PRTT: Precomputed Radiance Transfer Textures

SIRIKONDA DHAWAL, CVIT, KCIS, IIIT-Hyderabad, India
AAKASH KT, CVIT, KCIS, IIIT-Hyderabad, India
P.J. NARAYANAN, CVIT, KCIS, IIIT-Hyderabad, India

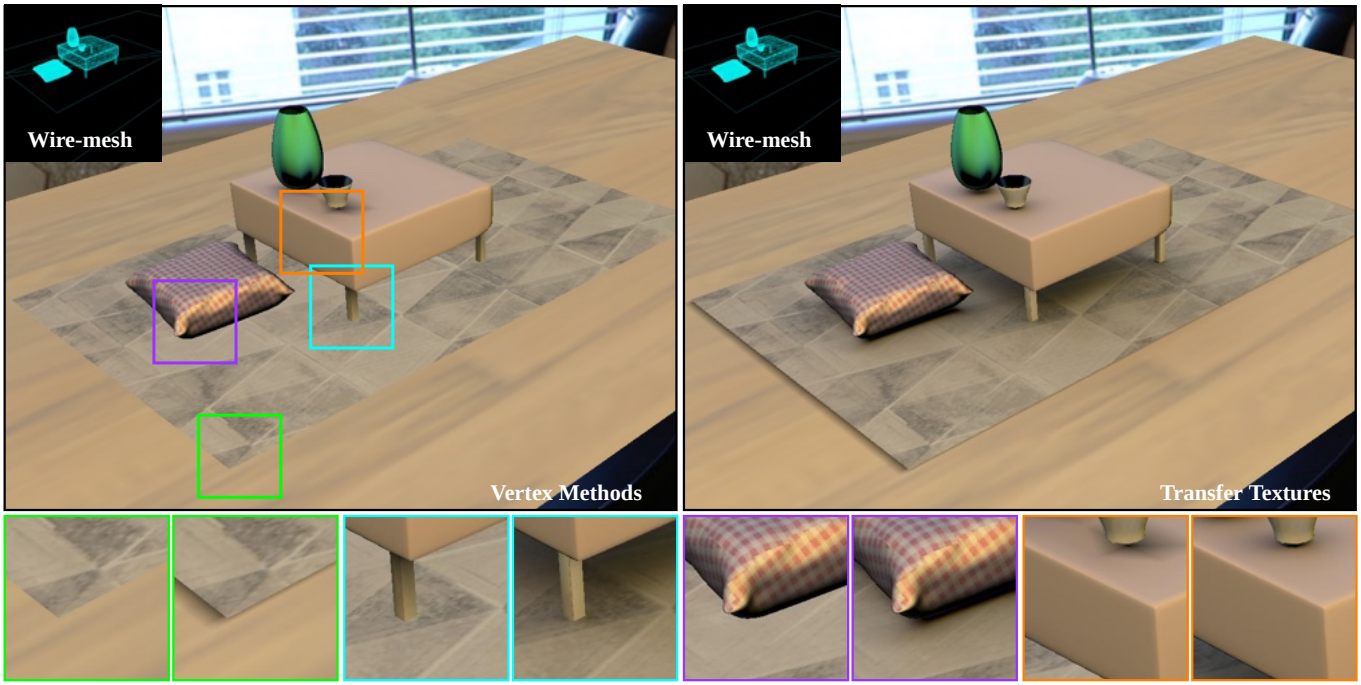

Fig. 1. We present transfer textures to decouple mesh resolution from transfer storage and sampling. Our method precomputes and stores transfer in a texture that is sampled in a fragment shader. The above scene has a minimally tessellated rug, floor and table (shown in wireframe insets). Traditional triple product method on vertex shader (left) is unable capture shadows due to insufficient sampling of the transfer function. In contrast transfer textures run on the fragment shader and accurately captures all details for the same tessellation. Note that the scene has spatially varying color and roughness.

Precomputed Radiance Transfer (PRT) can achieve high quality renders of glossy materials at real-time framerates. PRT involves precomputing a k-dimensional transfer vector of Spherical Harmonic (SH) coefficients at specific points for a scene. Most prior art precomputes transfer at vertices of the mesh and interpolates color for interior points. They require finer mesh tessellations for high quality renderings. In this paper, we explore and present the use of textures for storing transfer. Using *transfer textures* decouples mesh resolution from transfer storage and sampling which is useful especially for glossy renders. We further demonstrate glossy inter-reflections by precomputing additional textures. We thoroughly discuss practical aspects of transfer textures and analyze their performance in real-time rendering applications. We show equivalent or higher render quality and FPS and demonstrate results on several challenging scenes.

CCS Concepts: • **Computing methodologies** → **Rasterization**; **Ray tracing**; *Visibility*;

Additional Key Words and Phrases: Transfer, fragment-shader, Visibility, Textures

Authors' addresses: Sirikonda Dhawal, CVIT, KCIS, IIIT-Hyderabad, Hyderabad, Telangana, India, 500032, dhawal.sirikonda@research.iiit.ac.in; Aakash KT, CVIT, KCIS, IIIT-Hyderabad, Hyderabad, Telangana, India, 500032, aakash.kt@research.iiit.ac.in; P.J. Narayanan, CVIT, KCIS, IIIT-Hyderabad, Hyderabad, Telangana, India, 500032, pjn@iiit.ac.in.

