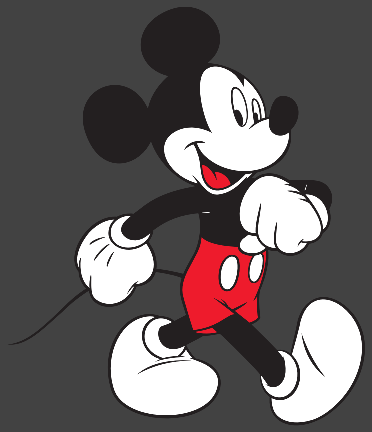
Progressive Expectation–Maximization for Hierarchical Volumetric Photon Mapping

Wenzel Jakob[1,2] Christian Regg[1,3] Wojciech Jarosz[1]

[1] Disney Research, Zürich
[2] Cornell University
[3] ETH Zürich



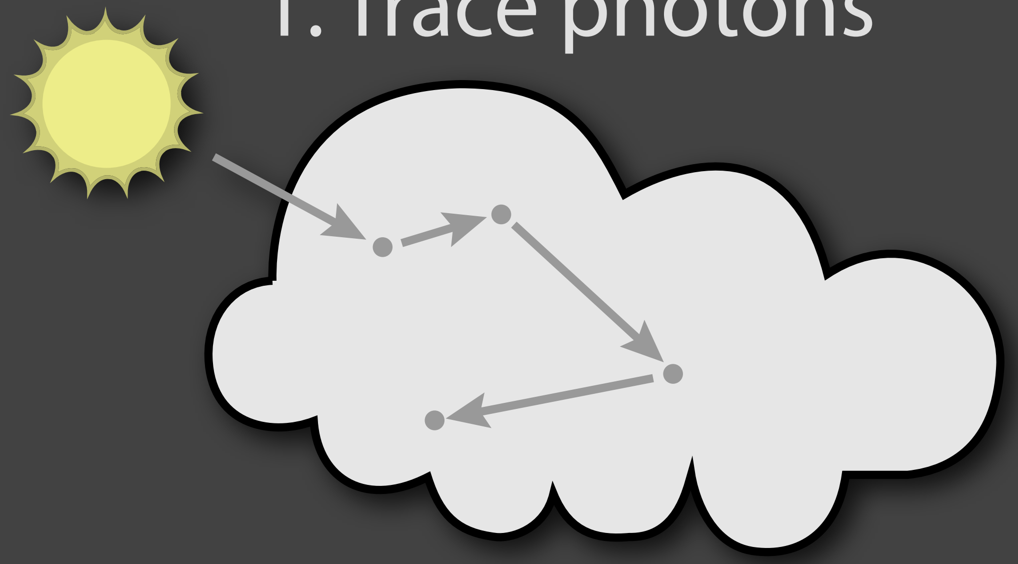
Thursday, 6 September 12

Thank you for the introduction. This project is joint work with Christian Regg and Wojciech Jarosz and was conducted during my time at Disney Research Zürich.

Motivation

Volumetric photon mapping

1. Trace photons

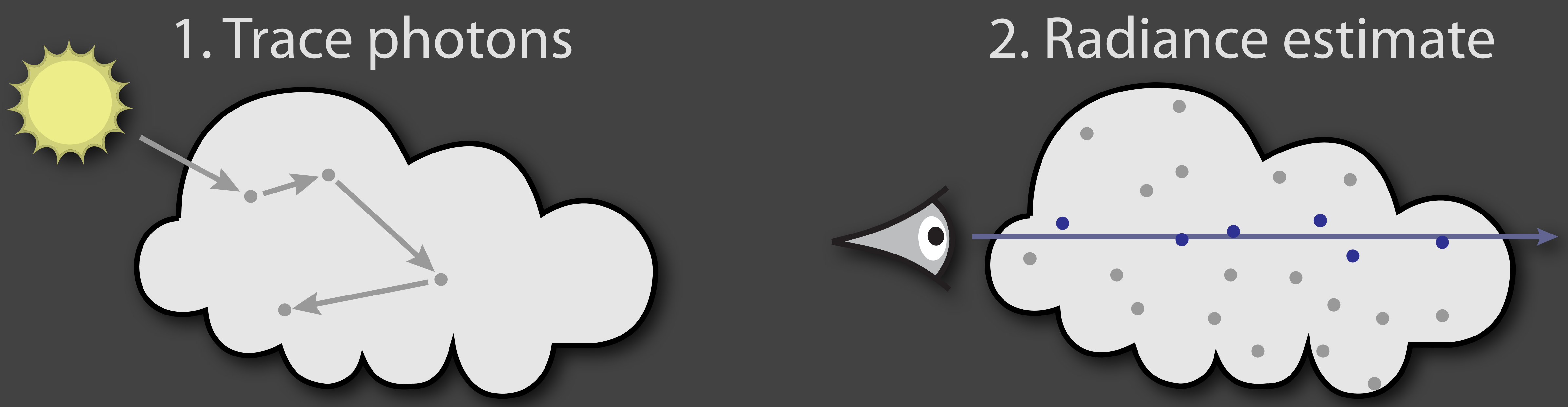


Thursday, 6 September 12

Suppose that we’re interested in rendering a participating medium with global illumination. A popular algorithm that does this is volumetric photon mapping. In this two-stage method, a preprocess step first generates and stores a large number of virtual scattering events named “photons”.
[CLICK]
And later during rendering, the illumination arriving along eye rays is then estimated using density estimation over those photons.

Motivation

Volumetric photon mapping

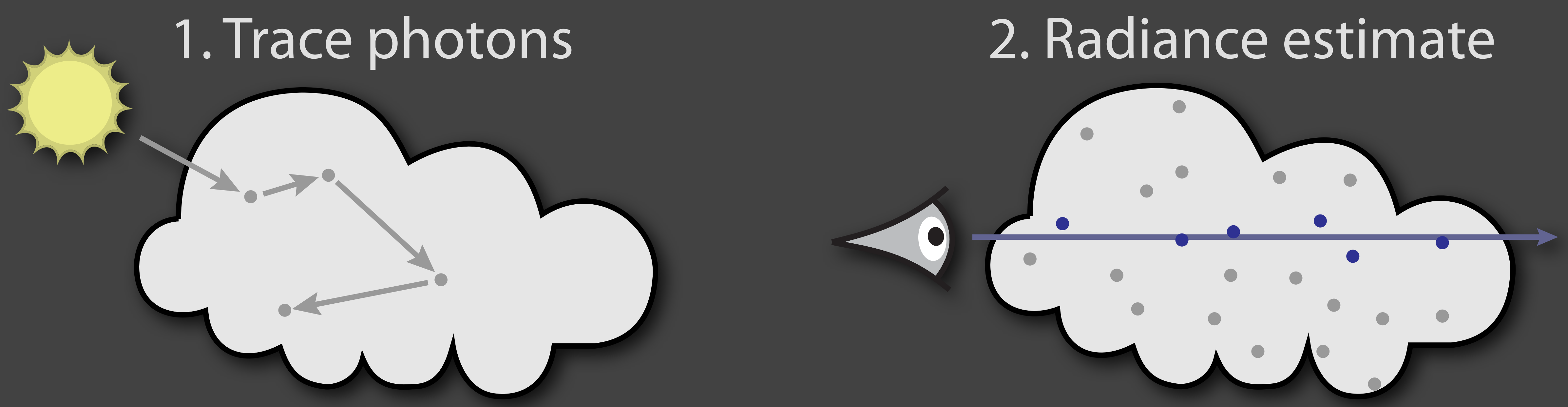


Thursday, 6 September 12

Suppose that we're interested in rendering a participating medium with global illumination. A popular algorithm that does this is volumetric photon mapping. In this two-stage method, a preprocess step first generates and stores a large number of virtual scattering events named "photons".
[CLICK]
And later during rendering, the illumination arriving along eye rays is then estimated using density estimation over those photons.

Motivation

Volumetric photon mapping



Issues

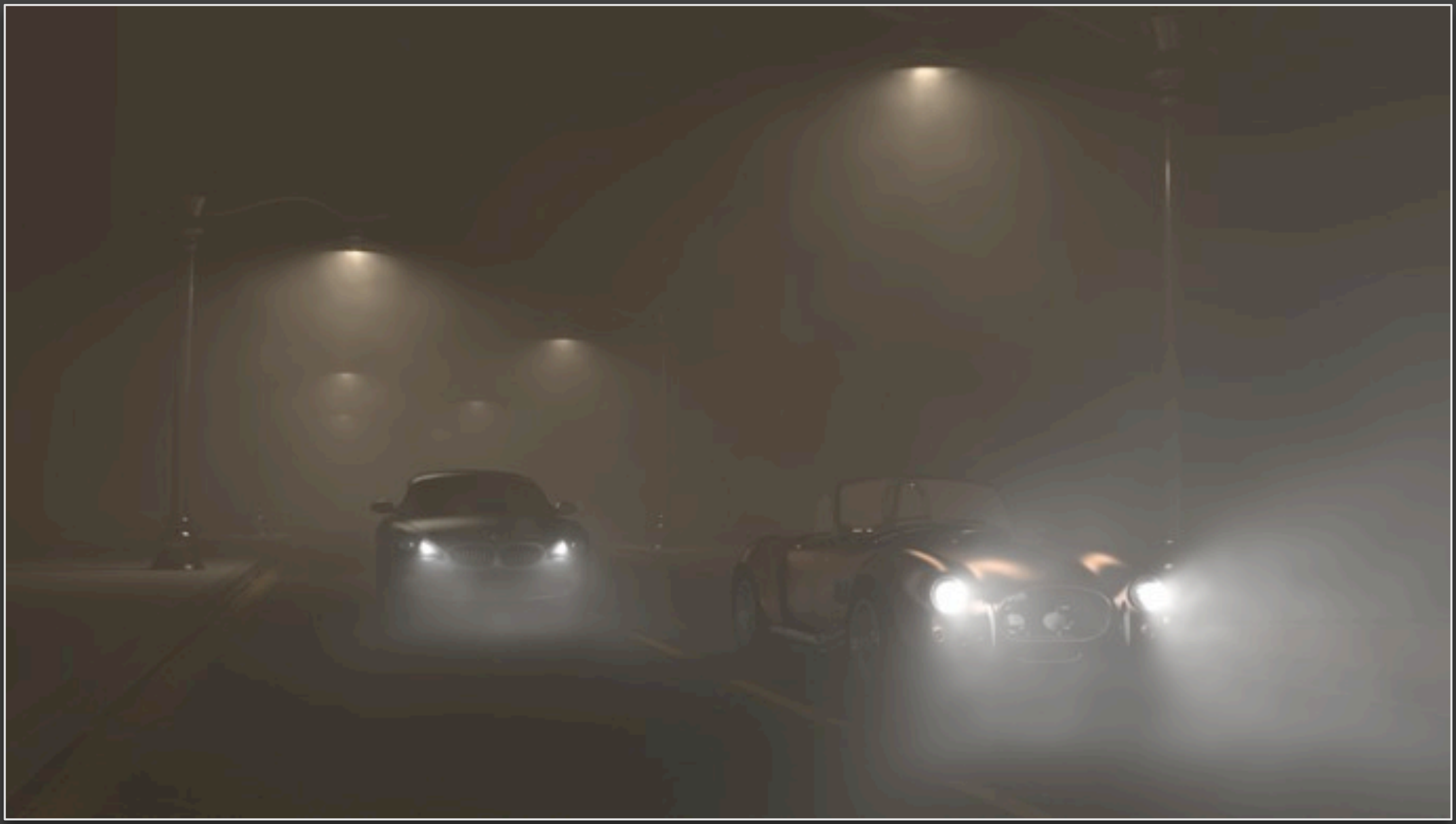
- high-frequency illumination requires *many* photons
- time spent on photons that contribute very little
- prone to temporal flickering

There are a several problems with this approach: first, to capture fine illumination details, an excessively large number of photons is necessary in practice, which leads to high memory consumption and long rendering times.

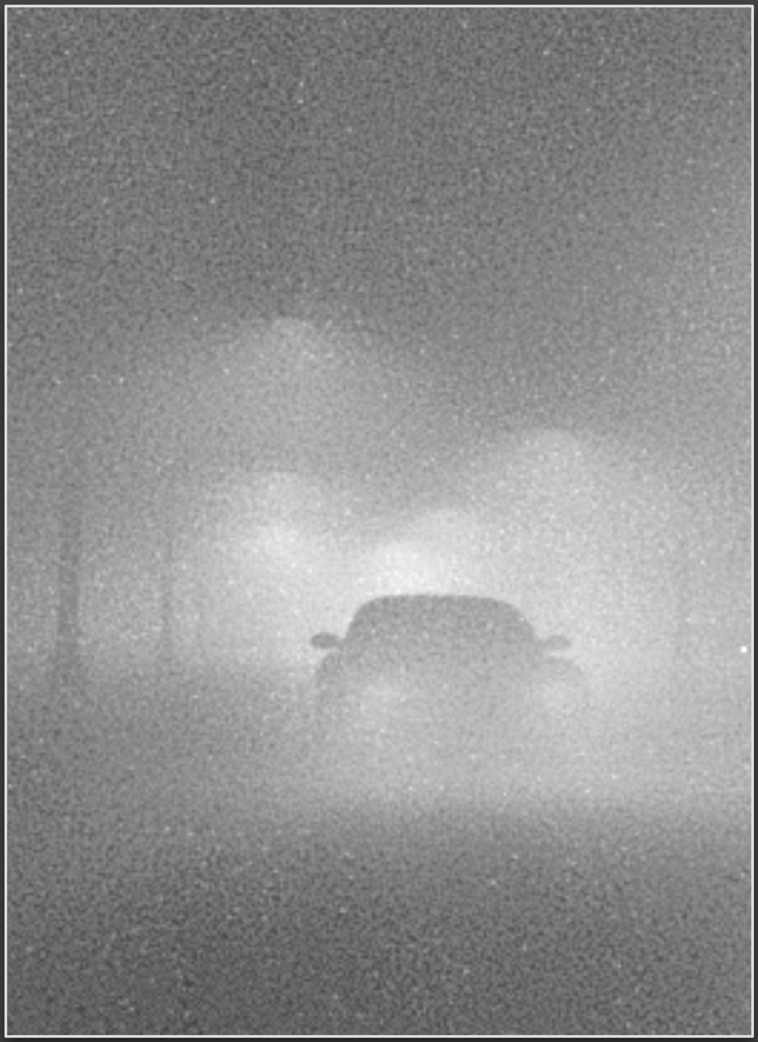
The rendering step tends to spend much time processing photons that contribute very little to the rendered image.

And finally, this technique is also prone to temporal flickering when rendering animations, since the photons are generated stochastically.

Motivation



Beam radiance estimate : 917K photons

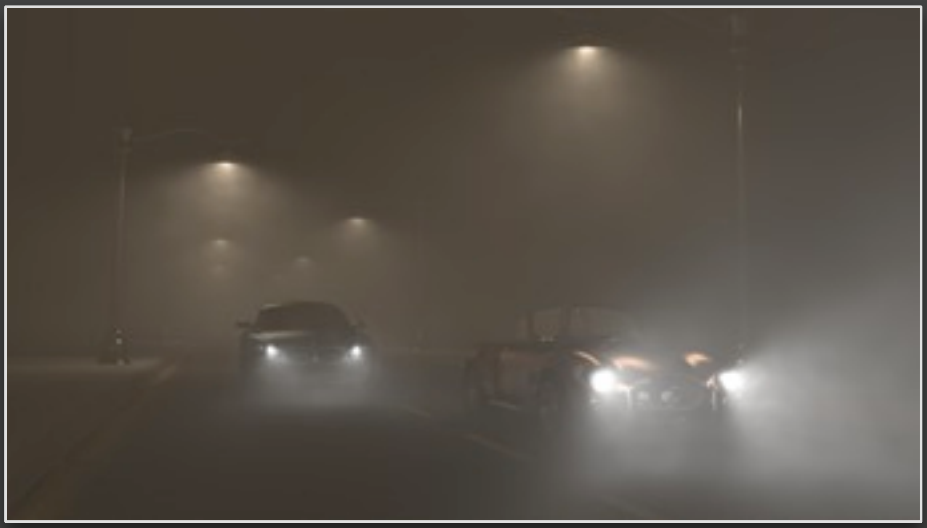


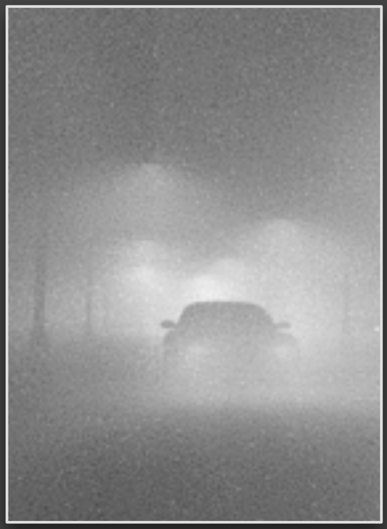
Per-pixel render time

Here is an example showing scene rendered with almost a million photons. It uses the beam radiance estimate, which is a variant of volumetric photon mapping. The right side shows the per-pixel render time, and you can see that a significant amount of time is spent in areas that are actually strongly attenuated in the rendering.

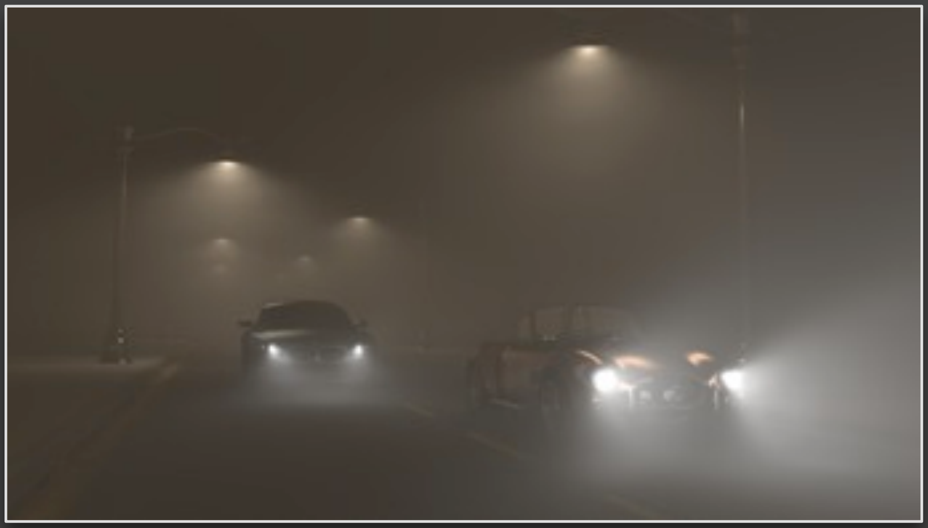
Motivation

Beam radiance estimate : 917K photons


Render time: 281 s


Per-pixel render time

Our method: 4K Gaussians


Render time: 125 s

Per-pixel render time

Our approach

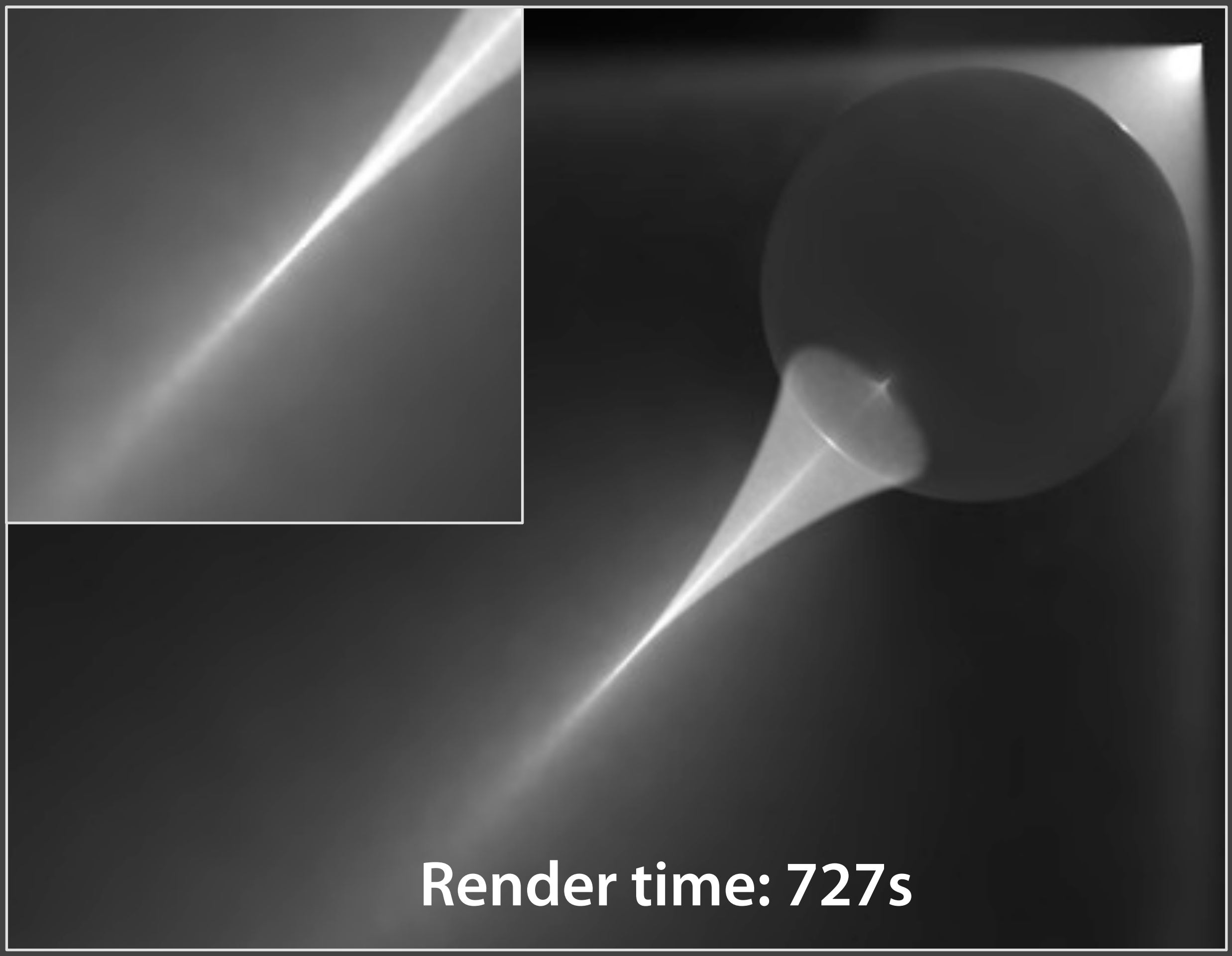
- represent radiance using a Gaussian mixture model (**GMM**)
- fit using progressive expectation maximization (**EM**)
- render with multiple levels of detail

In comparison, our method renders this scene faster using an alternative radiance representation that only uses four thousand terms for this scene. More specifically, we use a hierarchical Gaussian mixture model that is computed using a variant of the expectation maximization algorithm. We also use a hierarchical rendering stage, which can switch to lower levels of detail where it makes sense -- for instance, when drawing the attenuated distant light sources.

