Scalability with Many Lights 1

Lightcuts & Multidimensonal Lightcuts

Bruce Walter, Cornell University

Course: Realistic Rendering with Many-Light Methods

Note: please see website for the latest slides.

These slides are for the SIGGRAPH 2012 course Realistic Rendering with Many-Light Methods. This part is titled Scalability with many lights, part one and will be presented by Bruce Walter from Cornell University. These slides are subject to change until the actual time of presentation, so please see the course website to get the latest version.

Talk Overview

- Scalable solutions for many light rendering, Part I
  - Illumination at a single receiver: Lightcuts
    - "Lightcuts: a Scalable Approach to Illumination" by Walter, Fernandez, Arbree, Bala, Donikian, Greenberg, SIGGRAPH2005
  - Illumination over a pixel: Multidimensional Lightcuts
    - "Multidimensional Lightcuts" by Walter, Arbree, Bala, Greenberg, SIGGRAPH2006

This section will discuss the lightcuts approach to scalable many light rendering. It primarily covers the ideas from these two papers: Lightcuts and Multidimensional Lightcuts. First I will discuss the lightcuts method for efficiently computing the illumination at single point from many lights, then I will describe the multidimensional extension which computes the illumination over the entire region corresponding to a pixel.

Why many lights?

- Simulate complex illumination using point lights
  - Area lights
  - HDR environment maps
  - Sun & sky light
  - Indirect illumination
- More lights ➔ more accurate
  - And more expensive
  - Naive cost linear in lights

Area lights + Sun/sky + Indirect

Once we have a scalable solution for many point lights, we can use this to compute many types of complex illumination. The four examples are area lights, high dynamic range environment maps, sun & sky light, and indirect illumination. Moreover unified handling of different illumination types and enables new types of tradeoffs. For example, bright illumination from one source allows coarser approximations of other sources.

Accurate Approximation

SIGGRAPH2012

- Do we really need to evaluate all lights?
  - Thousands to millions of lights
  - Average contribution miniscule
  - Below human perceptual limits
    - Weber's Law (roughly 2%)

Textured area lights & indirect

- Evaluate a small subset
  - Chosen adaptively
  - Perceptually accurate approximation

Using more point lights creates a more accurate simulation, but once we have thousands to millions of point lights, evaluating them all would be expensive and wasteful, as the contribution from individual lights is often too small to be noticeable.

Scalable Algorithm

- Scalable solution for many point lights
  - Sub-linear cost per light

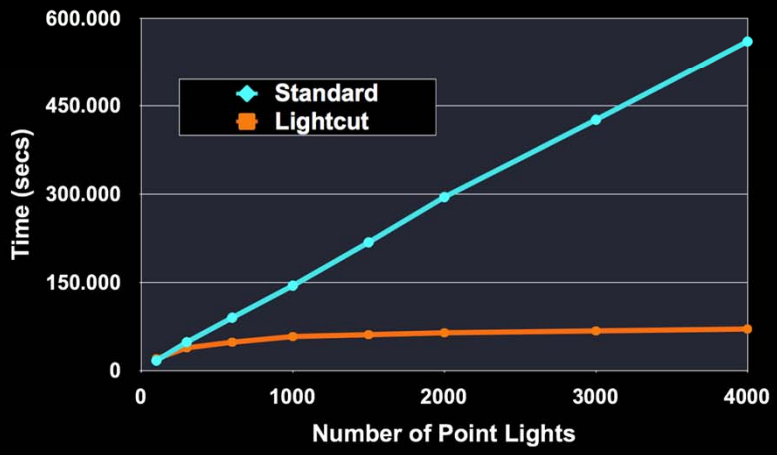
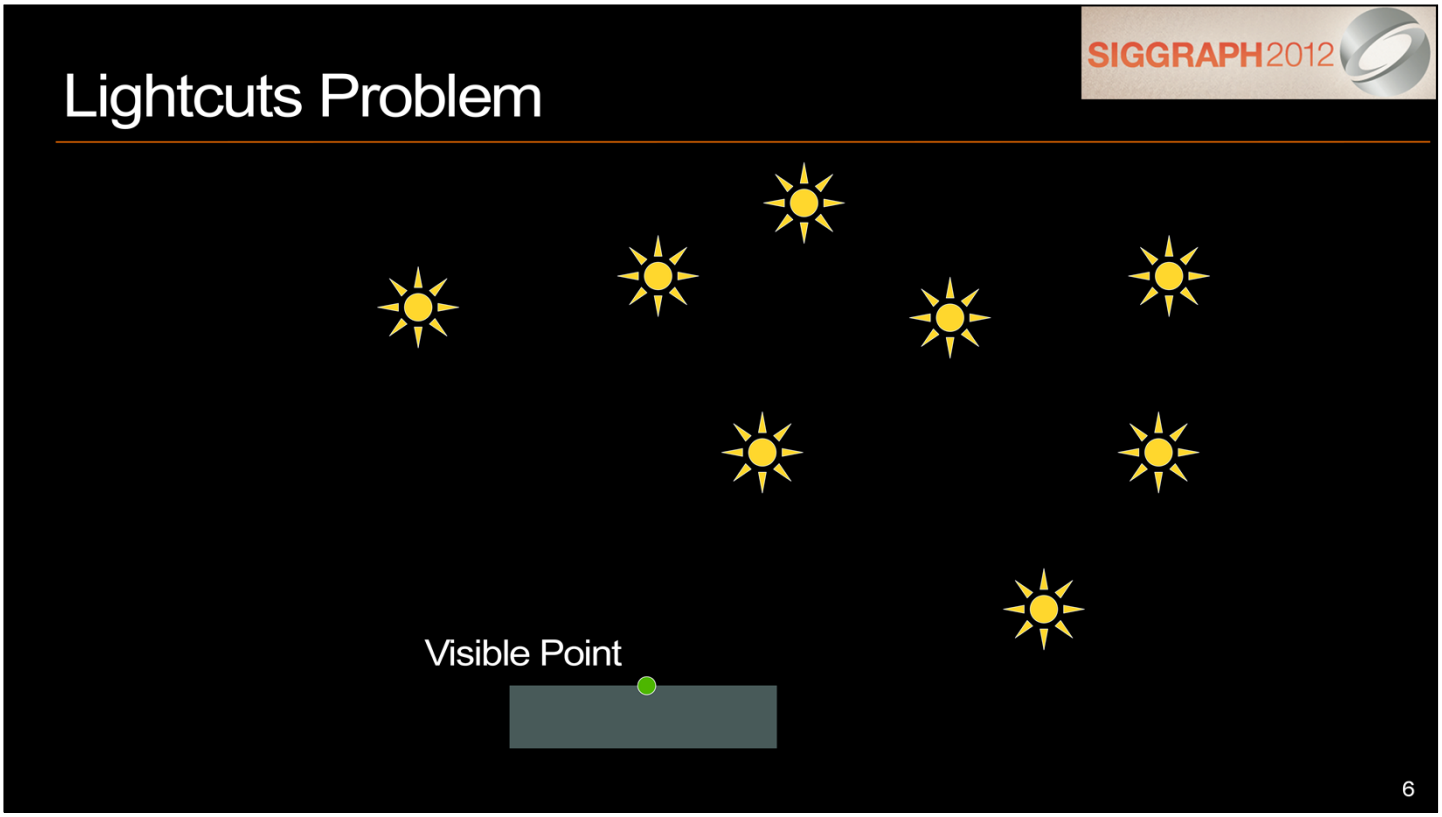


Tableau Scene
Environment map lighting & indirect

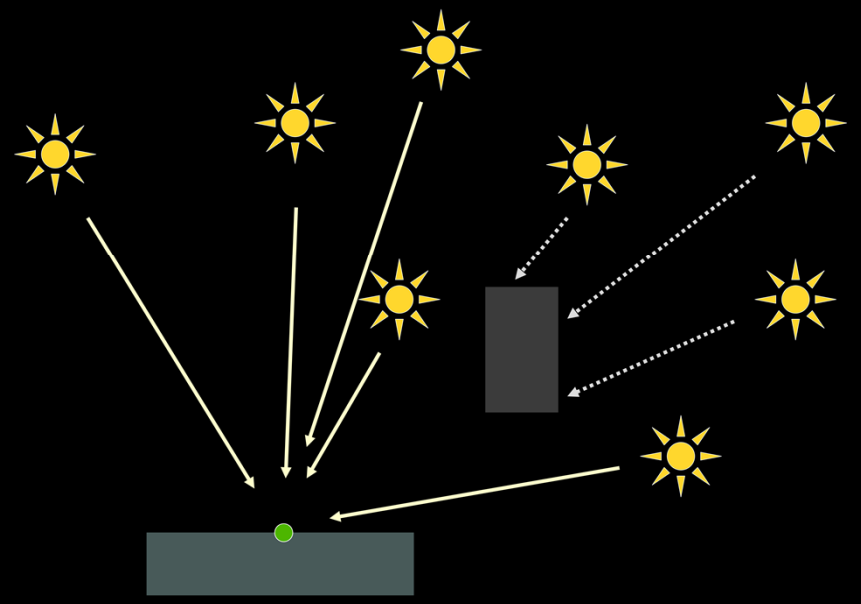
The core of lightcuts is a new scalable algorithm for efficiently approximating the light from many point lights. By many I mean thousands to millions. Here we show the time to compute this tableau scene with environment map illumination using varying numbers of lights. Evaluating each light individually gives a cost that increases linearly with the number of lights. Lightcuts' cost is strongly sub-linear and thus its advantage grows dramatically as the number of lights increases.

Lightcuts Problem



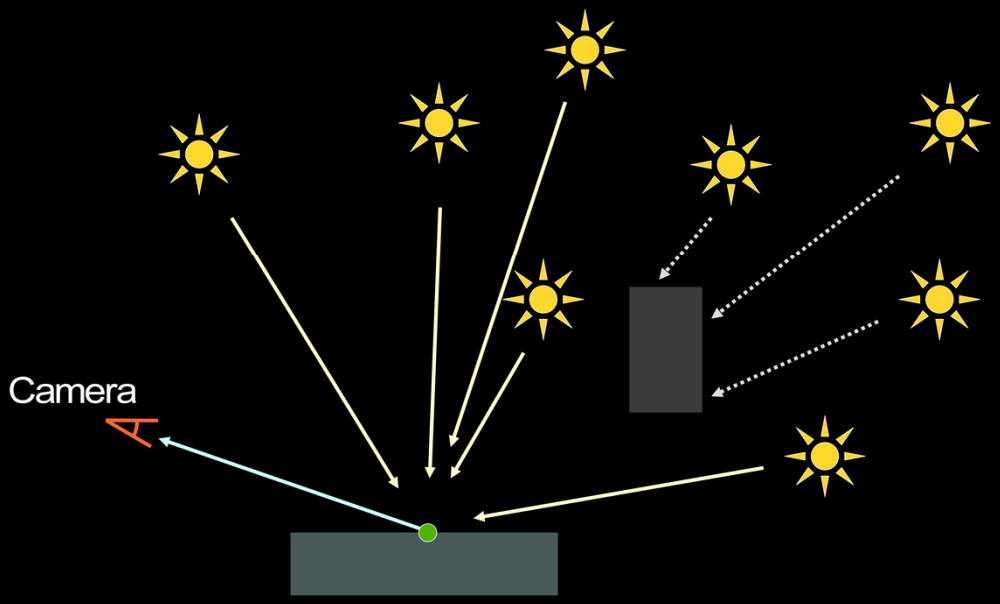
Given a set of points we want to compute their contribution at some point of interest.

Lightcuts Problem



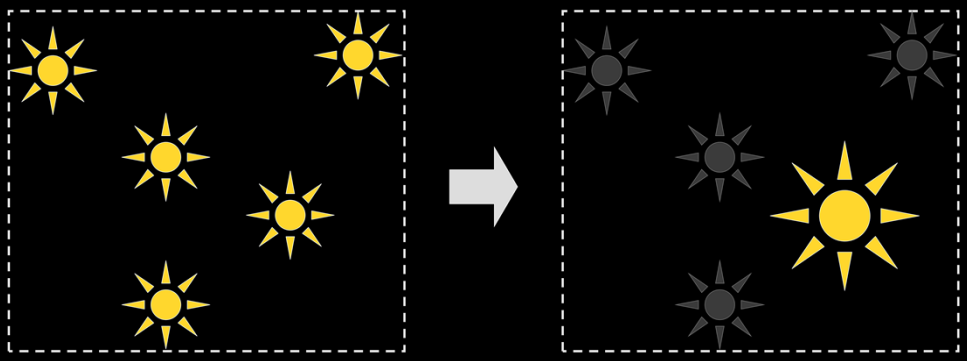
The lights do not contribute equally as some may be occluded.

Lightcuts Problem



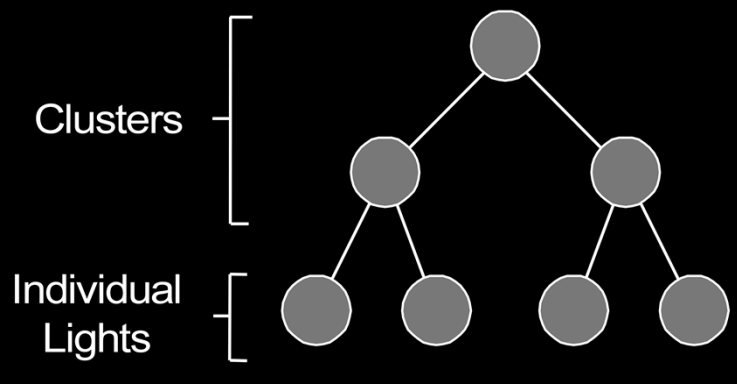
And ultimately we are interested only in light reflected towards a camera, so the material properties also effect their contribution.

Key Concepts



There are a few key concepts we need to understand the lightcuts approach. First is a light cluster where we approximate a group of lights by replacing them by a single brighter light called the representative light.

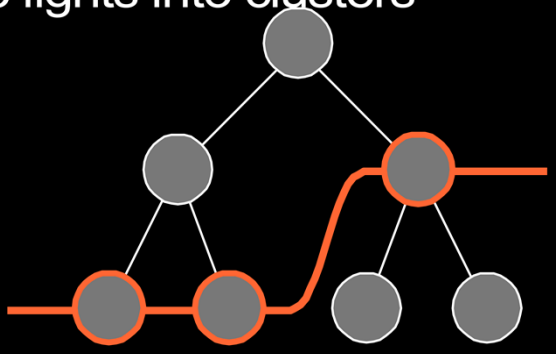
Key Concepts



Second is the light tree which is a binary tree of lights and clusters. The leaves of this tree are the individual lights while the interior nodes represent clusters which get progressively larger as we go up the tree.