1 INTRODUCTION

Ray Tracing is the method of choice for high-fidelity image generation. However, it is computationally expensive for real-time applications where only a few milliseconds of rendering time is available per-frame. 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. PRT uses Spherical Harmonic (SH) lighting [Ramamoorthi 2009] to efficiently store and render complex effects in both video games and offline rendering for movie production [Pantaleoni et al. 2010]. Recently, PRT has been extended to efficiently handle area light sources [Belcour et al. 2018; Wang and Ramamoorthi 2018; Wu et al. 2020] further increasing its utility.

The PRT framework proposed by [Sloan et al. 2002] pre-computes the *transfer* function and stores it in SH basis at vertices of the scene, possibly with compression [Sloan et al. 2003]. For a band l

SH projection with $k = l^2$ coefficients, this amounts to storing a k-dimensional vector for diffuse and $k \times k$-dimensional matrix for glossy materials at each vertex. Using the Triple Product formulation [Ng et al. 2004], it is possible to still store a k-dimensional vector per-vertex for glossy materials by additionally storing a global three-dimensional *tripling coefficient matrix*. A further modification of triple products fixes the light [Sloan 2008] and requires the storage of a two-dimensional global matrix. The advantage of this method is that it retains a compute complexity of $O(k^2)$ as opposed to $O(k^{5/2})$ of traditional triple products. Recently, the work of [Xin et al. 2021] reduces this time complexity to $O(k^{3/2})$, although a real benefit is gained only for large k.

The above PRT approaches store the transfer vectors/matrices on vertices of the mesh. While rendering, color is evaluated at each vertex and interpolated for internal points. This necessitates a reasonably dense mesh tessellation (high mesh resolution) for high-quality renders. There exists few methods like textured hierarchical PRT [McKenzie Chapter 2010] and PRT of D3D9 framework [Microsoft 2003] which leverage the continuous texture space to store transfer. The work of [Iwanicki and Sloan 2009] also uses texture space similar to [McKenzie Chapter 2010] with a specific focus on shadows. These methods focus only on diffuse reflection and do not demonstrate glossy reflection and inter-reflections with textures. Directly extending above mentioned methods for glossy renders requires k^2 coefficients per each texel resulting in heavy texture storage, because they use the original PRT formulation of [Sloan et al. 2002]. These extensions are thus infeasible in real-time scenarios (Table 1). Can methods like Textured-PRT be extended to handle glossy materials, inter-reflections, etc., at real-time rendering rates?

In this paper, we present *precomputed radiance transfer textures* (transfer textures) to efficiently store the transfer values using k coefficients per texel for both diffuse and glossy reflections. Our method uses the triple product formulation of PRT [Ng et al. 2004]. Transfer textures store more finely sampled transfer values and can evaluate color for each fragment in a fragment shader. This improves render quality even for coarsely tessellated meshes (Fig. 1). They can also support high-quality local effects like glossy renders, inter-reflections, and normal maps. Higher texture resolution results in greater precomputation effort but higher render quality at fixed render times. This provides a way to balance computation effort and rendering quality and to trade one off for the other. Texture space techniques like mip-mapping and texture sets can be used for greater efficiency. We describe methods to correctly and efficiently compute transfer textures and show real-time framerates and superior render quality at low mesh tessellations for several scenes. We further formulate and demonstrate inter-reflections using transfer textures by completely fixing the light. We also show transfer texture facilitate local effects like normal mapping. Finally, we compare and analyze run-times and storage against vertex-based approaches with the triple product method.

2 RELATED WORK

Precomputed Radiance Transfer (PRT). PRT was first proposed by [Sloan et al. 2002], where they projected environment lighting and light transport to SH basis for dynamic lighting for diffuse and glossy materials. Since then, PRT and SH have received lot of attention to efficiently compute SH basis [Snyder 2006], efficient rotation of SH [Nowrouzezahrai et al. 2012], compressing SH basis [Sloan et al. 2003], microfacet BRDFs [Lehtinen and Kautz 2003] and extending PRT for dynamic scenes [Zhou et al. 2005]. More recently, [Wang and Ramamoorthi 2018] extend PRT to support area lights achieving real-time frame rates for a few light sources. This work was later extended to support a large number of area lights while maintaining real-time frame rates [Wu et al. 2020]. All of these methods were either orthogonal to the core PRT framework or extended the core framework to support additional scenarios. In contrast, we improve the core PRT framework to compute and store transfer on a texture instead of at vertices.