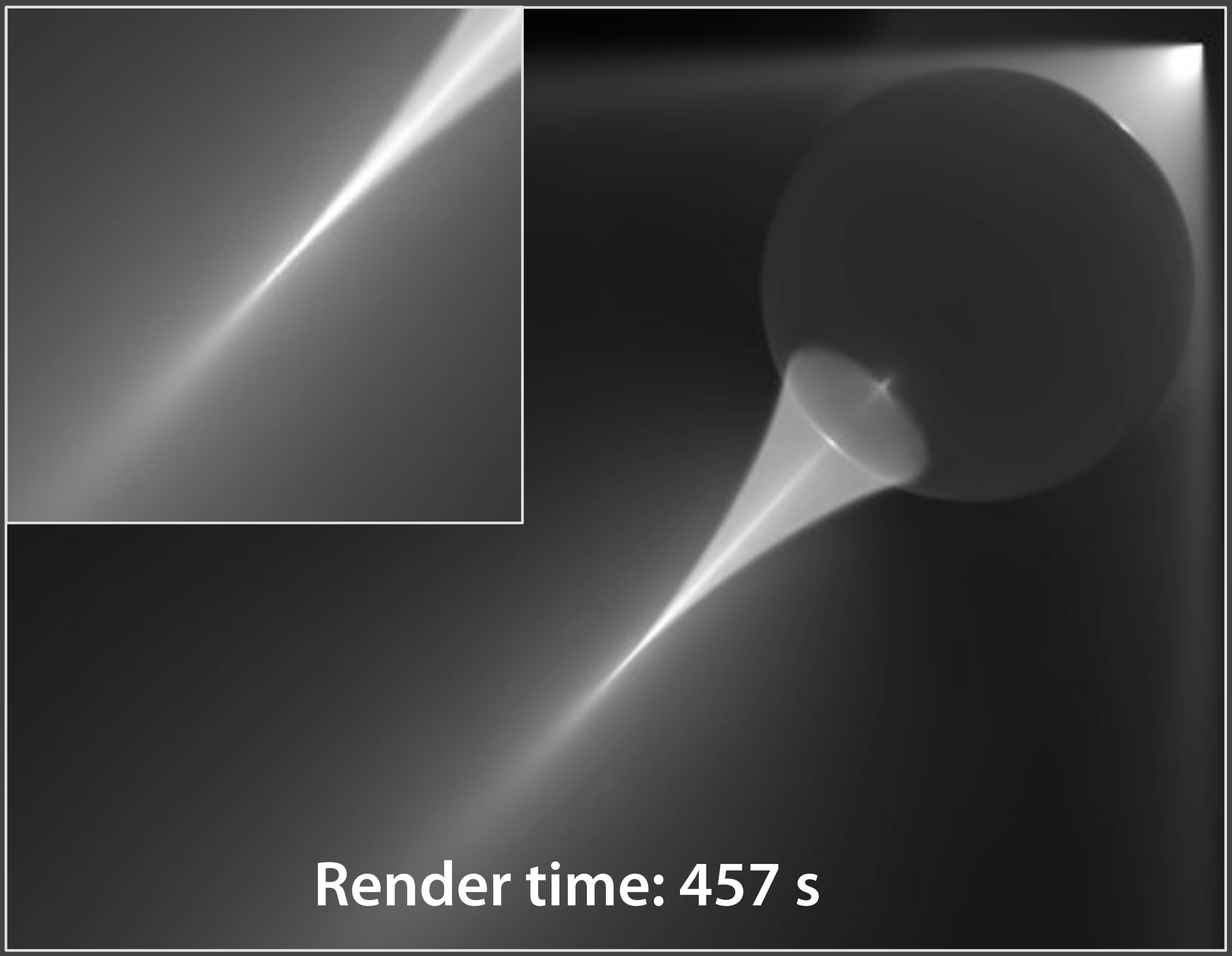
During the talk, I sometimes abbreviate expectation maximization as EM, and gaussian mixture models as GMMs.

Motivation

Beam radiance estimate : 4M photons



Our method: 16K Gaussians



Our approach

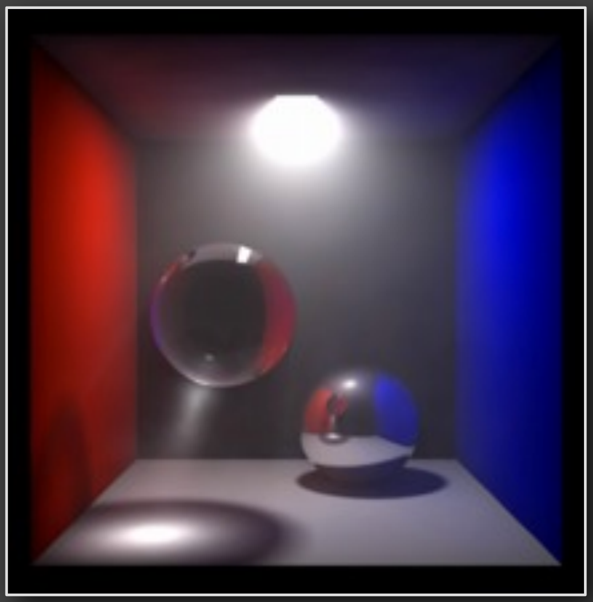
- represent radiance using a Gaussian mixture model (**GMM**)
- fit using progressive expectation maximization (**EM**)
- render with multiple levels of detail

Thursday, 6 September 12

Here, you can see another comparison between the Beam Radiance and our method -- this time showing a sphere caustic rendered with 4 Million photons and 16 thousand gaussians. Again, our method renders this image faster, and it intelligently blurs noise, **while** retaining important image features.

Related work (1/3)

- Volumetric photon mapping [Jensen and Christensen 98]



- The Beam Radiance Estimate (**BRE**) [Jarosz et al. 08]

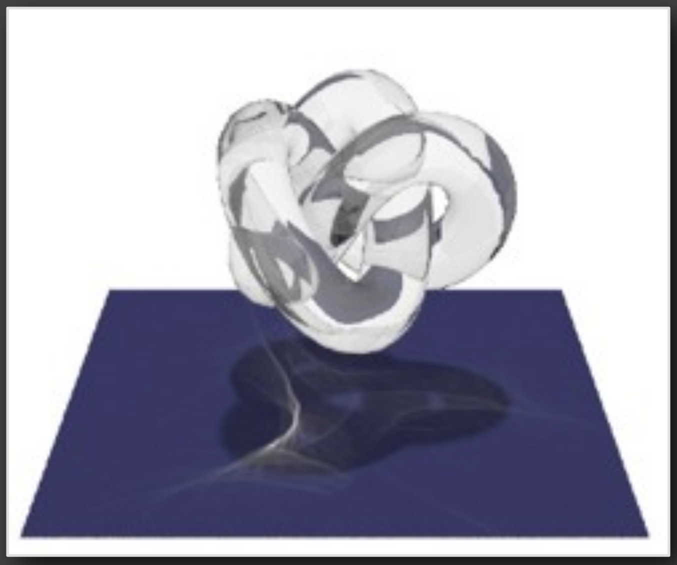
There are several related works in the areas of computer graphics, statistics, and machine learning.

Volumetric photon mapping proposed in 1998, was the first method that could render phenomena such as volume caustics at reasonable cost.

The beam radiance estimate, henceforth referred to as “BRE”, proposes a significant algorithmic improvement of the density estimation step in volumetric photon mapping. Instead of performing ray marching, it finds all photons along a camera ray, which results in better quality and faster rendering time. We

Related work (2/3)

- Diffusion based photon mapping
  [Schjøth et al. 08]



- Photon relaxation
  [Spencer et al. 09]

- Hierarchical photon mapping
  [Spencer et al. 09]

Thursday, 6 September 12

There are a number of techniques that solve related problems in surface rendering:

Diffusion based photon mapping achieves higher-quality estimates by introducing an anisotropic density query.

Photon relaxation adjusts the photons in a photon map by shifting their centers according to a set of heuristics.

And hierarchical photon mapping creates photon maps with multiple levels of detail, and dynamically chooses the appropriate resolution when executing a query.

In comparison, our technique has the benefit of all these three methods, but applied to the volumetric setting. It could be interpreted as using a special kind of photon map containing a small hierarchy of anisotropic photons, whose positions and other parameters are found using statistically sound optimization techniques.

Related work (3/3)

Rendering

- EWA splatting [Zwicker et al. 02]
- Meshless light transport [Lethinen et al. 08]
- Progressive photon mapping [Hachisuka et al. 08]

Thursday, 6 September 12

Anisotropic Gaussians have seen prior use in graphics, for example in the context of EWA splatting.
Other projects, such as meshless light transport, have explored the use of alternative hierarchical radiance representations.
Our method uses a progressive photon gathering step, which is similar in spirit to progressive photon mapping.
[CLICK]
We make use of two agglomerative clustering techniques by Goldberger et al. and Walter et al..
And finally, our progressive fitting algorithm builds upon the accelerated expectation maximization algorithm by Verbeek et al.

Related work (3/3)

Rendering

- EWA splatting [Zwicker et al. 02]
- Meshless light transport [Lethinen et al. 08]
- Progressive photon mapping [Hachisuka et al. 08]

Clustering and EM

- Clustering of GMMs [Goldberger et al. 05]
- Fast agglomerative clustering [Walter et al. 08]
- Accelerated EM [Verbeek et al. 06]

Thursday, 6 September 12

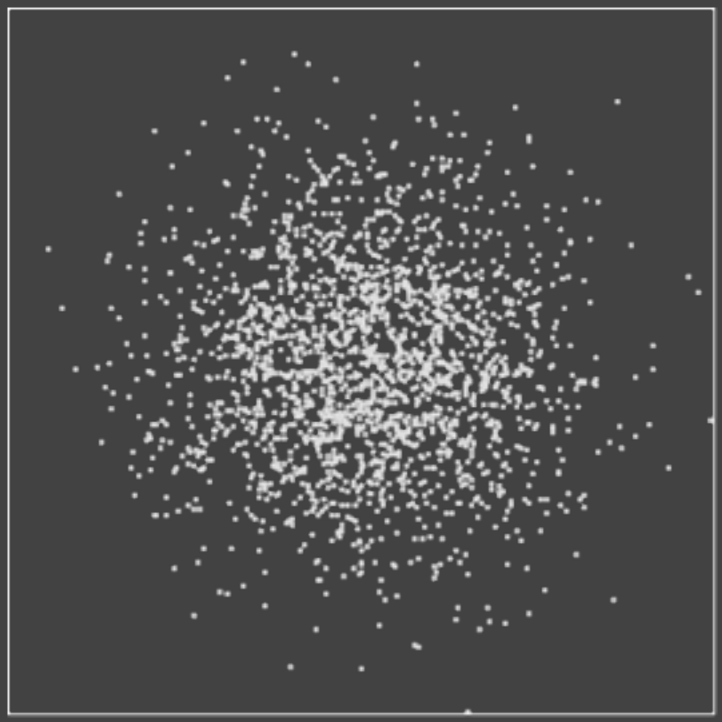
Anisotropic Gaussians have seen prior use in graphics, for example in the context of EWA splatting.
Other projects, such as meshless light transport, have explored the use of alternative hierarchical radiance representations.
Our method uses a progressive photon gathering step, which is similar in spirit to progressive photon mapping.
[CLICK]
We make use of two agglomerative clustering techniques by Goldberger et al. and Walter et al..
And finally, our progressive fitting algorithm builds upon the accelerated expectation maximization algorithm by Verbeek et al.

Density estimation

Thursday, 6 September 12

Let’s start with a bit of review:

Density estimation



$\overset{?}{\Rightarrow}$

Given photons
$x_1, x_2, \ldots$

approximately determine
their density f

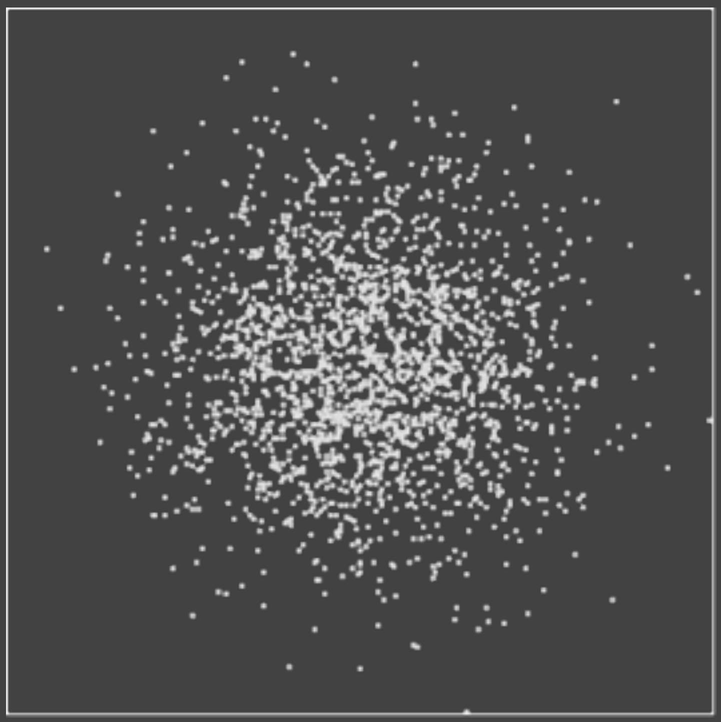
Thursday, 6 September 12

Suppose we are given a set of photons that we would like to use in a rendering. To do this, we'll need a way of determining their approximate density at [CLICK] arbitrary points.

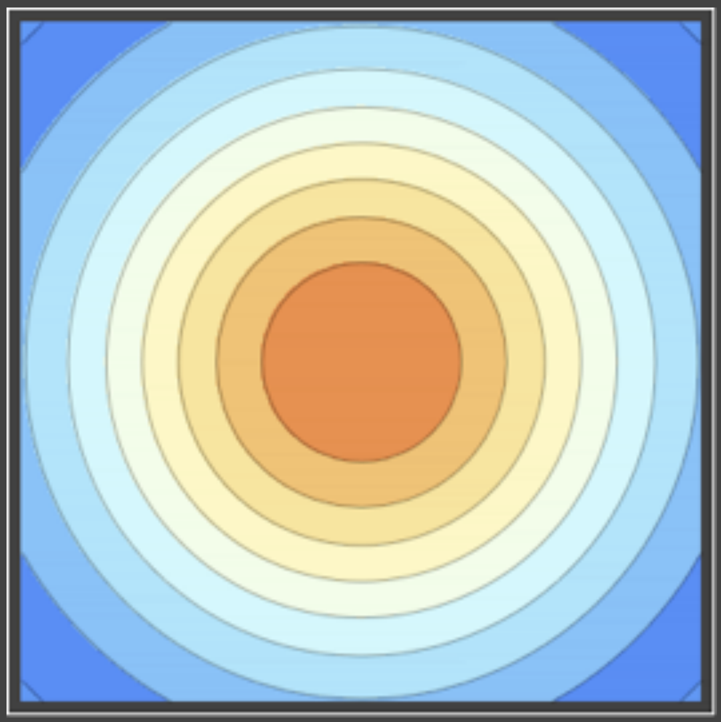
The non-parametric approach, which standard photon mapping uses, essentially works by [CLICK] counting the number of photons that fall into a small region around the point in question, and this needs to be repeated for every evaluation of the density function.

[CLICK] A different approach known as "parametric density estimation" assumes that the photons are drawn from a certain **known** distribution -- for instance, a 2D Normal distribution. In that case, all we need to do is to find the most suitable parameter values, and we have completely recovered the density function. This is the technique we will use in this paper.

Density estimation



$\overset{?}{\Longrightarrow}$



Given photons
$x_1, x_2, \ldots$

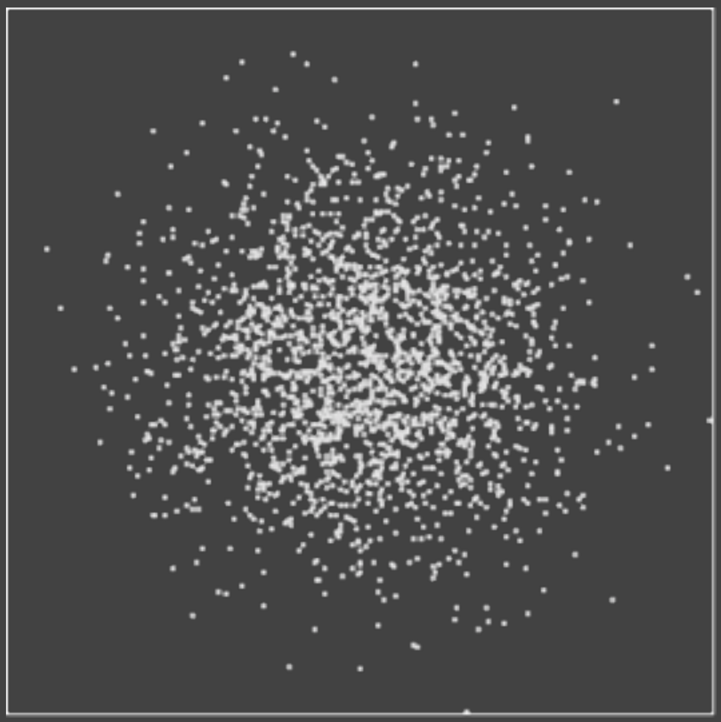
approximately determine
their density f

Suppose we are given a set of photons that we would like to use in a rendering. To do this, we'll need a way of determining their approximate density at [CLICK] arbitrary points.