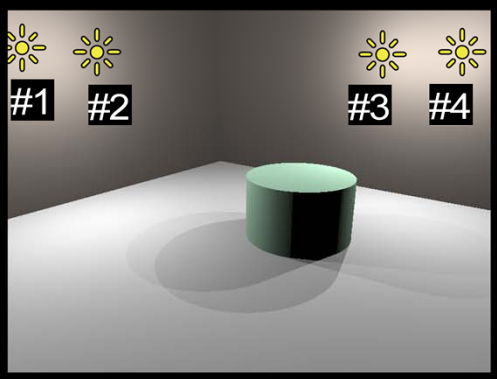
Key Concepts

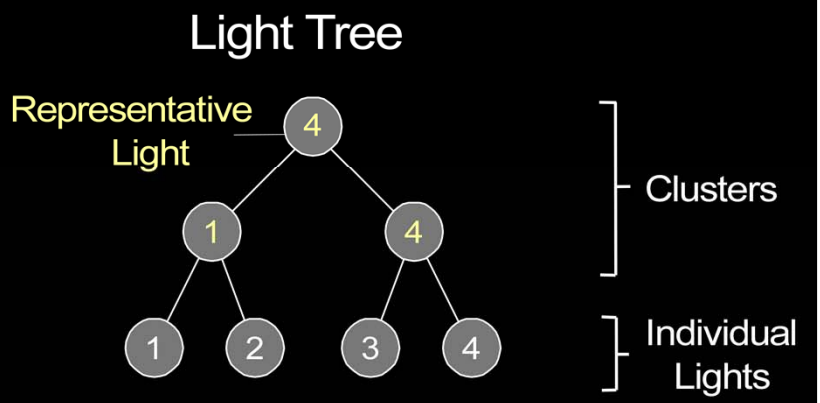
- Light Cluster
- Light Tree
- A Cut
  - A set of nodes that partitions the lights into clusters



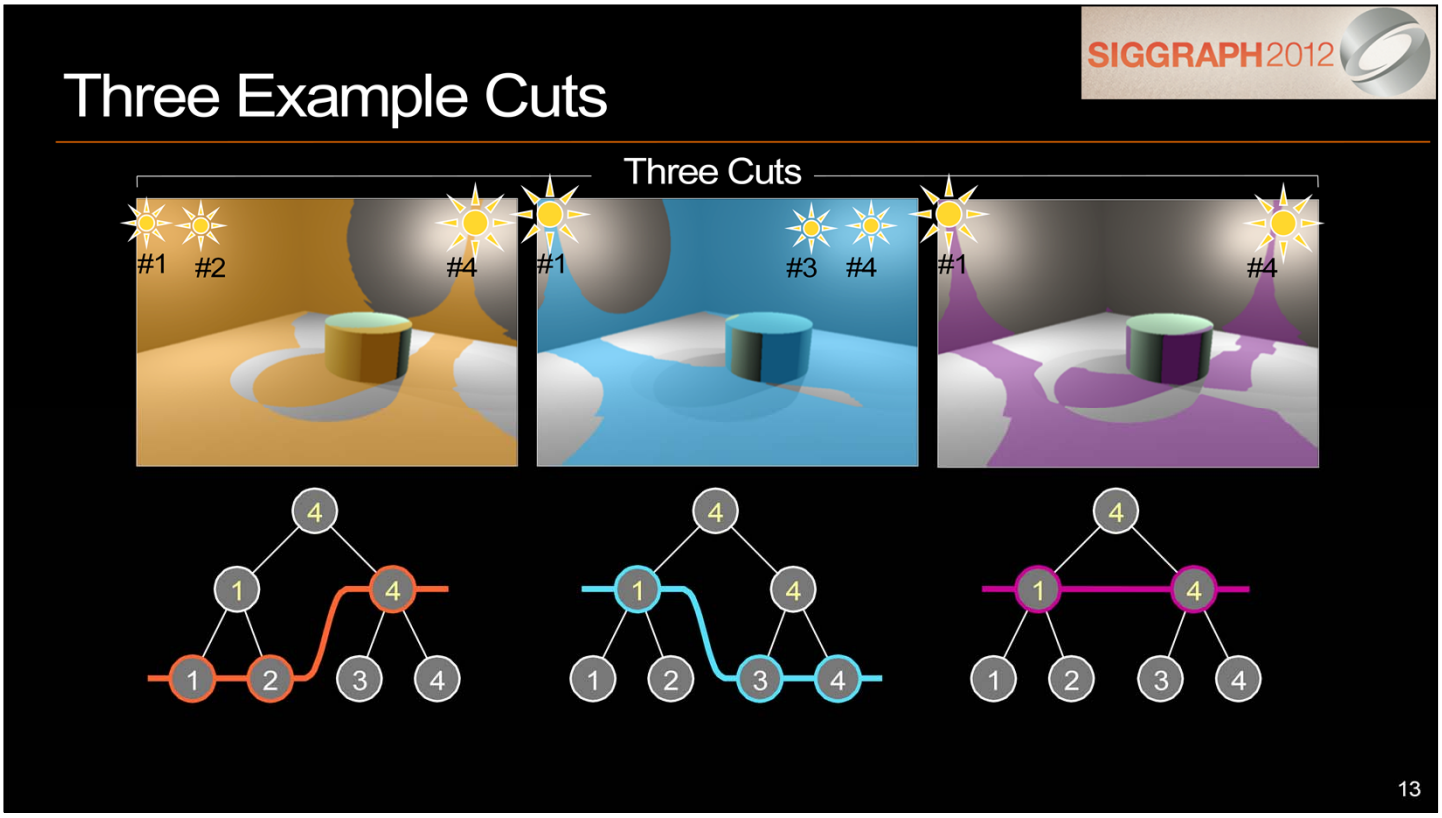
The third is a cut which is a set of nodes that partitions the lights into clusters. Here illustrated by the orange line.

Simple Example

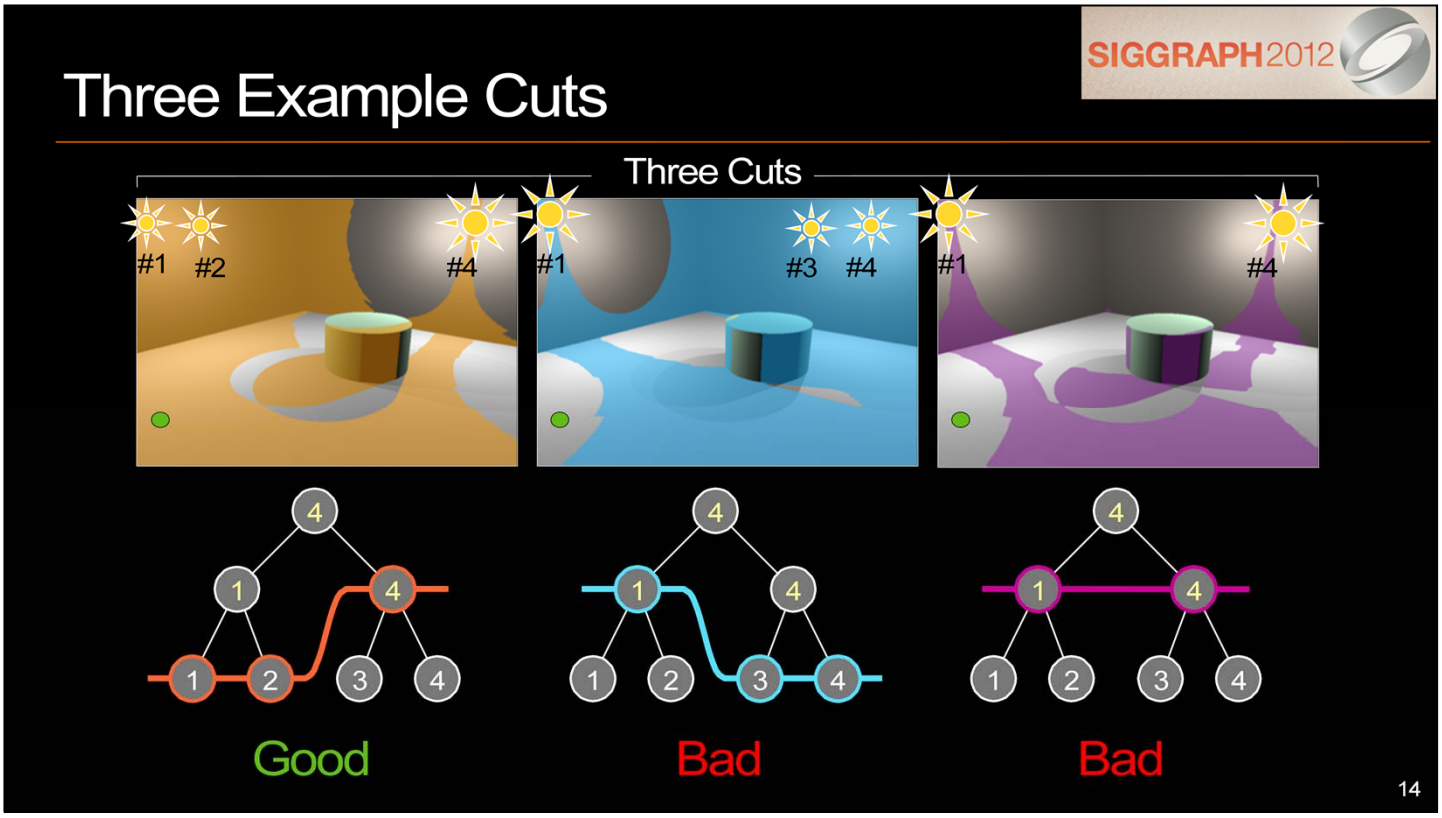




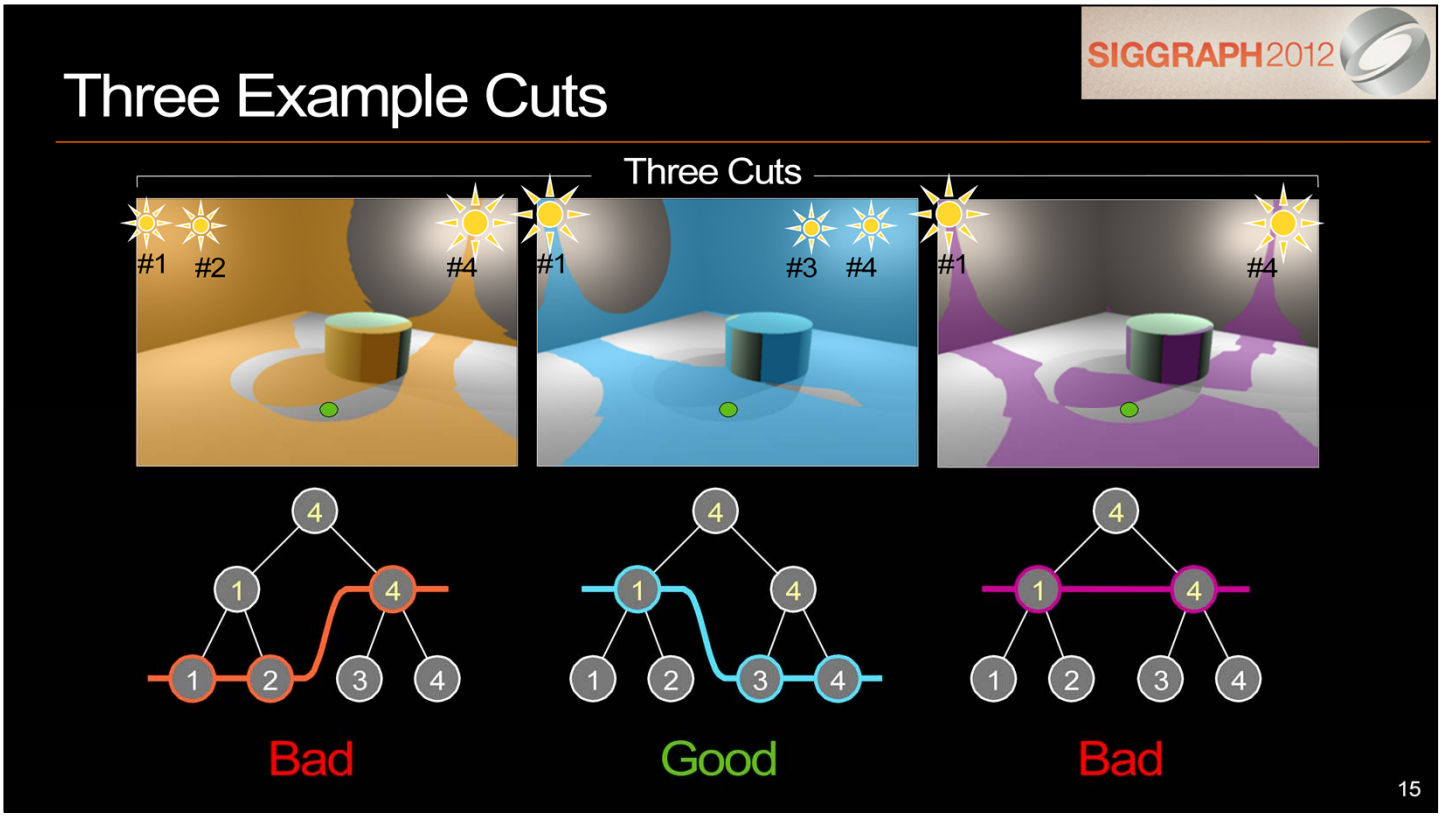
Here is a simple scene with four lights and its corresponding light tree.