Triple Products. Triple products naturally arise in computer graphics in the rendering equation. Triple products in wavelet basis and spherical harmonics have been studied in depth [Ng et al. 2004]. Today, state-of-the art in PRT uses triple products for dynamic relighting for diffuse and glossy scenes. By itself, triple product method has a compute complexity of $O(k^3)$. This method can be made more computationally efficient by fixing the lighting [Sloan 2008]. Specifically, triple products with fixed lighting in SH based PRT achieves a compute complexity of $O(k^2)$ and per-vertex storage of k-dimensional vector. We augment the triple product method with transfer textures and demonstrate superior rendering quality and real-time framerates.

Transfer stored on Textures. Storage of transfer on a texture was first suggested by [Sloan et al. 2002]. [Iwanicki and Sloan 2009; McKenzie Chapter 2010] formally demonstrated transfer storage on textures. These methods mainly focused on diffuse materials without inter-reflections. In this work, we focus primarily on glossy materials and also demonstrate inter-reflections with transfer textures.

3 BACKGROUND ON PRT WITH TRIPLE PRODUCTS

To make our document self-contained, we first review and discuss Precomputed Radiance Transfer with triple products, which is the current state of the art in PRT. The inspiration for the choice of using triple product formulation is discussed in Sect. 6.3. The rendering equation for direct lighting at point p is given by:

$$B^p(\omega_o) = \int_\Omega L(\omega_i)\rho^p(\omega_o, \omega_i)V^p(\omega_i)(\omega_i \odot n)d\omega_i, \qquad (1)$$

where ω_o is the direction towards the viewer from p, ω_i is the incoming direction on the unit hemisphere Ω and n is the surface normal at p. B^p is the reflected radiance in direction ω_o, L is the incoming environment light from ω_i, V^p is the binary visibility function and ρ^p is the Bi-directional Reflectance Distribution Function (BRDF). Eq. 1 is decomposed into the lighting L and transfer $T^p(\omega_i) = V^p(\omega_i)(\omega_i \odot n)$ which are then projected to the SH basis with coefficients T_i^p and L_i respectively. Triple products [Ng et al. 2004] formulate transferred radiance as:

$$L_k^p = \int_{S^2} y_k(\omega)\left(\sum_{i=1}^{n^2} T_i^p y_i(\omega)\right)\left(\sum_{j=1}^{n^2} L_j y_j(\omega)\right)d\omega = \sum_{ij} \tau_{ijk} T_i^p L_j, \quad (2)$$

B^p where τ_{ijk} is the triple product tensor (tripling coefficient matrix). The final color is calculated by convolution of L_k^p with BRDF coefficients and evaluation at reflection direction. The above formulation for PRT results in a k-vector for transfer stored at each point and $O(k^{5/2})$ compute [Ng et al. 2004] required for rendering. The compute efficiency can be further improved to $O(k^2)$ by fixing the light and precomputing a global product matrix: $(M)_{ik} = \sum_j \tau_{ijk} L_j$ [Sloan 2008].

4 TRANSFER TEXTURES

In this section, we begin with a description of computing and storing a band l SH projection of transfer on a texture (Sec 4.1). Next, we show how inter-reflections can be pre-computed and incorporated in our framework (Sec 4.2). Our implementation is described in Sec 5. Our approach achieves real-time frame rates and better render quality, especially on low tessellation meshes, as shown in Sec 6.

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_o ← Texture(w, h, l)                          // Init. texture.
2 G = OpenGL(M)                                   // G-Buffer
3 for t in T_o do
4     point = G[t.x][t.y].vertex
5     normal = G[t.x][t.y].normal
6     V = ComputeTransfer(point, normal)          // Path tracing
7     V_sh = SHProject(V)
8     T_o[t.x][t.y] = V_sh
9 T_o = Dilate(T_o, 3)
```

4.1 Pre-computing Transfer Textures

The computation of transfer 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 pixel 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(Alg. 2) to interpolate vertex positions and normals based on their corresponding UV-Coordinates (Alg. 1, line 2). Next, we read the G-buffer and the scene geometry and evaluate the transfer function for each pixel in the buffer (Alg. 1, lines 4-6). The transfer obtained is then projected to SH basis and stored at the same pixel location in a initially empty texture T_o (Alg. 1, lines 7-8). Finally, T_0 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 B^p.

4.2 Pre-computing transfer textures for inter-reflections

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

$$B_i^p(\omega_o) = \int_\Omega (1 - V^p(\omega_i)) B^{pq}(x, \omega_i) \rho^p(\omega_o, \omega_i)(\omega_i \odot n) d\omega_i, \quad (3)$$

Algorithm 2: G-Buffer pass.(Vertex and fragment code)

