***CloudLight*: A System for Amortizing Indirect Lighting in Real-Time Rendering**

Cyril Crassin[1] David Luebke[1] Michael Mara[1,2] Morgan McGuire[1,2]
Brent Oster[1] Peter Shirley[1] Peter-Pike Sloan[1] Chris Wyman[1]
[1]NVIDIA [2]Williams College

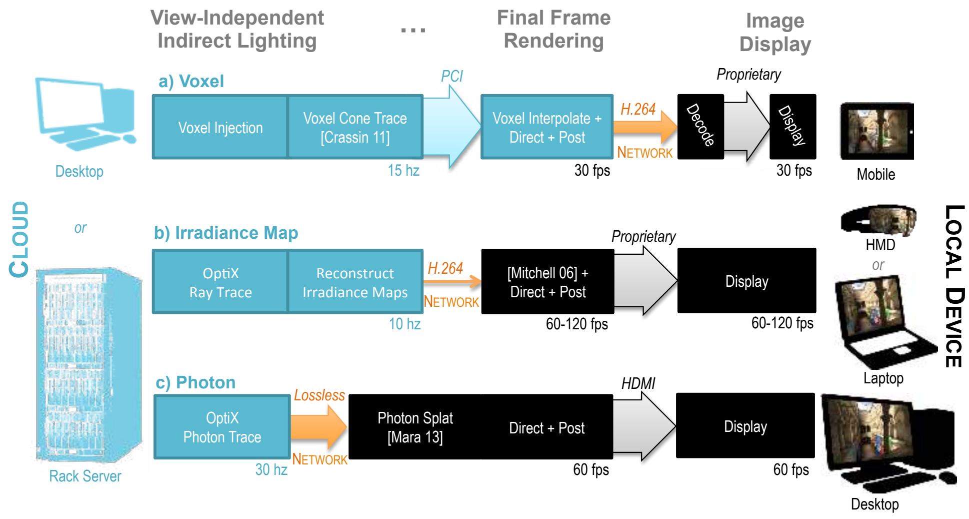

Figure 1. The CloudLight pipelines for three indirect illumination solutions: voxels, irradiance maps, and photons. The split between cloud (left blue) and local computation (right black) differs across targeted local client devices. The client may be in the same building as a desktop server, or elsewhere in the city or region when using a rack server. Arrow thickness denotes required bandwidth; note that we always place the network at the narrowest point.

Abstract

This paper describes the *CloudLight* system for computing indirect lighting asynchronously on an abstracted, computational "cloud," in support of real-time rendering for interactive 3D applications on a local client device.

We develop and evaluate remote-rendering systems for three different indirect illumination strategies: path-traced irradiance ("light") maps, photon mapping, and cone-traced voxels. We report results for robustness and scalability under realistic workloads by using assets from existing commercial games, scaling up to 50 simultaneous clients, and deployed on commercial hardware (NVIDIA GeForce GRID) and software (OptiX) infrastructure.

Remote illumination scales well enough for deployment under each of the three methods; however, they have unique characteristics. Streaming irradiance maps appear practical today

for a laptop client with a distant, rack-mounted server or for a head-mounted virtual reality client with a nearby PC server. Voxels are well-suited to amortizing illumination within a server farm for mobile clients. Photons offer the best quality for more powerful clients.

1. Introduction

Video game image quality on consoles and desktop has increased tremendously in recent years, but dramatic changes in the computing ecosystem pose challenges for continuing that rapid improvement. Today's game players are accustomed to not only rendering-quality increases over time, but also to increasing flexibility through portable devices. That category includes laptops, mobile phones, tablets, and head-mounted displays (HMDs) with mobile processors such as Samsung Gear VR and Microsoft HoloLens. As these become popular game platforms, consumer expectations of both increasing quality and portability create conflict.

Console and PC manufacturers can leverage parallelism, Moore's law, and brute force to increase computational resources on devices that have power cords, heat sinks, and large footprints. This approach is not available on portable platforms. Small, cordless devices limit both peak power consumption (due to thermal dissipation) and total energy (due to batteries). Such devices are thus unlikely [Chang et al. 2010] to reach the rendering performance of today's PCs, let alone improve later to achieve local rendering of advanced effects such as global illumination.

We began this project at NVIDIA's industry research lab because we were skeptical about cloud-based real-time global illumination. We took the three popular rendering strategies and sought to demonstrate the show-stopping limitations of each under a distributed system. Our initial goal was due diligence in wrapping up an internal research project.

We were surprised to find that all three strategies are in fact viable once properly adapted for the cloud. As detailed in this paper, there are bandwidth and client computation scaling limitations for different strategies, but these are not the overall "show stoppers" that we expected. Furthermore, each strategy is well-suited to a particular, practical scenario, so the methods complement each other well.

We were the most surprised to discover that up to half a second of the latency in indirect illumination is visually acceptable and, in fact, hard to notice. This is a critical perceptual issue for any distributed real-time rendering system. It is an immediately obvious conclusion when viewing our videos, but we had no theoretical basis or previous experience that would have predicted it before performing these experiments.

Collectively, these results have converted us from skeptics to *proponents* of distributed real-time rendering. We hope that some of our results will also surprise others and spur further research and development, and that our systems observations will assist in that.