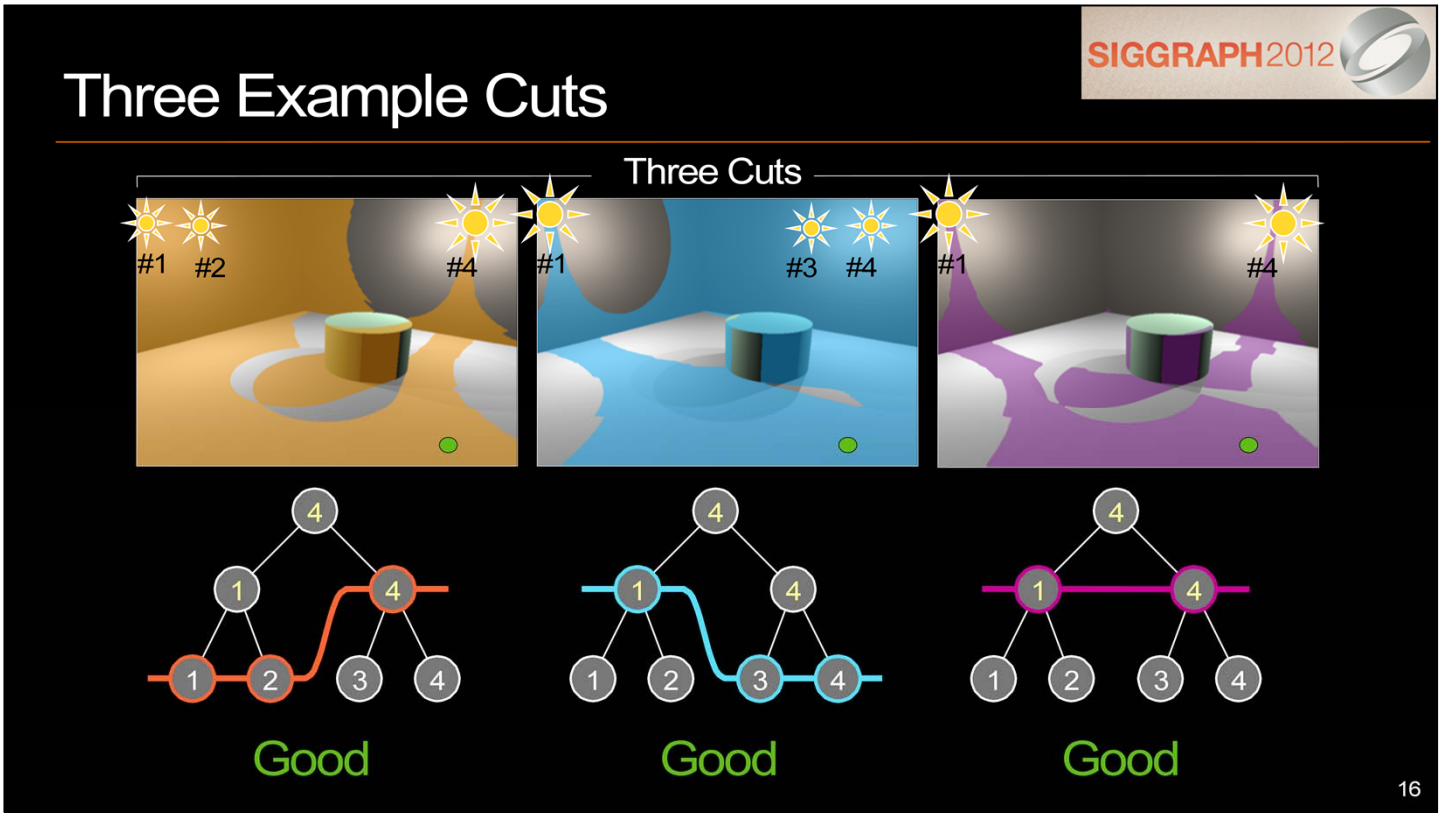
And here we show three example cuts through the light tree. Highlighted above each cut are the regions where that cut produces an accurate approximation of the exact illumination.



If we look at this green point on the left of the images, the orange cut produces a good approximation while the blue and purple cuts would cause too much error.



Conversely for this point in the center of the images, the blue cut is a good approximation while the orange and purple cuts are not usable. This illustrates an important point. We will want to use different cuts in different parts of the image.



For this point on the right side of the images, all three cuts usable. In this case the purple cut is the best choice because it will be the cheapest to compute as it contains the fewest nodes.

Algorithm Overview

- Pre-process
  - Convert illumination to point lights
  - Build light tree
- For each visible point
  - Choose a cut to approximate the local illumination
    - Bound maximum error of cluster approximation
    - Refine cluster if error bound is too large

To generate an image, first we perform a particle tracing to generate the VPL (virtual point lights). Then we build the light tree using a simple greedy approach. The more difficult problem is choosing the cuts. There is a cost vs accuracy tradeoff of cuts higher in the tree are cheaper while cuts lower the in tree are more accurate. We also need to avoid transition artifacts. Since we will use different cuts in different parts to the image, there will be transitions between places where we use a cluster and places where we refine it, and we don't want these to produce visible artifacts. We will actually use this to drive the cut selection.

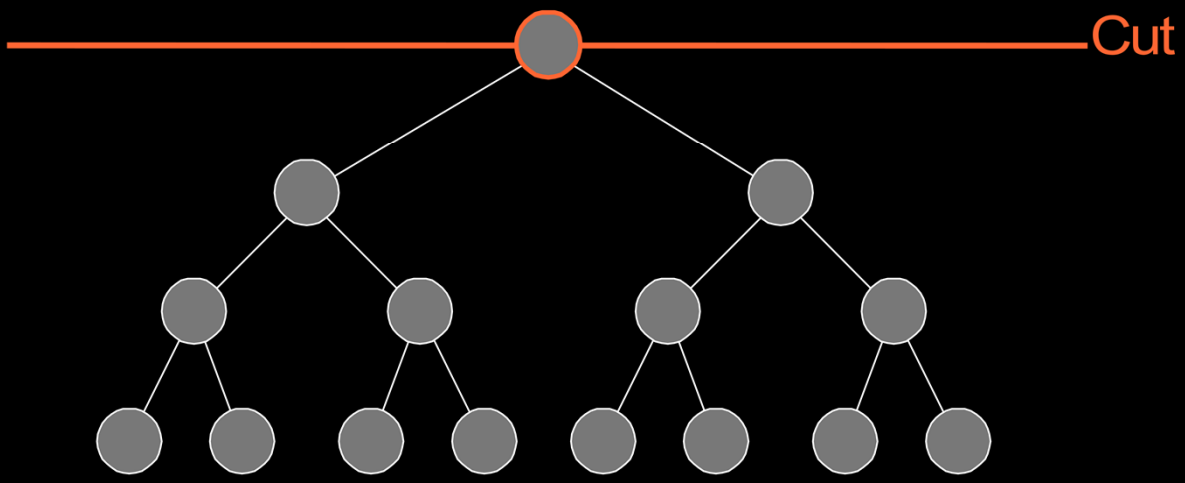
Perceptual Metric

- Weber's Law
  - Contrast visibility threshold is fixed percentage of signal
  - Used 2% in our results
- Ensure each cluster's error < visibility threshold
  - Transitions will not be visible
  - Used to select cut

We can use Weber's law which say the the minimal visible change is roughly a fixed percentage of the total signal. For our results we used a threshold of 2%. We can ensure that cluster transitions are not visible by guaranteeing that the clusters error always be less than this threshold. But we will need to be able to put an upper bound on the error introduced by a cluster.

Cut Selection Algorithm

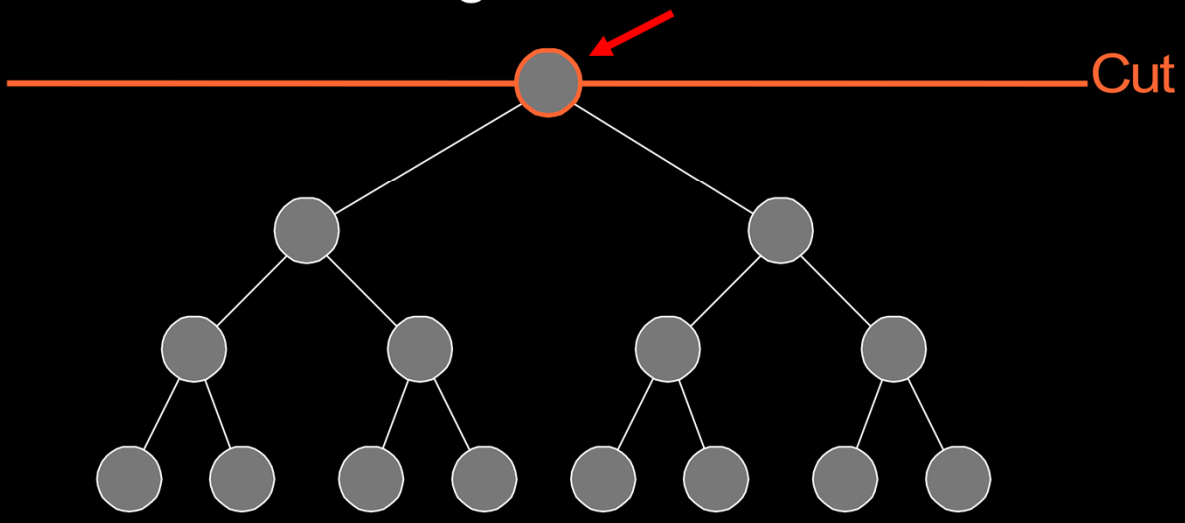
- Start with coarse cut (eg, root node)



We start with a coarse cut such as the root node of the tree.

Cut Selection Algorithm

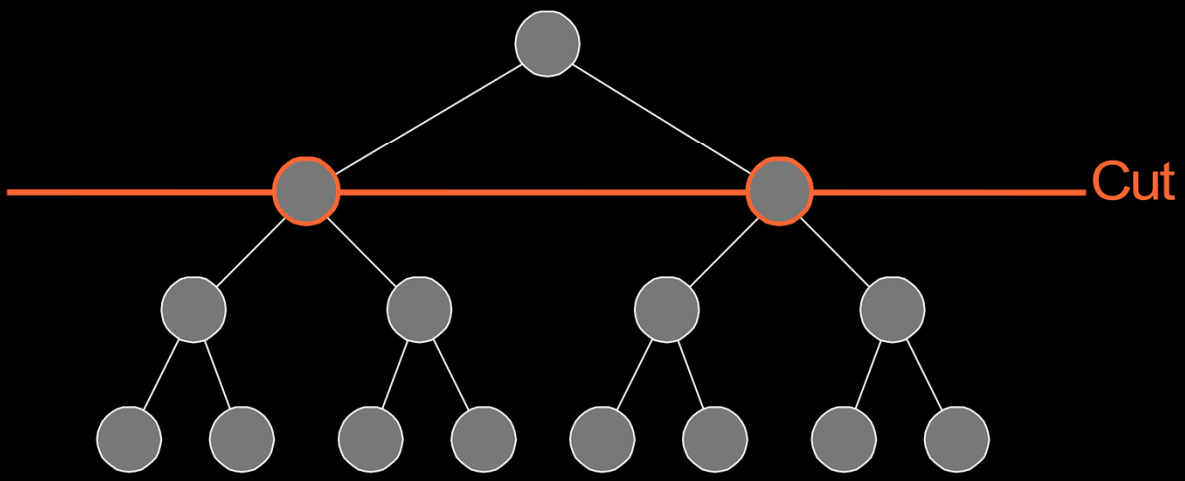
- Select cluster with largest error bound



Then we select the node with the largest error bound. (the root node in this case).

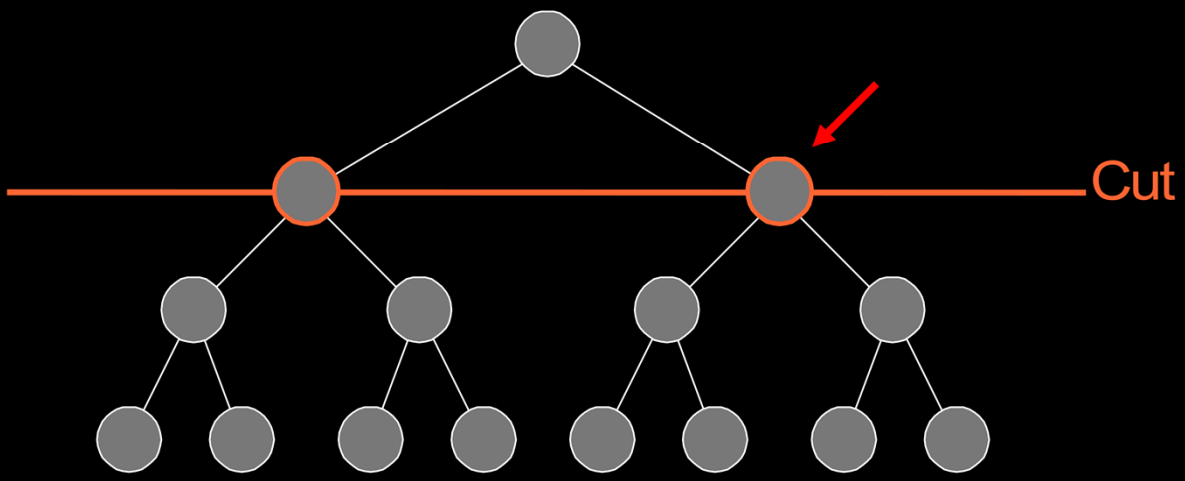
Cut Selection Algorithm

- Refine if error bound > 2% of total

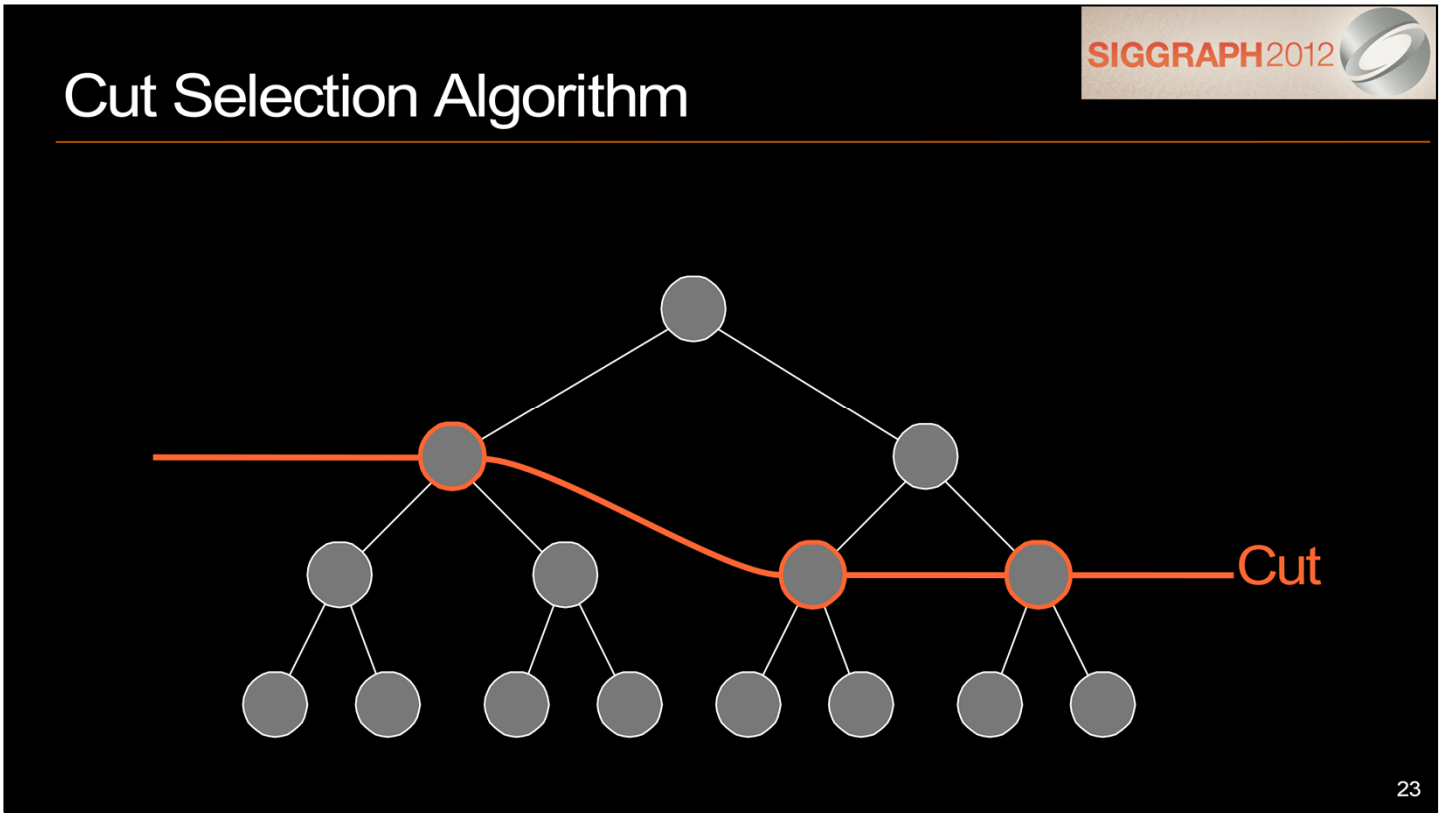


And refine this node if its error is greater than 2% of the estimated total. Refining means removing a node from the cut and replacing it with its children.