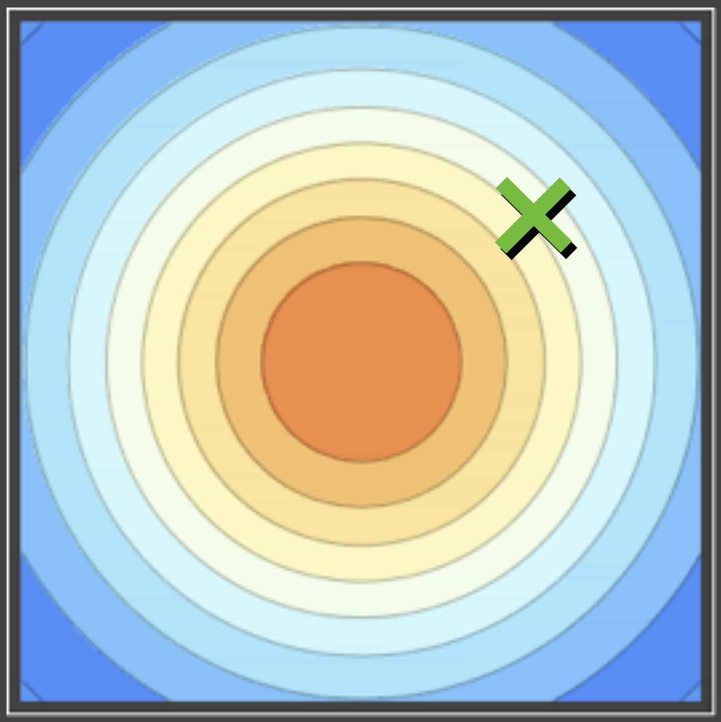
The non-parametric approach, which standard photon mapping uses, essentially works by [CLICK] counting the number of photons that fall into a small region around the point in question, and this needs to be repeated for every evaluation of the density function.

[CLICK] A different approach known as "parametric density estimation" assumes that the photons are drawn from a certain **known** distribution -- for instance, a 2D Normal distribution. In that case, all we need to do is to find the most suitable parameter values, and we have completely recovered the density function. This is the technique we will use in this paper.

Density estimation



$$\overset{?}{\Rightarrow}$$



Given photons $x_1, x_2, \ldots$

approximately determine their density f

Nonparametric:

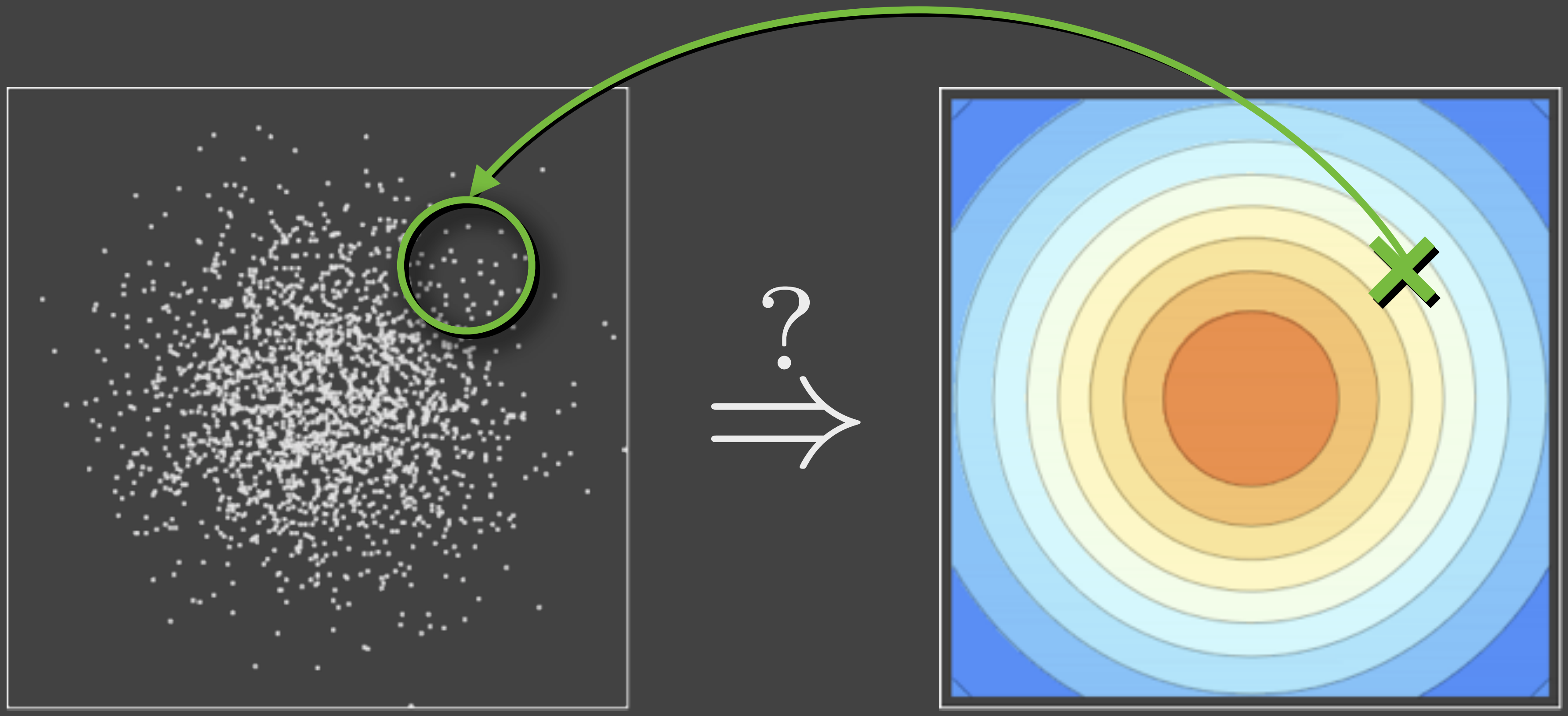
- Count the number of photons within a small region

Suppose we are given a set of photons that we would like to use in a rendering. To do this, we'll need a way of determining their approximate density at [CLICK] arbitrary points.

The non-parametric approach, which standard photon mapping uses, essentially works by [CLICK] counting the number of photons that fall into a small region around the point in question, and this needs to be repeated for every evaluation of the density function.

[CLICK] A different approach known as "parametric density estimation" assumes that the photons are drawn from a certain **known** distribution -- for instance, a 2D Normal distribution. In that case, all we need to do is to find the most suitable parameter values, and we have completely recovered the density function. This is the technique we will use in this paper.

Density estimation



Given photons $x_1, x_2, \ldots$

approximately determine their density f

Nonparametric:

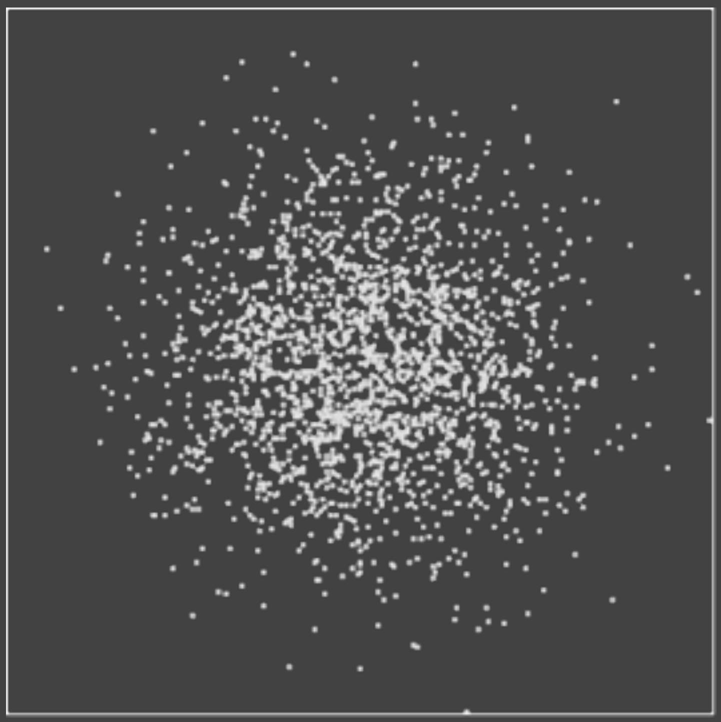
- Count the number of photons within a small region

Suppose we are given a set of photons that we would like to use in a rendering. To do this, we'll need a way of determining their approximate density at [CLICK] arbitrary points.

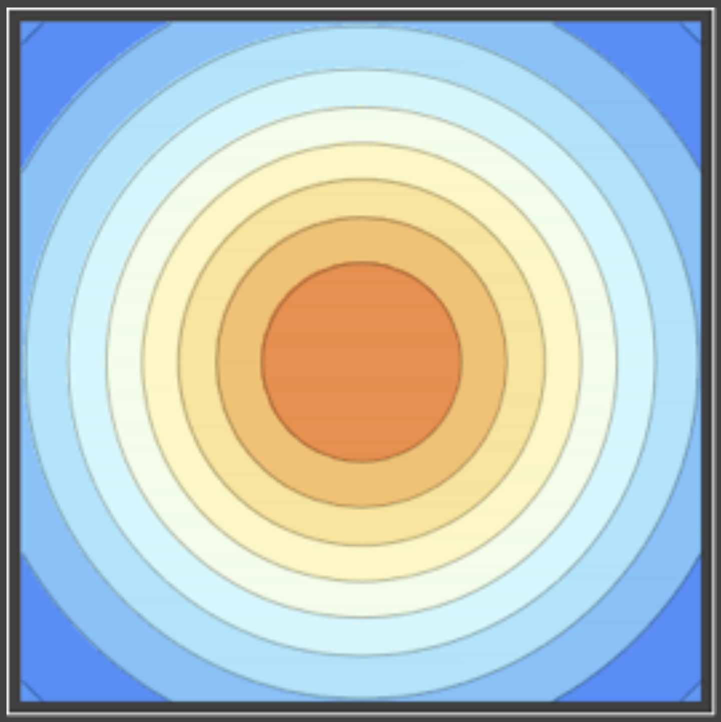
The non-parametric approach, which standard photon mapping uses, essentially works by [CLICK] counting the number of photons that fall into a small region around the point in question, and this needs to be repeated for every evaluation of the density function.

[CLICK] A different approach known as "parametric density estimation" assumes that the photons are drawn from a certain **known** distribution -- for instance, a 2D Normal distribution. In that case, all we need to do is to find the most suitable parameter values, and we have completely recovered the density function. This is the technique we will use in this paper.

Density estimation



$\overset{?}{\Longrightarrow}$



Given photons $x_1, x_2, \ldots$

approximately determine their density f

Nonparametric:

- Count the number of photons within a small region

Parametric:

- Find suitable parameters for a **known** distribution

Thursday, 6 September 12

Suppose we are given a set of photons that we would like to use in a rendering. To do this, we'll need a way of determining their approximate density at [CLICK] arbitrary points.

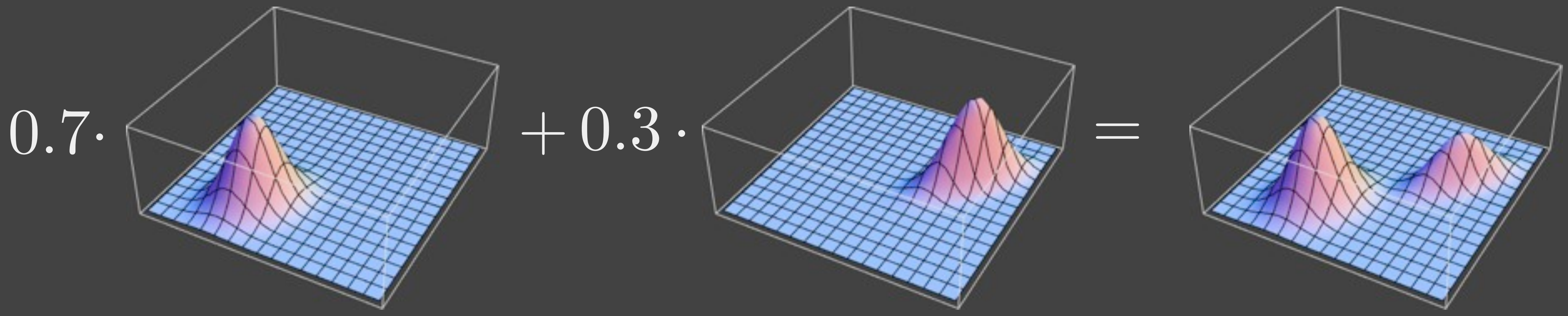
The non-parametric approach, which standard photon mapping uses, essentially works by [CLICK] counting the number of photons that fall into a small region around the point in question, and this needs to be repeated for every evaluation of the density function.

[CLICK] A different approach known as "parametric density estimation" assumes that the photons are drawn from a certain **known** distribution -- for instance, a 2D Normal distribution. In that case, all we need to do is to find the most suitable parameter values, and we have completely recovered the density function. This is the technique we will use in this paper.

Gaussian mixture models

- Photon density modeled as a weighted sum of Gaussians:

$$f\left(\mathbf{x} \mid \Theta\right) = \sum_{i=1}^{k} w_i \; g\left(\mathbf{x} \mid \Theta_i\right)$$



For this to work in practice, we'll need to use a distribution that is general enough so that it can work for arbitrary scenes. We use Gaussian mixture models, which are a popular choice in AI, data mining and statistics.

A gaussian mixture model is simply a weighted sum of Gaussian basis functions, each of which has a specifiable mean and covariance matrix, and you can see a simple 2D example at the bottom. The number of Gaussian terms k, is usually fixed. In our method, this is the main knob that affects the quality of the result.

Note that since we're interested in volumetric radiance, we actually use three-dimensional Gaussians in this paper.

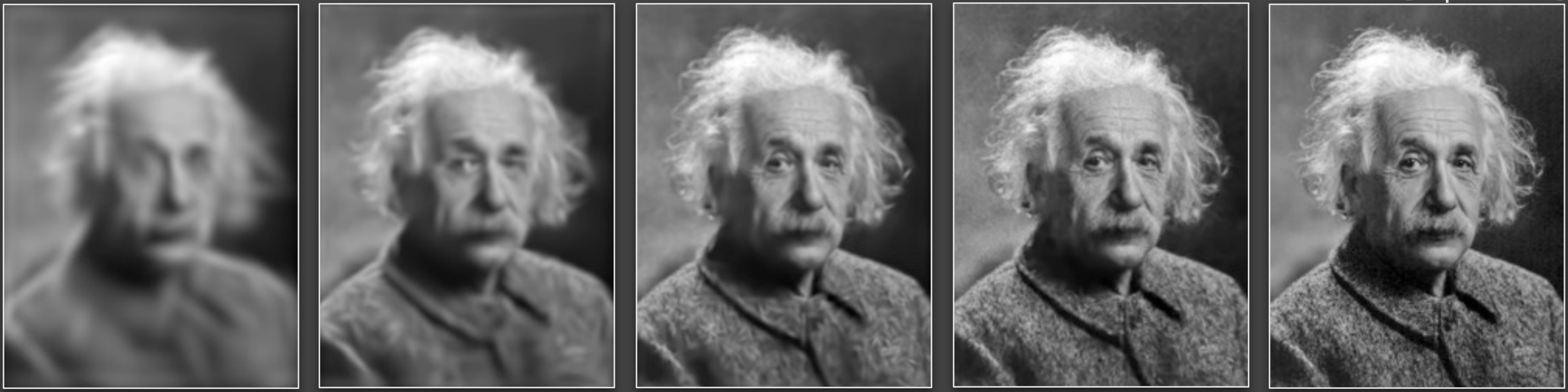
Of course, the real radiance distribution isn't necessarily of this type -- [CLICK]

Gaussian mixture models

- Photon density modeled as a weighted sum of Gaussians:

$$f(\mathbf{x} \mid \Theta) = \sum_{i=1}^{k} w_i \, g(\mathbf{x} \mid \Theta_i)$$

[Papas et al.]



256 Gaussians　1024 Gaussians　4096 Gaussians　16384 Gaussians　Target density

Thursday, 6 September 12

.. but with an increasingly large number of mixture terms, we can approximate it arbitrarily well. The bottom series of images shows a demonstration of this process applied to a 2D image.

Gaussian mixture models

- Photon density modeled as a weighted sum of Gaussians:

$$f\left(\mathbf{x} \mid \Theta\right) = \sum_{i=1}^{k} w_i \; g\left(\mathbf{x} \mid \Theta_i\right)$$

Unknown parameters Θ:

1. Weights 2. Means 3. Covariance matrices

Thursday, 6 September 12

The missing parameters then that need to be estimated are the weights, [CLICK] means, [CLICK] and covariance matrices for each one of the Gaussian terms. There will be many of them --usually we use somewhere between four and sixty-four thousand Gaussians.

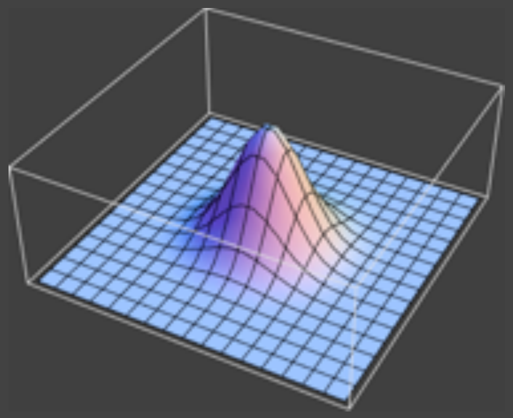
Gaussian mixture models

- Photon density modeled as a weighted sum of Gaussians:

$$f\left(\mathbf{x} \mid \Theta\right) = \sum_{i=1}^{k} w_i \; g\left(\mathbf{x} \mid \Theta_i\right)$$

Unknown parameters Θ:

1. Weights
2. Means
3. Covariance matrices



Thursday, 6 September 12

The missing parameters then that need to be estimated are the weights, [CLICK] means, [CLICK] and covariance matrices for each one of the Gaussian terms. There will be many of them --usually we use somewhere between four and sixty-four thousand Gaussians.

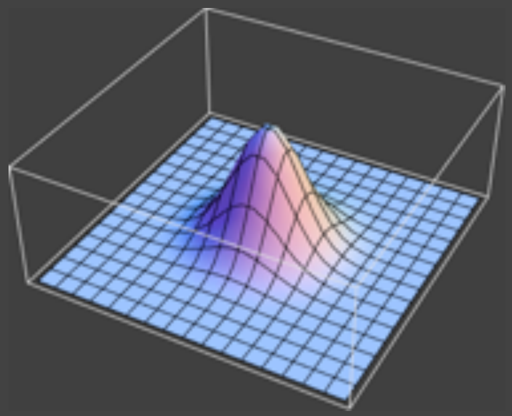
Gaussian mixture models

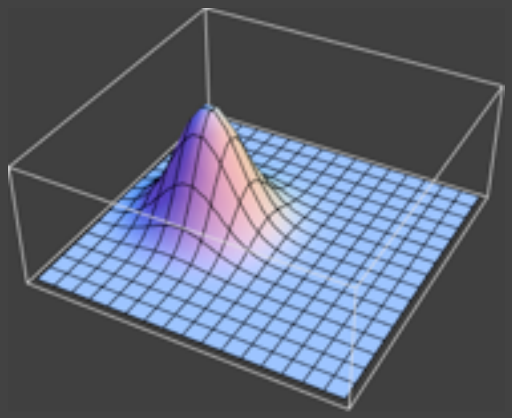
- Photon density modeled as a weighted sum of Gaussians:

$$f\left(\mathbf{x} \mid \Theta\right) = \sum_{i=1}^{k} w_i \; g\left(\mathbf{x} \mid \Theta_i\right)$$

Unknown parameters Θ:

1. Weights
2. Means
3. Covariance matrices





Thursday, 6 September 12

The missing parameters then that need to be estimated are the weights, [CLICK] means, [CLICK] and covariance matrices for each one of the Gaussian terms. There will be many of them --usually we use somewhere between four and sixty-four thousand Gaussians.

