Implementing Lightcuts

Miroslav Mikšík*

Department of Computer Science and Engineering
FEE, Czech Technical University
Prague / Czech Republic

Abstract

Conventional techniques for computing direct illumination have linear cost with growing number of light sources. *Lightcuts* is a new technique which has sublinear cost and thus becomes very useful when rendering scenes with a great number of lights. This report describes the basic version of lightcuts technique and provides insight into the implementation problems. We also describe an extension to lightcuts—lightcuts with multiple representatives—which suppresses some visual flaws of the basic version. Finally, the implementation will be verified on a set of test scenes and benchmarks.

Keywords: direct illumination, ray-tracing, shadows

1 Introduction

Conventional techniques for computing direct illumination have linear cost with growing number of light sources. *Lightcuts* [5] is a new technique which has sublinear cost. This technique becomes very useful in applications with a great number of lights such as replacing area lights with point lights, illumination by HDR environmental maps or indirect (global) illumination.

In this report we describe the basic version of lightcuts. There are several extensions to lightcuts such as *reconstruction cuts* [5] and especially *multidimensional lightcuts* [4] providing an effective way to create effects such motion blur, depth of field or volumetric lighting. We will not deal with these extensions in this report but we will descibe an extension to basic lightcuts based on multidimensional lightcuts—lightcuts with multiple representatives—which suppresses some visual flaws of the basic version.

The general idea of lightcuts is to reduce the number of lights used for computing illumination by grouping the similar lights into the clusters. The technique works in two phases. Before the scene is rendered, all the lights are grouped hierarchicaly based on their mutual similarity. This hierarchy is represeneted as the tree of clusters. During the rendering, such a cut of the tree is found that provides the best illumination meeting the given error criterion.

*e-mail: micro@sin.cvut.cz

2 Lightcuts Technique

Given the set of all lights $\mathbb{S}$ in the scene, the amount of light L received from the surface point $\mathbf{x}$ in the direction $\vec{\omega}$ is:

$$L_{\mathbb{S}}(\mathbf{x},\vec{\omega}) = \sum_{i\in\mathbb{S}} M_i(\mathbf{x},\vec{\omega})\; G_i(\mathbf{x})\; V_i(\mathbf{x})\; I_i \qquad (1)$$

The material term $M_i(\mathbf{x},\vec{\omega})$ represents reflective properties of the material from which the direct light is reflected. Usually, it is a product of the BRDF (*Bidirectional Reflectance Distribution Function*) function and the cosine of angle between the direction of illumination and the surface normal. The geometric term $G_i(\mathbf{x})$ represents geometry of the light source and attenuation of the light with growing distance. The visibility term $V_i(\mathbf{x})$ describes whether the light is visible or not. I_i is the intensity of the light.

As we may see, the computation of illumination is directly proportional to the number of lights. We may reduce the number of lights by grouping several lights into *clusters* $\mathbb{C} \subset \mathbb{S}$.

$$\begin{aligned} L_{\mathbb{C}}(\mathbf{x},\vec{\omega}) &= \sum_{i\in\mathbb{C}} M_i(\mathbf{x},\vec{\omega})\; G_i(\mathbf{x})\; V_i(\mathbf{x})\; I_i \approx \\ &\approx M_{\mathbb{C}}(\mathbf{x},\vec{\omega})\; G_{\mathbb{C}}(\mathbf{x})\; V_{\mathbb{C}}(\mathbf{x}) \sum_{i\in\mathbb{C}} I_i \end{aligned} \qquad (2)$$

The more similar lights we group, the better result we get. Material, geometric and visiblity terms of the cluster come from a *representative light* of the cluster. This representative is chosen during the cluster creation and it does not vary during scene rendering. The representative of the cluster is chosen from its children with probability proportional to thier intensities. An individual light is its own representative. A set of clusters such that each light is included in exactly one cluster is a valid partitioning of lights into clusters.

Using single partitioning over the whole scene will not give good results. We have to find a fast way to get good partitioning adaptively for each scene point.

2.1 Light Tree

To create a cluster partitioning we use a helper structure—*light tree*. Light tree is a binary tree where each leaf represents an individual light in the scene and each inner node represents a cluster which contains lights represented by the leaves under the node. If we create a cut such that for each path from a leaf to the root we pass exactly one node of the cut, we get a valid partitioning of lights into the clusters represented by the nodes of the cut. Because each light will be present in exactly one cluster. We will call this cut a *ligthcut*.

To gain the best result we want clusters to group the most similar lights. First of all, we separate the lights by its type (point, directional and oriented) and build a light tree over each type separately. We get 3 trees which we can think of as subtrees of one great tree.