Cut Selection Algorithm

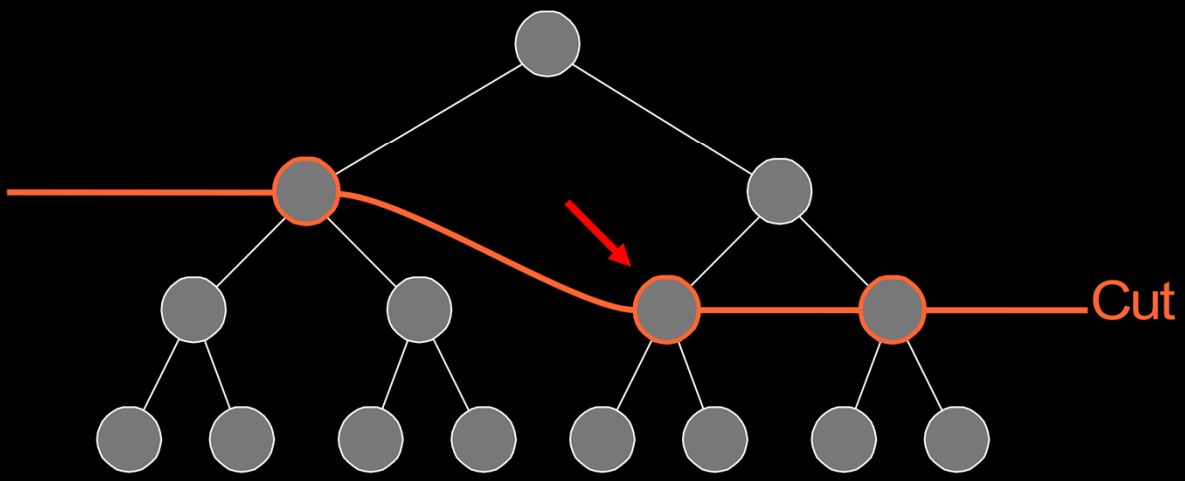


Then we again select the cut node with the largest error bound



And refine if its error bound is above threshold

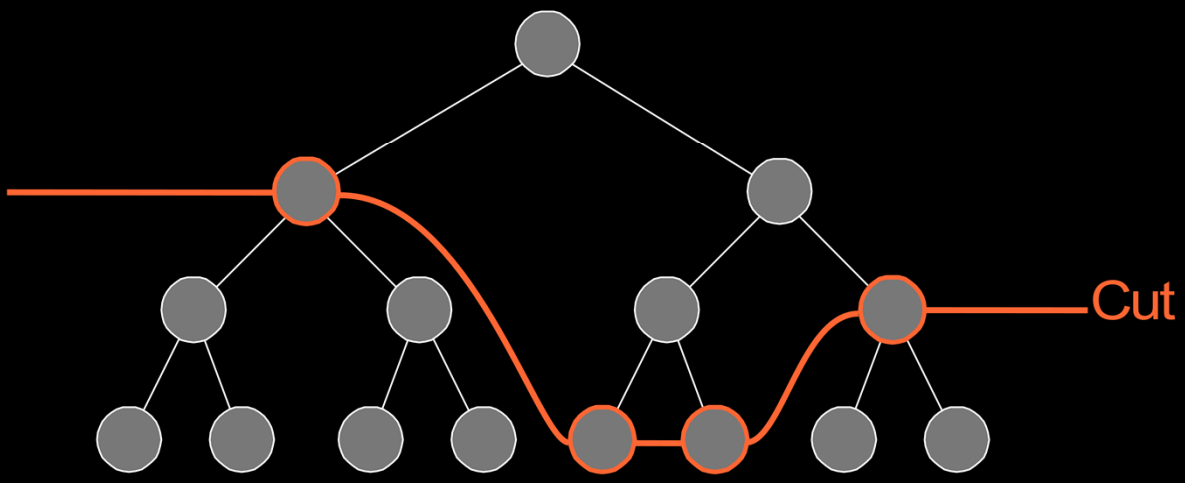
Cut Selection Algorithm



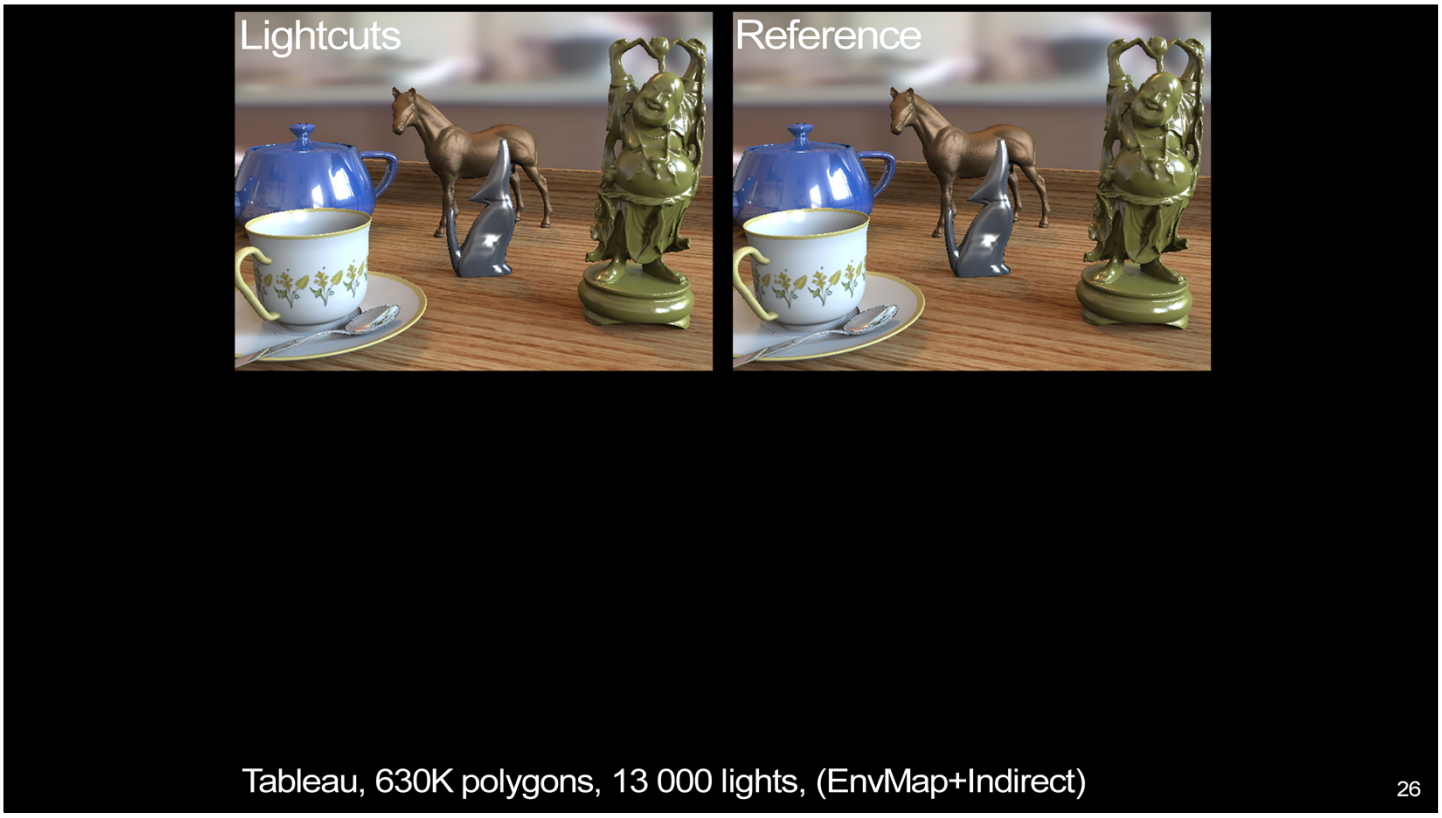
And repeat again

Cut Selection Algorithm

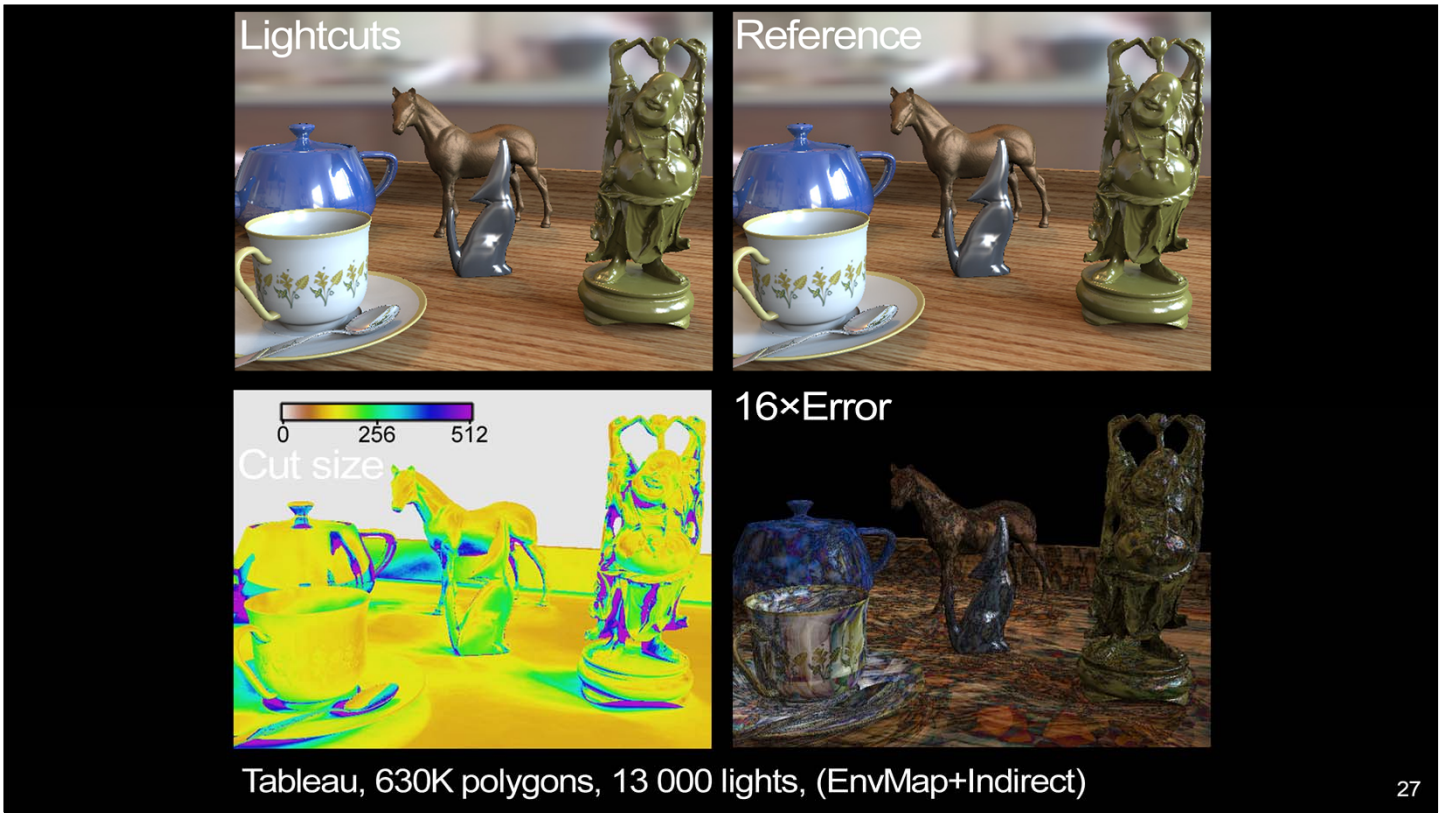
- Repeat until cut obeys 2% threshold



Until each node on the cut obeys our 2% threshold.



Here we show a result rendered with lightcuts and a reference image which looks identical.



A false color image shows how the cutsize varies over the image. Note that it is much smaller than the 13000 lights. We also show an exaggerated error image which shows places where transitions occur and the problems they would cause if we allowed them to large enough to be visible.

Lightcuts Recap

- Unified illumination handling
- Scalable solution for many lights
  - Locally adaptive representation (the cut)
- Analytic cluster error bounds
  - Most important lights always sampled
- Perceptual visibility metric

To recap, lightcuts is a unified framework for handling complex illumination. Its core is a new scalable solution for handling many lights and a locally adaptively illumination approximation called the cut. It is based on analytic cluster error bounds that guarantee that we will always sample the most important lights and a perceptual visibility metric.

Talk Overview

- Scalable solutions for many light rendering, Part I
  - Illumination at a single receiver: lightcuts
    - "Lightcuts: a Scalable Approach to Illumination" by Walter, Fernandez, Arbree, Bala, Donikian, Greenberg, SIGGRAPH2005
  - Illumination over a pixel: Multidimensional Lightcuts
    - "Multidimensional Lightcuts" by Walter, Arbree, Bala, Greenberg, SIGGRAPH2006

Next I will describe the multidimensional extension to lightcuts to efficiently compute the illumination over pixels rather than just at individual points.

