




Transfer Textures for Fast Precomputed Radiance Transfer

Sirikonda Dhawal , Aakash KT , P.J. Narayanan 

CVIT, KCIS, IIIT-Hyderabad, Telangana, India

Abstract

Precomputed Radiance Transfer (PRT) can achieve high-quality renders of glossy materials at real-time framerates. PRT involves precomputing a k -dimensional transfer vector or a $k \times k$ -matrix of Spherical Harmonic (SH) coefficients at specific points for a scene depending on whether the material is diffuse or glossy respectively. Most prior art precomputes values at vertices of the mesh and interpolates color for interior points. They require finer mesh tessellations for high-quality renders. In this work, we introduce transfer textures for decoupling mesh resolution from transfer storage and sampling specifically benefiting the glossy renders. Dense sampling of the transfer is possible on the fragment shader while rendering with the use of transfer textures for both diffuse as well as glossy materials, even with a low tessellation. This simultaneously provides high render quality and frame rates.

CCS Concepts

• **Computing methodologies** → **Rasterization; Ray tracing; Visibility;**

1. Introduction & Related Work

Precomputed Radiance Transfer (PRT) offloads expensive computations of ray tracing to a pre-computation step, after which the stored data can be utilized for real-time photorealistic rendering. The core PRT framework as proposed by [SKS02] precomputes the transfer function and stores it in SH basis at vertices of the scene. The focus of this work is the storage of transfer in a texture instead, effectively decoupling mesh resolution from transfer sampling and storage. Prior works like [McK10] and PRT of D3D9 leverage the continuous texture space to store transfer but are limited to diffuse reflection, due to their choice of formulation [SKS02] and extending their work directly will lead to heavy texture storage (Tbl. 1). Usage of transfer textures facilitates the evaluation of color for each fragment in a fragment shader, in contrast to current PRT approaches that evaluate color for each vertex in a vertex shader. The advantage of a fragment shader-based approach is that it improves render quality, especially for lowly tessellated meshes (Fig. 2). We describe methods to correctly compute transfer textures. We augment the triple product method with our transfer textures and demonstrate real-time framerates and superior render quality. We also formulate and demonstrate inter-reflections using transfer textures by completely fixing the light. Our work can be seen as an extension of Textured-PRT [McK10] for glossy materials and inter-reflections without incurring additional memory-budget (Tbl. 1).

2. Computing Transfer Textures

The transfer computation involves shooting multiple rays from a point p in the scene and then evaluating and projecting the transfer to the SH basis. For transfer textures, there are N scene points p

corresponding to each texel t in the texture. The mapping between t and p is defined by the UV coordinates. To efficiently compute the transfer texture, we leverage G-Buffers to interpolate surface positions and normals based on their corresponding UV-Coord. (Alg. 1, line 2). We read the G-buffer and the scene geometry and evaluate the transfer function for each texel in the buffer (Alg. 1, lines 4-6). The transfer obtained is then projected to SH basis and stored at that texel in T_o (Alg. 1, lines 7-8). Finally, T_o is dilated to ensure that all points inside a triangle receive a transfer value. At run-time, we fetch transfer T_o and use it with the triple product formulation to obtain the color. This allows storage of k -dimensional vector instead of $k \times k$ -matrix for each texel.

ALGORITHM 1: Pre-computing and storing the transfer texture.

```

Input:  $\mathcal{M}, w, h, l$ : Mesh  $\mathcal{M}$ , width  $w$  & height  $h$ , SH band  $l$ .
Output:  $T_o$ : Precomputed transfer texture
1  $T \leftarrow \text{Texture}(w, h, l)$  // Init. texture.
2  $\mathcal{G} = \text{OpenGL}(\mathcal{M})$  // G-Buffer
3 for  $t$  in  $T$  do
4   point, normal =  $\mathcal{G}[\text{t.x}][\text{t.y}].\text{vertex}, \mathcal{G}[\text{t.x}][\text{t.y}].\text{normal}$ 
5    $V = \text{ComputeTransfer}(\text{point}, \text{normal})$  // Path tracing
6    $V_{sh} = \text{SHProject}(V)$ 
7    $T_o[\text{t.x}][\text{t.y}] = V_{sh}$ 
8  $T_o = \text{Dilate}(T_o, 3)$ 

```

2.1. Incorporating inter-reflections

Inter-reflected radiance B_i^p at point p can be modeled as:

$$B_i^p(\omega_o) = \int_{\Omega} (1 - V^p(\omega_i)) B^{pa}(x, \omega_i) \rho^p(\omega_o, \omega_i) (\omega_i \odot n) d\omega_i, \quad (1)$$