1.1. Simple Cloud Rendering Already Exists

Remote rendering [Koller et al. 2004; Pajak et al. 2011; Chen et al. 2011; Choy et al. 2012; Shi et al. 2012; Manzano et al. 2012] systems attempt to reconcile the trends of higher-quality rendering and lower-power platforms by offloading computation over a network. Such systems also have many advantages beyond improved image quality. These include, for example, virtualization for load-balancing and simplified IT maintenance. Moving computation farther from the player simplifies cooling and form-factor considerations and reduces software piracy. All of these changes reduce support and maintenance costs.

Across all computing applications, today's servers are increasingly abstracted as *the cloud*, servers with reliability similar to a utility [Armbrust et al. 2010] that are similarly commoditized and abstract, from both the consumer's and developer's perspective.

Several commercial distributed rendering systems were recently announced or released [Lawton 2012]. Some address the home-scale scenario in which a local PC-class server supports a mobile client. These include the Nintendo WiiU's remote display, Sony Playstation 4 streaming to a PS Vita, Valve Steam's streaming from a PC to a TV, and NVIDIA streaming from a PC to their SHIELD console or from SHIELD to a TV. Microsoft HoloLens and Oculus Rift leverage a local PC for augmented and virtual reality head-mounted displays. Our voxel and irradiance-map pipelines address these scenarios, including the full-frame streaming approach.

Other systems address the regional scenario. NVIDIA GeForce GRID/SHIELD and Sony PlayStation Now services currently stream from central servers, OTOY recently announced a cloud path-tracing game engine [OTOY 2014], Unity suggested a future engine version will path-trace light maps in the cloud [Mortensen and Cupisz 2014], and Amazon EC2 now offers thousands of cloud GPU instances. Our irradiance map and photon pipelines address these regional scenarios.

Nearly all prior work uses a very simple approach for remote rendering: synchronously map rendering for each users' final image frame to a single server. 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. These previous approaches limit amortization and scalability (both critical requirements for economic cloud deployments) and couple local device-rendering latency to network latency.

1.2. Advancing Cloud-Rendering Architecture

We explore cloud-aware pipelines that simultaneously compute results for multiple users and pipelines that separate view-dependent final frames from view-independent illumination terms. These present new opportunities to trade off latency, bandwidth, and image quality, as well as leverage local device computation to reduce perceived latency.

To advance remote rendering beyond today's full-frame video-streaming approaches, we introduce the *CloudLight* system comprising three pipelines. Each maps an indirect lighting algorithm onto a distributed infrastructure (shown in Figure 1). These pipelines demonstrate the feasibility of distributing a real-time graphics pipeline across a network and explore the design-space tradeoffs. A key takeaway is that different illumination algorithms present varying bandwidth and latency requirements within the rendering pipeline. Those requirements change the ideal location for the network within the pipeline. The pipeline location of the network in turn determines the computing resources required on server and client, and thus determines viable deployment scenarios for the algorithm.

1.3. Contributions

This paper reports answers to the high-level questions that we ourselves first asked around latency, scalability, and practicality for cloud graphics under different illumination algorithms: voxels, irradiance maps, and photons. We present the design of distributed rendering systems optimized for each. To satisfy the constraints of a practical and reproducible system, we build on commercially-deployed systems including the OptiX ray-tracer, the hardware video encoder and decoder in NVIDIA GPUs, and the GRID server product. We examine trade-offs among the systems in terms of implementation, deployment, and image quality and measure performance of the three designs under loads from 1 to 50 clients per server node.

2. Background and Related Work

We briefly review major previous work in the many areas of graphics and networking relevant to CloudLight.

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 share computations from individual units. That interconnect may be the Internet, but it may also be a local network, wireless network, or bus.

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 among 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.

Remote rendering has been broadly adopted (e.g., [Brodlie et al. 2004; Koller et al. 2004]) for non-gaming applications. These systems typically stream video or progressive images, but a few send graphics commands [Paul et al. 2008]. Sending graphics commands effectively places network latency between CPU and GPU; our system instead places network links between algorithmic components at locations chosen to minimize the perception of latency. 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 ; Tamm et al. 2012], and this work shares high-level goals with our own but is very different in detail due to the rendering goals of the very different datasets. 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 shows that indirect lighting provides important but weak spatial and albedo estimation cues [Hu et al. 2000; Sinha and Adelson 1993]. Although little objective work explores whether indirect lighting improves the subjective viewer experience [Thompson et al. 2011], 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 how existing global illumination algorithms map to the cloud rather than formally investigating perceptual questions as done by Yu et al. [2009]. Our video shows illumination under varying latencies, 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 and others [2009; 2014] 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 that a variety of algorithms can split work between clients and servers.

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 usability 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 distributed systems must handle unpredictable communications.

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 [2012] created an early prototypethat later led to the full CloudLight system described in this paper. We presented an intermediate form between Mara's thesis and this work in a SIGGRAPH 2013 short talk.

3. System

3.1. Hardware Architecture and Scene Characteristics

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 or NVLink). 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 receive and decode a 720 p H.264 stream in real time with less than 200 ms of latency. While such networks are not available everywhere, these modest requirements are comparable to home video streaming and less than current cloud gaming systems. The photon 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.

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

Table 1. Characteristics of our three lighting algorithms.

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 in order 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-popular strategies for offline and near–real-time 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 in the literature.