Pixels are complex regions

$$\text{Pixel} = \int_{\text{Time}} \int_{\text{Pixel Area}} \int_{\text{Lights}} L(\mathbf{x}, \omega) \ldots$$

Let me first describe the problem we're trying to solve. In order to compute high quality realistic images we need to be able to simulate multiple complex expensive phenomena. For instance the image on the right includes complex illumination, both from a captured environment map and indirect illumination, plus anti-aliasing and, on the right side of the image, motion blur for the spinning roulette wheel. Thus for each pixel we need to integrate the contribution over multiple domains.

Pixels are complex regions

$$\text{Pixel} = \int_{\text{Volume}} \int_{\text{Time}} \int_{\text{Pixel Area}} \int_{\text{Lights}} L(\mathbf{x}, \omega) \ldots$$

If we also want to include participating media then we also need to integrate over the volume of the scene. For example in this smoky kitchen scene which shows shafts of sunlight streaming into the kitchen.

Pixels are complex regions

$$\text{Pixel} = \int_{\text{Aperture}} \int_{\text{Volume}} \int_{\text{Time}} \int_{\text{Pixel Area}} \int_{\text{Lights}} L(\mathbf{x},\omega)\ldots$$

And if we want to add depth of field or camera focus such as shown in this scene then we need to also integrate over the aperture of the camera.

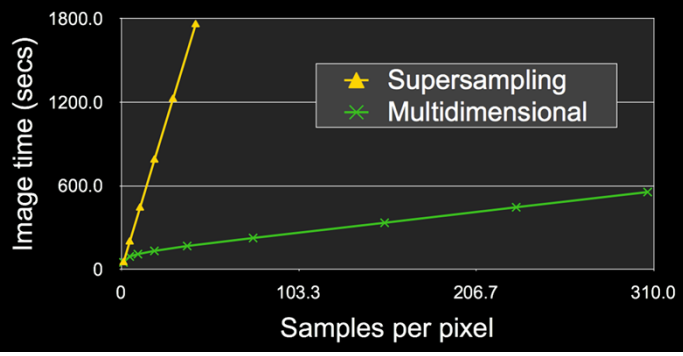
Pixels are complex regions

A new method will be presented on Tuesday that uses the Multidimensional Lightcuts framework to handle a wider range of materials including glossy reflections, subsurface or BSSRDF, and complex volumetric models such as cloth.

Come to the talk on Tuesday to hear more details.

Insight

- Complex integrals over multiple dimensions
  - Requires many eye samples
- Holistic approach
  - Solve the complete pixel integral

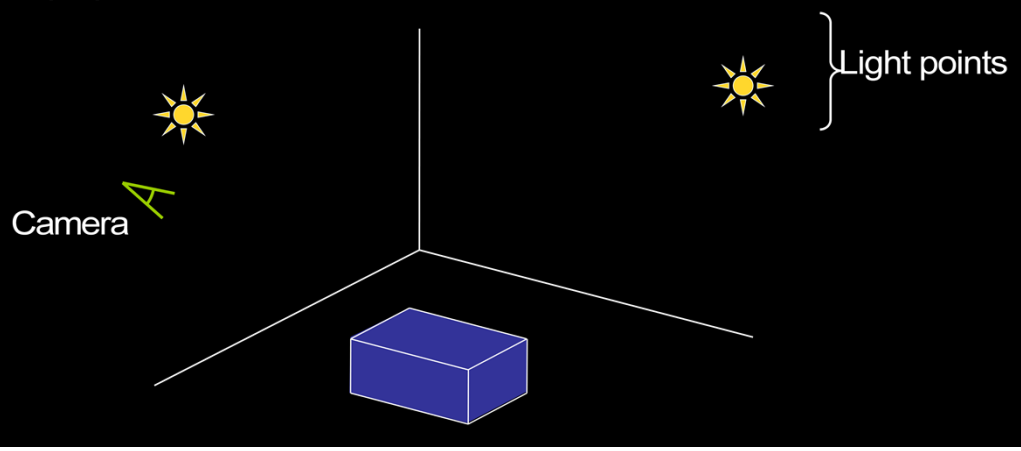


What we want is compute total pixel values for our images which means multiple integrals. While many previous techniques have tried solving each integral separately but this does not result in a scalable solution. One of the key insights here is that solving the complete pixel integral is fundamental to achieving a scalable solution.

This example shows another way in which the system is scalable. We take a single scene and progressively add more and more complex effects. Thus the image on the upper left is a direct only solution for the area lights and sun. The image on the upper right also include indirect illumination. The image on the lower left also added smoke or participating media to the scene and finally the image on the lower right further adds motion blur. Even though this image includes multiple complex effects its rendering time only increases by a factor of 2.2 compared to the direct only solution which is much better than in traditional approaches where adding each effect would greatly increase the image time.

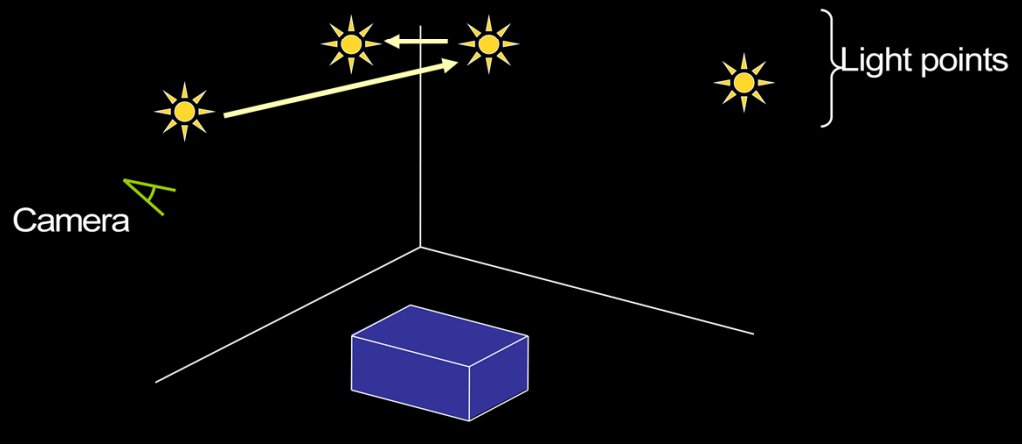
Point Sets

- Discretize full integral into 2 point sets
  - Light points (L)
  - Gather points (G)



So now let's describe how the system works. We discretize the full integral equation using two point sets. First we convert all the light sources in the scene to point light approximations.

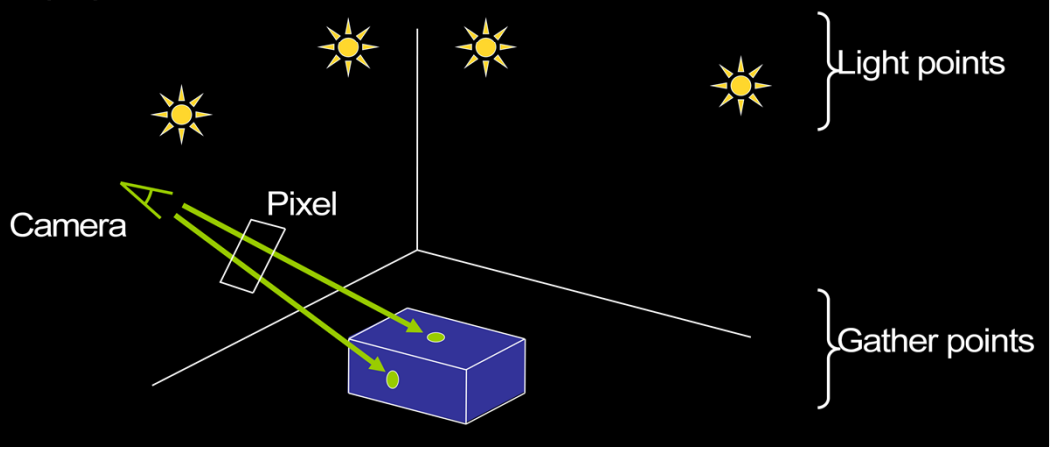
Point Sets



We also trace particles from the light to create additional light points that account for the effect of indirect illumination.

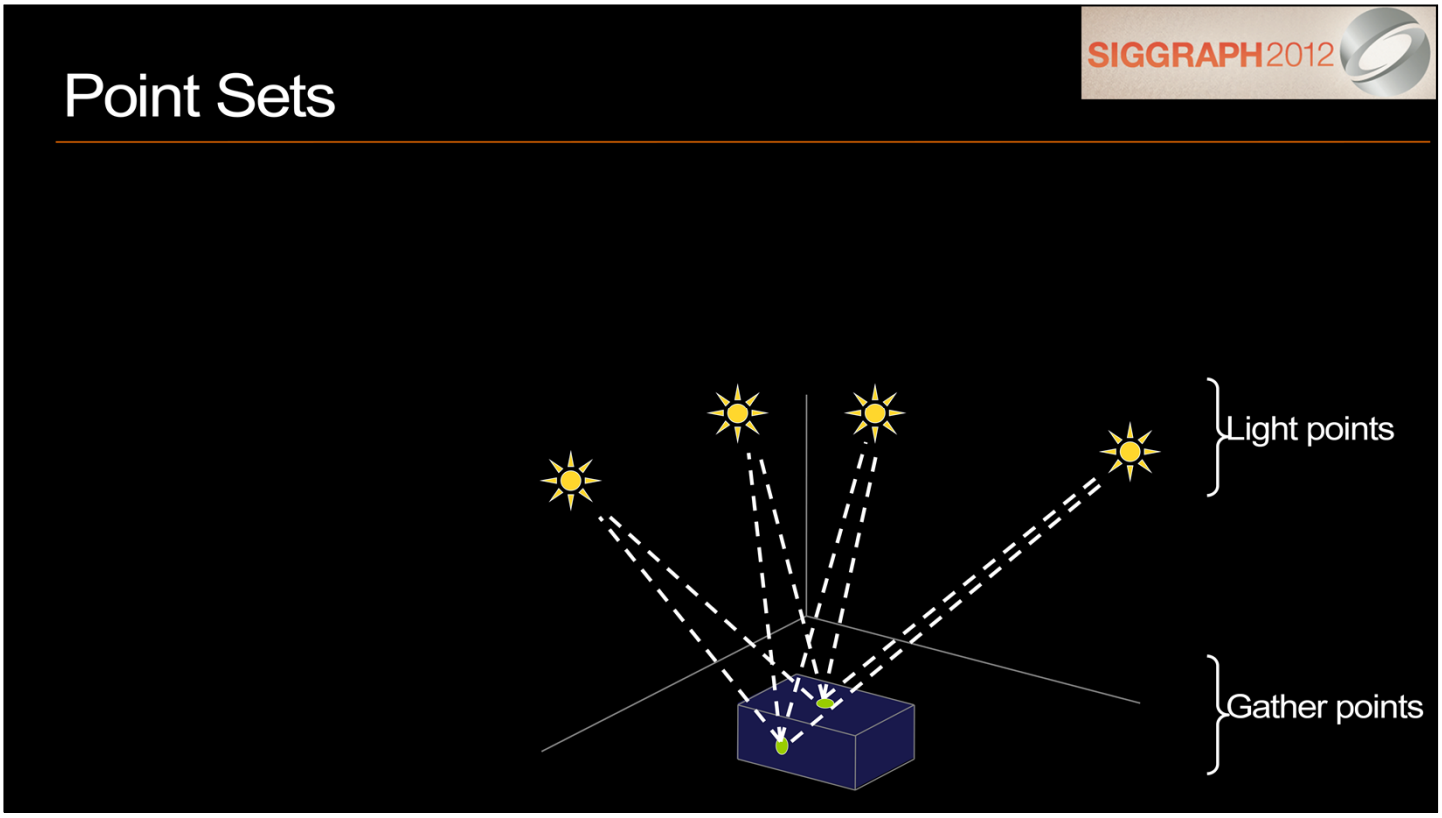
Point Sets

- Discretize full integral into 2 point sets
  - Light points (L)
  - Gather points (G)



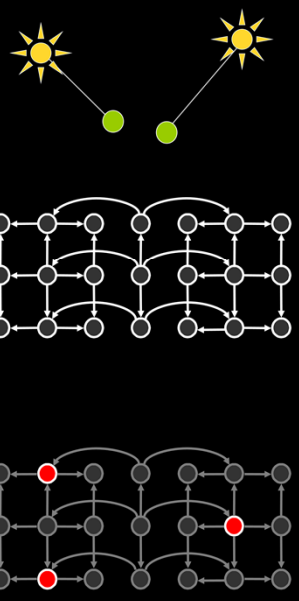
Then for each pixel we trace rays from the camera through the pixel to generate gather points. Note that these point can be generated over time, over the volume, camera aperture, etc. to account for the different integral domains.

Point Sets



The the set of all gather-light pairs forms a very large set of transport paths that can accurate approximate the full integral.

