Lightcuts: An Implementation In The Physically Based Rendering Code Base

Nico R. Pampe npampe@mines.edu
Chris S. Butler chrbutle@mines.edu

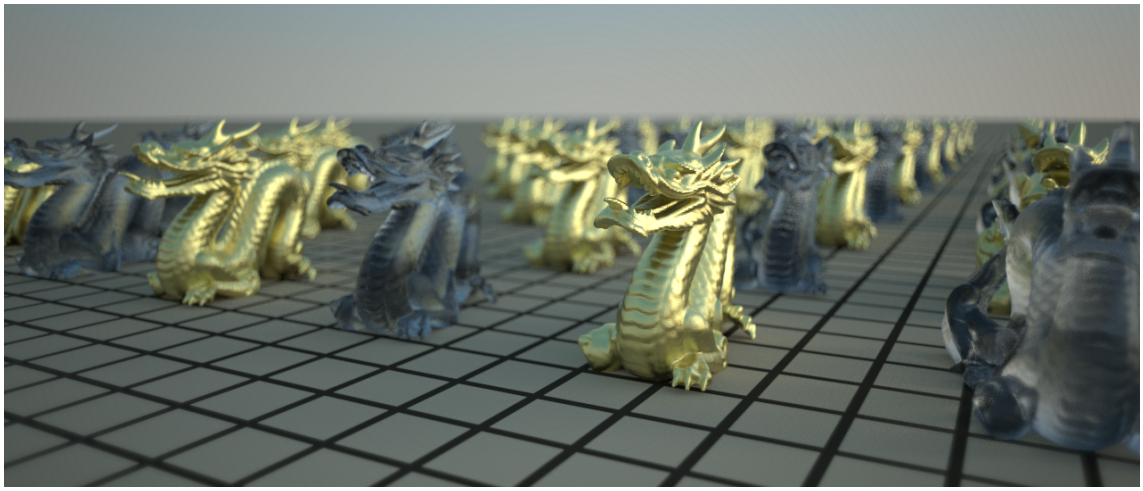

Figure 1: *Demo scene from PBRT-v3 scenes*

Abstract

Lightcuts are a scalable framework for computing realistic illumination. At their basic function, lightcuts help to handle arbitrary geometry, non-diffuse materials, and illumination from a variety of sources. Lightcut is an algorithm developed for accurately approximating illumination from many point lights. It is our goal to implement the lightcuts algorithm into the code base of PBRT [Pharr and Humphreys 2010] following the work describing the lightcut algorithm [Walter 2005].

Keywords: radiosity, global illumination, offline rendering, PBRT

1 Introduction

In the field of computer graphics, rendering a 2D image from a description of a 3D scene, research had been focused on creating scenes with complex geometry and materials. These scenes are geometrically complete and realistic up to the point of lighting the scene, where most of the computation occurs. Costs for rendering typically scale linearly with the number of light sources in a scene. With this in mind, the approach to rendering a scene is usually focused on methods to use a minimal amount of lights, but some scenes require a greater number of light sources. This is where the lightcuts algorithm comes in, which is a scalable algorithm that can efficiently handle computing the illumination of large numbers of point, spot, and directional lights.

The lightcuts algorithm can be implemented into the rendering pipeline following a text book called, Physically Based Rendering: From Theory to Computation (PBRT) [Pharr and Humphreys 2010], to speed up the rendering time for scenes with a large number of lights. The lightcuts framework requires three main steps in order to experience results. First, all lights in the scene are transformed into point, spot, or direction lights. Next, the light rays are traced and virtual lights are created. Finally, a binary tree of the lights in the scene can be constructed. PBRT's code base for version two already had global lights implemented, thus we only had to focus on creating the light tree for efficiency.

Having a scalable algorithm helps in handling extremely large numbers of light sources in a scene. This can make a 2D scene more realistic since shortcuts are no longer needed to limit the number of lights in a model.

The rest of this paper is organized as follows. The project motivation is discussed in Section 2. We present the lightcut algorithm in Section 3 and give details of our implementation in Section 4. Both section have a heavy focus on the required lights needed to build and traverse the binary tree. Results are shown in Section 5, and conclusions are Section 9.

2 Project Motivation

Our previous semester of university featured a course on real-time rendering where we learned about the basic ideas behind computer graphics - including projection matrices, geometric primitives, and simple lighting algorithms. Being real-time, the algorithms and API design were focused on, keeping things real-time. Speed, and by extension parallelism, was prioritized above accuracy. Wanting to explore the space more thoroughly, we began looking into offline rendering.