Light tree construction starts with individual lights. In each step we group those 2 lights/clusters which will create the smallest cluster. This procedure is repeated until one big cluster is left—we got the root of the light tree.

The size of the cluster is determined by the similarity metrics:

$$\|\mathbb{C}\| = I_{\mathbb{C}} \left[\alpha_{\mathbb{C}}^2 + c^2 \left(1 - \cos\beta_{\mathbb{C}}\right)^2 \right], \tag{3}$$

where $I_{\mathbb{C}}$ is intensity of the cluster (i.e. the sum of grouped lights' intensities), $\alpha_{\mathbb{C}}$ is diagonal length of the cluster bounding box and $\cos\beta_{\mathbb{C}}$ is half-angle of the cone bounding maximum emission directions of grouped lights. Maximum emission direction is considered only for oriented lights. Point lights does not have such direction. The direction of directional lights will be treated as the position on the unit sphere. Constant c defines proportion between spatial and directional similarity. For oriented lights we set it to the diagonal length of the scene's bounding box. For point and directional lights it has no meaning so we set it to zero.

Tree building is not sublinear and it can be expensive but we need to build it only once. Moreover, we can use the same tree for multiple animation frames as far as the lights are kept unchanged.

2.2 Light Cuts

When computing illumination on the surface point $\mathbf{x}$, we need to find an optimal tree cut which will be a compromise between the cut size (and hence the number of the lights used) and the error caused by the cluster approximation. This cut is created by progressive refinement of the partitioning until the maximum error criterion (usually 2% of total radiance) is met.

We start with a very coarse cut (e.g. the root of the tree). For each cluster in the cut we compute the estimated radiance (2) and its upper bound (4). Then we select the cluster with the greatest upper bound. If its relative error (i.e. ratio of the error upper bound (4) and the total estimated radiance (2)) is greater than our error criterion we replace this cluster by its children. This is repeated until all the clusters have relative error less than the criterion. In dark places the size of the cut can grow unduly so we define a maximim limit of the cut size (usually 1000).

Since both the exact (1) and the estimated (2) radiance is less than the radiance upper bound, the radiance upper bound is also an upper bound of the maximum error $\varepsilon_{\mathbb{C}}$ (i.e. the absolute difference of exact and estimated radiance). The radiance upper bound is calculated from the Equation (2):

$$\varepsilon_{\mathbb{C}} \leq L_{\mathbb{C}}^{\urcorner}(\mathbf{x}, \vec{\omega}) = M_{\mathbb{C}}^{\urcorner}(\mathbf{x}, \vec{\omega})\; G_{\mathbb{C}}^{\urcorner}(\mathbf{x})\; V_{\mathbb{C}}^{\urcorner}(\mathbf{x}) \sum_{i \in \mathbb{C}} I_i \tag{4}$$

All we need to do now is to find upper bounds of material, geometric and visibility terms of a cluster.

2.2.1 Visibility Term

The visibility term has values between 0 (totally invisible light) and 1 (totally visible light). It is difficult to calculate its upper bound so we have to do with $V_{\mathbb{C}}^{\urcorner}(\mathbf{x}) = 1$.

2.2.2 Geometric Term

The geometric term for point, oriented and directional light is:

Light Type	Point	Oriented	Directional
$G_i(\mathbf{x}) =$	$\dfrac{1}{f_A(\|\mathbf{y}_i - \mathbf{x}\|)}$	$\dfrac{\max(\cos\phi_i, 0)}{f_A(\|\mathbf{y}_i - \mathbf{x}\|)}$	1

where $\mathbf{y}_i$ is the light position, $f_A(d)$ is the attenuation function in form of $A + Bd + Cd^2$ (coefficients A, B, C are non-negative) and ϕ_i is the angle between the direction of maximum emission and the direction $\mathbf{x} - \mathbf{y}_i$.

Upper bound of directional light is obvious: $G_{\mathbb{C}}^{\urcorner}(\mathbf{x}) = 1$. For point light we have to find a lower bound of the function $f_A(d)$. Since this function is non-decreasing we have to find a lower bound of the distance d which is the distance of the point $\mathbf{x}$ from the cluster $\mathbb{C}$, therefore $G_{\mathbb{C}}^{\urcorner}(\mathbf{x}) = 1/f_A(\|\mathbb{C}, \mathbf{x}\|)$. Oriented light is similar to point light but in addition we have to find an upper bound of cosine of the emission angle ϕ_i. We could use simple bound of one but we can do better that this.

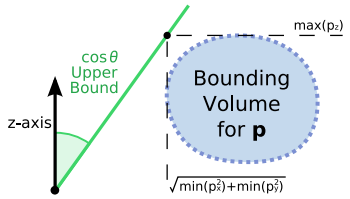


Figure 1: Lower bound of the angle (and hence the upper bound of its cosine) for the bounding volume.

