Composite Lighting Simulations with Lighting Networks

Recent years have seen the introduction of many different lighting simulation techniques for computing the distribution of light in a virtual environment. For environments without participating media (volumetric effects), simulation requires solving the radiance or rendering equation numerically. Many of the available techniques restrict the lighting effects they simulate by using a simplified version of the radiance equation to speed up the calculation. Examples include radiosity algorithms that only consider diffuse reflections but can achieve fast simulations using advanced hierarchical and clustering methods.

On the other hand, Monte-Carlo path or photon tracing techniques simulate illumination by recursive stochastic sampling of illumination in the environment, starting with rays from the virtual camera or the light sources, respectively. Although this technique solves the general form of the radiance equation and can therefore account for any lighting effect, it converges rather slowly and is best used for computing illumination via highly specular surfaces.

The basic finite element and Monte-Carlo methods have also been combined in hybrid, two-pass, and multi-pass methods. For example, the irradiance caching technique is based on path tracing but also includes smooth basis functions (similar to radiosity) to interpolate indirect illumination from previously cached computations. Two- and multi-pass techniques try to exploit the advantages of multiple lighting algorithms by combining the effects computed by each. For example, several methods employ a radiosity computation followed by a second view-dependent ray-tracing pass for capturing specular highlights. One method also includes a geometric simplification step for speeding up the radiosity pass.

In addition, several techniques include Monte-Carlo path and photon tracing for simulating specular and caustic light paths. Usually, these methods split the description of reflection and/or illumination into separate parts (high and low spatial frequency components), each simulated with a different lighting algorithm before being combined. Note, this split is the same for the complete scene, and only a specific combination of algorithms is allowed.

In that sense, the Lighting Networks approach builds on top of these methods. It generalizes the concept of hybrid multi-pass techniques and extends it in several ways. Thus, a Lighting Network does not constitute a new simulation algorithm in itself, it provides a way to best use existing algorithms for a specific environment. During development it turned out that Lighting Networks also provide an ideal environment for developing new lighting simulation algorithms and explicitly showing the similarities and differences between lighting simulation approaches.

In the Lighting Networks approach, each lighting algorithm is considered a Lighting Operator or LightOp for short (see the next section for a formal definition). Each LightOp takes illumination information as input and generates new illumination information as output after having simulated part of the global lighting effects in the environment, starting with rays from the virtual camera or the light sources, respectively. Although this technique solves the general form of the radiance equation and can therefore account for any lighting effect, it converges rather slowly and is best used for computing illumination via highly specular surfaces.

The basic finite element and Monte-Carlo methods have also been combined in hybrid, two-pass, and multi-pass methods. For example, the irradiance caching technique is based on path tracing but also includes smooth basis functions (similar to radiosity) to interpolate indirect illumination from previously cached computations. Two- and multi-pass techniques try to exploit the advantages of multiple lighting algorithms by combining the effects computed by each. For example, several methods employ a radiosity computation followed by a second view-dependent ray-tracing pass for capturing specular highlights. One method also includes a geometric simplification step for speeding up the radiosity pass.

In addition, several techniques include Monte-Carlo path and photon tracing for simulating specular and caustic light paths. Usually, these methods split the description of reflection and/or illumination into separate parts (high and low spatial frequency components), each simulated with a different lighting algorithm before being combined. Note, this split is the same for the complete scene, and only a specific combination of algorithms is allowed.

In that sense, the Lighting Networks approach builds on top of these methods. It generalizes the concept of hybrid multi-pass techniques and extends it in several ways. Thus, a Lighting Network does not constitute a new simulation algorithm in itself, it provides a way to best use existing algorithms for a specific environment. During development it turned out that Lighting Networks also provide an ideal environment for developing new lighting simulation algorithms and explicitly showing the similarities and differences between lighting simulation approaches.

In the Lighting Networks approach, each lighting algorithm is considered a Lighting Operator or LightOp for short (see the next section for a formal definition). Each LightOp takes illumination information as input and generates new illumination information as output after having simulated part of the global lighting effects in the scene. Thus, many of the lighting algorithms mentioned above can act as a LightOp in our approach with almost no changes.