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 amoong 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 clients in a multiplayer environment; each has significantly different user-side reconstruction costs; and network requirements vary in both bandwidth and latency.

Voxels (Figure 1(a)) represent indirect irradiance as a directionally varying, low-dimensional quantity on a sparse 3D lattice [Crassin et al. 2011]. The large memory footprint of the voxel grid prevents transmission of voxels directly to users. Instead, we reconstruct lighting on the cloud and stream fully-rendered frames to users. 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.

Irradiance maps (Figure 1(b)) represent indirect irradiance in texture light maps [Mitchell et al. 2006]. We gather indirect light at texels interactively on the cloud using ray tracing. A client receiving irradiance maps must only decode transmitted H.264 data and combine with locally-computed direct lighting, so relatively low-power user hardware suffices.

Photons (Figure 1(c)) represent indirect light as point-sampled particles [Mara et al. 2013]. Client light reconstruction is relatively expensive, requiring recent GPUs for interactivity. Photons have high client requirements, but offer higher image quality and reduced authoring costs compared to irradiance maps.

We'll refer several times to the observation that good light-map paramterizations are hard, and this is a major motivation for moving beyond the irradiance maps. Paramterization is hard for several reasons. Texels may wrap around corners or pass under walls, thus observe radically different lighting conditions across their area that appear as light or darkness leaks in the final result. Arbitrary amounts of padding are needed around atlased surfaces within a texture to account for MIP map and bilinear filtering. The cuts created in the map to allow for flattening of complex geometry and isolated objects create high-frequency artifacts in the final illumination. Small and thin objects (e.g., foliage, poles) require at least one texel and both distort equal-area mappings and create islands in the map that are wasteful and hard to filter. Objects with dynamic topology are, of course, impossible for any static paramterization scheme to handle well. These issues are well known in industry and frequently mentioned in production talks (e.g., [Boulton 2013], [Iwanicki 2013]), but not well acknowledged in the academic literature, which presents paramterization as a solved problem.

4. Implementation

We implemented the three algorithms described in Section 3.2 using the pipeline structure from Figure 1. Each consists of two C++ programs (cloud/server and user/-client) connected by the network. To make direct comparisons, all six programs share underlying model management, networking, GPU and CPU timers, OpenGL, and OptiX code.

To maximize throughput, all systems are pipelined from the GPU to CPU to NIC, and are multithreaded within the GPU and CPU for further pipelining. This is implemented by placing a queue between each step of each system (queues are implicitly provided within OpenGL and OptiX steps). Our reliable, threaded network messaging system is built on the enet UDP library, and we've released it as open source in the G3D Innovation Engine as the `NetConnection` class.

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 parametrizations, a key advantage. However, voxels are subject to light leaks where a single voxel is unable to represent the different illumination conditions on opposite sides of a very thin wall that bisects it.

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 for performance it uses a separate GPU from the per-voxel sampling.

Our voxel approach performs these steps (color-coded according to the stages in Figure 1):

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 final frames

5. Encode each frame with H.264 and stream out

7. 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:

- combining voxels into larger ‘bricks’;
- wavelet voxel encoding for finer octree levels;
- restricting GPU-to-GPU transfers to a minimal octree cut;
- asynchronous DMA transfers between GPUs;
- progressive, frequency-dependent decompression.

The bricks are explained in the previous cone-tracing work [Crassin et al. 2011]. Wavelet encoding consists of storing the difference from the parent level at each octree node for the output, rather than absolute values. This is equivalent to a 3D Haar transform. Because the deltas are lower-magnitude than the values, we can reduce precision. For GPU-to-GPU transfers, we need only transfer data that changed each frame. This automatically avoids sending voxels for empty space as well, since their values never change. Under progressive decompression, we update the final frame GPU’s data structures in breadth-first order from the octree root. Like progressive JPEG display on the web, this allows us to meet real-time frame constraints using low-frequency illumination before the full details have been transferred or decompressed.

Collectively, these enhance data transfer by reducing the amount and precision of voxel data, limiting transmission to visually-important voxels, and leveraging asynchronous communication. We speed reconstruction (and further reduce bandwidth) by computing at full resolution only in areas that contain high-frequency geometric detail.

Under the design considerations for a cloud renderer, Voxels rate:

- *Client power.* Requires only H.264 decoding; no 3D.
- *Asynchronous updates.* Computations appear synchronous at the client GPU, but actually occur asynchronously on two other GPUs in the cloud.
- *Bandwidth and latency.* Requires the bandwidth of video-streaming and latency similar to VoIP, which are viable on all non-HMD platforms.
- *Progressive refinement.* Multi-resolution octrees enable progressive, coarse-to-fine updates.
- *Ease of use.* Pipelines need to be rewritten for voxels.

4.2. Irradiance Maps

Our irradiance map strategy slots into game engines that already use directional light map illumination, such as Unreal Engine and the Source Engine. Existing systems typically use static, offline "pre-baked" irradiance maps. We leave the local device-renderer unmodified (it is a simple forward renderer) 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 compress irradiance maps using a hardware H.264 encoder prior to transmission and decompress it client-side with an optimized CUDA decoder.

Our irradiance-map system follows these key steps:

1. (Offline) Generate global unique texture parametrization
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

7. Decode on the client
8. 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 implemented two irradiance map servers strategies. The first gathers irradiance naively at each texel using an OptiX-based ray-tracer [Parker et al. 2010].

