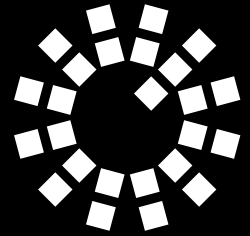


SONY PICTURES

imageworks

25TH ANNIVERSARY

IMPORTANCE SAMPLING OF MANY LIGHTS WITH ADAPTIVE TREE SPLITTING

Alejandro Conty & Christopher Kulla

SIGGRAPH 2017

THE PROBLEM WE TRY TO SOLVE

- Scenes with many emitters
  - In the form of many light sources (cities, buildings)
  - Or as mesh lights with many triangles
- What we want to avoid:
  - A traditional pathtracer would linearly scan all lights at every intersection
  - Naive random picking is too noisy

Scenes with many lights. It's becoming a very common problem in production. Hundreds, even thousands of lights in the form of cityscapes or big interiors with structural lighting. The traditional raytracing approach of sequential light loops is too slow to deal with this cases.

- Cityscapes, structural lighting
- Sequential light loop not suitable

A STOP AT TERMINOLOGY BEFORE WE BEGIN ...

Local lights

Lights are considered *local* in a scene if there is a small upper bound to the number of lights affecting any given point. (**eg.** a cityscape at night, a christmas tree in daylight) No aggregated lighting: **most lights affect a small part of the scene**.

Global lights

Lights are considered *global* in a scene if a large number, maybe thousands, may be required to shade a single point. (**eg.** many lamps on the ceiling of a room, a christmas tree in a dark room) Aggregated lighting: **many dim lights add up to throw significant light** somewhere.

Just to clarify, there's a difference between local and global lights. Let's think of christmas tree. In daylight, its little lights behave as local because they are only relevant to the nearby geometry, but in a dark room their small contributions add up to significant light for the whole scene. So their behavior is global. And the latter is the real problem we're solving here.

- We consider two types of lights
- A christmas tree in daylight is local
- A christmas tree in a dark room is global

MOTIVATION

- Classical unbiased methods only efficient for **local lights**
  - They deterministically return *all lights* needed for a shading point
  - It requires the artist to keep the light scope short
- Avoid any linear scan of the lights at render time
- Exploit orientation of emitters
- Usable for **mesh-lights** with many triangles

The mesh-light problem

Without any aid, mesh-lights suffer from poor sampling. Too often a distant or pointing away triangle is chosen. Not to mention sample stratification.

Local lights are an already solved. But what happens with a mesh light? All its tiny triangles add up just like global lights. And they suffer from poor sampling, because you don't want to sample distant or facing away triangles as much as the nearby ones.

- Local lights not a problem
- Mesh lights sum up all their triangles like global lights
- Poor sampling, distance and orientation have to be accounted

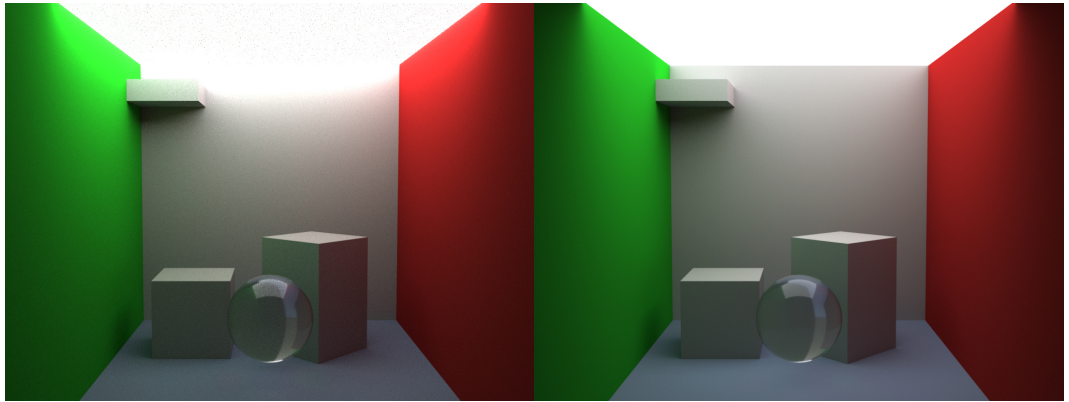
LOCAL VS GLOBAL LIGHTS

- **Local lights** affect a *small part* of the scene
  - Any given shading point is only affected by a few lights (1, 3, 8 …)
  - All of them can be shaded
  - An acceleration structure that returns those lights is good enough and clean (**no random sampling needed**)
- **Global lights** can potentially affect *the whole scene*
  - Hundreds or thousands might be needed for a single shading point
  - We cannot afford shading all, a **stochastic method is necessary**
  - Classic acceleration structure not good enough!