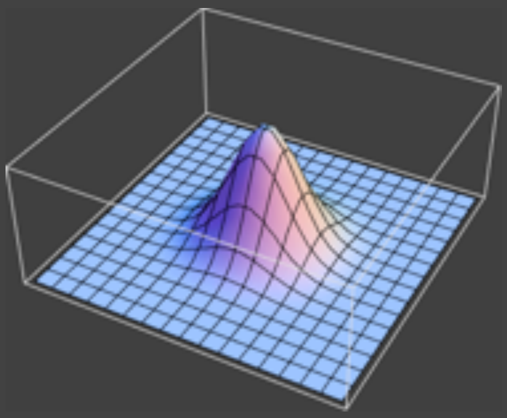
Gaussian mixture models

- Photon density modeled as a weighted sum of Gaussians:

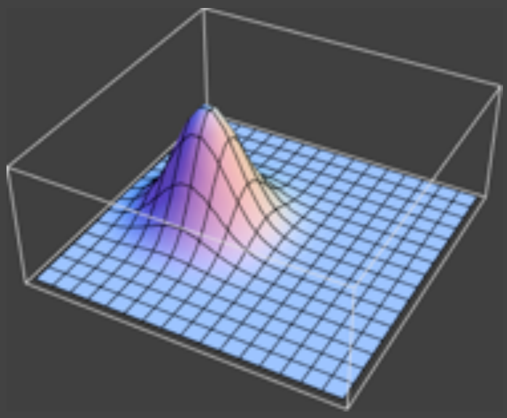
$$f\left(\mathbf{x} \mid \Theta\right) = \sum_{i=1}^{k} w_i \; g\left(\mathbf{x} \mid \Theta_i\right)$$

Unknown parameters Θ:

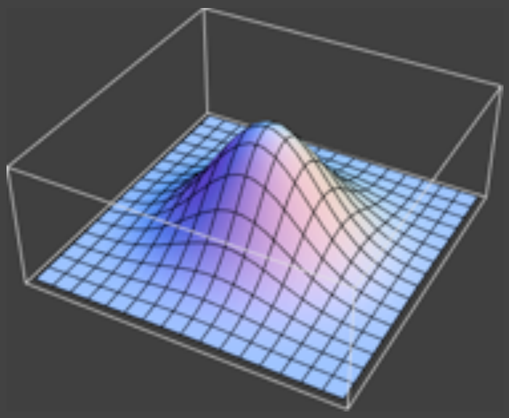
1. Weights

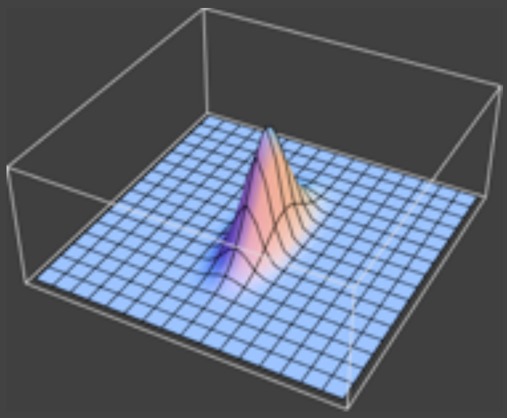


2. Means



3. Covariance matrices





The missing parameters then that need to be estimated are the weights, [CLICK] means, [CLICK] and covariance matrices for each one of the Gaussian terms. There will be many of them --usually we use somewhere between four and sixty-four thousand Gaussians.

Maximum likelihood estimation

Approach: find the “most likely” parameters, i.e.

$$\underset{\substack{\uparrow\\\text{Estimated parameters}}}{\Theta^*} := \operatorname*{argmax}_{\Theta} \prod_{i=1}^{n} \overset{\substack{\text{Mixture model}\\\downarrow}}{f}\left(\underset{\substack{\uparrow\\\text{Photon locations}}}{x_i} \mid \Theta\right)$$

Thursday, 6 September 12

Having settled on parametric density estimation and Gaussian mixtures, we need some way of finding suitable parameter values.

An intuitive way of phrasing this problem is to find the parameters that are the most **likely.** Mathematically, this can be expressed as a maximization problem, where we evaluate the density function at every photon, and look for the parameters that maximize this combined probability.

This is a standard approach known as maximum likelihood estimation, and the good news is that there is a powerful solution algorithm [CLICK] known as expectation maximization, which works particularly well when this approach is used together with Gaussian mixture models.

Maximum likelihood estimation

Approach: find the "most likely" parameters, i.e.

Mixture model

$$\underbrace{\Theta^*}_{\text{Estimated parameters}} := \operatorname*{argmax}_{\Theta} \prod_{i=1}^{n} \overset{\text{Mixture model}}{f} \left(\underbrace{x_i}_{\text{Photon locations}} \mid \Theta \right)$$

Estimated parameters

Photon locations

➔ **Expectation maximization**

Thursday, 6 September 12

Having settled on parametric density estimation and Gaussian mixtures, we need some way of finding suitable parameter values.

An intuitive way of phrasing this problem is to find the parameters that are the most **likely.** Mathematically, this can be expressed as a maximization problem, where we evaluate the density function at every photon, and look for the parameters that maximize this combined probability.

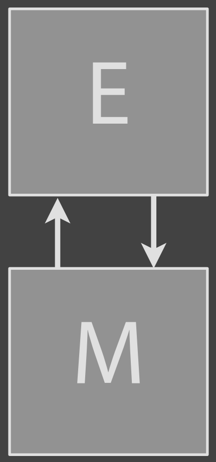
This is a standard approach known as maximum likelihood estimation, and the good news is that there is a powerful solution algorithm [CLICK] known as expectation maximization, which works particularly well when this approach is used together with Gaussian mixture models.

Expectation maximization

- Two components:

E-Step: establish soft assignment between photons and Gaussians

M-Step: maximize the expected likelihood



EM can be thought of as a soft version of K-means, also known as lloyd-relaxation. It consists of two components:
    The E-step establishes a soft assignment between every point and every Gaussian in the mixture model,
and the M-step uses these assignments to maximize a likelihood function.
As we keep running these E and M-steps, our parameters will improve steadily until convergence.

There are a two issues though [CLICK]: first of all, EM will only find a locally optimal solution -- we can't do much about that, except giving it a relatively good starting guess.
    And secondly, plain EM is extremely slow if not impractical when we're dealing with millions of photons and tens of thousands of Gaussians. In a moment, we'll take a look at how this problem can be solved.

Expectation maximization

- Two components:

  E-Step: establish soft assignment between photons and Gaussians

  M-Step: maximize the expected likelihood

$$\begin{tikzcd} \boxed{E} \arrow[d, shift left] \\ \boxed{M} \arrow[u, shift left] \end{tikzcd}$$

- Finds a locally optimal solution

  ➔ good starting guess needed!

- Slow and scales poorly — $\mathcal{O}(n^2)$ (where n: photon count)

EM can be thought of as a soft version of K-means, also known as lloyd-relaxation. It consists of two components:

The E-step establishes a soft assignment between every point and every Gaussian in the mixture model,

and the M-step uses these assignments to maximize a likelihood function.

As we keep running these E and M-steps, our parameters will improve steadily until convergence.

There are a two issues though [CLICK]: first of all, EM will only find a locally optimal solution -- we can't do much about that, except giving it a relatively good starting guess.

And secondly, plain EM is extremely slow if not impractical when we're dealing with millions of photons and tens of thousands of Gaussians. In a moment, we'll take a look at how this problem can be solved.

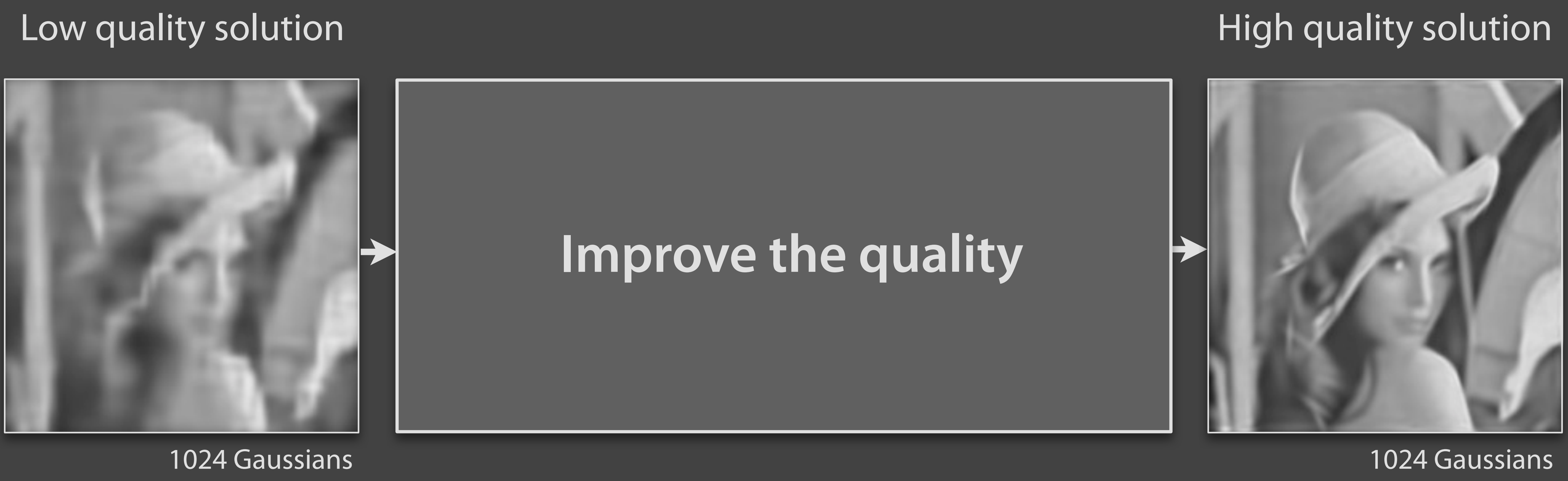
High-level fitting overview

Thursday, 6 September 12

[PAUSE....] Here is a high-level overview. [CLICK]

Since it our method also works in 2D, we demonstrate it on the famous Lena image. The input is a low quality Gaussian mixture, which is progressively refined until convergence.

High-level fitting overview

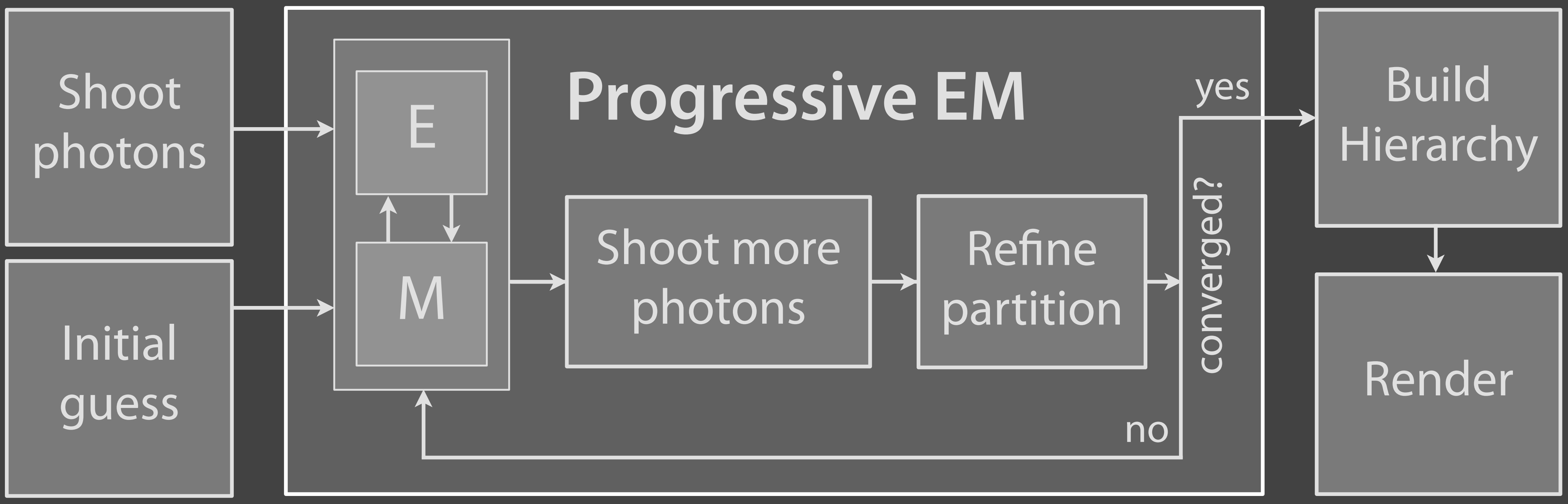


Thursday, 6 September 12

[PAUSE....] Here is a high-level overview. [CLICK]

Since it our method also works in 2D, we demonstrate it on the famous Lena image. The input is a low quality Gaussian mixture, which is progressively refined until convergence.

Pipeline overview

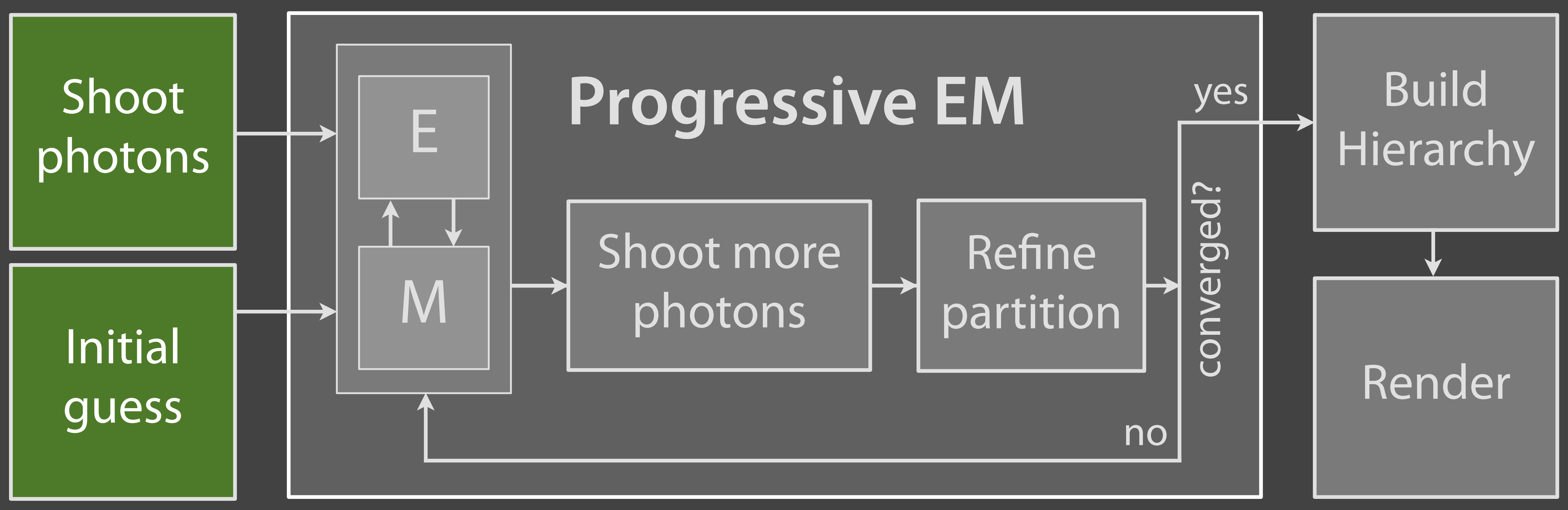


..and here is a more detailed pipeline diagram. We first start with a photon map and an initial guess [CLICK], which also computed from the photon map. The photon shooting pass is standard, hence it won’t be covered in this talk, and we’ll postpone discussion of the starting guess until later.

[CLICK] After these two steps, we run our progressive EM to improve the initial guess. It does essentially the same thing as plain EM, but much more efficiently. Our algorithm may continually shoot further photons if it is deemed necessary to improve the quality of the end result, and it runs until convergence is achieved.

[CLICK] We then create a level of detail representation, which allows us to render complex scenes efficiently.

Pipeline overview



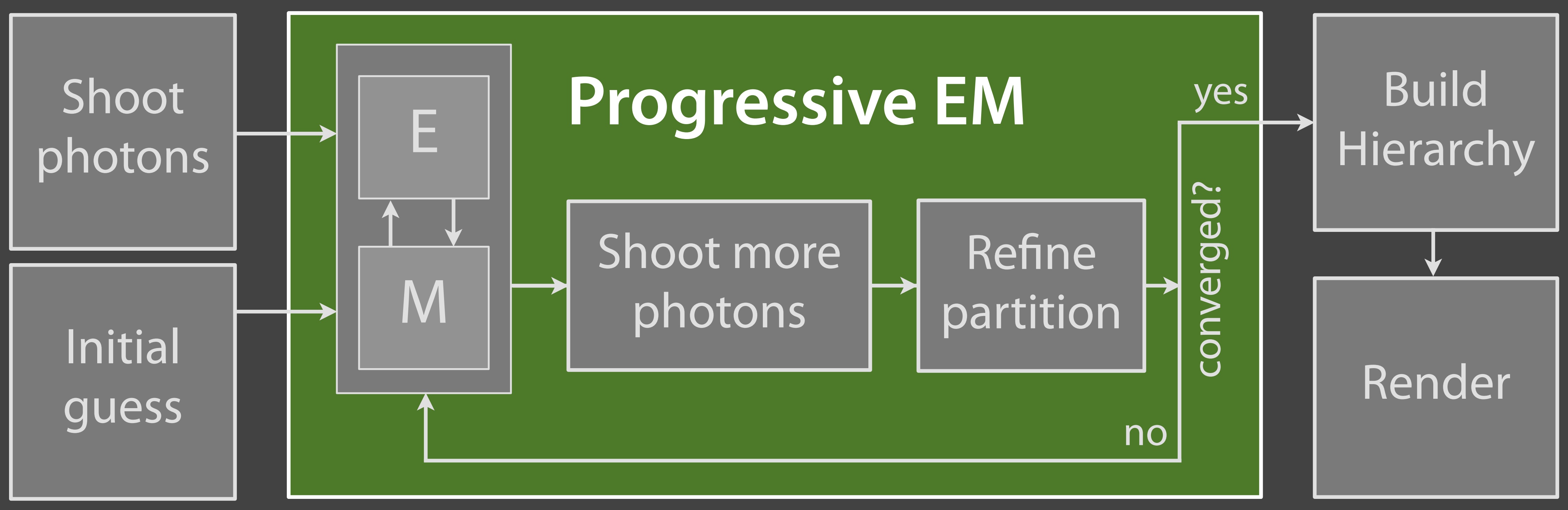
Thursday, 6 September 12

..and here is a more detailed pipeline diagram. We first start with a photon map and an initial guess [CLICK], which also computed from the photon map. The photon shooting pass is standard, hence it won't be covered in this talk, and we'll postpone discussion of the starting guess until later.

[CLICK] After these two steps, we run our progressive EM to improve the initial guess. It does essentially the same thing as plain EM, but much more efficiently. Our algorithm may continually shoot further photons if it is deemed necessary to improve the quality of the end result, and it runs until convergence is achieved.

[CLICK] We then create a level of detail representation, which allows us to render complex scenes efficiently.

Pipeline overview



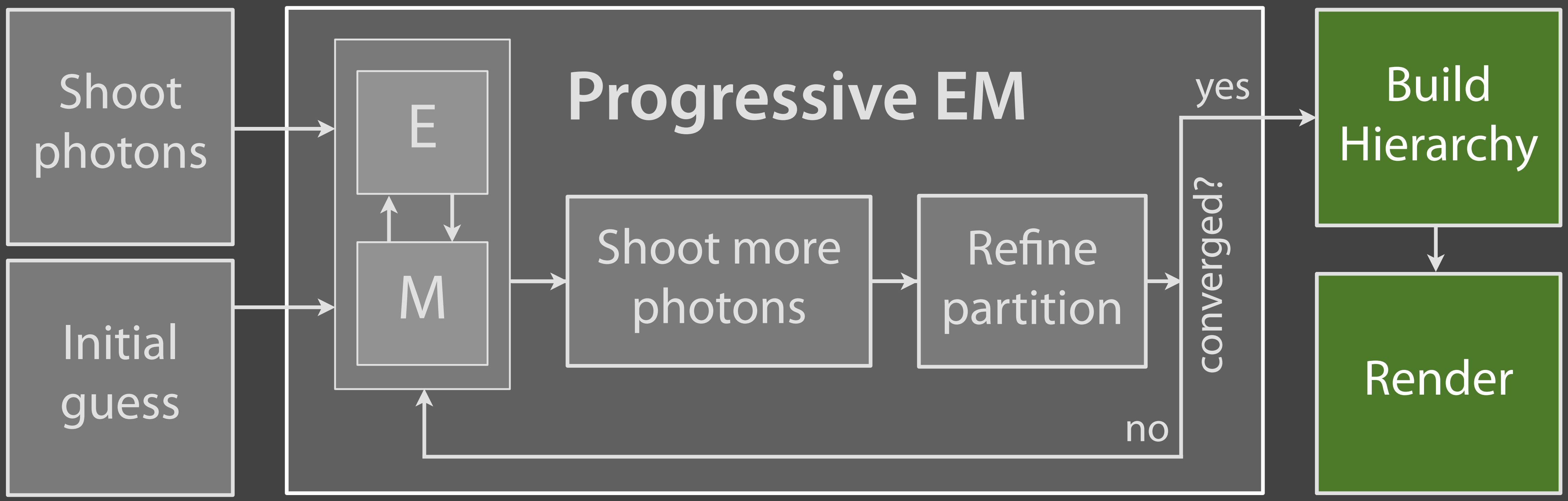
Thursday, 6 September 12

..and here is a more detailed pipeline diagram. We first start with a photon map and an initial guess [CLICK], which also computed from the photon map. The photon shooting pass is standard, hence it won't be covered in this talk, and we'll postpone discussion of the starting guess until later.

