Fast Global Illumination Approximations on Deep G-Buffers

Michael Mara[1] Morgan McGuire[1] Derek Nowrouzezahrai[2] David Luebke[1]

[1]NVIDIA [2]University of Montreal

Direct + Ambient | Direct + (1-AO) × Ambient + Radiosity + Mirror Rays

Figure 1: *Left:* Direct and hemispherical ambient illumination in *San Miguel* (6.5M triangles, 968 draw calls). *Right:* Direct lighting, approximate radiosity, mirror reflections, and AO computed from a two-layer deep G-buffer in 5 ms at 1080p on NVIDIA GeForce Titan. The G-buffer was generated in a single 30 ms geometry pass. See our evaluation section for faster results on more game-like scenes.

Abstract

Deep Geometry Buffers (G-buffers) combine the fine-scale and efficiency of screen-space data with much of the robustness of voxels. We introduce a new hardware-aware method for computing two-layer deep G-buffers and show how to produce dynamic indirect radiosity, ambient occlusion (AO), and mirror reflection from them in real-time. Our illumination computation approaches the performance of today's screen-space AO-only rendering passes on current GPUs and far exceeds their quality. Our G-buffer generation method is order-independent, guarantees a minimum separation between layers, operates in a (small) bounded memory footprint, and avoids any sorting. Moreover, to address the increasingly expensive cost of pre-rasterization computations, our approach requires only a single pass over the scene geometry. We show how to apply Monte Carlo sampling and reconstruction to these to efficiently compute global illumination terms from the deep G-buffers.

The resulting illumination captures small-scale detail and dynamic illumination effects and is substantially more robust than screen space estimates. It is necessarily still view-dependent and lower-quality than offline rendering. However, it is real-time, temporally coherent, and plausible based on visible geometry. Furthermore, the lighting algorithms automatically identify undersampled areas to fill from broad-scale or precomputed illumination. All techniques described are both practical today for real-time rendering and designed to scale with near-future hardware architecture and content trends. We include pseudocode for deep G-buffer generation, and source code and a demo for the global illumination sampling and filtering.

This is the sRGB version of the paper. A "brighter" gamma version is in our supplement. We recommend on-screen viewing with the version that most clearly differentiates the images in figure 1.

1. Introduction

Screen-space illumination methods are widely used today for real-time rendering. For example, screen-space ambient occlusion (AO) enjoys pervasive application because it strongly impacts image quality, maps well to GPU architectures, and is a very fast approximation. This popularity comes despite its well-known shortcomings for underestimation and view dependence. Meanwhile, voxel-based approaches for illumination show great promise but have not been widely deployed because of scalability and fidelity concerns. We observe that many of the benefits of screen-space and voxels can be combined. In this paper we refine several different ideas from the literature into a **practical**, **robust**, and **real-time** lighting solution. Those three constraints are essential for the games industry. So, we follow the successful example of screen-space AO and ensure them by relaxing ra-

diometric accuracy. Note that a radiometrically-accurate but inefficient technique would not have "less error" from the perspective of a developer or player, since missing a frame at 30 Hz is a grave animation error compared to undersampling indirect light. We characterize the nature of the sampling error introduced and sketch how to reduce it when more computational performance is available.

From an implementer's viewpoint, in this paper we specifically show how to extend today's popular-but-fragile, screen-space AO-only passes to robust AO computation, and even into indirect illumination without significantly increasing the cost over single-layer AO alone. We achieve this through careful compression, cache management, sampling, and reconstruction, aided by a new kind of data structure: the deep geometry buffer with minimum separation (deep G-buffer), generated efficiently in a single pass over geometry. We intentionally developed this illumination solution as an extension of existing G-buffer generation and illumination passes. Working with existing rendering passes reduces the software engineering cost of integration with existing engines, and replacing the AO-only pass with a general indirect lighting pass in roughly the same performance profile is a natural evolution for game engines.

We consider deep G-buffer generation separately from the illumination algorithms that use it because the G-buffer has more applications than indirect lighting (e.g., direct illumination, depth of field, motion blur, reprojection) in a rendering engine. The two-layer deep G-buffer generation method that we present is more expensive than a single-layer method on today's content and GPUs, although it is usually faster than previous methods like depth peeling. We optimized it on current GPUs, but specifically designed it to target and scale with near-future applications and GPUs. We observe that games have increasing pre-rasterization GPU workloads, such as tessellation and skinning, that make multiple passes over source geometry prohibitively expensive. We also observe that while the first-generation architectures with geometry shader units were not particularly efficient, geometry shader throughput is increasing in recent architectures and there is no inherent reason that it should incur any performance penalty in the future. To follow these trends, the deep G-buffer generation method limits its access to geometry to a single pass and relies on a geometry shader to perform multi-layer rendering.

Our motivation to compute robust, dynamic indirect illumination quickly is simple to understand. The implementation of the techniques that we describe for it is also simple, at least once seen. We provide pseudocode, C++ and GLSL source code, and a demonstration application for an optimized implementation of radiosity to ensure that implementing it is truly simple and unambiguous. However, deriving that solution, analyzing its quality, and explaining the constraints for efficient execution are more complex tasks. That is because the exposition requires moving through a set of interlocking hardware details, algorithms, and mathematics, e.g., from bandwidth management and architecture trends to quasi Monte Carlo estimators. We considered presenting the AO, radiosity, reflection ray casting, temporal filtering, and single-pass data structure generation ideas as separate articles, but discovered that the overlap and motivation greatly benefitted from a single, comprehensive presentation.

Many computer science problems are best solved by a hybrid of broad- and fine-scale algorithms. In computer networking this leads to the "last mile/link/kilometer" design approach, that is, tree leaves are often handled separately from internal nodes. Likewise in sorting algorithms, a radix or quick sort will often arrange data on a broad scale for an insertion sort to handle at a small one. Some examples of this principle in computer graphics are:

Domain	*Broad-Scale Solutions*	*Fine-Scale Solutions*
Visibility	occlusion culling, depth sorting, frustum culling	z-buffer
Geometry	geometry, subdivision surfaces, displacement map	bump map, normal map
Materials	different scattering (BSDF) models	texture maps of coefficients
Lighting	baked light map, baked light probe, irradiance volume, sparse voxel octtree	screen-space, **deep G-buffer**

The deep G-buffer lighting terms such as the AO, radiosity, and ray tracing that we present are intended to provide fine detail and address dynamic content, but are also *intended to be paired with a fallback broad-scale solution*. We use a currently-popular solution of static light probes as the fallback in our results. We believe that despite increasing geometry and material complexity in modern content, normal maps and texture maps will always be a part of computer graphics, because it is simply infeasible to scale geometry and materials to the fine scale of pixel level and below. Likewise, we suspect that real-time global illumination methods will increase in scope and efficiency at the broad scale, but the illumination techniques presented in this paper will long remain useful for the "last mile" of fine-scale lighting.

1.1. Contributions

As we describe in the following section, this paper builds on many existing ideas. For example, many previous research papers have experimented with using multiple views or layers to improve screen-space effects [SA07, RGS09, VPG13, DS05, Hac05, BS09a, BS09b].

The novel value of our work is in bringing these many ideas together carefully and comprehensively, and then mapping them to the constraints of hardware architecture. We

contribute a theory for working with deep G-buffers, analysis of their use in a renderer, and practical details necessary for deploying them. Specifically, this paper contributes:

1. A method for generating a deep G-buffer in a single pass over geometry on a GPU, designed to scale for performance on near-future GPUs and content (sec. 2.3).
2. An adaptation of Scalable Ambient Obscurance for deep G-buffers (sec 3.1)
3. A robust, physically-motivated radiosity algorithm for deep G-buffers (sec. 3.2).
4. Camera-space Quasi Monte Carlo sampling (sec. 3.4).
5. Details of screen-space ray tracing (sec. 3.6).
6. Quantitative performance analysis (sec. 4).
7. Qualitative analysis of the major sources of error in our radiosity approximation (sec. 4).
8. Images, video, and a demo for quality evaluation; these are strictly better than previous single-layer AO, and often subjectively *much* better (sec. 4 and supplement).
9. Source code for an optimized implementation of deep G-buffer radiosity, AO, and temporal filtering (supplement).

1.2. Related Work

Many have observed that multiple images increase the robustness of screen-space methods. This paper focusses on multiple layers from a single camera because doing so grants three essential advantages:

1. **Practicality**: leverages the same regular parameterization as voxels to simplify sampling and reconstruction;
2. **Perception**: ensure that visible and nearly-visible surfaces are sampled well, so frames are self-consistent; and
3. **Performance**: amortize pre-rasterization operations such as occlusion culling, tessellation, and skinning.

