bottom-level acceleration structures (BLAS) either hold triangles or AABBs. While AABBs are originally meant to be used with intersection shaders, we repurpose them for decals and lights. A decal affects all surfaces within an OBB (Fig. 2) whereas a point light has a sphere of influence. From these regions, we construct one AABB per decal and light. All AABBs go into BLASs. We can combine multiple BLASs in a single top-level acceleration structure (TLAS), which facilitates instancing and rigid animations. Decals and lights use separate TLASs.

Parameters and texture descriptors for decals and lights are packed into large arrays that we access during shading with the index of the AABB. In order to apply decals and compute lighting we have to enumerate all overlapping decals and light sources during the shading pass. To this end, we use ray queries in a fragment shader. We trace a ray of length zero starting at the position of the fragment (Fig. 2). Any hit means that one of the AABBs overlaps the shading point. In that case, we immediately use its index to retrieve its attributes from the array. We proceed to check if the fragment is contained in the tighter bounding volume. If so, we perform shading for light sources.

For a decal, we sample its textures. As with deferred decals, we have to compute derivatives of texture coordinates manually by transforming screen space derivatives of the world space position. With visibility buffers, that causes mipmapping errors at depth discontinuities. Deferred decals are prone to the same issue.

The order in which our approach enumerates decals is arbitrary. Thus, to combine overlapping decals we use different weights $w_i$ and layers. To place a decal on top of others, this decal can be assigned a higher layer. These layers are then blended from bottom to top. If multiple decals are asigned to one layer, they are blended, based on their weight. For this we use weighted-blended order independent transparency (OIT) [McGuire and Bavoil, 2013]. For each attribute $c_0$ of the shading data (albedo, normal, roughness or metallicity), we compute the blended attribute by

$$\frac{\sum_{i=1}^{n} c_i \alpha_i w_i}{\sum_{i=1}^{n} \alpha_i w_i} \left(1 - \prod_{i=1}^{n} (1 - \alpha_i)\right) + c_0 \prod_{i=1}^{n} (1 - \alpha_i) \,.$$

The attribute value $c_i$ comes from the decal textures, just like the opacity $\alpha_i$. This approach lets us handle any number of overlapping decals, while giving us flexibility, in cases where weighted-blended OIT produces unnatural results.

**Table 1: Total frame time in ms**

| Nr | Scene | Decals | Local lights |
|---|---|---|---|
| 1. | Sun temple | 6 large | 0 |
| 2. | Bistro | 100 small | 0 |
| 3. | Emerald square | 100k small | 0 |
| 4. | Emerald square | 0 | 28k small |

| Forward | Off | Ours | Vis. buffer | Off | Ours |
|---|---|---|---|---|---|
| 1. | 0.75 | 1.20 | 1 | 0.65 | 0.99 |
| 2. | 2.90 | 3.06 | 2 | 1.95 | 2.07 |
| 3. | 5.1 | 16.0 | 3 | 2.7 | 5.0 |
| 4. | - | 27.8 | 4 | - | 10.8 |

| Deferred | Off | Deferred Decals | Ours |
|---|---|---|---|
| 1. | 0.74 | 0.89 | 1.16 |
| 2. | 2.18 | 2.30 | 2.32 |
| 3. | 3.1 | 5.5 | 6.4 |

## 3 RESULTS

For evaluation, we run our renderer on an NVIDIA RTX 2070 Super. Our technique uses the VK_KHR_ray_tracing extension. Figure 1 demonstrates that our method can faithfully handle many decals and lights. Table 1 provides timings to assess the overhead of our approach and to compare it to deferred decals. We use forward rendering with depth prepass, deferred rendering with a 112-bit g-buffer and deferred rendering with a 32-bit visibility buffer [Burns and Hunt, 2013].

With a low number of decals, all variants have a modest overhead. As we push towards tens of thousands of lights or decals, the cost of our method or deferred decals dominates the frame time, especially in forward rendering, but rendering remains highly interactive. Deferred decals are usually faster than our method.

Thus, our approach is not attractive as replacement for deferred decals at this time. However, it is compelling in situations where deferred decals are not applicable, especially with visibility buffers or ray tracing. A direct comparison to clustered shading would be interesting future work. The source code of our renderer is available on github: https://github.com/scratlantis/vulkan-renderer

## ACKNOWLEDGMENTS

The Emerald Square was created by Nicholas Hull for NVIDIA. The bistro and sun temple were donated by Amazon Lumberyard and Epic Games, respectively. Parts of our Vulkan renderer are based on examples by Sascha Wilhelms.

## REFERENCES

Christopher A. Burns and Warren A. Hunt. The visibility buffer: a cache-friendly approach to deferred shading. *Journal of Computer Graphics Techniques (JCGT)*, 2 (2):55–69, 2013.

Pope Kim. Screen space decals in warhammer 40,000: Space marine. In *ACM SIGGRAPH 2012 Talks*, SIGGRAPH '12, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450316835. doi: 10.1145/2343045.2343053. URL https://doi.org/10.1145/2343045.2343053.

Morgan McGuire and Louis Bavoil. Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques (JCGT)*, 2(2):122–141, December 2013. ISSN 2331-7418. URL http://jcgt.org/published/0002/02/09/.

Ola Olsson, Markus Billeter, and Ulf Assarsson. Clustered deferred and forward shading. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, pages 87–96, 2012.