[CLICK] After these two steps, we run our progressive EM to improve the initial guess. It does essentially the same thing as plain EM, but much more efficiently. Our algorithm may continually shoot further photons if it is deemed necessary to improve the quality of the end result, and it runs until convergence is achieved.

[CLICK] We then create a level of detail representation, which allows us to render complex scenes efficiently.

Pipeline overview



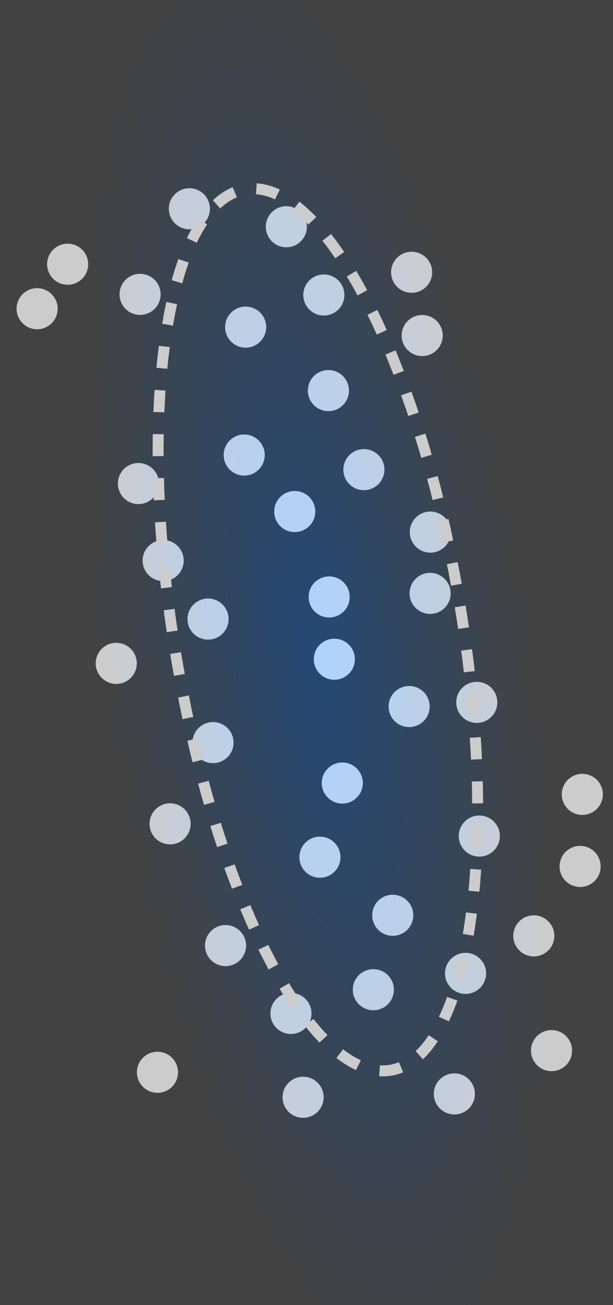
Thursday, 6 September 12

..and here is a more detailed pipeline diagram. We first start with a photon map and an initial guess [CLICK], which also computed from the photon map. The photon shooting pass is standard, hence it won't be covered in this talk, and we'll postpone discussion of the starting guess until later.

[CLICK] After these two steps, we run our progressive EM to improve the initial guess. It does essentially the same thing as plain EM, but much more efficiently. Our algorithm may continually shoot further photons if it is deemed necessary to improve the quality of the end result, and it runs until convergence is achieved.

[CLICK] We then create a level of detail representation, which allows us to render complex scenes efficiently.

Plain EM



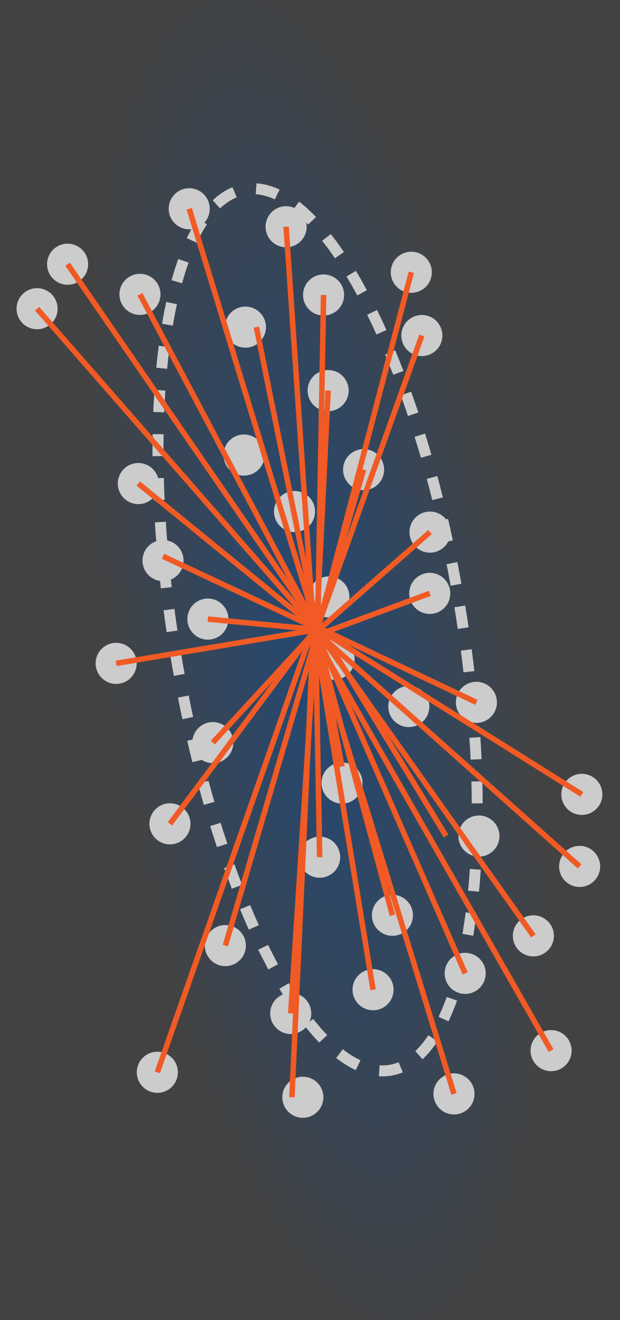
Each photon exerts a pull on nearby Gaussian components

Thursday, 6 September 12

Here's a simple illustration of expectation maximization:

On an intuitive level, one could think of think EM as a kind of evolving system, where each photon exerts a force onto nearby Gaussian components, pulling them towards it as to be covered by the density function. This happens until an equilibrium is reached. Here, you can see a single elliptical Gaussian which is fit to a group of photons.
[CLICK]
In Plain EM, we need to account for the interaction of every single photon with every Gaussian.
This is excessive, given the little information that an individual photon contains, and clearly won't work even for relative small-scale datasets.

Plain EM



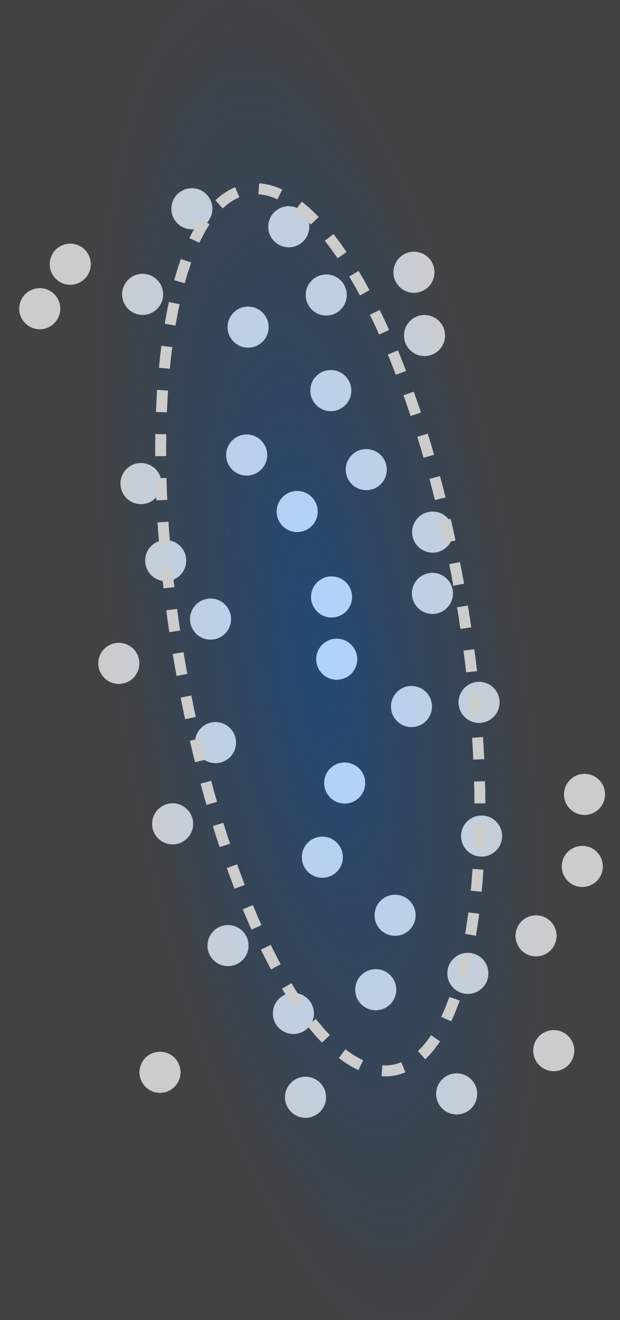
Each photon exerts a pull on nearby Gaussian components

Thursday, 6 September 12

Here’s a simple illustration of expectation maximization:

On an intuitive level, one could think of think EM as a kind of evolving system, where each photon exerts a force onto nearby Gaussian components, pulling them towards it as to be covered by the density function. This happens until an equilibrium is reached. Here, you can see a single elliptical Gaussian which is fit to a group of photons.
[CLICK]
In Plain EM, we need to account for the interaction of every single photon with every Gaussian.
This is excessive, given the little information that an individual photon contains, and clearly won’t work even for relative small-scale datasets.

Plain EM

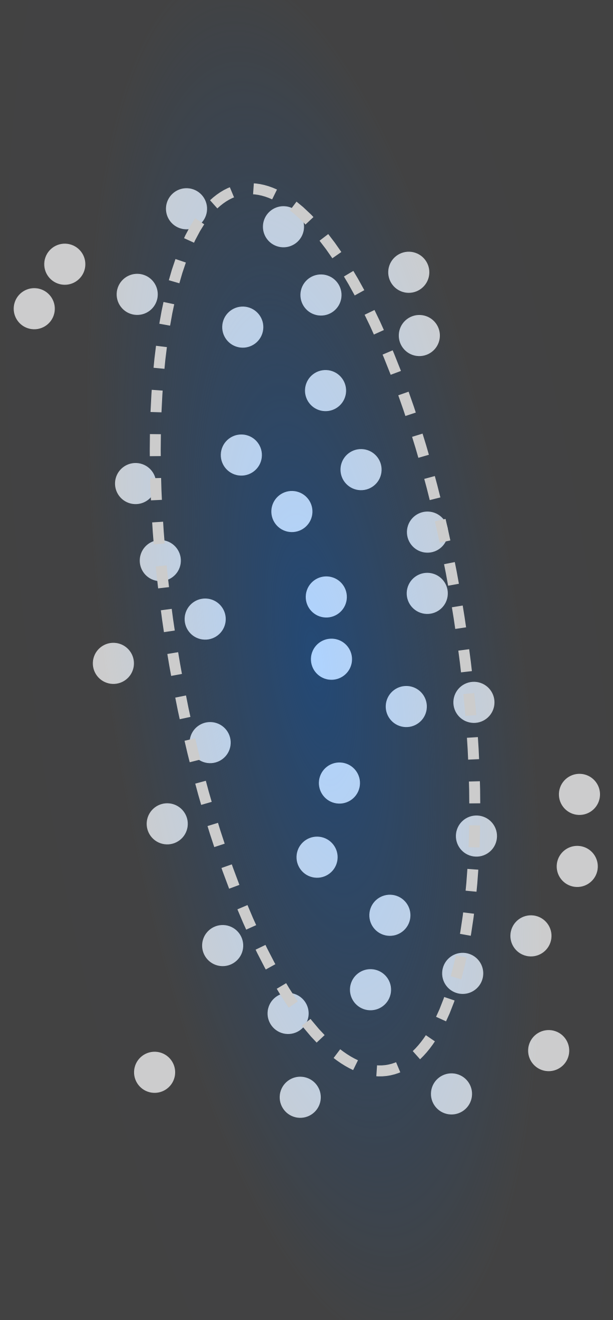


→ Accelerated EM by [Verbeek et al. 06]

Thursday, 6 September 12

To address the performance issue, we base our fitting stage on an algorithm from data mining known as accelerated EM, which was presented by Verbeek et al. in 2006. While this is much faster than plain EM, it was designed for the small GMMs used in this field, usually in the tens of components. To let it scale to the problems of our size, we made several modifications, that we'll describe shortly.

Accelerated EM



Thursday, 6 September 12

Accelerated EM works by creating cells [CLICK] surrounding the photons.

Each cell summarizes the photons that fall inside it using their count, mean, and average outer product of the positions with themselves -- these are essentially the 0th, 1st, and 2nd order statistics.

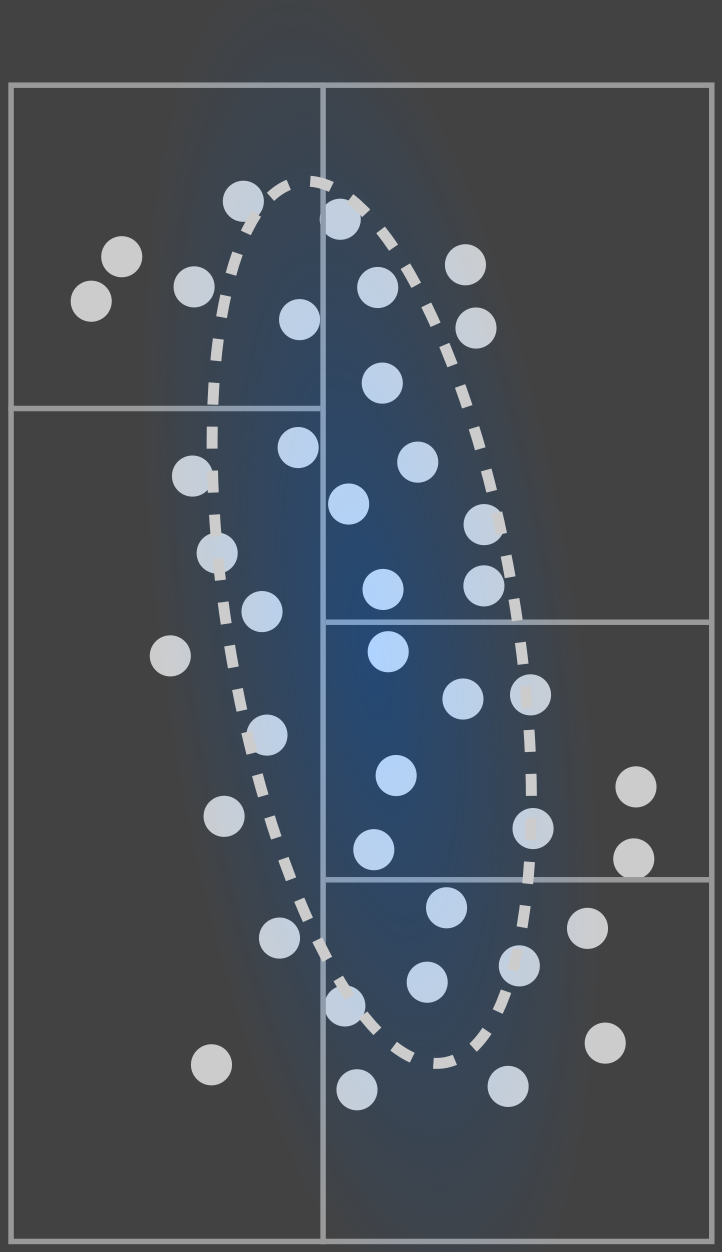
[CLICK]

Now, the entire EM algorithm can be formulated only in terms of cell-Gaussian interactions, and the original photons may be discarded. These statistics are quite expressive, hence we can use many fewer cells than photons, which leads to shorter running times. [PAUSE]

You may remember that earlier, we talked about the necessity of having an initial guess when running EM: the cells are also useful in this context, since we can trivially create a rough initial guess from them -- the details are in the paper.

Of course, one important thing to note is that the quality of the solution can be poor if the chosen cells are too course.

Accelerated EM



Stored cell statistics:

- photon count
- mean position
- average outer product

Initial Guess:

- obtained from cell statistics

Thursday, 6 September 12

Accelerated EM works by creating cells [CLICK] surrounding the photons.

Each cell summarizes the photons that fall inside it using their count, mean, and average outer product of the positions with themselves -- these are essentially the 0th, 1st, and 2nd order statistics.

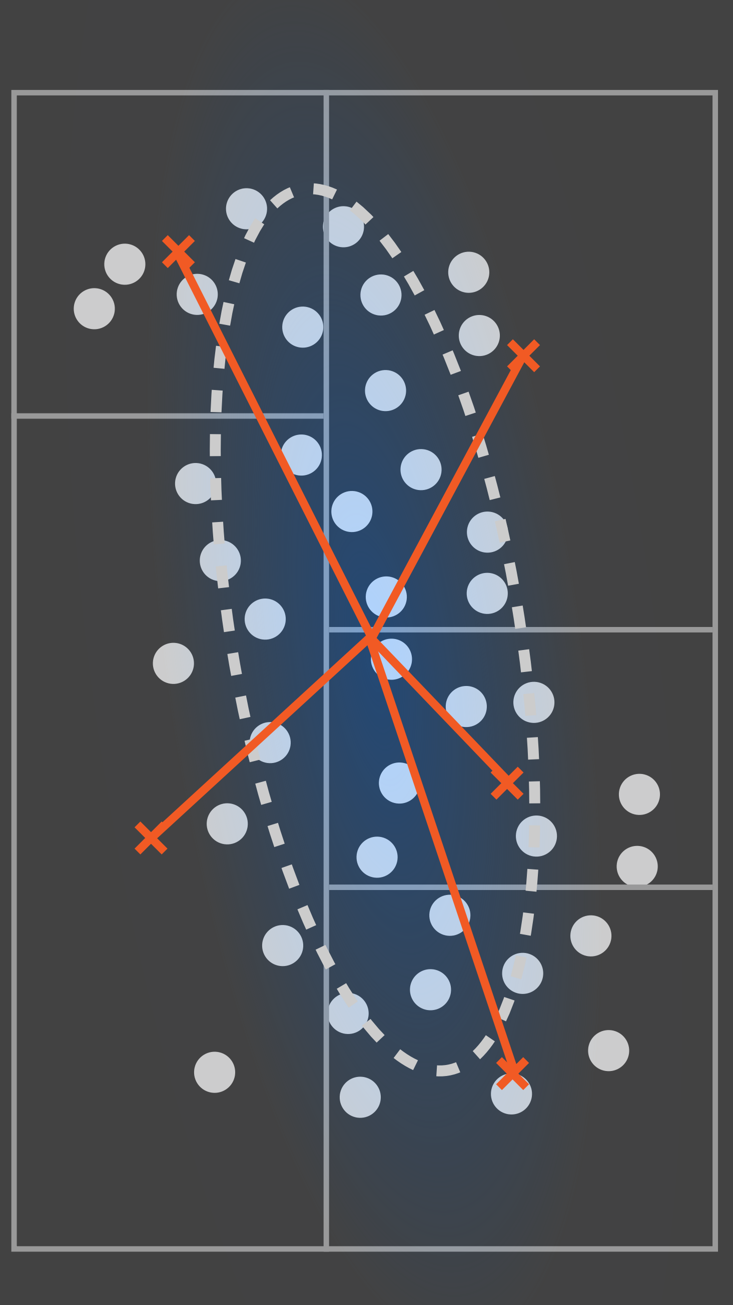
[CLICK]

Now, the entire EM algorithm can be formulated only in terms of cell-Gaussian interactions, and the original photons may be discarded. These statistics are quite expressive, hence we can use many fewer cells than photons, which leads to shorter running times. [PAUSE]

You may remember that earlier, we talked about the necessity of having an initial guess when running EM: the cells are also useful in this context, since we can trivially create a rough initial guess from them -- the details are in the paper.