Key Concepts



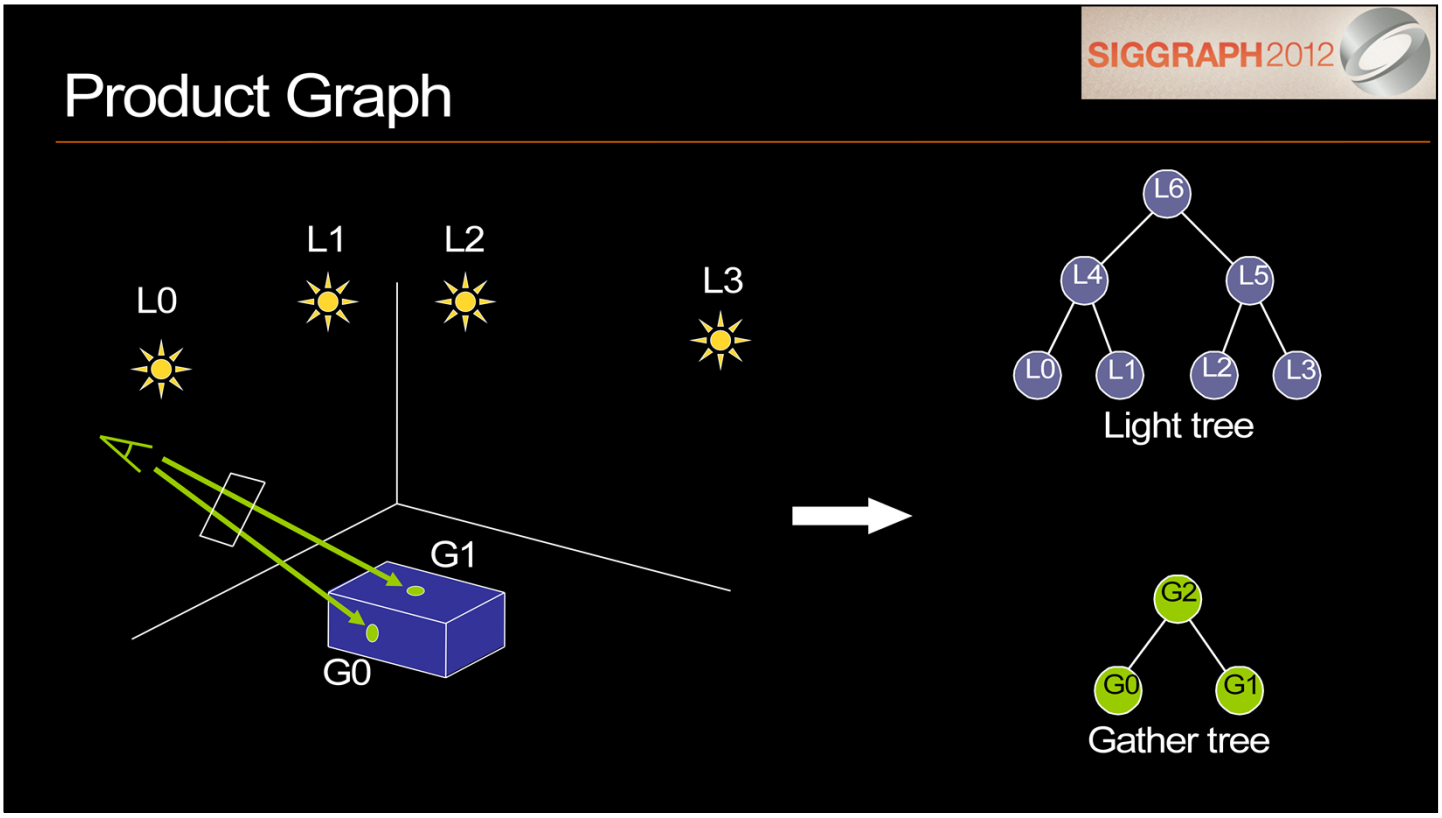
There are several key concepts we use for this scalable solution. First we reduce all the integrals to a unified representation, namely pairs of gather and light points. Then we form a hierarchy of clusters over the set of pairs which we call the product graph. Then we use this hierarchy to dynamically select a appropriate partitioning of the pairs into clusters which we call a cut. To achieve this we need an inexpensive way to approximate each cluster, to bound the error introduced by this approximation and an perceptual metric.

Product Graph

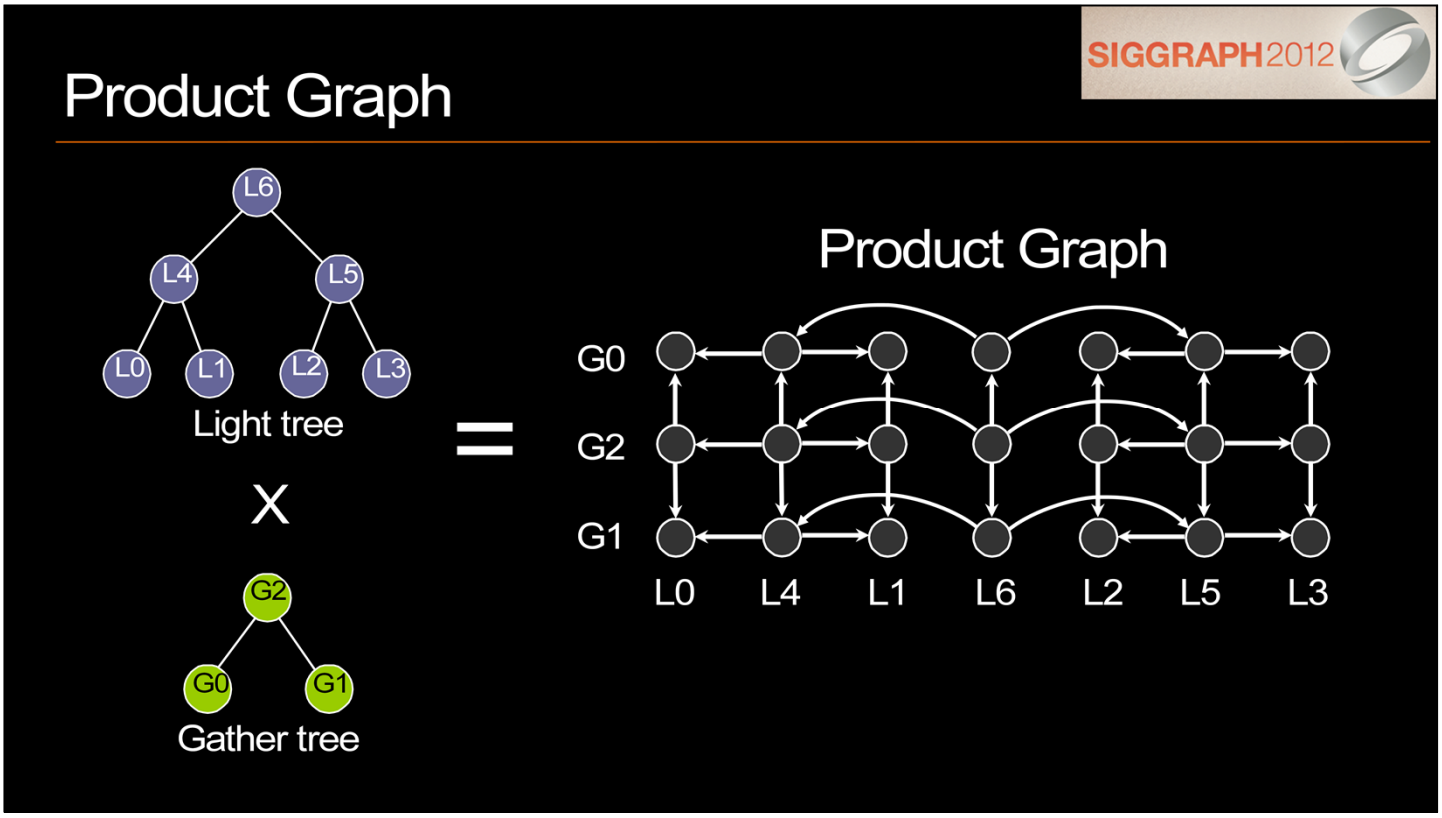
SIGGRAPH2012

- Explicit hierarchy would be too expensive
  - Up to billions of pairs per pixel
- Use implicit hierarchy
  - Cartesian product of two trees (gather & light)

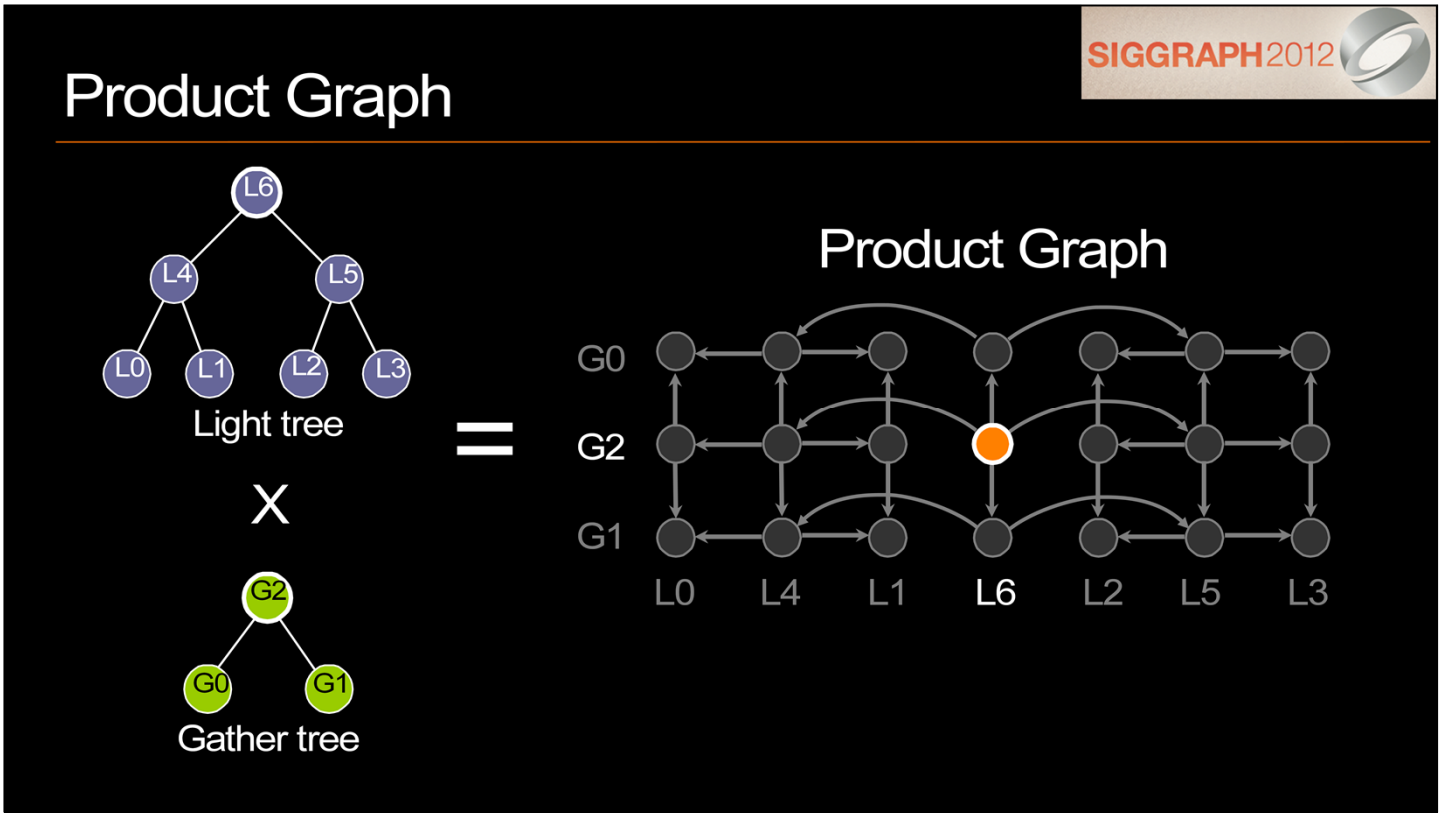
Since we could have up to billions of pairs per pixel, explicitly constructing a hierarchy over them would be too expensive. Instead we are going to create an implicit hierarchy via the cartesian product of two trees, the gather tree and the light tree.



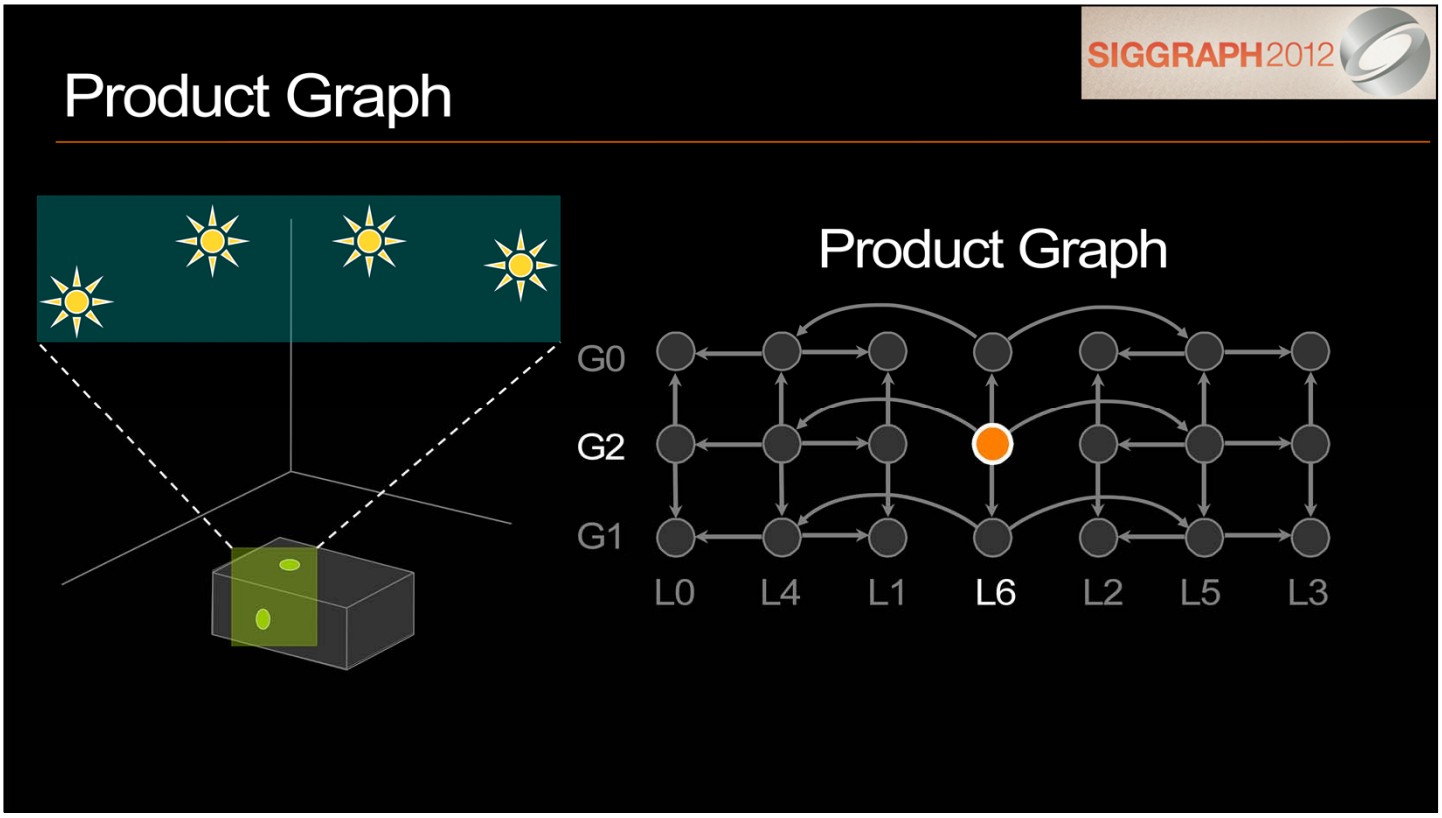
So first we generate all the light points and cluster them together into a light tree. Then for each pixel we generate the gather points and cluster them together into a gather tree.