Generating Layers. Several approaches can render multiple geometry layers from a single view. In order of decreasing memory footprint: clip-space voxelization [Sch12, CG12], F- and A-buffers [MP01, Car84], ZZ-buffers [SS89], *k*-buffers and other bounded A-buffer approximations [LV00, MB07, BCL*07, SML11, Sal13], frequency A-buffer approximations [YK07, SA09, JB10], and depth peeling [Eve01, BM08]. Of these, *depth peeling* is particularly interesting in the context of effects that benefit most from a small number (two or three) depth layers since it has the smallest memory footprint: prior work shows that the quality and robustness of screen-space global illumination significantly benefits from even one additional layer [SA07, RGS09, VPG13].

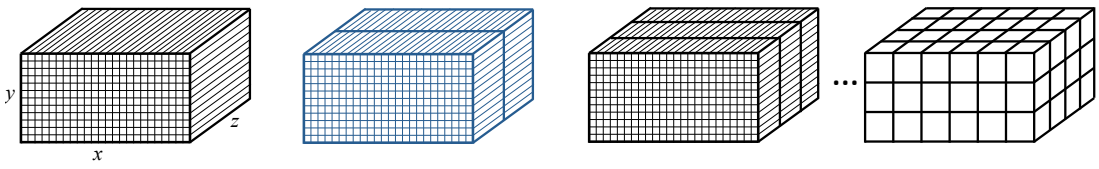
State-of-the-art approaches for computing the second-closest surface using a single depth peel require either two passes over the scene geometry [BM08] or a single pass with programmable blending [Sal13]. Neither guarantees a minimum separation. Our higher-performance and order-independent solution requires only a single geometry pass, no programmable blending, and works in bounded memory.

Indirect Light. Our shading methods are most directly related to directional occlusion [RGS09], Vardis et al.'s AO variant [VPG13] and Bavoil and Sainz multi-layer horizon-based AO approach [BS09a, BS09b]. The former two approaches use multiple views (and note their performance drawbacks), the latter uses a two-layer depth buffer without minimum separation; we extend these approaches to multiple layers with minimum separation and show how to apply these approximations to arbitrary indirect bounces and specular reflection. Our entire shading approach is incorporated atop a scalable gathering framework [MML12] and bears some similarities to previous image-space gathering techniques [DS05, SHRH09, NRS14].

2. Deep G-Buffers with Minimum Separation in 1 Pass

Motivation for Multiple Layers. A traditional, *single-layer G-buffer* [DWS*88, ST90] and a camera frustum *voxelization* store the same data and parameterize the same space as a regular grid. Therefore, they are the same class of data structure, representing the extremes of the continuum pictured in figure 2. Traditional G-buffers have high *xy*-resolution and the lowest possible *z*-resolution; they store a single voxel for each pixel. In contrast, a camera-space voxelization has uniform *xyz*-resolution, measured in either homogeneous (e.g., [ED06]) or world (e.g., [CNLE09]) space.

Between these extremes are what we call *deep G-buffers*. This data structure parameterization has been in use for a few years in niche applications [Per07, Cha11, NRS14]. It is a generalization of layered depth images [SGHS98, PLAN98, Eve01, MB07] to layering a full G-buffer for shading purposes. We extend the previous work with important con-



a) 1-Layer G-Buffer "Screen space" **b) 2-Layer Deep G-Buffer** c) 3-Layer Deep G-Buffer d) Camera-Space Voxels

Figure 2: A continuum of data structures on the content of the homogeneous clip-space view frustum. Traditional G-buffers (a) have high *xy*-resolution, minimal *z*-resolution, and choose the closest surface to fill each voxel. Traditional voxels (d) have medium *xyz*-resolution and average surface properties within each voxel. This paper extensively analyzes intermediate case (b).

straints on the layers and an efficient method for generating them.

Traditional, single-layer G-buffer pixels and voxels both store a regularly-sampled grid of surface point properties suitable for light transport (i.e., shading), such as position, surface normal, and reflectance parameters. In the simplest case, these may be simply a screen-space depth buffer and binary voxelization, from which position and normal (by gradient) are inferred. The generation methods may differ in how they select the value for each element. G-buffer generation typically chooses each pixel's properties as those of the surface closest to the camera that passes through the *xy* center. Some voxel generation methods set the voxel properties from the surface closest to the *xyz* voxel center [LK10], while others average the properties of all surfaces within the voxel [CNLE09]. We extend these generation strategies with a new minimum separation selection method for deep G-buffers.

Both data structures have the advantage of decoupling the cost of illumination computation from geometric complexity. G-buffers are widely used in the games industry for screen-space effects such as screen-space AO and reflections. They have been successful there despite the fragility of computing those effects from a single layer because they leverage the data already computed for deferred [DWS*88] or forward+ [HMY12] shading and capture fine detail. Real-time research has shown that voxels-based algorithms are more stable for broad-scale illumination, but they have not yet been adopted widely in the industry because they do not scale well and require entirely new data and method for generating them that, unlike G-buffer pixels, are not already present in the system for direct illumination or other effects.

Modern rasterization is designed for visible surface determination and "local" shading operations. When "global" scene information is required for shading, rasterizing multiple views or layers can help. Shadow mapping [Wil78] is perhaps the earliest example of this, where rasterized depth from the light's viewpoint is used for shadowing in the camera's view. Reflective shadow maps [DS05] and orthographic depth-peeling [Hac05] extend this idea to more complex shading; other methods have shown that screen-space methods can be made more robust using many views [SA07, RGS09, VPG13].

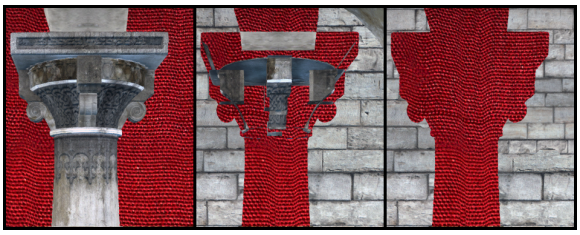
Motivation for Single-Pass Generation. Many images provide more data, but generating many images from different viewpoints is expensive. Doing so requires processing geometry that is not visible to the primary camera, thus increasing the cost of submitting geometry for rasterization rather than amortizing the cost of a single submission over multiple layers. Furthermore, the most important geometry for affecting perception of the final image is often that which is visible, or nearly visible, to the viewer. A separate view from, for example, the light's viewpoint, may capture significant amounts of information that do not directly impact this image...or that impact the image, but if the viewer was not aware of that geometry because it is not visible, might not notice the absence of its impact compared to undersampling of visible geometry. This is why Bavoil and Sainz [BS09a, BS09b] restricted their multiview input for ambient occlusion estimation to near the view frustum, producing a depth-only deep G-buffer. However, using multiple layers of a single view alone is insufficient. Under traditional methods for generating layers, the vertex processing overhead is still incurred for each layer and the total cost is linear in the number of layers. So, it is not sufficient to reduce the desired views to multiple layers of a single camera: we must also produce those layers in a single pass over the scene geometry.

To quantify the penalty for generating multiple views or multiple layers of a scene, we asked three developers of high-performance commercial game engines to profile their renderers. They found that between one sixth and one third of the rendering time for a frame is spent on tasks that occur *prior* to rasterization in the graphics pipeline. Such tasks include scene graph traversal, frustum and occlusion culling, tessellation, displacement mapping, procedural geometry generation, skeletal animation, and transformation [Bra13, Buk13, McG13].This means that even in the limiting case of writing a one-pixel G-buffer with no rasterization or pixel processing cost at all, processing the scene geometry twice to generate two different views incurs significant cost. Furthermore, the pre-rasterization cost of the graphics pipeline has been increasing as culling and geometry processing becomes more sophisticated and more content is animated.

Motivation for Minimum Separation. We observe that, in practice, the second-closest surface to the camera often fails to capture the most useful information for shading purposes. Decals, non-convex geometry, and fine detail often create local structure that occlude the most useful secondary surface. For example, traditional depth peeling in *Sponza* (figure 3b) reveals the second fold of the column's molding and not (figure 3c) the full red tapestry behind the column.

To resolve this local structure problem, we enforce a *minimum separation* between layers. When generating the G-buffer, we select only those fragments that immediately accessible beyond a constant distance Δz past the primary visible surfaces.

Note that a k-buffer cannot resolve this problem in bounded memory, even with single-pass programmable blending variants [Sal13]. We need more than a $k = 2$ buffer to guarantee this minimum separation, since the goal is to produce two *specific* layers from a $k = \infty$ buffer, not the first two layers. That is, until all surfaces have been rasterized, each pixel has no way of knowing the minimum acceptable depth for the second layer, so all surface fragments must be stored. We now proceed to describe a set of algorithms to



a) Primary b) Traditional c) Minimum Separation

Figure 3: Traditional depth peeling provides little additional information in areas with local structure. Enforcing our new minimum separation captures the next significant surface.

Texture Format	*Contents*	
RG16	Screen-Space Velocity	
RG16	Normal (Oct32)	
RGBA8	Lambertian RGB Color	Unused
RGBA8	Glossy RGB Color	Glossy Exponent
Depth32F	Depth	
	← 32 bits →	

Table 1: Each layer of the G-buffer as produced by the generation pass, in 160 bits/pixel.

