CloudLight: A system for amortizing indirect lighting in real-time rendering

Cyril Crassin David Luebke Michael Mara Morgan McGuire Brent Oster Peter Shirley Peter-Pike Sloan Chris Wyman
NVIDIA

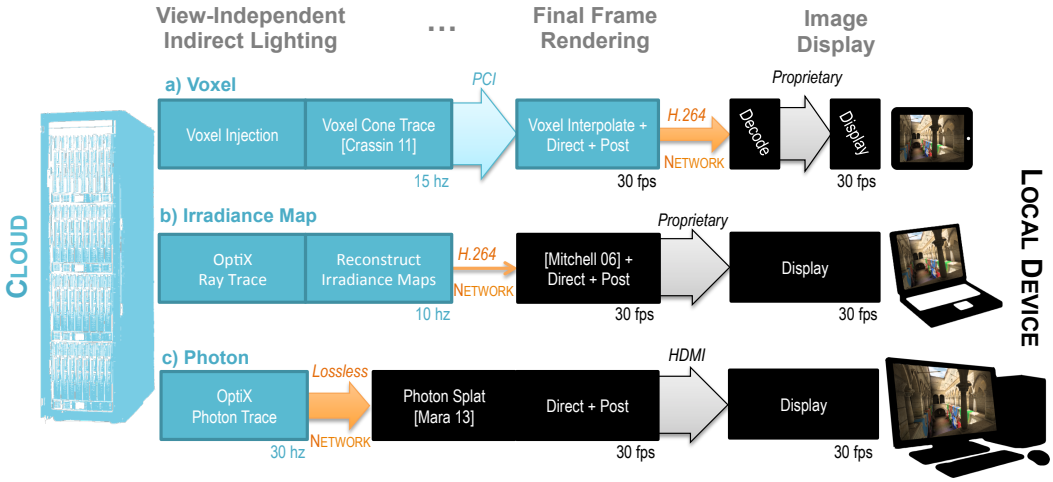

Figure 1: *The CloudLight pipeline, as implemented for three known global illumination algorithms: voxels, irradiance maps, and photons. Note the shift from cloud (blue) to local computation (black), suggesting use of different algorithms depending on the targeted user device. Arrow thickness indicates required bandwidth; note we always put the network at the narrowest point.*

Abstract

We introduce *CloudLight*, a system for computing indirect lighting in the Cloud to support real-time rendering for interactive 3D applications on a user's local device. CloudLight maps the traditional graphics pipeline onto a distributed system. That differs from a single-machine renderer in three fundamental ways. First, the mapping introduces potential asymmetry between computational resources available at the Cloud and local device sides of the pipeline. Second, compared to a hardware memory bus, the network introduces relatively large latency and low bandwidth between certain pipeline stages. Third, for multi-user virtual environments, a Cloud solution can amortize expensive global illumination costs across users. Our new CloudLight framework explores tradeoffs in different partitions of the global illumination workload between Cloud and local devices, with an eye to how available network and computational power influence design decisions and image quality. We describe the tradeoffs and characteristics of mapping three known lighting algorithms to our system and demonstrate scaling for up to 50 simultaneous CloudLight users.

1 Introduction

Image quality in video games has increased tremendously in recent years, but dramatic changes in the computing ecosystem pose challenges for continuing that rapid improvement. As client-server computing migrates into a multitude of consumers' everyday activities, it changes expectations for where and how people work and play. Today's servers are increasingly abstracted into *the Cloud*[1], collections of servers with reliability similar to a utility [Armbrust et al. 2010] that are similarly commoditized and abstracted, from both the consumer's and developer's perspective. With clients increasingly shifting to mobile phones and tablets, games and graphics face a new challenge. Today's most popular devices lack the computational horsepower to render advanced effects such as global illumination, yet consumers expect continually increasing graphics quality. Further, mobile devices (including laptops) are thermally limited and unlikely to reach the rendering performance of today's PCs by relying only on Moore's Law [Chang et al. 2010]. *Cloud graphics* offers a solution, replacing local client rendering with remote rendering [Koller et al. 2004; Chen et al. 2011; Choy et al. 2012; Shi et al. 2012; Manzano et al. 2012]. Such systems have many advantages beyond improved image quality (e.g., virtualization; lower costs and piracy) and have started appearing commercially [Lawton 2012]. Existing systems demonstrated the viability of Cloud graphics and provided inspiration for our work.

While design of rendering pipelines for PCs and consoles is reasonably well understood, design of Cloud pipelines is in its infancy. Nearly all prior works, including commercial systems, use a very simple offload approach: synchronously map rendering for each user's frame to a single server. (Remote offline-rendering systems like Autodesk360 [Dac 2013] also do this, but there latency is irrelevant so we we consider that beyond the scope of this real-time focused paper.) Full-frame remote rendering is relatively easy to deploy for legacy applications by simply running them within a virtual computing environment and streaming their output as video. However, it limits amortization and scalability, both critical requirements for economic Cloud deployments, and couples local device rendering latency to network latency. An attractive alternative is to make Cloud-aware pipelines that simultaneously compute results for multiple users. That presents new opportunities to tradeoff la-

NVIDIA Technical Report NVR-2013-001, July 2013.
© 2013 NVIDIA Corporation. All rights reserved.

[1] We capitalize Cloud throughout the text to emphasize this definition.

tency, bandwidth, and image quality, as well as leveraging local device computation to reduce perceived latency.

We introduce *CloudLight*, a new system for mapping complex indirect lighting algorithms onto existing and future Cloud architectures. In this paper, we map three known lighting algorithms onto a distributed infrastructure (shown in Figure 1) to demonstrate feasibility of shared client-server rendering and explore its tradeoffs. A key takeaway is that algorithms have varying bandwidth and latency requirements that change the ideal location of network links, which partition work between devices. We see this as a first step into a whole new distributed ecosystem with broad opportunities for new ideas and algorithms.

Distributing lighting computation over multiple machines is a natural extension of the parallelism already in multicore CPU ray tracers and massively parallel GPU rasterizers. It is also one that has been proposed in a few settings previously and deployed in some specialized contexts like multi-node ray tracers and VR CAVEs. While the idea is not novel, our investigation is an industry- and systems-oriented approach to that could be deployed at consumer scale on current and near-future hardware. The recent advances in global illumination algorithms and new emergence of Cloud-based graphics hardware, such as the , make this the right time to revisit. This paper reports answers to the high-level questions that we ourselves first asked around latency, scalability, and practicality. It then covers some of the pragmatic details of our evaluation of alternative strategies and observations about how existing strategies must be adapted for a distributed context.