Consider a simpler case shown in Figure 1. For each point $\mathbf{p} = [p_x, p_y, p_z]$ we have an angle θ between the origin-to-$\mathbf{p}$ direction and the z-axis. Cosine of that angle is:

$$\cos\theta = \frac{p_z}{\sqrt{p_x^2 + p_y^2 + p_z^2}}$$

By maximalizing the nominator and minimalizing or maximalizing the denominator (depending of the sign of the nominator) we get an upper bound of the cosine:

$$\cos\theta \leq \begin{cases} \frac{\max(p_z)}{\sqrt{\min(p_x^2)+\min(p_y^2)+(\max(p_z))^2}} & \max(p_z) \geq 0 \\ \frac{\max(p_z)}{\sqrt{\max(p_x^2)+\max(p_y^2)+(\max(p_z))^2}} & \text{otherwise} \end{cases} \quad (5)$$

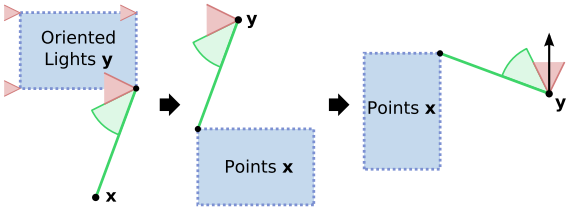


Figure 2: Transformation of the cluster of oriented lights. After this transform we can use Equation (5) to get an upper bound of the emission angle.

This approach can be generalized by the transform depicted in Figure 2. For each light let's have pair of points $[\mathbf{y}_i, \mathbf{x}]$. We translate each pair by the vector $-\mathbf{y}_i$ resulting all lights to be positioned to the origin and creating a cluster of points $\mathbf{x}_i = \mathbf{x} - \mathbf{y}_i$. This cluster has the same dimensions as the original cluster of lights. Consecutively, we apply such transform that projects bounding cone orientation onto the z-axis. We got the problem from Figure 1 and thus we can use Equation (5). If the bound angle is less than the half-angle of the bounding cone (i.e. it lies within the cone) we use the cosine upper bound of one. Otherwise we use cosine of the difference between the bound angle and the bounding cone half-angle.

2.2.3 Material Term

The material term has usually form of a product of the BRDF function and the cosine of the angle between the surface normal at $\mathbf{x}$ and the direction $\mathbf{y}_i - \mathbf{x}$. To bound the cosine we use the method descibed in the previous section leaving the BRDF function to bound. Equation of the modified Phong BRDF is:

$$f_r(\Theta_i, \Theta_o) = k_d \frac{1}{\pi} + k_s \frac{n+2}{2\pi} \cos^n \alpha, \quad (6)$$

where k_d is diffuse reflectivity, k_r is specular reflectivity, n is specular exponent and α is angle between the perfect reflection direction and the eye direction. Only the angle α depends on the light position and it can be bounded by the already known method.

Not only Phong, but any BRDF can be used if we are able to estimate its upper bound. For example, upper bound of Ward BRDF can be found in [3].

3 Building a Light Tree

In section 2.1 we have introduced the light trees as they are defined in the original lightcuts paper [5]. However, the paper does not descibe how to build such a tree, even though it is the most expensive part of the lightcuts technique. If we used a naive approach we would have to compare all cluster to each other in each step of clustering, select the closest ones, group them and start over again. This approach has cost $\mathcal{O}(n^3)$. But we have a way how to build it at a better cost of $\mathcal{O}(n \log n)$.

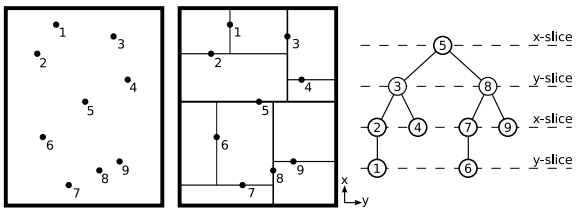


Figure 3: Example of a kd-tree in 2D.

We start with individual lights. For each light[1] we find its closest neightbour in metrics (3) and we insert this pair into a heap priority queue. We fetch from the queue the pair with the smallest distance. If this pair is valid (i.e. the clusters have not been grouped so far) we create a new cluster out of them, find its closest neighbour and put this new pair into the queue. In the case we have fetched a pair where the first cluster (i.e. the cluster for which we have been looking the closest neighbour) is already grouped, the pair can be discarded. If the second cluster is already

[1] We may think of the individual lights as of the clusters having one light, therefore from now on, we will refer to lights as the clusters.