Game	*Year*	*Bits/pixel*	
Killzone 2	2009	128	[Val09]
StarCraft II	2010	192	[FM08]
Battlefield 3	2011	160	[Cof11]
Crysis 3	2013	96	[CRW13]
Ryse	2013	128	[Sch14]
inFAMOUS: Second Son	2014	328	[Ben14]
Destiny	2014	96	[TTV13]

Table 2: G-buffer sizes for some recent games.

robustly identify these important secondary surfaces with a small, bounded memory footprint.

2.1. G-buffer Format

Table 1 shows the format for one layer of the deep G-buffer that we used in our experiments. This is comparable in size to the G-buffers used in recent games described in table 2.

2.2. A Strawman Two-Pass Algorithm

Listing 1 outlines a multi-pass depth peeling algorithm to generate a deep G-buffer for frame t. Each frame buffer render target is a two-element texture array, a feature supported by current GPUs. We denote the depth buffer layers $Z_t[0]$ and $Z_t[1]$. The geometry shader applies the current transformation T_t to each triangle, encompassing/abstracting all model-view-projection and skinning transformations.

```
1  // 1st Pass
2  submit geometry with:
3    geometryShader(tri):
4      emit T_t(tri) to layer 0
5    pixelShader(x,y,z):
6      return S(x,y,z)
7
8  // 2nd Pass
9  submit geometry with:
10   geometryShader(tri):
11     emit T_t(tri) to layer 1
12   pixelShader(x,y,z):
13     if (z > Z_t[0][x,y] + Δz): return S(x,y,z)
14     else: discard the fragment
```

Listing 1: A strawman *two-pass* deep G-buffer generator with minimum separation Δz using depth peeling. Our method improves significantly on this baseline.

For $\Delta z = 0$, this algorithm corresponds to traditional depth peeling [BM08] and, for $\Delta z > 0$, it guarantees minimum separation. The pixel shader applies an arbitrary shading function S: e.g., for G-buffer generation, S would simply output material properties. It is possible (and often preferable on present-day GPUs) to implement this algorithm using two separate frame buffers, without texture arrays and a geometry shader. Our algorithmic structure is chosen so as to make the analogy and notation clear in the following section.

2.3. Efficient Single-Pass Generation Algorithms

Listing 2 generates two layers with a minimum separation in a *single pass* over the geometry, by rendering to both layers simultaneously. To identify fragments in the second layer, we require an oracle to predict the depth buffer's first layer *before* that buffer has been rendered. We will describe four variants of our algorithm, each corresponding to a different approximation of such an oracle.

Delay Variant. By adding one frame of latency so that the transformations for the next frame T_{t+1} are known at render time, we can perfectly predict the next frame's first depth layer. Frame t reads (line 22) from the oracle computed from the previous frame, and generates the oracle for frame $t+1$ (lines 4, and 25-26) to satisfy the induction. This variant gives perfect output but requires one frame of latency; in certain cases (e.g., triple buffering) such latency may already be present but, typically, we would like to avoid it.

Previous Variant. By simply using the previous frame's first depth layer as an approximate oracle, approximation error increases only as object and camera motion increase. This can be acceptable in some cases. First, errors will only appear in the second layer, not on visible surfaces. Second, the errors are only in the minimum separation: the second

```
submit geometry with:
  geometryShader(tri)
    emit T_t(tri) to layer 0
    emit T_t(tri) to layer 1
    if (VARIANT == Delay) || (VARIANT == Predict):
      emit T_{t+1}(tri) to layer 2

  pixelShader(x,y,z):
    switch (layer):
    case 0: // 1st layer; usual G-buffer pass
      return S(x,y,z)

    case 1: // 2nd G-buffer layer: choose the comparison texel
      if (VARIANT == Delay) || (VARIANT == Predict):
        L = 2 // Comparison layer
        C = (x,y,z) // Comparison texel
      else if (VARIANT == Previous):
        L = 0; C = (x,y,z)
      else if (VARIANT == Reproject):
        L = 0; C = (x_{t-1}, y_{t-1}, z_{t-1})

      if (z_C > Z_{t-1}[L][x_C, y_C] + Δz): return S(x,y,z)
      else: discard the fragment

    case 2: // Depth only write to predict Z_{t+1}[0]; no shading
      return // We only reach this case for Delay and Predict
```

Listing 2: Our new, efficient *single-pass* deep G-buffer generator with minimum separation Δz.

layer still captures only surfaces at the correct positions at time t. Third, there will only be errors in final moving objects, and we note that motion overrides perception of precise intensities and even shape [SA11].

Predict Variant. We can predict T_{t+1} using velocities from any underlying physics/animation simulation, or extrapolation from vertices at $t-1$ and t. When velocity prediction is accurate, this variant yields perfect results (equivalent to Delay), but **without** latency. When it is inaccurate, the same arguments that hold for the Previous variant apply here.

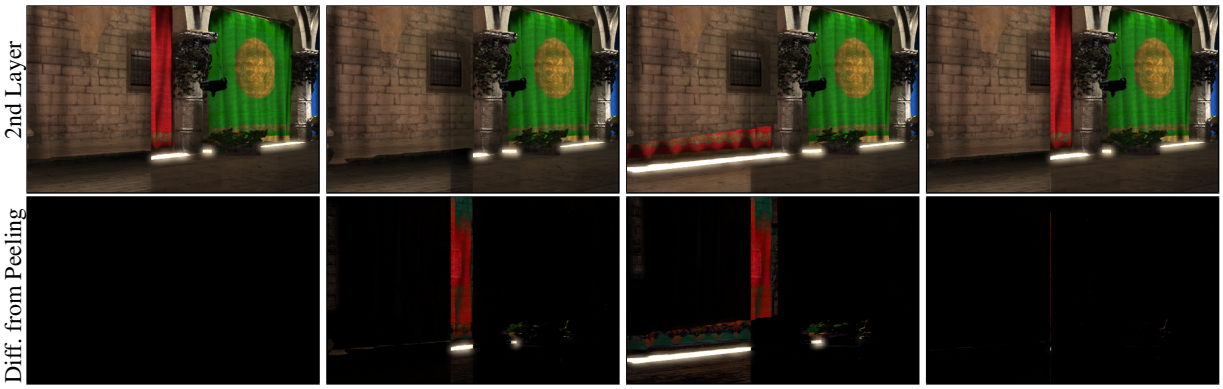
Reproject Variant. Here, we apply *reverse reprojection* [NSL*07] to perform a minimum separation test against the first depth layer from frame $t-1$: we use vertex positions from $t-1$ to compute the screen coordinates and depth C for the visibility test. Note that old depth values are not warped forward: instead *visibility* is computed in the "past". This is susceptible to errors around moving objects, but less so than Predict because it can use perfect hindsight velocities from $t-1$. Note that many techniques require such velocities for use e.g. in screen-space motion blur and antialiasing.

Figure 4 (top) compares the second layer surfaces obtained from each variant, with fast camera motion in *Sponza*; Figure 4 (bottom) compares to ground truth minimum separation. Previous and Predict can produce large errors. Reproject limits errors to tight regions around silhouettes and adds no latency, we identify it as our principal solution.

3. Three Applications to Global Illumination

Several applications can benefit from layered deep G-buffers, including stereo image reprojection, depth of field, transparency, motion blur, and global illumination. We focus on the latter.

We first extend screen-space AO to deep G-buffers in section 3.1, using AO to modulate a light probe. Despite the popularity of screen-space AO, extensions of it to screen-space radiosity have yet to enjoy similar adoption. We suspect this is due primarily to the artifacts present in single-layer screen-space solutions. To address this, we generalize our robust AO solution to a robust single-bounce radiosity method in section 3.2. Multi-bounce radiosity (section 3.3) is much more challenging as it requires more samples to reduce variance. We extend our method with multiple bounces and then add temporal smoothing and reverse reprojection to amortize the additional computation, reducing the cost back to that of computing a single bounce per frame. We note computing radiosity with a deep G-buffer is similar to Reflective Shadow Maps [DS05]. The main differences are that,



(a) Delay (b) Previous (c) Predict (d) Reproject

Figure 4: Top: Second-layer surfaces captured by variants of Listing 2 with a moving camera in *Sponza*. Bottom: Differences from ground truth produced by Listing 1. Delay is perfect but has latency; Reproject is nearly as good and adds no latency.

by operating exclusively in camera space, we can amortize the cost using work already performed in a deferred-shading pipeline, allowing us to simulate higher-order effects involving objects visible to the viewer but not to the light.

Finally, we investigate deep G-buffer mirror reflection tracing in section 3.6. As future work, we plan to investigate glossy reflections by either modifying the reflection rays to use pre-filtered incident lighting (computed on each layer of the deep G-buffer) or by modifying the BSDF in our radiosity algorithm, depending on the width of the glossy lobe.