Our contributions include developing and evaluating prototypes, demonstrating feasibility of Cloud indirect lighting for game-scale assets, and extending existing indirect light computation strategies to a concrete distributed setting based around commercially-deployed systems like OptiX, hardware video encoders, and GRID. We also identify and empirically examine trade-offs in mapping lighting algorithms to a Cloud pipeline. We explore different distributed architectures and algorithms, studying indirect lighting representations requiring computation on client devices (irradiance maps and photons) as well as optimizing communications between GPUs inside a server node (for voxels). Finally, we demonstrate scaling of CloudLight to 50 users per server node, allowing better quality global illumination for similar per-user costs. Despite these successes, our goal is not to present and advocate for a specific system. Instead, we wish to document what we've done in order to expose the underlying issues in the first, second, and third designs that we think that any team might reasonably investigate as Cloud lighting solutions. We believe that demand for such systems is inevitable and seek to address both "is this feasible?" questions as well as the natural followup from a practical standpoint.

2 Background

Our work touches a vast body of prior work in graphics and computer science. Thus our discussion here will be incomplete and we restrict ourselves to the most fundamental information.

The Cloud abstracts a collection of servers combined with some measure of reliability and transparency, allowing a view of computation much like a utility [Armbrust et al. 2010]. While this spins the age-old concept of server infrastructure, the nuance is important: it targets ordinary consumers, providing a user-friendly experience that "just works." For games, the Cloud includes a heterogeneous set of computers, connected by fast interconnect, that offload shared or overflow computations from user devices.

User devices include phones, tablets, consoles, and traditional desktop and laptop PCs. These clients connect via relatively limited networks (in bandwidth and latency), compared to links in a traditional graphics pipeline. While there is a continuum of user device power, most devices fall into three broad categories: low-power (e.g., phones and some tablets), medium-power (e.g., some tablets, laptops, and some PCs), and high-power (e.g., gaming PCs). While consumer preferences between these are unpredictable, we see the categories as intrinsically stable as power largely follows form factor and heat dissipation. Functionally, we assume low-power devices can stream video; medium-power devices can do basic rendering (e.g., z-buffer, direct light, texture mapping); and high-power devices can add moderately sophisticated work beyond basic graphics.

Computer networks have several well-studied characteristics that affect Cloud rendering. Importantly, both bandwidth and latency vary over time and geographical region. In many countries streaming video services are quite popular, empirically demonstrating sufficient bandwidth for at least moderate quality video. Network and video streaming services co-evolved into a complex and still-changing ecosystem [Rao et al. 2011]. Another widely used and growing application, two-way voice and video communications [Karapantazis and Pavlidou 2009], requires low latency rather than high bandwidth.

Cloud gaming systems already exist commercially. Existing systems just stream video to the client, and do not include indirect lighting. These systems resemble the software architecture of our voxel-based illumination algorithm in Section 3.2, albeit with lower quality visuals. Massively multiplayer games often use lightweight servers to moderate player interactions, requiring fairly low latency for smooth game play, although required bandwidth is small [Suznjevic et al. 2009]. Researchers have augmented video with extra data to hide latency and showed this trade of bandwidth for reduced latency practical in certain cases [Boukerche and Pazzi 2006; Shi et al. 2012].

Remote rendering has been broadly adopted (e.g., [Brodlie et al. 2004; Koller et al. 2004]). These systems typically stream video or progressive images, but a few send graphics commands [Paul et al. 2008]. While sending graphics commands effectively places a network between CPU and GPU, our system instead places network links between algorithmic components. The large data sets at servers used in scientific visualization applications have inspired work where both the server and local machine do work [Engel et al. 2000; Luke and Hansen 2002; Tamm et al. 2012], and this work shares high-level goals with our own but is very different in detail due to the very different datasets rendering goals. Remote industrial design systems compute indirect lighting, but as they focus on accuracy (rather than interactivity) latency considerations are less important [Denk 2011].

Parallel renderers explore similar issues to our Cloud pipeline. Parallel systems distribute global illumination over machines, but most emphasize utilization [Chalmers and Reinhard 2002], not latency and amortization over users. Thus, the design space and bottlenecks are sufficiently different that they do not extend well to interactive Cloud rendering.

Perception research has examined indirect lighting, showing it sometimes provides weak spatial and albedo estimation cues [Hu et al. 2000; Sinha and Adelson 1993]. Little objective work explores whether indirect lighting improves subjective viewer experience [Thompson et al. 2011], though the continuing push for global illumination in games and film suggests it is desirable. Because its perceptual role in improving visual quality for games is poorly understood, we focus on showing existing global illumination algorithms map to the Cloud rather than formally investigating perceptual questions. Our video shows illumination under varying latency,

and readers can draw their own conclusions.

Indirect lighting can be computed many ways, usually involving at least some ray tracing [Whitted 1980], particularly in batch applications [Dutre et al. 2003]. Ritschel et al.'s [2012] recent survey explores various techniques applied in interactive settings; see surveys by Dachsbacher et al.'s [2009; 2013] for general coverage of indirect light computation. We use three different approaches to cover a reasonable portion of the interactive global illumination design space and demonstrate a variety of algorithms can split work between clients and servers.

We note two theses with related material. Klionsky [2011] created a general distributed system and applied it to light field rendering and fluid simulation. Mara [2011] created an early prototype irradiance map and light probe remote rendering system that later led to the full CloudLight system described in this paper.

3 System

We aim to provide interactive 3D graphics experiences that exceed the abilities of a user's local device and enable amortization of the resulting rendering costs when scaling to many clients, e.g., in the context of multiplayer games. In this paper, we demonstrate CloudLight's feasibility on existing networks and hardware, and explore how the choice of global illumination algorithm affects system design and performance.

A conventional single-user interactive renderer computes and stores indirect light into a view-independent data structure, which is queried when rendering each viewpoint. The high cost of updating indirect light requires a powerful processor; even today, relatively few applications compute dynamic global illumination. In contrast, the cost of direct light is more modest. Desktops, laptops, tablets, and phones can all render basic direct lighting for scenes using well-known algorithms. With increasing consumer demand for entertainment on low power devices, splitting computation for these components between Cloud and user allows higher quality rendering on a range of client devices.

Important questions we investigate include: Do global illumination algorithms map well to distributed computation? What client bandwidth is needed to share Cloud-computed illumination? What latency does splitting a rendering pipeline introduce? Can local and global illumination be asynchronous without introducing objectionable latency? How does Cloud lighting scale with number of users? Overall, this provides a rich design space our system explores using three separate lighting algorithms (see Figure 1).