```
1  Program VertexShader:
2      vec4 gl_Position;              // In-built variable
3      in vec3 p, n;                  // Scene Point, Normal
4      in vec2 uv;                    // UV co-ordinates
5      out vec3 vertex;               // Interpolated in Frag.
6      out vec3 normal;               // Interpolated in Frag.
7      void main():
8          gl_Position = vec4(uv.x, uv.y, 0.0, 1.0);
9          vertex = p; normal = n;
10 Program FragmentShader:
11     in vec3 vertex;
12     in vec3 normal;
13     out vec4 gPos;                 // G-Buffer
14     out vec4 gNorm;                // G-Buffer
15     void main():
16         gPos = vec4(vertex,1.0);
17         gNorm = vec4(normal, 1.0); /* w → alpha
               channel                */
```

where B^{pq} is the radiance from a secondary hit-point q towards p and B_i^p is the inter-reflected radiance [Sloan et al. 2002]. 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 $B^{p_1 q_1}$ can easily be precomputed given a zero-bounce transfer texture T_o from Alg. 1 (See Fig. 2). The radiance from q_1 towards p_1

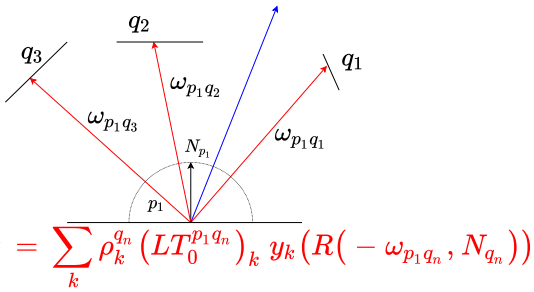

Fig. 2. **Inter-reflections:** The radiance $B^{p_1 q_n}$ (red lines) towards a point p_1 from a secondary hit-point q_n can be computed by first fetching transfer at q_n using the zero-bounce transfer texture T_o, applying the light L followed by convolution with the BRDF at q_n and evaluation at reflected direction along the normal at q_n. $B^{p_1 q_n}$ forms an indirect environment map which is projected to SH and stored at p_1 in an additional texture.

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 *one-bounce inter-reflection* texture T_1. At run-time, the inter-reflected

radiance is obtained by convolving B_i^{pq} fetched from T_1 with the BRDF SH ρ_i^p and evaluating at the reflection direction. The final color is given as: $B^p(\omega_o) + B_i^p(\omega_o)$. Alg. 1 can be easily extended to compute the second bounce texture T_2 and so on. The number of textures required is linear in the number of bounces in this setting, and the final color is just their summation.

4.3 Handling dense UV-packing

In the previous section we described methods for efficient computation of transfer textures. Usage of these textures requires UV co-ordinates each vertex to be defined. To obtain UV unwrapping of scene geometry, we used *Smart UV-Unwrap* or *Light Map Pack* from Blender 3D [Blender 2021]. One caveat with UV unwrapping is that dense packing of UV islands may cause overlaps which manifest as rendering artefacts. *Smart UV-Unwrap* does not guarantee non-overlapping islands while *Light Map Pack* leads to texture wastage and tiny pixel coverage for some parts of the geometry. In such scenarios, texture-sets are beneficial. Consider an example scene as

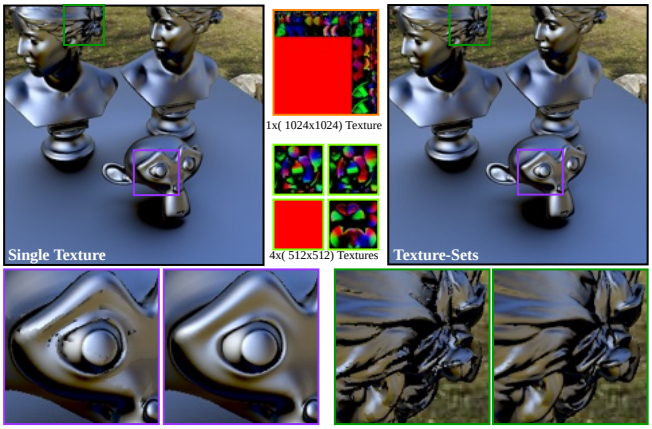

Fig. 3. **Texture sets:** TRM:Two Roza, one Monkey, We demonstrate the use of texture-sets with TRM(right) where 4 small textures are assigned to each piece of geometry against the use of single texture in case of TRM(left). The artefacts can be seen clearly in the Monkey's eyes and Roza's hair as depicted in insets. Note that in both cases, the memory requirements are the same (single 1024×1024 texture v/s four 512×512 textures).