We did most of this exploration through PBRT and its accompanying code base. Here, physical accuracy was prioritized above raw run-time, which showed in both algorithm and API design. It allowed us to focus more on high level concepts, such as doing computations in units of radiance and then converting to RGB at the end, rather than low level details, like trying to maximize GPU bandwidth utilization. This is not to say that speed was not important - PBRT makes several important design decisions to improve cache performance and reduce memory allocations - but it wasn't a dominating theme. In contrast, using the laws of physics to accurately simulate the scene was.

For this paper, we narrowed down our topic selection with two primary goals. The first was to dive deeper into an important con-

cept with PBRT. We had particular interest in working more with lights and illumination algorithms, the exercise problem presented in PBRT, chapter 15, exercises 15.9, discussed an an algorithm to speed up global illumination [Pharr and Humphreys 2010] was a natural fit. The second goal was to gain experience working on a large, unknown code-base. PBRT-v2 is approximately 60 thousand lines of source code and PBRT-v3 is approximately 190 thousand lines.[1]

3 The Lightcut Algorithm

The lightcut algorithm, as previously stated in Section 1, is an approach to rendering an involved scene with a large number of lights that is geometrically complex. In order for the algorithm to be implemented, the scene must first be transformed into a scene consisting of several lights: point, spot, or directional lights. With the instant radiosity [Keller 1997] approach to global illumination, many virtual point lights can be created in the scene to achieve these workable lights. Then, the lights must be built into a tree such that partitioning of the lights can be done dynamically.

3.1 Workable Lights

For the lightcut algorithm, directional lights are needed to simplify sampling and and tracing. This improves the creation of virtual lights. For a scene, all the lights need to be point. spot, or directional lights. After each light has been transformed, the virtual lights for the scene can be built. The virtual lights each act as an individual point light.

Point, Spot, And Directional: For each of the lights in a scene, they must be converted to either a point, spot, or directional light. This is to improve the virtual light creation. To do this, each type of light has a different approach. Starting with area lights, a grid of spot lights is created to represent the area in which light is transmitted. The grid of spot lights acts as an area light since they shine in one direction but can be used as a cone of illumination. For the edges of the gird, the cones of the spot lights are increased to give the same effect of an area light. This same concept is applied to infinite area lights, where a grid of spot lights is created with a max size. Once the grid reaches the edge of a practical space, the spot lights' cones are increased to cover a plane rather than a small direction. They act as hemispheres on the edges of the bounded infinite grid. The remaining lights, texture project and goniophotometric diagram lights require a clever solution with directional and point lights, the basic idea being to sample from the original light and create either directional or point lights from the samples. [Haines and Greenberg 1986]

Virtual Lights: Virtual lights are generated in a technique that uses instant radiosity to generates a particle approximation of the diffuse radiance in a scene. Virtual lights are created by tracing a ray from a light source and creating a point light at the location of interest. This is where a ray intersects a geometric object and would experience reflection. Instead, a virtual point light is created with the value of how many reflections the ray has gone through, and traces the rays again. After the process is done, the scene has many virtual point lights that have been precomputed which becomes constant time to evaluate. [Keller 1997]

[1]The command `wc $(find -name *.cpp -type f) -l | tail -n 1` was run from each respective project root directory. This command counts the total lines in each C++ source code file, regardless of content. It does not count lines in header files or anything which does not end in `.cpp`. It is intended to be used only as a rough estimate of code base complexity.

3.2 Light Tree

A *light tree* is a binary tree where the leaves are individual lights and the interior nodes are clusters of lights bellow them in the tree. The tree allows quick partitioning of the clusters. A *cluster*, $\mathbb{C} \subseteq \mathbb{S}$, is defined as a set of point lights and a representative light $j \in \mathbb{C}$. With a cluster, the direct illumination can be approximated from the representative light's material, geometric, and visibility terms for all the lights in the cluster, as shown in equation 1.

$$\begin{aligned} L_{\mathbb{C}}(\boldsymbol{x}, \omega) &= \sum_{i \in \mathbb{C}} M_i(\boldsymbol{x}, \omega) G_i(\boldsymbol{x}) V_i(\boldsymbol{x}) I_i \\ &\approx M_j(\boldsymbol{x}, \omega) G_j(\boldsymbol{x}) V_j(\boldsymbol{x}) \sum_{i \in \mathbb{C}} I_i \end{aligned} \tag{1}$$

This gives us the cluster intensity ($I_{\mathbb{C}} = \sum I_i$) which can be precomputed and stored within the cluster. Evaluating a cluster approximation is now equal to the cost of evaluating a single light, thus we can replace a cluster with a single, brighter light. The light tree is now a set of individual lights and nodes of brighter light built from the clusters. With the tree built, a *cut* can be made through the tree that defines a set of nodes such that every path from the root of the tree to a leaf will contain exactly one node from the cut. Each cut corresponds to a valid partitioning of the lights into clusters.