3.1. Ambient Occlusion

We extend Scalable Ambient Obscurance [MML12] (SAO) to leverage our layered deep G-buffer and devise a new sampling scheme to further improve its quality (section 3.4). The original algorithm compensates for undersampling behind primary surfaces (which dominates its sampling error) with a coarser, biased estimator. Our improvements produce a more plausible shading falloff, avoid view-dependent halos on moving objects, and reduce noise.

Ambient Visibility $(1-AO)$ at a view-space point X is:

$$AV(X) = \max\left(0, 1 - \sqrt{\frac{\pi}{N}\sum_{i=1}^{N}\max\left(0, A_{i,0}, A_{i,1}\right)}\right) \quad (1)$$

$$A_{i,j} = AO\left(X, R(Z[j], i)\right) \quad (2)$$

where N is the sample count , $R(Z,i)$ reconstructs the position of the i^{th} sample using depth buffer Z, and AO is the Ambient Occlusion at X due to a sample at Y:

$$AO(X,Y) = \left(1 - \frac{\vec{v}\cdot\vec{v}}{r^2}\right)\cdot\max\left(\frac{\vec{v}\cdot\hat{n}_X - \beta}{\sqrt{\vec{v}\cdot\vec{v}+\varepsilon}}, 0\right), \quad (3)$$

where $\vec{v} = Y - X$, r is the sample pattern radius (see section 3.4), and $\hat{n}_X$ is the normal at X. This formulation corresponds roughly to SAO's AV with a union of occluders in both layers, but without any of the ad-hoc falloff terms.

Our improved sampling scheme (section 3.4) benefits from explicit normals, and we pack the camera-space Z and normal values for the two layers into a single texture each, as shown in table 3 (the radiosity inputs are unused for AO). For all applications, we employ a modified bilateral reconstruction that includes normal and plane weights to prevent blurring across surface discontinuities.

3.2. Single-Scattered Radiosity

Soler et al. [SHRH09] devised a screen-space radiosity approximation that we extend in a radiometrically correct fashion. Thereafter, we extend the technique to our deep G-buffer, and propose performance and aesthetically motivated modifications.

The total irradiance $E(X)$ incident at point X (with reflectivity ρ_X) due to the radiosity $B(Y)$ emitted from the closest point Y in direction $\hat{\omega}$ from X is [CG85, ICG86]

$$E(X) = \int_\Omega \frac{B(Y)}{\pi}\max(\hat{n}_X\cdot\hat{\omega}, 0)\, d\hat{\omega}\,. \quad (4)$$

We estimate this integral numerically as

$$E(X) \approx \frac{2\pi}{M}\sum_{\text{samples}} B(Y)\max(\hat{\omega}\cdot\hat{n}_X, 0), \quad (5)$$

where $\hat{\omega} = \frac{Y-X}{||Y-X||}$. The highest-quality version of our approximation samples N points Y from both G-buffer layers, but only uses the M for which both

$$(\hat{\omega}\cdot n_X) > 0 \text{ and} \quad (6)$$

$$(\hat{\omega}\cdot n_Y) < 0. \quad (7)$$

As for AO, we assume mutually visibility between X and Y. If the eqn. 7 test is omitted, then the bandwidth for the entire sampling process can be significantly reduced because n_Y need not be read for each sample. Eliminating that test increases error in the mean of the radiosity approximation, but by allowing more samples in less time it also reduces the variance. Thus, one can choose a solution that suppresses bias or one that suppresses noise. The incident irradiance at X scatters to outgoing radiosity as

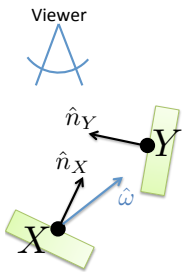
$$B(X) = E(X)\cdot\rho_X\cdot\text{boost}(\rho_X), \text{ where} \quad (8)$$

$\text{boost}(\rho) = 1$ conserves energy. We scale reflectivity by

$$\text{boost}(\rho) = \frac{\max_\lambda \rho[\lambda] - \min_\lambda \rho[\lambda]}{\max_\lambda \rho[\lambda]}, \quad (9)$$

where λ is the wavelength or "color channel". This amplifies scattering from saturated surfaces to enhance color bleeding, which helps visualize results and is also aesthetically desirable in entertainment applications (see [Hal10]). Indeed, we are motivated by the common practice in modern art training that promotes artificial exaggeration of color bleeding effects to enhance proximity cues and scene coherence.

The radiosity $B(Y)$ in the initial input is simply the Lambertian shading from (boosted) direct illumination. We iteratively re-apply equations 5 and 8 over multiple frames to simulate multiple bounces, as detailed in section 3.3.



In addition to the illumination value, the radiosity pass writes the confidence value M/N at each pixel, reflecting the fraction of samples affecting the result. At pixels where confidence is close to 1.0, many nearby points were discovered in the deep G-buffer that produced a reliable radiosity result (given our other simplifying assumptions). At pixels where the confidence is close to zero, most samples taken from the deep G-buffer were not representative of surfaces that could affect that pixel because they were backfacing, so the result is unreliable. During shading, we therefore linearly interpolate

Texture Format	*Contents*	
RGBA8	Layer 0 Normal n (Oct16)	Layer 1 Normal n (Oct16)
R11G11B10F	Layer 0 Previous Bounce Radiosity B	
R11G11B10F	Layer 1 Previous Bounce Radiosity B	
RG32F	Layer 0 Camera-space z	
	Layer 1 Camera-space z	
	← 32 bits →	

Table 3: Input to the radiosity algorithm, packed into 160 bits/pixel to minimize bandwidth and fetch instructions.

between a broad-scale or precomputed illumination solution and the dynamic deep G-buffer radiosity by the confidence score. In the results for this paper and the demo application, we used static radiance and irradiance probes, a common industry solution [MG12]. We chose this because it was the easiest to integrate. However, light maps, sparse voxel lighting, irradiance volumes, and per-vertex lighting are all viable alternatives.

The radiosity algorithm takes as input the deep G-buffer shown in table 1 and efficiently packed data from table 3. On modern GPUs, bandwidth is at a premium (both to DRAM and to cache) and packing data both optimizes the cache and amortizes the cost of issuing a texture fetch instruction and its execution. So, we pack the frequently sampled data into low precision and memory-adjacent locations. This includes camera-space depth (which, combined with camera information and texel location encode full position information) for both layers into a single buffer, and use the Oct16 encoding [CDE*14] to pack both layer's normals into the same RGBA8 buffer.

The Scalable Ambient Occlusion algorithm introduced a depth MIP-map computed by rotated-grid downsampling, which improves cache coherence when sampling over a large radius in screen space. We inherit that optimization.

3.3. Multiple-Scattered Radiosity

Multiple scattered radiosity requires N samples at each iteration and, in order to decouple render cost from the number of bounces, we incorporate information from previous frames in two ways: first, we only advance illumination by one bounce per frame using **progressive computation**; second, we apply **temporal filtering** by extending our bilateral reconstruction across time to pixels from the previous frame. In each case, we reverse-reproject sample locations to account for motion. This differs from the reverse-reprojection of depth in our oracle derivations in section 2.3, but has the same benefits and drawbacks.

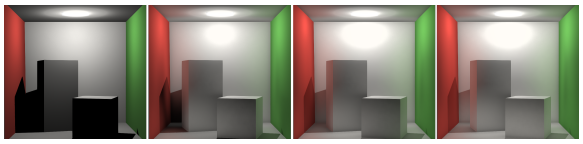
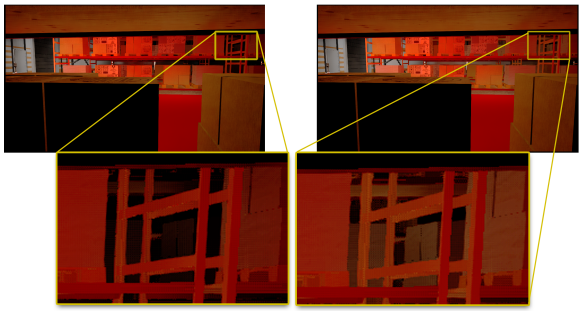


Figure 5: Direct light and radiosity after 1, 2, and 100 frames/bounces.



a) Propagating first layer b) Deep propagation

Figure 6: Gathering radiosity in *Warehouse* from two G-buffer layers but only (a) propagating within the first underestimates multibounce radiosity in areas of high depth complexity compared to (b) two-layer propagation.

a) Incremental reprojection error b) Variance from disocclusions

Figure 7: Temporal filtering artifacts under vertical camera movement in *Kitchen* at $\alpha = 0.98$, mitigated at $\alpha = 0.85$.

```
// tau[N-1] = optimal number of spiral turns for N samples
const int tau[ ] = {1, 1, 2, 3, 2, 5, 2, 3, 2, 3, 3, 5, 5,
3, 4, 7, 5, 5, 7, 9, 8, 5, 5, 7, 7, 7, 8, 5, 8, 11, 12, 7,
10, 13, 8, 11, 8, 7, 14, 11, 11, 13, 12, 13, 19, 17, 13,
11, 18, 19, 11, 11, 14, 17, 21, 15, 16, 17, 18, 13, 17,
11, 17, 19, 18, 25, 18, 19, 19, 29, 21, 19, 27, 31, 29, 21,
18, 17, 29, 31, 31, 23, 18, 25, 26, 25, 23, 19, 34, 19, 27,
21, 25, 39, 29, 17, 21, 27};
```

Listing 3: Discrepancy-minimizing number of turns τ.