3.1 Hardware Architecture and Scene Characteristics

With Cloud rendering in its infancy, server hardware and infrastructure may change dramatically in upcoming years. Hardware imposes design constraints, so we make some assumptions to provide a concrete target. In particular, we assume servers contain multiple GPUs connected through a fast bus (e.g., PCIe). Each server has at least one high-end GPU with hardware-assisted H.264 video encoding. We assume consumer devices have sufficient network and compute resources to a least stream and decode a 1080p H.264 stream in realtime with less than 200ms of latency. While such networks are not available everywhere, similar bandwidth is already required by existing systems (e.g., home video streaming) and such latency is already feasible for many users. One variant of our system explores how to take advantage of increased local computation and network bandwidth.

We test our system on scenes that mimic asset type and complexity found in modern games. Today's game assets co-evolved with existing hardware and lighting algorithms, so their authoring often does not highlight indirect lighting. However, we believe our scenes provide a reasonable computational "stress test" and allow sufficient comparison and extrapolation of performance on current assets to demonstrate feasibility of Cloud rendering.

3.2 Indirect Lighting Strategies

Given the volume of literature on interactive global illumination and the many ways to partition computation between Cloud and local users, we chose to implement three algorithms with very different computation, memory, network, and reconstruction requirements to explore the vast design space (see Table 1). While these techniques may not span the entire space (or include the "optimal" approach), we chose them to provide some empirical insights into advantages and challenges in the relatively uncharted territory of interactive Cloud rendering. We believe that they do represent the currently-dominant strategies for offline and near-realtime global illumination–path, photon, and cone/beam tracing. They are thus the strongest candidates for acceleration and deployment in a real-time cloud assisted renderer, and their quality and robustness have already been established well explored in the literature.

	Voxels	**Irradiance maps**	**Photons**
Costs			
User compute	medium	low	high
Bandwidth	high	low	medium
Server parallelism	gather + scatter	gather	scatter
Features			
Parameterization	not needed	needed	not needed
Compression	delta + wavelet	video	bit-packing

Table 1: *Characteristics of our three indirect lighting algorithms.*

This section provides a high level overview of the three algorithms that we tested, with specific implementation details in Section 4. Figure 1 shows the mapping of the algorithm pipelines onto Cloud, network, and user resources. As data structure, indirect lighting computation, and lighting reconstruction differ greatly between our three algorithms, very different mappings to system resources are most suitable. Note a few key points: for all three algorithms, indirect lighting is computed in the Cloud; conceptually, all three trivially allow amortization of indirect lighting over multiple users; each has significantly different user-side reconstruction costs; and network requirements vary in both bandwidth and latency. For our purposes, we want to answer whether indirect lighting (in any or all of our algorithms) fits well onto the Cloud, whether we can amortize lighting over dozens of users, and whether asynchronous lighting computations present objectionable visual artifacts.

As part of CloudLight, we implemented the following three lighting algorithms: cone-traced sparse voxel global illumination, path-traced irradiance maps, and real-time photon mapping.

Voxels represent indirect irradiance as a directionally varying, low-dimensional quantity on a sparse 3D lattice [Crassin et al. 2011]. Reconstructing indirect light from voxels is relatively inexpensive, though more expensive than from 2D textures. The large memory footprint of the voxel grid preventing transmission of voxels directly to users. Instead, we reconstruct lighting on the Cloud and stream fully-rendered frames to users. This approach's multi-resolution representation allows the use of lower resolutions for fast objects or camera movements and when high quality solutions are not yet available. With a world-space voxel structure, computation can be precisely focused to compute indirect light only where visible, and multiple GPUs inside a server can easily exchange data to collaborate. This is the only method that must render full frames

on a server to reduce bandwidth to the end user. It distributes the rendering pipeline across three GPUs with two splits–one between indirect and direct, and one between direct and display. It is the first step from today's currently-deployed full-frame streaming with dedicated resources per user to a future in which computation is amortized over multiple users on the server side.

Irradiance Maps represent indirect irradiance in texture light maps [Mitchell et al. 2006]. Typically these textures are static, computed offline during authoring. We gather indirect light at texels interactively on the Cloud using ray tracing. Importantly, geometry must be parameterized to allow a mapping of geometry to individual irradiance map texels. Although commonly done, producing a parameterization is laborious and difficult. A client receiving irradiance maps must only decode transmitted H.264 data and combine with locally-computed direct lighting, so relatively weak user hardware suffices. As maps may be broadcast to multiple users, computation trivially amortizes. Incrementally adding multi-bounce lighting is straightforward by gathering from the prior frame's irradiance map. This strategy maps well to current game engines because it minimizes required network bandwidth and changes to the underlying client-side renderer.

Photons represent indirect light as point sampled particles [Mara et al. 2013]. As photons may have independent lifetimes, intelligent management allows reuse between frames and multiple users. This also allows parallelization over multiple cloud GPUs and the ability to progressively update photons in batches for a more immediate response to changing lighting. Client light reconstruction is relatively expensive, requiring recent GPUs for interactivity. However, photons put few demands on scene authoring, requiring neither parameterization nor voxelization. Using photons offers a tradeoff requiring higher user hardware computation in exchange for high image quality and reduced authoring costs. This is our most aggressive approach. It has the potential for the highest quality, especially for glossy indirect reflections, but looks to a future generation of client-side devices and network infrastructure for practical deployment.

We could of course render full full frames on the server using irradiance maps or photons instead of voxels. We have in fact implemented a fallback path for those strategies in which a separate server process launches and renders full frames when a mobile client connects to the server. However, for sufficiently powerful clients, those methods present a lower-bandwidth, lower-latency solution if the client participates in rendering, so we do not report further results on this path. The thrust of our analysis of the voxel strategy is on distributing the indirect and direct light computation between two server-side GPUs that do not share an address space but do share a high-performance bus. Streaming of the final frames to the client is identical to existing solutions and independent of the indirect light strategy, so we give only cursory analysis of that already-explored aspect of the design space.

3.3 System Design

We had to address a number of key design issues with CloudLight: supporting relatively underpowered clients; asynchronously updating direct and indirect lighting; designing for unpredictable bandwidth and latency; refining progressively and smoothly when full updates are infeasible; and ease of incorporation into existing authoring and rendering pipelines. None of these issues are new; Loos et al. [2011] handle weaker clients, Martin and Einarsson [2010] update light incrementally and asynchronously, progressive refinement has been proposed frequently (e.g. [Cohen et al. 1988]), and all network systems must handle unpredictable communications.

We do not argue for one ideal illumination strategy for CloudLight. All three of our prototypes represent different points in the multi-dimensional design space of our criteria. Sparse voxel global illumination supports any client that decodes video and relies on client bandwidths and latencies on par with currently popular network services, but authoring pipelines must change to handle voxel-based lighting, and it does not receive the benefits of decoupling indirect illumination and user framerate. Irradiance maps support relatively low-powered devices, can update indirect light asynchronously, use bandwidth comparable to streaming video, and easily incorporate into existing engines using light maps. However, progressive irradiance map updates are tricky and parameterizing complex scenes is challenging. Photons refine lighting progressively and asynchronously to easily handle dynamic scenes and are straightforward to add to existing rendering systems; however, photons require a capable client device and consume significantly higher bandwidth than our other approaches.