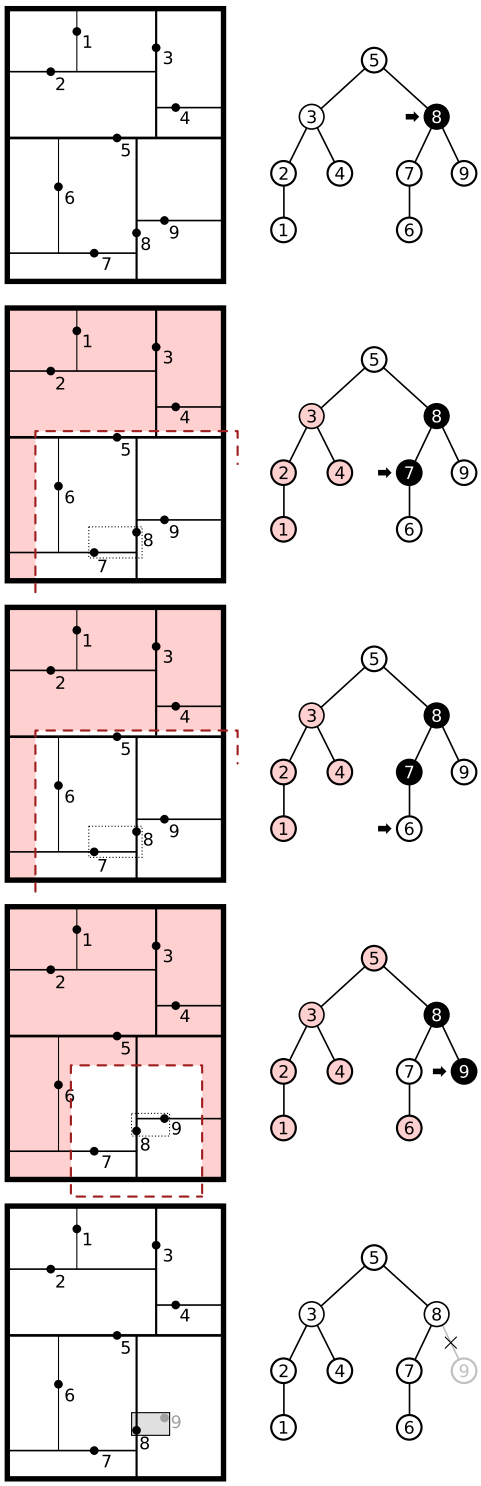


Figure 4: Searching for the closest neighbour of point 8. We search the subtree of node 8 considering the entire space. Each time we find a closer neighbour as we have we determine the distance boundary where it is possible to find a closer neighbour. The space beyond this boundary is left out from search.

grouped we find the new closest neighbour and insert the new pair into the queue again. This action is repeated until one big cluster is left—the root of the tree.

To find the closest neighbour effectively we use a kind of BSP tree (*Binary Space Partitioning Tree*)—a *k*d-tree [1] in our case. In *k*d-tree, each node represents a point which defines a plane (always perpendicular to one of the coordinate axes) splitting the space into two halves. As we go deeper into the tree the space is being split into smaller parts. Figure 3 shows a simplified example of a *k*d-tree.

A *k*d-tree is built as follows. We choose a coordinate axis (e.g. x) and we sort all points[2] by this coordinate. We choose the median point and create a node out of it. The node contains a pointer to the associated light. We repeat the procedure on the newly created halves but with the y-coordinate, etc. We rotate the axes consecutively and repeat until all the points are in the tree. At this stage each node represents one light.

We should understand that the *k*d-tree is used only for the nearest neighbour lookup. The way of choosing the split plane does not affect the final light tree at all.

Metrics (3) respects not only positional similarity but also clusters' intensities and orientations, therefore the *k*d-tree search must be modified (see Fig. 4). Searching for the closest neighbour of the point **A**, we start with the subtree of the node **A** and then go to the upper levels of the tree. The search is stopped when we are sure there is no better solution. Each time we find a point closer than we have we determine the distance boundary where it is possible to find a closer neighbour. This boundary is usualy greater than the Euclidean distance between **A** and the found point because it also respects intensities and orientations of lights (see Appendix A).

When two lights are grouped into a cluster we update the corresponding *k*d-tree nodes so that they point to the newly created cluster. Even more, we may remove one of the *k*d-tree nodes to prevent cluster from being tested more than once. Usually, when a node is removed the entire *k*d-tree should be rebuilt. That is expensive. Instead of that, we mark this node as invalid and we do not remove it until its entire subtree is invalid (see Fig. 5). To decide which node sould be marked we use the one that is on the lower level in the *k*d-tree.