Progressive Computation. We accumulate higher-order bounces by incorporating the previous frame's final indirect irradiance buffer E_{t-1} in Equation 8, thus simulating n bounces in n frames (figure 5). Reprojection avoids ghosting in the presence of dynamic objects, but light will still linger for many frames on a surface. To limit this effect, we damp the forward propagation of E_{t-1} by factor $0 < \delta \leq 1$. Damping intentionally underestimates illumination. We compensate for that bias by adding a small amount of environment lighting from a static light probe above the confidence value.

We also propagate radiosity between the two layers, which is important for multiple scattering in scenes with high depth complexity (Figure 6). The marginal cost of propagating in the second layer is negligible since it shares gathered samples from the first layer.

Temporal Filtering To smooth our undersampling noise, we compute an exponentially-weighted moving average $E_t = E(1-\alpha) + \text{reproject}(E_{t-1})\alpha$ and revert to $E_t = E$ for pixels where the reprojected point is not within 1cm of either layer (which is indicative of an incorrect velocity estimate). We recommend (and use) $\alpha = 0.85$, except where noted. If α is too large (e.g., 0.95), then dynamic lighting will experience latency and two kinds of artifacts may appear in each frame (Figure 7): despite detecting failed reprojections, ghosting can still appear from incrementally accumulated reprojection errors (each within the 1cm threshold), and rejecting too many samples due to reprojection disocclusion increases the variance at each pixel.

3.4. Quasi Monte Carlo Sampling

For our AO and radiosity solutions, we place N sample taps in a spiral of τ turns and radius r', similarly to McGuire et al. [MML12], however we optimize the pattern's parameters to minimize (2D) discrepancy [Shi91] for quasi-Monte Carlo (QMC) integration. We amortize computation over both layers by sampling the same points in each. The i^{th} sample at (x,y) is accessed from texel

$$(x,y) + h_i \hat{u}_i, \text{ where} \tag{10}$$

$$h_i = r'\zeta_i \tag{11}$$

$$\hat{u}_i = (\cos\theta_i, \sin\theta_i) \tag{12}$$

$$\theta_i = 2\pi\zeta_i\tau + \phi \tag{13}$$

$$\zeta_i = \tfrac{1}{N}(i + 0.5). \tag{14}$$

We rotate all samples by an azimuthal angle ϕ chosen according to a hash on (x,y), and the sample tap MIP level m_i is $m_i = \lfloor \log_2(h_i/q) \rfloor$. The constant q is the screen-space radius at which to first increment MIP levels, chosen based on the texture cache size.

We compute the optimal values of τ (to the nearest integer) that minimize discrepancy for each N offline and choose the appropriate value from listing 3 at run-time. McGuire et al. fixed $N = 9$ and found $\tau = 7$ optimal by manual tuning; however, the quality of their result degenerates if $\tau = 7$ is used for $N \neq 9$.

Figure 8 illustrates the impact of our optimized QMC sample placement. All three images take the same time to compute (for 99 AO samples). The left-most image has high discrepancy ($\tau = 7$) and exhibits banding since all $\phi = 0$. The center image varies ϕ, but the impact of discrepancy is still manifested as clusters of noise. Choosing the optimal τ yields a higher quality result (right).

3.5. Recommended Radiosity Parameters

There are five content-independent parameters for the radiosity method. These should be chosen based on the desired performance and image quality for the target hardware and application. We recommend the three parameter sets in table 4, which are supplied as presets in the demo. Increasing the number of raw samples N (from which the number of spiral turns, τ, is completely determined by listing 3) reduces sample variance. Increasing the number of spatial reconstruction filter taps reduces noise in the final image, but also blurs out high-frequency illumination. Including the n_Y test improves contrast and reduces bias. Raising the minimum MIP level when computing radiosity potentially increases variance of low-frequency terms, leading to large-scale flickering, but has a significant impact on performance because it affects cache coherence. The deep G-buffer input fills a guard band around the frame to help stabilize

Figure 8: AO from 99 spiral taps *(left)* without rotation and suboptimal τ, *(center)* unbiased in 2D via pattern rotation, and *(right)* with τ chosen by our screen-space QMC optimizer. Results shown without reconstruction to illustrate the variance reduction.

Preset	*Radiosity Samples (N)*	*Reconstruction Filter Taps*	*Use n_Y Test (eqn. 7)*	*Minimum MIP Level*	*Fraction of Guard Band Shaded*
High Performance	13	9	No	3	10%
Balanced	14	11	Yes	2	50%
High Quality	30	13	Yes	0	80%

Table 4: Three parameter sets for our radiosity algorithm.

the result under camera motion. The output can fill a more narrow guard band because it only contributes as the previous bounce's result. Thus one can increase performance at the expense of robustness for 2nd and higher order indirect light by reducing the fraction of the guard band extent within which radiosity computation occurs.

We tuned all parameter sets for NVIDIA GeForce 770, a mid-range, one generation old GPU. This is a reasonable target for games currently in development, since this represents what one might expect to be low-end bandwidth and processing rates by the time such games are completed. Recall that we expect the deep G-buffer generation process to be faster on future hardware, although we do not expect bandwidth to increase proportionally.

We tuned the *High Performance* to minimize evaluation time at the lowest image quality we found acceptable. It just barely suppresses flickering and noise artifacts and gives a heavily biased image, but is stable and fast. This what one might desire for a game with tight performance constraints, e.g., a high-speed first-person game like *Call of Duty*. We tuned *High Quality* to the point where increasing parameters gave minimal quality increase. *Balanced* is at the knee in our perception of the quality vs. performance curve. We recommend it for a less twitchy game such as *Portal* or *Skylanders*.

3.6. Reflection Ray Tracing

We adapt Sousa et al.'s method [SKS11] for screen-space mirror reflections, as well as an additional radiosity result in section 4, to use the deep G-buffer. We march reflection rays in camera space, projecting each point into both G-buffer layers: if the ray lies within $[z, z+\Delta z]$ of either point (x, y, z) at a pixel then we consider that a hit and set the outgoing radiance of that pixel as the incoming radiance along the reflection direction (see our supplemental code for our full implementation). After a maximum distance, or outside the guard band, we fall back to an environment map lookup.

4. Evaluation

We detail our experimental evaluation of single-pass **generation** of layered deep G-buffers with minimum separation, and their application to **global illumination** (GI) in the scenes from table 5. All results were measured at 1920×1080 on a NVIDIA GeForce Titan GPU.

Scene	*Source*	*Tris*	*Chrs*	*Mshs*
Office	g3d.sf.net	10k	0	17
Grass	turbosquid.com	180k	0	6
Kitchen	turbosquid.com	370k	0	77
Warehouse	turbosquid.com	640k	34	89
Sponza	Crytek	850k	0	56
Old City	turbosquid.com	1.2M	0	100
Dockside	*Call of Duty: Black Ops 2*	2.3M	8	20
Op925	*Battlefield 3*	2.8M	32	66
San Miguel	Evolución Visual	5.9M	0	1196

Table 5: Test scenes used in this paper, with triangle, animated character, and mesh counts.

	1-Layer	*2-Layer G-Buffer*			
Scene	*G-Buffer*	*Depth Peel*	*Reproject*	*Previous*	*Predict or Delay*
Sponza	3.0 ms	6.4 ms	**6.3** ms	6.3 ms	9.5 ms
Dockside	4.1	7.9	**7.8**	7.7	11.9
Op925	5.4	11.6	**11.4**	11.4	17.4
Kitchen	3.1	6.2	**4.6**	4.5	5.9
Old City	2.5	**4.6**	6.3	6.0	9.2
San Miguel	14.5	33.0	**29.8**	29.8	39.5
Grass	1.5	**3.2**	3.7	3.7	6.1
Office	0.7	1.8	**1.0**	1.0	1.4
Warehouse	3.5	7.3	**6.9**	6.8	9.2

Table 6: Time to generate a G-buffer. For reference, generating two layers via depth peeling always costs about twice as much as generating a single layer. Our new method using the 'Reproject' variant is often faster, even on current hardware with a slow geometry shader, and despite our use of an inexpensive vertex shader (compared to many current game engines).

4.1. Performance

Table 6 shows that single-pass generation already outperforms depth peeling for complex scenes (e.g., *San Miguel*) on a high-end GPU today, but underperforms on simple scenes (e.g., *Grass*); recall that we designed it for future GPUs with better geometry shader throughput.