4 Implementation

We implemented the three algorithms described in Section 3.2 using the pipeline structure from Figure 1. Each consists of two separate C++ programs (Cloud and user) connected via the network. To make comparisons as meaningful as possible, all six programs share a significant amount of infrastructure code, including model management, networking, GPU and CPU timers, and OpenGL and OptiX wrapper code.

4.1 Voxels

Our voxel global illumination approach builds on sparse-octree global illumination [Crassin et al. 2011], and can be thought of as a multi-resolution octree irradiance cache or a 3D light map. Using this approach avoids constructing surface parameterizations, a key advantage. On the Cloud, indirect light is gathered to a directionally varying irradiance sample at every multi-resolution voxel. To reconstruct indirect light, we trace cones through this voxel grid (similar to a traditional photon map final gather) to generate view-dependent indirect light for each client. This view-dependent reconstruction also occurs in the Cloud, though it uses a separate GPU from the per-voxel sampling.

Our voxel approach follows these key steps:

1. Voxelize scene geometry (either offline or dynamically)
2. Inject light into and filter the sparse voxel grid
3. Trace cones through grid to propagate lighting
4. Use cone traced results to generate fully-illuminated frames
5. Encode each frame with H.264 and send to appropriate client
6. Decode H.264 on client and display the frame

Basic voxel lighting runs well on high-end PCs, so our efforts focused on mapping it to the Cloud. While view independent, the light injection and propagation steps require substantial resources. To ensure our computations amortize well over many clients, we propagate light via cone tracing to a view independent, per-voxel representation, rather than Crassin et al.'s [2011] per-pixel output.

After cone tracing, querying the resulting view-independent voxel irradiance cache occurs quite efficiently. Unfortunately, shipping a large voxel grid over the network for client reconstruction is infeasible. Instead we transfer the voxels to another Cloud GPU to reconstruct, compress, and send fully rendered frames to clients. Thus, our voxel algorithm uses one GPU (called the *global illumination GPU*) to generate view-independent data plus a smaller GPU (called the *final frame GPU*) to generate the view-dependent frames we send to clients.

To utilize fast GPU-to-GPU transfers, our global illumination and final frame GPUs reside in a single server. However, the significant data size of a voxel representation still requires several other strategies to compress data for efficient transfer:

- bricking voxels, with per-brick compaction;
- wavelet voxel encoding for finer octree levels;
- restricting GPU-to-GPU transfers to a minimal octree cut;
- asynchronous updates with DMA transfers between GPUs;
- progressive, frequency-dependent decompression.

Essentially, we speed transfers by reducing the amount and precision of voxel data, limiting transmissions to important voxels, and using asynchronous communication. We speed reconstruction (and further reduce bandwidth) by computing full resolution only in areas requiring high frequency detail.

On our design spectra from Section 3.3, voxels behave as follows:

- **Client power.** Requires only client H.264 decode.
- **Asynchronous updates.** Computations appear synchronous to client, but occur asynchronously on two GPUs in the Cloud.
- **Bandwidth and latency.** Requires latency similar to VoIP and bandwidth of video-streaming.
- **Progressive refinement.** Multi-resolution octree enables progressive, coarse-to-fine updates.
- **Ease of use.** Pipelines need updating to handle voxels.

4.2 Irradiance maps

Our irradiance map seamlessly fits into existing engines with directional light map illumination, such as Unreal Engine 3 and the Source Engine. Existing systems typically use static, offline "prebaked" irradiance maps. We leave the local device renderer unmodified but extend the system to stream dynamic textures for the illumination data. This keeps the client simple, as the only new logic for dynamic indirect light is a network decoder to interpret incoming irradiance maps.

As long as the server outputs compressed irradiance maps with the required performance, it can use any baking algorithm. We implemented two irradiance map servers. One gathers irradiance naively at each texel using an OptiX-based ray tracer [Parker et al. 2010]. The second, more sophisticated and efficient one first decomposes the irradiance map into coarse basis functions, and only gathers illumination once per basis. This approach requires an order of magnitude fewer rays for comparable performance, accelerating computation sufficiently to allow multiple updates of the entire irradiance map per second.

In both cases, we compress irradiance maps using a hardware H.264 encoder prior to transmission and decompress it client-side with an optimized CUDA decoder. We initially planned more sophisticated compression to mitigate artifacts, as H.264 encoding was not designed for the myriad discontinuities in irradiance maps. However, the masking effects of texturing and ambient occlusion on the low-frequency indirect illumination sufficiently mitigates the visual impact of compression artifacts, and the benefits of leveraging the highly performance and power efficient H.264 encoder outweigh the potential advantages of a custom irradiance map encoder.

Our irradiance map system follows these key steps:

1. (Offline) Generate global unique texture parameterization
2. (Offline) Cluster texels into basis functions
3. Gather indirect light at each basis function (or texel)
4. Reconstruct per-texel irradiance from basis functions
5. Encode irradiance maps to H.264; transmit to client
6. Decode on the client
7. Render direct light; use irradiance map for indirect light

Essentially, at every iteration we perform a texture-space deferred shading pass over the irradiance map (using a texture space G-buffer and current irradiance maps as input). We use OptiX to perform a gather of indirect light, either at every valid texel or once per basis function. We use the rasterizer to offload computation of direct light in texture space, improving performance. We tried numerous other ideas to reduce server cost for irradiance map creation. Importantly, using clustered bases significantly reduces the number of gather points. As a preprocess, we cluster mutually visible texels (e.g., not separated by walls) with similar normals. Each basis has a radius of influence, and when gathering at basis functions, we blend up to 8 bases to reconstruct per-texel irradiance.

Each irradiance map update gathers a single bounce of indirect light. We achieve multi-bounce lighting by consulting the prior irradiance map when gathering subsequent irradiance maps. We sought high memory coherency for rays traced in parallel by: reordering hemispherical QMC samples into clusters of coherent rays; tracing clustered rays in parallel (in a warp) rather than sequentially; and avoiding complex materials during irradiance map creation.

To eliminate popping due to sudden illumination changes or unexpected network latency, client-side temporal filtering can be achieved using an exponentially weighted average over multiple irradiance maps.

