Lighting Deep G-Buffers: Single-Pass, Layered Depth Images with Minimum Separation Applied to Indirect Illumination

Michael Mara   Morgan McGuire   David Luebke
NVIDIA

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

Figure 1: *Left:* Direct plus hemisphere ambient illumination in *San Miguel*, which has 6.5M triangles in 968 draw calls. *Right:* Real-time approximate global illumination based on a two-deep G-buffer computed in a single scene geometry submission. At 1920×1080 on NVIDIA GeForce TITAN, the single-pass two-deep G-buffer costs 33 ms, which is 28% less than two single layers by depth peeling. Given the deep G-buffer, all shading passes (direct, AO, radiosity, mirror) combined execute in 7 ms.

Abstract

We introduce a new method for computing two-level Layered Depth Images (LDIs) [Shade et al. 1998] that is designed for modern GPUs. The method is order-independent, can guarantee a minimum separation distance between the layers, operates within small, bounded memory, and requires no explicit sorting. Critically, it also operates in a single pass over scene geometry. This is important because the cost of streaming geometry through a modern game engine pipeline can be high due to work expansion (from patches to triangles to pixels), matrix-skinning for animation, and the relative scarcity of main memory bandwidth compared to caches and registers.

We apply the new LDI method to create *Deep Geometry Buffers* for deferred shading and show that two layers with a minimum separation distance make a variety of screen-space illumination effects surprisingly robust. We specifically demonstrate improved robustness for Scalable Ambient Obscurance [McGuire et al. 2012b], an extended *n*-bounce screen-space radiosity [Soler et al. 2009], and screen-space reflection ray tracing. All of these produce results that are necessarily view-dependent, but in a manner that is plausible based on visible geometry and more temporally coherent than results without layers.

1   Introduction

Efficient hardware rasterization of triangles enables modern real-time rendering. Rasterization excels in cases where surfaces with primary visibility are sufficient to compute shading at every pixel. In some cases it can be extended with more global information by

rasterizing another view. For example, a shadow map contains primary visibility information from the light source's view.

Previous work has shown that multiple views, such as reflective shadow maps [Dachsbacher and Stamminger 2005] or orthographic depth peeled projections [Hachisuka 2005] can capture enough global information to approximate some global illumination effects. However, when provided with only a single view, these effects miss too much information to provide temporally coherent results. The challenge is therefore to increase the efficiency of producing multiple views while also extending the shading to operate with as few views as possible.

Multipass rendering techniques have been popular over the past decade. However, modern games are increasingly limited by the cost of making multiple passes over the same geometry. Each pass recomputes scene traversal, frustum culling, depth sorting, tessellation, and transformations such as vertex-weighted skeletal animation. Submitting draw calls and changing graphics state also incur overhead. Low-level APIs such as bindless OpenGL [Bolz 2009], AMD's Mantle interface [AMD 2013], proprietary console low-level APIs, and strategies such as vertex pulling [Engel 2013] can mitigate this expense. In an informal survey [Brainerd 2013; Bukowski 2013; McGuire 2013], we asked game developers to measure the cost of the pre-rasterization portion of their rendering pipeline. For games currently in development, they benchmarked the net frame cost of rendering to a small 32×32 frame buffer to minimize the impact of pixel shading. They reported that one sixth to one third of the cost of rendering a view was spent in these scene and geometry portions of the pipeline.

These observations motivate us to generate multiple views from a *single* pass over the geometry. To do so, we turn to the Layered Depth Image (LDI) introduced by Shade et al. [1998]. This captures information about both primary and hidden layers of a single view, thus enabling shading algorithms to extract more information from a single pass. Previous research has investigated hardware warping and rendering of LDIs [Popescu et al. 1998], and hardware-assisted

generation of LDIs through methods such as depth peeling [Everitt 2001] and *k*-buffers [Myers and Bavoil 2007]. The notion of an LDI extends directly to other kinds of layered frame buffers. In particular, we create LDIs of Saito and Takahashi's Geometry Buffers (G-buffers) [**?**]. Single-layer G-buffers are commonly applied today for deferred shading as pioneered by Deering et al. [1988]. A layered, or *Deep G-Buffer* stores at each pixel of each layer the position, normal, and material parameters required for later shading. It can be thought of as a frustum voxelization of the scene that has very high resolution in camera-space *x* and *y* but extremely low resolution in *z.* This data structure has previously been used for deferred shading in the presence of transparency [Persson 2007; Chapman 2011]; we extend it to global illumination and consider efficient generation. Note that for some algorithms, such as ambient occlusion, this may be simply a layered depth buffer from which position and approximate normals can be reconstructed. In other cases, full shading information is required to estimate the scattering of indirect light.

There must be some *minimum separation distance* between layers for a deep G-buffer with a small number of layers to be useful for global illumination, as we show in the following section. Without this separation, fine local detail near the first layer prevents capturing information from farther back in the scene. We provide the first hardware algorithm to generate any kind of LDI, including a deep G-buffer, with this minimum separation. Furthermore, the algorithm achieves this in a single pass. We then show how to apply these deep G-buffers to robustly approximate various global illumination effects in real time, and contribute extensions of existing screen-space lighting techniques of mirror reflection, radiosity, and ambient occlusion. Our specific contributions are:

- Motivation for minimum separation
- The algorithm for generating LDIs with minimum separation in a single hardware rendering pass over geometry
- Application of deep G-buffers to screen-space illumination, using extensive reprojection and both spatial and temporal filtering for robustness and coherence
- Evaluation of illumination quality due to deep G-buffers versus single-layer results

Throughout this work, we assume future hardware with an efficient geometry shader and hierarchical early depth test for layered framebuffers. The geometry shader and layered framebuffer have previously suffered a chicken-and-egg problem: they are potentially powerful features underutilized by today's games because the hardware implementations are unoptimized, and hardware vendors have never had clear use cases to optimize for because they are rarely used. This paper implicitly seeks to inform future designs by presenting concrete and compelling new use cases. That is, our techniques are hardware-friendly, but the hardware could be improved to be friendlier to the techniques.

2 Related Work

Depth Peeling There are many techniques for capturing multiple layers at a pixel. Ordered by decreasing memory footprint, these include clip-space voxelization [Schwarz 2012; Crassin and Green 2012], the F-buffer [Mark and Proudfoot 2001] and A-buffer [Carpenter 1984], ZZ-buffer [Salesin and Stolfi 1989], *k*-buffers and other bounded A-buffer approximations [Lokovic and Veach 2000; Myers and Bavoil 2007; Bavoil et al. 2007; Salvi et al. 2011; Salvi 2013], frequency A-buffer approximations [Yuksel and Keyser 2007; Sintorn and Assarsson 2009; Jansen and Bavoil 2010], and depth peeling [Everitt 2001; Bavoil and Myers 2008].

Of these, *depth peeling* is particularly interesting for rendering effects that receive primary benefit from two or three depth layers at
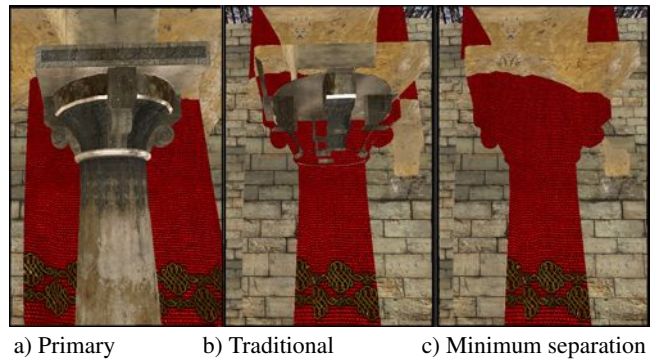

a) Primary   b) Traditional   c) Minimum separation

Figure 2: *Traditional single depth peel gives little information in areas of high local depth complexity. Minimum separation depth peel reveals the next major surface, which captures more global scene information.*

a pixel because it has the smallest memory footprint. Previous work shows that low-frequency screen-space global illumination effects are among those that gain significant quality and robustness from even one additional layer such as a depth peel [Shanmugam and Arikan 2007; Ritschel et al. 2009; Vardis et al. 2013].

Minimum Separation A state of the art, single depth peel pass returns the second-closest surface to the camera at each pixel. This can be implemented by with two passes over the scene geometry [Bavoil and Myers 2008], or with a single pass that uses programmable blending [Salvi 2013].

Despite the theoretical applications of a small number of depth layers, for complex and production assets, we observe that the second-closest surface often provides little additional information about the scene. That is because non-convex geometry, decals, and detail models often create excess local detail that obscure the broad contours of the scene under peeling. For example, in the *Sponza* scene shown in figure 2, a depth peel reveals only the second fold of the column's decorative moulding at the top and not the full red tapestry.

One solution to this local detail problem is to introduce a *minimum separation distance* between the first and second layers. That is, modifying the depth peel to select only fragments that are at least a constant distance beyond primary visible surfaces and in front of all other surfaces. This is trivial to implement in a two-pass depth peel but impossible to accomplish in bounded memory using programmable blending, since until all surfaces have been rasterized each pixel has no way of knowing what the minimum depth for the second layer is. That is, programmable blending allows rendering a *k*-buffer in a single pass, but we don't need a $k = 2$ buffer for this application: we need two *particular* layers from a $k = \infty$ buffer.