4 Implementation

The focus of the project is to implement the lightcuts algorithm into the PBRT's code base. Currently, version two of PBRT uses a global illumination that takes care of the first two steps for the lightcuts approach. The main focus is building the light tree and rendering the scene based off of the cuts on the tree.

4.1 PRBT's Approach

The following is an explanation of the PBRT's approach to global implementation and virtual lights and, additionally, the implementation of building the light tree. All of the code implementation occurs in the `src/integrators/igi.cpp` Most of the code snippets are left out and can be found at PBRT's github repository[2].

Virtual Lights API: PBRT's implementation of virtual light sources is done mostly in the `IGIIntegrator::Li()`. It samples direct lighting, recursively tracing rays while accounting for perfect specular reflection and refraction. After the direct lighting contribution has been computed at the point being shaded, the contribution from the virtual light sources is found for the fragment. The fragment uses the appropriate sample value from the sample to determine which of the sets of virtual lights to use for the point.

The `IGIIntegrator` used for global illumination creates a 2D array of virtual lights `vector<vector<VirtualLight*>>` that uses a bidirectional scattering distribution function (BSDF) of the last vertex in the lights path. It is approximated by a constant Lambertian term. PBRT allows for easy computation of the virtual lights equation with the values available to the integrator and in the `VirtualLight` structure.

[2]The following is a link to the PRBT version 2 repository: github.com/mmp/pbrt-v2

4.2 Building The Light Tree

In the `IGIIntegrator`, the light tree is is constructed using a system of structures for the nodes and leafs. To start, a `struct LightTreeNodeLeaf` is constructed which contains instances of `Spectrum`, `VirtualLight`, and `BBox` classes. The spectrum is the illumination intensity of the light. The VirtualLight contains they type of light used at the node (i.e. point light). Finally, the BBox is called a bounding box and checks for overlapping illumination of additional lights[3]. Each of these values are needed for the sections of the tree. Next, the `struct Leaf : LightTreeNodeLeaf` is constructed, which implements the values of spectrum, virtual light, and bounding box for a leaf of the tree. Finally, since the nodes of the tree are clusters, a `struct Cluster : LightTreeNodeLeaf` is constructed which has a left and right light-tree-node pointer. In addition, values of the virtual light, spectrum, and an additional bounding box are created for the cluster node. These variables are used when cutting the tree, since the clusters act as point lights from the summation of the light intensities.

In the `IGIIntegrator::Preprocess` function, after the virtual lights have been built, we loop through the nodes by checking each pair. Using a bottom-up approach, each tree is progressively built by combining pairs of lights and/or clusters. Pairs are chosen based off of which of them will create the smallest cluster, according to our cluster size metric $I_{\mathbb{C}}(\alpha_{\mathbb{C}}^2 + c^2(1 - \cos\beta_{\mathbb{C}})^2)$, where $\alpha_{\mathbb{C}}$ is the diagonal length of the cluster bound box and $\beta_{\mathbb{C}}$ is the half-angle of the bounding cone. From the structures, the cluster nodes have already computed their intensity from the left and right intensity lower in the tree.

4.3 Choosing Lightcuts

Using the light tree requires cuts through the nodes to simplify clusters. If done incorrectly, relative errors can manifest. A relative error requires an estimate of the total radiance before a cluster can be declared as usable. A very coarse cut is first used (e.g., the root of the light tree) and progressively refined until an error criterion is met. For each step, the current node in the cut and largest error bound are considered. If the error is greater than the error ratio times the current total illumination, the node is removed from the cut. Then, the two children from the current node are used to compute a new cluster with error bounds and replace the node. Finally, the estimate of the total radiance is updated. Otherwise, the cut obeys the error criterion and the current node is finished. Such a cut is defined as a *lightcut*.

5 Results

Our goals for this project were two-fold. We wanted an opportunity to explore an advanced topic in offline rendering - namely Lightcuts - and to work with a large code base. To this end, we were only partially successful. Our implementation is incomplete, but we learned a lot about working on larger code bases which we hope to re-apply to future endeavors.