The second strategy is a more sophisticated and efficient one. It first decomposes the irradiance map into coarse basis functions as an offline preprocess for the scene in a manner similar to precomputed radiance transfer. At run time, the system traces clusters of rays per basis instead of per texel. This approach requires an order-ofmagnitude fewer rays for comparable performance, accelerating computation sufficiently to allow multiple updates per second of the entire irradiance map.

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 eight 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, we apply client-side temporal filtering in irradiance-map space. This is simple: blending each map over the existing one as it is received gives an exponentially weighted average with no additional storage.

On our design spectra, we rate irradiance maps:

- *Client power.* Moderate-power client needed to render direct light plus decoded H.264 irradiance map. Viable for latest-generation phones/tablets, HMDs, and PCs.
- *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.* Game engines already use irradiance maps.

4.3. Photons

We use a standard photon tracer [Jensen 2001] implemented via a cloud-based OptiX engine. We 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. This fits naturally into a deferred-rendering pipeline. 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 resulting artifacts 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 a cloud-based ray-tracer
2. Transfer a bit-packed encoding of photons to clients
3. Replace old photon packets on the client with new ones
4. Scatter photons on the client for indirect lighting
5. Sum indirect light with locally-computed direct light

A key feature of this pipeline is 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. We ensure that each light emits an integer number of batches for full GPU utilization. Each photon stores direction, power, position, radius, and normalization factors packed into 20-bytes.

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, since 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 et al., but more importantly, it was the easiest to integrate into our full renderer. Mara et al.'s other methods require building complex GPU data structures and switching between compute and graphics mode, whereas splatting simply applies brute force with the rasterizer.

On our design spectra, photons rate as follows:

- *Client power.* Photon splatting requires a discrete GPU on the client. Not appropriate for HMD, laptop, or mobile.
- *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 parametrization, memory use reasonable, and straightforward reconstruction.

5. Results

Our experiments address the feasibility of cloud-based indirect lighting on the current hardware ecosystem. This relies on whether achievable bandwidth and latency allow practical indirect lighting. We also sought to empirically explore the algorithmic design space to better understand bandwidth and latency tolerances for illumination.

We tested various scenes with the complexity found in real games, as well as the common Cornell Box and Sponza test scenes. The "Old City" model is derived from one that we purchased on TurboSquid.com, "Dockside" is from *Call of Duty: Black Ops II* (by Treyarch, published 2012 by Activision), "Operation 925" is 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).

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), on town-scale connections across the Williams College campus, and home-scale networks with a single wireless router. 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 100 kilometers.

For a local network, the bandwidth and latency are trivially sufficient to stream photons between a powerful desktop computation cloud and a wireless mobile or head-mounted local client device. However, such local client devices aren't powerful enough for photon splatting and the latency of the voxel method is too high for a HMD, so we envision irradiance maps as the ideal method in this scenario.

For a wide-area network, 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. It 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 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. The worst cases include fast-moving light sources inside the scenes (including a full daylight cycle in one minute) under injected network latency of up to one second. While direct and indirect light are noticeably decoupled in these scenarios, at least for expert viewers, we believe even one second of latency provides relatively compelling indirect