So with meshes or many global lights affecting almost every shading point, if we don't stochastically sample a small subset we have a serious performance issue. With local lights this was not a problem. A simple hierarchy can quickly give you the handful of lights you need for the shading point.

- **Many global lights need stochastic sampling**
- **Local lights can be retrieved deterministically**

GLOBAL LIGHTS



This is a benchmark comparing a single area light on the right with a million tiny point lights on the left. Any of those little lights on the left has a negligible power, but all together they yield a similar result. We couldn't render the left picture with a local light approach because we would still need to shade all the million lights for every point. Our stochastic method renders it in a very reasonable time, only twice as slow as the single area light case.

- Million light benchmark
- A local light acceleration would take forever

METHOD OVERVIEW

- Build a BVH of all the emitters in the scene
- Define an *importance measure* for a light cluster and a point
- With a random number perform a traversal of the tree
- At each node we make a decision between left and right
  - Importance sets the left probability $P_{\mathbf{L}} = I_{\mathbf{L}} / (I_{\mathbf{L}} + I_{\mathbf{R}})$
  - Recursively descend and *stretch* the random number
  - At the leaves we will find the chosen light

Summarizing our method, we build a BVH of the emitter geometry similar to raytracing accelleration where each node is a cluster of lights. Then we define an importance measure so we can traverse the cluster tree making binary decisions at every level using a random number. We start with the root until we reach the leaves. And there we find our sampled light.

- BVH like in raytracing with clusters of lights
- Importance measure for binary decisions
- Top-down traversal

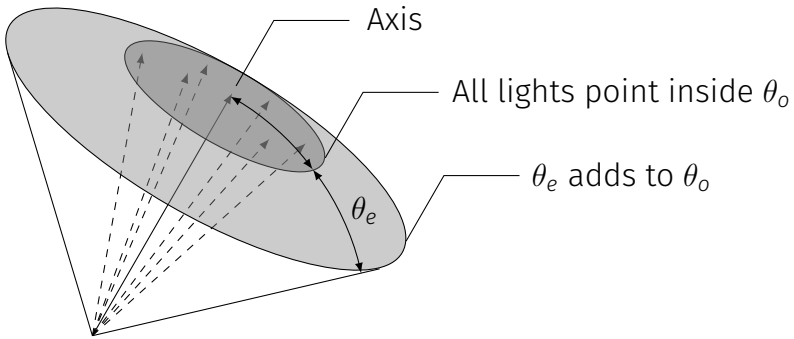
BOUNDING VOLUME HIERARCHY

- Clusters bounded in **3D world** space and **orientation**
- 3D bounds: standard bounding box
- Orientation bounds, one vector and two angles:
  - Axis: central emitter normal
  - Orientation cone $\boldsymbol{\theta_o}$: to cover all the normals in the cluster
  - Emission cone $\boldsymbol{\theta_e}$: maximum of the mission profiles of the emitters
  - Struct: `Vec3 axis; float theta_o, theta_e;`

The key of our tree is that it also takes into account orientation bounds in addition to the usual 3D box. This means that we need a vector and a couple of bounding angles.

- **The key is adding orientation bounds**
- **A vector and two bounding angles in addition to the box**

ORIENTATION BOUNDS



As you can see here, the first bounding angle theta o around the axis, encloses the orientation vectors of all the emitters in the cluster. The second, theta e, expands the bounds with the emission range. This range depends on the light type. Half pi for lambertian emitters, something smaller for spot lights. And in a cluster we take the maximum of all them.

- An angle around a central axis to enclose orientation vectors
- Another to extend the bounds with the emission profile
- Emission depends on light type

ORIENTATION BOUNDS EXAMPLES

- Sphere light
  - **Axis**: irrelevant (any vector)
  - $\boldsymbol{\theta_o} = \pi$
  - $\boldsymbol{\theta_e} = \pi/2$
- Quad light or triangle emitter
  - **Axis**: quad or triangle normal
  - $\boldsymbol{\theta_o} = 0$
  - $\boldsymbol{\theta_e} = \pi/2$
- Spot light
  - **Axis**: spot direction
  - $\boldsymbol{\theta_o} = 0$
  - $\boldsymbol{\theta_e} =$ spot's apperture

For example, a sphere light emits in every direction so the axis is irrelevant and the bounds cover the whole space of directions. A quad light emits in a single direction, its normal. And even though it has a zero angle orientation cone, the emission covers the whole hemisphere. And the spot light is very similar except the emission is limited by its aperture.