Indirect Light We've already referenced several screen-space indirect light methods. Our method is most directly related to Ritschel et al.'s [2009] directional occlusion and Vardis et al.'s [2013] AO approximation. Both previous techniques use multiple views and note the performance drawbacks of doing so. We make a straightforward extension to handling multiple layers and then show how to factor the approximations to compute arbitrary radiosity iterations and specular reflection and incorporate McGuire et al.'s [2012b] scalable gathering to make large gather radii and multiple iterations practical. Our GI applications are also closely related to previous image-space gathered illumination [Dachsbacher and Stamminger 2005; Soler et al. 2009].

2.1 Key GPU Concepts

We briefly describe the key GPU texture concepts used by our algorithms, in the language used by the OpenGL abstraction. In

OpenGL, camera-space "depth" or "z" inconsistently refers to a hyperbolically-encoded value from the projection matrix on the interval [-1, 1] (gl_FragCoord.z), that value remapped to [0, 1] (when reading from a depth sampler), an actual z coordinate that is negative in camera space, or a z coordinate that is positive. For simplicity, we use "depth" and "z" interchangeably as a value that increases monotonically with distance from the camera along the view axis and assume the implementer handles the various mapping discrepancies in the underlying API.

A 2D *MIP level* (we only use 2D in this paper) is a 2D array of vector-valued ("RGBA" vec4) pixels. A 2D *texture* is an array of MIP-levels. Element m contains 4^{-m} as many pixels as element 0. A *texture array* is an array of textures whose index is called a *layer*. A *framebuffer* is an array of texture arrays that are bound as a group for writing. The texture array index in a framebuffer is called an *attachment point* and is described by an enumeration constant (e.g., GL_COLOR_ATTACHMENT0, GL_DEPTH_ATTACHMENT), which we abbreviate as C0...Cn or Z. Note that there are many "offscreen" framebuffers in a typical rendering system, and they are independent of the "hardware framebuffer" used for display. A pixel inside framebuffer F_t for frame number ("time") t is fully addressed by:

$$F_t[attachment].\text{layer}(L).\text{mip}(m)[x,y] \qquad (1)$$

Geometry is encoded in attribute buffers, e.g., as indexed triangle lists, and modified by transformations during tessellation, hull, vertex, and geometry shaders. In this paper T_t denotes the frame time-dependent portion of that transformation, which for rigid bodies is the modelview projection matrix. The geometry shader selects the layer of the bound framebuffer's textures to write to independently for each emitted primitive. We use this to rasterize variations of an input triangle against each layer of a framebuffer during peeling.

3 Single-Pass with Minimum Separation

Listing 1 is pseudocode for generating multiple layers of frame t from geometry G_t under transformation T_t via multi-pass depth peeling. When $\Delta z = 0$, this code is the classic depth peel [Bavoil and Myers 2008]. When $\Delta z > 0$, it becomes our straightforward extension of multipass peeling that guarantees a minimum separation. The arbitrary shading function S can compute radiance to the eye, geometry buffers, or any other desired intermediate result. It is of course possible (and often preferable) to implement this algorithm using two separate frame buffers and no geometry shader. We present this structure to make the analogy and notation clear when moving to a single pass.

Listing 2 is our more sophisticated algorithm for directly generating a two-deep layered depth image with minimum separation in single pass over the geometry. It renders to both layers simultaneously. To do so perfectly, it requires an oracle that predicts the depth buffer's first layer *before* that buffer has been rendered. We now describe four variants of our algorithm that approximates that oracle in different ways.

a) Delay: Add one frame of latency (which may already be present in the rendering system, e.g., under double or triple buffering) so that the transformations for the next frame T_{t+1} are known at rendering time. This enables perfect prediction of the next frame's first depth layer. Frame t reads (in line 16) from the oracle computed for it by the previous frame, and generates the oracle for frame $t+1$ (in lines 9, 18, and 19) to satisfy the induction. The primary drawback of this variant is that it requires a frame of latency. Reducing latency, even back to single-buffering by racing the GPU's scanout, is increasingly desirable for some applications such as virtual and augmented reality.

```
1   setDepthTest( LESS )
2   bindFramebuffer( F_t )
3   clear()
4
5   submit G_t with:
6      geometry shader(tri):
7         emit T_t(triangle) to layer 0
8      pixel shader(x,y,z):
9         return S(x,y,z)
10
11  submit G_t with:
12     geometry shader(tri):
13        emit tri to layer 1
14     pixel shader(x,y,z):
15        if (z > F_t[Z].layer(0).mip(0)[x,y] + Δz): return S(x,y,z)
16        else: discard the fragment
```

Listing 1: Pseudocode for a baseline *two-pass* LDI/deep G-buffer with minimum separation Δz generation using depth peeling. Our method improves on this baseline. The code processes geometry G_t using an arbitrary shading/G-buffer write function S of the sampled fragment position x, y, z. The output resides in two layers of the texture arrays in F_t.

```
1   setDepthTest( LESS )
2   bindFramebuffer( F_t )
3   clear()
4
5   submit G_t with:
6      geometry shader(tri)
7         emit T_t(tri) to layer 0
8         emit T_t(tri) to layer 1
9         if (VARIANT == Delay) || (VARIANT == Predict):
10           emit T_{t+1}(tri) to layer 2
11
12     pixel shader(x,y,z):
13        switch (layer):
14        case 0: // First layer; usual G−buffer pass
15           return S(x,y,z)
16
17        case 1: // Second G−buffer layer: choose the comparison texel
18           if (VARIANT == Delay) || (VARIANT == Predict):
19              L = 2 // Comparison layer
20              C = (x,y,z) // Comparison texel
21           else if (VARIANT == Previous):
22              L = 0; C = (x,y,z)
23           else if (VARIANT == Reproject):
24              L = 0; C = (x_{t-1}, y_{t-1}, z_{t-1})
25
26           if (z_C > F_{t-1}[Z].layer(L).mip(0)[x_C,y_C] + Δz): return S(x,y,z)
27           else: discard the fragment
28
29        case 2: // Predict F_{t+1}[Z].layer(0).mip(0); no shading
30           return // We only reach this case for Delay and Predict
```

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

b) Previous: Simply use the previous frame's first depth layer as an approximation of the oracle. The quality of the approximation decreases as scene velocities (including the camera's) increase. In practice, this may be acceptable for three reasons. First, errors will only manifest in the second layer, not in visible surfaces. Second, the errors are only in the minimum separation value. The second layer still represents only surfaces that are actually in the scene and

2nd Layer

Diff. from Delay

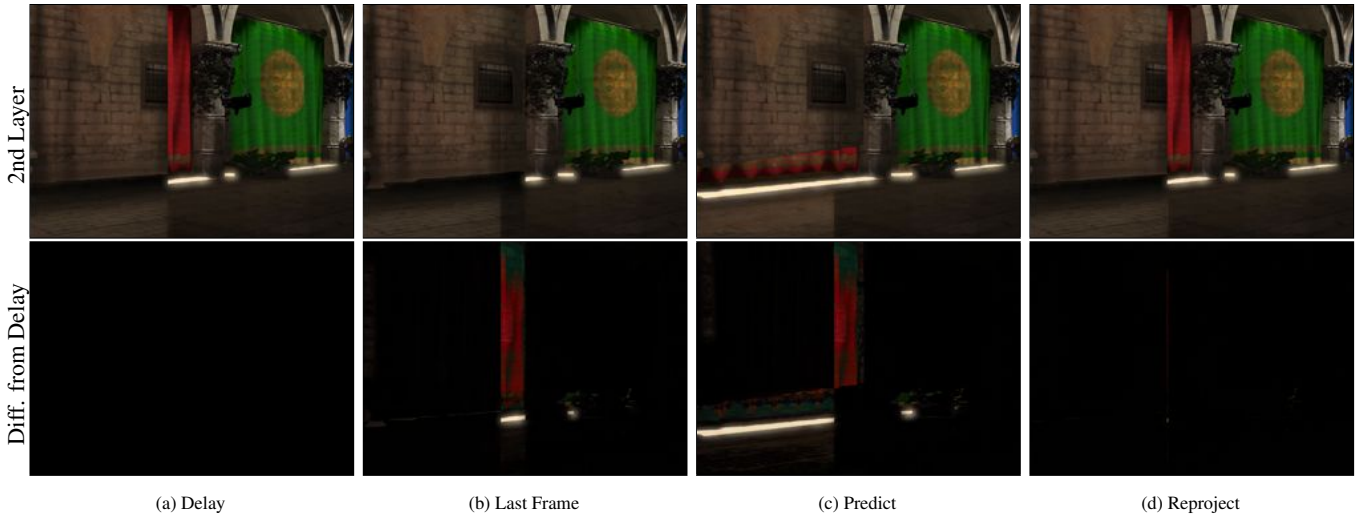(a) Delay   (b) Last Frame   (c) Predict   (d) Reproject

Figure 3: Top: Shaded images of the second layer, generated under each of the four algorithm variants, while rotating the camera erratically in *Sponza*, behind one of the corner pillars. Bottom: The diff of the top row images with a ground truth second layer generated through two-pass depth peeling. The Delay variant is equivalent to ground truth, since we have a perfect oracle. Under rotation (or fast lateral motion), the Last Frame variant is prone to high error, in this case, for example, it completely peels past the leftmost red banner even though the right part of the banner shows up in the ground truth version. The Predict variant is an improvement over Delay if the prediction is good. In this case, under erratic camera rotation, it fairs poorly, overpeeling in much of the same area as Last Frame, and underpeeling (and thus just duplicating the first layer) in other areas, including the bottom of the red banner. The Reprojection variant fairs better than either Last Frame and Predict, with only a small amount of error at some depth discontinuities.

are at the correct positions at time t. Third, a viewer may not notice errors in the resulting image since they will only occur near objects in motion. Motion overrides perception of precise image intensities and even shape [Suchow and Alvarez 2011], and furthermore, the artifacts may themselves be blurred in the image if motion blur effects are applied.

c) Predict: Predict T_{t+1} using the physics and animation system's velocity values, or computing them by forward differences from the previous and current frame's vertex positions. When the velocity available is accurate, this gives perfect results like variant A but without latency. When the velocity is inaccurate, the errors and mitigating factors from the Previous variant apply.

d) Reproject: Use the previous frame's layer 0 depth and use *reverse reprojection* on each fragment to peel against it. This is a variant on reverse reproduction caching [Nehab et al. 2007]. We propagate the previous frame's camera-space positions through the system to the vertex shader, and use it to compute the screen coordinates and depth value of the fragment in the previous frame's depth buffer. We are essentially propagating visibility a frame forward in time. This method is susceptible to reprojection error around silhouette edges, but has the added benefits over the Predict variant of not requiring a third depth layer and always having accurate velocities. Since this method requires the previous frame's camera-space positions, it can increase bandwidth and skinning costs. However, many production systems already have this information available in the pixel shader, for use in effects such as screen-space motion blur [McGuire et al. 2012a].

