Deferred Neighboring Radiance Transfer

Salvatore Giuffrida
Utrecht University
ICA - 4300300

Figure 1. A classic Cornell Box scene showing the effect of DNRT

Abstract

This paper introduces Deferred Neighboring Radiance Transfer, a new precomputation-based Global Illumination technique. DNRT modifies pre-existing light solutions for static environments enhancing them with reflections and occlusions from dynamic objects. DNRT is particularly well suited for integration with lightprobing techniques as it can exploit light probes data to achieve faster performance. Prior to execution, DNRT computes a set of radiance transfer coefficients. At run-time, working in screen-space, these coefficients are combined with data from the light probes and the G-buffer to generate a new radiance solution. This approach allows DNRT to maintain a low computational cost, with relatively high quality results. DNRT's computational cost grows linearly with the number of dynamic objects in the environment. For scenarios with a single dynamic object, DNRT is 10 times more efficient than current videogame industry state of practice methods like SSAO and HBAO. On high-end GPUs DNRT's time cost is below 0.1 ms per dynamic object.

1 Introduction

When trying to build a convincing virtual environment, employing a good illumination model is crucial, as a scene that is badly lit can look dull and lifeless. The digital worlds presented to us by modern videogames, although astonishingly detailed, still show in many instances signs of struggle with this particular aspect of their composition. Many techniques have been developed in recent years to tackle this problem, with varying levels of success, both in terms of performance and quality. Among these techniques, is here presented *Deferred Neighboring Radiance Transfer*, a new method developed to enrich existing lighting technologies adding to their output one bounce of indirect light from dynamic objects onto neighboring geometry.

1.1 Local and Global Illumination models

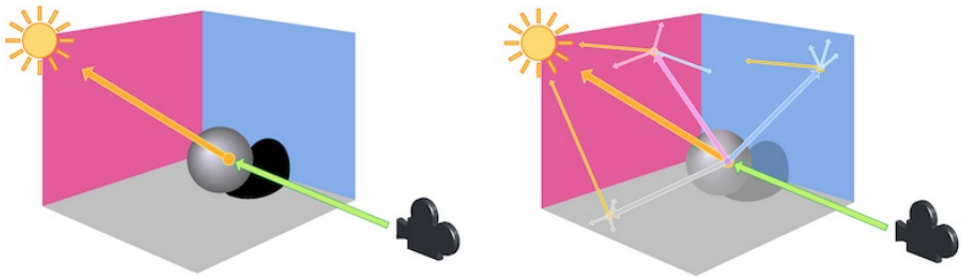


Figure 2. Local Illumination (left) vs Global Illumination (right) [Sidefx 2016].

In computer graphics the problem of correctly simulating light propagation within

an environment is often deconstructed by distinguishing between direct and indirect illumination. The first refers to a source directly shining on a target, while the latter incorporates the phenomenon of a source reaching a target through multiple bounces within the scene.

Local models only simulate direct light. This enables them to be very efficient, as the only complex operation that they have to solve is visibility between the source and the target, for which shadowmapping [Williams 1978] has long been a reliable solution.

Global Illumination models, differently from their local counterpart, try to correctly simulate both direct and indirect light. This yields far more realistic results, which come, however, at a high cost in terms of performance. The computational gap between the two models, global and local, is due to the higher dimensionality of the problem the former is trying to solve. As illustrated in Figure 2, while for each shading point, local models need to consider only the directions that stretch towards the light sources, global models have to account for all possible directions in the upper hemisphere of the interested point. The situation is worsened by the fact that for these additional directions a light value is not readily available and has to be computed through a recursive process. This, in turn, leads to a far higher computational complexity, thus the lower performance.

1.2 Videogames and Illumination Models

Gaming applications have always presented peculiar constraints that make the development of a convincing illumination model particularly difficult. For an optimal user experience, videogames have to run at a framerate that maintains a consistent level of performance over 30 FPS. This is fundamental as it conveys a sense of fluidity and helps minimizing input latency. Furthermore, the resources available for rendering are limited, as they have to be shared with other modules, like physics, AI and networking. Given these constraints, the high level of performance offered by local models has made them for many decades the de facto method of choice for videogames. However, this has led to an overall underwhelming visual fidelity.

On the other end of the spectrum, classic Global Illumination methods, like raytracing [Whitted 1979], photon mapping [Jensen 1996] and radiosity [Goral et al. 1984], still present prohibitive performance for gaming applications. Their unsuitability derives from the fact that these methods were initially designed to produce results that would look as photo-realistic as possible, pushing on image fidelity over performance.

To bridge the gap between the resources currently available in consumer machines and the computational complexity of the aforementioned methods, researchers have developed different approximated Global Illumination models that try to offer a good

compromise between performance and quality. Numerous are the assumption and simplifications behind these techniques, like the discretization of the environment in voxels [Crassin et al. 2011; Kaplanyan and Dachsbacher 2010], the light sources reduction to Virtual Point Light [Keller 1997; Dachsbacher and Stamminger 2005], or the (partial) precomputation of the lighting calculations [Sloan et al. 2002]. In the crowded scenario of approximated Global Illumination Techniques finds now its place DNRT, a new entry in the category of precomputation based methods.

1.3 Paper Overview

The remaining part of this paper will start with an analysis of related work in Section 2. Various Global Illumination techniques will be classified in different groups; their strengths and weaknesses will be highlighted. Section 3 will provide an overall view on DNRT. This will then be expanded upon in more detail in Section 4 and 5, which will, respectively, present the mathematical basis on which DNRT is built and describe its current implementation. Section 6 will introduce the method and scenarios used to test DNRT. Results will then be analyzed and discussed in Section 7. Final remarks on DNRT and an outlook on possible future work will be provided in Section 9.

2 Related Work

This Section gives a brief overview of the approximated Global Illumination techniques that are most relevant to DNRT. These techniques have been organized in categories, each representing a different approach to real-time Global Illumination.

2.1 Precomputed Radiance Transfer Methods

Precomputed Radiance Transfer techniques aim to provide a time efficient Global Illumination solution by transferring part of the computational load to an offline process. In other words, PRT techniques split GI-related calculations between a precomputation phase and a run-time phase.

PRT techniques work with non-deformable objects that maintain their material properties (albedo and glossiness) through time. With these assumptions, during the precomputation phase, it is possible to calculate a radiance transfer operator for each shading point in the scene. In most PRT applications, the objects' vertices are selected as shading points [Slomp et al. 2006]. The radiance transfer operator is then used at run-time to transform incoming radiance into outgoing radiance.

Initially, PRT techniques focused on shading each object of the scene separately, in isolation from all others, managing to simulate only phenomena like self-reflections

and self-shadowing [Sloan et al. 2002; Lehtinen and Kautz 2003].

To support interaction among objects, Neighboring Radiance Transfer [Sloan et al. 2002] has been developed. This technique works adequately when objects influence each other in couples, but not does merge well the effect of more than two objects acting on a single point. NRT's limitations have been partially overcome by more modern techniques employing a Radiance Transfer Field approach [Pan et al. 2007; Iwasaki et al. 2007]. However, these techniques still present evident limitations in terms of performance.

2.2 Screen Space Methods

Screen Space Global Illumination methods use information available from different rendering buffers (frame-buffer, depth-buffer and normal-buffer) to approximate the 3D composition of a scene and enrich it with Global Illumination lighting. As a consequence, the level of performance offered is dependent on the screen resolution, rather than on the geometric complexity of the scene.

Screen Space Ambient Occlusion [Shanmugam and Arikan 2007; Mittring 2007] modifies a rendered image adding to it close proximity shadows. Each pixel's neighborhood is examined with samples from a spherical volume to detect the presence of occluders. The result is an occlusion factor used to darken the pixel's color. The use of a spherical volume for sampling causes SSAO to present visual artifacts, such as edges being excessively bright or dark based on their convexity. The problem is aggravated by the fact that the sphere's radius is calculated in screen-space and is constant for every pixel.

Horizon Based Ambient Occlusion [Bavoil et al. 2008] improves on SSAO by using a hemispherical sampling volume. The hemisphere is centered around the pixel's normal and has a radius dependent on the pixel's view position (bigger the distance from the view origin, smaller the hemisphere). This helps reducing the visual artifacts that characterize SSAO.

Screen Space Directional Occlusion [Ritschel et al. 2009], differently from SSAO and HBAO, takes into consideration the lighting direction to generate more realistic shadows. SSDO is also capable of simulating one bounce of indirect light exploiting the information gathered from the occlusion samples. In its basic form, SSDO generates occlusions and indirect light only for objects appearing in the framebuffer. To overcome this limitation SSDO uses renderings from multiple camera angles and depth peeling [Everitt 2001]. This, however, has a strong negative impact on performance.

2.3 Dynamic Real-time Methods

Global Illumination lighting can be dynamically recalculated at each frame with real-time performance when using a discretized representation of the environment. This approach is adopted by methods like Light Propagation Volumes [Kaplanyan and Dachsbacher 2010] and Voxel Cone Tracing [Crassin et al. 2011].

The Light Propagation Volumes method partitions the environment in a series of cascated 3D grids. At each frame, radiance is injected into the grid cells that contain a light source and then propagated to the surrounding cells of the grid. The light contained in each cell is represented using a Spherical Harmonics approximation [Green 2003]. LPV can offer real-time performance, but suffers from visual artifacts like light leaking.

The Voxel Cone Tracing technique discretizes the scene using a sparse voxel octree data structure that, for each point to shade, is queried using a limited number of cones. This process enables VCT to offer accurate results, especially when confronted with LPV. However, this comes at high cost in terms of performance and memory requirements.

2.4 Radiosity-based Methods

Radiosity based techniques produce view-independent solutions. This characteristic is very useful in the context of gaming applications, as these often feature cameras characterized by continuous and unpredictable movements. The classic radiosity method [Goral et al. 1984], however, yields non-interactive framerates when applied to complex scenarios like the ones presented in modern games. This is due to the recursiveness of its formulation and to the cost of visibility and form-factor calculations.

In [Cohen and Greenberg 1985] an iterative gathering approach is used instead of recursion, but performance is still capped by its low convergence rate. Progressive Refinement [Cohen et al. 1988] ensures a faster convergence rate by replacing the radiosity gathering approach with a shooting approach.

Incremental Radiosity [Chen 1990] computes the radiosity solution of a given frame as a modification of the solution from the previous frame. This is done using Progressive Refinement to shoot *redistribution* radiosity, which serves to account for changes in the scene's geometry in-between frames. Cross-Redistribution [Van De Hoef 2013] improves the redistribution process' performance by making it solely dependent on moving patches.

Anti-radiance [Dachsbacher et al. 2007] eliminates visibility computations by shoot-

ing negative radiance from the occluding patches' back-face. The radiance direction has to be preserved, hence directional bins are used. This produces blurrier shadows.

While the above mentioned techniques vastly improve on the performance of the original radiosity method, they are still not efficient enough to be used in gaming applications. This is caused, among other reasons, by the high cost of calculating the form-factors among patches. The middleware Enlighten [Geomerics 2016] addresses this problem calculating the form-factors in a precomputation phase. This grants an important boost in performance, that allows Enlighten to reach real-time framerates. However, precomputation can only be used for static patches. Enlighten employs Irradiance volumes [Greger et al. 1998], or analogous light probing techniques, to implement radiance transfer from static patches onto dynamic meshes. It does not currently simulate, however, the converse phenomenon, i.e. the influence of moving meshes onto static geometry (and onto other moving meshes).

3 Deferred Neighboring Radiance Transfer

This section gives a high level overview of DNRT. Its goals and application domain are defined. Its connections to the techniques examined in Section 2 are underlined. The intuitive explanation of the method provided here will be reinforced with a solid theoretical derivation in Section 4.

3.1 Radiance Redistribution

Light within a scene propagates based on the geometric relationships between objects. When a dynamic object changes its position it affects these geometric relationships raising the necessity for a new lighting solution. In many Global Illumination techniques [Crassin et al. 2011; Kaplanyan and Dachsbacher 2010], this problem is tackled by recomputing all light interactions at each frame. This approach can lead to the execution of unnecessary calculations since the geometric relationship between static objects remains unchanged and the only modified light interactions are the ones passing through dynamic objects [Van De Hoef 2013].

The Cross-redistribution [Van De Hoef 2013] method defines a redistribution process that updates at each frame the current light solution (rather than recalculating it from the beginning) solely depending on positional changes of the dynamic objects. This approach allows Cross-redistribution to offer better performance than previous radiosity-based techniques. Nonetheless, it is not capable of maintaining a real-time framerate for complex environments like the ones featured in modern high-profile videogames.

3.2 Real-time Radiance Redistribution

DNRT combines the observations at the base of Cross-redistribution with a PRT-based approach with the goal of achieving real-time performance in high-poly environments. DNRT limits its scope to only one bounce of indirect light and relies on a set of approximations presented in the rest of this paper.

As base of its redistribution process, at each frame, DNRT takes a pre-existing radiance solution calculated for a static environment and modifies it by adding reflections and occlusions from dynamic objects. In the current DNRT implementation, the pre-existing radiance solution is computed using Enlighten [Geomerics 2016]. This technology subdivides the environment in large static patches and precomputes the form-factor between them. At run time, this allows for fast radiance redistribution among static geometry.

To model reflections and occlusions from dynamic objects, knowing the incoming radiance at their respective positions is necessary. Computing this radiance at run-time is, however, computationally expensive. To solve this problem with higher efficiency, DNRT relies on light probing technologies inspired by Irradiance Volumes [Greger et al. 1998]. Incoming radiance is transformed in reflected radiance and subtractive radiance using a precomputed radiance transfer operator, similarly to Neighboring Radiance Transfer techniques (see next Subsection).

3.3 DNRT Framework

Analogously to previous NRT methods, DNRT is built on a theoretical framework that categorises the environment in (see Figure 3):

- O, the dynamic object whose effect on the scene's geometry needs to be simulated
- S, the set of infinitesimal area elements s_i that make up O's surface.
- P, a dense set of samples p_i surrounding O.
- R, the set of infinitesimal area elements r_i that make up the surface of any object in the scene that is affected by O.