shown in Fig. 3. This scene contains 441K triangles, all of which are packed into a single 1024×1024 texture (Fig. 3, left). As shown in the insets, this leads to artefacts. A better approach is to use texture-sets, which means assigning individual textures to each object in the scene (Fig. 3, right). In this case, each UV island can occupy the entire space of the texture thus eliminating artefacts.

5 IMPLEMENTATION DETAILS

We implement Alg. 2 in Python using the ModernGL [Dombi 2020] framework. We generate and store the resulting G-buffers for each scene in a pre-process step. Alg. 1 is implemented in Python and uses Embree [Wald et al. 2014] for efficient ray intersection tests. We project to band $l = 5$ (25 coefficients) real spherical harmonics. As mentioned in Sect. 4.1 dilation is required to ensure that all points in the scene receive a transfer value. Experimentally, we found a dilation of three to be sufficient which may need adjustment depending on the scene complexity. The time taken for generating transfer textures for a scene like in Fig. 1 is approximately three hours.

Our real-time renderer is also implemented in the ModernGL framework. We implement the triple product (TP) and triple product with fixed light (TPFL) methods augmented with our transfer textures. Rendering is done in the fragment shader using the generated transfer textures for the respective scene. We render all scenes with glossy materials with spatially varying roughness on a workstation with an NVIDIA RTX 3090 with a resolution of 1920×1080. An important detail is that we use the early depth pass to prune fragments that are not visible thus avoiding unnecessary computations. We use a texture resolution of 1024×1024 texture as we have found it to be best trade-off in between memory and quality for our scenes.

Table 1. Scene configurations. We list all scenes used in this paper, with their corresponding number of triangles and FPS with triple product (TP) and triple product fixed light (TPFL) methods on both vertex and fragment shaders (with transfer textures). Our approach achieves real-time framerates on all scenes.

Scene	**# tris.**	**Vert. (Trad.)**		**Frag. (Ours)**	
		TP	TPFL	TP	TPFL
Dragon (Fig 4)	1.3M	3.62	41.2	5.2	151.2
TRM (Fig 4)	441K	10.2	116.2	15.2	202.9
Room (Fig 1)	21K	352.3	2432.7	83.6	568.2
Plants (Fig 4)	18K	363.2	2597.6	6.7	168.3

6 RESULTS & EVALUATION

In this section, we present glossy rendering results including inter-reflections using transfer textures on the fragment shader. We compare the renderings with traditional vertex shader based approaches. We also discuss and demonstrate the use of normal maps with transfer textures which is not possible with traditional vertex based PRT. Finally, we analyze the memory requirements and give a lower bound of FPS for tranfer texture usage in a fragment shader. Rendering results are demonstrated on four scenes whose statistics and performance comparisons are given in Table 1.

6.1 Glossy rendering & Inter-reflections

Fig. 4 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 function. 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. We note that the TP/TPFL methods on vertex shader approach the render quality of our transfer textures with a highly tessellated ground plane, as shown in the *high-tessellation* renderings. We note that

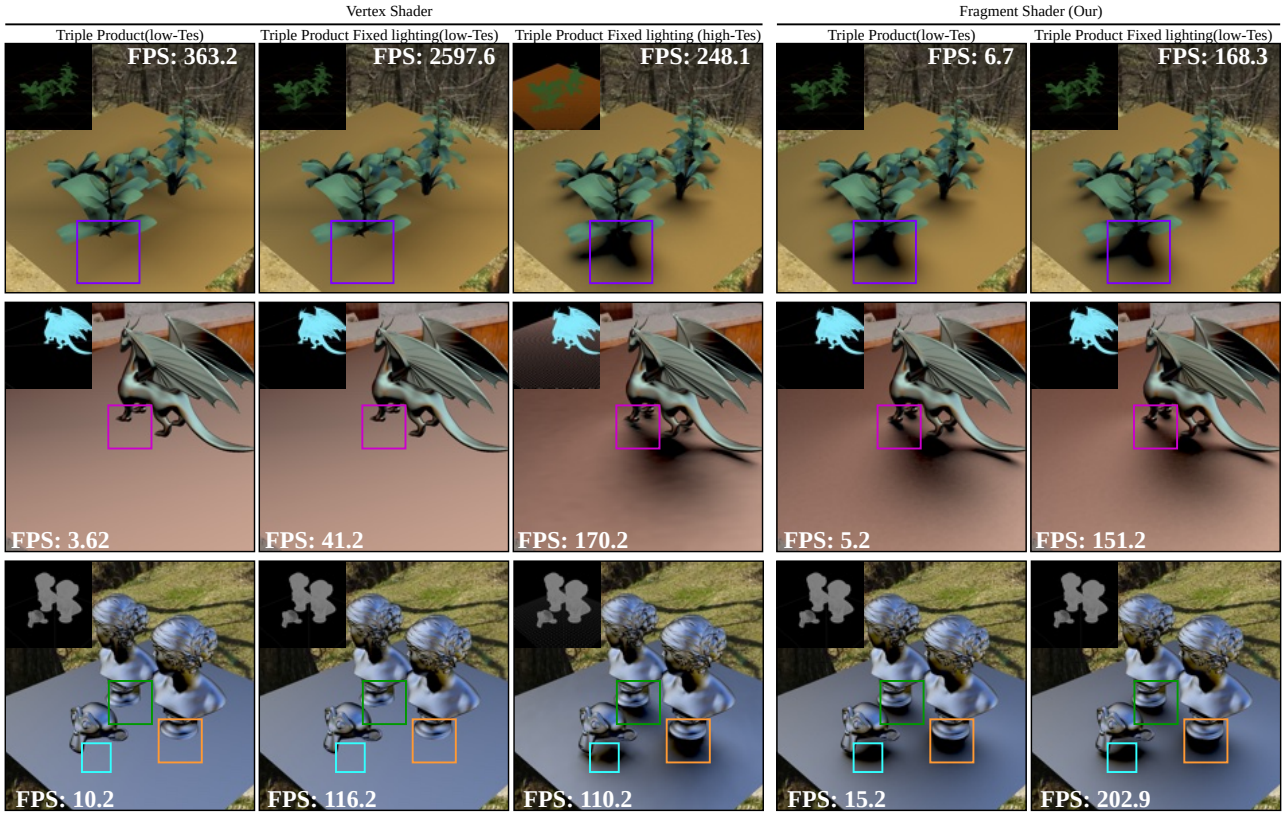