Irradiance maps lie as follows on our design spectra from Section 3.3:

- **Client power.** Moderate power client needed, to render direct light plus decoded H.264 irradiance map.
- **Asynchronous updates.** New irradiance maps computed asynchronously, incorporated on client as they arrive.
- **Bandwidth and latency.** Consumes bandwidth equivalent to video streaming. Latency tolerant with client-side filtering.
- **Progressive refinement.** We increase path length by one each iteration by seeding with the current irradiance map. We could (but currently do not) use hierarchical basis functions to increase resolution with each iteration.
- **Ease of use.** Existing rendering pipelines use irradiance maps.

4.3 Photons

We use a standard photon tracer [Jensen 2001] implemented via a Cloud-based OptiX engine. We compact and compress the photons for transmission to the clients, which then render indirect illumination from them via a screen-space scatter approach (e.g., Mara et al. [2013]), rather than a traditional final gather. To produce timely updates, we continually trace photons in small batches and transmit them as soon as they are complete, rather than waiting for all photons in the scene. This allows convergence in time, similar to frameless rendering or real-time path tracing approaches. Because indirect light often changes gradually (in world space), in many cases the artifacts resulting from this are hard to perceive while the short update time between a scene change and new illumination being sent to the client is always beneficial.

Our photon map implementation follows these key steps:

1. Trace photons using Cloud-based ray tracer;
2. Transfer a bit-packed encoding of photons to clients;
3. Expire old photon packets on client; replace with new ones;
4. Scatter photons into client view to accumulate indirect light;
5. Sum indirect light with locally-computed direct illumination.

A key feature of this pipeline is our photon batching. A global parameter controls photon count per emitted watt of illumination, which sets total photons per iteration. We group these into fixed-sized batches, with all photons in each batch emitted from one light. To ensure full GPU utilization and avoid noise for dim lights, we add additional photons (and renormalize) so each light emits an integer number of batches. Each photon stores direction, power, position, radius, and normalization factors packed into a 20-byte structure. We defer normalization to the client to preserve precision; this precision could be ignored to regain some network bandwidth.

Batching has many advantages. Common ray origins and directions dramatically improve memory coherence (and performance) when traversing ray acceleration structures. Tracing and transmitting small batches reduces latency between interaction and first visible change. Fixed batch sizes simplify memory allocations and transfers at multiple stages in our pipeline. When lighting changes, identifying stale photons is straightforward, as batches directly correspond to specific lights; we can reshoot only photons whose corresponding light changed. For dynamic geometry, only photon batches that interact with this geometry need updating.

Once photons reach the client we use an image-space splatting approach to gather indirect light, in particular the 2.5D bounds method of Mara et al [2013]. This uses a deferred render pass, which expands photons to a polygonal approximation of their area of influence. A photon density estimation kernel runs over all covered pixels, with results output to a low resolution additive accumulation buffer. We apply a bilateral upsample to get a full-resolution indirect illumination buffer. This approach was one of the fastest approaches explored by Mara and was easily incorporated into our renderer.

Given our design spectra from Section 3.3, photons fall as follows:

- **Client power.** Photon reconstruction requires powerful client.
- **Asynchronous updates.** Photons computed asynchronously; incrementally incorporated client-side.
- **Bandwidth and latency.** High bandwidth, due to photon size. Progressive nature provides good latency tolerance.
- **Progressive refinement.** Can update subset of photons, including just those for dynamic lights or objects.
- **Ease of use.** Needs no surface paramterization, memory use reasonable, and straightforward reconstruction.

5 Results

Our experiments address whether Cloud-based indirect lighting is feasible in the current hardware ecosystem. Feasibility rests on whether achievable bandwidth and latency allow practical indirect lighting for remote users. We also sought to empirically explore the algorithmic design space to better understand bandwidth and latency tolerances for illumination.

We tested various scenes with complexity found in real games, as well as the common Cornell Box and Sponza test scenes. We purchased the "Old City" model from TurboSquid, "Dockside" comes from *Call of Duty: Black Ops II* (by Treyarch, published 2012 by Activision), "Operation 925" comes from *Battlefield 3: Close Quarters* (by DICE, published 2011 by EA), and "Ironworks" is a re-textured scene from QuakeLive (by id Software in 2010). The conditions under which we have permission to use Dockside and Operation 925 prohibit us from showing them with full materials.

5.1 Network

We tested on wired and wireless networks, on continent-scale connections from Santa Clara, CA to Salt Lake City, UT (1200 km) and Salt Lake City, UT to Williamstown, MA (3600 km), and on town-scale connections across the Williams College campus. Except at extreme distances, latency is not primarily driven by physical distance, but by the number of network hops. That is because the queuing and packet transmission times at routers are high relative to the time taken for light to cover a hundred kilometers.

We consider the town or city-wide scale to be the most plausible deployment scenarios. One could envision existing Internet access and content providers extending their local server nodes with 3D rendering capabilities in the same way that they already have extended those nodes to support media caching, digital goods stores, video streaming services, and massive multiplayer games. Our scalability results are reported for the college campus network because it provided a network under real-world load circumstances on which we could physically manage 50 client machines.

Continent-scale links provide proof of robustness but no interesting quantitative or qualitative results. There, latency was proportional to traditional ping time and bandwidth simply varied with the quality of service purchased. Wireless clients increase variance of latency substantially, but we did not observe them to shift the mean latency significantly compared to the cost of tracing illumination on the server. Our irradiance maps and photons strategies are specifically designed to be robust to both high and variable latency.

5.2 Latency

Figure 2 shows the test scenes as rendered by CloudLight. It is well known that the underlying global illumination algorithms that we built into the system produce pleasing static images. The qualitative research question is how the image quality is affected for dynamic scenes in the presence of latency and compression within the rendering pipeline. To address this, our video shows results for the three system variants on real systems and networks. They also shows sequences rendered with artificial, controlled latency in order to explore the perceptual impact of latency for indirect light. We consider these video results to be the our most compelling analysis of latency in a cloud renderer and refer the reader to them–no perceptual study or graph could help one to judge what is acceptable latency as much as one's own experience.

While perception of artifacts from latency and lossy compression depends on context and user task, we sought to construct both typical and worst cases. In the worst cases, we rapidly change illumination with fast-moving light sources inside the scenes, including a full daylight cycle in one minute, while experiencing high network latency of up to one second. While direct and indirect light are noticeably decoupled, at least for expert viewers, we believe even a second of latency provides relatively compelling indirect lighting. It is interesting to note that computationally hard cases, such full global illumination as large scenes like Operation 925, are also the ones for which we observe that humans have poor intuition for correct. Computationally trivial cases like the Cornell Box are ones in which we have good intuition for the lighting result and notice latency. So, the realistic use case of complex scenes happens to be the one for which the primary limitation of the system–perceived latency–is mitigated.