Figure 3 shows fully shaded output from each of the four variants for a frame of animation from *Sponza*, along with differences from the Delay version, which gives ground truth. The Previous variant, and the Predict variant when prediction fails, can yield errors over large portions of the screen. The Reproject variant localizes error to small portions of silhouette edges in non-pathological scenarios.

4 Applications

There are many potential applications of a deep G-buffer, including global illumination, stereo image reprojection, depth of field, transparency, and motion blur. We evaluated adaptations of screen-space global illumination methods to accept deep G-buffers as input with the goal of increasing robustness to occlusion. Screen-space ambient obscurance as a term for modulating environment map light probes is currently widely used in the games industry, so we begin there.

Recognizing that the derivation of ambient occlusion is a subset of that of single-bounce radiosity, we generalize the AO algorithm to radiosity. We observe despite the popularity and apparent success of screen-space ambient occlusion, screen-space radiosity is currently uncommon. We hypothesize that this is because computing radiosity from a single layer or view is not effective, and that the deep G-buffer can makes it practical. Multiple-bounce radiosity requires many samples to converge, so we use temporal smoothing and reverse reprojection to obtain those samples while only incurring cost equivalent to computing a single bounce per frame.

We note that radiosity computed on a deep G-buffer is similar to that computed by Reflective Shadow Maps [Dachsbacher and Stamminger 2005], which use a second view from the light's perspective. The major differences are that by working in the camera's space we can amortize the cost of work already performed in a deferred-shading pipeline and compute higher-order effects involving objects visible to the viewer but not the light. We speculate that objects visible (or nearly visible) to the viewer are the most important for plausible rendering.

Finally, we investigate screen-space mirror reflection tracing. As future work we plan to investigate glossy reflection by modifying the reflection rays to use pre-filtered versions of the screen or modifying the BSDF in the radiosity algorithm, depending on the angular width of the desired glossy lobe.

4.1 Ambient Occlusion

We extend the Scalable Ambient Obscurance [McGuire et al. 2012b] algorithm to take advantage of the deep G-buffer data structure and make some additional quality improvements. The additional quality improvements are motivated by the increased robustness; they address sources of error that are dominated by undersampling of the scene in the original algorithm.

Collectively, our changes produce more plausible shading falloff, avoids the viewer-dependent white halos from the previous work, and at the same number of samples reduce noise in the final result.

The method begins by extending the MIP computation accept the deep G-buffer as input. That pass outputs a single texture map with multiple MIP levels of the depth buffer. The MIP levels are computed by rotated-grid sampling (as in the original SAO) of camera-space z values. To amortize texture fetch address calculation, at each MIP level we pack the two depth layers into two color channels of an RG32F texture.

From the projection matrix and these z values the algorithm reconstructs camera-space positions. McGuire et al. recovered camera-space surface normals from this MIP-mapped depth buffer. To better match the increased quality from multiple layer we revert to passing an explicit camera-space normal buffer for the first layer. We do not compute MIP maps of normals or require a second layer because only normals at full resolution of surfaces visible in the final image appear in the algorithm.

The net solid-angle weighted Ambient Visibility (1 - AO) at a camera-space point X in the layer 0 buffer from N samples is

$$AV(X) = \left[\max\left(0, 1 - \sqrt{\frac{\pi}{N} \sum_{i=1}^{N} \max\left(AO(X, R_i), AO(X, G_i), 0\right)} \right) \right]^{\sigma},$$
(2)

where σ is the intensity scale ($\sigma = 1.0$ is a default), R_i is the ith sample from the R channel and G_i is the corresponding G channel value. That is, we use the previous work's algorithm but consider the union of occlusion from both layers.

Ambient Occlusion at point X with normal $\hat{n}_X$ due to a sample at Y, where $\vec{v} = Y - X$ (all in camera space coordinates) is

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

This intermediate value can be negative due to the falloff factor as a result of amortizing the max against zero out into equation 2.
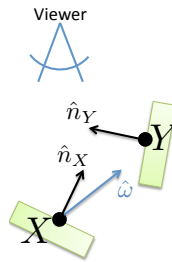
4.2 Radiosity

Soler et al. [2009] introduced a screen-space radiosity approximation. We derive a radiometrically correct form-factor computation for such a method, note the sources of error, extend it to work with our deep G-buffer, and then give performance and aesthetically motivated alterations for a practical algorithm.

The total irradiance $E(X)$ incident at patch X with normal $\hat{n}_X$ due to the radiosity $B(Y)$ exiting each unoccluded patch Y with normal $\hat{n}_Y$ and area A_Y is [Cohen and Greenberg 1985]

$$E(X) \approx \sum_{\text{all visible } Y} A_Y B(Y) \frac{\max(-\hat{\omega} \cdot \hat{n}_Y, 0) \max(\hat{\omega} \cdot \hat{n}_X, 0)}{\pi ||Y - X||^2}, \quad (4)$$

where $\hat{\omega} = \frac{Y - X}{||Y - X||}$. The rightmost fraction is the form factor and both E and B have units of W/m^2. This approximation is accurate

when $||Y - X||^2 \gg A_Y$, i.e., when $\hat{\omega}$ is nearly constant across the patches.

From the irradiance, an additional scattering event at X with reflectivity ρ gives the outgoing radiosity at X,

$$B(X) = E(X) \cdot \rho_X \cdot \text{boost}(\rho_X), \quad (5)$$

where choosing $\text{boost}(\rho_X) = 1$ conserves energy. BSDF scaling by

$$\text{boost}(\rho) = \frac{\max_\lambda \rho[\lambda] - \min_\lambda \rho[\lambda]}{\max_\lambda \rho[\lambda]}, \quad (6)$$

where λ = wavelength or "color channel", selectively amplifies scattering from saturated surfaces to enhance color bleeding. This is useful for visualization in our results. It is also aesthetically desirable in some applications, e.g., this was used in the *Mirror's Edge* [Halén 2010] video game. Anecdotally, an art professor informs us that it is a common practice by artists to artificially increase color bleeding in paintings as a proximity cue and to increase scene coherence.

The initial radiosity $B(Y)$ at each patch Y is simply the Lambertian shading term under direct illumination (with boosting). Each patch is represented by the depth and normal at a pixel. Its area A_Y is that of the intersection of the surface plane through the pixel with the frustum of the pixel (we simplify this below). To compute multiple scattering events, simply apply equations 4 and 5 iteratively.

4.2.1 Sources of Error

There are three sources of error in our radiosity approximation when applied to a deep G-buffer:

1. It overestimates reflection by assuming all Y are visible at X.

2. It underestimates reflection because surfaces not in the frame buffer are not represented.

3. Some patches may be close together, violating the distance assumption of equation 4.

We limit the first source of error by sampling only within a fixed-radius (e.g., 2m) world-space sphere about X. We limit the second one by considering two depth layers and including a guard band around the viewport. We mitigate the third source by clamping the maximum contribution of a patch.

Surfaces parallel to eye rays, backfacing to the camera, and behind the camera remain unrepresented in the two-layer framebuffer. This error is inherent in our approach and represents a tradeoff of using a fast depth peel instead of a robust but slower voxelization.

Although it is possible to construct arbitrarily bad cases, we hypothesize that the surfaces represented in a two-layer framebuffer with a guard band may be the most perceptually important surfaces to represent. That is because those are the surfaces observed by the viewer as well as those likely to be revealed in adjacent frames. For a complex environment, the viewer may be more sensitive to illumination interactions between observed objects than between those not currently in view.

Finally, an image without GI has error because it underapproximates lighting. The key questions for an application are whether some plausible dynamic global illumination with bias is better than none at all and whether the artifacts of this approach are acceptable in a dynamic context.

4.3 Spatial Sampling Pattern

For both ambient occlusion and radiosity, we choose sample taps as described in McGuire et al. [2012b] but optimize the parameters to minimize discrepancy in the pattern. We place s direct samples in

a spiral pattern, spatially varying the orientation about each pixel. Let (x, y) be the pixel coordinate of the patch we are shading, and let r' be the screen-space sampling radius. Sample i is taken from texel $(x, y) + h_i \hat{u}_i$, where

$$
\begin{aligned}
\text{Let } \alpha_i &= \tfrac{1}{s}(i + 0.5) \\
h_i &= r' \alpha_i; \quad \theta_i = 2\pi\alpha_i\tau + \phi \quad (7) \\
\hat{u}_i &= (\cos\theta_i, \sin\theta_i). \quad (8)
\end{aligned}
$$

We rotate the entire pattern for each output pixel (x, y) by angular offset ϕ to obtain relatively unique samples at adjacent pixels, which will then be combined by a bilateral reconstruction filter. Constant τ is the number of turns around the circle made by the spiral.