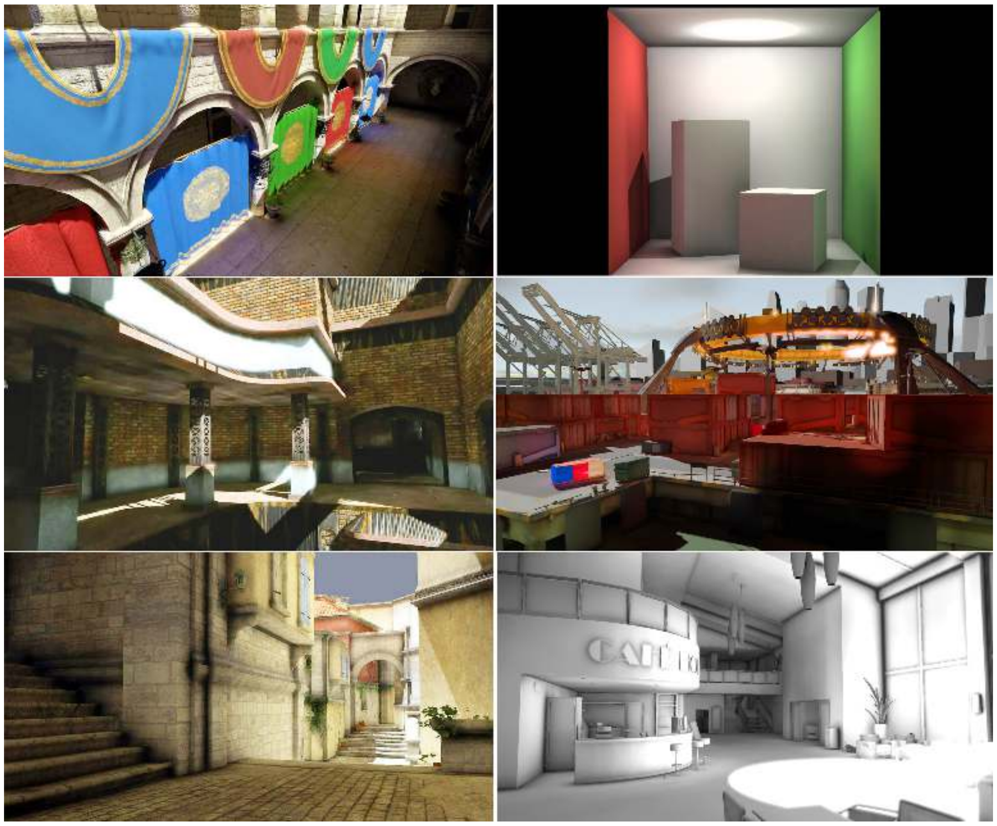

Figure 2. Images of the scenes used for testing, as seen on the client with real-time GI. Three shipped with commercial games, two are academic benchmarks, and we modeled the lower-left Old City to represent future games designed for dynamic indirect lighting.

lighting. It is interesting to note that computationally hard cases, like full GI in large scenes such as Operation 925, are also the ones for which humans have poor intuition for correct lighting. Computationally trivial cases such as the Cornell Box are the only ones in which humans have good intuition for the lighting result and notice the 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 with very different properties. The remaining sections delve into the individual algorithm results in more detail. The irradiance-map algorithm requires a quality surface parametrization, 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 suitable parametrization within a reasonable time frame, despite direct assistance and tools from the game developers who authored them.

	Cornell Box	**Sponza**	**Ironworks**	**Old City**	**Dockside**	**Operation925**
Polygon count	34	262,000	442,000	1,210,000	1,773,000	2,598,000
						Voxels
Indirect (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
GPU-to-GPU bandwidth	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
Server-to-client 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
Server-to-client 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	0.2 ms	1.1 ms	2.3 ms	6.1 ms	2.7 ms	6.8
Server-to-client bandwidth (mean)	16 Mbps	34 Mbps	38 Mbps	25 Mbps	34 Mbps	43Mbps
Local device stored photons	3.4k	207k	416k	257k	528k	1511k
Local device indirect time	8 ms	26 ms	30 ms	28 ms	28 ms	30.2 ms

Table 2. Measurements for all three algorithms with a GeForce Titan (4.5 TFlops / 250 W) cloud and GeForce 670 (2.46 TFlops / 170 W) local device. The voxel variant also has a Quadro K5000 (2.1 TFlops / 122 W) in the cloud to perform the final frame rendering and H.264 encode. Colors used 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.

All of our indirect-lighting algorithms run in the cloud on a GeForce Titan (4.5 TFlops / 250 W). 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 (2.1 TFlops / 122 W) as this secondary GPU. Timing numbers for client-side photon reconstruction occurred on a GeForce 670 (2.46 TFlops / 170 W).

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 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 that 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

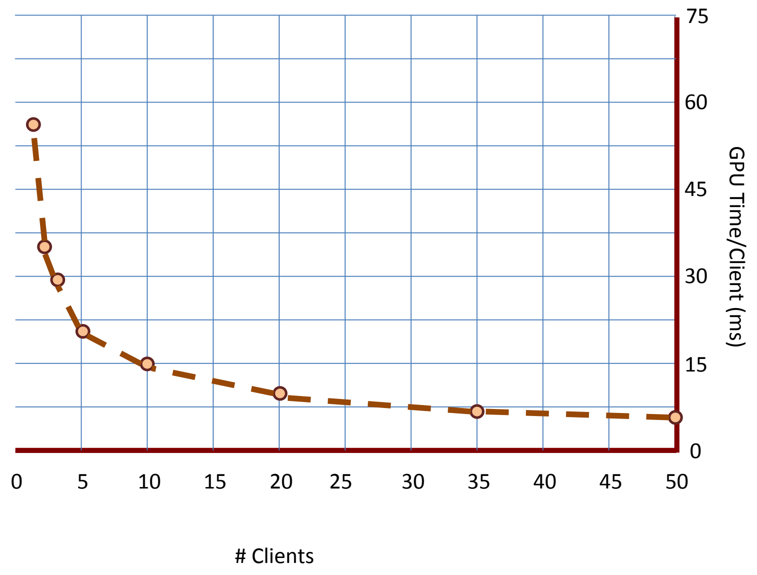

Figure 3. Per-client GPU costs for up to 50 users with voxel lighting in Sponza. All clients use a single view-independent voxel grid, but each receives a unique view-dependent rendering. (Our other two strategies have fixed rendering cost, so are independent of client count.)

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 that 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-parametrizations. This illustrates the main problem with irradiance maps: paramterizations are difficult to acquire for complex scenes, 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 with 50 clients (all of the computers at our disposal).