- Three examples: sphere, quad, spot

TREE NODE REQUIREMENTS

Total node size = 56 bytes

```
struct Node
{
    float          energy;    // Total energy under this node
    int            nemitters; // Number of emitters under this node
    int            offset;    // >= 0 left child, otherwise emmiter offset
    struct {                  // Orientation bounds
        Vec3  axis;
        float theta_o;
        float theta_e;
    } bounds_o;

    struct {                 // World space bounds
        Vec3 min, max;
    } bounds_w;
};
^^|
```

And just for reference, here is how we internally represent a node of our tree. We fit it in only 56 bytes, making the tree lightweight so a million lights would only take 100 megabytes. We have never needed that much, but we wanted to be prepared for the eventuality. I've already shown that kind of benchmark.

- Code reference
- Fit a million lights in 100mb
- Still haven't reached that number

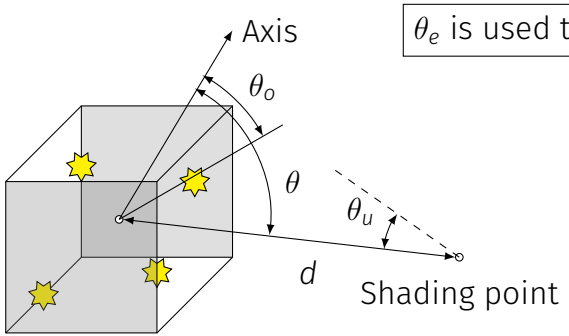
CLUSTER IMPORTANCE MEASURE

- Contained **energy**
- Inverse square **distance** from 3D bounds center
- Conservative **cosine factor** to the orientation bounds
  - Angle to axis minus orientation cone
  - Compute cosine of the angle difference
  - Clamp by the emission angle (0 outside)
- The size of the 3D bounds implies an uncertainty angle $\boldsymbol{\theta_u}$ that we take into account for the conservative cosine by adding it to the orientation cone

Remember we need to traverse our tree making random decisions between branches. For this purpose we define an importance measure. We take into consideration energy, distance and a cosine factor from the orientation bounds. You just have to be careful because the positions of the emitters are unknown inside the box, so there is an intrinsic error we have to account for and we call it the uncertainty angle.

- Remember tree traversal and random decisions
- Measure importance with energy, distance and cosine
- Intrinsic error error, uncertainty angle

IMPORTANCE FROM A SHADING POINT



θ_e is used to clip $\theta - \theta_o - \theta_u$

$$I = E \cdot \cos(\mathrm{clamp}(\theta - \theta_o - \theta_u, 0, \theta_e))/d^2$$

This would be a cluster of lights and a shading point. We define its importance as the product of the energy times the cosine with the orientation cone over the square distance to the center. This distance might be off for most emitters if the cluster is big, so we lose accuracy as we get closer. I will not get into too much detail about the uncertainty angle because of lack of time, but feel free to ask us later.

- Importance as a product of energy, cosine and inverse square distance
- Distance from the center will be off from reality, accuracy loss
- Ask later for details about uncertainty angle

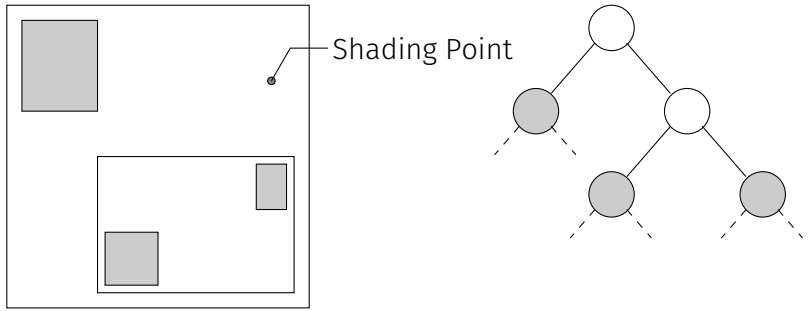
SAMPLING THE TREE

- Option A: draw a single sample
  - Based on importance take a sample from left or right branch
  - Process is applied recursively
- Option B: split!
  - We take a sample from the left branch ***and*** the right branch
  - Process is applied recursively until option A is chosen
- Split is applied using an expected variance heuristic

With our importance measure, we can now sample lights from the tree. If we just want to select a light, we do as I mentioned earlier: recursively descend the tree making binary decisions for left or right branches. That's option A. But if the cluster is too big or too close we may want to shade more than one light. So instead of choosing left or right, we choose both and get at least two lights. The process is then spplied again to both branches until option A is chosen.