Table 7 shows that the incremental cost of adding a second layer in GI computation is small. Our algorithms amortize the pixel iteration, sample tap computation, and framebuffer overhead – only the bandwidth cost increases measurably when adding more samples.

	Radiosity			AO
Scene	*Max Perf.*	*Balanced*	*Max Quality*	
Kitchen	2.1 + 0.5 ms	3.2 + 0.4 ms	5.4 + 1.0 ms	1.4 + 0.1 ms
Sponza	2.0 + 0.7	3.4 + 0.5	6.3 + 0.9	1.4 + 0.0
Old City	2.1 + 0.4	3.5 + 0.4	6.1 + 0.5	1.8 + 0.1
Dockside	1.8 + 0.5	3.2 + 0.3	6.1 + 0.3	1.7 + 0.1
Op925	2.2 + 0.5	3.6 + 0.3	6.3 + 0.3	1.7 + 0.0
San Miguel	2.2 + 0.5	3.5 + 0.5	6.0 + 0.7	1.7 + 0.0

Table 7: Execution times for two-layer deep G-buffer GI (including spatial and temporal reconstruction filtering), formatted as first layer + second layer. Amortized overhead makes the incremental cost for the second layer small. For our scenes with mirror reflective surfaces, the ray trace cost was *Kitchen*: 1.3 + 0.3 ms; *Dockside*: 1.7 + 0.1 ms; *San Miguel*: 1.0 + 0.2 ms.

4.2. Parameter Selection

G-buffer generation depends on a scene-dependent minimum separation constant, Δz: if Δz is too small, then the second layer will capture superfluous local detail; if Δz is too large, then the second layer will capture surfaces that are too distant and may miss important features. For example, in figure 9, $\Delta z = 1$ m fails to capture the blue wall behind the column and will instead see through to the green wall.

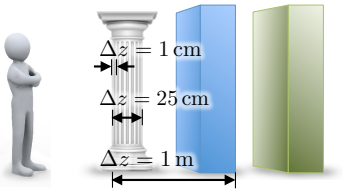


Figure 9

However, we consistently observe robust and stable image quality over a wide range of Δz settings, even for high depth complexity. We simply used $\Delta z = 50$ cm for all scenes in this paper, including the dense grass shown in figure 10.

Figure 11 demonstrates the impact of the temporal weight α on undersampling noise: our temporal filter is an exponentially-weighted moving average, so the useful range of α is on the high end of the unit interval; we generally recommend $\alpha = 0.85$.

4.3. Image Quality

As few as two layers can make a significant contribution to the appearance of scenes that have high depth variance and

Figure 10: Two-layer AO is significantly more coherent under motion than one-layer AO, even for high depth complexity scenes like *Grass* (see video results).

Figure 11: Increasing the temporal filter weight α decreases noise from undersampling in a single frame.

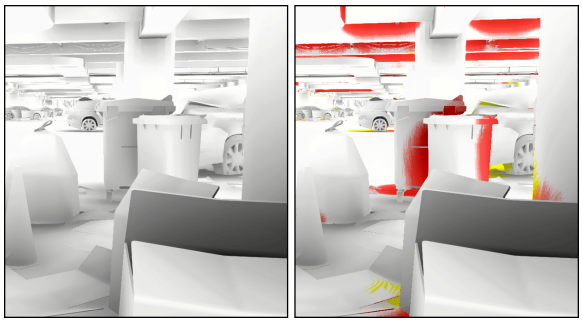
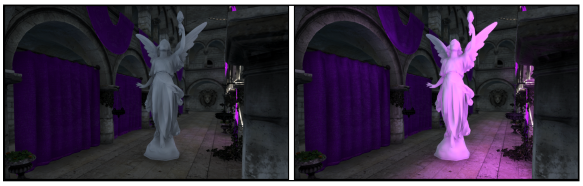


Figure 12: *Left:* Ambient occlusion in the *Op925* parking garage from *Battlefield 3* using two layers. *Right:* Areas where artifacts occur without (red) two-layers and (yellow) minimum separation.

Figure 13: Top: An aisle view in *Warehouse*. Bottom: For the view *across* an aisle, a single layer (left) misses the floor, yielding a result inconsistent with the top image. Two layers (right) give a more plausible result.



a) 1 layer G-buffer b) 2-layer deep G-buffer

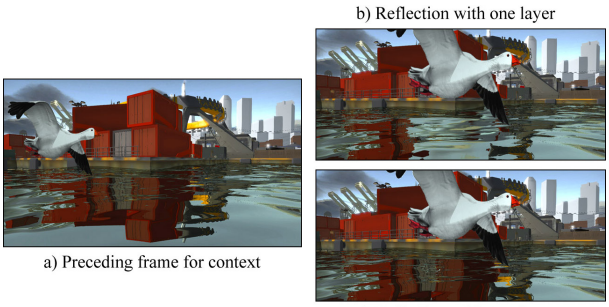
Figure 14: Radiosity in *Sponza*. A second layer captures reflection off the barely-seen purple banners on the right.

depth complexity. Figure 12 shows an increased quality for AO with a second layer in the deep G-buffer. Figures 13 and 14 demonstrate increased robustness to occlusion and viewpoint changes for radiosity. Figure 15 shows an increased quality for specular reflection.

Figures 1, 13, 14, and 16 also demonstrate that a layered deep G-buffer can provide sufficient information to indirectly illuminate large regions that receive no direct light, provided that direct light appears somewhere in the framebuffer (e.g., the second layer or guard band). The results inherently depend on the viewpoint, but in a way that has two desirable properties: radiosity and AO fade out smoothly as surfaces approach glancing angles, so illumination never "pops". Furthermore, our results are self-consistent for surfaces that are in (or nearly in) view.

There are three sources of error in our deep G-buffer radiosity approximation:

1. It overestimates E by assuming Y is **visible** from X.



b) Reflection with one layer

a) Preceding frame for context

c) Reflection with two layers

Figure 15: A second layer makes the red containers' reflection in *Dockside* more stable to passing foreground objects.

2. It underestimates E because surfaces not in the deep G-buffer are not represented.
3. The 2D spiral is biased away from the ideal 3D hemisphere **sampling**.
4. Ignoring the sample backface ($\hat{n}_Y$) test overestimates E.

Figure 17 explores these assumptions by presenting the same scene rendered offline with various assumptions relaxed. This is a qualitative analysis: we're concerned with where one perceives differences from ground truth after end-to-end processing, not the absolute numerical error.

Subimage (a) uses geometric ray casting against triangles and true 3D hemisphere samples with the full backface test to produce a ground-truth radiosity + direct illumination solution. The other subimages show all valid combination of approximations, cumulating in (f) as our fastest approximation. The major effects that we observe are *under*-estimation of indirect light when not using geometric ray casting: (a) vs. (b); *over*-estimation of indirect light when discarding the n_Y backfacing test: (e) vs (f). Although there is some global loss of contrast, these somewhat cancel, and the full offline result (a) and fastest real-time approximation (f) compare reasonably.

In these examples, we find that the approximation of assuming perfect visibility contributes less perceptible error than the 2D pattern bias, probably because of the fundamental assumption inherited from screen-space AO: closely-located surfaces that are facing each other often have few occluders between them. A distant or backfacing surface contributes little radiosity, so its visibility does not impact the result significantly.

We use reverse reprojection in multiple-scattered radiosity for both progressive computation and temporal filtering. In each case, reverse reprojection creates disoccluded regions ("holes") at newly revealed locations. Figure 18 shows the effect of disocclusion on progressive computation (the impact on filtering is comparable). Because the second layer can fill many disocclusions and radiosity has a wide gather kernel, the perceptible impact on the final image is small.