The original Scalable Ambient Obscurance paper reported manually chose $\tau = 7$ and $s = 9$. We present a general method for generating good values of τ for any number of sample taps. Discrepancy, as introduced to the graphics literature by Shirley [Shirley 1991], is widely used as a measure of how well distributed a set of sample points are over the unit square.

If you have a set of points on the unit square, you can define the local discrepancy of any subrectangle as the absolute value of the difference between the area of the subrectangle and the number of sample points inside the subrectangle divided by the total number of points (i.e., how accurately you can estimate an integral on the subrectangle using only the sample points.) The discrepancy is then defined as the supremum of the local discrepancies of all rectangles with one corner at the origin. This definition allows for a simple $O(n^2)$ algorithm to compute the discrepancy. To make discrepancy robust to 90-degree rotations of the sampling patterns, the definition can be changed to be the supremum of the local discrepancies of all subrectangles at any location.

We can still compute that relatively efficiently (since this is an offline process) by modifying the simple algorithm, which makes it $O(n^4)$: loop over all pairs of points that share their coordinates with points in the set (the x and y values of a given point can come from separate points in the sample point set), calculate the local discrepancy of the box defined by the pair, with and without the edges included, and take the maximum of them all.

We make a natural modification of discrepancy for point sets on the unit circle by replacing the boxes with annular sectors. Because the sampling pattern we use is a simple spiral of radius r, samples are distributed with density $1/r$. Thus, instead of sampling with respect to area, one should sample with respect to weighted area for a density distribution of $1/r$. For an annular sector this is the product of the radial length and the angle subtended and is equivalent to sampling on the side of a cylinder.

We optimized the pattern by exhaustively searching for the discrepancy-minimizing integer values of τ for given sample count. The optimal values for 1 through 99 samples are given in listing 3. We format that matrix as a C array for convenience in copying it to an implementation.

We note that the manually-tuned constant from the original paper coincides with our minimal discrepancy values. That paper apparently chose its constants well by eye, but manual tuning would not be practical for the larger parameter space that we consider. We choose the MIP level m_i of the sample tap:

$$
m_i = \lfloor \log_2(h_i/q) \rfloor \quad (9)
$$

Constant q is the screen-space radius increment at which we switch MIP levels. We chose it to maintain texture cache efficiency instead of sample accuracy.