- Ready to go for random decision between left or right
- If too close we split, sample both left and right
- Get at least two lights, repeat until option A is better

SPLIT TRAVERSAL



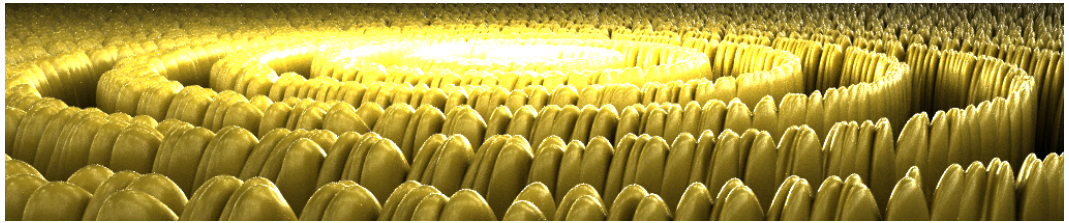
We draw three samples from the gray nodes, the other two upper in the tree are split according to our heuristic

Let's see this with an example. On the left we see the cluster hierarchy in the space and on the right the associated logical tree. The root's box is so big it contains the shading point, so it is split. And from its two children, the closer one on the bottom is still selected for splitting, so we end up grabbing three lights from the three gray nodes. These are randomly chosen from whatever number of lights they might contain.

- Clusters in space and the logical hierarchy
- Root and bottom subbranch are split
- Three lights chosen randomly from those three subtrees

SPLIT EFFECT

This scene has **10k point lights** in a spiral over the stones

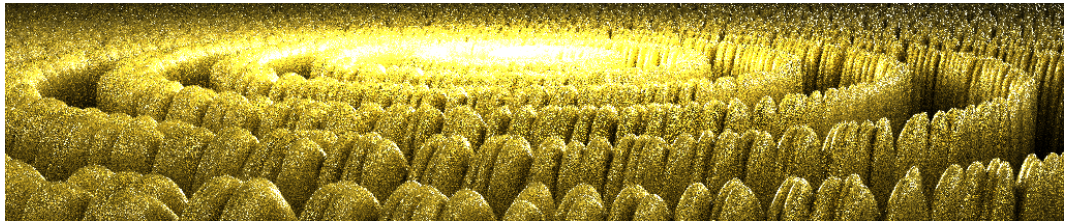


Split On: 4 minutes. Adaptive splitting shades more than one light per shading point.

This is what a render with our split policy looks like. Ten thousand lights rendered in 4 minutes. Only the closer clusters are split to shade more than one light.

SPLIT EFFECT

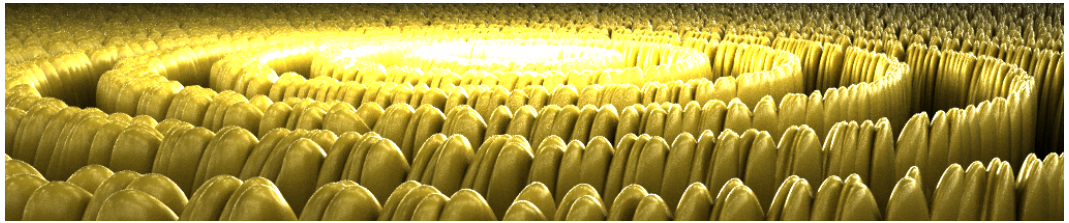
This scene has **10k point lights** in a spiral over the stones



Split off: 43 seconds. Only one light is shaded per shading point at every hit. It is much faster but also very noisy.

Without splitting the result is very noisy. And of course much faster since a single light is shaded for every subpixel. But what if we just increase the number of samples to compensate?

This scene has **10k point lights** in a spiral over the stones



Split off, 64 samples: 17 minutes, still noisier. We can try to compensate for the lack of splitting by going up to 64 samples (lights shaded). But it's more expensive and still doesn't get to the split quality.

Here is the result. We have to go as high as 64 to get a similar result. It is four times slower than splitting and still a bit noisier. More importantly, with the split system the artist doesn't have to bother about sample counts to get a clean render.

SPLIT HEURISTIC

- Approximate **solid angle** of the cluster times **BSDF peak**
- BSDF peak is computed as follows:
  1. Sample a random direction from the BSDF
  2. Compute a conservative maximum cosine with the vector to the cluster's center
  3. Evaluate a simple GGX model with the BSDF roughness
- User sets a threshold for the split mechanism to fire
- Easy to normalize the range
  - 0.0 never splits, only one light will be shaded per point
  - 1.0 always splits, all lights will be shaded

To make the split decision on a cluster we take into account its subtended solid angle from the shading point. Also the BSDF peak in case it is highly specular pointing to the cluster. All together it gives us a number, easy to normalize from zero to one. The user can then set a threshold to control how aggressive splitting will be. In practice we have a default threshold that nobody ever touches. It works fine for most scenes.