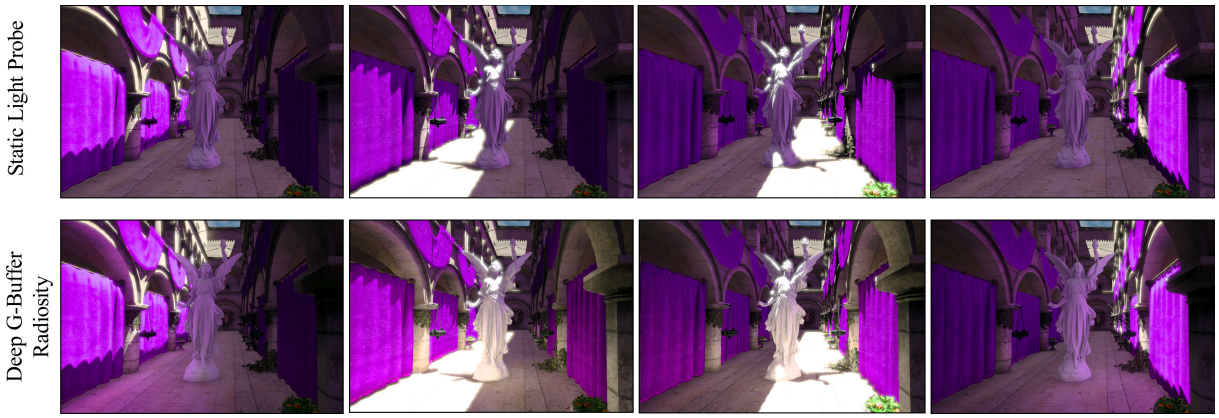
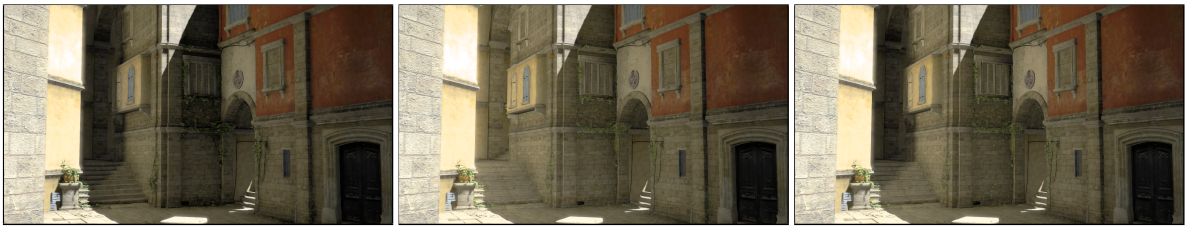


Figure 16: Top: Sponza under dynamic lighting with a static light probe lighting solution. Bottom: The same scene and lighting conditions using deep G-buffer radiosity, where indirect lighting changes plausibly and exhibits color bleeding and large-scale soft shadowing.

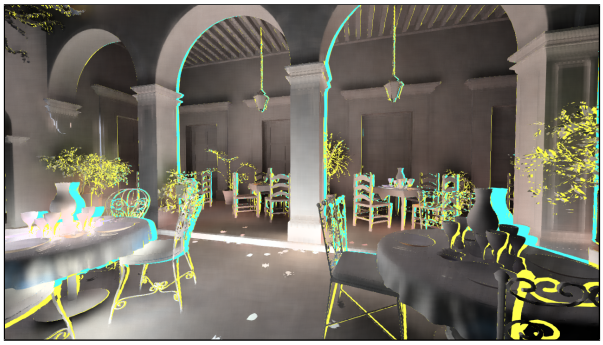
	(a) Ground truth	(b) Approximated visibility	(c) Approximated sampling
Sampling:	3D hemisphere	3D hemisphere	2D spiral
Visibility:	Geometric ray cast	Deep G-buffer ray cast	Geometric ray cast
Use $\hat{n}_Y$ *Test*:	Yes	Yes	Yes



	(d) Section 3.2	(e)	(f)
Sampling:	2D spiral	2D spiral	2D spiral
Visibility:	Deep G-buffer ray cast	None	None
Use $\hat{n}_Y$ *Test*:	Yes	Yes	No

Figure 17: Six estimates of single-scattered radiosity distinguish the qualitative impact of the underlying three approximations in our fast method. The combination of 3D sampling without a visibility test does not make sense, so it is not represented.

(a) [Ideal] converged irradiance for a static camera at position 1.



(b) Converged position 1 reprojected to position 2.

(c) [Ideal] converged irradiance at position 2 for a static camera.

(d) One frame of moving a camera from position 1 to 2.

(e) Difference between dynamic (d) and idealized static (c) results at position 2, scaled 3×.

(f) Final image lit by the moving camera irradiance from (d) and direct illumination.

Figure 18: The impact of reprojection (cyan = disoccluded in layer 1; yellow = disoccluded in both layers) on radiosity.

5. Conclusions

We introduced the notion of a deep G-buffer with minimum separation and a novel single-pass method for generating one, with four variants for solving the problem of needing the first layer's depth before it is rendered. We then applied that data structure to realistic, high-performance global illumination using present-day hardware. Our robust deep G-buffer indirect illumination sampling frames prior work on similar effects in the context of radiometric estimation, and we show how our deep G-buffers can be applied to such sampling applications.

While multiple layers increase robustness in these important scenarios, we showed that *both* the minimum separation criterion and our single-pass implementation are essential to generating high-performance, high-quality results. Finally, we described a sampling and spatio-temporal reconstruction strategy optimized for both image quality and performance.

Discussion. Our results illustrate, sometimes surprisingly, that we can often approach quality associated with offline global illumination using deep G-buffers. The techniques fail gracefully and in ways that self-identify undersampled regions, allowing fallback to a broad-scale strategy such as precomputed light probes or dynamic sparse voxel octtrees.

All of our single pass methods can generalize from 2 to k G-buffer layers, but the Prediction variant requires rendering $2k-1$ layers per frame ($k-1$ for depth-only). The Reprojection (and less desirable Previous) variants require rendering only k layers per frame.

Although the sampling biases that we intentionally introduced for performance affect radiometric correctness, we observe that results are still stable and give the perception of plausible global illumination. We speculate that the reason that the results are pleasing despite radiometric bias is rooted in similar properties for manually constructed scenes and images. We observe that 2D artists and live action film directors only seek to match correct illumination for the scene at the broad scale. This is most evident on a television set, where just off camera, an entire sound stage appears rather than the remainder of the environment implied by the set. The broad illumination from the missing half of the set is supplied by large stage lights and diffusers. Within the set, physics dictates that fine-scale illumination is accurate. Our methods have a similar property, since they capture the fine-scale geometry that is visible to the viewer but fall back to broad-scale for that which is far from being seen.

Index of Supplemental Materials

We include video results and a C++/OpenGL implementation of deep G-buffer ambient occlusion and radiosity. The ray casting used for our video and static results are included with the demo source but are not used in the demo. Temporal filtering is built directly into the open source support library. Unlike the shader-based lighting passes of AO and radiosity, single-pass deep G-buffer generation is both simpler to implement and more API dependent, so we leave it as pseudocode in this paper rather than providing executable code.

`Mara2014DeepGBufferDemo.zip` Demonstration application:

- `README.TXT` Information on the demo.
- `DeepGBufferRadiosity.exe` Demo compiled for 64-bit Windows 7 and 8 machines, tested on NVIDIA Kepler-architecture GeForce GPUs.
- `source/` C++/OpenGL source for the demo, which requires the G3D Innovation Engine [MMO14] to build.
- `data/shader/reverseReprojection.glsl` Reverse reprojection sample code.
- `data/shader/reconstructFromDepth.glsl` Position from depth and deep G-buffer ray tracing sample code.

`Mara2014DeepGBuffer.mp4` Video results.

Acknowledgements

We thank Aaron Lefohn for motivating us to investigate single-pass generation and constant feedback on the demonstration program, John Hughes for working with us on the radiometric derivations, Frank Meinl and Guillermo M. Leal Llaguno for their 3D models, Treyarch and DICE for the use of their models, and Guedis Cardenas for rigging some of the scenes. The publicly-distributable models used in our experiments are available from http://graphics.cs.williams.edu/data; the others are from TurboSquid and the named games.

An early version of this work appeared in a 2013 NVIDIA technical report [MML13], now superseded by this article.

References

[BCL*07] BAVOIL L., CALLAHAN S. P., LEFOHN A., COMBA JO A. L. D., SILVA C. T.: Multi-fragment effects on the GPU using the *k*-buffer. In *I3D* (2007), ACM, pp. 97–104. 3

[Ben14] BENTLEY A.: Engine postmortem of inFAMOUS: Second Son, 2014. GDC Talk. URL: http://adruab.net/wp-images/GDC14_infamous_second_son_engine_postmortem.pdf. 5

[BM08] BAVOIL L., MYERS K.: *Order independent transparency with dual depth peeling*. Tech. rep., NVIDIA, 2008. 3, 5

[Bra13] BRAINERD W.: Profiling results on Playstation4 at Activision Maine, October 2013. Personal comm. 4

[BS09a] BAVOIL L., SAINZ M.: Multi-layer dual-resolution screen-space ambient occlusion. In *ShaderX*^{7}, Engel W., (Ed.). 2009. 2, 3, 4

[BS09b] BAVOIL L., SAINZ M.: Multi-layer dual-resolution screen-space ambient occlusion. In *SIGGRAPH 2009 Talks* (New York, NY, USA, 2009), ACM, pp. 1–1. 2, 3, 4

[Buk13] BUKOWSKI M.: Profiling results on NVIDIA GeForce 670 at Vicarious Visions, October 2013. Personal comm. 4

[Car84] CARPENTER L.: The A-buffer, an antialiased hidden surface method. *SIGGRAPH 18*, 3 (Jan. 1984), 103–108. 3

[CDE*14] CIGOLLE Z. H., DONOW S., EVANGELAKOS D., MARA M., MCGUIRE M., MEYER Q.: A survey of efficient representations for independent unit vectors. *Journal of Computer Graphics Techniques (JCGT) 3*, 2 (April 2014), 1–30. URL: http://jcgt.org/published/0003/02/01/. 8