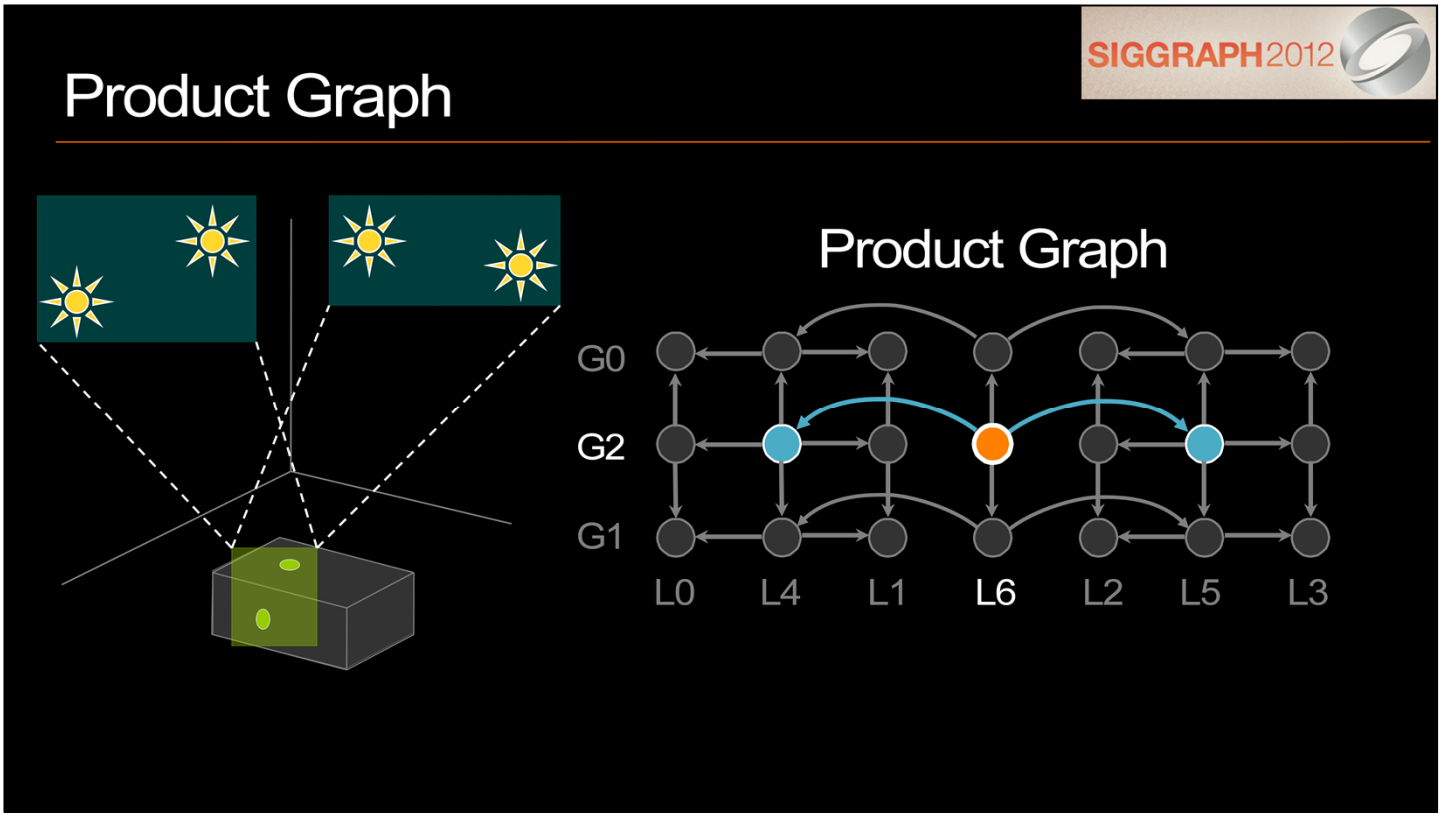
Then we can use these trees to define the product graph which forms a hierarchy over the set of all gather-light pairs. Each node in the product graph corresponds to pairing of one node from the light tree and one node from the gather tree.



For example the source node of the product graph corresponds to the pairing of the root node of the light tree and the root node of the gather tree.

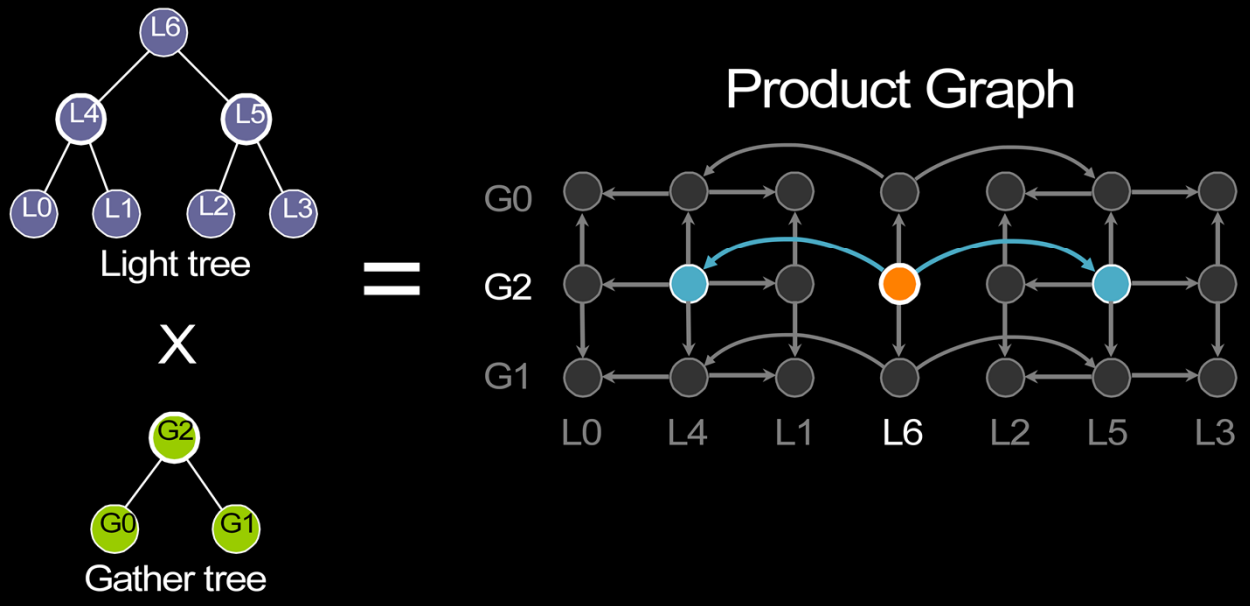


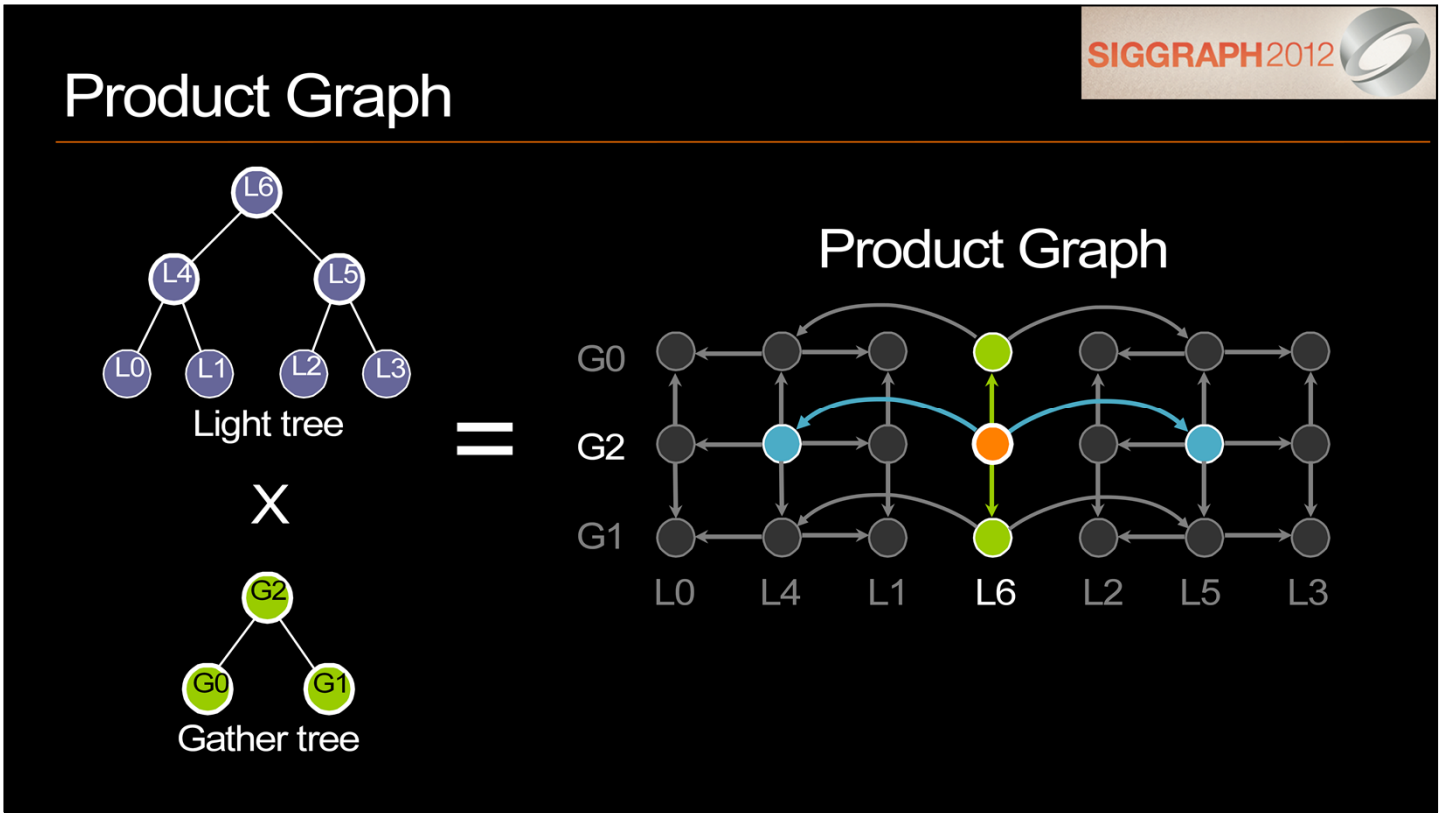
In the scene this corresponds to combining all the light points into a single cluster, combining all the gather points into a single cluster and then treating the set of all pairs as just a single interaction between these clusters. In general that is not going to be an accurate enough approximation of the pixel value, so we need a way to refine this approximation.



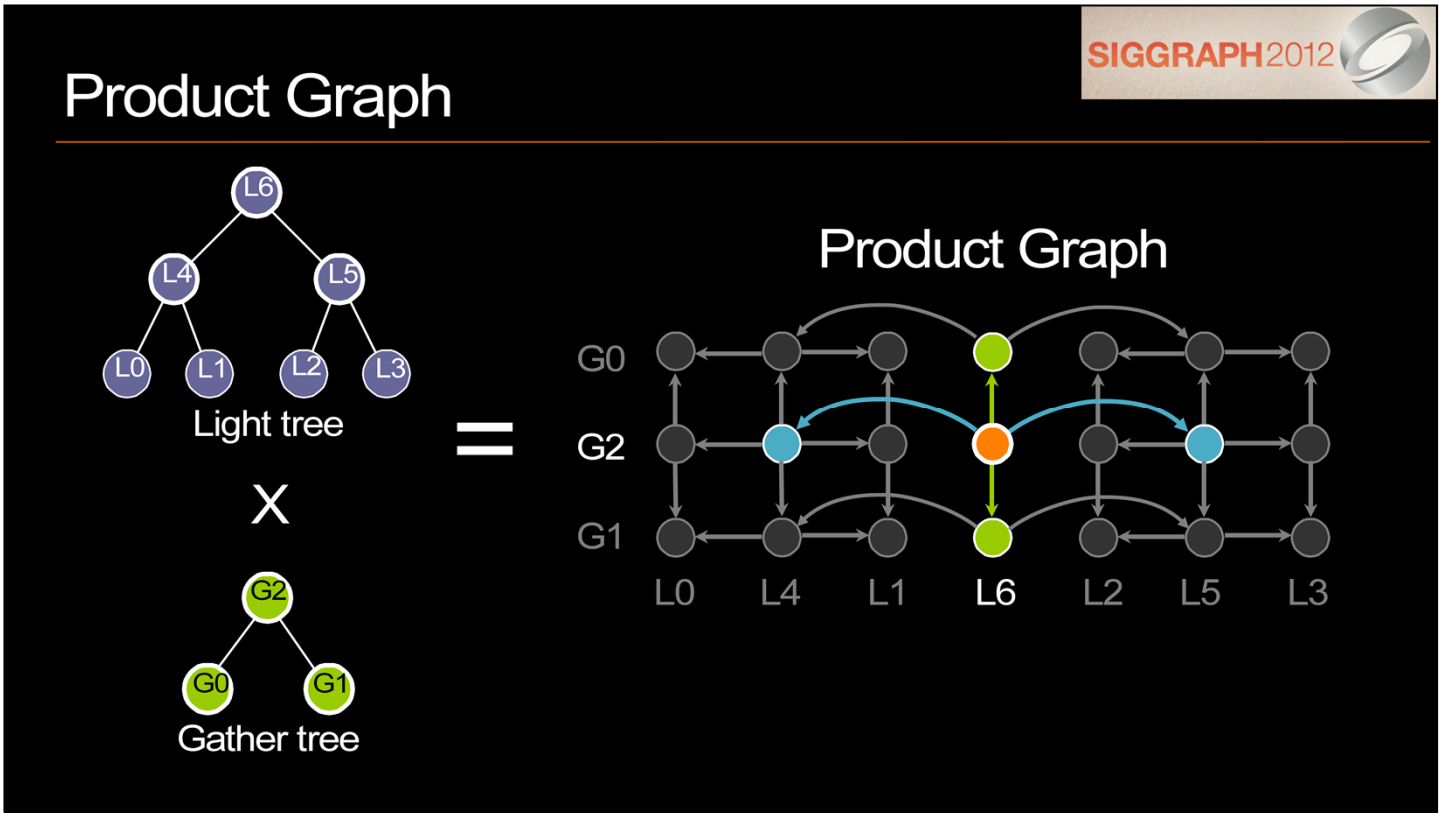
For example we can split the lights into two clusters and then treat the set of all pairs as two interactions. One between the gather points and the light cluster on the left and the other between the gather points and the light cluster on the right. This corresponds to moving along the blue arrows in the product graph. It also corresponds to moving one step down in the light tree.