DNRT works in two phases. First is the precomputation phase, an offline process in which a radiance transfer operator is associated to each sample p_i. This linear operator will be later used at run-time to transfer incoming radiance at O's position into outgoing radiance from O toward all the elements in R. During this process the

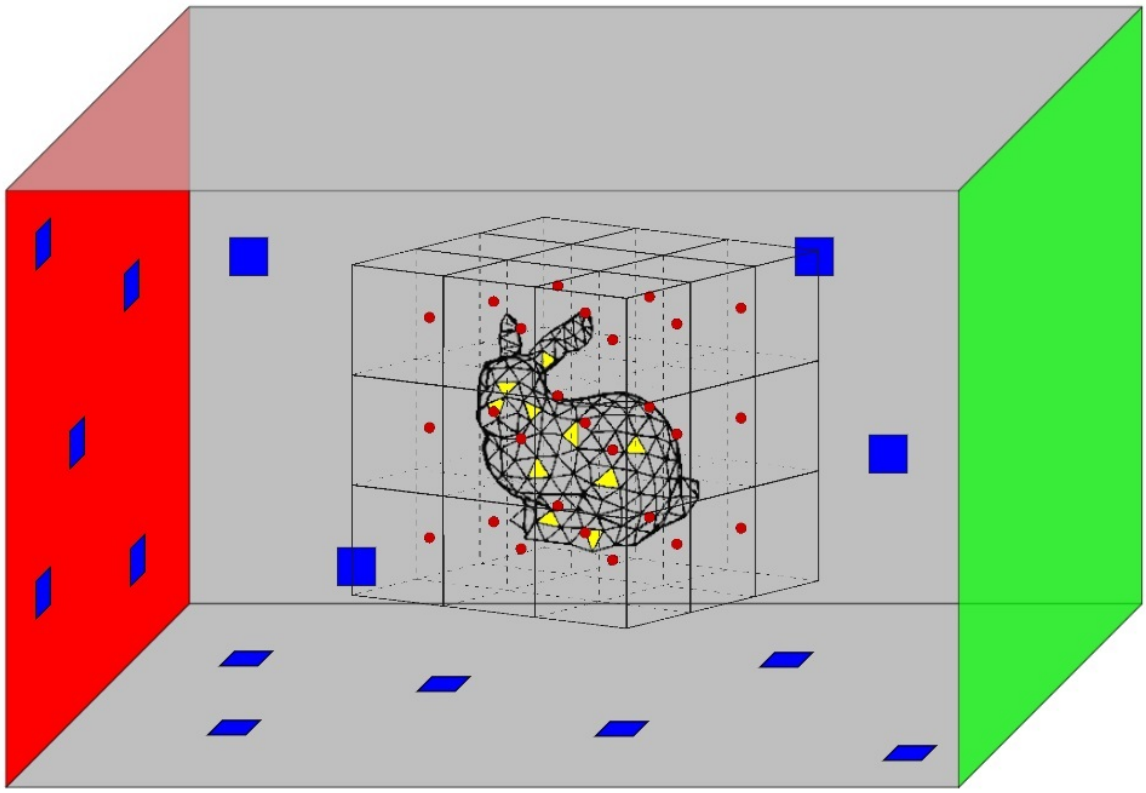


Figure 3. s_i in yellow; r_i in blue; p_i in red

samples p_i work as aliases for the actual area elements r_i, since these are not known during the precompute phase (for further details see Subsection 4.1.3).

At run-time, given the position of a certain element r_i, the closest sampling point p_i is retrieved from O's neighborhood and its transfer operator is selected. The transfer operator is applied to the incoming radiance in O. The resulting value represents one bounce of indirect light and is used to modify r_i's final color. Note that, differently from previous PRT methods, in DNRT, when applying the transfer operator, a negative value may be obtained. This way occlusions are simulated.

3.4 Contributions

While sharing its framework and two phases approach with already existing PRT techniques [Sloan et al. 2002; Pan et al. 2007], DNRT presents a substantial set of innovations that set it apart from previous methods. More specifically, DNRT provides:

1. A new radiance transfer operator that replaces previous matrix multiplications [Sloan et al. 2002] with a simple dot product.

2. An anti-radiance approach for occlusions that enables the method to effectively work as a post-process effect on an already existing solution.
3. An integration with light probing technologies which now benefit from an additional bounce of indirect light from dynamic objects.
4. A screen space solution powered by a deferred-rendering based, full GPU implementation.

Notice that, while in previous methods [Sloan et al. 2002] the incoming radiance function for the dynamic object O was obtained at run-time by rendering a cubemap at its position, DNRT's integration with light probing technologies enables the method to forgo this process and fetch the radiance directly from the probes, thus raising the potential for better performance (see Section 5).

Finally, it is important to highlight the value of DNRT's integration within the Unity Engine, which is presented in detail in Section 5. This particular aspect shows that the method is suitable and ready to be used in actual videogame production.

4 From the Rendering Equation to DNRT

As introduced in Subsection 3.4, DNRT modifies a preexisting Global Illumination solution, generated for a static environment, adding the influence of dynamic objects. The initial static GI solution will be referenced in the rest of this paper as L_{init}.

When a dynamic object O is inserted into the scene, L_{init} has to be modified in two ways:

- Radiance reflected by O from any surface element $r_j \in R$ toward a specific element r_i has to be added. This additional radiance will be referenced as L_{refl}.
- Radiance received by r_i from any surface element r_j prior to O's inclusion in the scene has to be subtracted if r_j is no longer directly visible from r_i's position. This subtractive radiance will be referenced as L_{occl}.

Notice that, as previously stated, DNRT limits its scope to only one bounce of indirect light. When applying the above listed modifications to L_{init}, the change operated should initiate a ripple effect that influences the solution for multiple bounces thereafter. These contingent effects are not currently simulated by DNRT.

The next two subsections give, respectively, a formal definition of L_{refl} and L_{occl}. From their initial definition, a reformulation that is prone to (partial) precalculation is derived. Subsection 4.3 puts everything together giving a final equation for DNRT.

4.1 Precomputing L_{refl}

In order to be able to express L_{refl} in a form that is fit for precomputation, it is first necessary to provide a strict mathematical definition for such function. This is done using an adapted formulation of the classic rendering equation [Kajiya 1986]:

$$L_{refl}(\omega_o, r_i) = \int_D L_{refl}(\omega_{s_i}, r_i)\rho_{r_i}(\omega_s, \omega_o)h(r_i, a_i)(\omega_s \cdot n_{r_i})_+ d\omega_{s_i} \tag{1}$$

The equation above models the radiance reflected by a dynamic object O towards an area element r_i. The symbol ρ_{r_i} represents the BRDF function in r_i, while n_{r_i} is the normal for that specific area element. D is the solid angle projected by O onto r_i and is used as the integration domain instead of the entire hemisphere Ω. This ensures that only radiance coming from O's surface elements s_i is captured. To highlight this, the incoming radiance direction is represented with the symbol ω_{s_i}, rather than the usual ω_i. Cases where an intruding object partially covers O's projection onto r_i's upper hemisphere, are handled using a visibility function h. This function returns 1 if the first area element a_i in direction $-\omega_{s_i}$ is part of O's surface S, 0 otherwise. The $(\omega_s \cdot n_{r_i})_+$ dot product is used to weight incoming radiance based on its angle of incidence. Negative values are clamped to 0, as indicated by the $_+$ subscript.

Notice that Equation 1 cannot be directly used to modify L_{init} at run-time: it is simply too computationally expensive due to the integral calculations involved. However, with the right set of assumptions and approximations, it can be reformulated in a way that gives ample opportunities for precomputation, making the final computation far more efficient.

Since DNRT considers only completely diffuse reflectors, the BRDF function ρ_{r_i} can be replaced with $\frac{\rho_{r_i}}{\pi}$, where ρ_{r_i} is the reflectivity factor (albedo) in r_i. Furthermore, given that the dynamic object O is also diffuse, the radiance $L_{refl}(\omega_{s_i}, r_i)$ coming onto r_i from any of the area elements s_i can be expressed as $\frac{I(s_i)\rho_{s_i}}{\pi}$, where I represents the irradiance function and ρ_{s_i} is the albedo at s_i. This leads to the following equation:

$$\begin{aligned} L_{refl}(r_i) &= \int_D \frac{I(s_i)\rho_{s_i}}{\pi} \cdot \frac{\rho_{r_i}}{\pi} h(r_i, a_i)(\omega_{s_i} \cdot n_{r_i})_+ d\omega_{s_i} \\ &= \rho_{r_i} \int_D \frac{I(s_i)\rho_{s_i}}{\pi^2} h(r_i, a_i)(\omega_{s_i} \cdot n_{r_i})_+ d\omega_{s_i} \end{aligned} \tag{2}$$

Notice that while ρ_{r_i} has been factored out of the integral operator, $\frac{1}{\pi^2}$ is still within its scope. This will allow to execute the division during precomputation, rather than at run-time, which slightly improves DNRT's efficiency.

By assuming that the irradiance does not vary around the object O, I can be expressed as a function of the normal at s_i, rather than as a function of s_i itself. Notice that this same assumption is used in most systems that employ irradiance maps [Ramamoorthi and Hanrahan 2001]. In mathematical terms, this means that $I(s_i)$ is now replaced with $I(n_{s_i})$:

$$L_{refl}(r_i) = \rho_{r_i} \int_D \frac{I(n_{s_i})\rho_{s_i}}{\pi^2} h(r_i, a_i)(\omega_{s_i} \cdot n_{r_i})_+ d\omega_{s_i} \tag{3}$$

While this equation is simpler than the one presented at the beginning of this section, it is still not precomputable: $I(n_{s_i})$, n_{r_i} and $h(r_i, a_i)$ are all unknown prior to runtime; r_i's relative position to O, which influences the domain D, is also not known.

4.1.1 *Removing the I dependency*

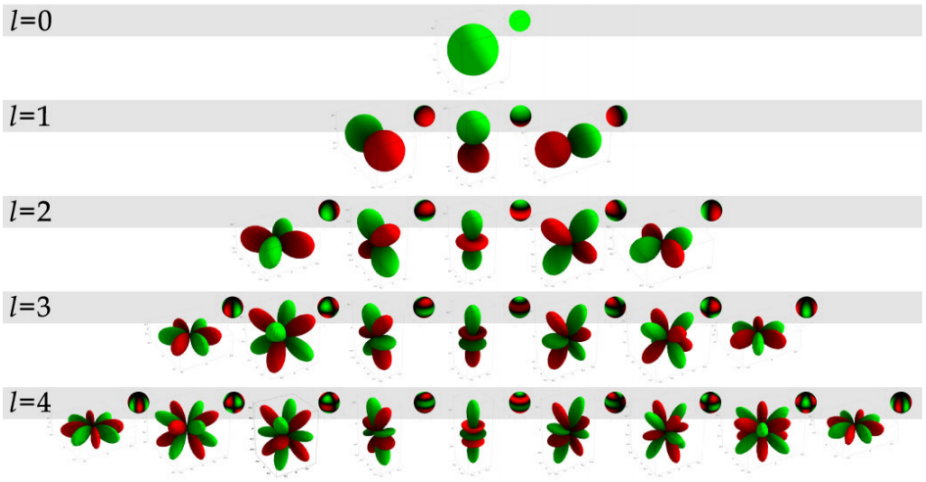


Figure 4. Spherical Harmonics basis functions are organized in bands, shown above up to the 5th level. The use of more bands gives a more accurate approximation [Green 2003].

The dependency on I during the precompute phase can be removed by substituting this irradiance function with its Spherical Harmonics [Green 2003] representation:

$$\begin{aligned} I(n_{s_i}) &= \sum_{j=1}^{n} c_i Y_j(n_{s_i}) \\ \implies L_{refl}(r_i) &= \rho_{r_i} \sum_{j=1}^{n} c_j \int_D \frac{Y_j(n_{s_i})\rho_{s_i}}{\pi^2} h(r_i, a_i)(\omega_{s_i} \cdot n_{r_i})_+ d\omega_{s_i} \end{aligned} \tag{4}$$

By using I's SH representation, Equation 4 has been split into a sum of n integrals,

where n is the number of basis functions Y_j used for the projection in the frequency domain. In most applications that make use of SH projection, DNRT included, this number is equal to 4 or 9, corresponding, respectively, to a 2-bands approximation and a 3-bands approximation (see Figure 4).

4.1.2 Removing the h dependency

Currently, the problem of handling the possible intrusion of objects between O and a certain receiving area r_i does not have a solution at the precompute stage. For this reason, h is simply disregarded during precomputation. At run-time, differently from [Sloan et al. 2002], DNRT offers an approximated solution to the intrusion problem based on omnidirectional shadowmapping. This, however, has a substantial negative impact on performance. A more in-depth description of this solution is presented in Subsection 5.2.3, while a breakdown of its performance is given in Subsection 7.1.

4.1.3 Removing the r_i dependency

To overcome the lack of knowledge in regards to r_i's position during the precomputation stage, a sample-based approach is used. As introduced in Subsection 3.3, each dynamic object O is surrounded by a dense set of samples p_i. A reflected radiance value is calculated and assigned to each sample. Mathematically, this means substituting r_i with p_i, as follows:

$$L_{refl}(p_i) = \rho_{r_i} \sum_{j=1}^{n} c_j \int_D \frac{Y_j(n_{s_i})\rho_{s_i}}{\pi^2} (\omega_{s_i} \cdot n_{r_i})_+ d\omega_{s_i} \tag{5}$$

Samples basically work as placeholders for real receivers during the precomputation phase. At run-time, a receiver r_i examines O's neighborhood and selects the samples p_i whose relative position to O is the closest to r_i's own relative position to O. The radiance transfer operators of the selected samples p_i are then interpolated. The resulting radiance transfer operator is finally used to calculate the reflected radiance in r_i.