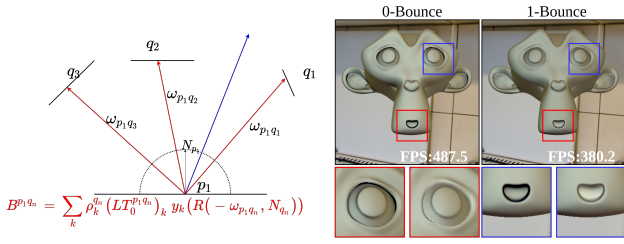


Figure 1: (a) *Interreflection Method*(left), (b) *Interreflection Results*(left:0-bounce, right:1-bounce)

where B^{pq} is the radiance from a secondary hit-point q towards p . First, we factor out $1 - V^p(x, \omega_i)$ by only integrating over rays which hit some geometry. For a scene point p_1 and a secondary hit q_1 , the radiance $B^{p_1q_1}$ can easily be precomputed given a zero-bounce transfer texture T_o from Alg. 1 (See Fig. 1a). The radiance from q_1 towards p_1 is obtained using the triple product formulation by fetching T_o to obtain transfer at q_1 . This is done for all hit-points from p_1 . This radiance now forms an *indirect environment map* for the point p_1 , which is then projected to SH basis resulting in a k -vector B_i^{pq} , which is stored in a separate *inter-reflection* texture T_1 . At run-time, the indirect radiance can be obtained by convolving B_i^{pq} fetched from T_1 with the BRDF SH ρ_i^p and evaluating at the reflection direction.

3. Implementation

We implement Alg. 1 using Embree. The resulting transfer texture is then used in OpenGL shaders for rendering. We use texture resolution of 1024×1024 . We implement the triple product method with our transfer textures on a fragment shader with an early depth pass. All scenes are rendered with glossy materials for band $l = 5$ (25-coefficients) SH projection with an NVIDIA GPU at a resolution of 1920×1080 . The scene performance comparisons and configurations are given in Fig. 2 and Tbl. 1 respectively.

4. Results & Evaluation

We first show the results of the Triple Product (TP) method and Triple Product with Fixed Light (TPFL) method using our transfer

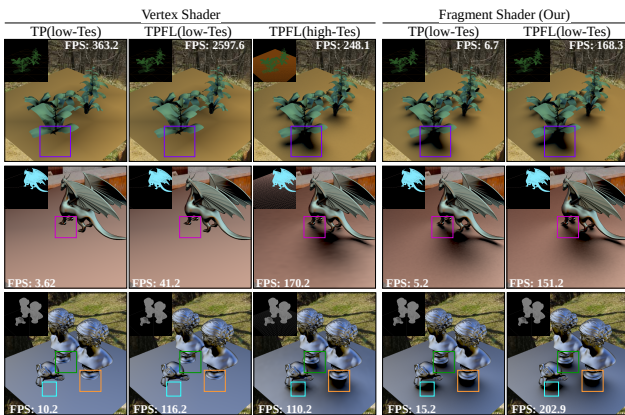


Figure 2: *Results*

Table 1: *Memory Requirements*

Scene	# tris.	Vert. Mem.		Tex. Mem.	
		[SKS02]	[NRH04]	[McK10]	Ours
Dragon	1.3M	5.2GB	215.8MB	2.5GB	100MB
TRM	441K	2.3GB	139.3MB	2.5GB	100MB
Plants	18K	64MB	2.5MB	2.5GB	100MB

textures. Fig. 2 shows the renders for three scenes: *Plants*, *Dragon* and *TRM* (*Two Roza, one Monkey*). All scenes have a ground plane, which is minimally tessellated, as shown in the wireframe insets. The TP and TPFL methods on vertex shader are unable to capture proper shadows on the ground plane due to sparse sampling of the transfer. In contrast, the TP method on the fragment shader using our transfer textures properly reproduces shadows on the plane, albeit at a very low FPS. The TPFL method with transfer textures also achieves a similar render quality at a higher FPS. The TP method on vertex shader approaches the render quality of our transfer textures with a highly tessellated ground plane. We note that this requires the addition of *redundant* vertices. We further note that such situations frequently arise in production, for example, on large surfaces with minimal curvature (walls, floor). In such cases, all previous PRT methods on vertex shaders require the redundant vertices to store the transfer, in contrast to our transfer textures method. Next, we demonstrate inter-reflections using transfer textures with the method described in Sec 2.1. The zero-bounce and one-bounce renders with their corresponding FPS are shown in Fig. 1b for *Monkey*. Because of the additional texture fetch, convolution and evaluation operations, the FPS with inter-reflections is slightly lower (albeit still real-time).

5. Conclusion, Limitations & Future work

In this paper, we presented *transfer textures* for decoupling mesh tessellation from transfer sampling and storage to accommodate glossy materials and inter-reflections. We described methods to efficiently and correctly compute these textures and also demonstrated the incorporation of inter-reflections in a restricted setting. Finally, we demonstrated real-time framerates for rendering with transfer textures on the fragment shader and superior render quality for minimally tessellated meshes. A limitation of transfer textures is that inter-reflections essentially bake the lighting and BRDF, i.e. they cannot be changed without re-computation. We note that the work of [SKS02] also bakes BRDF (including albedo) into their transfer matrices for inter-reflections. We would like to address this issue for future extensions of this work. A more comprehensive version of this work can be found [here](#)

References

- [McK10] MCKENZIE CHAPTER, HARRISON LEE. “Textured Hierarchical Precomputed Radiance Transfer”. (2010) 1, 2.
- [NRH04] NG, REN, RAMAMOORTHY, RAVI, and HANRAHAN, PAT. “Triple Product Wavelet Integrals for All-Frequency Relighting”. *ACM Trans. Graph.* (Aug. 2004), 477–487 2.
- [SKS02] SLOAN, PETER-PIKE, KAUTZ, JAN, and SNYDER, JOHN. “Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments”. *ACM Trans. Graph.* 21.3 (July 2002), 527–536. ISSN: 0730-0301 1, 2.