Product Graph



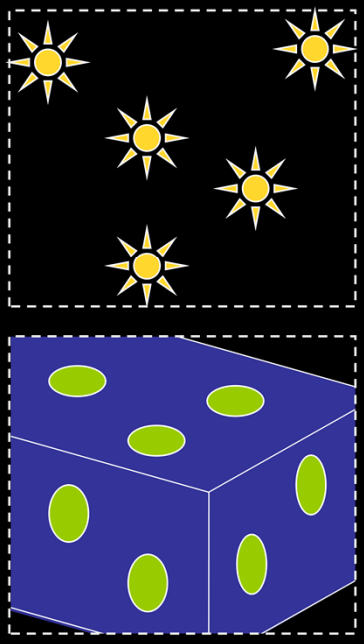


We could also have refine the interaction by moving one step down in the gather tree (ie splitting the gather cluster) which corresponds to the green arrows in the product graph. A key thing to note here is that we do not ever have to actually create the product graph. Instead we can implicitly traverse the product graph by simultaneously walking both the light and gather trees. And thus we never actually instantiate the product graph.



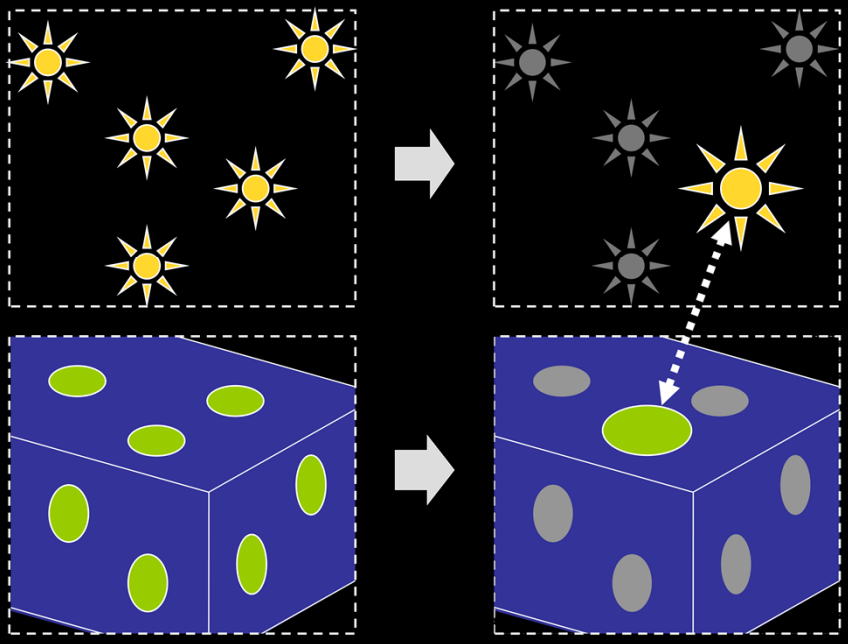
We could also have refine the interaction by moving one step down in the gather tree (ie splitting the gather cluster) which corresponds to the green arrows in the product graph. A key thing to note here is that we do not ever have to actually create the product graph. Instead we can implicitly traverse the product graph by simultaneously walking both the light and gather trees. And thus we never actually instantiate the product graph.

Cluster Representatives



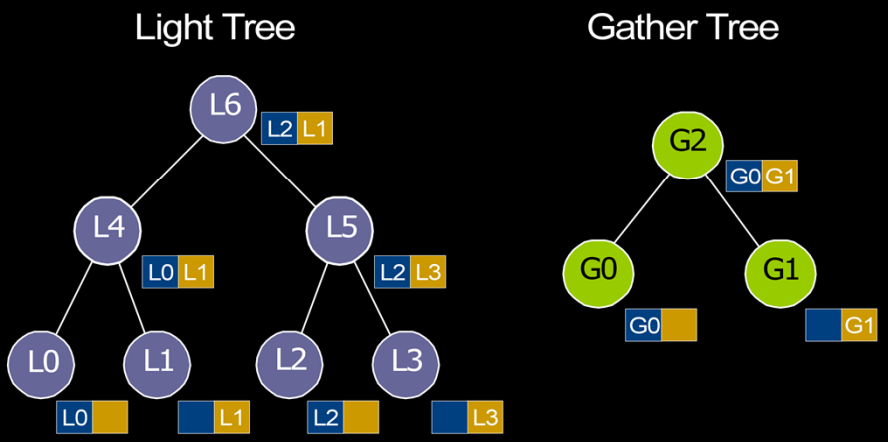
A cluster consists of a set of gather points and a set of lights and represents all possible pairings between the gather and light points. To cheaply approximate a cluster, we select a single pair and evaluate it, scaled by an appriopriate factor based on the probability of picking that pair.

Cluster Representatives



Each time we evaluate a cluster, we select one light as its representative. We pick one of the light to be the representative light and similarly one representative gather point. Then we approximate all the gather-light pairs using just the single pair of the representative gather point and the representative light point.

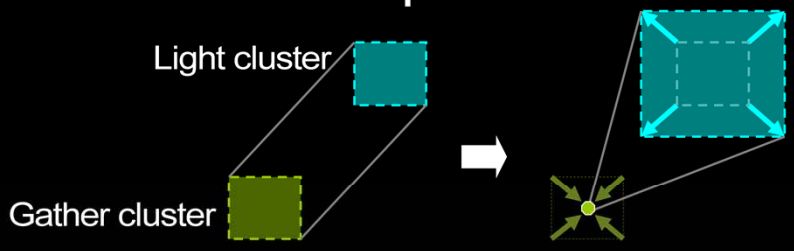
Multiple Representatives



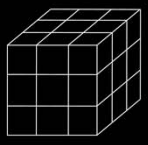
One important change from Lightcuts is that we store multiple representatives per cluster node. This is essential when integrating over time as only nodes at the same instant of time can be connected, but is also very useful even not integrating over time. It helps reduce correlation between pixels and reduce error regardless of which refinement type is chosen.

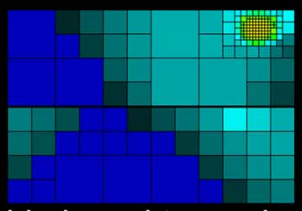
Error Bounds

- Collapse cluster-cluster interactions to point-cluster
  - Use Minkowski sums



- Combine bounds over multiple points
  - Rasterize into cube-maps


Cubemap


Unfolded quad-tree cubemap

We also need to generalize the error bounding to include gather cluster to light clusters rather than just the point to cluster bounds in the original Lightcuts. However we can collapse cluster-cluster interactions back into point-cluster interactions using Minkowski sums where we shrink one cluster to a point while simultaneously expanding the other cluster. We also need to combine the bounds from all points in a cluster into a common representation. We do this by rasterizing them into cubemaps that cover the sphere of directions. In the paper we used fixed resolution cubemaps but we have since switched to adaptive cubemaps (quad-tree on each face) because these better handle glossy BRDFs and

other strongly directional distributions.

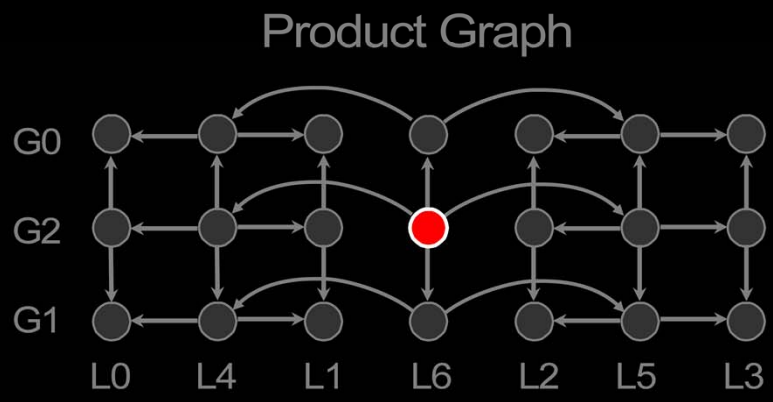
Algorithm Summary

- Once per image
  - Create lights and light tree
- For each pixel
  - Create gather points and gather tree for pixel
  - Adaptively refine clusters in product graph until all cluster errors < perceptual metric

So now let us summarize our algorithm. Once per image, we generate the set of light points and cluster them into a light tree. Then for each pixel we generate gather points and cluster them into a gather tree. Then we start with a coarse cut in the product graph and adaptively refine it until the each node's error is less than our perceptual metric.

Algorithm Summary

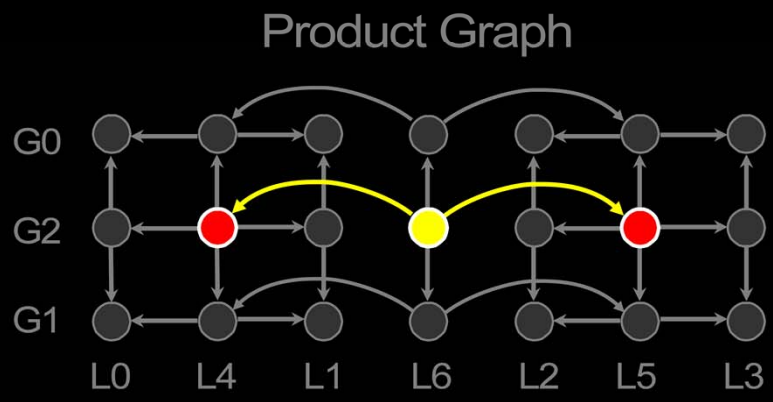
- Start with a coarse cut
  - Eg, source node of product graph



For example we can start with the coarsest cut which is simply the source node of the product graph.

Algorithm Summary

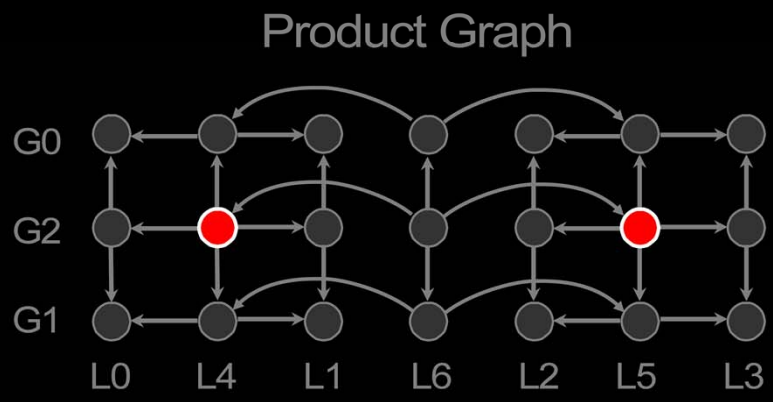
- Choose node with largest error bound & refine
  - In gather or light tree



Then at each step we choose the node with the largest error bound and refine it by moving one step down in either the gather tree or the light tree. In this example we've chosen to move one step down in the light tree.

Algorithm Summary

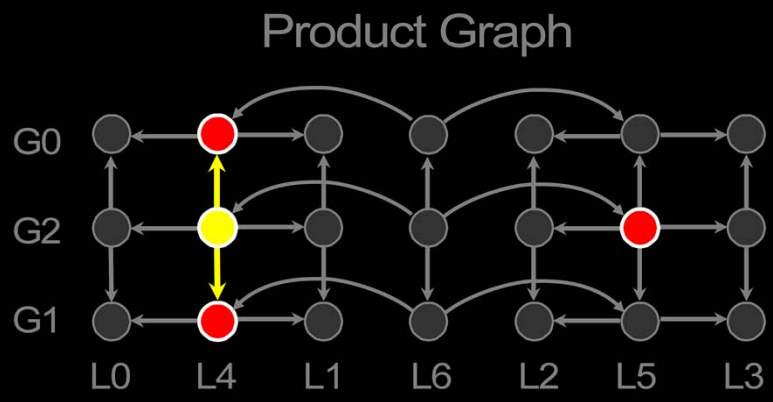
- Choose node with largest error bound & refine
  - In gather or light tree



Then we repeat the process.

Algorithm Summary

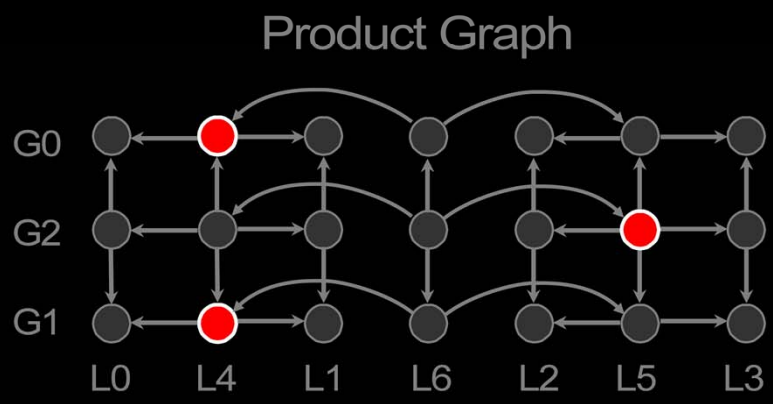
- Repeat process



Select the node with the largest error bound and refine it, in this case we chosen to refine by moving one step down in the gather tree.

Algorithm Summary

- Until all clusters errors < perceptual metric
  - 2% of pixel value (Weber's law)



We keep refining until all node in the cut have error bounds less than our perceptual metric, which is 2% of the estimated pixel value and is based on Weber's law.

Roulette

7,047,430 Pairs per pixel Time 590 secs
Avg cut size 174 (0.002%)

This is an example with a spinning roulette wheel which has strong motion blur. Here we are using 256 temporal samples per pixel for an average of over 7 million potential pairs per pixel. However the cut size is only 174 which means that we only actually evaluated 174 pairs which is only a tiny fraction of all the pairs.

Conclusions

- Unified handling of complex effects
  - Motion blur, participating media, depth of field etc.
- Product graph
  - Implicit hierarchy over billions of pairs
- Scalable scene and illumination complexity
  - Millions to billions of point pairs per pixel

Many lights rendering unifies the handling of complex illumination and the multidimensional lightcuts approach similarly unifies the handling of many effects including motion blur, participating media, and depth of field. In unifies the representation of all effects as pairs of gather and light points and uses an implicit hierarchy over the pairs called the product graph. Together with error bounds and a perceptual metric this algorithm is both both scalable and accurate.

The End

That concludes my talk and I'd be happy to answer any questions.

Schedule

- 2:00 (05 min) … Introduction & Welcome (Křivánek)
- 2:05 (30 min) … Instant Radiosity (Keller)
- 2:35 (30 min) … Handling difficult light paths (Hašan, Křivánek)
- 3:05 (25 min) … Scalability with many lights I (Walter)

- 3:30 (15 min) … Break

- 3:45 (20 min) … Scalability with many lights II (Hašan)
- 4:05 (35 min) … Real-time many-light rendering (Dachsbacher)
- 4:40 (30 min) … ML in Autodesk® 360 Rendering (Arbree)
- 5:10 (05 min) … Conclusion - Q & A (All)