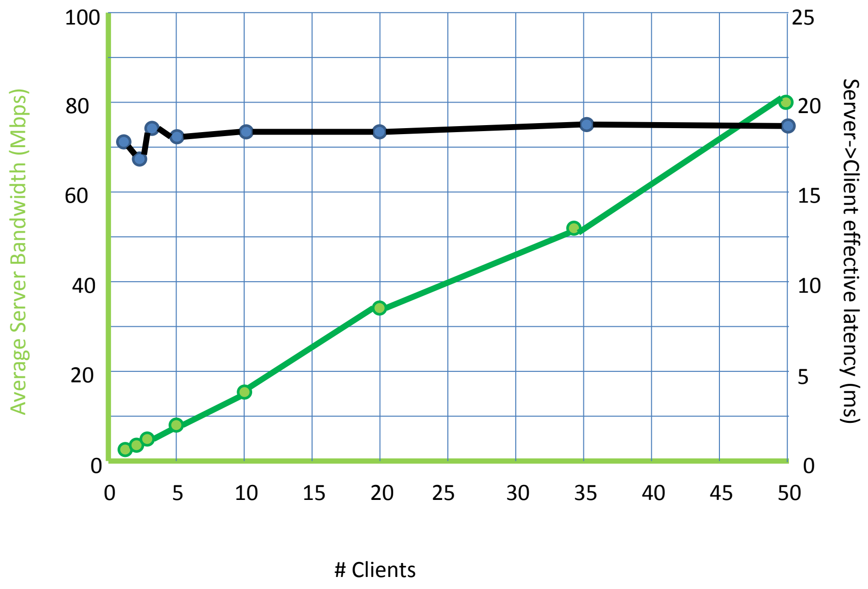

Figure 4. Bandwidth and latency measured at the server for irradiance mapping with up to 50 client machines in the Ironworks scene. Black: latency vs. number of clients. Green: Server's bandwidth vs. number of clients.

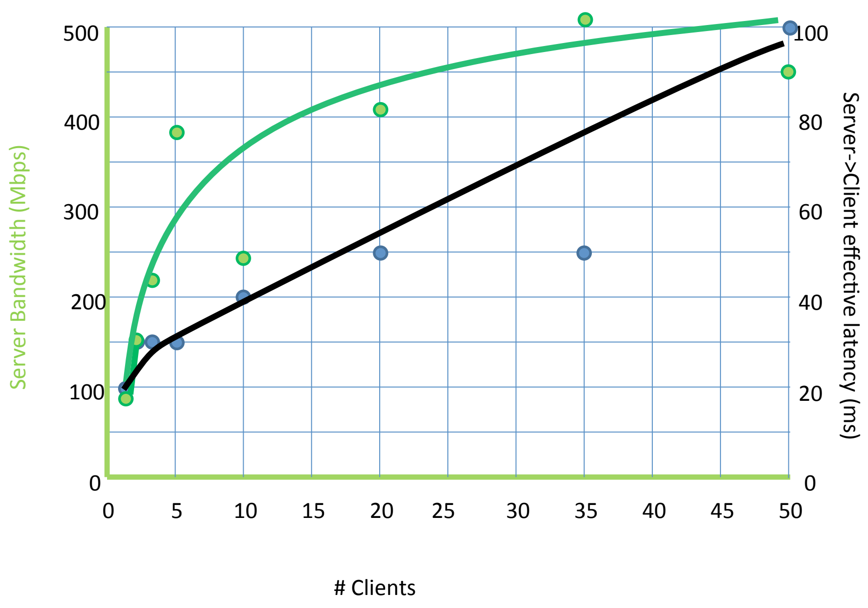

Figure 5. Scaling of photon mapping bandwidth and latency for up to 50 client machines on Old City.

5.5. Photon Results

Photons do not require parametrizations, 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 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 abut three times faster than our 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 described and evaluated CloudLight, a framework for interactive cloud-based indirect lighting systems. Our main contribution is an exploration of how to compute indirect lighting with a range of algorithms for distributed architectures. We demonstrated scalability up to 50 users on a real network, amortizing the indirect lighting across users. We also 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 latencies from an aggressively distributed cloud architecture can be acceptable.

All of our variants amortize global illumination. Irradiance maps and photons go a step farther and address the latency issue; they render direct light locally to enable immediate response to user input irrespective of network conditions. That is, they decouple network latency from final frame rendering latency. Moreover, because the indirect light 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. Fallback to last-known or prebaked values is unavailable for voxels because the gather phase occurs in the cloud.

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, the voxel approach does not scale to large scenes. We mitigate this limitation via a multi-resolution approach by computing indirect illumination at intermediate octree nodes instead of leaves for distant voxels.

Irradiance mapping requires the lowest bandwidth of our algorithms, with image latency lower than voxels due to local rendering of the final frame on the client. It also integrates easily into existing game engines, many of which (such as Unity and Unreal Engine 4) will generate the light-map texture coordinate parametrization directly. We see this as the most likely distributed rendering structure for near-term deployment on mobile devices up to laptops, and the only currently-viable strategy for head-mounted displays. That is because HMDs have sufficient resources to provide the local direct illumination rendering and require very low latency on the final image to match head tracking.

In comparison to irradiance maps, photons require significantly more bandwidth and client computation. However, the photons method eliminates the need for a global parametrization and allows smaller, progressive updates that enable fast reaction to significant scene changes. The image quality is best with photons, and we see this as a viable target for the next generation of consoles and current PCs.

6.1. Future Work

In the near future, thermal limits on portable devices are unlikely to be overcome. To continually improve visual quality at the rate consumers have come to expect for tablets and HMDs, the only solution may be to move some computation to the cloud at either the regional or home scale.