In contrast to traditional monolithic lighting simulations, Lighting Networks permit arbitrary combinations of these basic LightOps in order to obtain a composite lighting simulation. Formalizing the representations in which illumination information can be taken as input and generated as output lets you connect different techniques into a network of simulation algorithms working in combination. This flexible combination easily obtains special lighting effects difficult to compute with a single or a fixed combination of algorithms. In that sense, Lighting Networks generalize the basic ideas of Shade Trees and similar approaches and apply them to a more
complex domain—global simulation of illumination.

Moreover, the ability to restrict LightOps to certain parts of the scene permits tailoring the global lighting simulation to a user’s needs (see Figure 1). For example, a user can restrict the simulation domain of costly algorithms (like radiance computations) to a small subset of the scene, thereby simulating selected lighting effects even in complex scenes.

Lighting Networks provide an ideal tool in many applications that use automatic lighting computations, such as in producing animations, where computing the full global illumination in a scene is often too costly. However, some selected global illumination effects—like indirect illumination on a setup’s walls—can considerably enhance the animation’s realism. By taking the direct illumination of brightly illuminated surfaces as input for a radiosity solution that generates output only on the walls and considers the rest of the environment only as shadowing objects, we can easily generate the required effects with very little computational effort and thus avoid such tricks as painted illumination maps. Adding special effects like illumination via a mirror or a caustic generated by a wine glass becomes trivial with Lighting Networks.

Several issues need addressing to use Lighting Networks efficiently. This article tackles them in turn.

**Lighting operators and networks**

In the following, we motivate the use of LightOps from the formal solution of the radiance equation. We then demonstrate how these LightOps can easily combine into a Lighting Network, representing a composite lighting simulation.

A lighting simulation algorithm computes an approximate solution to the radiance equation1

\[ L(\mathbf{y}, \mathbf{\omega}) = L_e(\mathbf{y}, \mathbf{\omega}) + L_r(\mathbf{y}, \mathbf{\omega}) \quad \text{with} \]

\[ L_r(\mathbf{y}, \mathbf{\omega}) = \int_{\Omega} f_r(\mathbf{\omega'}, \mathbf{y}, \mathbf{\omega}) L(\mathbf{x}(\mathbf{y}, \mathbf{\omega'}), -\mathbf{\omega'}\mathbf{\hat{y}}_y, d\mathbf{\omega'} \]

where \( L(\mathbf{y}, \mathbf{\omega}) \) is the surface radiance at point \( \mathbf{y} \) in direction \( \mathbf{\omega} \), \( L_e \) is the surface emission, \( L_r \) is the reflected radiance, \( f_r \) is the bidirectional reflectance distribution function (BRDF), and \( \mathbf{x}(\mathbf{y}, \mathbf{\omega'}) \) is a point on a distant surface seen from point \( \mathbf{y} \) in direction \( \mathbf{\omega'} \).

We can write this equation in operator notation12 as

\[ L = L_e + KG L \]

or

\[ ML = L_e, \quad \text{with} \quad M = I - KG \]

where \( G \) is the nonlocal, purely geometric field radiance operator that describes visibility and computes the incident radiance at point \( \mathbf{y} \) from the outgoing surface radiance at other points in the environment, \( K \) is the local reflection operator that computes reflected radiance from incident field radiance, and \( KG \) is the combined transport operator.

Given that the operator norm of \( KG \) is less than one, the radiance equation \( ML = L_e \) can formally be solved by the Neumann series:

\[ L = L_e + (KG)L_e + (KG)^2L_e + (KG)^3L_e + \ldots \]

This equation shows that the solution results from summing over all contributions of light paths involving zero to infinite reflections in the scene.

Since we cannot hope to exactly solve the radiance equation, we actually apply an approximate transport operator \( KG \) in each step along a light path.