- **Decision based on subtended solid angle and peak BSDF**
- **All comes down to a number we normalize between 0 and 1**
- **Control splitting with a threshold**
- **Nobody changes the default**

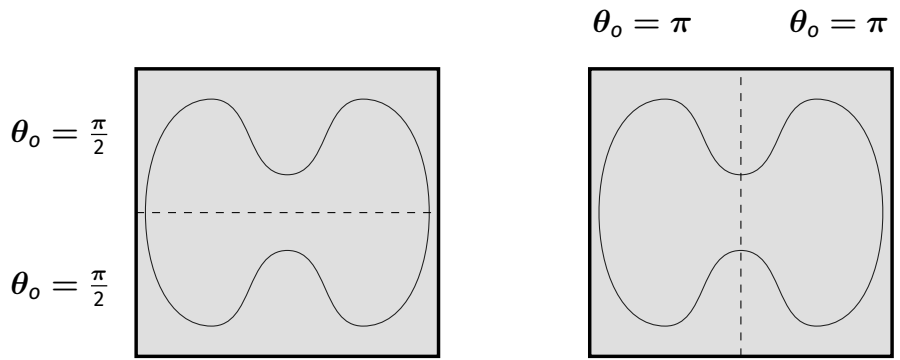
TREE CONSTRUCTION

- Top down BVH process that splits boxes based on a heuristic
- For raytracing we minimize the cost of traversing a branch times the probability of hitting it (SAH)
- Here we minimize the probability of sampling a branch
- **We always split in world space**, but we also use the orientation bounds to estimate the quality of a split
- **Surface Area Orientation Heuristic** (SAOH)

The tree construction is very similar to any raytracing BVH, except we added orientation into the mix for the heuristic. We breake up our clusters in 3D space axes, but we also look at the orientation bounding code of the resulting clusters to evaluate the quality of the split. We named this the surface area orientation heuristic.

- Same as raytracing BVH but with orientation
- Break boxes in 3D space, but looking at orientation for quality
- We call it SAOH

IDEAL SPLIT



We prefer the left split as it also reduces the bounding cone for the orientation of the emitters

Like in this is example, where a typical SAH wouldn't care for either of those splits, our modified one chooses the left one because it also reduces the orientation cones for the two branches from pi to half pi.

- SAH would not tell difference between these splits
- Our SAOH prefers the left one as it reduces orientation bounds

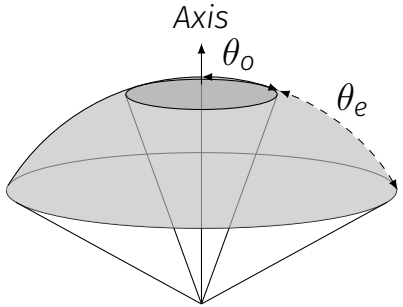
SURFACE AREA ORIENTATION HEURISTIC

- Extended the measured space from box surface to ***surface*** $\times$ Ω
- Product of area M_A and effective solid angle M_Ω
- M_Ω takes into account how we cosine-weight the importance of a cluster's orientation
- $M_A \cdot M_\Omega$ is a hint of the probability of sampling a cluster
- Unlike ray-tracing's SAH, we then balance the cluster's energy instead of the number of primitives

This modified heuristic just extends the box surface space with the solid angle space of the orientation bounds adding two more dimensions. It is just a product of two measures that gives us a hint of the probability of the cluster affecting some point in the scene.

- Go from 2D surface space to 4D surface + cone measure space
- A product of two measures to get a hint for probability

BOUNDING CONE MEASURE



$$M_\Omega = 2\pi \left[(1 - \cos\theta_o) + \int_{\theta_o}^{\theta_o+\theta_e} \cos(\omega - \theta_o)\sin(\omega) d\omega \right]$$

Cone of θ_o solid angle + cosine weighted θ_e sector

I won't go into too much detail of our orientation bounds measure. Suffice to say we add the solid angle of the orientation disc to the cosine weighted measure of the emission portion. This portion is important because otherwise we might think reducing a small disc to half is very good, when in fact it doesn't make a difference if the emission angle is big.