Fig. 4. We show results of TP and TPFL on the fragment shader using our transfer textures (Bottom). We compare renderings with traditional vertex shader based approaches on the top. For minimal tessellations, our method accurately renders shadows whereas previous methods are unable due to insufficient sampling of transfer. The third row (high-tessellation) shows that renderings using traditional methods approximately approach the quality of transfer textures on addition of more vertices. Note that low-FPS in case of Dragon low-tessellation and TRM low-tessellation in vertex based TP and TPFL is due to their high resolution geometry.

this requires the addition of *redundant* vertices. We further note that such situations frequently arise in production, for example with walls in a room or any large surface with minimal curvature (Fig 1). In such cases, all previous PRT methods on vertex shaders require the addition of avoidable vertices to store the transfer on leading to drop in performance, as opposed to our transfer textures method. Additional renders with different phong exponents and environments maps for four different scenes are shown in Fig. 7 & 8. Next, we demonstrate inter-reflections using transfer textures with the method described in Sec 4.2. The zero-bounce and one-bounce renders with their corresponding FPS are shown in Fig. 5 for two scenes: *Monkey* and *Roza*. Because of extra texture fetch, convolution and evaluation operations the FPS with inter-reflections is slightly lower, albeit still real-time. As described in Sect. 4.2, additional bounces can be added with additional pre-computed textures.

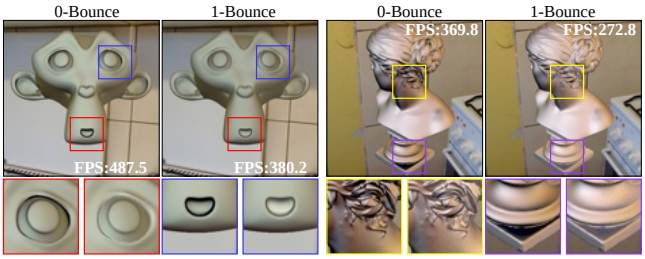

Fig. 5. We demonstrate inter-reflections with transfer textures on two scenes: Diffuse Monkey (left) and glossy Roza (right). Renders with inter-reflection maintain real-time frame-rates, albeit slightly lesser than zero-bounce renderings.

6.2 Normal Maps

Transfer textures make it possible to use normals maps during pre-computation. This translates to lesser vertices during rendering as finer detail can instead be embedded in the normal map. Consider precomputation in traditional vertex based PRT. In this case if a normal map is applied, it only ever affects the transfer at those vertices thus loosing detail within each face when using high frequency normal-texture. With transfer textures we can output the *shading normal* from the normal map instead of the *geometric normal* in the G-buffer during precomputation. In Alg. 1 line 6, the transfer will

Fig. 6. **Normal Maps:** Transfer textures make it possible to use the shading normals from normal maps instead of the geometric normals. This is difficult to achieve with traditinal vertex based PRT. The above scene shows a minimally tessellated ground plane (wireframe, right) with a normal map. The scene is rendered with TPFL and transfer textures. All normal map details are preserved.

then be computed at the shading normal instead. Since this texture is used to fetch transfer during rendering, all normal map details are preserved. We show renderings with normal maps in Fig. 6. The detail on the floor is due to the normal map without any additional vertices, as can be seen in the wireframe insets.

6.3 Memory Requirements