[CG85] COHEN M. F., GREENBERG D. P.: The hemi-cube: a radiosity solution for complex environments. *SIGGRAPH 19*, 3 (July 1985), 31–40. 7

[CG12] CRASSIN C., GREEN S.: *Octree-based sparse voxelization using the GPU hardware rasterizer*. CRC Press, 2012. 3

[Cha11] CHAPMAN J.: Deferred rendering, transparency & alpha blending, January 2011. Blog post. URL: http://www.john-chapman.net/content.php?id=13. 3

[CNLE09] CRASSIN C., NEYRET F., LEFEBVRE S., EISEMANN E.: GigaVoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *I3D* (NY, NY, USA, 2009), ACM, pp. 15–22. 3, 4

[Cof11] COFFIN C.: SPU-based deferred shading for battlefield 3 on playstation 3, 2011. GDC Talk. URL: http://dice.se/wp-content/uploads/Christina_Coffin_Programming_SPU_Based_Deferred.pdf. 5

[CRW13] CHRIS RAINE T. S., WENZEL C.: Rendering technologies of crysis 3, 2013. GDC Talk. URL: http://www.crytek.com/cryengine/presentations/the-rendering-technologies-of-crysis-3. 5

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *I3D* (2005), ACM, pp. 203–231. 2, 3, 4, 6

[DWS*88] DEERING M., WINNER S., SCHEDIWY B., DUFFY C., HUNT N.: The triangle processor and normal vector shader: A VLSI system for high performance graphics. *SIGGRAPH 22*, 4 (June 1988), 21–30. 3, 4

[ED06] EISEMANN E., DÉCORET X.: Fast scene voxelization and applications. In *I3D* (2006), ACM SIGGRAPH, pp. 71–78. 3

[Eve01] EVERITT C.: *Interactive order-Independent transparency*. Tech. rep., NVIDIA, 2001. 3

[FM08] FILION D., MCNAUGHTON R.: Starcraft II effects & techniques. In *Advances in real-time rendering in 3D graphics and games course notes*, Tatarchuk N., (Ed.). August 2008. 5

[Hac05] HACHISUKA T.: *High-Quality Global Illumination Rendering Using Rasterization*. GPU Gems 2 – Addison-Wesley Professional, 2005, ch. 38. 2, 4

[Hal10] HALÉN H.: Style and gameplay in the Mirror's Edge, July 2010. Stylized Rendering in Games SIGGRAPH Course. 7

[HMY12] HARADA T., MCKEE J., YANG J. C.: Forward+: Bringing deferred lighting to the next level. In *Eurographics 2012-Short Papers* (2012), The Eurographics Association, pp. 5–8. 4

[ICG86] IMMEL D. S., COHEN M. F., GREENBERG D. P.: A radiosity method for non-diffuse environments. In *SIGGRAPH* (1986), ACM, pp. 133–142. 7

[JB10] JANSEN J., BAVOIL L.: Fourier opacity mapping. In *I3D* (2010), ACM, pp. 165–172. 3

[LK10] LAINE S., KARRAS T.: Efficient sparse voxel octrees. In *I3D* (NY, NY, USA, 2010), ACM, pp. 55–63. 4

[LV00] LOKOVIC T., VEACH E.: Deep shadow maps. In *SIGGRAPH* (2000), ACM Press, pp. 385–392. 3

[MB07] MYERS K., BAVOIL L.: Stencil routed a-buffer. In *SIGGRAPH Sketches* (2007), ACM. 3

[McG13] MCGUIRE M.: Profiling results on NVIDIA GeForce 660 at Unknown Worlds, October 2013. Personal comm. 4

[MG12] MICKAEL GILABERT N. S.: Deferred radiance transfer volumes. Game Developers Conference. URL: http://twvideo01.ubm-us.net/o1/vault/gdc2012/slides/Programming%20Track/Stefanov_Nikolay_DeferredRadianceTransfer.pdf. 8

[MML12] MCGUIRE M., MARA M., LUEBKE D.: Scalable ambient obscurance. In *HPG* (June 2012). 3, 7, 9

[MML13] MARA M., MCGUIRE M., LUEBKE D.: Lighting deep G-buffers: Single-pass, layered depth images with minimum separation applied to indirect illumination, December 2013. NVIDIA Technical Report. 15

[MMO14] MCGUIRE M., MARA M., OTHERS: G3D Innovation Engine. http://g3d.sf.net, 2014. C++ library, version 10. 15

[MP01] MARK W. R., PROUDFOOT K.: The F-buffer: a rasterization-order FIFO buffer for multi-pass rendering. In *Graphics Hardware* (2001), ACM, pp. 57–64. 3

[NRS14] NALBACH O., RITSCHEL T., SEIDEL H.-P.: Deep screen space. In *I3D* (2014), ACM, pp. 79–86. 3

[NSL*07] NEHAB D., SANDER P. V., LAWRENCE J., TATARCHUK N., ISIDORO J. R.: Accelerating real-time shading with reverse reprojection caching. In *Graphics Hardware* (2007), Eurographics Association, pp. 25–35. 6

[Per07] PERSSON E.: Deep deferred shading, November 2007. Blog post. URL: http://www.humus.name/index.php?page=3D&ID=75. 3

[PLAN98] POPESCU V., LASTRA A., ALIAGA D., NETO M. D. O.: Efficient warping for architectural walkthroughs using layered depth images. In *IEEE Visualization* (1998), pp. 211–215. 3

[RGS09] RITSCHEL T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *I3D* (2009), ACM, pp. 75–82. 2, 3, 4

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on GPUs. In *I3D* (2007), ACM, pp. 73–80. 2, 3, 4

[SA09] SINTORN E., ASSARSSON U.: Hair self shadowing and transparency depth ordering using occupancy maps. In *I3D'09* (New York, NY, USA, 2009), ACM, pp. 67–74. 3

[SA11] SUCHOW J. W., ALVAREZ G. A.: Motion silences awareness of visual change. *Curr. Bio. 21*, 2 (2011), 140 – 143. 6

[Sal13] SALVI M.: Pixel synchronization: solving old graphics problems with new data structures, 2013. SIGGRAPH Courses: Advances in real-time rendering in games. 3, 4

[Sch12] SCHWARZ M.: Practical binary surface and solid voxelization with Direct3D 11. In *GPU Pro 3*, Engel W., (Ed.). A K Peters/CRC Press, 2012, pp. 337–352. 3

[Sch14] SCHULZ N.: Moving to the next generation - the rendering technology of ryse, 2014. URL: http://www.crytek.com/download/2014_03_25_CRYENGINE_GDC_Schultz.pdf. 5

[SGHS98] SHADE J., GORTLER S., HE L.-W., SZELISKI R.: Layered depth images. In *SIGGRAPH* (1998), ACM, pp. 231–242. 3

[Shi91] SHIRLEY P.: Discrepancy as a quality measure for sample distributions. In *Eurographics* (1991), Elsevier, pp. 183–194. 9

[SHRH09] SOLER C., HOEL O., ROCHET F., HOLZSCHUCH N.: *A Fast Deferred Shading Pipeline for Real Time Approximate Indirect Illumination*. Tech. rep., Institut National de Recherche en Informatique et en Automatique, 2009. 3, 7

[SKS11] SOUSA T., KASYAN N., SCHULZ N.: Secrets of CryENGINE 3 graphics technology. In *SIGGRAPH Courses* (New York, NY, USA, 2011), ACM. 10

[SML11] SALVI M., MONTGOMERY J., LEFOHN A.: Adaptive transparency. In *HPG* (2011), ACM, pp. 119–126. 3

[SS89] SALESIN D., STOLFI J.: The ZZ-buffer: A simple and efficient rendering algorithm with reliable antialiasing. In *PIXM'89* (1989), pp. 415–465. 3

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-d shapes. *SIGGRAPH 24*, 4 (1990), 197–206. 3

[TTV13] TATARCHUK N., TCHOU C., VENZON J.: Destiny: From mythic science fiction to rendering in real-time. In *SIGGRAPH 2013 Talks* (New York, NY, USA, 2013), ACM. URL: http://advances.realtimerendering.com/s2013/Tatarchuk-Destiny-SIGGRAPH2013.pdf. 5

[Val09] VALIENT M.: The rendering technology of killzone 2, 2009. GDC Talk. URL: http://www.slideshare.net/guerrillagames/deferred-rendering-in-killzone-2-9691589. 5

[VPG13] VARDIS K., PAPAIOANNOU G., GAITATZES A.: Multi-view ambient occlusion with importance sampling. In *I3D* (2013), ACM, pp. 111–118. 2, 3, 4

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *SIGGRAPH 12*, 3 (Aug. 1978), 270–274. 4

[YK07] YUKSEL C., KEYSER J.: *Deep Opacity Maps*. Tech. rep., Dept. of Comp. Sci., Texas A&M University, 2007. 3