We can now define a LightOp \( O \) as computing a set of light paths or a transport pattern, described as a set of products of the approximate transport operator \( KG \). As an example, a LightOp \( O^{\text{rad}} \) that computes traditional radiosity for diffuse reflection would compute all products in the set

\[ O^{\text{rad}} = \left\{ \prod_{i=0}^{N} K_{i}^{\text{Rad}} G \mid \forall N \geq 0 \right\} \quad (1) \]
Basis as input. Sampling IllumLightOp using IllumBasis by a converter to a quadtree illumination contrast, LightOp. In contrast, (b) shows the illumination after conversion to a quadtree IllumBasis by a converter LightOp using the point-sampling IllumBasis as input.

\[
\begin{align*}
    \mathbf{O}^{\text{Rad}} &= \sum_{i=0}^{N} \mathbf{G}^{\text{Rad}}_{i} \\
    \mathbf{O}^{\text{C}} &= \prod_{i=0}^{M} \mathbf{K}^{\text{C}}_{i} \mathbf{g} \\
    \forall M, N \geq 0
\end{align*}
\]

**Representation of illumination**

So far, we have introduced LightOps as a mathematical tool for grouping certain patterns of reflection operators in light paths. However, we aim to implement LightOps as modules in a rendering system that can easily be combined to yield a composite illumination simulation. For this combination to work, different LightOps must be able to exchange illumination data in a common format.

We cannot expect that a single format for illumination data is sufficient to describe the different forms of illumination information that various simulation algorithms compute. We therefore introduce the concept of an illumination basis (or IllumBasis for short). An IllumBasis describes a basis in which illumination is represented in order to be transferred from one LightOp to another.

Commonly used IllumBases are the point-sampling basis for a single outgoing radiance sample \( L(\mathbf{x}, \mathbf{w}) \), a point-sampling basis for an incident light field, consisting of an irradiance value plus a list of incident radiance samples \( I(\mathbf{x}, \mathbf{w}) \mathbf{d} \mathbf{\omega} \), a hierarchical quadtree basis with constant or higher order basis functions \( \mathbf{K}^{\text{Rad}} \) used by many of the finite element LightOps, or the photon map basis \( \mathbf{K}^{\text{Rad}} \) with its implicit radial basis functions computed from the location of a set of photons in the scene. Even the commonly used ambient term \( \mathbf{K}^{\text{Rad}} \) is a trivial form of an IllumBasis that can be made available by radiosity LightOps.

LightOps can only be connected within a Lighting Network if they “speak a common language,” meaning they support a common IllumBasis. Fortunately, many LightOps can easily support multiple IllumBases both for input and output. For instance, a LightOp implementing hierarchical radiosity usually offers both a quadtree and a point-sampling IllumBasis as output. In the latter case, it simply evaluates the quadtree basis at the point sample’s location.

Each LightOp can be queried for the list of IllumBases it supports for input and output. This query interface is used during configuration of the Lighting Network before starting the simulation. It prevents incompatible combinations of LightOps and selects a particular representation for illumination data, in case two LightOps to be connected support multiple common IllumBases.

Mathematically, at each connection between
LightOps the illumination data computed by the upstream LightOp is projected from the basis used by the LightOp internally into the common IllumBasis. In the context of our discussion of operators above, we consider this projection as part of the approximate transport operator implemented by the LightOp. Note that, for a particular LightOp, we can avoid the potential loss of accuracy due to this projection by offering only the computational basis used by the LightOp internally and other closely related bases for both input and output.

**Converter LightOps**

LightOps that do not support a common IllumBasis can still be connected using the concept of a *converter LightOp*. A converter LightOp does not perform any actual lighting simulation, but simply encapsulates a basis change or projection operator that converts from a given input into a given output basis. As it turned out, the overhead of having separate converter LightOps is negligible compared to implicit conversions integrated with other LightOps. Thus, we implemented almost all conversions as separate LightOps. An example for a converter LightOp is the conversion of the point-sampling basis usually offered by the standard direct illumination LightOp into the quadtree basis required by a radiosity LightOp. This converter LightOp samples the illumination via the point-sampling interface and builds an appropriate quadtree basis that captures the direct illumination effects with a given accuracy.