Of course, one important thing to note is that the quality of the solution can be poor if the chosen cells are too course.

Accelerated EM



Stored cell statistics:

- photon count
- mean position
- average outer product

Initial Guess:

- obtained from cell statistics

Thursday, 6 September 12

Accelerated EM works by creating cells [CLICK] surrounding the photons.

Each cell summarizes the photons that fall inside it using their count, mean, and average outer product of the positions with themselves -- these are essentially the 0th, 1st, and 2nd order statistics.

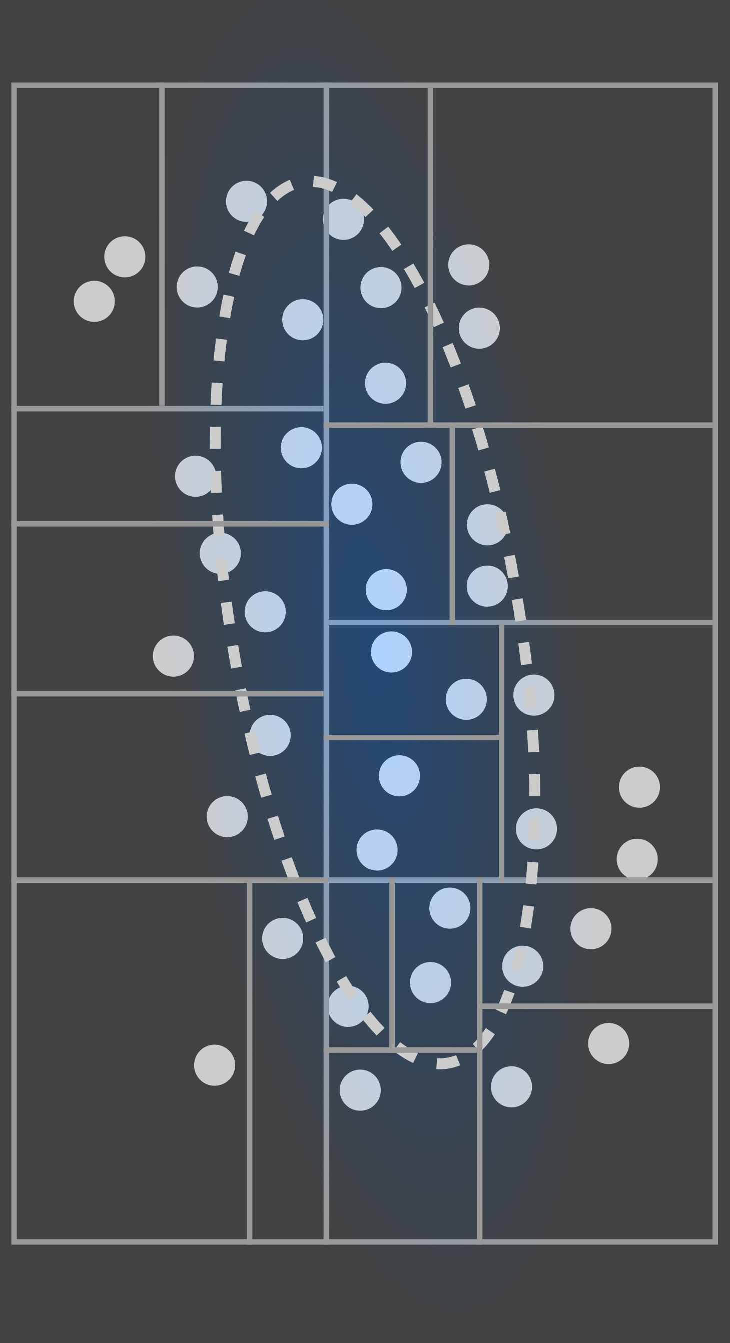
[CLICK]

Now, the entire EM algorithm can be formulated only in terms of cell-Gaussian interactions, and the original photons may be discarded. These statistics are quite expressive, hence we can use many fewer cells than photons, which leads to shorter running times. [PAUSE]

You may remember that earlier, we talked about the necessity of having an initial guess when running EM: the cells are also useful in this context, since we can trivially create a rough initial guess from them -- the details are in the paper.

Of course, one important thing to note is that the quality of the solution can be poor if the chosen cells are too course.

Accelerated EM



Stored cell statistics:

- photon count
- mean position
- average outer product

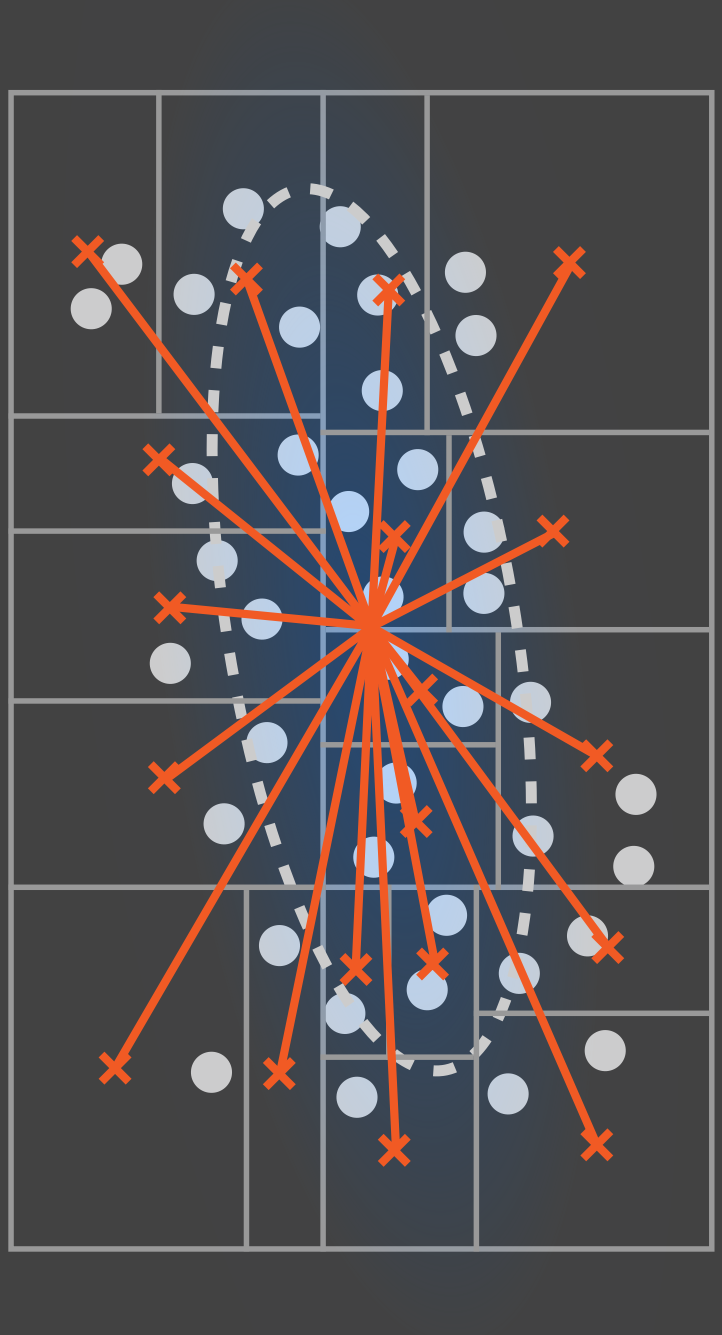
Initial Guess:

- obtained from cell statistics

The solution that accelerated EM offers is reminiscent to multi-grid methods. We can just switch to increasingly finer cells partitions [CLICK] as the algorithm runs.

One of the modifications we made at this point [CLICK] is to improve this refinement criterion so that it works with massive datasets.

Accelerated EM



Stored cell statistics:

- photon count
- mean position
- average outer product

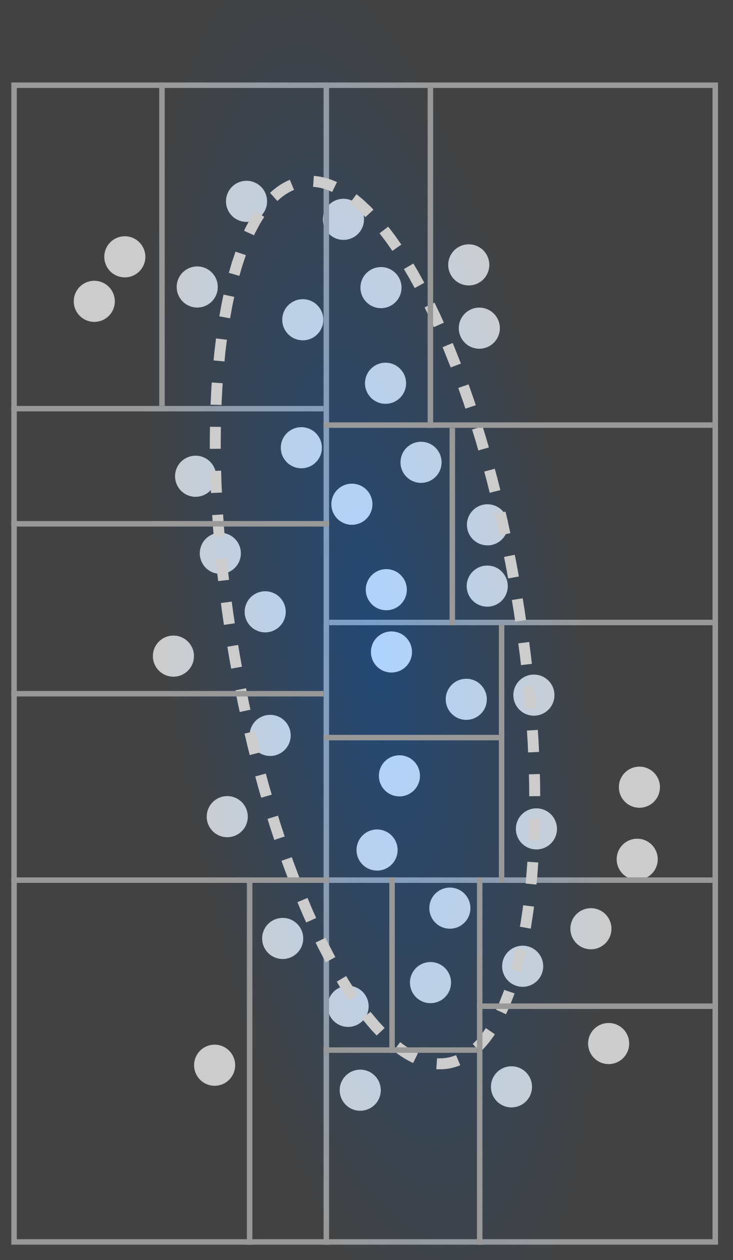
Initial Guess:

- obtained from cell statistics

The solution that accelerated EM offers is reminiscent to multi-grid methods. We can just switch to increasingly finer cells partitions [CLICK] as the algorithm runs.

One of the modifications we made at this point [CLICK] is to improve this refinement criterion so that it works with massive datasets.

Progressive EM



Stored cell statistics:

- photon count
- mean position
- average outer product

Initial Guess:

- obtained from cell statistics

Our modifications:

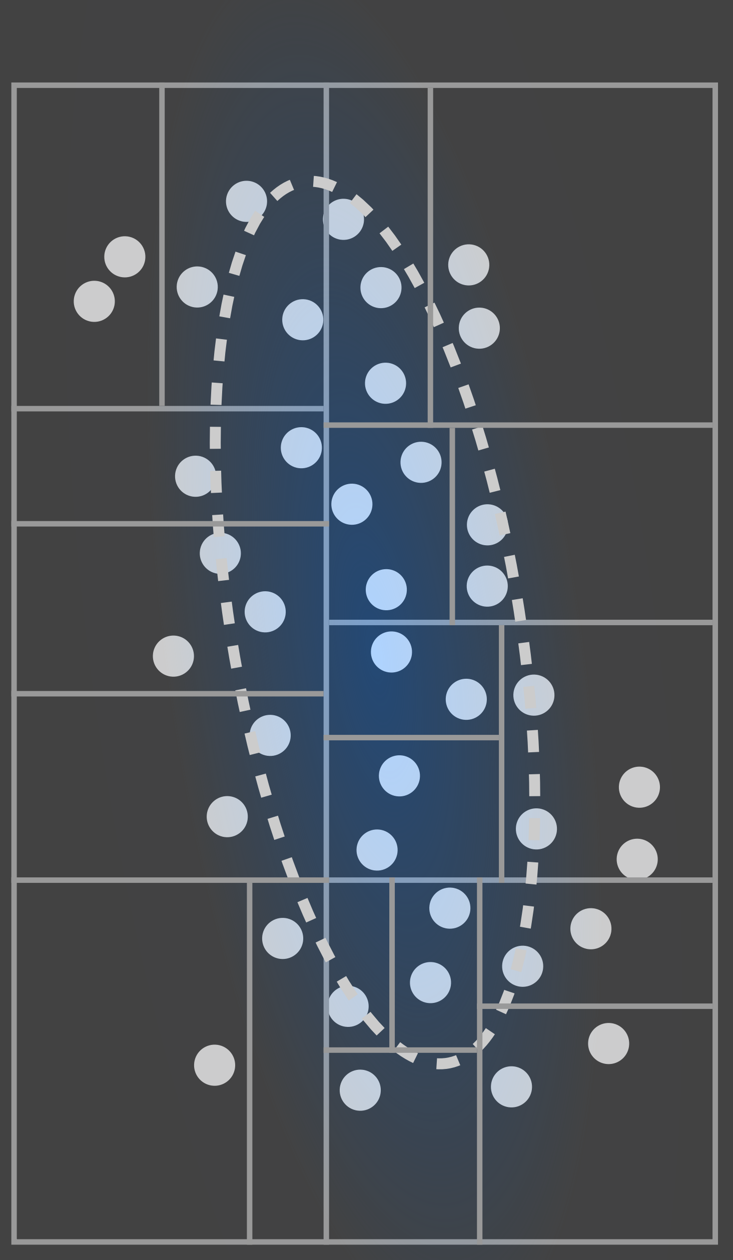
- better cell refinement

Thursday, 6 September 12

The solution that accelerated EM offers is reminiscent to multi-grid methods. We can just switch to increasingly finer cells partitions [CLICK] as the algorithm runs.

One of the modifications we made at this point [CLICK] is to improve this refinement criterion so that it works with massive datasets.

Progressive EM



Stored cell statistics:

- photon count
- mean position
- average outer product

Initial Guess:

- obtained from cell statistics

Our modifications:

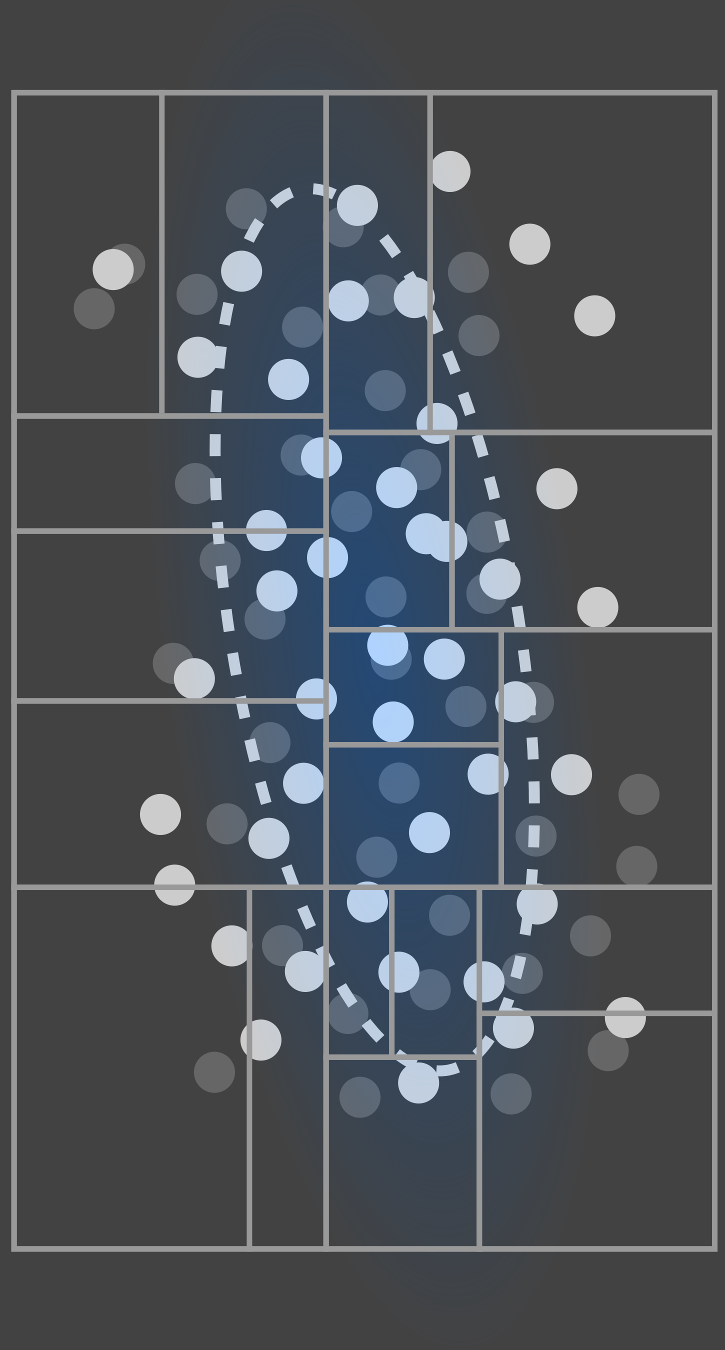
- better cell refinement
- progressive photons shooting passes

Thursday, 6 September 12

Another modification that we made is based on the observation that after storing photon statistics in cells, the photons themselves are not needed anymore and can be discarded. This helps the method maintain a low memory footprint, but it also means that we can afford to shoot further photons later on in a spirit similar to progressive photon mapping and online algorithms that accept streaming data [CLICK]. These additional photons can then be incorporated into the already existing statistics, and afterwards they are thrown away.

[CLICK] We use this approach in the paper, since it is computationally cheap and causes the statistics to become increasingly accurate over time, which in turn leads to a higher-quality solution.