As is common practice, we compensate for the relatively small number of samples with a bilateral reconstruction blur. We extend the previous work's reconstruction with a normal weight and plane weight to prevent blurring across most surface discontinuities.

```
// tau[n-1] = optimal number of spiral turns for n samples
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.

4.4 Using Information from Previous Frames

Our algorithm incorporates information from previous frames for radiosity in two ways. First, it estimates **multibounce** radiosity by converging to n-bounces over n frames. Second, it applies **temporal smoothing** to reduce sampling noise using a moving average of previous frames.

While our AO gives high quality results under a small number of samples and 1-bounce radiosity is acceptable, to compute n-bounce radiosity one needs a very large number of samples since each bounce integrates a full hemisphere. To reduce the number of samples required for a visually compelling result, we instead compute radiosity incrementally, advancing by one scattering event per frame. We track separately the direct illumination result and (infinite) additional bounces, reverse-reprojecting visibility tests so that they can be performed against historical values and applying a temporal envelope to limit propagation of error. Of course, AO can be temporally filtered as well to reduce the number of samples for that pass even further, as was done in *Battlefield 3*. We did not find this necessary in our experimental system where the radiosity computation dominates the AO.

4.4.1 Multibounce Radiosity

Our algorithm gathers radiosity due to the 2nd and higher order scattering from the previous frame's final indirect irradiance buffer E_{t-1}, reverse-reprojecting sample locations to account for object and camera motion, and using it to calculate the initial radiosity buffer to feed into equation 4. This allows n bounces for the cost of one per frame. We scale previous bounces by $p \in [0, 1]$ so that they decay over time when lighting conditions change. This decay intentionally underestimates illumination, which is compensated for with a dim environment map ambient term. Multiple iterations quickly converge, see figure 4.

The deep G-buffers store information about points that are unseen in the final image but which may contribute to radiosity. We've already shown the impact of considering direct illumination's contribution from each layer to 1-bounce radiosity. For multi-bounce radiosity, the contribution of second-and-higher order bounces from the second layer is less significant than the contribution of direct illumination. However, it can make a perceptible impact on the final
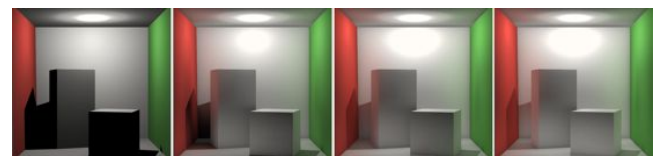

Figure 4: From left to right: Direct illumination only, one bounce indirect, two bounce indirect, converged multiple bounce indirect.

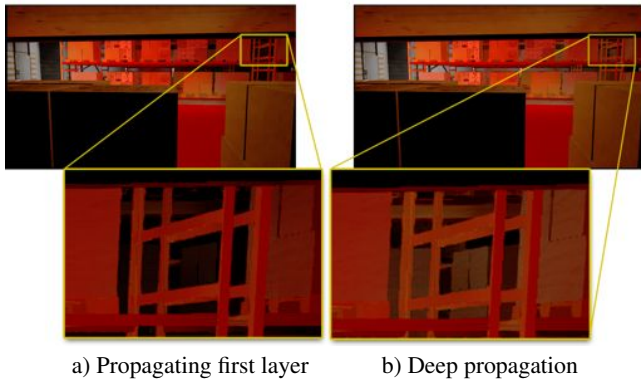a) Propagating first layer   b) Deep propagation

Figure 5: The two subfigures above depict *Warehouse* with multi-bounce radiosity computed on a deep G-buffer. The scene has high depth complexity and experiences direct illumination through sky-lights that scatters off the red floor. The callout boxes show a detail of the third aisle from the camera, visible through shelves. Gathering from two layers but propagating radiosity only within the first layer fails to light objects visible in the distance in the figure on the left. Propagating radiosity on both layers improves result by lighting even objects in the background, as shown on the right.

Scene	Source	Tris	Chars	Meshes
Op925	*Battlefield 3*	2.8M	24	50
Dockside	*Call of Duty: Black Ops 2*	2.3M	0	13
Sponza	Real-time benchmark	650k	0	56
San Miguel	Offline benchmark	5.9M	0	1196
Grass	turbosquid.com	180k	0	6
Office	g3d.sf.net	10k	0	17
Warehouse	turbosquid.com	640k	32	89
Old City	turbosquid.com	1.15M	0	100
Kitchen	turbosquid.com	370k	0	77

Table 1: Test scenes used in this paper.

image in scenes with high depth variance and high depth complexity, see figure 5.

Our implementation is bandwidth limited by the samples gathered. Using the same samples to simultaneously compute indirect irradiance on two layers thus incurs no observable runtime cost because the increase in write bandwidth is proportionally negligible and the compute cost is hidden under the memory cost.

Gathering the second through nth light bounce from a reprojected frame is robust because disoccluded regions (the pixels for which reverse-reprojection does not find a corresponding sample in the previous depth buffer) do not appear directly in screen space. The irradiance gather filter is somewhat similar to a huge blur kernel, so unless a disoccluded region affects a large fraction of the entire image its impact on the result is limited, see figure 6.

Maintaining two layers of course reduces the impact of disocclusions significantly, making it less likely that one will affect a sizable portion of the entire image. Figure 7 extends the previous *San Miguel* example to show that under typical motion, where objects traverse a small percentage of the frame per picture in scenes with moderate depth complexity, the second layer can often fill the disocclusions.

4.4.2 Temporal Filtering

Our algorithm reverse-reprojects and blends E_{t-1} by factor α into the current frame final E_t buffer at each pixel (i.e., without gathering) to compute an exponential weighted moving average that reduces sample variance. When reverse reprojection fails at a pixel, we simply zero the weight for the previous frame. With a sufficiently high α value, varying the tap locations does not result in temporal flickering, and can greatly improve the smoothness of the results, even for very low number of taps. See figure 15.

4.5 Reflection Ray Tracing

Sousa et al. [2011] describe an efficient method for screen-space ray tracing with few samples. We apply the same idea by simply ray marching along the reflection ray through both depth buffer layers and treating each pixel as having the minimum scene thickness. If the ray hits a texel in the depth buffer layers, then we use that texel's outgoing radiance as the incident radiance at the mirror surface and apply the source pixel's scattering function. If the ray fails to hit a surface before some arbitrary maximum distance (we use 4 meters in all of our results), then we fall back to the environment map value in that direction. Using MIP-maps to speed the tracing in the same manner as for AO samples, one can easily sample tens of texels per pixel.

Listing 4 in the appendix gives our full GLSL shader implementation for the screen-space ray trace portion of this computation. In one location it uses a compile-time FOR-loop pragma to expand the two layer test in a convenient way (this could then be extended to multiple layers easily). If applying our code without our preprocessor (which is available as open source in the G3D Innovation Engine), the loop must be unrolled by hand.

5 Results

We report representative experimental results that particularly consider the following issues for single-pass **generation** of deep G-buffers (which we have taken to mean "with minimum separation between layers" throughout this paper), and their application to **global illumination**:

1. Despite targeting hypothetical future hardware, how does single-pass deep G-buffer generation performance compare with depth peeling on a current high-end GPU? (table 2)

2. What is the performance impact of deep G-buffer input versus a single layer for global illumination? (table 3)

3. Can as few as two layers make a significant contribution to the appearance of scenes with high depth variance and complexity? (figures 8, 9, 18, 19)

4. How sensitive are complex scenes to having a single minimum-separation constant? (figure 9 and videos)

5. How do varying parameters affect the temporal filtering? (figures 15,17)

6. What is the impact on global illumination of disocclusion under reprojection during camera or object motion? How significant is the second layer at mitigating artifacts? (figures 6, 7)

7. Deep G-buffers are inherently view dependent. How well can one light large areas of complex scenes that receive no direct illumination? (figures 1, 11, 12, 14, 19)

5.1 Scenes

We constructed the test scenes described in table 1 from video game, stock (e.g., TurboSquid.com), and benchmark assets. We

(a) E_{t-1} after 100 frames without camera motion.
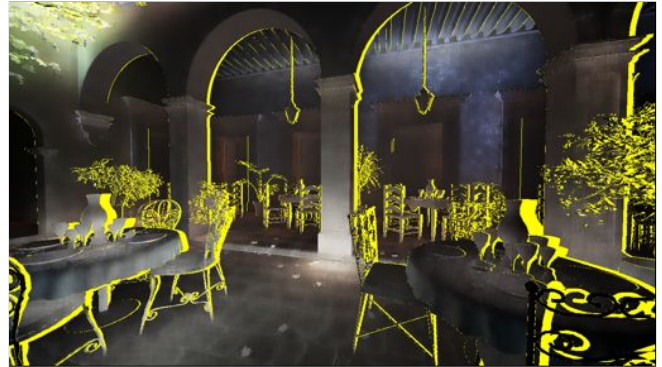
(b) E_{t-1} after 100 frames at a different camera position and reprojected to match the camera from (a). Bright yellow shows disocclusions, which we intentionally do not fill from the second layer in this experiment.

(c) E_t computed from direct illumination and E_{t-1}. The indirect irradiance is converged, so this is identical to (a).

(d) E_t computed from direct illumination and the reprojected E_{t-1} in (b), treating disoccluded pixels as black. Gathering during the indirect irradiance pass distributes the impact of disocclusion.

(e) Difference of (c) and (d) scaled by 5. This is the error introduced by reprojection.

(f) Final shaded image using the reprojected result from (d).

Figure 6: Because the irradiance operator has a wide kernel, the net impact of disocclusions can be low. In these images we approximated indirect irradiance in disoccluded regions as zero to exacerbate the error; using an ambient constant instead further limits their impact.

(a) Dark green: disocclusions filled by the second layer. Yellow: remaining disocclusions after considering both layers.

(b) Indirect irradiance buffer E_t computed from (a).

(c) Remaining error ($\times 5$) in the indirect irradiance buffer E_t

Figure 7: Reprojection error decreases by using the second layer.

Scene	Radiosity (Good)	Radiosity (Fast)	AO	Ray trace
Sponza	6.7/10.0 ms	3.9/5.9 ms	2.6/2.7 ms	N/A
Dockside	7.4/14.1	3.9/6.0	3.2/3.4	
Op925	8.0/13.5	4.1/5.7	2.5/2.7	
Old City	6.9/12.1	3.8/5.5	2.7/2.9	
Kitchen	5.1/7.3	3.3/4.9	2.9/3.0	1.3/1.6 ms
San Miguel	9.6/12.8	3.6/4.9	2.5/2.7	1.0/1.2

Table 3: Time for screen-space passes of various illumination operations at 1920×1080 on NVIDIA GeForce Titan.

added skeletally-animated meshes to model the GPU workload of not only the characters but also dynamic trees, machines, vehicles, and other objects (e.g., windmill sails, gears) found in a typical scene, as well as rigid bodies such as crates.