- No time for details
- Add solid angle of orientation disc to cosine weighted emission
- Without emission a small disc reduced to half would look good

SPLIT COST

Probability of sampling a cluster $P(C)$

$$P(C) \simeq M_A(C) \cdot M_\Omega(C) \cdot Energy(C)$$

- Quality of a split is inverse to $cost = \frac{P(C_{left}) + P(C_{right})}{P(C)}$
- The three world space axes are scanned for best split
- We also add a regularization factor to penalize thin boxes
- Balancing this measure also has good effects on *sample stratification*

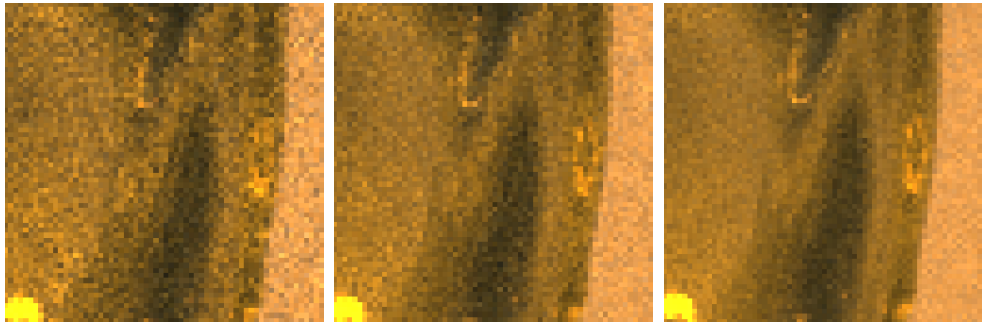
All together results in a cost function for splitting a cluster of lights that we follow from top to bottom. A regularization factor we added to penalize thin boxes also helped us with sample stratification.

- We end up with a cost function for cluster division
- We build from top to bottom
- A regularization factor avoids thin boxes, good for stratification

RESULTS

I’m going to show you the results with this simple scene where we have a spiral meshlight. Its triangles are placed into our tree just the same as if they were individual lights.

RESULTS



The left image just assigns a static probability to all the triangles based on their emission power. All shading points are lit using the same distribution. In the middle we have our method using only the distance part of the sampling heuristic, which is already a big improvement. And on the right we see our method using both distance and orientation for the heuristic, which is even better. The orientation factor avoids sampling pointing-away or almost perpendicular triangles.

EXTENDING TO VOLUME RENDER

- Compute closest point in the line integral to the cluster's center
- Apply simple inverse decay instead of squared
- Orientation: get the point that maximizes dot product with axis

Compute

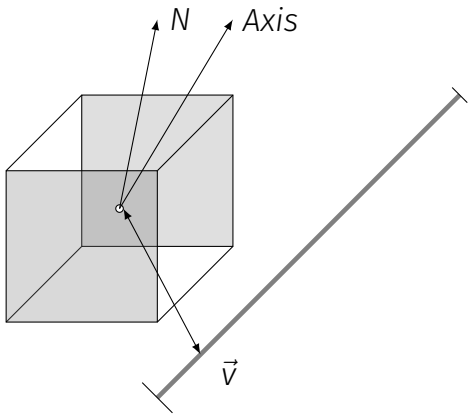
$$\arg\max_{\vec{v}} \vec{v} \cdot \vec{a}$$

where $\vec{v}$ runs on the (ray direction) – (cluster's center) plane and $\vec{a}$ is the cluster's axis. It is done by simple derivative cancellation of a parametrict $\vec{v}$ vector.

Using the tree for volume lighting is also possible. The way we do it is evaluate the heuristic using the closest point from the volume segment to the cluster's center and similarly finding the position that maximizes the cosine with the orientation axis. This is purely a maximization problem that can be solved by derivative cancellation.

- A volume segment instead of a point
- Just find the closest point and the one that maximizes cosine
- Simple maximization problem solved by derivative cancellation

IMPORTANCE FROM A VOLUME SEGMENT



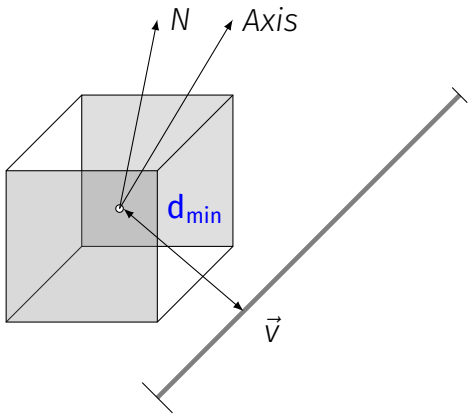
A conservative approach

The vector $\vec{v}$ runs along the volume line integral and on the plane with normal *N*. The stop that minimizes the distance d_{min} might not coincide with the minimum angle θ_{min}. But we use both values to compute a conservative importance measure.

Here we see visually how, given a volume segment, we find the minimum distance and angle to compute the heuristic. Note that the point that minimizes one might not be the same that minimizes the other, but for simplicity we find those independently and assume they happen at the same location. This still produces a conservative heuriustic that guarantees we don't underestimate a light cluster.