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 computations even more tolerant of network latency. Our systems rely on a server and traditional network routing, yet many distributed systems for non-graphics applications leverage peer-to-peer communication and computation. 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, 50–58. URL: http://doi.acm.org/10.1145/1721654.1721672. 3, 4

BOULTON, M. 2013. Static lighting tricks in Halo 4. *GDC talk*. URL: http://www.scribd.com/doc/232233438/Static-Lighting-Tricks-in-Halo#scribd. 8

BRODLIE, K. W., DUCE, D. A., GALLOP, J. R., WALTON, J. P., AND WOOD, J. D. 2004. Distributed and collaborative visualization. *Computer Graphics Forum 23*, 2, 223–251. URL: http://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2004.00754.x/abstract. 5

CHALMERS, A., AND REINHARD, E. 2002. *Practical Parallel Rendering*. A K Peters, Ltd., Natick, MA. 5

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. URL: http://dx.doi.org/10.1109/JPROC.2009.2035451. 2

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 *Proceedings of the 19th ACM International Conference on Multimedia*, ACM, New York, 1269–1272. URL: http://doi.acm.org/10.1145/2072298.2071991. 3

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 *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, IEEE Press, Piscataway, NJ, 2:1–2:6. URL: http://dl.acm.org/citation.cfm?id=2501560.2501563. 3

COHEN, M. F., CHEN, S. E., WALLACE, J. R., AND GREENBERG, D. P. 1988. A progressive refinement approach to fast radiosity image generation. *SIGGRAPH Comput. Graph. 22*, 4, 75–84. URL: http://doi.acm.org/10.1145/378456.378487. 6

CRASSIN, C., NEYRET, F., SAINZ, M., GREEN, S., AND EISEMANN, E. 2011. Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum 30*, 7. URL: http://dx.doi.org/10.1111/j.1467-8659.2011.02063.x. 7, 9, 10

DACHSBACHER, C., AND KAUTZ, J. 2009. Real-time global illumination for dynamic scenes. In *ACM SIGGRAPH 2009 Courses*, ACM, New York, 19:1–19:217. URL: http://doi.acm.org/10.1145/1667239.1667258. 6

DACHSBACHER, C., KIVNEK, J., HAAN, M., ARBREE, A., WALTER, B., AND NOVK, J. 2014. Scalable realistic rendering with many-light methods. *Computer Graphics Forum 33*, 1, 88–104. URL: http://dx.doi.org/10.1111/cgf.12256. 6

DENK, C. 2011. At the verge of change: How HPG drives industrial decision-making. *HPG Keynote talk*. URL: http://highperformancegraphics.org/previous/www_2011/media/Keynotes/HPG2011_Keynote_Denk.pdf. 5

DUTRE, P., BALA, K., AND BEKAERT, P. 2003. *Advanced Global Illumination*. A K Peters, Ltd., Natick, MA. 6

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, Los Alamitos, CA, 449–452. URL: http://dx.doi.org/10.1109/VISUAL.2000.885729. 5

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 the Conference on Visualization '00*, IEEE Computer Society Press, Los Alamitos, CA, USA, 179–185. URL: http://dl.acm.org/citation.cfm?id=375213.375238. 5

IWANICKI, M. 2013. Lighting technology of "The Last of Us". In *ACM SIGGRAPH 2013 Talks*, ACM, New York, 20:1–20:1. URL: http://doi.acm.org/10.1145/2504459.2504484. 8

JENSEN, H. W. 2001. *Realistic Image Synthesis Using Photon Mapping*. A K Peters, Ltd., Natick, MA. 12

KARAPANTAZIS, S., AND PAVLIDOU, F.-N. 2009. VoIP: A comprehensive survey on a promising technology. *Computer Networks 53*, 12. URL: http://dx.doi.org/10.1016/j.comnet.2009.03.010. 5

KLIONSKY, D. 2011. *A New Architecture for Cloud Rendering and Amortized Graphics*. PhD thesis, Carnegie Mellon University. CMU-CS-11-122. 6

KOLLER, D., TURITZIN, M., LEVOY, M., TARINI, M., CROCCIA, G., CIGNONI, P., AND SCOPIGNO, R. 2004. Protected interactive 3D graphics via remote rendering. *ACM Trans. Graph. 23*, 3, 695–703. URL: http://doi.acm.org/10.1145/1015706.1015782. 3, 5

LAWTON, G. 2012. Cloud streaming brings video to mobile devices. *Computer 45*, 2, 14–16. URL: http://dx.doi.org/10.1109/MC.2012.47. 3

LOOS, B. J., ANTANI, L., MITCHELL, K., NOWROUZEZAHRAI, D., JAROSZ, W., AND SLOAN, P.-P. 2011. Modular radiance transfer. *ACM Trans. on Graph. 30*, 6, 178. URL: http://doi.acm.org/10.1145/2070781.2024212. 6

LUKE, E. J., AND HANSEN, C. D. Semotus visum: a flexible remote visualization framework. In *Proceedings of the Conference on Visualization*, IEEE Computer Society, Washington, DC, 61–68. URL: http://dl.acm.org/citation.cfm?id=602099.602107. 5

MANZANO, M., HERNANDEZ, J., URUENA, M., AND CALLE, E. 2012. An empirical study of cloud gaming. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*, IEEE Press, Piscataway, NJ, 17:1–17:2. URL: http://dl.acm.org/citation.cfm?id=2501560.2501582. 3