The two actual game scenes are representative of game geometry for quality evaluation but not for performance. That is because the way that the developers exported them for us created an unrepresentative number of draw calls. Dockside contains many small meshes with simplified materials, making it overly expensive to submit to the GPU. Op925 is a single giant draw call with no materials, making it overly inexpensive.

San Miguel has a large number of high-definition materials, and contains a reasonable number of meshes for that content. However, that content is substantially more complex than typical real-time assets, so it is a good predictor of future, but not current, real-time workloads.

5.2 Single-Pass Layer Generation

Table 2 shows the time to compute the layered representation. The new single-pass methods are faster than depth peeling, that is, less than twice as expensive as submitting the scene geometry twice. With prediction the cost increases, a topic that we discuss in our conclusions.

5.3 Global Illumination Effects

Table 3 reports the time to compute ambient occlusion, radiosity, and specular reflection by screen-space ray tracing for our test scenes. Each entry contains two times separated by a slash. The first time is for shading using a single layer of input. The second time is for shading using a two-layer deep G-buffer. In most cases, amortization allows image fidelity to increase disproportionally to rendering time, so the second time is often much less than twice the cost of the first. We only report ray trace time for *Kitchen* and *San Miguel* because the other scenes do not contain specular surfaces.

5.3.1 Ambient Occlusion

Figure 8 shows the difference between one depth layer and two layers with a minimum separation for computing AO. With a single layer, occlusion is underestimated at all areas highlighted in red in the left image. With a second layer but no minimum separation, it is still missing at the areas highlighted in yellow. The right image shows the final two-layer result. The difference is even more striking in motion, as shown in the supplementary video results. There, the missing occlusion appears as a light halo around depth discontinuities that is revealed as objects move past each other due to motion parallax.

Even with only two layers, we observe that AO is temporally stable in the presence of high depth complexity and gives good results for a variety of minimum separation constants for scenes like *Grass* (figure 9).
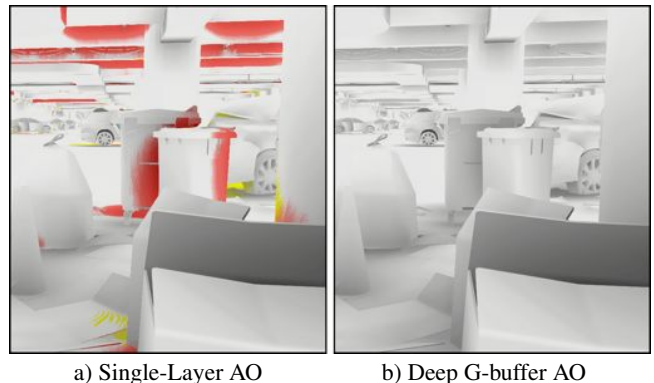

a) Single-Layer AO   b) Deep G-buffer AO

Figure 8: Ambient occlusion in the *Op925* parking garage from the game *Battlefield 3*. Both multiple layers *and* minimum separation are required for a robust result shown in (b). Red: occlusion under-estimated by a single layer. Yellow: AO still underestimated if using two layers without a minimum separation.

5.3.2 Radiosity

Figures 10, 11, 12, 13, and 14 demonstrate the quality of screen-space radiosity, both with and without multiple bounces. In all cases screen-space radiosity is a substantial quality improvement over flat ambient or environment map indirect terms.

Figure 15 demonstrates the effectiveness of temporal filtering at removing high frequency noise in a static environment, while the supplementary videos demonstrate its effectiveness at removing temporal flicker in dynamic environments. Temporal filtering is susceptible to certain artifacts, including erroneous lighting from

	1-Layer		2-Layer G-Buffer via...		
Scene	G-Buffer	Depth Peel	Our 'Previous'	Our 'Reproject'	Our 'Predict' and 'Delay'
Sponza	4.5 ms	9.6 ms	8.4 ms	8.4 ms	10.5 ms
Dockside	12.7	26.4	19.4	19.6	26.0
Op925	4.2	9.0	11.7	11.7	22.5
Kitchen	3.4	6.5	5.8	5.8	8.1
Old City	9.9	22.0	15.3	15.5	20.8
San Miguel	25.1	51.6	33.1	33.4	60.1
Grass	1.6	3.8	4.0	4.0	6.9
Office	0.4	1.3	0.7	0.7	1.1
Warehouse	9.3	16.4	13.0	13.4	17.4

Table 2: Comparison of the time to compute a single G-buffer to methods for producing a two-layer deep G-buffer, measured on NVIDIA GeForce Titan at 1920×1080.

Figure 9: Deep G-buffer AO using only two layers remains surprisingly coherent under motion, even for high depth complexity (see video results). It is also relatively insensitive to the exact minimum separation constant.

past lighting environments (see figure 16), and ghosting from reproduction error (see figure 17). In both cases the artifact can be mitigated by choosing relatively low values of α; all images and videos using temporal filtering in the paper and its supplements use $\alpha \leq 0.9$ unless specifically demonstrating possible artifacts.

Using a deep G-buffer adds significant robustness to screen-space radiosity, specifically in mitigating severe underestimates of indirect light, as seen in figure 18 and figure 19.

5.3.3 Reflection

Figure 20 shows screen space reflection of trees in a lake with a two-layer deep G-buffer, rendered with a 100-tap ray march through the depth buffer for both layers. The ripples in the lake are displaced geometry, not a normal map or distortions of a separately rendered image. Figure 21 shows a detail of the lake taken from near the base of the thick tree trunk. When rendered with a single layer, areas of high depth complexity miss reflected surfaces that are captured in the second layer (as shown in the center image).

Figure 22 demonstrates the gameplay impact of robust reflection. A character is walking through a starbase. The reflective crates in (a) use one layer of screen space reflection, so they can only reflect objects that are visible to the camera. Adding a second layer in (b) allows the crates to reflect objects around the corner, including the approaching monster which won't be visible to the camera for several frames, as shown in (c). One-layer screen-space reflection is a rendering trick to make objects look shinier; two-layers makes it a tactical tool for players.

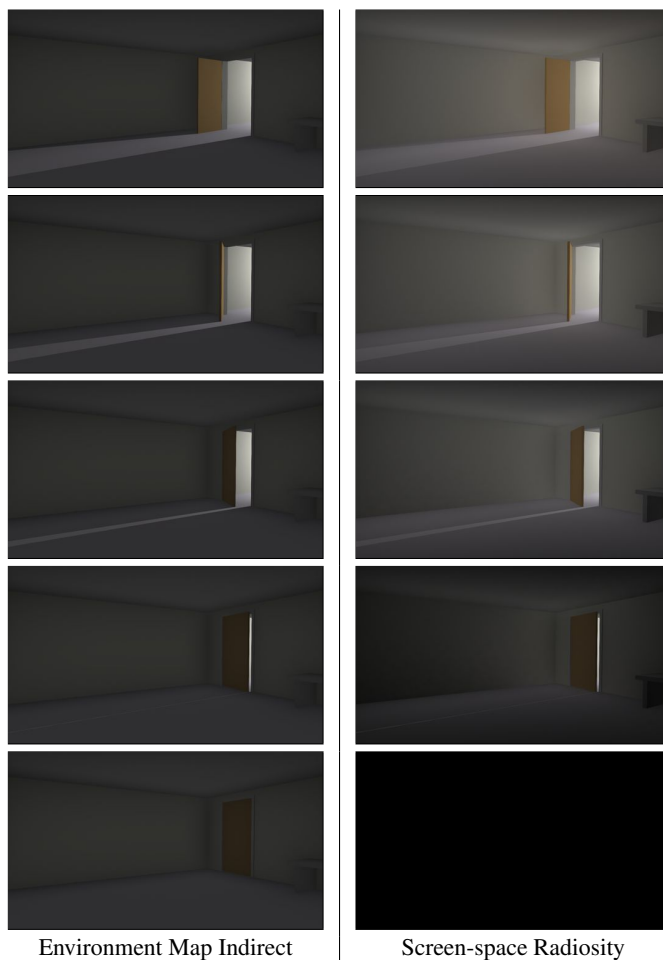Environment Map Indirect   Screen-space Radiosity

Figure 10: Screen-space radiosity can capture simple dynamic illumination phenomena well compared to a static environment map probe. We show a completely unlit room with a door to a very bright adjacent door as the door closes through multiple frames of animation.

(a) No temporal filtering.

(b) Temporal filtering with $\alpha = 0.9$.

Figure 15: Temporal filtering smoothes the noise due to undersampling in (a), producing the result in (b) without loss of spatial high-frequencies. The top row are the final indirect irradiance buffers, the bottom row are the final images. Both sets of images were produced with 3 taps per pixel per frame, with the taps randomly rotated each frame.
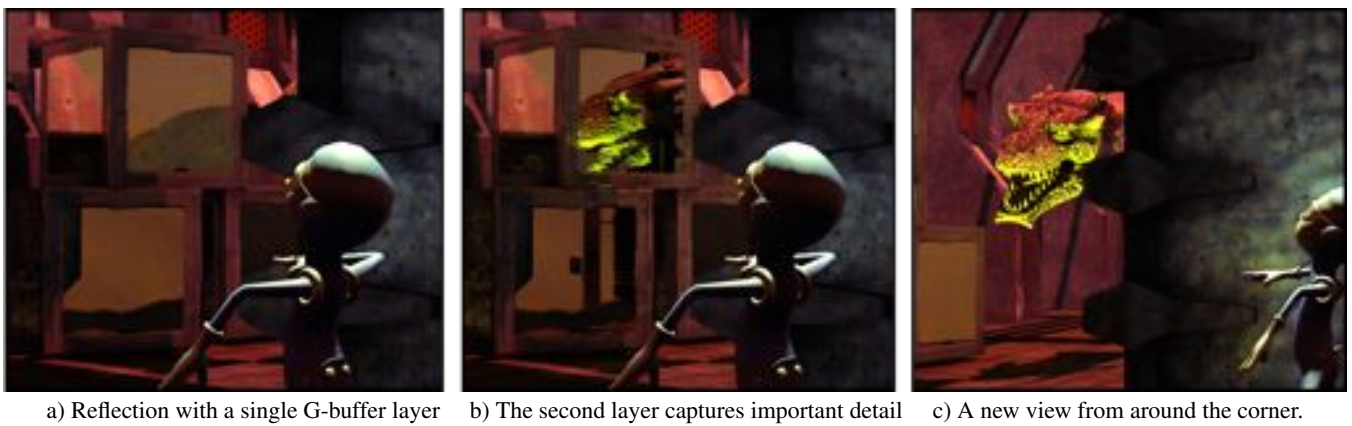

a) Reflection with a single G-buffer layer   b) The second layer captures important detail   c) A new view from around the corner.

Figure 22: Screen-space mirror reflection tracing is more robust with two layers. In this case, the monster around the corner does not appear in the shiny crates when using a single layer for reflection. The deep G-buffer's second layer captures it, which would be important for gameplay.

Figure 11: *Top: Sponza* with direct illumination and ambient occlusion of an environment probe, representative of typical real-time lighting today. *Bottom*: Replacing the environment light probe with screen-space radiosity brings out color bleeding and increases lighting contrast.

6 Conclusions

We've introduced single-pass deep G-buffer generation and demonstrated its viability even on today's hardware. Under our assumption of a faster geometry stage (and the implicit assumption of more efficient layered framebuffer rendering in the future), this technique will perform radically better and we hope that work such as this will present compelling use cases for the co-development of such hardware and software. We've also explored some applications to robust screen-space approximation of indirect illumination, putting some previous work and screen-space GI ideas that are currently topical in the field into a traditional radiometric framework and showing how deep G-buffers improve their sampling.

The three elements of our title are all essential for this application: multiple layers increase robustness, but must be generated with minimum separation, and *single-pass* generation is essential for performance. Screen-space GI methods are too viewer-dependent for most scenes without both multiple layers and separation between them, and generating multiple layers or views in is not viable for many applications without a single-pass method like the one we describe.

All of our single pass methods can generalize from 2 to N layers of a G-buffer, but the Prediction Variant requires rendering $2N - 1$ layers per frame ($N - 1$ being depth-only). The Reprojection (and inferior Last Frame) methods require rendering only N layers.

Interesting future work includes applications of single-pass deep G-buffers or color buffers to post-processed motion blur, depth of field, and reprojection for nearby views. Reprojection is particularly interesting for the case of reducing latency in cloud-based rendering and in rapidly generating stereo image pairs for virtual reality.

Figure 12: *Top:* 1-bounce radiosity ($p = 0$) with a spotlight on the green banner in *Sponza*. *Bottom:* The converged result using $p = 1$.

We were surprised by the image quality possible from only screen-space information, since screen-space input obviously depends on the viewer and misses much of the information about the scene. We originally hypothesized that world-space ray tracing or light probes would be required to supplement the screen-space data for visually compelling results. We now find that the screen-space data combined with static environment maps may suffice for current game standards. We speculate that one reason that deep screen-space techniques work well is that the lighting is consistent with what the viewer currently sees (and is about to see due to layers and guard bands). That is, the results are self-consistent even though they are not consistent with the true model, for which humans are likely unable to anticipate correct complex light transport.

A natural question is whether including a reflective shadow map [Dachsbacher and Stamminger 2005] in the gather pass from the direct illumination would justify increasing the cost of shadow map generation. Shadow rendering typically receives a throughput boost in hardware up to 300% because it is depth-only, avoiding the cost of launching fragment shader threads as well as the bandwidth and computation cost of shading. Deep G-buffer generation also performs some additional overdraw compared to a normal geometry buffer pass, however it operates at a point in the pipeline that does not have such an extremely optimized alternative and is often at lower resolution than a shadow map. We think that the cost of rendering a reflective shadow map will likely be justified and plan more experiments evaluating the impact on robustness.