McKenzie demonstrated textures with diffuse PRT using Sloan's formulation. Consider directly implementing Sloan's glossy formulation with textures instead. This amounts to storing a $k \times k$ matrix per texel which quickly becomes intractable, even for reasonably small textures. Thus augmenting the triple product formulation to transfer textures is a clear choice. The memory requirements for vertex as well as texture (fragment) based approaches is shown in Table 2. The former's memory requirements depend on the scene complexity whereas it is constant for textures. Furthermore, a direct extension of Sloan's method to textures is infeasible, as shown in the fifth column (2.5 GB per texture).

Table 2. Memory requirements for vertex based and transfer texture based approaches for a 1024×1024 texture. Note that McKenzie uses Sloan's approach which results in large textures for glossy rendering.

Scene	**# tris.**	**Vert. Mem.**		**Tex. Mem.**	
		Sloan	[Ng et al.]	McKenzie	Ours
Room	21K	64MB	2.5MB	2.5GB	100MB
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

6.4 Lower Bound on FPS

Since transfer textures are used in fragment shaders with an early depth pass, we achieve a lower bound on the FPS. The computation is roughly the same for each fragment and the worst case is when all fragments contain some geometry to be processed and rendered. This is in contrast to vertex based approaches, where run-time depends on the number of vertices in the scene. We demonstrate this in Fig. 4 in the *TRM* (441K verts) and *Dragon* (1.3M verts) scenes. Here, the FPS is lower for vertex based approach as compared to fragment based approach with transfer texture, in both TP and TPFL.

7 CONCLUSIONS, LIMITATIONS & FUTURE WORK

In this paper, we presented *precomputed radiance transfer textures* for decoupling mesh tessellation from transfer sampling and storage for glossy rendering. We described methods to efficiently and correctly compute these textures and also demonstrated incorporation of inter-reflections using additional precomputed textures. We compared our renderings with traditional vertex based PRT approaches and thoroughly analyzed the memory requirements of transfer textures. We demonstrated real-time framerates for rendering with transfer textures on the fragment shader and superior render quality for minimally tessellated meshes. Additionally, we gave a lower bound on the FPS which will be useful in performance analysis in production. Our approach inherits the advantages of texture based optimizations like textures-sets, mip-maps and level of detail which can be easily incorporated. Although we demonstrate on a fixed texture resolution, it can be tailored accordingly depending on the hardware constraints and rendering quality needed. This is in contrast to vertex based methods that provide vertex count as the only control knob and little control over level of detail.

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 [Sloan et al. 2002] 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.

REFERENCES

Laurent Belcour, Guofu Xie, Christophe Hery, Mark Meyer, Wojciech Jarosz, and Derek Nowrouzezahrai. 2018. Integrating Clipped Spherical Harmonics Expansions. *ACM Trans. Graph.* 37, 2, Article 19 (mar 2018), 12 pages. https://doi.org/10.1145/3015459

Blender. 2021. *Blender - a 3D modelling and rendering package.* Blender Foundation, Blender Institute, Amsterdam. http://www.blender.org

Szabolcs Dombi. 2020. ModernGL, high performance python bindings for OpenGL 3.3+. https://github.com/moderngl/moderngl.