- The two points might to coincide
- Compute independently and assume they coincide
- The result is still conservative

IMPORTANCE FROM A VOLUME SEGMENT



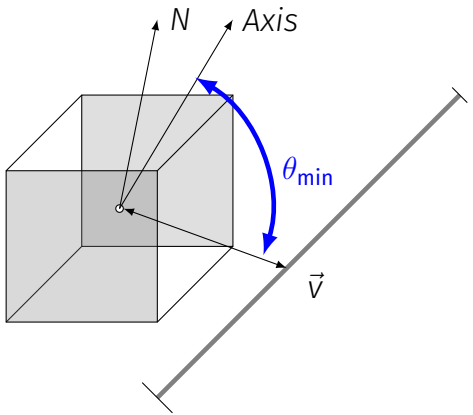
A conservative approach

The vector $\vec{v}$ runs along the volume line integral and on the plane with normal N. The stop that minimizes the distance d_{min} might not coincide with the minimum angle θ_{min}. But we use both values to compute a conservative importance measure.

Here we see visually how, given a volume segment, we find the minimum distance and angle to compute the heuristic. Note that the point that minimizes one might not be the same that minimizes the other, but for simplicity we find those independently and assume they happen at the same location. This still produces a conservative heuriustic that guarantees we don't underestimate a light cluster.

- The two points might to coincide
- Compute independently and assume they coincide
- The result is still conservative

IMPORTANCE FROM A VOLUME SEGMENT



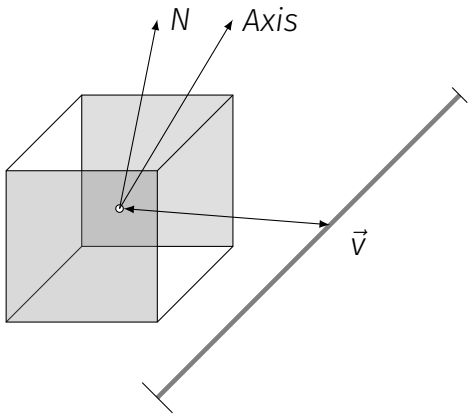
A conservative approach

The vector $\vec{v}$ runs along the volume line integral and on the plane with normal N. The stop that minimizes the distance d_{min} might not coincide with the minimum angle θ_{min}. But we use both values to compute a conservative importance measure.

Here we see visually how, given a volume segment, we find the minimum distance and angle to compute the heuristic. Note that the point that minimizes one might not be the same that minimizes the other, but for simplicity we find those independently and assume they happen at the same location. This still produces a conservative heuriustic that guarantees we don't underestimate a light cluster.

- The two points might to coincide
- Compute independently and assume they coincide
- The result is still conservative

IMPORTANCE FROM A VOLUME SEGMENT



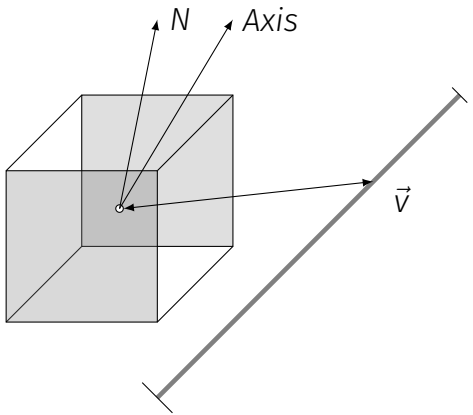
A conservative approach

The vector $\vec{v}$ runs along the volume line integral and on the plane with normal *N*. The stop that minimizes the distance d_{min} might not coincide with the minimum angle θ_{min}. But we use both values to compute a conservative importance measure.

Here we see visually how, given a volume segment, we find the minimum distance and angle to compute the heuristic. Note that the point that minimizes one might not be the same that minimizes the other, but for simplicity we find those independently and assume they happen at the same location. This still produces a conservative heuriustic that guarantees we don't underestimate a light cluster.

- The two points might to coincide
- Compute independently and assume they coincide
- The result is still conservative

IMPORTANCE FROM A VOLUME SEGMENT



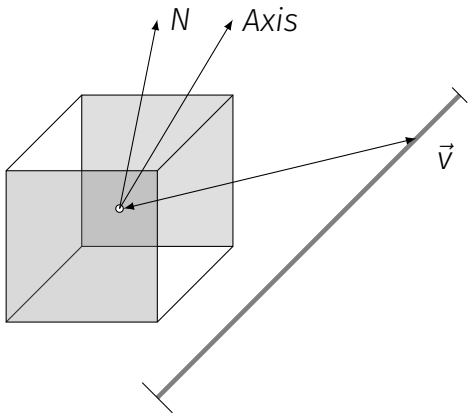
A conservative approach

The vector $\vec{v}$ runs along the volume line integral and on the plane with normal N. The stop that minimizes the distance d_{min} might not coincide with the minimum angle θ_{min}. But we use both values to compute a conservative importance measure.