4.1.4 Removing the n_{r_i} dependency

Most NRT techniques approach the problem of not knowing the receiver's normal at the precompute stage using convolution (see Appendix A) or by applying SH projection to the function $(\omega_{s_i} \cdot n_{r_i})_+$. DNRT, instead, during precomputation, replaces n_{r_i}, the normal at the receiving area element r_i, with n_o, an intermediate normal vector pointing towards the dynamic object O's centroid. This substitution leads to the

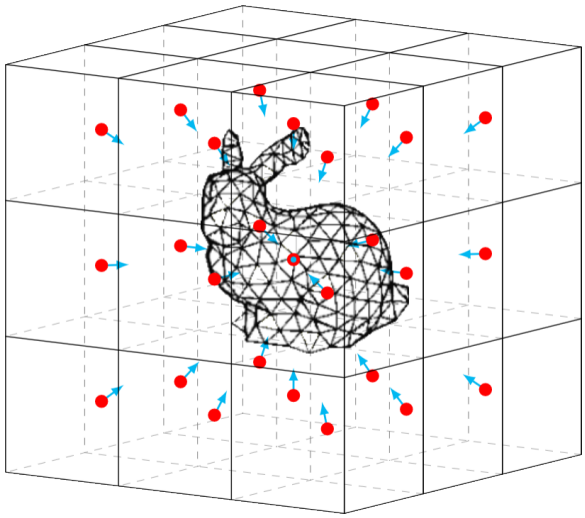


Figure 5. During precomputation each sampling point is directed toward the dynamic object's center following the normal direction n_o.

following equation:

$$L_{refl}(p_i) = \rho_{r_i} \sum_{j=1}^{n} c_j \int_D \frac{Y_j(n_{s_i})\rho_{s_i}}{\pi^2} (\omega_{s_i} \cdot n_o)_+ d\omega_{s_i} \tag{6}$$

To account for the actual orientation of the each receiving area element r_i, at run-time, $L_{refl}(\cdot)$ is scaled by the dot product of n_o and n_{r_i}:

$$L_{refl}(p_i) = \rho_{r_i}(n_o \cdot n_{r_i})_+ \sum_{j=1}^{n} c_j \int_D \frac{Y_j(n_{s_i})\rho_{s_i}}{\pi^2} (\omega_{s_i} \cdot n_o)_+ d\omega_{s_i} \tag{7}$$

Equation 7 has a simple, intuitive explanation. During the precomputation phase, DNRT reduces the dynamic object O to a point light source emitting (reflected) radiance from O's centroid o. This radiance is captured in its full intensity by orientating each sampling point in direction n_o. Classic lambertian lighting [Klett et al. 1760] is then used at run-time to scale the radiance intensity based on the receiving area element normal n_{r_i} and the light direction represented by n_o.

The use of the intermediate normal n_o constitutes an approximation in DNRT's lighting model. This approximation works particularly well for convex models with uniform albedo, while can bias the solution in the case of concave models with highly varying albedo, especially when positioned at short distances (see Figure 6).

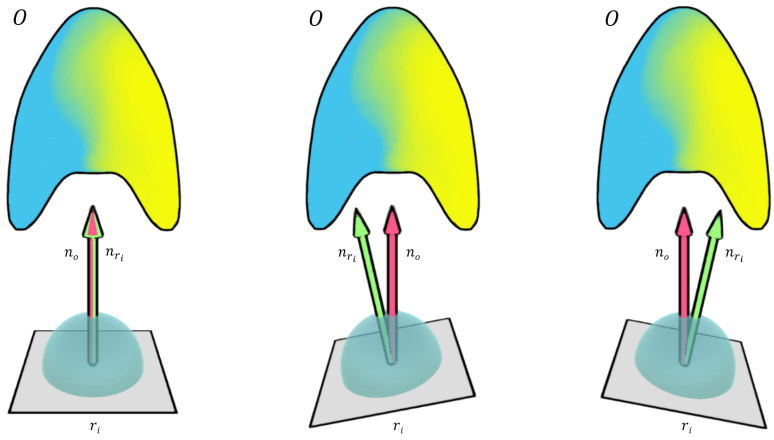


Figure 6. *Left:* n_o and n_{r_i} are equal and no approximation error is introduced due to the intermediate normal approach. *Middle and Right:* n_o and n_{r_i} differ; the radiance stored during precomputation is equally blue and yellow; at run-time both the blue and the yellow color channel are scaled equally based on $(n_o \cdot n_{r_i})_+$ instead of raising the blue channel and lowering the yellow channel (middle) or vice versa (right).

4.1.5 Precomputing the Radiance Transfer Operator

Examining Equation 7 it can be noticed that, thanks to the derivation in the previous subsections, every quantity within the integral operator is known at the precompute stage. This allows to simply evaluate each of the n integrals, assigning them a single scalar value:

$$t_j = \int_D \frac{Y_j(n_{s_i})\rho_{s_i}}{\pi^2}(\omega_{s_i} \cdot n_o)_+ \tag{8}$$

Considering Equation 7 as a whole, only the irradiance SH coefficients c_j and the normal n_{r_i} are unknown while offline. At run-time, the former need to be computed on-the-fly has detailed in Subsection 5.2.1, while the latter can be simply retrieved from the normal-buffer. Given the definition in Equation 8, Equation 7 can now be rewritten as follows:

$$L_{refl}(p_i) = \rho_{r_i}(n_o \cdot n_{r_i})_+ \sum_{j=1}^{n} c_j t_j \tag{9}$$

It is necessary to highlight that each coefficient t_j is specific to a single sample p_i. This means that radiance transfer coefficients are available only for a discrete number

of positions. However, DNRT should be able to provide a set of coefficients for any position surrounding the dynamic object O. This is achieved by interpolating for each point in space among the coefficients t_j of the closest samples p_i. Formally, this is expressed defining a set of functions T_j that take as argument an arbitrary area element r_i and return the corresponding interpolated coefficients t_j. Effectively, each function T_j can be imagined as a spatial scalar field for the corresponding coefficient t_j. This leads to the final reflected radiance equation:

$$L_{refl}(r_i) = \rho_{r_i}(n_o \cdot n_{r_i})_+ \sum_{j=1}^{n} c_j T_j(r_i) \tag{10}$$

4.2 Precomputing L_{occl}

DNRT models indirect light occlusions using an anti-radiance approach [Dachsbacher et al. 2007]:

- first, incoming radiance at a receiving area element r_i is computed without considering occlusions from dynamic objects.
- second, the amount of radiance that should have been occluded by dynamic objects is calculated and subtracted from the non-occluded incoming radiance.

Notice that a classic visibility-based approach for occlusions would require direct adjustments to the way the initial solution L_{init} is computed. More specifically, a modification of the visibility function used for each receiving area element r_i would be necessary. By using anti-radiance, DNRT can, instead, take L_{init} as-is and simply modify its value, rather than changing the calculations that lead to it.

As already introduced at the beginning of this Section, in this paper the anti-radiance function (also referred as subtractive radiance) is indicated with the symbol L_{occl}. L_{occl} can be defined as the sum of all the original radiance L_{init} that reaches a receiving area element r_i passing through an area element s_i, with s_i being part of the dynamic object O's surface. This translates to the following equation:

$$L_{occl}(r_i) = \rho_{r_i} \int_D \frac{L_{init}(\omega_{s_i})}{\pi} h(r_i, s_i)(\omega_s \cdot n_{r_i})_+ d\omega_{s_i} \tag{11}$$

A new derivation is necessary to present Equation 11 in a precomputable form. As $L_{init}(\omega_{s_i})$ is not known during precomputation, the scaled irradiance function $\frac{I(\omega_{s_i})}{\pi}$ is used in its place:

$$L_{occl}(r) = \rho_{r_i} \int_D \frac{I(\omega_{s_i})}{\pi^2} h(r_i, s_i)(\omega_s \cdot n_{r_i})_+ d\omega_{s_i} \tag{12}$$

Notice that the two functions $\frac{I(\omega_{s_i})}{\pi}$ and $L_{init}(\omega_{s_i})$ are not perfectly equivalent. This is because the irradiance function not only captures radiance coming onto r_i from all occluded area elements r_j, but it may also mistakenly capture radiance from unoccluded area elements r_k (see Figure 7). A possible solution to this problem is presented in Appendix B.

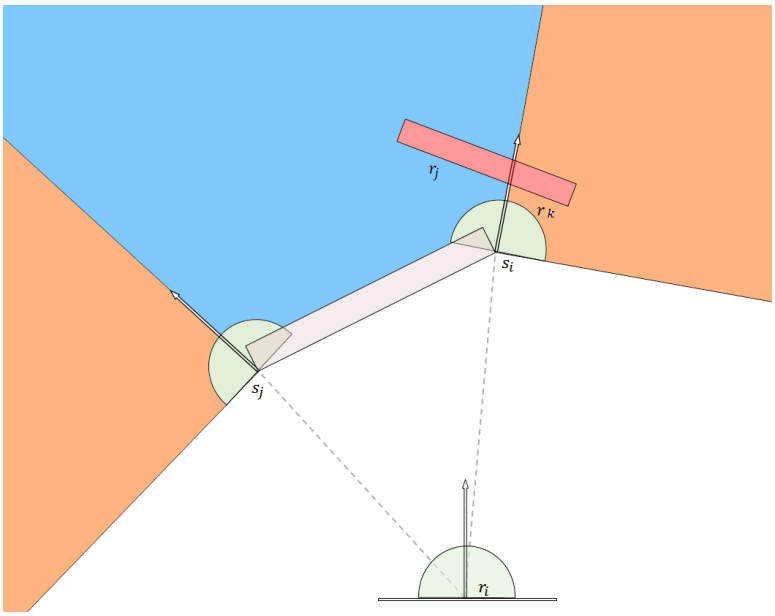


Figure 7. Only radiance from the blue area should be occluded. However, radiance from the orange area is captured as well.

Using an analogous derivation to the one provided for L_{refl} in Subsection 4.1, a specific set of radiance transfer coefficients k_j can be calculated for L_{occl}:

$$k_j = \int_D \frac{Y_j(\omega_{s_i})}{\pi^2} (\omega_{s_i} \cdot n_o)_+ d\omega_{s_i} \tag{13}$$

Notice that each coefficient k_j is specific to a single sampling point p_i. To associate a set of occlusion coefficients to any position in O neighborhood another field function K_j is defined. This leads to the following final equation for subtractive radiance:

$$L_{occl}(r_i) = \rho_{r_i}(n_o \cdot n_{r_i})_+ \sum_{j=1}^{n} c_j K_j(r_i) \tag{14}$$

4.3 Final Equation

The results from Subsection 4.1 and Subsection 4.2 can be now put together in the following equation for DNRT:

$$\begin{aligned} L_{final}(r_i) &= L_{init}(r_i) + L_{refl}(r_i) - L_{occl}(r_i) \\ &= L_{init}(r_i) + \rho_{r_i}(n_o \cdot n_{r_i})_+ \sum_{j=1}^{n} c_j T_j(r_i) - \rho_{r_i}(n_o \cdot n_{r_i})_+ \sum_{j=1}^{n} c_j K_j(r_i) \\ &= L_{init}(r_i) + \rho_{r_i}(n_o \cdot n_{r_i})_+ \sum_{j=1}^{n} c_j (T_j(r_i) - K_j(r_i)) \end{aligned} \tag{15}$$

L_{final} is the function representing the radiance at each area element r_i after DNRT has been applied to the initial solution L_{init}. Since, for each j, T_j and K_j are both known during precomputation, they can be combined in a single function M_j, giving the following final equation:

$$L_{final}(r_i) = L_{init}(r_i) + \rho_{r_i}(n_o \cdot n_{r_i})_+ \sum_{j=1}^{n} c_j M_j(r_i) \tag{16}$$

As further explained in Section 5, this combination will allow to lower DNRT's storage requirements and will also grant a performance improvement. In the rest of this paper, the difference between L_{refl} and L_{occl} will be referenced as L_{DNRT}, hence:

$$L_{final}(r_i) = L_{init}(r_i) + L_{DNRT}(r_i) \tag{17}$$

5 Implementing DNRT

The following is a brief description of how the theoretical results obtained in Section 4 have been used to develop an implementation for DNRT. As already introduced, the technique can be divided in two steps, precomputed and real-time. These are analyzed, respectively, in Subsection 5.1 and Subsection 5.2.

5.1 Precompute Step

The task of the precomputation step is to calculate the radiance transfer coefficients defined in Equation 8 and Equation 13. A different set of coefficients t_j and k_j has to be calculated for each sampling point p_i around each dynamic object O.

Listing 1: DNRT Precomputation Step

```
O ← Load(Obj, Tex);
o ← ComputeBoundingSphere(O).center;
grid ← BuildGrid(o);
foreach p_i ∈ grid.vertices do
    for j ← 0 to n do
        T_j(p_i) ← MonteCarloIntegration(O, j, reflection);
        K_j(p_i) ← MonteCarloIntegration(O, j, occlusion);
        M_j(p_i) ← T_j(p_i) − K_j(p_i);
        StoreToBitmaps(M_j(p_i));
    end
end
```

While DNRT's run-time module has been integrated within the Unity Engine, the precomputation step has been developed as a standalone C++ application. The application's inner-working can be summarized as follows:

1. A dynamic object O is loaded into the program along with its albedo texture.
2. O's centroid is determined. The centroid is defined as the object's bounding sphere's center.
3. The position of each sample p_i in O's neighborhood is fixed. Currently, a simple, regular grid is built around O. Each vertex of the grid constitutes a sampling point.
4. For each sampling point p_i, the corresponding set of radiance transfer coefficients t_j and k_j is calculated.
5. The coefficients of each sample are finally compressed and stored, ready to be loaded and used at run-time.

Operations (4) and (5) are particularly complex and deserve a more in-depth description, provided respectively in the Subsection 5.1.1 and Subsection 5.1.2. For a more formal rendition of the algorithm described above refer instead to Listing 1.

5.1.1 Coefficients Calculation

As seen in Equation 8 and Equation 13, to calculate each coefficient t_j and k_j of each sampling point p_i, an integral operation has to be solved. In the implementation here described, this has been done employing Monte-Carlo integration [Robert and Casella

1999]. In order to compute a given integral, Monte-Carlo integration requires a set of samples from its domain. In DNRT's case, the value of each sample is retrieved using raytracing [Whitted 1979]. Notice that, given a point p_i, to compute each of its coefficients t_j and k_j a separate Monte-Carlo integration process is necessary. However, exploiting the similarities in the coefficients' definitions, for a given sampling direction, a single raytracing query can be used to fetch the value of each coefficient's Monte-Carlo sample in that specific direction. This vastly reduces the number of raytracing queries necessary, granting a shorter processing time.