Figure 2 shows the conversion of illumination on some desk items caused by direct and caustic illumination through glass items (black in this image) from a lamp with an internal reflector. The two images show the illumination both using the point sampling and the quadtree IllumBasis.

Converter LightOps offer the additional benefit that common computations can be separated into specialized modules. For instance, we have implemented the sampling of direct or other illumination available via the point-sampling IllumBasis into a converter LightOp that offers a quadtree IllumBasis as output. Thus, the non-trivial functionality of efficient light-source sampling need not be replicated for other LightOps that can take a quadtree IllumBasis as input. This separate implementation now allows for a centralized optimization by techniques such as adaptive shadow testing and others. The use of different conversion and sampling strategies for testing or other purposes can still be connected using the concept of a converter LightOp.

**Domain decomposition**

Now that we have the ingredients to build flexible Lighting Networks, we need to address two other issues before we have a true composite lighting simulation: domain decomposition and reflection classification. Without these, a Lighting Network would be restricted to a simple linear pipeline to avoid accounting for the same light paths in multiple LightOps. We address the first issue in this section and the second in the next section.

Existing lighting techniques usually apply the same algorithm for computing a solution for the complete scene. In contrast, we aim to compute illumination differently for different parts of the scene. This increases the simulation’s speed by restricting costly simulations to regions where they produce visible results or letting a user selectively include or exclude certain lighting effects.

This approach somewhat resembles another, which used a geometrically simplified environment for speeding up the radiosity computation for indirect illumination. Although we do not provide simplification directly in the Lighting Networks concept, restricting the domain of LightOps together with automatic clustering techniques implements the same general idea.

The full domain of a LightOp consists of four individual domains:

- **Input domain**: The input domain specifies the subset of the scene that the LightOp considers as light sources.
- **Output domain**: The LightOp computes illumination only for this subset of the scene. For consistency reasons the output domain of an upstream LightOp must match the input domain of the connected downstream LightOp.
- **Reflection domain**: Only this subset of the scene is included in the reflection computation. In other words, the reflection operator is assumed to be zero for objects not in this domain.
- **Visibility domain**: Finally, the visibility domain specifies those objects that are considered for visibility computations. Thus, it basically specifies the domain of the field radiance operator $G$.

Domain decomposition can be used in various ways. By restricting any of the first three domains of two LightOps to mutually exclusive subsets of the scene, we guarantee that the LightOps compute different light paths. Domains may also help speed up the computation by restricting a LightOp to relevant parts of a scene.

For example, you could assign one LightOp for each room in a building, using the geometry visible from this room for all of its domains. Finally, special effects can be created by limiting the effects of a LightOp to a particular subset of the scene.

Figure 3 (next page) shows an example of using a LightOp restricted to a particular domain. In this case, we used radiosity for computing the indirect illumination in the room. In Figure 3a the radiosity solution is computed on all surfaces. In Figure 3b the ceiling is considered a secondary light source, and the light reflects only on the walls. The second solution required less than one-tenth of the time of using radiosity algorithms with clustering and much less than one-hundredth the time when using standard hierarchical radiosity. For the remaining surfaces we used the ambient term to estimate the indirect illumination. Differences show up as decreased contrast and accuracy, in particular at the table legs.

For the remainder of this article, domains will be identified by *tags*. Each object in a scene is marked with exactly one tag for each type of domain. Thus, the scene’s objects are partitioned into mutually exclusive...
3 Domain decomposition quickly approximates a global illumination solution.

(a) The radiosity solution is computed on all surfaces.
(b) A radiosity solution is computed only on the walls, the floor, and the ceiling; the ambient term is used to estimate indirect illumination for the remaining surfaces.

sets for each domain. LightOps, on the other hand, may have a set of tags for each domain, indicating the subsets of the scene on which they operate. To ensure consistency between output and input domains of connected LightOps, these tags are associated with the connection between them.

We chose this simple scheme to label domains mainly to simplify the following discussion. More elaborate schemes can of course be used for implementing Lighting Networks.