Here we see visually how, given a volume segment, we find the minimum distance and angle to compute the heuristic. Note that the point that minimizes one might not be the same that minimizes the other, but for simplicity we find those independently and assume they happen at the same location. This still produces a conservative heuriustic that guarantees we don't underestimate a light cluster.

- The two points might to coincide
- Compute independently and assume they coincide
- The result is still conservative

IMPORTANCE FROM A VOLUME SEGMENT



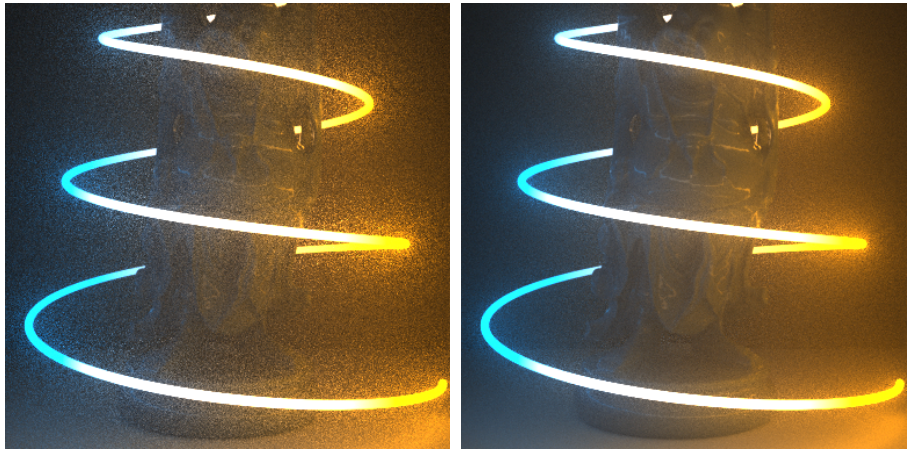
A conservative approach

The vector $\vec{v}$ runs along the volume line integral and on the plane with normal *N*. The stop that minimizes the distance d_{min} might not coincide with the minimum angle θ_{min}. But we use both values to compute a conservative importance measure.

Here we see visually how, given a volume segment, we find the minimum distance and angle to compute the heuristic. Note that the point that minimizes one might not be the same that minimizes the other, but for simplicity we find those independently and assume they happen at the same location. This still produces a conservative heuriustic that guarantees we don't underestimate a light cluster.

- The two points might to coincide
- Compute independently and assume they coincide
- The result is still conservative

RESULTS



Energy sampling VS our method

You can see here an equal time comparison, rendered in a couple of minutes. It's a big improvement, specially near the emitters of the spiral.

OUR METHOD IN PRODUCTION

- Savings with respect to the previous local acceleration structure:
  - 40 to 60% time saving for scenes from 100 to 1,000 lights
  - 10 to 20% even in scenes with just 10 lights
- Artists no longer worry about the range of a light, just **keep it physical!**

This is now our default lighting algorithm. The performance improvement is comparing to the previous acceleration that required all lights to be carefully tunned by the artists to have a narrow influence. They are now free from that. Kind of a ”go nuts” message, but the performance is holding. And we sometimes get speed ups even for as few as 10 lights. And of course our mesh lights got much better too.

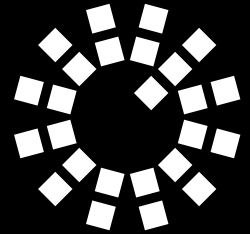
- This is our lighting system now for all productions
- Savings from 40 to 60% with many lights
- Comparing to the old local acceleration
- Artists free from adjusting light ranges, go nuts on lights

KNOWN ISSUES

- Noise discontinuities due to splitting
  - Usually worth it for the noise reduction
  - Adaptive render can resolve them
  - Future work: more intelligent split
- Powerful and occluded lights
  - They drain samples from others
  - Currently fixed by user intervention: force sampling
- Non-physical decay rate
  - Confuses the importance heuristic

The system is not perfect. We sometimes get noise discontinuities due to sampling splitting. They usually go away at the final quality or otherwise our adaptive render kicks in and solves the problem. Also very strong and occluded lights can misguide the sampling, or lights with non physical decay. But these are not very common and we have a way to single out a particular light and exclude it from the heuristics and do traditional light loop. It is normally just one or two naughty lights when it comes up.

- Noise discontinuities due to splitting
- Powerful occluded lights or non physical decay
- Fairly uncommon, we work around single-ing them out



SONY PICTURES

imageworks

25TH ANNIVERSARY

Thank You! Questions?