While efficiency is not the main priority during precomputation, minimizing the processing time associated with this phase is important; an extremely long waiting time (in the order of days) would make DNRT effectively unusable in actual videogame production. On this note, to further reduce the time cost associated with the precomputation step, both raytracing and Monte-Carlo integration have been implemented on the GPU using CUDA. Raytracing has been accelerated using a BVH data structure built following the model proposed in [Karras 2012].

5.1.2 Compression and Storage

Once the radiance transfer coefficients have been computed, they have to be compressed and stored for later usage during the run-time phase.

Compression is important as it lowers DNRT's memory requirements. Since, as seen in Equation 16, the coefficients t_j and k_j will ultimately be subtracted to each other, it is possible to directly store their difference $m_j = t_j - k_j$. This not only reduces the amount of memory required by DNRT, but also improves performance by anticipating the subtraction operation to the precompute stage and, more importantly, by reducing the necessary data bandwidth. Furthermore, while coefficients m_j are represented using a $float$ data type during their calculation, prior to storage they are cast to a $char$ format. This can be done with only a minor loss in precision as, accordingly with the definitions in Equation 8 and Equation 13, the value of each coefficient m_j falls within a limited range (see Table 1).

Notice that, for a 32x32x32 sampling grid, storing coefficients t_j and k_j separately and in $float$ format would take up to 4.72 MB of memory. Coefficients m_j represented in a $char$ format have instead a memory footprint of 884.74 KB.

Finally, particular attention has been given to the layout used to store the coefficients m_j, as data organization can have a considerable impact on performance. For each slice of the grid surrounding the dynamic object O, 7 images are saved to disk memory (a bitmap format is currently used). The first 6 have a 32bits format (RGBA), while the last uses only 24bits (RGB). Table 2 shows the coefficients layout for each pixel. The image slices are assembled into 7 Texture3D volumes, representing the grid in its

m_j index	Range
0	[-0.2820, 0.0000]
1	[-0.9772, 0.9772]
2	[-0.9772, 0.9772]
3	[-0.9772, 0.9772]
4	[-2.1850, 2.1850]
5	[-2.1850, 2.1850]
6	[-1.2615, 1.2615]
7	[-2.1850, 2.1850]
8	[-1.0925, 1.0925]

Table 1. Value range for each coefficient m_j

entirety, when DNRT's run-time module is first booted (see Subsection 5.2.1).

Image Index	Pixel.Red	Pixel.Green	Pixel.Blue	Pixel.Alpha
0	$m_0.red$	$m_1.red$	$m_2.red$	$m_3.red$
1	$m_0.green$	$m_1.green$	$m_2.green$	$m_3.green$
2	$m_0.blue$	$m_1.blue$	$m_2.blue$	$m_3.blue$
3	$m_4.red$	$m_5.red$	$m_6.red$	$m_7.red$
4	$m_4.green$	$m_5.green$	$m_6.green$	$m_7.green$
5	$m_4.blue$	$m_5.blue$	$m_6.blue$	$m_7.blue$
6	$m_8.red$	$m_8.green$	$m_8.blue$	$\nexists$

Table 2. Coefficients layout per image

Notice that this particular data disposition has been chosen to minimize, at run-time, the number of texture fetch operations and the number of GPU instructions overall. In particular, when a 4 SH coefficients approximation is used, the computation of the sum $\sum_{j=1}^{n} c_j m_j$ from Equation 16 is reduced in DNRT's run-time shader (see Subsection 5.2.2) to 3 texture sampling operations and 3 dot product operations (one for each color channel). For a 9 SH coefficients approximation, 7 texture fetch operations, 6 dot product operations, 1 vector multiplication and 2 vector sums are necessary.

5.2 Real-time Step

DNRT's run-time module has been developed as a post-processing effect for the Unity Engine's deferred rendering pipeline. This highlights the simplicity with which DNRT can be integrated with existing light probing technologies. Furthermore, it helps showing that DNRT is ready for use in actual videogame production.

DNRT's rut-time module can be divided in three components: CPU setup, GPU

shader and visibility shadowmapping. The following subsections provide a brief description of each of these components.

5.2.1 CPU setup

DNRT's impact on the CPU is minimal, as the only role of the central processor is to send to the GPU the data necessary for the execution of DNRT's shader (see Subsection 5.2.2).

When the Unity application is first launched, the image slices generated during the precomputation stage (see Subsection 5.1.2) are merged into 7 Texture3D volumes, which are then readily set as shader resources on the GPU.

At each frame and for each shader pass, the SH irradiance coefficients c_j at the dynamic object O's position are retrieved. More specifically, they are obtained by sampling and interpolating the coefficients of the light probes closest to O. If O has been subject to any rotation, the coefficients c_j are rotated accordingly using an SH rotation matrix [Green 2003]. This is necessary to ensure that coefficients c_j and m_j have the same coordinate system. Coefficients c_j are finally set on the GPU memory along with O's position, scale and rotation matrix.

5.2.2 GPU shader

On the GPU side, DNRT is implemented using a single pixel shader. This shader is applied on the framebuffer as a multi-pass post-processing effect. To each dynamic object corresponds one shader pass. Each pass modifies the previous solution adding reflections and occlusions from the corresponding dynamic object. This particular implementation has been chosen for the simplicity with which it can be integrated within the Unity Engine. In the future, however, a more direct integration within Unity's deferred lighting pass should be investigated, as it may remove unnecessary overhead.

The task of DNRT's shader is to combine the data generated during the precompute stage, the data set by the CPU (see Subsection 5.2.1) and the data from the G-buffer to compute Equation 16, here repeated for convenience:

$$L_{final}(r_i) = L_{init}(r_i) + \rho_{r_i}(n_o \cdot n_{r_i})_+ \sum_{j=1}^{n} c_j M_j(r_i) \tag{16}$$

The equation is calculated for each fragment f_i in the frame buffer, or better for each corresponding area element r_i. Notice that most of the parameters in Equation 16 are readily available to the shader. In particular:

– $L_{init}(r_i)$ can be fetched from the framebuffer, as it is simply the color of f_i before the shader is applied.

Listing 2: DNRT Shader (4 SH coefficients version)

```
r_i ← GetAreaElement(f_j);
n_{r_i} ← G-Buffer(f_j).normal;
o ← GetCentroid(O);
L_init(r_i) ← Framebuffer(f_j);
if IsInside(r_i, OuterVolume) then
    uvw_{r_i} ← GetTex3dCoords(r_i, o);
    // Using texture clamping, the value at the inner
       volume's boundary is automatically fetched when
       u, v, or w is greater than 1.
    m_{0,1,2,3}.red ← Sample (Texture3dVolume_0, uvw_{r_i});
    m_{0,1,2,3}.green ← Sample (Texture3dVolume_1, uvw_{r_i});
    m_{0,1,2,3}.blue ← Sample (Texture3dVolume_2, uvw_{r_i});
    lamb ← dot (n_{r_i}, n_o);
    L_DNRT(r_i).red ← lamb × dot (m_{0,1,2,3}.red, c_{0,1,2,3}.red);
    L_DNRT(r_i).green ← lamb × dot (m_{0,1,2,3}.green, c_{0,1,2,3}.green);
    L_DNRT(r_i).blue ← lamb × dot (m_{0,1,2,3}.blue, c_{0,1,2,3}.blue);
    if NotInside(r_i, InnerVolume) then
        ApplyQuadraticFalloff(L_DNRT(r_i), r_i, o);
    end
    return L_init(r_i) + L_DNRT(r_i);
else
    return L_init(r_i);
end
```

– ρ_{r_i} and n_{r_i}, which represents, respectively, r_i's albedo and normal vector, can be fetched from the G-buffer.

– The irradiance coefficients c_j are available in the shader's constant buffer, as they have been previously set by the CPU.

To determine the value of $M_j(r_i)$ a few passages are necessary. First, r_i's relative position to the dynamic object O is computed. This position is then converted to a set of uvw_{r_i} coordinates. Using uvw_{r_i}, the coefficients $m_j = M_j(r_i)$ are sampled from the 7 Texture3D volumes generated during precomputation. Notice that, as anticipated in Subsection 4.1.5, the value of each coefficient m_j is the result of an interpolation process that involves the coefficients of the samples p_i closest to r_i. As these coefficients are stored in a voxel from the above mentioned Texture3D volumes, DNRT can interpolate among them implicitly by simply enabling hardware texture filtering. After

retrieving the coefficients m_j, DNRT's shader has available all the data necessary to compute Equation 16. See Listing 2 for further details.

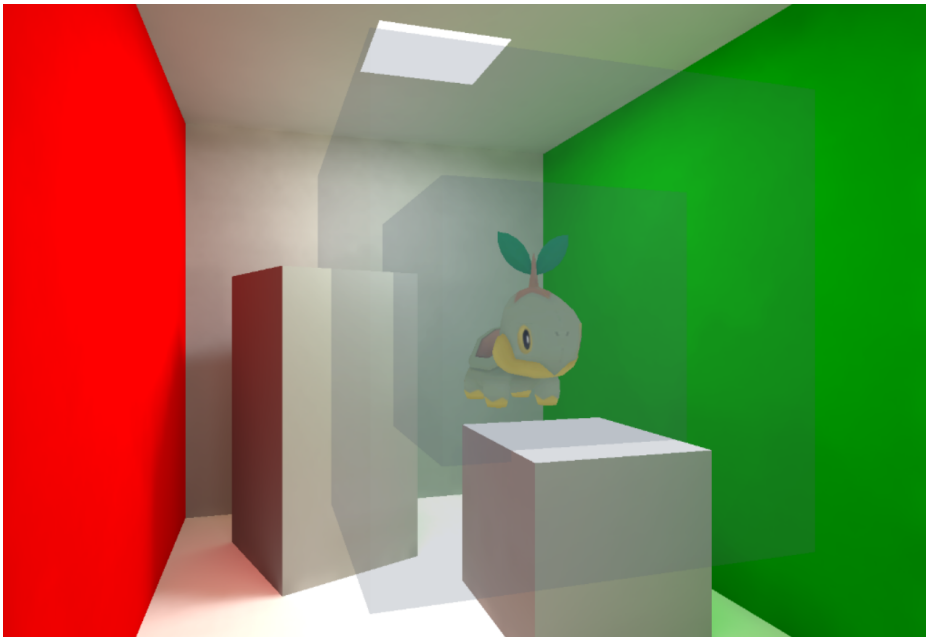


Figure 8. A dynamic object surrounded by an *inner* and an *outer* volume.

It should be noted that, in most cases, the influence of a dynamic object O on the environment is very subtle and is perceivable only for its immediate surroundings. This is used to boost DNRT's performance. More specifically, the dynamic object O is surrounded with two volumes (see Figure 8):

- if r_i is within the *inner* volume the shader is applied.
- if r_i is outside both volume, the shader skips r_i's corresponding fragment f_i.
- if r_i is within the *outer* volume, but outside the *inner* one, it receives "scaled radiance".

The last point refers to the fact that the outer volume is used as a buffer zone necessary to avoid abrupt chromatic changes. Throughout the outer volume, the radiance calculated by DNRT at the inner volume's boundary is gradually scaled to zero with the inverse of the squared distance.

5.2.3 Handling the Intrusion Problem

As seen in Subsection 4.1.2, during the precomputation step is not possible to handle situations in which an intruding object positions itself between the reflector O and

a receiver. This means that given two area elements r_i and r_j in O's neighborhood, they are both always influenced by O even in cases when, because of their respective positions, one should obstruct the other.

To address this problem, DNRT provides the option to enable an omnidirectional shadowmap for each dynamic object. The current implementation uses a cubemap, rendered at each frame. The cubemap's origin is positioned at the corresponding dynamic object's centroid and DNRT is applied to an area element r_i only if it survives the depth test. This solution presents two problems: (1) it reduces the dynamic object to a single point, while it should be handled as an area reflector; (2) it has a considerable impact on performance (see Subsection 7.1). More sophisticated techniques like Imperfect Shadowmapping [Ritschel et al. 2008] and Percentage-Close Soft Shadows [Fernando 2005] will be examined in the future to address this issues (see Section 9).

6 Method

Figure 9. Cornell Box (left); Corridor Lighting Example (middle); Sponza (right)

To analyze DNRT's performance and the quality of its output, four different tests have been designed. These tests have been executed in three different environments: the classic Cornell Box [Cornell 2016] (Test 1 and Test 2), the Corridor Lighting Example scene from Unity's Asset Store [Unity 2016] (Test 3), and the Sponza environment from Crytek [Crytek 2016] (Test 4).

Notices that, as DNRT only handles diffuse reflection, all environments' materials have been changed to fully diffuse. Subsections 6.1, 6.2, and 6.3 provide further details about each scenario and the associated tests. Table 3 gives instead a brief summary of the scenarios' polygonal complexity along with the dynamic objects' complexity.

In order to give context to DNRT's performance and to the quality of its output, the results from each test have been compared with the results from a set of competing

Scene	Polygons	Vertices
Cornell Box	38	72
Corridor	243.975	275.461
Sponza	279.105	153.635

(a) *Scenes' Polygonal complexity*

Model	Polygons	Vertices
Model 1	1.572	1.509
Model 2	1.532	1.209
Model 3	2.598	1.947
Model 4	777	983
Model 5	1.534	1.416
Model 6	1.510	1.536
Model 7	1.496	1.670
Model 8	728	795
Model 9	1.593	1.770
Model 10	1.502	1.139

(b) *Models' Polygonal complexity*

Table 3. Polygonal complexity of scenes and models.

technologies. Further details in regards to these technologies and the reasoning behind their choice are provided in Subsection 6.4.

Finally, Subsection 6.5 gives a brief run down of all the different hardware configurations on which the aforementioned tests have been executed. Performance on each different configuration has been measured using Unity's built-in profiling tools.

6.1 Cornell Box

The Cornell Box scenario has been used as background for two different tests.

Test 1 aims to determine which of DNRT's possible configurations offers the best compromise between quality and performance. Two factors have been taken into account: the number of SH coefficients used to represent the radiance transfer operator and the resolution of the Texture3D volumes containing such coefficients. Based on these parameters Test 1 has been split in 6 different instances, each representing a different configuration (see Table 4). In all testing instances, the Cornell Box has been populated with 10 different dynamic objects.