In this article, we use domain decomposition mainly as a tool for restricting the domain of a LightOp. Nonetheless, this information can also be used for distributing the computation of different LightOps over a network of computers or efficiently executing them in parallel threads.

**Classification of reflection operators**

Each of the LightOps in a network computes an approximation to the true local reflection operator \( K \), which is generally given by a BRDF. A simple solution would be to assign each surface to only a single LightOp using the domain decomposition technique just described. However, since many LightOps are often complementary in the reflection effects they compute best, it’s very useful to allow more than a single LightOp to compute reflections on each surface.

An example for this composite reflection computation would be the use of a radiosity LightOp \( O^r \) for computing the diffuse term of a Phong reflection model and the use of a Monte-Carlo photon tracing LightOp \( O^p \) for the specular term, similar to Heckbert. This approach can only be applied if the Phong reflection model is used for all reflection computations. A more general solution must be found for removing this restriction from Lighting Networks. Generally, all that we can assume about a reflection operator is that the BRDF describing the reflection is in \( L^2 \) space and that any of our LightOps will need to project it into some finite dimensional space for its computation anyway.

The fact that the transport operator \( M \) is a linear operator allows for splitting an arbitrary reflection operator into a sum of terms. Each of these terms may then be computed individually and the results can be combined later on. This is equivalent to a set of light paths, where each path evaluates a different term of the reflection operator. Therefore, these light paths and the corresponding reflection computations are complementary and non-redundant. To guarantee completeness of our simulation, we must ensure that the final result contains the contributions of all these light paths.

On the other hand, reflection computations may also be incompatible. This happens if both algorithms account for the same reflection. For example, in addition to the two operators \( O^D \) and \( O^S \) that together approximate the Phong model as introduced above, we can also approximate the complete model by a single diffuse operator \( O^{D'} \) (see Figure 4). Although it’s a less accurate approximation, it can be useful and easily computed by integrating both the diffuse and specular term over the complete hemisphere. In this case, \( O^{D'} \) is incompatible with both \( O^D \) and \( O^S \) because it computes redundant reflection information or light paths.

**Classification scheme**

The added flexibility of using multiple different representations for reflectance requires classifying any two reflection operators as computing redundant light paths or not. We derive our classification scheme by the following formal projection steps (see also Figure 5):

1. projection of the general reflection operator \( K \) into a primary reflection basis \( P \),
2. selection of terms \( b \) of the primary basis to be computed by a LightOp.

First of all, a primary reflection basis is selected for approximating a BRDF. Similar to illumination represented as terms of different illumination bases, a given BRDF can be projected into a number of different reflection bases. In the above example, the surface’s true reflectance has been approximated by using the Phong model as the primary reflection basis \( P^{\text{Phong}} \), with two terms \( b^{\text{Phong}}_D \) and \( b^{\text{Phong}}_S \) for the diffuse and specular reflection. For a different LightOp it might be more appropriate to approximate the same reflection in another primary basis \( P^{\text{Ward}} \) using Ward’s physically based reflection model that has different diffuse and specular terms \( b^{\text{Ward}}_D \) and \( b^{\text{Ward}}_S \).

Each projection of a particular operator is represented in our framework by a ReflectionApprox object that...
identifies the particular reflection basis and offers access to the reflection operator’s different coefficients in this basis. Note that a particular basis need not approximate the full BRDF. Depending on the requirements of a LightOp, a projection of the BRDF with one of its directions fixed will often suffice.

Only a few approaches have been published for projecting general reflection functions—for instance, procedural shaders into a particular basis. Unfortunately, most of them do not suit our task well. Thus, until more general projection algorithms become available, the reflection bases that prove the most applicable are the commonly used reflection models themselves, including simple variations as mentioned above. Spherical harmonics17 or spherical wavelets18 promise to be more appropriate for this task.

In the second step, similar to IllumBases, each LightOp provides a list of primary reflection bases. For each of these bases the LightOp also provides a list of terms that it can account for in its simulation. During the Lighting Network configuration, a single primary basis and those terms to be simulated are assigned to each LightOp.