Michal Iwanicki and Peter-Pike Sloan. 2009. Normal Mapping with Low-Frequency Precomputed Visibility. In *SIGGRAPH 2009: Talks (SIGGRAPH '09).* ACM, New York, NY, USA, Article 52, 1 pages. https://doi.org/10.1145/1597990.1598042

Jaakko Lehtinen and Jan Kautz. 2003. Matrix Radiance Transfer. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics (I3D '03).* Association for Computing Machinery, New York, NY, USA, 59–64. https://doi.org/10.1145/641480.641495

Harrison Lee McKenzie Chapter. 2010. Textured Hierarchical Precomputed Radiance Transfer. (2010).

Microsoft. 2003. *Microsoft DirectX 9 Programmable Graphics Pipeline.* Microsoft Press, USA.

Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. 2004. Triple Product Wavelet Integrals for All-Frequency Relighting. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 477–487. https://doi.org/10.1145/1015706.1015749

Derek Nowrouzezahrai, Patricio Simari, and Eugene Fiume. 2012. Sparse Zonal Harmonic Factorization for Efficient SH Rotation. *ACM Trans. Graph.* 31, 3, Article 23 (June 2012), 9 pages. https://doi.org/10.1145/2167076.2167081

Jacopo Pantaleoni, Luca Fascione, Martin Hill, and Timo Aila. 2010. PantaRay: Fast Ray-Traced Occlusion Caching of Massive Scenes. *ACM Trans. Graph.* 29, 4, Article 37 (July 2010), 10 pages. https://doi.org/10.1145/1778765.1778774

Ravi Ramamoorthi. 2009. *Precomputation-Based Rendering.* NOW Publishers Inc.

Peter-Pike Sloan. 2008. Stupid spherical harmonics (sh) tricks. In *Game developers conference*, Vol. 9. 42.

Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *ACM Trans. Graph.* 21, 3 (July 2002), 527–536.

Peter-Pike J. Sloan, J. Hall, J. Hart, and John M. Snyder. 2003. Clustered principal components for precomputed radiance transfer. *ACM SIGGRAPH 2003 Papers* (2003).

John Snyder. 2006. *Code Generation and Factoring for Fast Evaluation of Low-order Spherical Harmonic Products and Squares.* Technical Report MSR-TR-2006-53. 9 pages.

Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. 2014. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *ACM Trans. Graph.* 33,

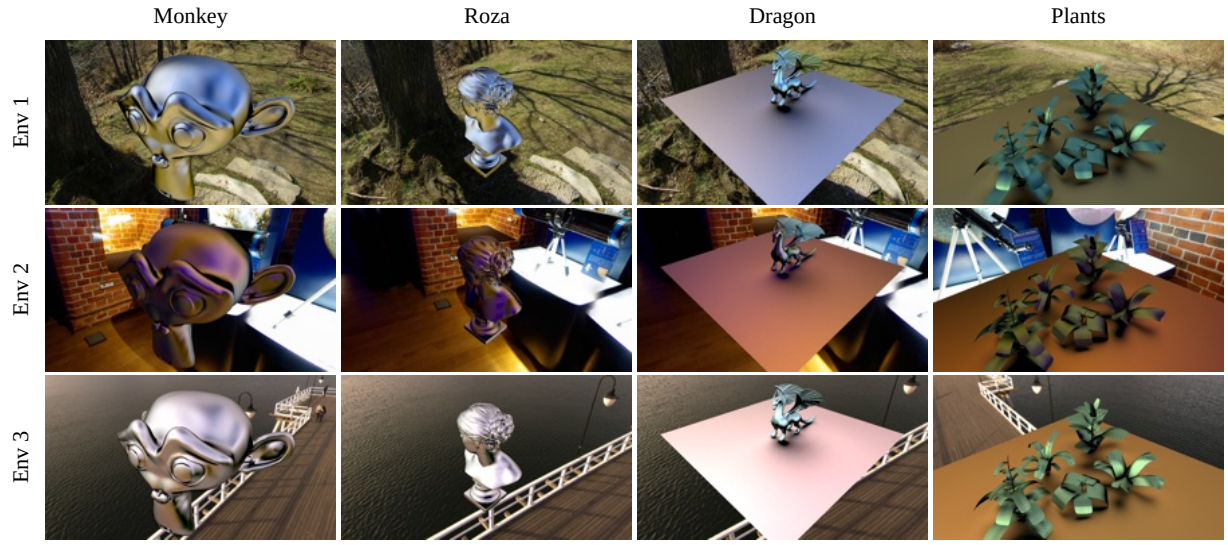

Fig. 7. Results of our transfer textures method on different light settings

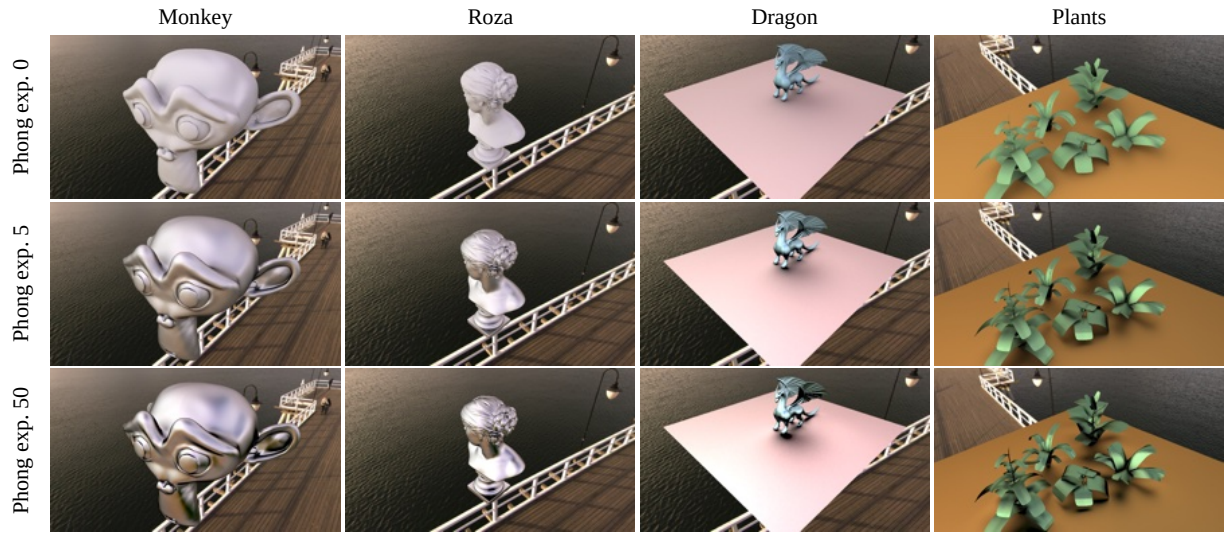

Fig. 8. Results of our transfer textures method with different Phong BRDFs with same light

4, Article 143 (July 2014), 8 pages. https://doi.org/10.1145/2601097.2601199

Jingwen Wang and Ravi Ramamoorthi. 2018. Analytic Spherical Harmonic Coefficients for Polygonal Area Lights. *ACM Trans. Graph.* 37, 4, Article 54 (July 2018), 11 pages. https://doi.org/10.1145/3197517.3201291

Lifan Wu, Guangyan Cai, Shuang Zhao, and Ravi Ramamoorthi. 2020. Analytic Spherical Harmonic Gradients for Real-Time Rendering with Many Polygonal Area Lights. *ACM Trans. Graph.* 39, 4, Article 134 (July 2020), 14 pages. https://doi.org/10.1145/3386569.3392373

Hanggao Xin, Zhiqian Zhou, Di An, Ling-Qi Yan, Kun Xu, Shi-Min Hu, and Shing-Tung Yau. 2021. Fast and Accurate Spherical Harmonics Products. *ACM Trans. Graph.* 40, 6, Article 280 (dec 2021), 14 pages. https://doi.org/10.1145/3478513.3480563

Kun Zhou, Yaohua Hu, Stephen Lin, Baining Guo, and Heung-Yeung Shum. 2005. Precomputed Shadow Fields for Dynamic Scenes. *ACM Trans. Graph.* 24, 3 (July 2005), 1196–1201. https://doi.org/10.1145/1073204.1073332