We hoped to have an implementation using the light trees to render the scene; to construct them using similar methods as described in [Keller 1997] to have sample/pixel dependent cuts through the tree. Instead, we construct a rudimentary tree in a first-come first-serve manner. During scene initialization we iterate over the virtual lights and construct our tree. We do not make any attempt at grouping lights by spatial locality. We do not keep track of a representative light, although the infrastructure is there. We merely place the lights into a tree and combine intensities as we merge nodes into clusters. The resulting tree keeps track of intermediate intensities for a select, if somewhat unusual, subset of the virtual lights.

[3]The BBox is needed for additional steps in the rendering pipeline not discussed in this paper, but is important to note while building the light tree.

```
 Cluster (11.5, 12.2, 11.5)
| Cluster (5.6, 6.0, 5.6)
| | Cluster (2.0, 2.1, 2.0)
| | | Leaf (1.1, 1.2, 1.1)
| | | Leaf (0.9, 1.0, 0.9)
| | Cluster (3.6, 3.8, 3.6)
| | | Leaf (2.0, 2.1, 2.0)
| | | Leaf (1.6, 1.7, 1.6)
| Cluster (5.9, 6.3, 5.9)
| | Cluster (2.2, 2.3, 2.2)
| | | Leaf (0.5, 0.6, 0.5)
| | | Leaf (1.7, 1.8, 1.7)
| | Cluster (3.7, 3.9, 3.7)
| | | Leaf (2.0, 2.1, 2.0)
| | | Leaf (1.6, 1.8, 1.6)
```

Figure 2: *An example light tree. Each cluster has exactly two children.*

After we construct the tree, we recursively print it off. Nodes are either clusters or leaves. Leaves have no children and terminate that path. Clusters have exactly two children - either more clusters or leaves. Figure 2 shows a snippet of a print out from a scene. The root node is shown first, with its two children indented. Vertical lines provide a visual cue for nodes which are at the same level in the tree. The intensity at the node (the light of a leaf, or the sum of child intensities for clusters) is converted to RGB and printed off after the node type. Here we see that the total intensity of the scene is `11.5, 12.2, 11.5`. This is a straight sum of its two children, which in turn is a sum down until you reach the base leaf nodes. These leaf nodes correspond 1-to-1 to the virtual lights in the scene.

A proper implementation of the algorithm would proceed to use this tree when sampling lights in the scene. We, however, do not. Instead, we construct the tree, print it, and then dispose of it.

Our second main goal for the project was to gain experience working on a large code base and to push ourselves on our "soft skills". Modifying PBRT to add significant new features was an extremely challenging endeavor, which consumed most of our time on the project. Given more time, we believe we would be able to complete our implementation of the Lightcuts algorithm in PBRT. Given the time frame we setup for ourselves, we were unable to reach our goal.

6 Conclusion

The lightcut algorithm is a great way to increase the rendering time of a scene. It can greatly reduce the number of shadow rays needed to computation illumination from a variety and large number of lights. In particular, it specializes in complex scenes with detailed geometry and glossy figures. Given the method for building the light trees and light cuts, these scenes can be reconstructed to further speed shading.

In addition, it was our goal to better understand and learn a large code base, such as PBRT. We spent a large portion of our time exploring the different methods implemented and learning where the light cuts algorithm should be implemented. Most of our time was spent recreating a function or idea into the code base while it was implemented somewhere else. Having worked through the project,

working with a large code base becomes extremely difficult as to not waste time by implementing a method. By the end, we where able to understand how to sift through a project and not get stuck on the small details. These fine points would cause us to lose track of our end-goal.

Acknowledgements

To Dr. Jeffrey Paone, for the early Monday morning meetings.

References

HAINES, E. A., AND GREENBERG, D. P. 1986. The light buffer: A shadow-testing accelerator. IEEE, vol. 6, 6–16.

KELLER, A. 1997. Instant radiosity. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 49–56.

PHARR, M., AND HUMPHREYS, G., 2004-2015. Physically based rendering: From theory to implementation.

PHARR, M., AND HUMPHREYS, G. 2010. *Physically Base Rendering: From Theory to Implementation, Second Edition*. Elsevier, Inc.

SEGOVIA, B., IEHL, J. C., MITANCHEY, R., AND PROCHE, B. 2006. Bidirectional instant radiosity. In *The Eurographics Association*, T. Akenine-Mller and W. Heidrich, Eds.

WALTER, B. 2005. Lightcuts: A scalable approach to illumination. In *Proceedings of SIGGRAPH 2005*, ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series.