Table 2 reports quantitative behavior of our three system variants. This table demonstrates the feasibility of Cloud-based indirect illumination algorithms, even those with very different properties. The subsections below delve into the individual algorithm results in more detail. The irradiance map algorithm requires a quality surface parameterization, especially for clustering texels into basis functions. This is a hard problem that is a limitation of that approach, and for scenes marked "n/a," we were unable to create a

Figure 2: *Images of the scenes used for testing, as seen on the client with real-time dynamic indirect light.*

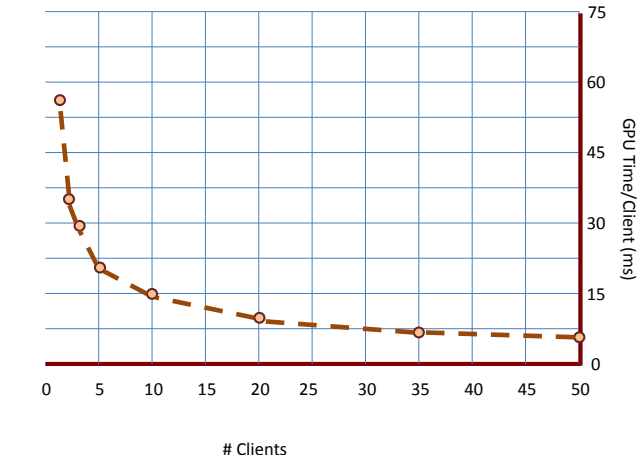

Figure 3: *Per-client GPU costs for up to 50 users with our voxel indirect lighting on Sponza. All clients use a single view-independent voxel grid, but each receives a unique view-dependent rendering. (Our other two strategies have fixed rendering cost, so are independent of the number of clients.)*

suitable parameterization within a reasonable timeframe even with direct assistance and tools from game developers.

All our indirect lighting algorithms run in the Cloud on a GeForce TITAN. The voxel algorithm streams video to the user and relies on a secondary GPU to render view-dependent frames and perform H.264 encoding. Because this secondary GPU leverages direct GPU-to-GPU transfer features of NVIDIA Quadro cards (to quickly transfer voxel data), we use a Quadro K5000 as this secondary GPU. Timing numbers for client-side photon reconstruction occurred on a GeForce 670.

5.3 Voxel Results

Table 2 shows voxel updates are quite efficient, but require sizable memory and significant bandwidth when transferring data between nodes. Using fast GPU-to-GPU transfers enables us to send even

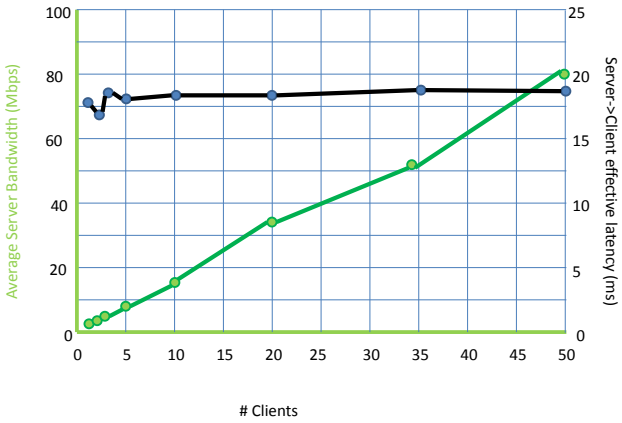

Figure 4: *Bandwidth and latency measured at the server for irradiance mapping with up to 50 client machines in the Ironworks scene. Black: latency vs. number of clients. Green: Server's bandwidth vs. number of clients.*

sizable voxel grids between GPUs within a single server. A second final frame GPU turns view-independent voxel data into view-dependent per-client frames, which undergo hardware H.264 encoding prior to streaming to clients.

Figure 3 shows per-client GPU costs for up to 50 unique users rendering from a single, view-independent voxel grid. This shows the common overhead becomes insignificant once there are many clients. Our tests show even here, with our most expensive workload, each K5000 GPU can simultaneously serve 5 clients with a consistent 30 frames per second (and 25 clients at 12 Hz). We do not show a graph of bandwidth because it simply scales linearly in the number of clients. Bandwidth varies highly based on compression settings and scene content, so any bandwidth measurement and claim would be highly subjective. However, to report at least a bound on the useful range, we observed between 3 and 15 Mbps per client across all settings and (dynamic) scenes.

5.4 Irradiance Map Results

The second block in Table 2 shows irradiance maps have low bandwidth requirements, due to the H.264 encoding of our transferred light maps. Many of our scenes have no timings because they do not have suitable UV-parameterizations. This illustrates the main problem with irradiance maps: paramterizations are difficult to acquire for complex scenes [Boulton 2013], particularly when adding the requirement to construct basis functions representing illumination over dozens or hundreds of nearby texels.

We tested irradiance map scaling for various numbers of unique clients (see Figure 4). This requires sending a copy of the irradiance maps to each user. The per-user latency is essentially constant up to 50 users. As new users require no additional server computation, latency depends exclusively on network congestion. With irradiance maps, we were unable to saturate our network even with 50 clients (all the computers at our disposal).

5.5 Photon Results

Photons do not require parameterizations, so (like voxels) we were able to test on all scenes. Photons require more client-side horsepower than our other approaches, as evident from the grey lines of Table 2. Due to larger bandwidth requirements, photon mapping