The goal of Test 2 is to study how DNRT's performance scales with respect to the number of dynamic reflectors present in the scene. From a qualitative standpoint, this test helps analyzing how well DNRT combines the influence of different dynamic objects on the environment. To achieve these objectives, Test 2 has been split into three instances, in which the Cornell Box has been populated, respectively, with 1, 5 and 10 dynamic objects.

Note that the Cornell Box used in Test 1 and 2 is a slightly modified version of the

Instance	Resolution	SH coefficients
1	16x16x16	4
2	32x32x32	4
3	64x64x64	4
4	16x16x16	9
5	32x32x32	9
6	64x64x64	9

Table 4. Cornell preliminary tests configurations

classic scene. For convenience reasons, the usual area light has been approximated with a bright point light. Furthermore, to maximize the amount of indirect light within the scene, the Cornell Box has been closed off with a back wall.

6.2 Corridor Scene

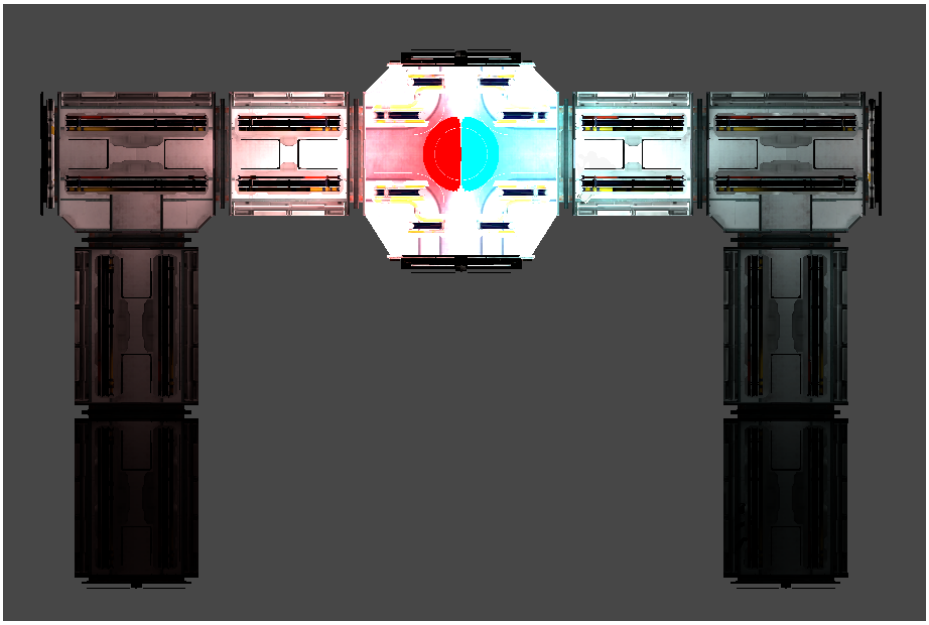


Figure 10. Top view of the modified Corridor Lighting Example Scene.

The Corridor Lighting Example [Unity 2016], populated with a single dynamic object (Model 3), is host of Test 3. This test has the goal of examining DNRT's performance and output quality when the technique is applied to an indoor scenario. Of particular interest are situations in which the only illumination available is indirect light generated by a high number of bounces. A secondary objective is to show that DNRT can maintain a convincing level of performance in scenes with a certain geometric

complexity, that resemble more closely actual videogames levels.

To better suit the aforementioned objectives, the Corridor Scene has been subject to minor adjustments: all light sources have been disabled and replaced with a single, bright spherical emitter, positioned at the center of the environment (see Figure 10). Note that this emitter can illuminate the environment's corners only through multiple-bounces of indirect light.

6.3 Sponza

The Sponza environment, populated with one dynamic object (Model 1), has been chosen to host Test 4. The objective of this test is to analyze DNRT's output quality when the method is applied to outdoor scenarios, characterized by complex interactions between direct and indirect light. Furthermore, Test 4 shares the secondary objective of Test 3, i.e. proving that DNRT can deliver a solid level of performance in high-poly environments.

Given its use in many Global Illumination papers, the Sponza scenario grants also the possibility to compare DNRT, albeit indirectly, with other techniques like Light Propagation Volumes [Kaplanyan and Dachsbacher 2010] and Voxel Cone Tracing [Crassin et al. 2011].

6.4 Competing Technologies

To better contextualize DNRT's results, the method has been tested against two competing techniques: SSAO [Shanmugam and Arikan 2007] and HBAO [Bavoil et al. 2008]. These techniques have been chosen since, similarly to DNRT, they apply occlusions in screen space using information from the G-buffer (for further details see 2). Furthermore, since they are often employed in high-profile games, a favorable comparison may prove DNRT's suitability for actual use in videogame production.

As a point of reference, a ground-truth solution is provided for each test. This solution has been generated using Unity's lightmaps' baking functionality. Enlighten's starting solution is also provided to highlight, by contrast, DNRT's contributions.

6.5 Hardware Configurations

Four different hardware setups have been used to test DNRT's performance. These hardware configurations are listed in Table 5. Their choice aims to provide a good balance, spanning from low-tier GPUs to more advance ones. Both desktop and mobile hardware configurations are included. GPUs from multiple hardware manufacturers (Nvidia, AMD and Intel) are represented.

CPU	GPU
Intel i7 4720HQ 2.60 GHz (3.60 GHz Turbo)	Intel HD Graphics 4600
Intel i7 4720HQ 2.60 GHz (3.60 GHz Turbo)	NVIDIA GTX 960M 4 GB
Intel i5 2500k 3.30 GHz (3.70 GHz Turbo)	AMD ATI HD 6950 2GB
Intel Xeon E5-1620 v3 3.50 GHz (3.60 GHz Turbo)	NVIDIA GTX 970 4GB

Table 5. Hardware configurations used for testing.

7 Results

The results from the four tests described in Section 6 are here presented. Performance and visual quality are analyzed separately in Subsection 7.1 and Subsection 7.2.

7.1 Performance

This Section gives an analysis of DNRT's performance in Test 1 to 4, comparing the method with the chosen competing techniques, SSAO and HBAO. Note that all measurements indicate average values over a 10 seconds span. All tests have been executed at a 1366×768 resolution. Unity's built-in profiling tools have been used to measure the overall time duration of each frame. The execution time of DNRT, SSAO and HBAO within a frame (referred as Method's Cost) has been measured using the Intel GPA software.

7.1.1 Intrusion Problem and time efficiency

		Cornell Box - Test 2 - 10 Models	
		Frametime (ms)	Method's Cost (ms)
Intel HD 4600	Unity	34.482	
	DNRT Basic	58.823	29.712
	DNRT Intrusion	76.923	33.078
AMD HD 6950	Unity	2.747	
	DNRT Basic	4.219	1.450
	DNRT Intrusion	7.663	1.618
NVIDIA GTX960M	Unity	2.398	
	DNRT Basic	4.608	2.236
	DNRT Intrusion	8.695	2.538
NVIDIA GTX970	Unity	0.940	
	DNRT Basic	1.661	0.800
	DNRT Intrusion	7.042	1.151

Table 6. Time comparison between Unity's base performance (Unity) and DNRT with the intrusion detection option enabled (DNRT Intrusion) and disabled (DNRT Basic).

As introduced in Subsection 5.2.3, DNRT addresses the intrusion problem, i.e. the problem of dealing with an object positioned between the dynamic reflector O and a receiver r_i, using cubic shadowmapping. During testing this approach has yielded underwhelming results both in terms of quality and performance. Visual artifacts are analyzed in Subsection 7.2.1. In regards to the time cost, Table 6 compares DNRT's performance with cubic shadowmapping disabled (DNRT Basic) and enabled (DNRT Intrusion). Configuration 2 from Table 4 has been used (32x32x32 resolution with 4 SH coefficients).

The cost of DNRT's shader is only marginally higher when the intrusion detection option is enabled. This is because the only additional operation is a depth test for each considered fragment. However, the overall frametime (1/framerate) is considerably longer due to the current cubic shadowmapping approach to the intrusion problem. For each frame, this solution requires to render the environment 6 times per dynamic object. This results in low performance, especially when multiple objects are within the scene. In the future more performant solutions may be explored (see Subsection 9.4). At the moment, all remaining tests have been executed with the intrusion detection option disabled.

7.1.2 Cornell Box - Test 1

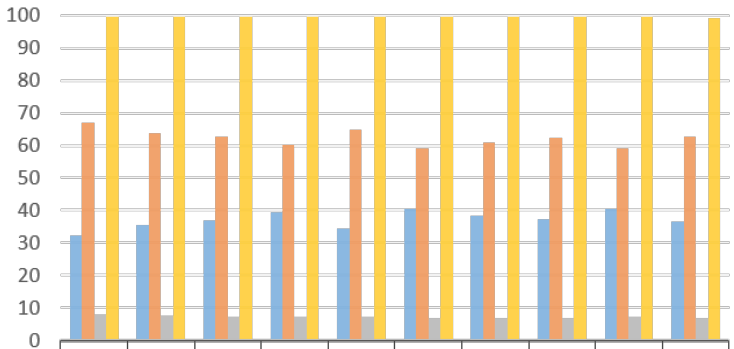
In order to find which of DNRT's possible configurations offers the best compromise between quality and performance, in Test 1, six different combinations of its input parameters have been examined (see Table 4). Performance results for different Texture3D resolutions have been omitted, as no measurable impact on DNRT's time cost related to this parameter was detected. The number of SH coefficients used, instead, significantly influences DNRT's time cost, as depicted in Table 7.

		Cornell Box - Test 1	
		Frametime (ms)	Method's Cost (ms)
Intel HD 4600	DNRT 4 SH	62.500	29.712
	DNRT 9 SH	83.333	53.693
AMD HD 6950	DNRT 4 SH	4.219	1.450
	DNRT 9 SH	4.608	1.800
NVIDIA GTX960M	DNRT 4 SH	4.608	2.236
	DNRT 9 SH	6.622	4.302
NVIDIA GTX970	DNRT 4 SH	1.661	0.800
	DNRT 9 SH	2.375	1.546

Table 7. DNRT's 4 SH coefficients configuration compared against the 9 SH coefficients configuration.

Using the 9 SH coefficients configuration leads to a time cost growth that goes from a

24.14% increase registered for the AMD ATI HD 6950 GPU up to a 93.25% measured on the NVIDIA GTX970. This loss in performance is due to the fact that DNRT's shader is bandwidth bound, with the sampling operations to fetch data from the Texture3D volumes constituting a bottleneck. Figure 11 presents several occupancy parameters of DNRT's shader, in a 9 SH coefficients configuration, across 10 passes (1 per object, Cornell Box - Test 2) for a rendering frame captured on the Intel HD 4600 GPU using the Intel GPA software. Due to the sampler bottleneck, DNRT's pixel shader function is forced to an idling state for 60% of its execution time.



	1	2	3	4	5	6	7	8	9	10
PS EU Active %	32.3	35.56	36.84	39.42	34.6	40.38	38.41	37.27	40.46	36.6
PS EU Stall %	67.13	63.88	62.6	60.03	64.87	59.08	61.06	62.19	59.03	62.82
Sampler Bottleneck %	8	7.68	7.3	7.16	7.06	6.88	7.02	6.8	7.08	6.84
Samplers Busy %	99.42	99.43	99.43	99.44	99.45	99.45	99.47	99.46	99.48	99.36

Figure 11. DNRT shader occupancy parameters. PS EU is the pixel shader execution unit.

The lower performance given by a 9 coefficients configuration would be acceptable only if matched by considerable gains in visual quality. However, this is not the case, as argued in Subsection 7.2.2. Therefore, the 4 coefficients configuration has been chosen as DNRT's standard setup and used for the remaining tests (Test 2, 3 and 4).

7.1.3 Cornell Box - Test 2

From a performance analysis of the results from Test 2, it emerges that DNRT's time cost is linearly dependent on the number of dynamic objects on screen. This trend is consistent across all 4 considered hardware configurations, as shown in Table 8. SSAO and HBAO, as expected, do not show any dependence on this factor.

While DNRT is 10 times faster than SSAO and HBAO in scenarios featuring only one dynamic object, its performance aligns with the results from its two competitors when 10 dynamic objects are present in the environment.

Notice that the linearity of DNRT's performance is not surprising given the current

		Cornell Box - Test 2					
		1 Model		5 Models		10 Models	
		Frametime (ms)	Method's Cost (ms)	Frametime (ms)	Method's Cost (ms)	Frametime (ms)	Method's Cost (ms)
Intel HD 4600	Unity	30.303		31.250		34.482	
	DNRT	33.333	2.708	43.478	14.417	58.823	29.712
	SSAO	50.000	19.371	52.631	20.340	52.631	20.232
	HBAO	62.500	31.387	62.500	31.901	62.500	31.213
AMD HD 6950	Unity	2.739		2.739		2.747	
	DNRT	2.754	0.191	3.412	0.746	4.219	1.450
	SSAO	4.651	1.646	4.672	1.665	4.651	1.682
	HBAO	4.807	2.021	4.830	1.994	4.878	2.021
NVIDIA GTX960M	Unity	2.325		2.380		2.398	
	DNRT	2.398	0.202	3.333	1.092	4.608	2.236
	SSAO	4.166	1.821	4.166	1.825	4.273	1.825
	HBAO	4.291	1.966	4.255	1.969	4.524	2.105
NVIDIA GTX970	Unity	0.842		0.903		0.940	
	DNRT	0.914	0.077	1.322	0.395	1.661	0.800
	SSAO	1.633	0.743	1.658	0.746	1.760	0.750
	HBAO	1.607	0.734	1.683	0.759	1.607	0.741

Table 8. Results from Test 2. DNRT exhibits a linear dependency on the number of dynamic objects.

implementation. DNRT's shader is to the applied in multiple passes, with one pass per dynamic objects. Each pass is likely to a similar time cost, hence the linear growth with respect to the number of dynamic objects. A future single-pass implementation directly integrated within Unity's deferred rendering pass, may present a sublinear growth.