MARA, M., LUEBKE, D., AND MCGUIRE, M. 2013. Toward practical real-time photon mapping: Efficient gpu density estimation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, 71–78. URL: http://doi.acm.org/10.1145/2448196.2448207. 8, 13

MARA, M., 2012. CloudLight: A distributed global illumination system for real-time rendering. Williams College Undergraduate Thesis, Williamstown, MA. 6

MARTIN, S., AND EINARSSON, P. 2010. A real-time radiosity architecture for video games. In *Advances in Real-Time Rendering in 3D Graphics and Games, SIGGRAPH 2010 course*. URL: http://advances.realtimerendering.com/s2010/Martin-Einarsson-RadiosityArchitecture(SIGGRAPH%202010%20Advanced%20RealTime%20Rendering%20Course).pdf. 6

MITCHELL, J., MCTAGGART, G., AND GREEN, C. 2006. Shading in Valve's source engine. In *ACM SIGGRAPH 2006 Courses*, ACM, New York, 129–142. URL: http://www.valvesoftware.com/publications/2006/SIGGRAPH06_Course_ShadingInValvesSourceEngine.pdf. 8

MORTENSEN, J., AND CUPISZ, K., 2014. Advanced lighting techniques in Unity, March. GDC Talk. URL: https://docs.google.com/file/d/0B11iL4IgOgWLdFhzalpmX2kzSmM. 3

OTOY, 2014. OTOYs Brigade engine to launch on Amazon EC2, bringing photorealistic next-generation cloud gaming to all, March. URL: https://home.otoy.com/. 3

PAJAK, D., HERZOG, R., EISEMANN, E., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2011. Scalable remote rendering with depth and motion-flow augmented streaming. *Computer Graphics Forum 30*, 2, 415–424. URL: `http://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2011.01871.x/abstract`. 3

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 Trans. on Graph. 29*, 4, 66:1–66:13. URL: `http://doi.acm.org/10.1145/1778765.1778803`. 11

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, 627–639. URL: `http://doi.ieeecomputersociety.org/10.1109/TVCG.2007.70631`. 5

RAO, A., LEGOUT, A., LIM, Y.-S., TOWSLEY, D., BARAKAT, C., AND DABBOUS, W. 2011. Network characteristics of video streaming traffic. In *Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies*, ACM, New York, 25:1–25:12. URL: `http://doi.acm.org/10.1145/2079296.2079321`. 5

RITSCHEL, T., DACHSBACHER, C., GROSCH, T., AND KAUTZ, J. 2012. The state of the art in interactive global illumination. *Computer Graphics Forum 31*, 1, 160–188. URL: `http://dx.doi.org/10.1111/j.1467-8659.2012.02093.x`. 6

SHI, S., NAHRSTEDT, K., AND CAMPBELL, R. 2012. A real-time remote rendering system for interactive mobile graphics. *ACM Trans. Multimedia Comput. Commun. Appl. 8*, 3s, 46:1–46:20. URL: `http://doi.acm.org/10.1145/2348816.2348825`. 3

SINHA, P., AND ADELSON, E. 1993. Recovering reflectance and illumination in a world of painted polyhedra. In *Proceedings of the 4th International Conference on Computer Vision*, IEEE, Washington, DC, 156–163. URL: `http://dx.doi.org/10.1109/ICCV.1993.378224`. 5

TAMM, G., FOGAL, T., AND KRÜGER, J. 2012. Hybrid distributed rendering. In *Poster at IEEE LDAV Symposium 2012*. 5

THOMPSON, W. B., FLEMING, R. W., CREEM-REGEHR, S. H., AND STEFANUCCI, J. K. 2011. *Visual Perception from a Computer Graphics Perspective*. A K Peters Ltd., Natick, MA. 5

WHITTED, T. 1980. An improved illumination model for shaded display. *Communications of the ACM 23*, 6. URL: `http://dx.doi.org/10.1145/358876.358882`. 6

YU, I., COX, A., KIM, M. H., RITSCHEL, T., GROSCH, T., DACHSBACHER, C., AND KAUTZ, J. 2009. Perceptual influence of approximate visibility in indirect illumination. *ACM Trans. Appl. Percept. 6*, 4 (Oct.), 24:1–24:14. URL: `http://doi.acm.org/10.1145/1609967.1609971`. 5

Author Contact Information

Cyril Crassin
ccrassin@nvidia.com

David Luebke
dluebke@nvidia.com

Michael Mara
mmara@nvidia.com

Morgan McGuire
momcguire@nvidia.com

Brent Oster
boster@nvidia.com

Peter Shirley
Now at Purity, LLC
shirley@purity.mobi

Peter-Pike Sloan
Now at Activision
ppjsloan@gmail.com

Chris Wyman
cwyman@nvidia.com

Cyril Crassin, David Luebke, Michael Mara, Morgan McGuire, Brent Oster, Peter Shirley, Peter-Pike Sloan, and Chris Wyman, *CloudLight*: A system for amortizing indirect lighting in real-time rendering, *Journal of Computer Graphics Techniques (JCGT)*, vol. 4, no. 4, 1–27, 2015
http://jcgt.org/published/0004/04/01/

Received:	2015-05-04		
Recommended:	2015-06-21	Corresponding Editor:	Angelo Pesce
Published:	2015-10-15	Guest Editor-in-Chief:	Eric Haines