In combination with a LightOp’s reflection domain, the primary reflection basis and the required terms can be used for classifying its reflection computations. Any two LightOps \( O \) and \( O' \) using different primary reflection bases in the same domain are trivially incompatible, since both compute inseparable approximations of the same reflection effects. Two reflection operators in the same reflection domain are compatible if and only if they use the same primary basis and the two sets of terms they account for have no common terms.

Applying our classification scheme to the example above, all LightOps \( O^D, O^S \), and \( O^C \) use the same primary Phong basis, but the first LightOp is incompatible with the other two due to computing common terms. A LightOp using Ward’s primary basis would be incompatible with any of these three operators.

This formal classification scheme adds almost no overhead to the actual reflection computations. In most cases, the projections are trivial and can be completely omitted in the implementation. However, we need this scheme for determining redundancy and completeness of a Lighting Network, as presented next.

### Lighting Networks and regular expressions

A major issue with the Lighting Networks’ flexibility is that it’s easy to build networks that miss certain light paths or that account for them multiple times. It cannot be expected that an inexperienced user can handle this for nontrivial networks. In the following, we point out a strong connection between Lighting Networks and regular expressions, and their equivalent finite automata—extending the ideas introduced by Heckbert.13

We build on the notion of light paths describing bounces of photons in the environment. Each reflection of a photon is classified and labeled according to the domains of the reflection and the class of reflection operator that the LightOp computing the reflection is simulating. We assign the same label to each class of reflections by constructing an “alphabet” of labels in the following way.

First, we define \( T_i, T_o, \) and \( T_r \) as the sets of all tags in each of the input, output, and reflection domains, respectively. \( T_b \) is the set of all terms of the primary reflection bases. Each reflection of a photon is now uniquely classified by a 4-tuple \( p = (t_i, t_o, t_r, t_b) \), where \( t_i \) is an element of the corresponding sets \( T_i \). The tuple specifies the photon’s source and destination domain, the set of reflecting surfaces, and the term used for computing the reflection. Two reflections compute redundant information if and only if they have different primary bases \( P \) or are identified by the same tuple \( p \). As a result, we can uniquely label the reflection computation by this tuple and thus one bounce in a light path.

Note that we did not include the visibility domain in this formalism. The visibility domain speeds up computations by restricting the visibility computation to a
subset of the scene. In that sense, it only modifies the local field radiance operator \( G \) of a LightOp.

Extending Heckbert’s notation, the base alphabet \( \Sigma \) of our regular expressions now consists of all possible tuples \( p \). Moreover, we also add the letter \( G \) to \( \Sigma \) for marking the transport of photons between surfaces, and \( L \) and \( E \) for marking the start and end of a light path at the light source and the eye, respectively. All possible light paths in the environment can now be described by the regular expression \( L(G(p_0|p_1|...|p_{n-1}))GE \), enumerating all possible light paths that use one of the \( n \) primary reflection bases labeled by one of the tuples \( p_i \) at each bounce. Additionally, converter LightOps map to the empty regular expression, since they never compute reflections but only convert between different illumination representations.

This formalism can be implemented for Lighting Networks by building the regular expressions of each LightOp as a “syntax” graph with the above tuples as nodes. Upon request, each LightOp returns a regular expression for the light paths simulated by its internal configuration. The Lighting Network’s manager then combines the individual expressions to get the full regular expression of the network.

Using well-established results from formal language theory, we can now perform operations on this formal representation of a Lighting Network. The network can be checked for redundant computations by recursively checking that alternatives cannot generate the same patterns. To perform this operation, we simply use Boolean operations on regular expressions. We compute the intersection between every two subexpressions of an alternative and check for the empty expression after simplifying the result. Note that this check for redundancy also lets us localize the redundant computation in the network.

To check the Lighting Network’s completeness, we must compute the complement of the full regular expression and again check for the empty regular expression. Completeness is meaningless if we have restricted the domain of some LightOps on purpose. A solution is to explicitly assign these ignored light paths to a Null-LightOp, which “approximates” the reflection operator by zero and thus terminates any light paths. All this functionality is best hidden behind a suitable user interface.