Consider deep G-buffer generation and screen-space techniques in the context of the gaming industry's current target platforms. Many of today's important midrange gaming platforms, including mobile devices and recent game consoles (e.g., the Playstation4), do not have the computational power for true dynamic global illumination. Therefore we consider it possible that screen-space techniques like those that we've demonstrate will become pervasive and continue to dominate through the end of the decade. Screen-space

Figure 13: *Top:* 1-bounce radiosity ($p = 0$) in *San Miguel. Bottom:* The converged result using $p = 1$.

AO and some screen-space reflection have already followed this arc. Single-pass deep G-buffer generation on a current game console may not be desirable in the way that it will likely soon be on newer GPUs, but we expect that even with two-pass depth peeling or a single layer screen-space GI may soon become a common approximation.

Looking to higher-end platforms and farther out in time, we expect approaches like voxel cone tracing and photon mapping to become viable for real-time graphics. Per-pixel path tracing may even become viable. However, true world-space GI algorithms inherently do not scale well and have high variability in frame times. That is why even offline film rendering heavily employs render caches, compositing, and screen-space post-processing. So, true GI methods are perhaps most interesting for real-time in cases where they operate at reduced spatial resolution and complexity can be bounded. In parallel with that observation, we also note that the advent of hardware tessellation and displacement mapping did not make older methods such as Blinn's bump and texture mapping obsolete–instead, those techniques are used for fine-scale detail while more accurate methods manage larger scales. Connecting these two arguments, we speculate that over a decade from, now true GI will operate at something like half-meter scale and techniques like screen-space AO and GI will still be employed for the finer scales. Using multiple depth layers to keep those fine-scale results robust will then likewise continue to be important.
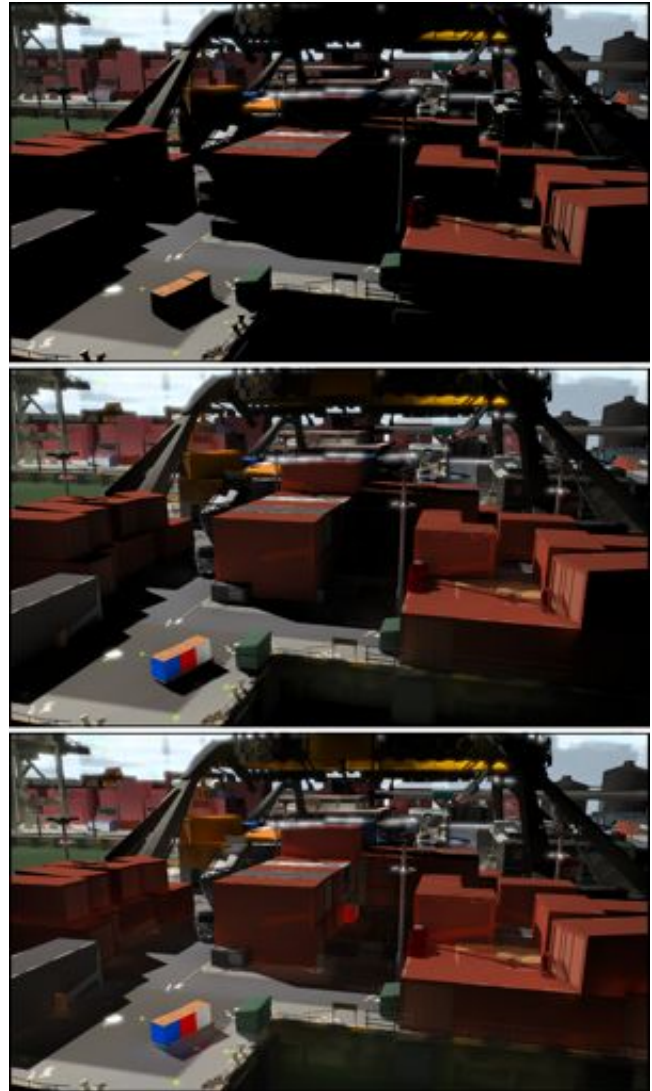
Acknowledgements

Figure 14: *Dockside* from the *Call of Duty: Black Ops 2* game, shown with only low-frequency textures. From top to bottom: Direct illumination, single-bounce radiosity, and converged multiple bounce radiosity.

Figure 16: *Sponza* with a spot light that has just moved from the banner to the ground, rendered with a single bounce ($p = 0$). *Left:* $\alpha = 0.99$; the strong green bounce lighting from the previous lighting environment is a prominent artifact because α is too large and prevents the result from adapting to the new lighting conditions. *Right:* $\alpha = 0.9$; most of the erroneous lighting is gone.

Figure 17: Animation frame from *San Miguel* in which the camera rapidly orbits the table. *Left:* $\alpha = 0.98$ creates ghosting artifacts due to reprojection error in temporal filtering. *Right:* $\alpha = 0.9$ reduces the ghosting artifacts. This value is still higher than we would recommend for most scenes.
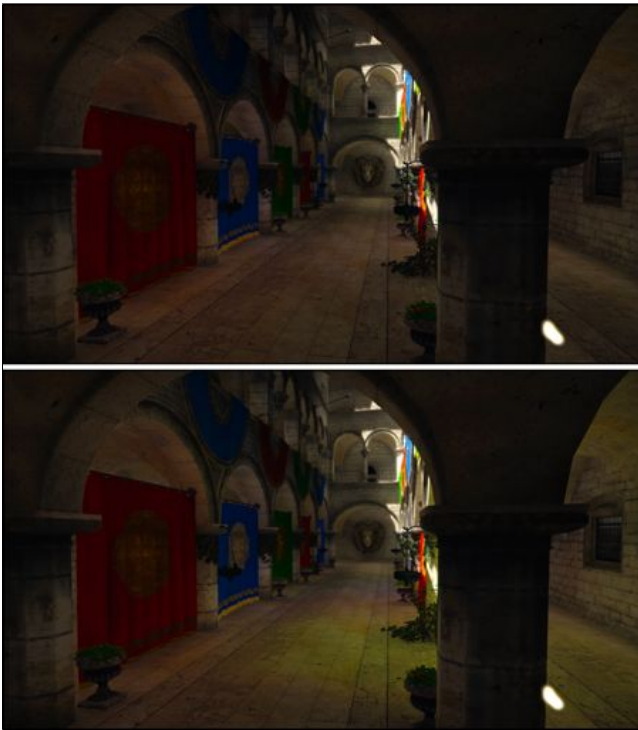

Figure 18: *Top:* Radiosity in *Sponza* using a single G-buffer. Much of the color bleeding onto the floor is missing because the input is too dependent on the viewpoint. *Bottom:* Improved radiosity using a deep G-buffer due to the second layer capturing additional information about the scene.
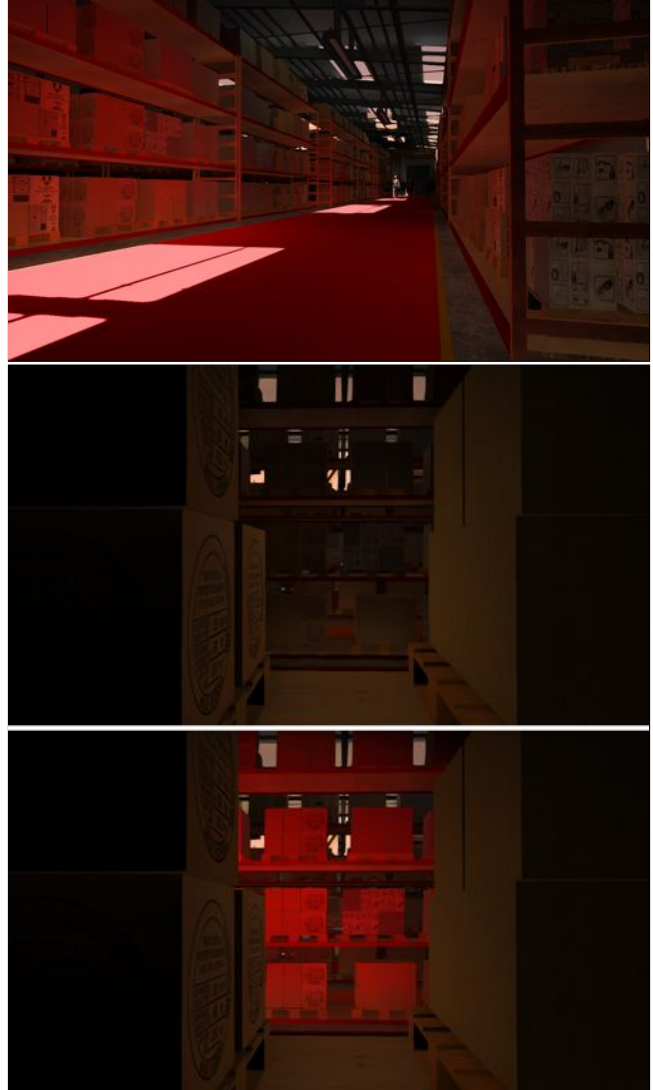

Figure 19: *Top*: *Warehouse*, looking down an aisle. Sunlight passes through skylights and scatters off the red floor, tinting the boxes and shelves. *Middle*: Looking through a shelf across an aisle, using only a single layer. The indirect light from the floor is incorrectly absent. *Bottom*: A deep G-buffer captures the floor, which does not have primary visibility. The illumination is more consistent with the alternative viewpoint in the top row.

Figure 20: Screen-space reflection from a two-layer deep G-buffer in *Lake*.

Figure 21: *Left:* Reflection with only one layer. *Center:* With both layers. *Right:* The difference, showing the errors corrected by the second layer.

References

AMD, 2013. Mantle and Radeon R9 press release, 10. http://amd.com/us/press-releases/pages/amd-unleashes-r9-2013oct07.aspx.

BAVOIL, L., AND MYERS, K. 2008. Order independent transparency with dual depth peeling. Tech. rep., NVIDIA.

BAVOIL, L., CALLAHAN, S. P., LEFOHN, A., COMBA, JO A. L. D., AND SILVA, C. T. 2007. Multi-fragment effects on the gpu using the k-buffer. In *I3D'07*, ACM, New York, NY, USA, 97–104.

BOLZ, J., 2009. OpenGL bindless extensions. http://developer.download.nvidia.com/opengl/tutorials/bindless_graphics.pdf.

BRAINERD, W., 2013. Game engine profiling results on playstation4 at activision maine, October. Personal communication.

BUKOWSKI, M., 2013. Game engine profiling results on nvidia geforce 670 at vicarious visions, October. Personal communication.

CARPENTER, L. 1984. The A-buffer, an antialiased hidden surface method. *SIGGRAPH'84 18*, 3 (Jan.), 103–108.

CHAPMAN, J., 2011. Deferred rendering, transparency & alpha blending, January.

COHEN, M. F., AND GREENBERG, D. P. 1985. The hemi-cube: a radiosity solution for complex environments. *SIGGRAPH'85 19*, 3 (July), 31–40.

CRASSIN, C., AND GREEN, S. 2012. *Octree-Based Sparse Voxelization Using The GPU Hardware Rasterizer*. CRC Press.

DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In *I3D'05*, ACM, New York, NY, USA, 203–231.

DEERING, M., WINNER, S., SCHEDIWY, B., DUFFY, C., AND HUNT, N. 1988. The triangle processor and normal vector shader: A vlsi system for high performance graphics. *SIGGRAPH Comput. Graph. 22*, 4 (June), 21–30.

ENGEL, W., Ed. 2013. *Introducing the Programmable Vertex Pulling Rendering Pipeline*. CRC Press, 21–38.

EVERITT, C. 2001. Interactive order-independent transparency. Tech. rep., NVIDIA.

HACHISUKA, T. 2005. High-quality global illumination rendering using rasterization.

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

JANSEN, J., AND BAVOIL, L. 2010. Fourier opacity mapping. In *I3D'10*, ACM, New York, NY, USA, I3D '10, 165–172.

LOKOVIC, T., AND VEACH, E. 2000. Deep shadow maps. In *SIGGRAPH'00*, ACM Press, New York, NY, USA, 385–392.

MARK, W. R., AND PROUDFOOT, K. 2001. The F-buffer: a rasterization-order fifo buffer for multi-pass rendering. In *Graphics Hardware*, ACM, New York, NY, USA, 57–64.

MCGUIRE, M., HENNESSY, P., BUKOWSKI, M., AND OSMAN, B. 2012. A reconstruction filter for plausible motion blur. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '12, 135–142.

MCGUIRE, M., MARA, M., AND LUEBKE, D. 2012. Scalable ambient obscurance. In *HPG'12*.

MCGUIRE, M., 2013. Game engine profiling results on nvidia geforce 660 at unknown worlds, October. Personal communication.

MYERS, K., AND BAVOIL, L. 2007. Stencil routed a-buffer. In *ACM SIGGRAPH 2007 sketches*, ACM, New York, NY, USA.

NEHAB, D., SANDER, P. V., LAWRENCE, J., TATARCHUK, N., AND ISIDORO, J. R. 2007. Accelerating real-time shading with reverse reprojection caching. In *Proceedings of the 22Nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, GH '07, 25–35.

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

POPESCU, V., LASTRA, A., ALIAGA, D., AND NETO, M. D. O. 1998. Efficient warping for architectural walkthroughs using layered depth images. In *IEEE Visualization*, 211–215.

RITSCHEL, T., GROSCH, T., AND SEIDEL, H.-P. 2009. Approximating dynamic global illumination in image space. In *I3D'09*, ACM, New York, NY, USA, 75–82.

SALESIN, D., AND STOLFI, J. 1989. The ZZ-buffer: A simple and efficient rendering algorithm with reliable antialiasing. In *PIXM'89*, 415–465.

SALVI, M., MONTGOMERY, J., AND LEFOHN, A. 2011. Adaptive transparency. In *HPG'11*, ACM, New York, NY, USA, 119–126.

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

SCHWARZ, M. 2012. Practical binary surface and solid voxelization with Direct3D 11. In *GPU Pro 3*, W. Engel, Ed. A K Peters/CRC Press, Boca Raton, FL, USA, 337–352.

SHADE, J., GORTLER, S., HE, L.-w., AND SZELISKI, R. 1998. Layered depth images. In *SIGGRAPH'98*, ACM, New York, NY, USA, 231–242.

SHANMUGAM, P., AND ARIKAN, O. 2007. Hardware accelerated ambient occlusion techniques on GPUs. In *I3D'07*, ACM, New York, NY, USA, 73–80.

SHIRLEY, P. 1991. Discrepancy as a quality measure for sample distributions. In *In Eurographics '91*, Elsevier Science Publishers, 183–194.

SINTORN, E., AND ASSARSSON, U. 2009. Hair self shadowing and transparency depth ordering using occupancy maps. In *I3D'09*, ACM, New York, NY, USA, 67–74.

SOLER, C., HOEL, O., ROCHET, F., AND HOLZSCHUCH, N. 2009. A fast deferred shading pipeline for real time approximate indirect illumination. Tech. rep., Institut National de Recherche en Informatique et en Automatique.

SOUSA, T., KASYAN, N., AND SCHULZ, N. 2011. Secrets of CryENGINE 3 graphics technology. In *SIGGRAPH '11 Courses*, ACM, New York, NY, USA.

SUCHOW, J. W., AND ALVAREZ, G. A. 2011. Motion silences awareness of visual change. *Current Biology 21*, 2, 140 – 143.

VARDIS, K., PAPAIOANNOU, G., AND GAITATZES, A. 2013. Multi-view ambient occlusion with importance sampling. In *I3D'09*, ACM, New York, NY, USA, 111–118.

YUKSEL, C., AND KEYSER, J. 2007. Deep opacity maps. Tech. rep., Department of Computer Science, Texas A&M University.

A Ray Tracing Implementation

Listing 4 is our GLSL implementation of ray-scene intersection against the surface defined by the deep G-buffer's depth layers, implemented by ray marching. The interesting parameters are:

- **depthBuffer01** Camera-space linear Z floating-point buffer, with R = layer 0 and G = layer 1.

- **clipInfo** Clipping plane information: $(z\_f == -\inf)$? $vec3(z\_n, -1.0f, 1.0f) : vec3(z\_n * z\_f, z\_n - z\_f, z\_f)$, where the near and far plane constants z_n and z_f are both negative values.

- **jitterFraction** A number between 0 and 1 used to offset the ray randomly at each pixel.

- **distance** The maximum distance to trace. On return, the actual distance traced.

- **hitTexCoord** The [0, 1] texture coordinate in the depth buffer of the hit location.

- **which** Index of the layer hit.

The function returns true if something was hit before the maximum distance or limiting number of iteration steps and false otherwise.