Progressive EM



Stored cell statistics:

- photon count
- mean position
- average outer product

Initial Guess:

- obtained from cell statistics

Our modifications:

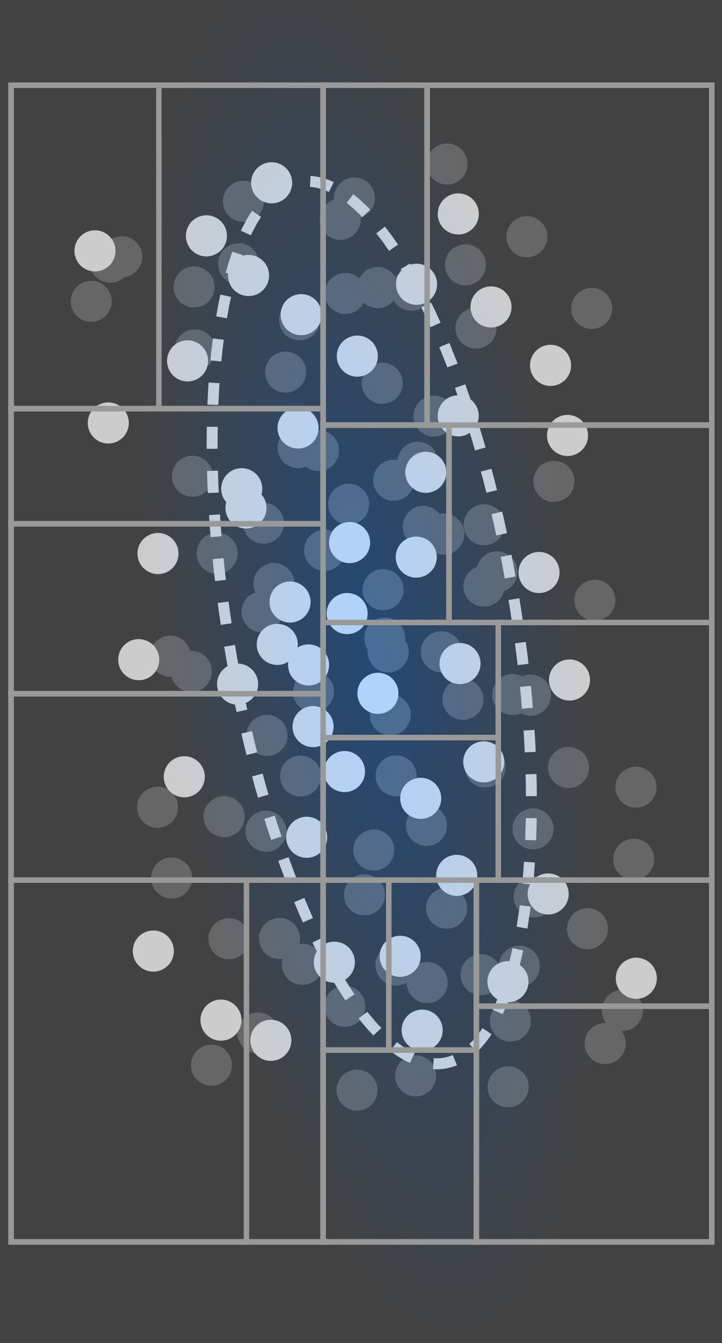
- better cell refinement
- progressive photons shooting passes

Thursday, 6 September 12

Another modification that we made is based on the observation that after storing photon statistics in cells, the photons themselves are not needed anymore and can be discarded. This helps the method maintain a low memory footprint, but it also means that we can afford to shoot further photons later on in a spirit similar to progressive photon mapping and online algorithms that accept streaming data [CLICK]. These additional photons can then be incorporated into the already existing statistics, and afterwards they are thrown away.

[CLICK] We use this approach in the paper, since it is computationally cheap and causes the statistics to become increasingly accurate over time, which in turn leads to a higher-quality solution.

Progressive EM



Stored cell statistics:

- photon count
- mean position
- average outer product

Initial Guess:

- obtained from cell statistics

Our modifications:

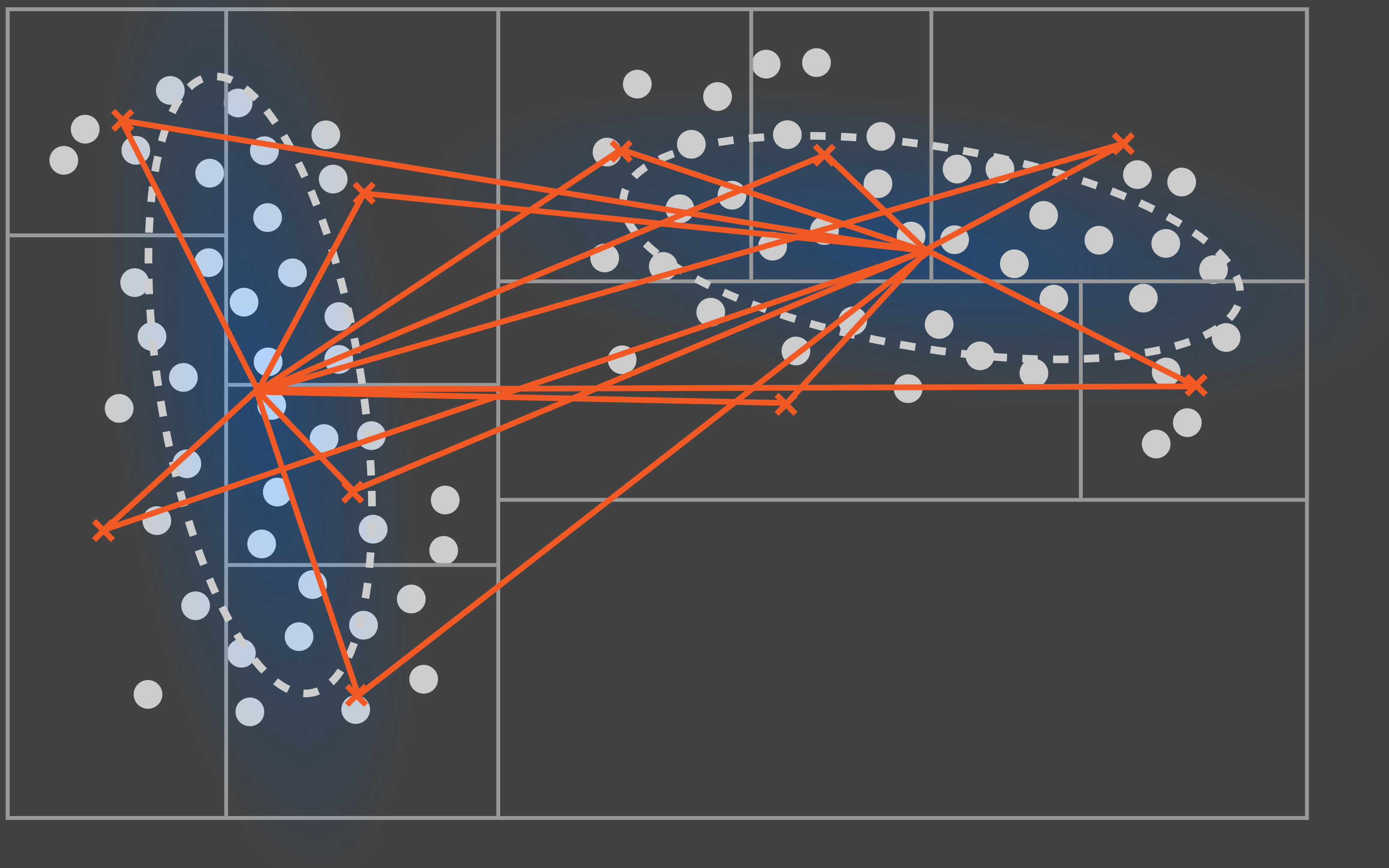
- better cell refinement
- progressive photons shooting passes

Thursday, 6 September 12

Another modification that we made is based on the observation that after storing photon statistics in cells, the photons themselves are not needed anymore and can be discarded. This helps the method maintain a low memory footprint, but it also means that we can afford to shoot further photons later on in a spirit similar to progressive photon mapping and online algorithms that accept streaming data [CLICK]. These additional photons can then be incorporated into the already existing statistics, and afterwards they are thrown away.

[CLICK] We use this approach in the paper, since it is computationally cheap and causes the statistics to become increasingly accurate over time, which in turn leads to a higher-quality solution.

Progressive EM



Stored cell statistics:

- photon count
- mean position
- average outer product

Initial Guess:

- obtained from cell statistics

Our modifications:

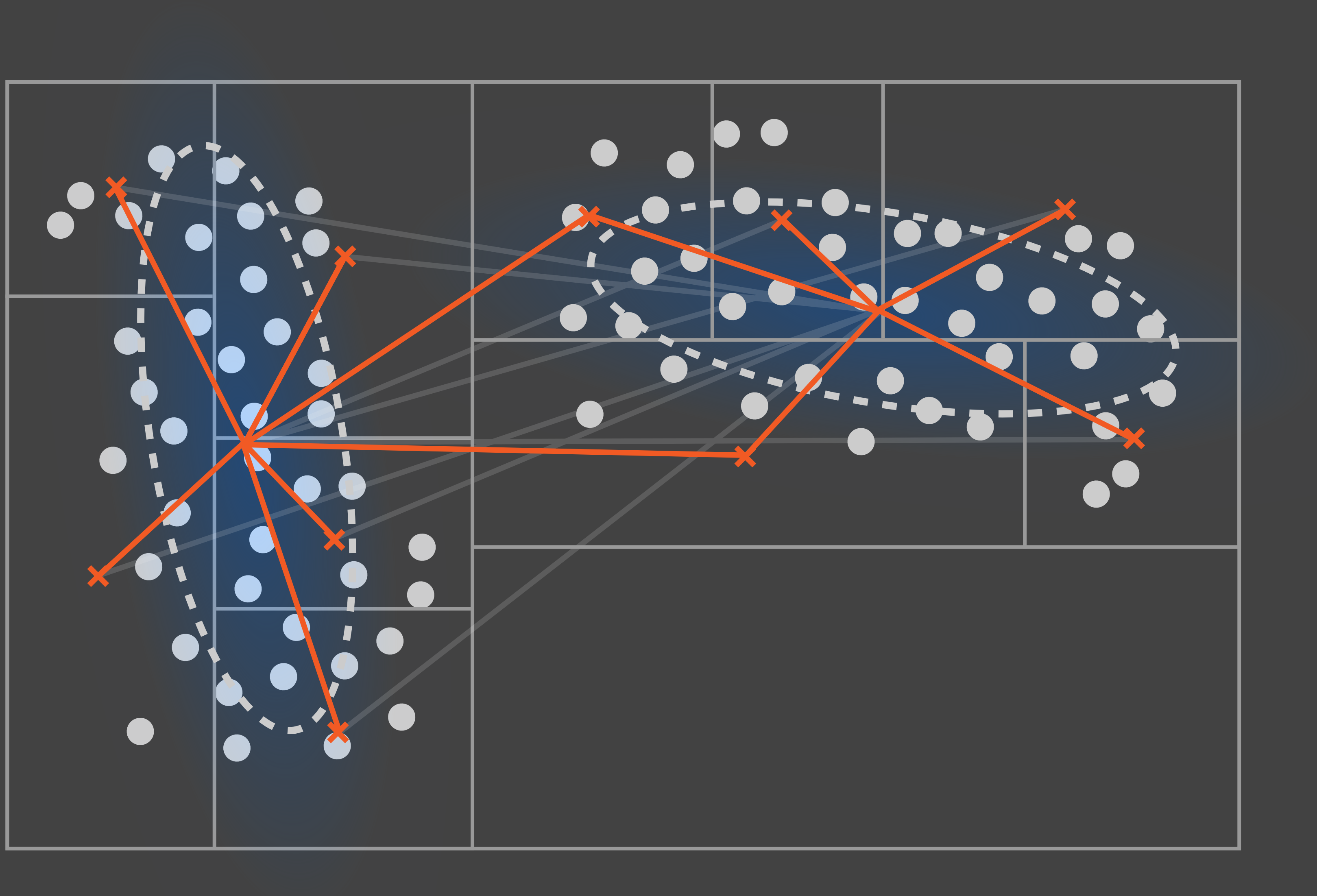
- better cell refinement
- progressive photons shooting passes
- reduced complexity $\mathcal{O}(n^2) \rightarrow \mathcal{O}(n \log n)$

Thursday, 6 September 12

And finally, thanks to the exponential decay in Gaussians, we can cull away interactions [CLICK] between distant Gaussians and cells, which improves the running time of the algorithm from O(n^2) to O(n log n). For more details, please refer to the paper.

We call the algorithm resulting after these modifications “Progressive EM”.

Progressive EM



Stored cell statistics:

- photon count
- mean position
- average outer product

Initial Guess:

- obtained from cell statistics

Our modifications:

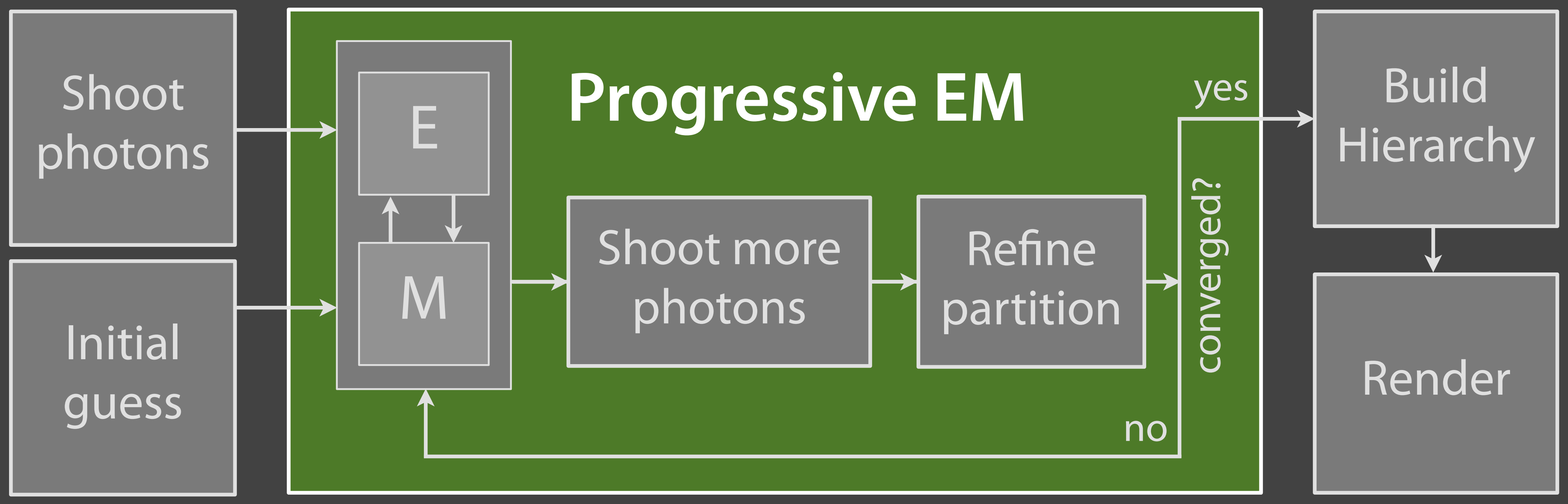
- better cell refinement
- progressive photons shooting passes
- reduced complexity $\mathcal{O}(n^2) \to \mathcal{O}(n \log n)$

Thursday, 6 September 12

And finally, thanks to the exponential decay in Gaussians, we can cull away interactions [CLICK] between distant Gaussians and cells, which improves the running time of the algorithm from O(n^2) to O(n log n). For more details, please refer to the paper.

We call the algorithm resulting after these modifications “Progressive EM”.

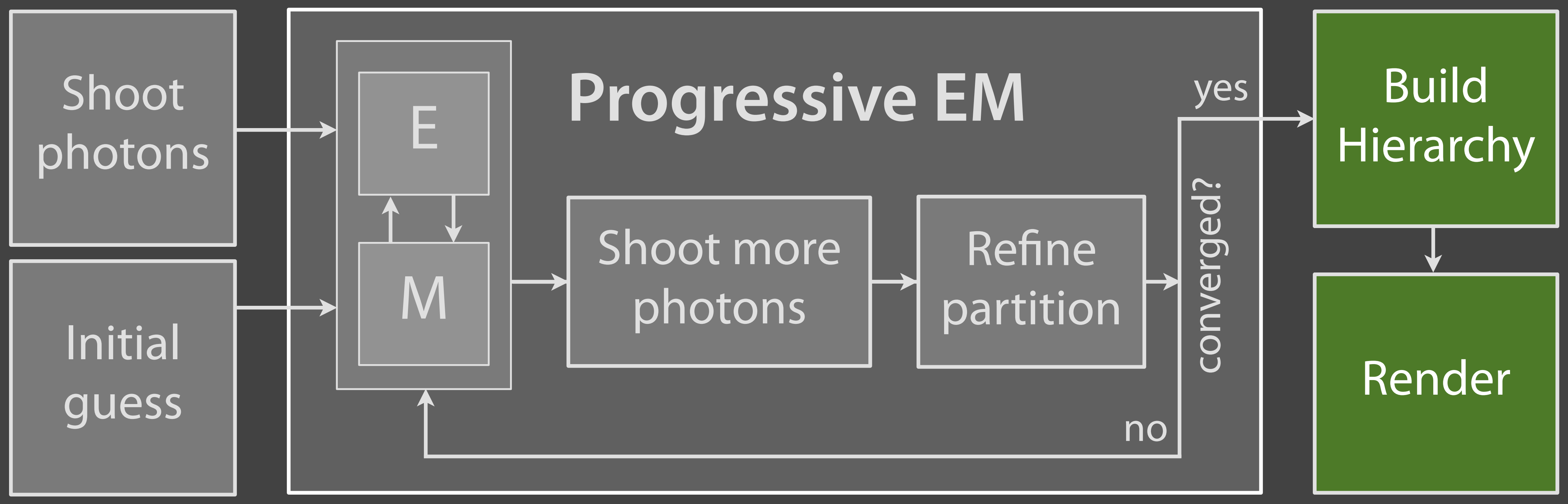
Pipeline overview



Thursday, 6 September 12

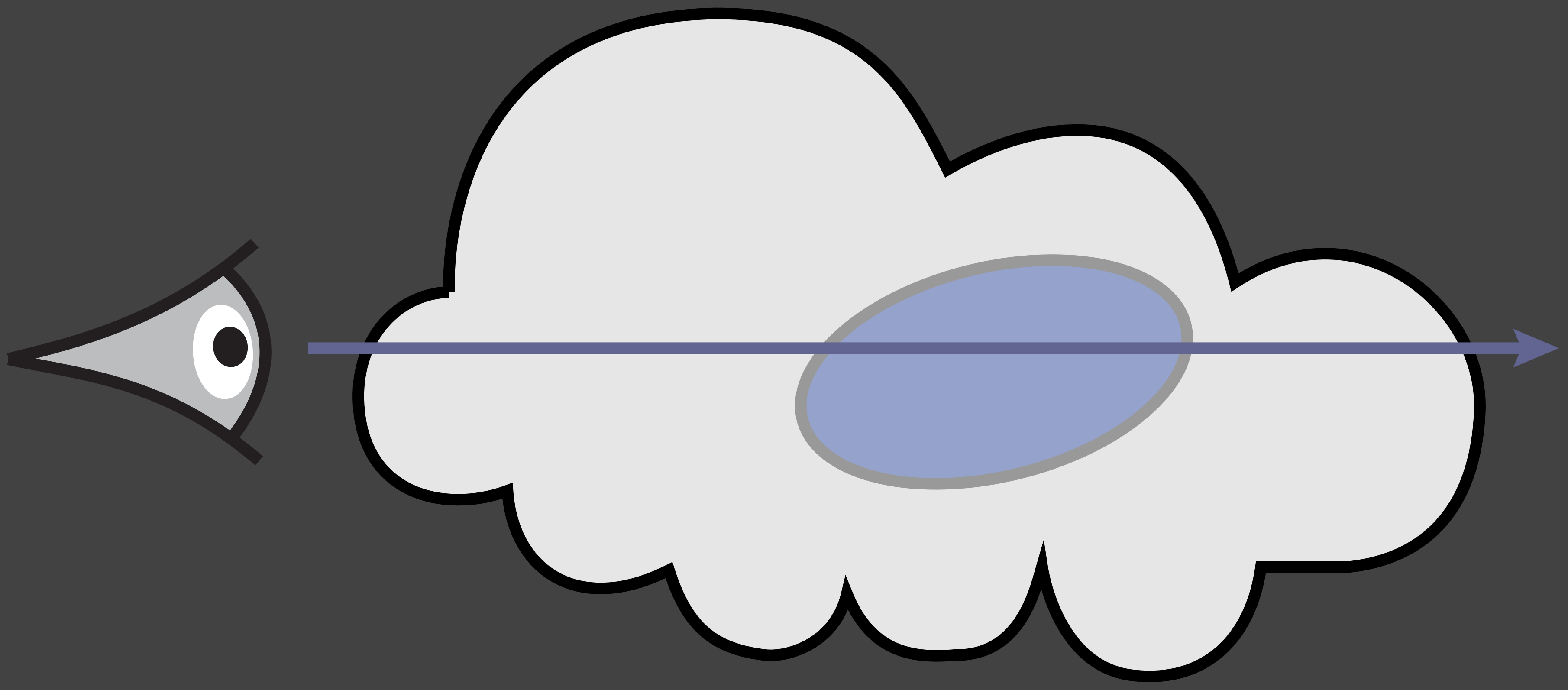
Progressive EM outputs a fully converged mixture model to the next stage. We'll now focus on how to render it.

Pipeline overview



Progressive EM outputs a fully converged mixture model to the next stage. We'll now focus on how to render it.