Polygon count	**Cornell box** 34	**Sponza** 262,000	**Ironworks** 442,000	**Old city** 1,210,000	**Dockside** 1,773,000	**Operation 925** 2,598,000
Voxels						
Total indirect update (GI GPU)	19 ms	28 ms	28 ms	40 ms	45 ms	32 ms
Light injection	11 ms	9 ms	13 ms	20 ms	26 ms	19 ms
Clear	1 ms	3 ms	2 ms	3 ms	3 ms	2 ms
Gather	7 ms	15 ms	12 ms	16 ms	16 ms	12 ms
GI GPU voxel memory	0.13 GB	2.6 GB	2.2 GB	2.7 GB	2.0 GB	2.8 GB
Intra-server bandwidth (GPU-to-GPU)	2.8 Gbps	6.7 Gbps	4.5 Gbps	6.1 Gbps	3.7 Gbps	1.9 Gbps
Final frame time	5 ms	12 ms	11 ms	14 ms	16 ms	10 ms
H.264 encode (per client)	4.2 ms	4.2 ms	4.2 ms	4.2 ms	4.2 ms	4.2 ms
Local device bandwidth (mean)	3 Mbps	5 Mbps	6 Mbps	6 Mbps	6 Mbps	3 Mbps
Irradiance maps						
Number of basis functions	50	39,000	6,300	n/a	n/a	n/a
Rays per update	26k	20.0M	1.6M	n/a	n/a	n/a
Cloud update time	7.5 ms	214 ms	40 ms	n/a	n/a	n/a
Irradiance map texels	128^2	1024^2	1024^2	n/a	n/a	n/a
H.264 encode	≈0.5 ms	3 ms	3 ms	n/a	n/a	n/a
Local device bandwidth	0.16 Mbps	1.5 Mbps	1.7 Mbps	n/a	n/a	n/a
Local device indirect light size	768 kB	48 MB	48 MB	n/a	n/a	n/a
Photons						
Number of batches	1	13	7	9	8	9
Photons per batch	10,000	40,000	100,000	100,000	400,000	160,000
Memory footprint	67 kB	6.1MB	8.3MB	5.1MB	10.4MB	30.22 MB
Iteration interval (total)	6.1 ms	9.8 ms	31.8 ms	20 ms	32.9 ms	28.3 ms
Photon trace	5.4 ms	7.8 ms	26.8 ms	11.5 ms	28.6 ms	19.5 ms
Photon compaction	0.5 ms	0.9 ms	2.7 ms	2.4 ms	1.6 ms	2.0 ms
GPU to CPU to NIC transfers	0.2 ms	1.1 ms	2.3 ms	6.1 ms	2.7 ms	6.8
Local device bandwidth (mean)	16 Mbps	34 Mbps	38 Mbps	25 Mbps	34 Mbps	43Mbps
Stored photons on local device	3.4k	207k	416k	257k	528k	1511k
Local device indirect computation	8 ms	26 ms	30 ms	28 ms	28 ms	30.2 ms

Table 2: *Measurements for all three algorithms with a GeForce TITAN cloud and GeForce 670 local device. Colors match Figure 1: cyan cloud, orange network, gray local device."B" = bytes, "b" = bits. Not shown: local device GPU H.264 decode takes about 1 ms at 1080p but is impractical to measure directly.*

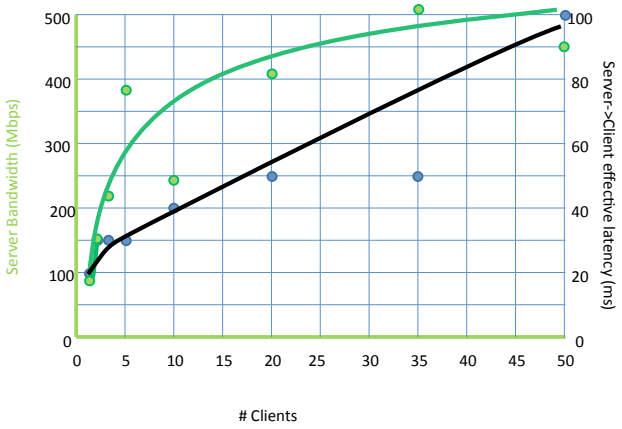

Figure 5: *Scaling of photon mapping bandwidth and latency for up to 50 client machines on Old City.*

does not scale as well as irradiance mapping. Figure 5 shows good scaling to around 30 users, after which bandwidth limits are reached and latency starts increasing from increased network congestion.

Examining the split between Cloud and client photon costs in Table 2, it may appear both could feasibly be performed client-side. However, our Cloud GPU is around three times faster than clients and allows amortizing photon emission across all users.

For our scenes, we tuned batch size to maximize transferred lighting without delaying updates more than a frame over base network latency. With increasing client connections, we reach maximal bandwith quite quickly; while we can easily afford emitting additional photons on the server, this would increase bandwidth (and hence latency) for every batch of photons sent to clients. This suggests we could afford additional computation to select more intelligent photons, e.g., using photon relaxation to compute better photons. Alternatively, finding an optimal photon compression would reduce per-photon bandwidth. This is a somewhat unique situation for interactive global illumination techniques: having excess computation to spend on optimizing sampling.

6 Summary and Discussion

We demonstrated CloudLight, a framework for interactive Cloud-based indirect lighting systems. Our main contribution explored

how to compute indirect lighting with a range of algorithms for distributed architectures. We also demonstrated scalability up to 50 users on a real network, amortizing the indirect lighting across users. We showed a variety of indirect lighting representations and compression techniques, including separating view-independent and view-dependent computations between GPUs and repurposing H.264 for irradiance map compression. We found, empirically, that only coarse synchronization between direct and indirect light is necessary and even latencies from an aggressively distributed Cloud architecture can be acceptable.

All of our systems amortize global illumination. Irradiance maps and photons go a step farther and address the latency issue; they render direct illumination on local clients, which enables immediate response to user input, irrespective of network conditions. Moreover, because the indirect illumination is view-independent it is robust to temporary network outages. In the worst case, the last known illumination is reused until connectivity is restored, which is no worse than the pre-baked illumination found in many game engines today.

The choice of global illumination algorithm has major implications for target bandwidth and client devices. Our voxel approach allows extreme amortization of resources and places almost no computational burden on client devices. However, voxels do not scale to large scenes. We mitigate this limitation via a multiresolution approach, underestimating distant indirect illumination in the process.

Irradiance mapping requires the lowest bandwidth of our algorithms, with latency lower than voxels due to utilization of client computational resources. It also integrates easily into existing game engines. Unfortunately, irradiance maps require a global geometric parameterization. While decades of research have provided a multitude of parameterization techniques, these do not address problems specific to global illumination: handling light leaking where texels lie below walls or keeping world-space samples close in texture space for efficient clustering into basis functions. We see the authoring burden of parameterization as one reason developers are moving towards other techniques, e.g., light probes.

In comparison to irradiance maps, photons require significantly more bandwidth and client computation; however, they eliminate the need for a global parameterization and allow smaller, progressive updates that enable fast reaction to significant scene changes.

6.1 Future Work

In the near future, thermal limits on mobile devices are unlikely to be overcome. To continually improve visual quality at the rate consumers have come to expect, the only solution may be to move some computation to the Cloud.

A wealth of future work suggests itself, and while we answered many initial questions with our prototype CloudLight system, we raised many more. Global illumination is increasingly being used in games, but psychologists have done few experiments involving indirect light. A better perceptual model could make indirect light even more latency tolerant. Our systems rely on a centralized server, yet many distributed networks use peer-to-peer communication. Is there an efficient way to compute indirect lighting via peer-to-peer systems? Another question is whether latency can be managed using client postprocessing, for instance using image warping on video streams with both image and depth.

References

ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I., AND ZAHARIA, M. 2010. A view of cloud computing. *Communications of the ACM 53*, 4.

BOUKERCHE, A., AND PAZZI, R. W. N. 2006. Remote rendering and streaming of progressive panoramas for mobile devices. In *Proceedings of the conference on Multimedia*.