**Relaxation of the Lighting Network**

Our definition of Lighting Networks specifically allows for cycles in the graph of LightOps. Due to this flexibility, the ordering of computations in the network is nontrivial. Computations in a LightOp may depend on the result of the same computation due to cycles. In that sense, a Lighting Network is in itself a linear system with backward coupling in the presence of cycles. Consequently, a relaxation approach must be used for computing a global solution of a Lighting Network with cycles.

To allow for an efficient relaxation of the network, each LightOp may be requested to prepare its output based on the input illumination’s current state. The LightOp then returns an upper bound on the changes to its output (such as total change in energy) due to these computations. This bound can then be used by the net-
work manager to control the relaxation process.

A simple relaxation procedure starts a recursive computation at the exit nodes of the graph and follows the connections upstream. At each node in the graph we request the LightOp to compute a lighting simulation after all input LightOps have been recursively processed. Cycles are broken where an input node is currently on the stack. In this case, we use the old output of that LightOp, which may be zero. Iterating this recursive scheme results in a Gauss-Seidel relaxation of the complete graph. The information about the amount of changes to the output of each LightOp can be used to terminate the iteration.

A more intelligent method relaxes first those nodes of the graph that are connected to the output of the node that has indicated the largest change.

This relaxation scheme may also be used for progressive refinement of the global solution. Passing an error criterion (such as maximum change in energy or minimum area subdivision) to the solution procedure of a LightOp controls the accuracy of each relaxation step. An initial quick solution to the Lighting Network can be computed by starting with large error bounds, which can then progressively be reduced for later iterations of the relaxation procedure until convergence occurs. This way, coarse results can be made available to the user quickly and be refined over time. Figure 6 shows the progressive refinement of a scene.

Implementation

The Lighting Network technique has been implemented as part of an object-oriented and physically based rendering system.13 In this system, all queries for global illumination are sent to an object of the abstract lighting class, which manages the actual simulation. Many different lighting algorithms had been implemented for this system as derived classes, but it was difficult to use these algorithms in combination.

We extended this class scheme by implementing a new derived lighting simulation class using the Lighting Network approach as described here. Many of the available but isolated algorithms for the lighting simulation class were converted to LightOps and are now available to integrate in a simulation network. We have gradually deleted the old implementations, since they are fully replaced by the new LightOps.

Converting existing algorithms is usually straightforward. The management of domain tags and available IllumBases and ReflectionBases separates into a common LightOp base class. New code must be implemented for processing the input illumination and for checking that computations respect the domains of the LightOp. Projection of illumination from an internal representation into an output basis has either been trivial (an internal basis equal to an output basis), available (radiosity—point sampling of an internal basis), or separated into converter LightOps.

Results

To demonstrate the potential of the Lighting Network approach, we show the different intermediate results of a nontrivial network in Figure 7. Both the Direct LightOp and the Caustic LightOp get their input from the light sources in the scene. They compute the direct illumination and caustic light paths, respectively, where the Caustic LightOp only accounts for the specular term of Ward’s reflection model. A simple combiner LightOp sums the contributions of each of its input LightOps.

The output of the Direct LightOp and the Caustic LightOp are also routed to the input of the Radiosity LightOp. In contrast to the previous connections, we used a quadtree IllumBasis for this connection, which is generated by the conversion module PStoQT. Since the particular radiosity LightOp used in this configuration is based on computations with piecewise constant basis functions, another converter LightOp (QTRec) interpolates the result to obtain a piecewise linear approximation. This reconstructed output is then routed to the combiner LightOp.

To account for ideal specular reflection and transmission in the light path from the eye, we also added a special mirror LightOp to the input of the combiner. It immediately forms a cycle by taking its own input directly from the combiner LightOp.

The images in Figure 8 (next page) show partial results of the lighting simulation that pass through the appropriately marked connections in Figure 7. All these images were computed from the same lighting solution of the Lighting Network by temporarily connecting the output of the respective LightOps to the network’s manager for rendering each of the images.