7.1.4 Corridor Scene and Sponza - Test 3 and Test 4

Table 9 presents the results from Test 3 and Test 4. Although the scenarios on which these tests have been executed, respectively the Corridor Scene and Sponza, have a much higher polygonal complexity than the Cornell Box (see Table 3a), DNRT's performance is mostly unchanged. This shows that DNRT's time complexity is geometry independent.

It should be noted that the time costs for Test 3 are consistently higher than the ones regarding Test 4. This is because the model used in Test 3 is larger than the one used in Test 4. A higher scale means wider *inner* and *outer* volumes surrounding the dynamic object and, therefore, a higher number of fragments on which DNRT's shader is applied (see Subsection 5.2.2). In other words, due to the surrounding volumes optimization, DNRT exhibits a dependency on the dynamic objects' scale.

		Corridor Scene - Test 3		Sponza - Test 4	
		Frametime (ms)	Method's Cost (ms)	Frametime (ms)	Method's Cost (ms)
Intel HD 4600	Unity	27.027		50.000	
	DNRT	30.303	3.827	52.631	3.281
	SSAO	50.000	26.921	76.923	23.414
	HBAO	55.555	32.181	83.333	32.691
AMD HD 6950	Unity	2.487		4.065	
	DNRT	2.604	0.239	4.132	0.179
	SSAO	5.714	2.942	6.756	2.302
	HBAO	4.716	1.972	5.681	1.985
NVIDIA GTX960M	Unity	2.364		4.255	
	DNRT	2.398	0.296	4.484	0.266
	SSAO	4.672	2.229	7.299	2.328
	HBAO	4.587	2.200	7.142	2.226
NVIDIA GTX970	Unity	0.889		1.760	
	DNRT	0.956	0.112	1.785	0.098
	SSAO	1.915	0.980	3.067	1.109
	HBAO	1.798	0.812	2.724	0.822

Table 9. Results from Test 3 and Test 4.

7.1.5 Performance Summary

A measurable effect of the Texture3D resolution on DNRT's performance has not been detected. The number of SH coefficients used, instead, changes substantially DNRT's time cost. This is because DNRT is bandwidth bound and the usage of more SH coefficients entails a higher number of sampling operations.

DNRT runs 10 times faster than SSAO and HBAO when a one single dynamic object is present in the scene. However, the method's time complexity has exhibited a linear dependency on the number of dynamic objects. This means that DNRT's matches SSAO and HBAO's performance when 10 dynamic objects are part of the environment. Furthemore, DNRT performance is dependent on the dynamic object's scale, as this affects the size of the surrounding volumes (see Subsection 5.2.2) and therefore the number of considered fragments.

Finally, by presenting similar performance in all environments, DNRT has not shown a dependency on the geometric complexity of the scene.

7.2 Visual Quality

This Section gives an analysis of DNRT's results from a qualitative point of view. The output of each experiment is analyzed in a dedicated Subsection.

7.2.1 Intrusion Artifacts

DNRT solves the intrusion problem described in Subsection 5.2.3, by rendering a cubic shadowmap at each dynamic object's position. A receiver r_i is affected by a dynamic object O only if visible from its centroid. This constitutes a substantial simplification, as basically each dynamic object is treated as a point light, rather than an area reflector.

During experimentation, this approach has led to noticeable visual artifacts in the form of hard chromatic edges, as visible in Figure 12. Considering also the underwhelming performance shown in Subsection 7.1.1, it is clear that the adoption of a more sophisticated solution will be necessary in the future. A selected number of possible options are presented in Subsection 9.4.

Note that, because of the aforementioned limitations, the results described in the upcoming Subsections have been generated with the intrusion detection feature disabled.

Figure 12. The cubic shadowmapping solution to the intrusion problem leads to hard chromatic edges.

7.2.2 Cornell Box - Test 1

(a) *DNRT 4 SH coefficients, 32x32x32*

(d) *DNRT 9 SH coefficients, 32x32x32*

Figure 13. DNRT's output using 4 and 9 SH coefficients.

Each instance of Test 1 is distinguished by resolution and number of SH coefficients used; these parameters have a subtle, but noticeable effect on DNRT's output.

Although the use of 9 SH coefficients allows to strengthen certain details (see Figure 13), overall, the advantages in terms of quality over a 4 SH coefficients configuration are only minor and do not justify the considerable loss in performance caused by the use of a higher number of coefficients (see Subsection 7.1.2).

In regards to the Texture3D resolution, Figure 14 shows that this parameter affects the sharpness of DNRT's reflections and occlusions. The 16x16x16 configuration gives overly blurry results, yielding a considerable loss of detail. While the 64x64x64 configuration gives the sharpest effects, its difference from the 32x32x32 setup is not substantial. The latter, therefore, constitutes the better choice, given its much lower memory requirements (393.216 KB vs 3.145 MB per object).

Putting together the considerations on the coefficients number and the resolution made above, along with the performance analysis from Subsection 7.1.2, configuration 2 (see Table 4), which uses a 32x32x32 resolution and 4 SH coefficients, emerges as the one providing the best compromise between quality, performance and storage requirements. For this reason it has been selected as the standard configuration.

(a) *DNRT 4 SH coefficients, 16x16x16*

(b) *DNRT 4 SH coefficients, 32x32x32*

(c) *DNRT 4 SH coefficients, 64x64x64*

Figure 14. DNRT's using different Texture3D resolutions.

7.2.3 Cornell Box - Test 2

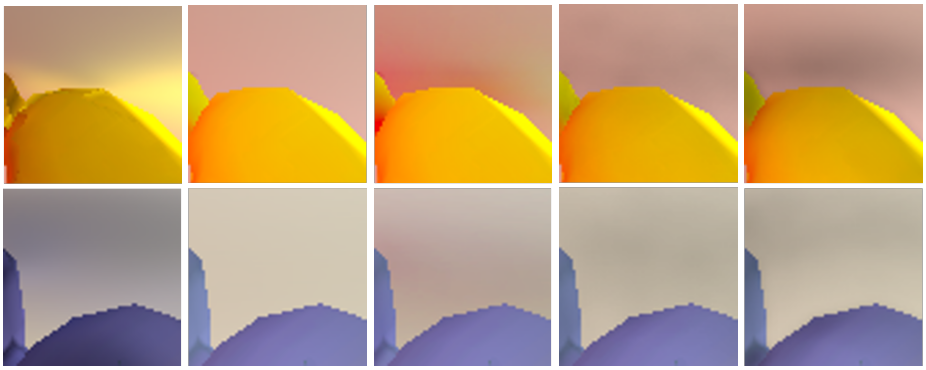
Test 2 is comprised of three separate instances, each featuring a different number of dynamic objects. Here a qualitative analysis of the third instance, in which 10 dynamic objects populate the Cornell Box, is provided. The results from instance 1 and 2 are presented in Appendix C along with other DNRT's output images.

Figure 16 presents a comparison between the ground-truth, Enlighten, DNRT, SSAO and HBAO. It is immediately noticeable that all techniques present important differences with the ground-truth.

As expected, the Enlighten' solution ignores the dynamic objects, and therefore lacks all the reflections and occlusions casted by these objects onto the environment.

SSAO and HBAO operate on an approximation of the environment that is solely based on the data available in the frame-buffer and the depth-buffer. As seen in Subsection 2.2, this leads to visual artifacts, like the incorrect darkening of concave edges due to the inaccurate detection of an occluder. This is particularly evident in Figure 16e and 16f. Furthermore, SSAO and HBAO simulate only close proximity shadows, hence their respective solutions do not feature any reflection effect.

DNRT is capable of representing both occlusions and reflections, although they lack intensity when compared with the ground-truth. This may be caused by the anti-radiance approximation discussed in Subsection 4.2. Nonetheless, DNRT represents a clear improvement over the Enlighten's solution, enhancing it with complex light interactions from dynamic objects onto the static environment and onto other objects. This is highlighted in Figure 16d that shows how neighboring objects clearly affect each other.



(a) *Ground-Truth* **(b)** *Enlighten* **(c)** *DNRT* **(d)** *SSAO* **(e)** *HBAO*

Figure 15. Test 2 - Detailed comparison.

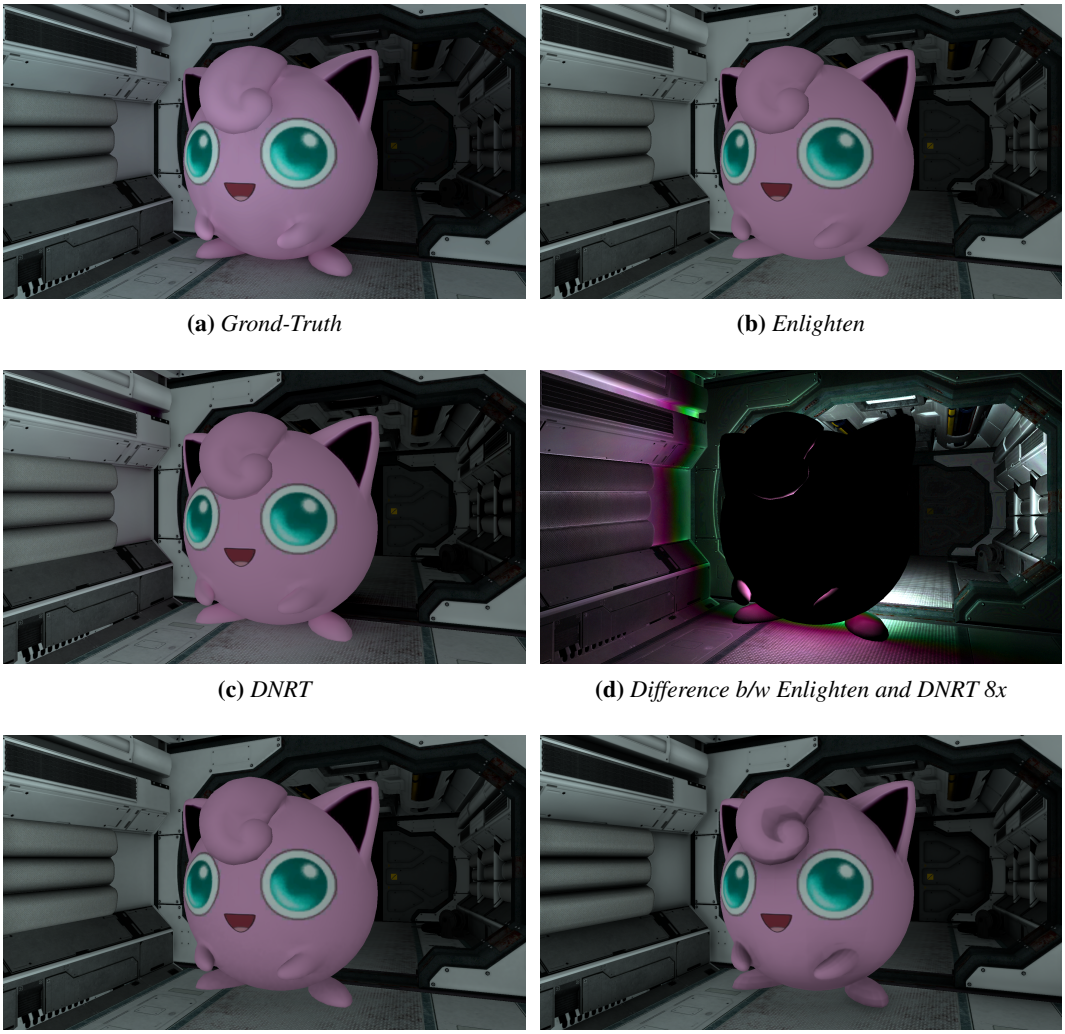
(a) *Ground-truth* **(b)** *Enlighten*

(c) *DNRT* **(d)** *Difference b/w Enlighten and DNRT 8x*

(e) *SSAO* **(f)** *HBAO*

Figure 16. Test 2 results.

7.2.4 Corridor Scene - Test 3



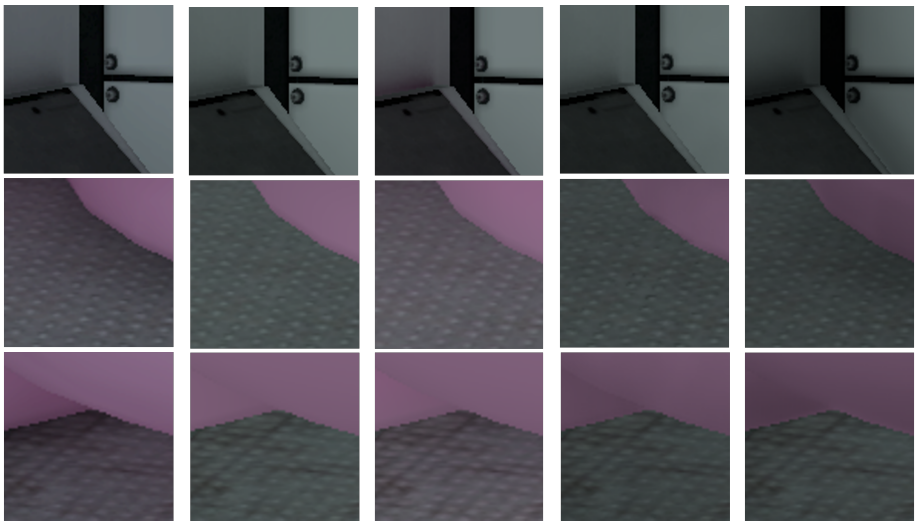
(a) *Grond-Truth* **(b)** *Enlighten*

(c) *DNRT* **(d)** *Difference b/w Enlighten and DNRT 8x*

(e) *SSAO* **(f)** *HBAO*

Figure 17. Test 3 results.