BOULTON, M. 2013. Static lighting tricks in Halo 4. *GDC talk*.

BRODLIE, K. W., DUCE, D. A., GALLOP, J. R., WALTON, J. P., AND WOOD, J. D. 2004. Distributed and collaborative visualization. In *Computer Graphics Forum*, vol. 23.

CHALMERS, A., AND REINHARD, E. 2002. *Practical parallel rendering*. AK Peters Limited.

CHANG, L., FRANK, D. J., MONTOYE, R. K., KOESTER, S. J., JI, B. L., COTEUS, P. W., DENNARD, R. H., AND HAENSCH, W. 2010. Practical strategies for power-efficient computing technologies. *Proceedings of the IEEE 98*, 2, 215–236.

CHEN, K.-T., CHANG, Y.-C., TSENG, P.-H., HUANG, C.-Y., AND LEI, C.-L. 2011. Measuring the latency of cloud gaming systems. In *ACM International Conference on Multimedia*, 1269–1272.

CHOY, S., WONG, B., SIMON, G., AND ROSENBERG, C. 2012. The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *IEEE Workshop on Network and Systems Support for Games*.

COHEN, M. F., CHEN, S. E., WALLACE, J. R., AND GREENBERG, D. P. 1988. A progressive refinement approach to fast radiosity image generation. In *Proceedings of SIGGRAPH*, vol. 22, 75–84.

CRASSIN, C., NEYRET, F., SAINZ, M., GREEN, S., AND EISEMANN, E. 2011. Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum 30*, 7.

2013. Eurographics 2013 - state of the art reports. Eurographics Association, Girona, Spain, C. Dachsbacher, J. Kriváanek, M. Hasan, A. Arbree, B. Walter, and J. Novák, Eds.

DACHSBACHER, C., AND KAUTZ, J. 2009. Real-time global illumination for dynamic scenes. In *ACM SIGGRAPH 2009 Courses*, ACM, New York, NY, USA, SIGGRAPH '09, 19:1–19:217.

DENK, C. 2011. At the verge of change: How HPG drives industrial decision-making. *HPG talk*.

DUTRE, P., BALA, K., AND BEKAERT, P. 2003. *Advanced global illumination*. AK Peters Limited.

ENGEL, K., ERTL, T., HASTREITER, P., TOMANDL, B., AND EBERHARDT, K. 2000. Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *Proceedings of IEEE Visualization*, IEEE Computer Society Press, 449–452.

HU, H. H., GOOCH, A. A., THOMPSON, W. B., SMITS, B. E., RIESER, J. J., AND SHIRLEY, P. 2000. Visual cues for imminent object contact in realistic virtual environment. In *Proceedings of Visualization*.

JENSEN, H. W. 2001. *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA.

KARAPANTAZIS, S., AND PAVLIDOU, F.-N. 2009. VoIP: A comprehensive survey on a promising technology. *Computer Networks 53*, 12.

KLIONSKY, D., 2011. A new architecture for cloud rendering and amortized graphics, August.

KOLLER, D., TURITZIN, M., LEVOY, M., TARINI, M., CROCCIA, G., CIGNONI, P., AND SCOPIGNO, R. 2004. Protected interactive 3d graphics via remote rendering. In *ACM Transactions on Graphics*, vol. 23.

LAWTON, G. 2012. Cloud streaming brings video to mobile devices. *Computer 45*, 2, 14–16.

LOOS, B. J., ANTANI, L., MITCHELL, K., NOWROUZEZAHRAI, D., JAROSZ, W., AND SLOAN, P.-P. 2011. Modular radiance transfer. In *ACM Transactions on Graphics*, vol. 30, 178.

LUKE, E. J., AND HANSEN, C. D. 2002. Semotus visum: a flexible remote visualization framework. In *Proceedings of IEEE Visualization*.

MANZANO, M., HERNANDEZ, J., URUENA, M., AND CALLE, E. 2012. An empirical study of cloud gaming. In *IEEE Workshop on Network and Systems Support for Games*.

MARA, M., LUEBKE, D., AND MCGUIRE, M. 2013. Toward practical real-time photon mapping: Efficient gpu density estimation. In *ACM Symposium on Interactive 3D Graphics and Games*.

MARA, M., 2011. CloudLight: A distributed global illumination system for real-time rendering.

MARTIN, S., AND EINARSSON, P. 2010. A real-time radiosity architecture for video games. *SIGGRAPH 2010 courses*.

MITCHELL, J., MCTAGGART, G., AND GREEN, C. 2006. Shading in Valve's source engine. In *ACM SIGGRAPH Courses*.

PARKER, S. G., BIGLER, J., DIETRICH, A., FRIEDRICH, H., HOBEROCK, J., LUEBKE, D., MCALLISTER, D., MCGUIRE, M., MORLEY, K., ROBISON, A., AND STICH, M. 2010. Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics* (August).

PAUL, B., AHERN, S., BETHEL, E. W., BRUGGER, E., COOK, R., DANIEL, J., LEWIS, K., OWEN, J., AND SOUTHARD, D. 2008. Chromium renderserver: Scalable and open remote rendering infrastructure. *IEEE Transactions on Visualization and Computer Graphics 14*, 3.

RAO, A., LEGOUT, A., LIM, Y.-S., TOWSLEY, D., BARAKAT, C., AND DABBOUS, W. 2011. Network characteristics of video streaming traffic. In *Conference on Emerging Networking Experiments and Technologies*.

RITSCHEL, T., DACHSBACHER, C., GROSCH, T., AND KAUTZ, J. 2012. The state of the art in interactive global illumination. In *Computer Graphics Forum*, vol. 31, 160–188.

SHI, S., NAHRSTEDT, K., AND CAMPBELL, R. 2012. A real-time remote rendering system for interactive mobile graphics. *ACM Transactions on Multimedia Computing, Communications, and Applications 8*, 3.

SINHA, P., AND ADELSON, E. 1993. Recovering reflectance and illumination in a world of painted polyhedra. In *Proceedings., Fourth International Conference on Computer Vision*.

SUZNJEVIC, M., DOBRIJEVIC, O., AND MATIJASEVIC, M. 2009. Mmorpg player actions: Network performance, session patterns and latency requirements analysis. *Multimedia Tools and Applications 45*, 1-3.

TAMM, G., FOGAL, T., AND KRÜGER, J. 2012. Hybrid distributed rendering. In *Poster at IEEE LDAV Symposium 2012*.

THOMPSON, W. B., FLEMING, R. W., CREEM-REGEHR, S. H., AND STEFANUCCI, J. K. 2011. *Visual perception from a computer graphics perspective*. AK Peters Limited.

WHITTED, T. 1980. An improved illumination model for shaded display. *Communications of the ACM 23*, 6.