A major advantage of the Lighting Networks is that the overhead introduced by the network connections is negligible compared to the computation times of the individual LightOps. This even proves true for LightOps that perform little computations per invocation, such as the direct illumination LightOp. The overhead is on the order of two to three function calls per illuminated sample point and thus compares well to other, less flexible multi-pass algorithms.

IEEE Computer Graphics and Applications 29
Future work

In this article we described a new approach to composite lighting simulation. By connecting different simulation algorithms to a Lighting Network—where partial global illumination solutions are passed between the algorithms—we can exploit the particular strengths of each algorithm.

The primary advantage of this approach is the freedom with which we can create individual configurations for the Lighting Network. Using domain decomposition and different representations for illumination, users or developers can easily create custom lighting simulation models. The approach also presents an open framework for composite lighting simulations where advanced algorithms, better basis functions for reflection and illumination, and new solution strategies can easily be integrated. The equivalence between the Lighting Network and regular expression provides means for controlling the freedom of building works and allows for checking their consistency.

While all this makes Lighting Networks an extremely versatile tool for research and development in lighting simulation, it’s best used as a tool for users. Traditionally, global illumination algorithms have offered few opportunities for configuration, tweaking, and creating special effects. Lighting Networks are a tool where the user may select where and to what extent the features of automatic lighting simulation prove appropriate or necessary. By adding special nonphysical LightOps, like subtraction or scaling, Lighting Networks can be a handy tool for a creative designer or animator.

We’ve identified several opportunities for future research. For example, we need to design a suitable user interface before an average user can use Lighting Networks. Modules of preconfigured subnetworks seem like a good approach here and have been used extensively. We must also do more research in representing reflection operators and integrating participating media into Lighting Networks. We have not studied how error analysis can work reliably in a network of simulation algorithms with its added complexity and layered solution techniques. However, due to the strong encapsulation of different operators in this framework, error analysis might be simplified for individual LightOps. As a result, the error analysis of the whole network may actually become simpler because we only need to deal with clearly separated algorithms in LightOps, instead of a single large multi-pass algorithm. The distributed nature of a Lighting Network directly leads to the idea of distributing individual LightOps and computing them in parallel.

Finally, finding ways to (semi-)automatically select appropriate lighting algorithms and their connections for a Lighting Network would be an interesting but daring area of research.

References

Philipp Slusallek is a researcher at the Computer Graphics Group of the University of Erlangen, Germany. He was responsible for the design of a commercial CAD system and is currently leading the Vision project. His research interests include physically based image synthesis and object-oriented software design. He received an MS in physics from the University of Tübingen and a PhD in computer science from Erlangen in 1995.

Marc Stamminger is a PhD student in the Computer Graphics Group at the University of Erlangen in Germany, where he also received his Masters in Computer Science in 1994. Since then he has worked on finite element methods for global illumination computations within the Vision rendering architecture, especially clustering algorithms for radiosity and radiance computations.

Wolfgang Heidrich is a PhD student at the Computer Graphics Group of the University of Erlangen, Germany. He received a Diplom in Computer Science from the University of Erlangen in 1995, and a MMath from the University of Waterloo, Canada in 1996. His research interests include hardware-accelerated and image-based rendering, global illumination, and interactive computer graphics.

Jan Christian Popp studied computer science and psychology at the University of Erlangen, Germany. He finished his Diplom in 1997 with a thesis about multi-pass rendering algorithms within the Vision project. Since then he has worked as a programmer for a multimedia company.

Hans-Peter Seidel is a full professor and chair of the Computer Graphics Group at the University of Erlangen, Germany, and an adjunct professor of computer science at the University of Waterloo, Canada. His research interests include computer graphics, computer-aided geometric design, and high-quality rendering. Seidel is chief editor of Eurographics’ journal, Computer Graphics Forum.

Contact Slusallek at the Universität Erlangen-Nürnberg, Computer Graphics Group, Am Weiselsgarten 9, 91058 Erlangen, Germany, e-mail slusallek@informatik.uni-erlangen.de.

---