In the scenario selected for Test 3, the Corridor Scene, indirect light has an overall direction, going from the environment's center towards the peripheries of its two corridors. When placing a big object midway through a corridor, radiance is bounced back toward the center and the latter part of the corridor is darkened as a result. This phenomenon is evident when comparing the ground-truth with the Enlightens' output (see Figure 17). In the Enlightens' solution light fades uniformly throughout the corridor. The ground-truth instead shows a clear jump in brightness caused by the presence of the dynamic object. In particular, the object occludes radiance, darkening the rest of the corridor. It also reflects radiance forward giving a pink tint to the



(a) *Ground-Truth* **(b)** *Enlighten* **(c)** *DNRT* **(d)** *SSAO* **(e)** *HBAO*

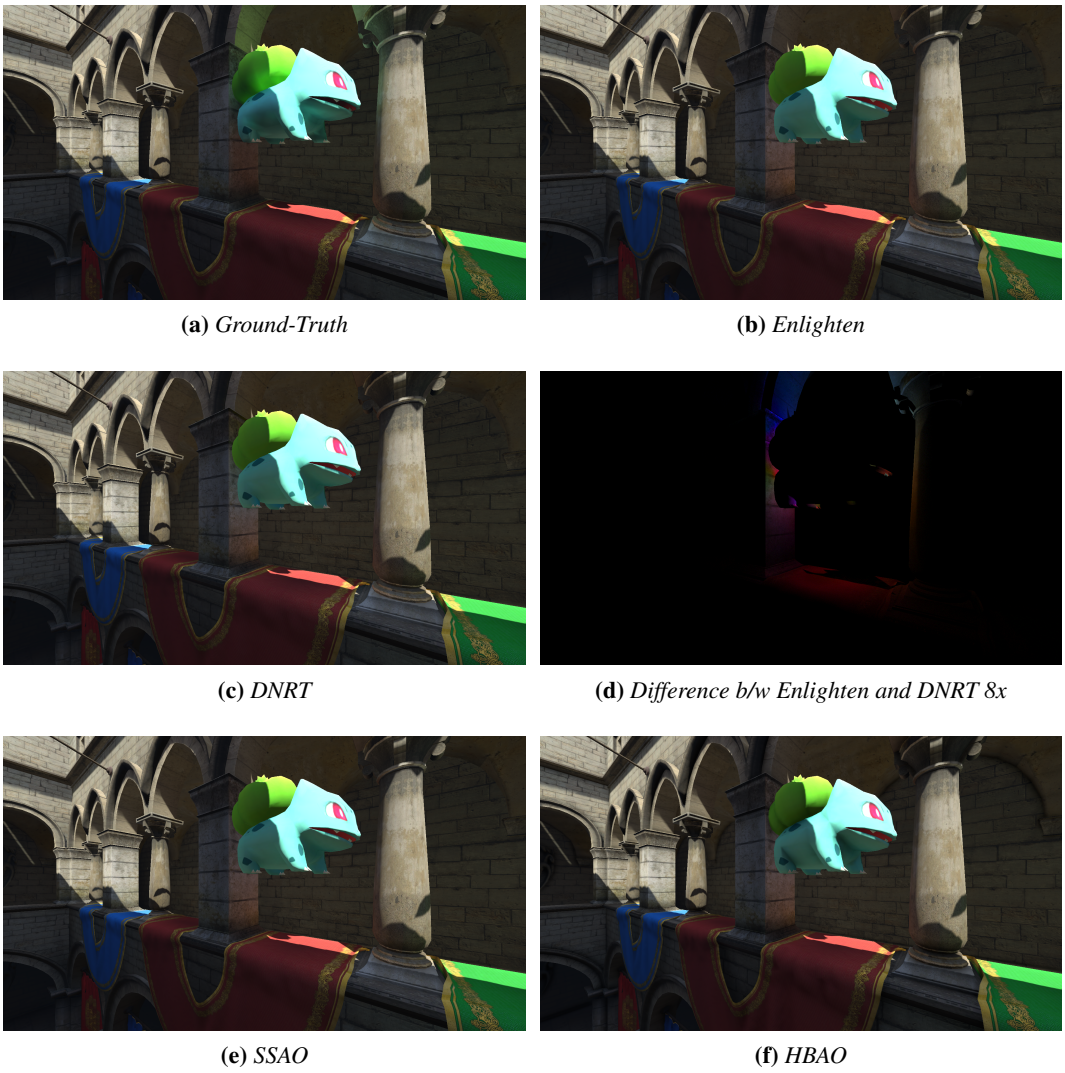
Figure 18. Test 3 - Detailed comparison.

corridor in that direction.

Since SSAO and HBAO only model close proximity occlusions, they cannot replicate the phenomena described above. DNRT gives instead a better approximation of the ground-truth, replicating both the pink reflection and the darkening of the latter part of the corridor. This approximation is not perfect: the overall chromatic tone of the scene is slightly different and fine shadowing details at the object's feet are lost (see Figure 18). The former is likely caused by the one-bounce limitation, while the latter may depend on the intermediate normal approximation and/or on the anti-radiance approximation. Furthermore, the end of the corridor in DNRT's solution is brighter than in the ground-truth. This could be a drawback of the surrounding volumes optimization 5.2.2: the corridor's end likely falls outside the *inner* volume, but inside the *outer* volume, where DNRT's effect is gradually diminished with a quadratic falloff.

It should be noted that by redistributing an pre-existing light solution (for static geometry) characterized by multiple-bounces of indirect light, DNRT effectively overcomes one of the limitations that characterize fully dynamic techniques like LPV [Kaplanyan and Dachsbacher 2010] and VCT [Crassin et al. 2011]. These techniques only simulate up to two bounces of indirect light.

7.2.5 Sponza - Test 4



(a) *Ground-Truth* **(b)** *Enlighten*

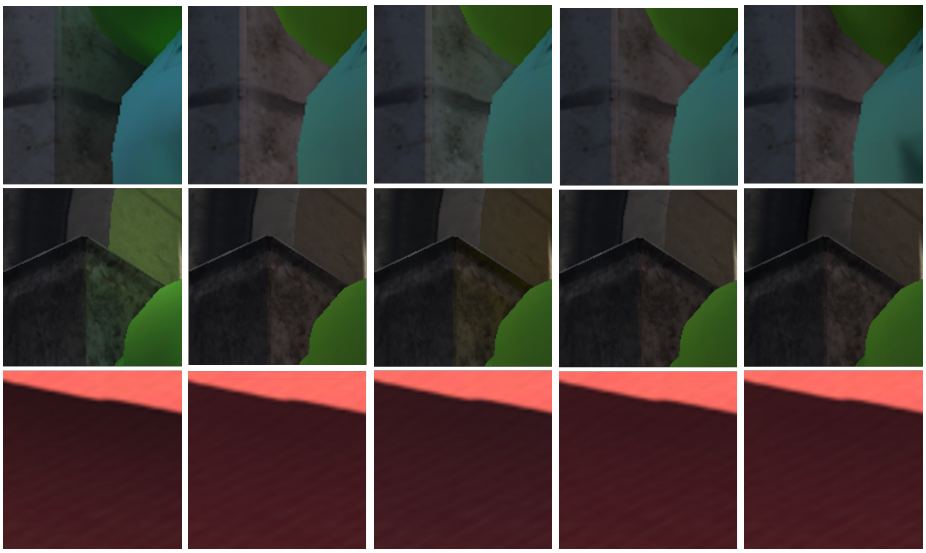
(c) *DNRT* **(d)** *Difference b/w Enlighten and DNRT 8x*

(e) *SSAO* **(f)** *HBAO*

Figure 19. Test 4 results.

Test 4 examines DNRT's behavior in an outdoor environment (Sponza) characterized by a mix of bright direct light and indirect light. DNRT's output is illustrated in Figure 19 along with the ground-truth and the output from the considered competing techniques.

Of particular interest is the interaction between the red cloth and the white marble pillar in the image's center (see Figure 20). In the ground-truth, the red indirect light coming from the cloth is dominated, midway toward the top of the pillar, by the green reflection coming from the dynamic object. DNRT replicates this phenomenon,



(a) *Ground-Truth* **(b)** *Enlighten* **(c)** *DNRT* **(d)** *SSAO* **(e)** *HBAO*

Figure 20. Test 4 - Detailed comparison.

which is instead missing in the Enlightens' output (the pillar has a red tint from top to bottom). The reflected light is, however, less intense. SSAO and HBAO simulate only close proximity occlusions, therefore this detail is missing from their output.

7.2.6 Visual Quality Summary

DNRT enriches a pre-existing radiance solution for static geometry with reflections and occlusions from dynamic objects. Results resemble the ground-truth, although a substantial difference is still present. This is mostly due to a lack of intensity, especially concerning indirect occlusions. The current anti-radiance approximation may be the root of this problem. Chromatic differences with the ground-truth are also caused by DNRT's one-bounce limitation.

When compared against SSAO and HBAO, DNRT shows several advantages: (1) it is capable of simulating reflections; (2) as shown in Subsection 7.2.4, it is not limited to close proximity occlusions; (3) it does not present artifacts like the typical darkening of concave edges; (4) it is capable of more accurate results by taking into account the radiance direction (it does not assume that the environment is illuminated by an ambient light).

8 Conclusion

This paper has introduced DNRT, a precomputation-based method capable of enriching the output of light probing technologies with one additional bounce (or occlusion) of indirect light from dynamic objects.

Thanks to its redistribution approach, inspired by incremental radiosity techniques, and its deferred rendering implementation, DNRT is capable of offering a level of performance in orders of magnitude higher than previous PRT-based neighboring techniques [Sloan et al. 2002; Lehtinen and Kautz 2003; Pan et al. 2007]. When applied in scenes with a limited number of dynamic objects, DNRT is also more efficient than current screen-space techniques.

Given the low time cost, DNRT may be used with success in actual videogame production. The method could be particularly effective for genres of games that, by design, present a limited number of dynamic objects, like fighting games and racing games. It may also be used selectively on the player's avatar in third person games and in a much wider breadth of applications.

In terms of image fidelity, DNRT gives perceptually convincing results. The addition of indirect light reflection and occlusion from dynamic objects onto static geometry and other dynamic objects enhances the output of technologies that solely simulate static-to-static and static-to-dynamic interactions (e.g. Enlighten). However, the gap from the ground-truth is still substantial. In the future, an extension to multiple bounces of indirect light and arbitrary normal directions, may help close this distance.

9 Future Work

While in its current state DNRT is capable of generating qualitatively satisfying results at a low computational cost, the method still offers ample space for numerous improvements. Here is provided a brief description of the ones that could benefit DNRT the most. For a full account, see Appendix B.

9.1 Multiple-bounces DNRT

As seen in Subsection 7.2, DNRT's output presents a notable difference from the ground-truth. This discrepancy is likely tied, at least in part, to the fact that DNRT simulates only one bounce of indirect light. Therefore, overcoming this limitation, may allow DNRT to generate far more accurate results.

A promising approach for a multiple-bounces extension could be based on an iterative solution, in which DNRT is applied numerous times per frame. The method would

intuitively work as follows:

1. Irradiance coefficients c_j are sampled from the light-probes.
2. Given a dynamic object O, the environment surrounding this object is rendered in a cubemap E_0.
3. Using the coefficients c_j as input, DNRT is applied to E_0 (rather than to the framebuffer). The result is a modified cubemap E_1.
4. E_1 is convolved [Ramamoorthi and Hanrahan 2001] and projected into SH coefficients e_{1_j}.
5. DNRT is applied on E_1 using e_{1_j} as input, rather than c_j. The result is a new cubemap E_2.
6. The process is repeated from step 4 for a predetermined number of iterations n.
7. Finally, DNRT is applied to the framebuffer using e_{n_j} as input.

The process described above employs, however, expensive operations, like the cubemap rendering and its convolution. Better performance could theoretically be achieved by modifying directly the irradiance coefficients, bypassing the cubemap and using instead an analytical function q defined as follows:

$$\begin{aligned} q(c_j, m_j) &= e_{0_j} \\ q(e_{i_j}, m_j) &= e_{i+1_j} \end{aligned} \tag{18}$$

Function q would, at each iteration, output a new set of irradiance coefficients based on the coefficients from the previous iteration and on the precomputed coefficients. However, at the moment, the existence of this function is only hypothetical. Therefore, part of the future work on DNRT will be spent on trying to give a concrete definition of function q.

9.2 Supporting Animating Objects

In its current form, DNRT only supports rigid objects. This means that objects can be scaled, rotated and translated within the environment, but cannot be deformed, nor animated. In the future, two strategies could be applied to overcome this limitation. First, the adoption of a partitioning approach, in which a dedicated sampling volume is associated to each of the object's sub-meshes, should constitute a simple extension that may enable rigid animations. For actual deformations, or more complex animations, a promising solution is to generate a set of Texture3D volumes for each

key-frame of the object's animation. At run-time, for each frame of animation, the radiance transfer coefficients would be obtained by interpolating between the Texture3D volumes of the two closest key-frames.

9.3 Supporting arbitrary normal directions

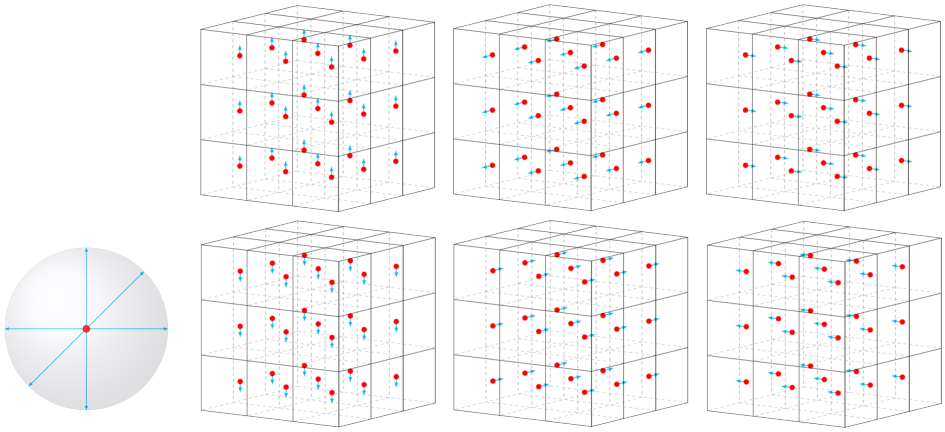


Figure 21. A set of normal directions $n_j \in N$ is preselected (left). For each normal n_j, a dedicated 3d sampling grid is built (right).

As seen in Subsection 4.1.4, DNRT employs during precomputation an intermediate normal direction n_o. This enables DNRT to use a vector representation for its radiance transfer coefficients, differently from other PRT-based techniques [Sloan et al. 2002], which instead utilize a matrix representation. Using a vector representation has numerous advantages: it lowers the storage requirements and the amount of data to be fetched from (texture) memory during the shader execution; it allows to apply the radiance transfer operation using a simple dot product, rather than a matrix multiplication. However, the use of an intermediate normal direction can have a negative impact on image fidelity, especially for situations like the one depicted in Figure 6. To solve this problem, in the future, two different approaches will be examined.