4 Lightcuts With Multiple Representatives

The basic lightcuts version we described uses the same representative light for cluster during entire rendering process. Even though the representative is chosen randomly

[2] By "point" we mean position of the light.

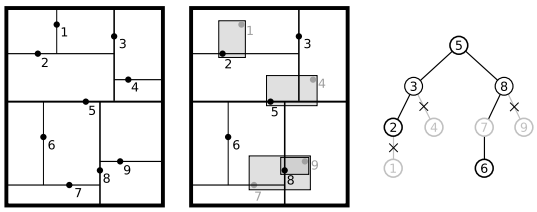


Figure 5: Removing of the nodes during clustering. After grouping two lights, one of the corresponding nodes is marked as invalid. The node is removed if its entire subtree is invalid.

during the light tree construction this feature induces a biased error resulting in artifacts. These artifacts are usualy below visibility threshold but they become visible in shadows' penumbrae (caused by loose upper bound of visiblity term). Having several representatives would allow us to choose actual representative randomly for each pixel and thus to convert a biased error into noise which is perceptually more pleasant (see Fig. 6).

Multidimensional lightcuts can handle multiple representatives and we have incorporated similar functionality into the basic lightcuts. Each cluster is given a list of its representatives. Representatives of a cluster are chosen from representatives of its children during the tree construction. Similarly to basic lightcuts, the probability of child's representative to be chosen is proportional to its intensity. The number of cluster's representatives is determined as follows.

Having clusters $\mathbb{C}$ and $\mathbb{D}$ which are children of cluster $\mathbb{C}\cup\mathbb{D}$, let $\lambda(\mathbb{C})$ be the number of representatives of cluster $\mathbb{C}$. Assuming $\lambda(\mathbb{C}) \geq \lambda(\mathbb{D})$, cluster $\mathbb{C}\cup\mathbb{D}$ will have either $\lambda(\mathbb{C})$ or $\lambda(\mathbb{C})+1$ representatives with probrability proportional to $\lambda(\mathbb{D})$:

$$P\left[\lambda(\mathbb{C}\cup\mathbb{D}) = \lambda(\mathbb{C})+1\right] = \frac{\lambda(\mathbb{D})}{\lambda(\mathbb{C})} \qquad (7)$$

This means that if both child clusters have the same number of representatives the parent cluster will have one more. Otherwise the number of representatives will be incremeted with probability proportional to balance between child clusters.

Because λ can increacse in each step at most by one the root will have at most h representatives, where h is the height of the light tree. For most scenes the light tree is almost balanced but there is possibility we get a very unbalanced tree, therefore we should define a maximum limit of λ (usually 16).

When computing illumination for the cluster we randomly choose one actual representative from representative list with probability proportional to representatives' intensities. To be able to find such representative the list is sorted by increasing intensity and each item of the list contains precomputed value of cumulative distribution function. The representative is found by picking a random number $\langle 0,1)$ and by scanning the list (using binary partitioning) for the first representative with distribution value greater than the random number.