```
1   bool castRay2(vec3 csOrigin, vec3 csDirection,
2     mat4 projectionMatrix, sampler2D depthBuffer01,
3     vec2 depthBufferSize, vec3 clipInfo,
4     float jitterFraction, int numSteps,
5     const float layerThickness, inout float distance,
6     out vec2 hitTexCoord, out int which) {
7
8     // Current point on the reflection ray in camera space
9     vec3 P = csOrigin;
10
11    // Camera space distance for each ray−march step
12    float stepDistance = distance / numSteps;
13    float rayBumpDistance = 0.02;
14
15    // Off screen
16    hitTexCoord = vec2(−1, −1);
17
18    // Amount that P increments by for every step
19    vec3 PInc = csDirection ∗ stepDistance;
20
21    P += PInc ∗ (jitterFraction + 0.5) + epsilon ∗ csDirection;
22
23    int s = 0;
24    for (s = 0; s < numSteps; ++s) {
25      // Project the trace point P into texture space.
26      // Note that the projection matrix maps to [−1, 1]
27      // coordinates after homogeneous division
28      vec4 temp = projectionMatrix ∗ vec4(P, 1.0);
29
30      // texture space P: Homogeneous division and remap to [0,1]
31      vec2 tsP = (vec2(temp.x, temp.y) / temp.w) ∗ 0.5 + vec2(0.5);
32
33      // Pixel space P
34      ivec2 psP = int2(depthBufferSize ∗ tsP);
35
36      // Camera space z of the background at each layer
37      vec2 sceneZ = texelFetch(depthBuffer01, psP, 0).rg;
38
39      // This is the depth range that the ray covers at this sample
40      float intervalStartRayZ = P.z − PInc.z ∗ 0.5;
41      float intervalEndRayZ = P.z + PInc.z ∗ 0.5;
42
43      float rayZMin = min(intervalStartRayZ, intervalEndRayZ);
44      float rayZMax = max(intervalStartRayZ, intervalEndRayZ);
45
46      vec2 sceneZMin = sceneZ − layerThickness;
47      vec2 sceneZMax = sceneZ;
48
49      // Use a macro instead of a real FOR loop because we
50      // need to use a BREAK statement below
51  # for (int layer = 0; layer < 2; ++layer)
52      // If the point on the ray is behind the scene point at the same
53      // location... (As an optimization, break out of the containing
54      // loop here, but don't handle the result until outside the loop)
55      if ((rayZMax >= sceneZMin[(layer)]) &&
56        (rayZMin <= sceneZMax[(layer)]) &&
57        (sceneZ[(layer)] > −99999)) {
58        // Hit...or off screen
59        hitTexCoord = tsP;
60        which = (layer);
61        break;
62      }
63  # endfor
64
65      P += PInc;
66    }
67
68    distance = (s + jitterFraction) ∗ stepDistance;
69    return (hitTexCoord.y >= GUARD_BAND_FRACTION_Y) &&
70      (hitTexCoord.x >= GUARD_BAND_FRACTION_X) &&
71      (hitTexCoord.x <= 1.0 − GUARD_BAND_FRACTION_X) &&
72      (hitTexCoord.y <= 1.0 − GUARD_BAND_FRACTION_Y);
73  }
```

Listing 4: Screen-space ray tracing code in GLSL for two depth buffer layers.