The first approach is to go back to a more classic matrix representation of the radiance transfer coefficients, using the derivation provided in Appendix A. However, eventual improvements in image quality may be out-weighted by the loss in performance and by the higher storage requirements, making the current version of DNRT still preferable.

An alternative approach is to extend the current DNRT implementation to support a predetermined set of normal directions $n_j \in N$ (see Figure 21). A precomputation iteration should be executed for each n_j, resulting in multiple sets of 7 Texture3D volumes. During the shader execution, the set whose associated normal n_j resembles

more closely the fragment's normal is selected. Except for this initial selection process, the shader would remain identical to the current implementation. This should minimize any loss in performance, while yielding an improvement in image quality. Storage would, however, be negatively affected, growing linearly with the cardinality of N.

9.4 Improved Solutions to the Intrusion Problem

In its current incarnation, DNRT provides a solution to the intruding object problem (see Subsection 5.2.3) based on omnidirectional shadowmapping. This solution is, however, lacking both in terms of performance and quality. In the future, more complex shadowing techniques should be examined. The use of methods such as Percentange-Close Soft Shadows [Fernando 2005] and Imperfect Shadowmaps [Ritschel et al. 2008] will likely bring substantial improvements in regards to both quality and performance. Even better results may be yielded by the more experimental technology of Precomputed Shadow Fields [Zhou et al. 2005].

References

BAVOIL, L., SAINZ, M., AND DIMITROV, R. 2008. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks*, ACM, 22. 5, 28

BERILLO, A. 1997. S3tc and fxt1 texture compression. *Copyright by Digit-Life. com 2003*, 21. 52

CHEN, S. E. 1990. Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system. *ACM SIGGRAPH Computer Graphics 24*, 4, 135–144. 6

COHEN, M. F., AND GREENBERG, D. P. 1985. The hemi-cube: A radiosity solution for complex environments. In *ACM SIGGRAPH Computer Graphics*, vol. 19, ACM, 31–40. 6

COHEN, M. F., CHEN, S. E., WALLACE, J. R., AND GREENBERG, D. P. 1988. A progressive refinement approach to fast radiosity image generation. *ACM SIGGRAPH Computer Graphics 22*, 4, 75–84. 6

CORNELL, 2016. URL: http://www.graphics.cornell.edu/online/box/data.html. 25

CRASSIN, C., NEYRET, F., SAINZ, M., GREEN, S., AND EISEMANN, E. 2011. Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum*, vol. 30, Wiley Online Library, 1921–1930. 4, 6, 7, 28, 41

CRYTEK, 2016. URL: http://www.crytek.com/cryengine/cryengine3/downloads. 25

DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM, 203–231. 4

DACHSBACHER, C., STAMMINGER, M., DRETTAKIS, G., AND DURAND, F. 2007. Implicit visibility and antiradiance for interactive global illumination. In *ACM Transactions on Graphics (TOG)*, vol. 26, ACM, 61. 6, 16

EVERITT, C. 2001. Interactive order-independent transparency. *White paper, nVIDIA 2*, 6, 7. 5

FERNANDO, R. 2005. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, ACM, 35. 25, 47

GEOMERICS, 2016. URL: http://www.geomerics.com/enlighten/. 7, 8

GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. 1984. Modeling the interaction of light between diffuse surfaces. In *ACM SIGGRAPH Computer Graphics*, vol. 18, ACM, 213–222. 3, 6

GREEN, R. 2003. Spherical harmonic lighting: The gritty details. In *Archives of the Game Developers Conference*, vol. 56. 6, 12, 22

GREGER, G., SHIRLEY, P., HUBBARD, P. M., AND GREENBERG, D. P. 1998. The irradiance volume. *IEEE Computer Graphics and Applications 18*, 2, 32–43. 7, 8

IWASAKI, K., DOBASHI, Y., YOSHIMOTO, F., AND NISHITA, T. 2007. Precomputed radiance transfer for dynamic scenes taking into account light interreflection. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, Eurographics Association, 35–44. 5

JENSEN, H. W. 1996. Global illumination using photon maps. In *Rendering Techniques 96*. Springer, 21–30. 3

KAJIYA, J. T. 1986. The rendering equation. In *ACM Siggraph Computer Graphics*, vol. 20, ACM, 143–150. 11

KAPLANYAN, A., AND DACHSBACHER, C. 2010. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, ACM, 99–107. 4, 6, 7, 28, 41

KARRAS, T. 2012. Thinking parallel, part iii: tree construction on the gpu. *Availavle: http://devblogs. nvidia. com/parallelforall/thinkingparallel-part-iii-tree-construction-gpu*. 20

KELLER, A. 1997. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 49–56. 4

KLETT, WITWE, E., DETLEFFSEN, PETER, C., ET AL. 1760. *IH Lambert... Photometria sive de mensura et gradibus luminis, colorum et umbrae*. sumptibus viduae Eberhardi Klett. 14

LEHTINEN, J., AND KAUTZ, J. 2003. Matrix radiance transfer. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, ACM, 59–64. 5, 44

MITTRING, M. 2007. Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses*, ACM, 97–121. 5

PAN, M., WANG XINGUO LIU, R., PENG, Q., AND BAO, H. 2007. Precomputed radiance transfer field for rendering interreflections in dynamic scenes. In *Computer Graphics Forum*, vol. 26, Wiley Online Library, 485–493. 5, 9, 44

RAMAMOORTHI, R., AND HANRAHAN, P. 2001. An efficient representation for irradiance environment maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, 497–500. 12, 45, 52

RITSCHEL, T., GROSCH, T., KIM, M. H., SEIDEL, H.-P., DACHSBACHER, C., AND KAUTZ, J. 2008. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics (TOG) 27*, 5, 129. 25, 47

RITSCHEL, T., GROSCH, T., AND SEIDEL, H.-P. 2009. Approximating dynamic global illumination in image space. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, ACM, 75–82. 5

ROBERT, C. P., AND CASELLA, G. 1999. Monte carlo integration. In *Monte Carlo Statistical Methods*. Springer, 71–138. 20

SHANMUGAM, P., AND ARIKAN, O. 2007. Hardware accelerated ambient occlusion techniques on gpus. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM, 73–80. 5, 28

SIDEFX, 2016. URL: https://www.sidefx.com/docs/houdini/nodes/out/ifd. 2

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Transactions on Graphics (TOG)*, vol. 21, ACM, 527–536. 4, 5, 9, 10, 13, 44, 46

SLOAN, P.-P. 2008. Stupid spherical harmonics (sh) tricks. In *Game developers conference*, vol. 9. 51

SLOMP, M. P. B., OLIVEIRA, M. M., AND PATRÍCIO, D. I. 2006. A gentle introduction to precomputed radiance transfer. *RITA 13*, 2, 131–160. 4

UNITY, 2016. URL: https://www.assetstore.unity3d.com/en/#!/content/33630. 25, 27

VAN DE HOEF, M. 2013. Real-time dynamic radiosity for high quality global illumination. Unpublished, 65. URL: http://www.marries.nl/wordpress/wp-content/uploads/2014/10/MarriesvandeHoefMasterThesis1.0.pdf. 6, 7

WHITTED, T. 1979. An improved illumination model for shaded display. In *ACM SIGGRAPH Computer Graphics*, vol. 13, ACM, 14. 3, 20

WILLIAMS, L. 1978. Casting curved shadows on curved surfaces. In *ACM Siggraph Computer Graphics*, vol. 12, ACM, 270–274. 3

ZHOU, K., HU, Y., LIN, S., GUO, B., AND SHUM, H.-Y. 2005. Precomputed shadow fields for dynamic scenes. In *ACM Transactions on Graphics (TOG)*, vol. 24, ACM, 1196–1201. 47

Appendix A – Arbitrary Normals and Convolution

This appendix provides an additional mathematical derivation that shows how to precompute L_{refl} without the use of an intermediate normal vector n_o. The starting point for this new derivation is Equation 5, here repeated for convenience:

$$L_{refl}(p_i) = \rho_{r_i} \sum_{j=1}^{n} c_j \int_D \frac{Y_j(n_{s_i})\rho_{s_i}}{\pi^2} (\omega_{s_i} \cdot n_{r_i})_+ d\omega_{s_i} \tag{5}$$

The contents of the integrals in Equation 5 are grouped under the following functions:

$$g(\omega_{s_i}) = (\omega_{s_i} \cdot n_{r_i})_+ \tag{19}$$

$$f_j(n_{s_i}) = \frac{Y_j(n_{s_i})\rho_{s_i}}{\pi^2} \tag{20}$$

Notice that n_{r_i} is not part of g's arguments. The reason behind this is that n_{r_i} is initially treated as a constant with value $+y$, i.e. is the vector pointing upwards. This assumption is only temporary as the Spherical Harmonics convolution property will allow to account for all other possible directions of n_{r_i}.

Function g is then projected onto the frequency domain:

$$g(\omega) = \sum_{t=1}^{r} g_t Y_t(\omega) \tag{21}$$

Each distinct function f_j is also projected into the frequency domain:

$$f_j(\omega) = \sum_{t=1}^{r} f_{jt} Y_t(\omega) \tag{22}$$

At run-time function g is used as a convolution kernel upon each function f_j. The result is a new composite function that is evaluated in direction n_{r_i}. Since g is a circular symmetric function, the convolution operation resolves into a simple application of the convolution theorem [Sloan 2008]:

$$(f_j \star g)_l^m = \sqrt{\frac{4\pi}{2l+1}} g_l^0 f_{jl}^m \tag{23}$$

Notice that the general theorem uses a double index notation, which has been avoided so far in order for simplicity purposes. Equation 23 can be rewritten in single index notation by using, in regards to function g, the index t' as a replacement for the combination $l = k, m = 0$, with $k \in [0, \sqrt{n}]$ (the $\star$ symbol indicates function composition):

$$(f_j \star g)_t = \sqrt{\frac{4\pi}{2l+1}} g_{t'} f_{jt} \tag{24}$$

Finally, the definition from Equation 24 is used in Equation 5 to obtain the following:

$$L_{refl}(p_i) = \rho_{r_i} \sum_{j=1}^{n} c_j \sum_{t=1}^{r} \sqrt{\frac{4\pi}{2l+1}} g_{t'} f_{jt} Y_t(n_{r_i}) \tag{25}$$

Using Equation 25 a higher accuracy in DNRT's output should be possible. However, the necessity to deal with a matrix rather than a vector of coefficients makes this approach inherently slower and perhaps less preferable.

Appendix B – Additional Future Work

In Section 9 the most critical ways to improve DNRT in the future have been investigated. However, there are many other smaller adjustments that could help raising DNRT's quality and efficiency.

B.1 Quadratic Samples Distribution

DNRT currently uses a regular grid to build a set of samples p_i around each dynamic object O. This means that the points p_i are distributed linearly in the space around O. However, the influence of a dynamic object on its surrounding is not linear, but rather decreases proportionally to the squared distance from the object itself. To better capture this behavior, a quadratic distribution, in which the sampling set is denser in the object's vicinity and more sparse at growing distances, could be used. This may improve DNRT's visual fidelity. It may also enable the coverage of a wider spacial range around the object, while using the same number of samples.

B.2 Improved Anti-Radiance

In the current DNRT implementation, anti-radiance is represented using the irradiance function at the occluder's back-face. As illustrated by Figure 7, this leads can lead to a substantial approximation error. Using the radiance at the occluder's back-face may grant better results. This would, however, yield the problem of obtaining the SH radiance coefficient from the irradiance coefficients c_j. A possible solution could be based on employing an opposite process to the one described in [Ramamoorthi and Hanrahan 2001].

B.3 PRT Preliminary Step

Rather than directly sampling the dynamic object's albedo during precomputation, better results may be achieved by first applying PRT on the model. This would enrich the object with local effects like self-shadowing and self-reflections. Once these effects are applied, sampling would be executed.

B.4 Texture Compression

While DNRT employs already a certain degree of compression to store the radiance coefficients in RGBA32 and RGB24 textures (see Subsection 5.1.2), memory requirements could be further reduced using texture compression methods such as S3 [Berillo 1997]. This, however, may also introduce visual artifacts.

Appendix C – Screenshots

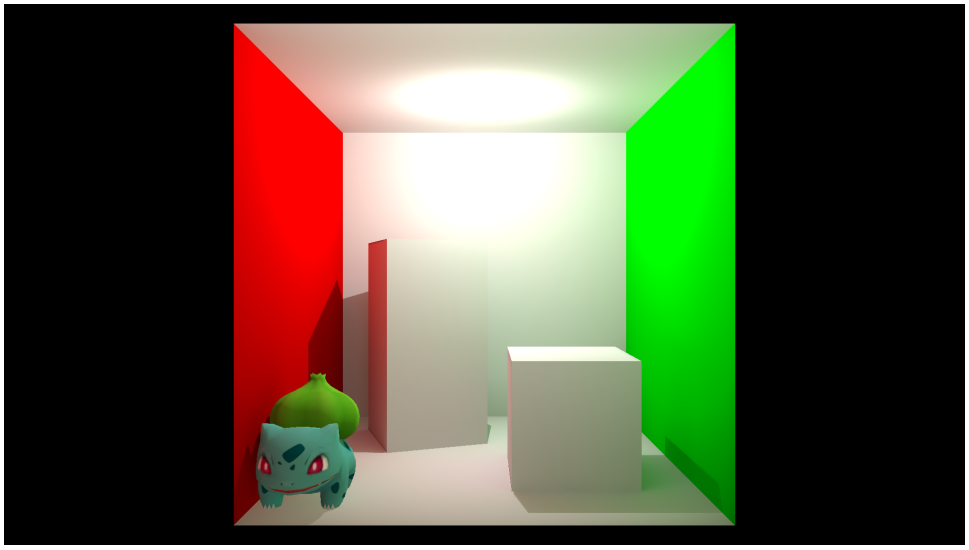


Figure 22. DNRT 1 dynamic objects

Figure 23. DNRT 5 dynamic objects