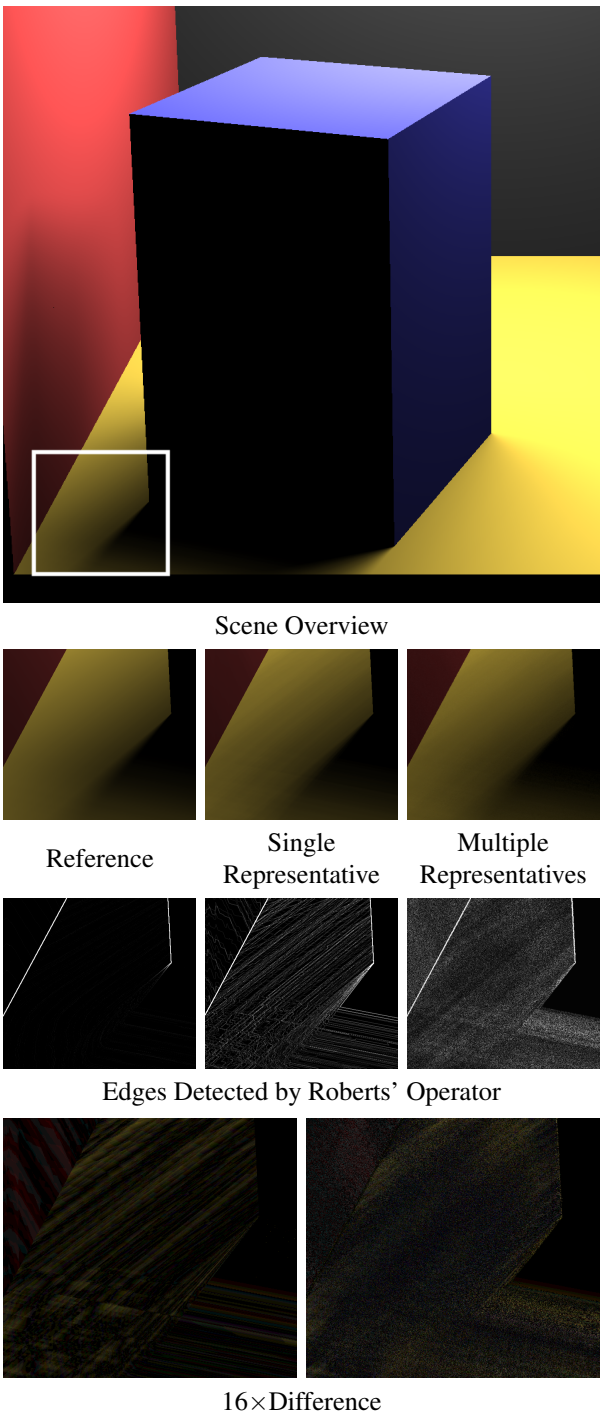


Figure 6: Comparison of basic lightcuts with one representative and modified lightcuts with multiple representatives. A biased error is converted into noise which is perceptually more pleasant.

5 Results and Conclusion

Lightcuts have been implemented in our C++ ray-tracer GOLEM [2]. To test our implementation we have used several scenes with high number of lights (see Appendix B). Table 1 compares the render times for conventional technique and lightcuts. According to measured times we may pronounce our implementation to be successful. Acceleration is obvious and output image quality is comparable to reference images. We may see that gained speedup varies a lot because it depends on the distribution of the light sources in the scene. We may also see that using multiple representatives produces almost no overhead.

Appendix C compares output quality of lightcuts for different error thresholds. Thanks to using multiple representatives, the recommended error threshold of 2% may be increased up to 10% without significant loss of quality and gaining about three times greater speedup. For preview rendering we may also use higher values.

Lightcuts have a great potential. Having strongly sublinear cost, this technique becomes very useful in applications with a great number of lights such as replacing area lights with point lights, illumination by HDR environmental maps or indirect (global) illumination. However there are still possibilities of improvement: we may look for tighter bounds or we may find a way to predict dark areas where light cuts grow excessively. At the time being, we work on implementation of multidimensional lightcuts which will allow us to illuminate multiple points in a single cluster refinement pass.

Model	Stairs (800x600)	Cornell Box (800x800)	Living Room (800x600)
Number of polygons	14826	32	31492
Number of lights	10000	10000	12000
Render time (conventional)	1:51:03	1:56:01	1:40:56
Tree build duration [sec]	1.1	0.9	1.2
Average cut size	185	203	237
Render time (basic lightcuts)	0:06:22	0:07:24	0:10:41
Render time (mult. repr.)	0:06:24	0:07:31	0:11:23
Speedup	**17.4**×	**15.4**×	**8.9**×

Table 1: Comparision of render times for conventional technique and lightcuts (both the basic version and the version with multiple representatives). Render times are given in *hours* : *minutes* : *seconds*.

References

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[2] Vlastimil Havran. GOLEM Rendering System, 1997. http://www.cgg.cvut.cz/GOLEM/.

[3] Bruce Walter. Notes on the Ward BRDF. Technical Report PCG-05-06, Cornell Program of Computer Graphics, April 2005.

[4] Bruce Walter, Adam Arbree, Kavita Bala, and Donald P. Greenberg. Multidimensional lightcuts. *ACM Trans. Graph.*, 25(3):1081–1088, 2006.

[5] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: A scalable approach to illumination. *ACM Trans. Graph.*, 24(3):1098–1107, 2005.

A Determining a Distance Boundary for *k*d-tree Search

Given a cluster $\mathbb{C}$ with intensity $I_{\mathbb{C}}$ and axis-aligned bounding box defined by minimum and maximum coordinates $\mathbf{B}_{\mathbb{C}} = [[x_0, y_0, z_0], [x_1, y_1, z_1]]$, let $d = \|\mathbb{C} \cup \mathbb{D}\|$ be the distance of clusters $\mathbb{C}$ and $\mathbb{D}$ in the metrics (3). By minimizing the intensity and orientation similarity terms in the Equation (3) we get an upper bound of diagonal length $\alpha_{\mathbb{C}\cup\mathbb{D}}$ of the box bounding both clusters $\mathbb{C}$ and $\mathbb{D}$:

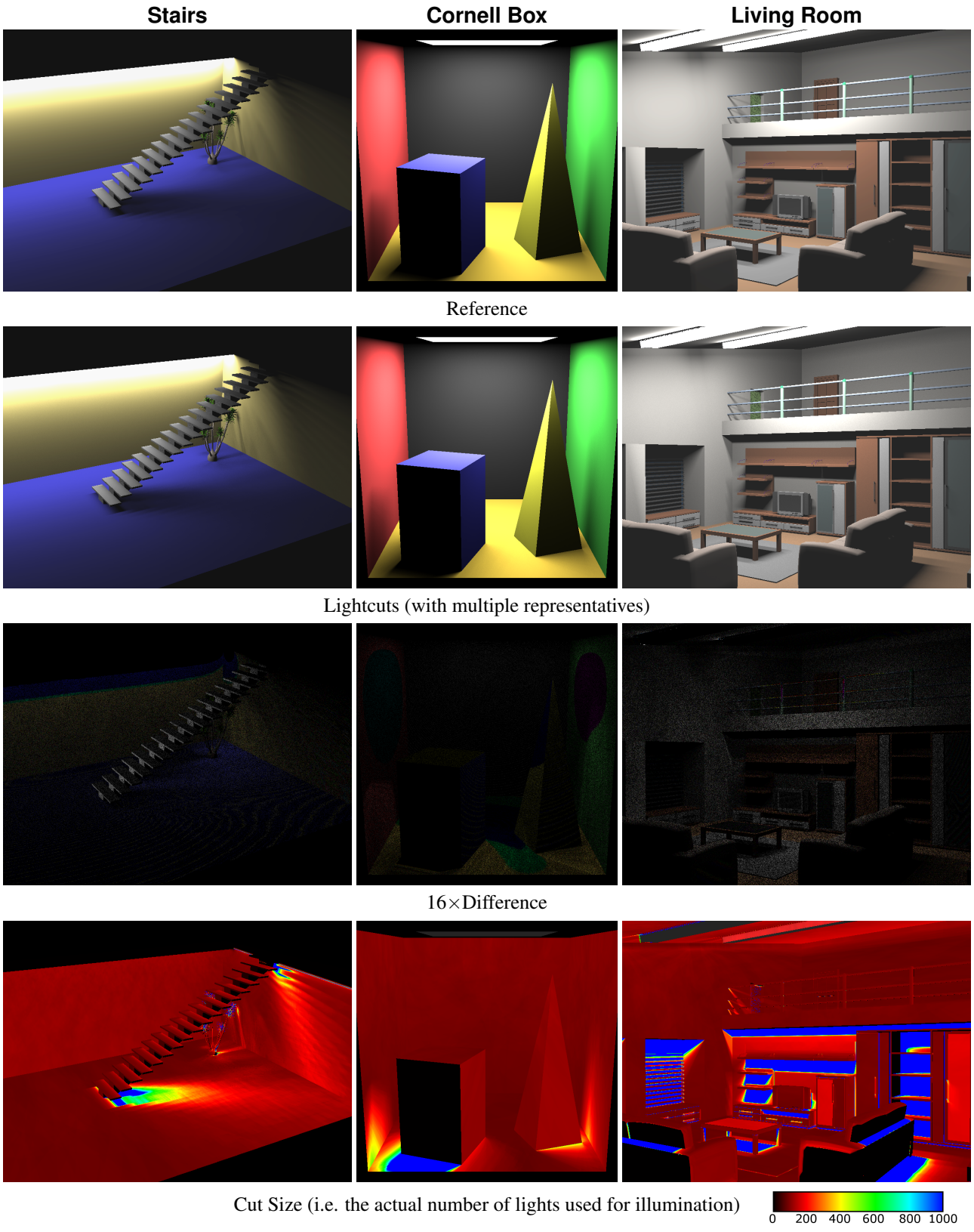
$$\alpha^2_{\mathbb{C}\cup\mathbb{D}} \leq \frac{\|\mathbb{C} \cup \mathbb{D}\|}{I_{\mathbb{C}}} \tag{8}$$

Coordinates of that box are therefore:

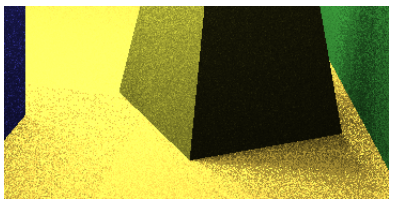
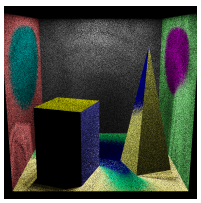
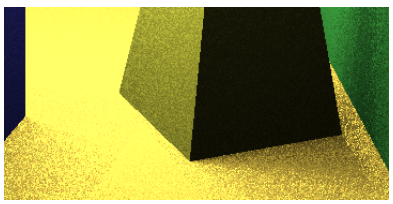
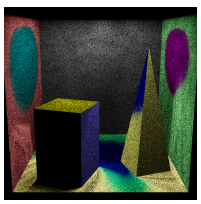
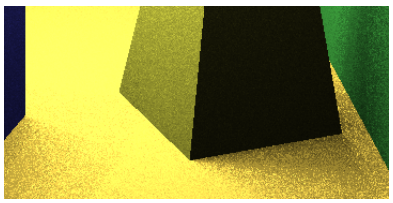
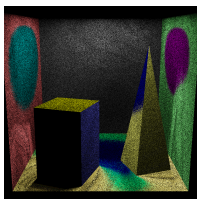
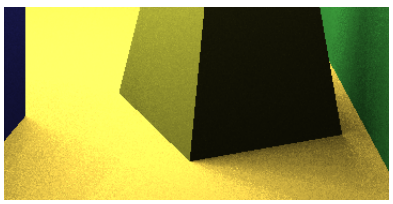
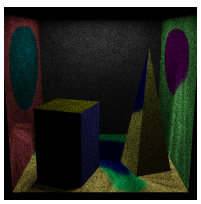
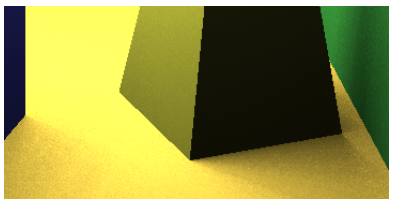
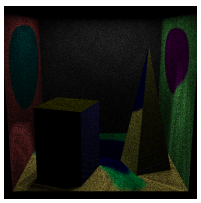
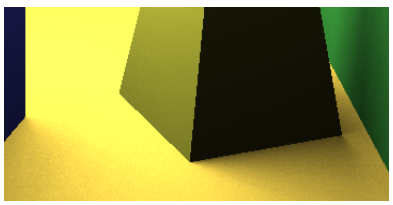
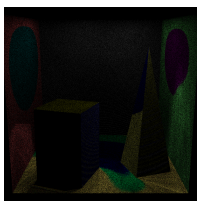
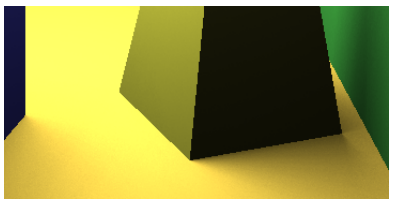
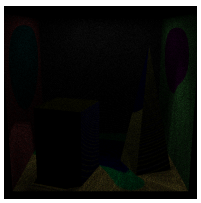
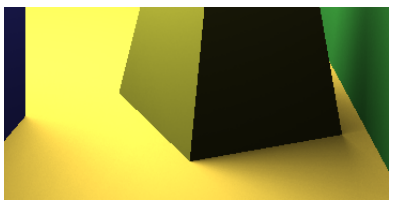
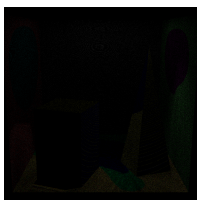
$$\mathbf{B}^{\neg}_{\mathbb{C}\cup\mathbb{D}} = [[x_1 - u, y_1 - v, z_1 - w], [x_0 + u, y_0 + v, z_0 + w]],$$
$$\begin{aligned} u &= \sqrt{\alpha^2_{\mathbb{C}\cup\mathbb{D}} - (y_1 - y_0)^2 - (z_1 - z_0)^2} \\ v &= \sqrt{\alpha^2_{\mathbb{C}\cup\mathbb{D}} - (z_1 - z_0)^2 - (x_1 - x_0)^2} \\ w &= \sqrt{\alpha^2_{\mathbb{C}\cup\mathbb{D}} - (x_1 - x_0)^2 - (y_1 - y_0)^2} \end{aligned} \tag{9}$$

where u is the maximum width, v is the maximum height and w is the maximum depth of a bounding box of diagonal length $\alpha_{\mathbb{C}\cup\mathbb{D}}$.

B Lightcuts Test Scenes



C Comparing Lightcuts Visual Quality for Different Error Thresholds

Result Detail	16×Difference	Error Threshold	Render Time [*hrs* : *min* : *sec*]	Speedup
		90%	0:00:59	118.0×
		70%	0:01:03	110.5×
		50%	0:01:11	98.0×
		20%	0:01:50	63.3×
		10%	0:02:33	45.5×
		5%	0:03:54	29.7×
		2%	0:07:31	15.4×
		1%	0:13:21	8.7×