Rendering



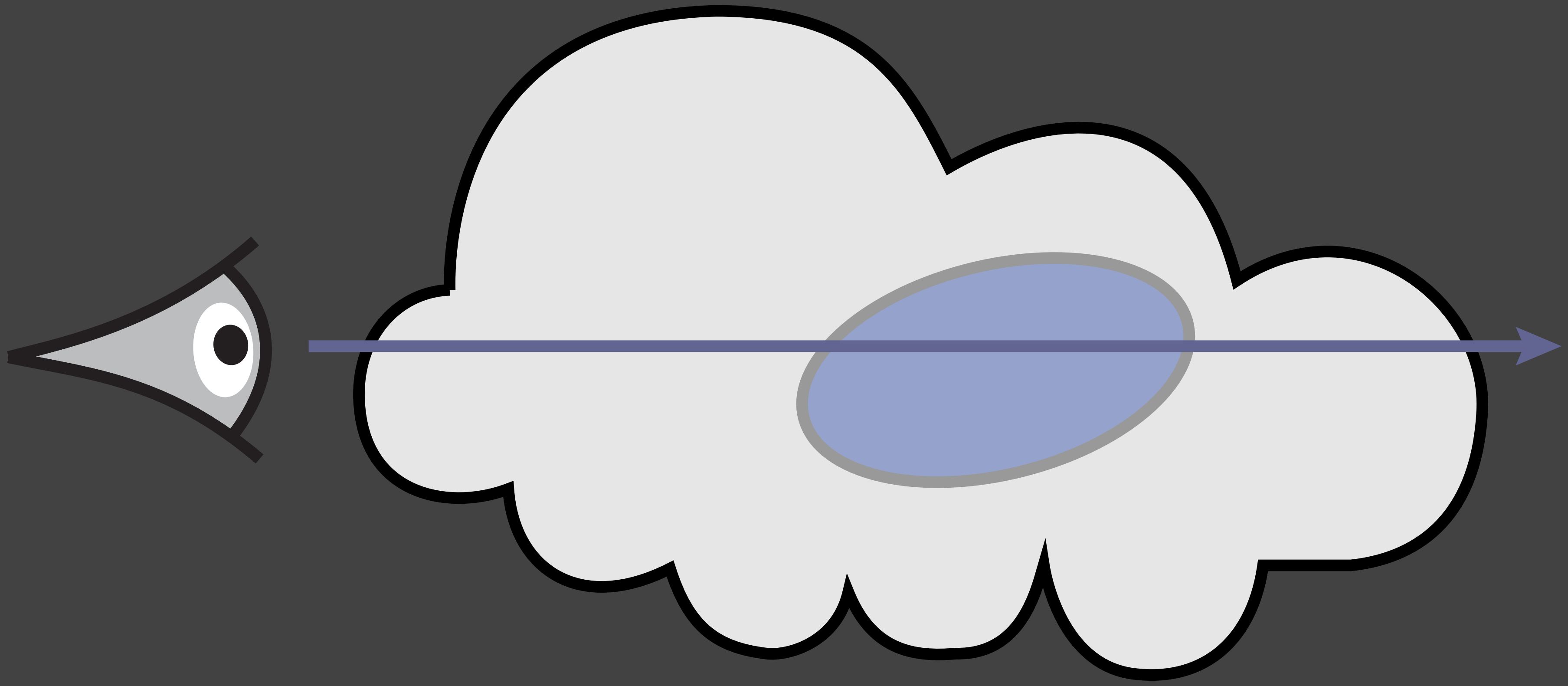
$$\text{pixel value} = \sum_{i=1}^{k} \text{contrib}(i)$$

Thursday, 6 September 12

In theory, we can already use this converged GMM in a brute-force rendering algorithm, which sums over the contributions of every single Gaussian to an eye ray. For homogeneous media, we have derived an analytic solution [CLICK] that gives the exact result in this case. For inhomogeneous media, it will be necessary to use numerical integration.

The problem with this brute-force summation is that when k is large, each pixel takes a long time to render.

Rendering



$$\text{pixel value} = \sum_{i=1}^{k} \text{contrib}(i)$$

$$\text{contrib}(i) = \int_a^b g(\mathbf{r}(t)|\bar{\Theta}_i)\, e^{-\sigma_t\, t} dt = C_0 \left[\text{erf}\left(\frac{C_3 + 2C_2 b}{2\sqrt{C_2}}\right) - \text{erf}\left(\frac{C_3 + 2C_2 a}{2\sqrt{C_2}}\right) \right]$$

...

Thursday, 6 September 12

In theory, we can already use this converged GMM in a brute-force rendering algorithm, which sums over the contributions of every single Gaussian to an eye ray. For homogeneous media, we have derived an analytic solution [CLICK] that gives the exact result in this case. For inhomogeneous media, it will be necessary to use numerical integration.

The problem with this brute-force summation is that when k is large, each pixel takes a long time to render.

Hierarchy construction

Goals:

- spatial acceleration data structure
- level-of-detail representation

To get better performance, we instead turn the GMM into a hierarchical representation. This serves two purposes: first of all, the hierarchy will serve as a spatial data structure to accelerate the rendering process.

And secondly, we will store intermediate levels of detail on the inner nodes of the hierarchy, which are used as a fast fall-back when the accuracy requirements are low enough. [CLICK]

The hierarchy construction process begins with the input list of Gaussian terms. We then find pairs that have a small symmetric Kullback-Leibler divergence and collapse them. [CLICK] Kullback-Leibler divergence is a popular information-theoretic distance measure between statistical distributions -- in our context, it is used to check how similar two Gaussians are. Repeating this process log n times finally creates a full binary tree, where each inner node contains a Gaussian that is a good approximation of the entire subtree rooted at its position.

Hierarchy construction

Goals:

- spatial acceleration data structure
- level-of-detail representation

Agglomerative construction:

- Repeatedly merge nearby Gaussians based on their Kullback-Leibler divergence

Thursday, 6 September 12

To get better performance, we instead turn the GMM into a hierarchical representation. This serves two purposes: first of all, the hierarchy will serve as a spatial data structure to accelerate the rendering process.

And secondly, we will store intermediate levels of detail on the inner nodes of the hierarchy, which are used as a fast fall-back when the accuracy requirements are low enough. [CLICK]

The hierarchy construction process begins with the input list of Gaussian terms. We then find pairs that have a small symmetric Kullback-Leibler divergence and collapse them. [CLICK] Kullback-Leibler divergence is a popular information-theoretic distance measure between statistical distributions -- in our context, it is used to check how similar two Gaussians are. Repeating this process log n times finally creates a full binary tree, where each inner node contains a Gaussian that is a good approximation of the entire subtree rooted at its position.

Hierarchy construction

Goals:

- spatial acceleration data structure
- level-of-detail representation

Agglomerative construction:

- Repeatedly merge nearby Gaussians based on their Kullback-Leibler divergence

To get better performance, we instead turn the GMM into a hierarchical representation. This serves two purposes: first of all, the hierarchy will serve as a spatial data structure to accelerate the rendering process.

And secondly, we will store intermediate levels of detail on the inner nodes of the hierarchy, which are used as a fast fall-back when the accuracy requirements are low enough. [CLICK]

The hierarchy construction process begins with the input list of Gaussian terms. We then find pairs that have a small symmetric Kullback-Leibler divergence and collapse them. [CLICK] Kullback-Leibler divergence is a popular information-theoretic distance measure between statistical distributions -- in our context, it is used to check how similar two Gaussians are. Repeating this process log n times finally creates a full binary tree, where each inner node contains a Gaussian that is a good approximation of the entire subtree rooted at its position.

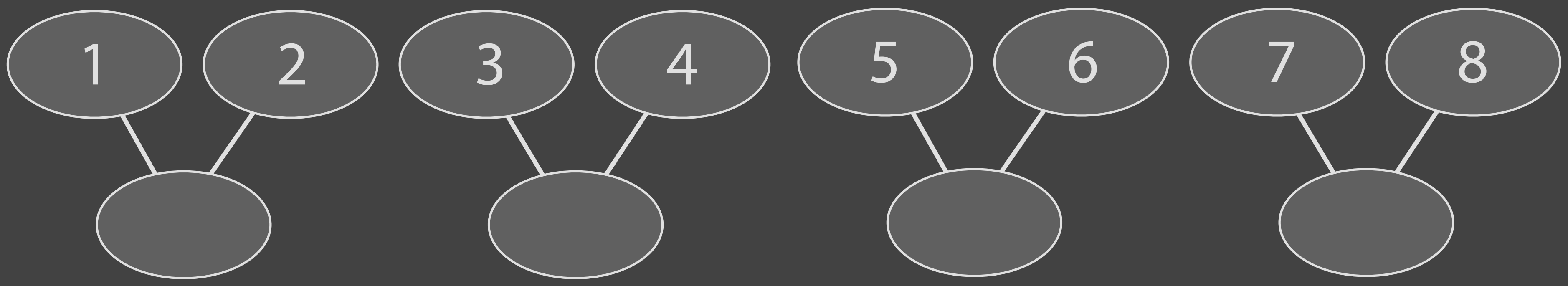
Hierarchy construction

Goals:

- spatial acceleration data structure
- level-of-detail representation

Agglomerative construction:

- Repeatedly merge nearby Gaussians based on their Kullback-Leibler divergence



To get better performance, we instead turn the GMM into a hierarchical representation. This serves two purposes: first of all, the hierarchy will serve as a spatial data structure to accelerate the rendering process.

And secondly, we will store intermediate levels of detail on the inner nodes of the hierarchy, which are used as a fast fall-back when the accuracy requirements are low enough. [CLICK]

The hierarchy construction process begins with the input list of Gaussian terms. We then find pairs that have a small symmetric Kullback-Leibler divergence and collapse them. [CLICK] Kullback-Leibler divergence is a popular information-theoretic distance measure between statistical distributions -- in our context, it is used to check how similar two Gaussians are. Repeating this process log n times finally creates a full binary tree, where each inner node contains a Gaussian that is a good approximation of the entire subtree rooted at its position.

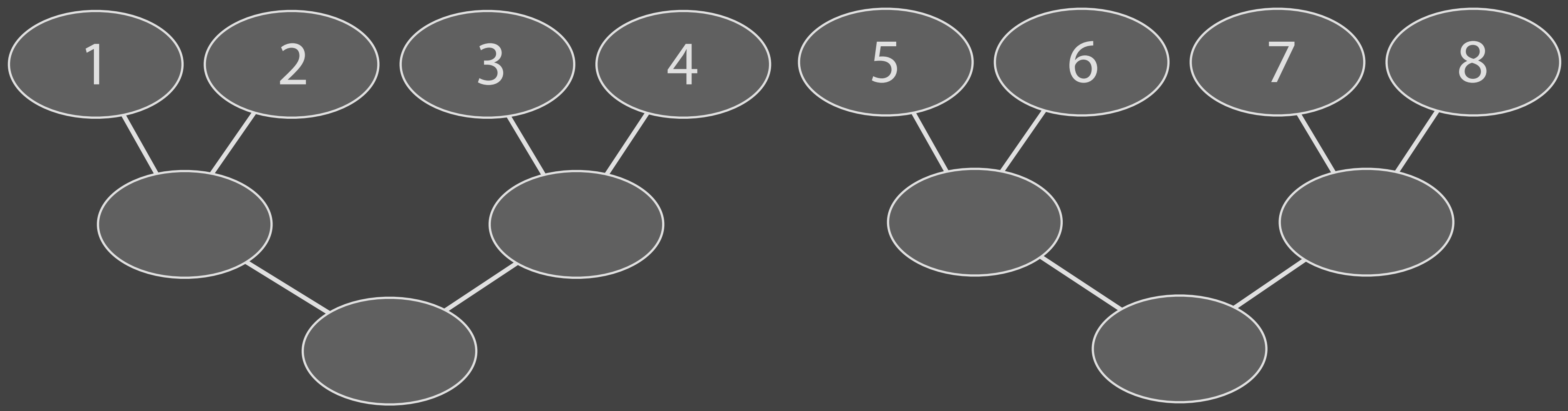
Hierarchy construction

Goals:

- spatial acceleration data structure
- level-of-detail representation

Agglomerative construction:

- Repeatedly merge nearby Gaussians based on their Kullback-Leibler divergence



Thursday, 6 September 12

To get better performance, we instead turn the GMM into a hierarchical representation. This serves two purposes: first of all, the hierarchy will serve as a spatial data structure to accelerate the rendering process.

And secondly, we will store intermediate levels of detail on the inner nodes of the hierarchy, which are used as a fast fall-back when the accuracy requirements are low enough. [CLICK]

The hierarchy construction process begins with the input list of Gaussian terms. We then find pairs that have a small symmetric Kullback-Leibler divergence and collapse them. [CLICK] Kullback-Leibler divergence is a popular information-theoretic distance measure between statistical distributions -- in our context, it is used to check how similar two Gaussians are. Repeating this process log n times finally creates a full binary tree, where each inner node contains a Gaussian that is a good approximation of the entire subtree rooted at its position.

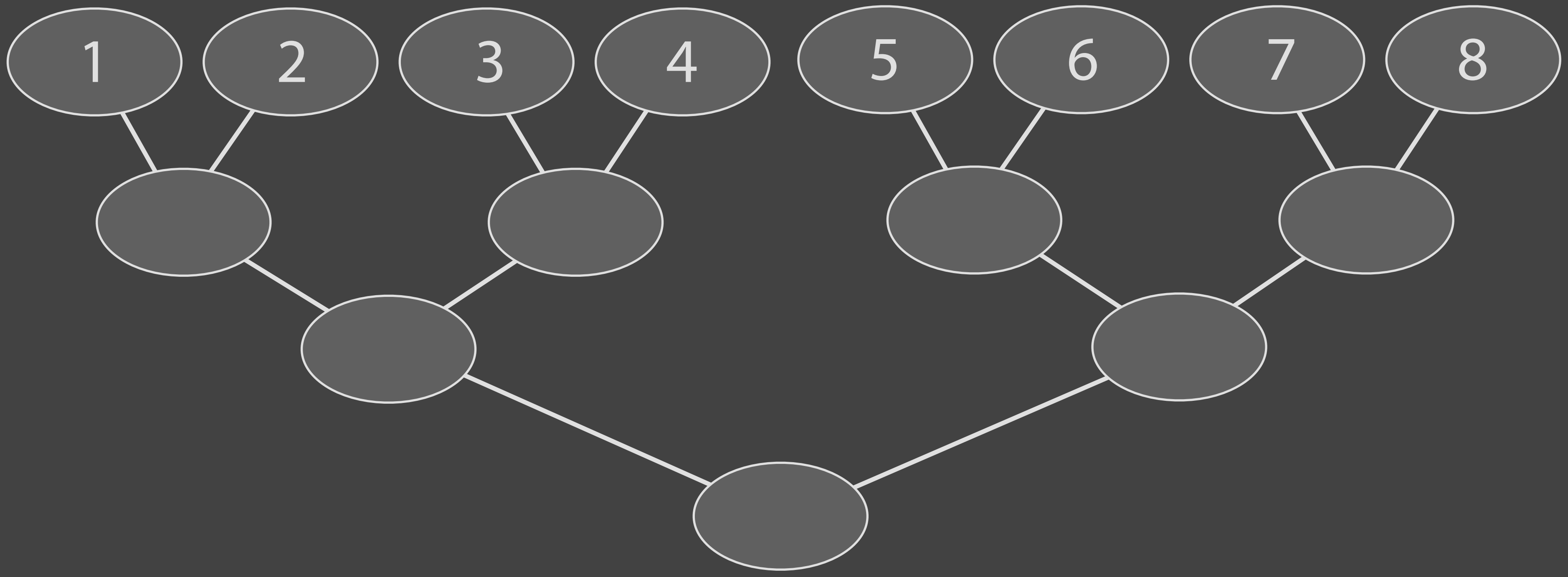
Hierarchy construction

Goals:

- spatial acceleration data structure
- level-of-detail representation

Agglomerative construction:

- Repeatedly merge nearby Gaussians based on their Kullback-Leibler divergence



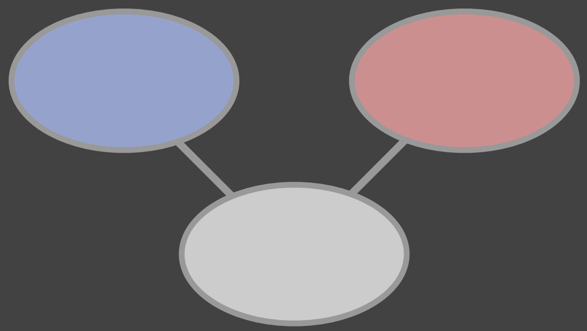
To get better performance, we instead turn the GMM into a hierarchical representation. This serves two purposes: first of all, the hierarchy will serve as a spatial data structure to accelerate the rendering process.

And secondly, we will store intermediate levels of detail on the inner nodes of the hierarchy, which are used as a fast fall-back when the accuracy requirements are low enough. [CLICK]

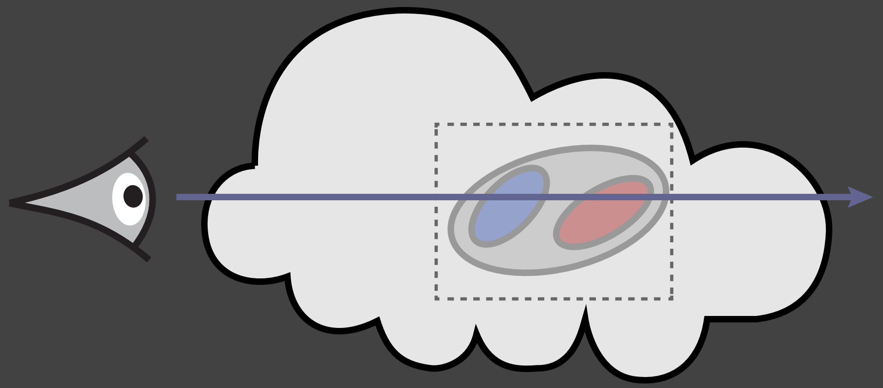
The hierarchy construction process begins with the input list of Gaussian terms. We then find pairs that have a small symmetric Kullback-Leibler divergence and collapse them. [CLICK] Kullback-Leibler divergence is a popular information-theoretic distance measure between statistical distributions -- in our context, it is used to check how similar two Gaussians are. Repeating this process log n times finally creates a full binary tree, where each inner node contains a Gaussian that is a good approximation of the entire subtree rooted at its position.

Rendering

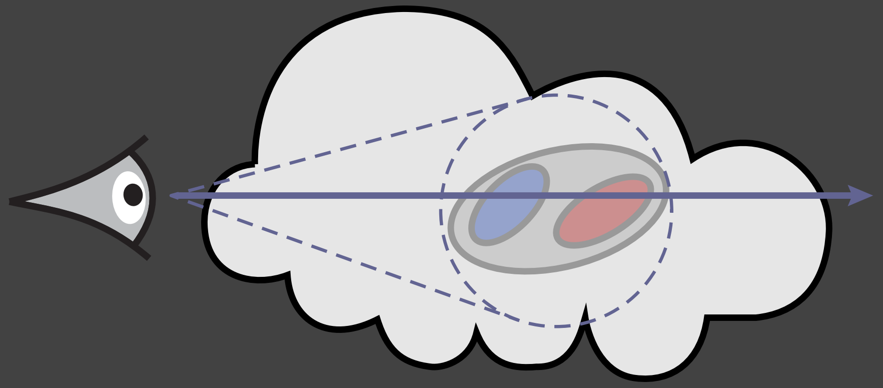
Example hierarchy:



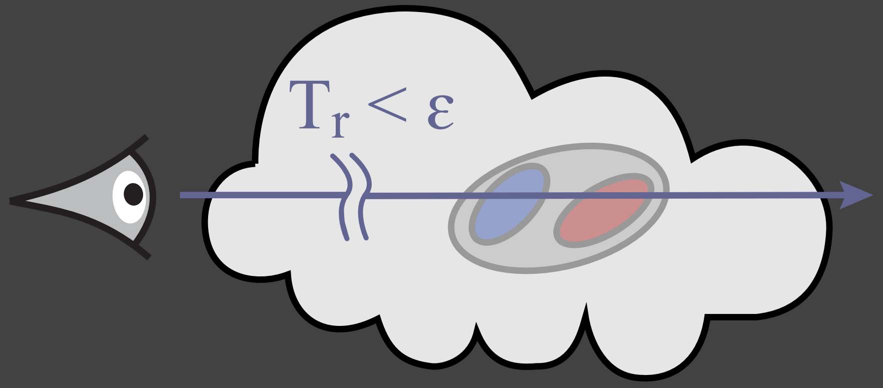
Criterion 1: bounding box intersected?



Criterion 2: solid angle large enough?



Criterion 3: attenuation low enough?



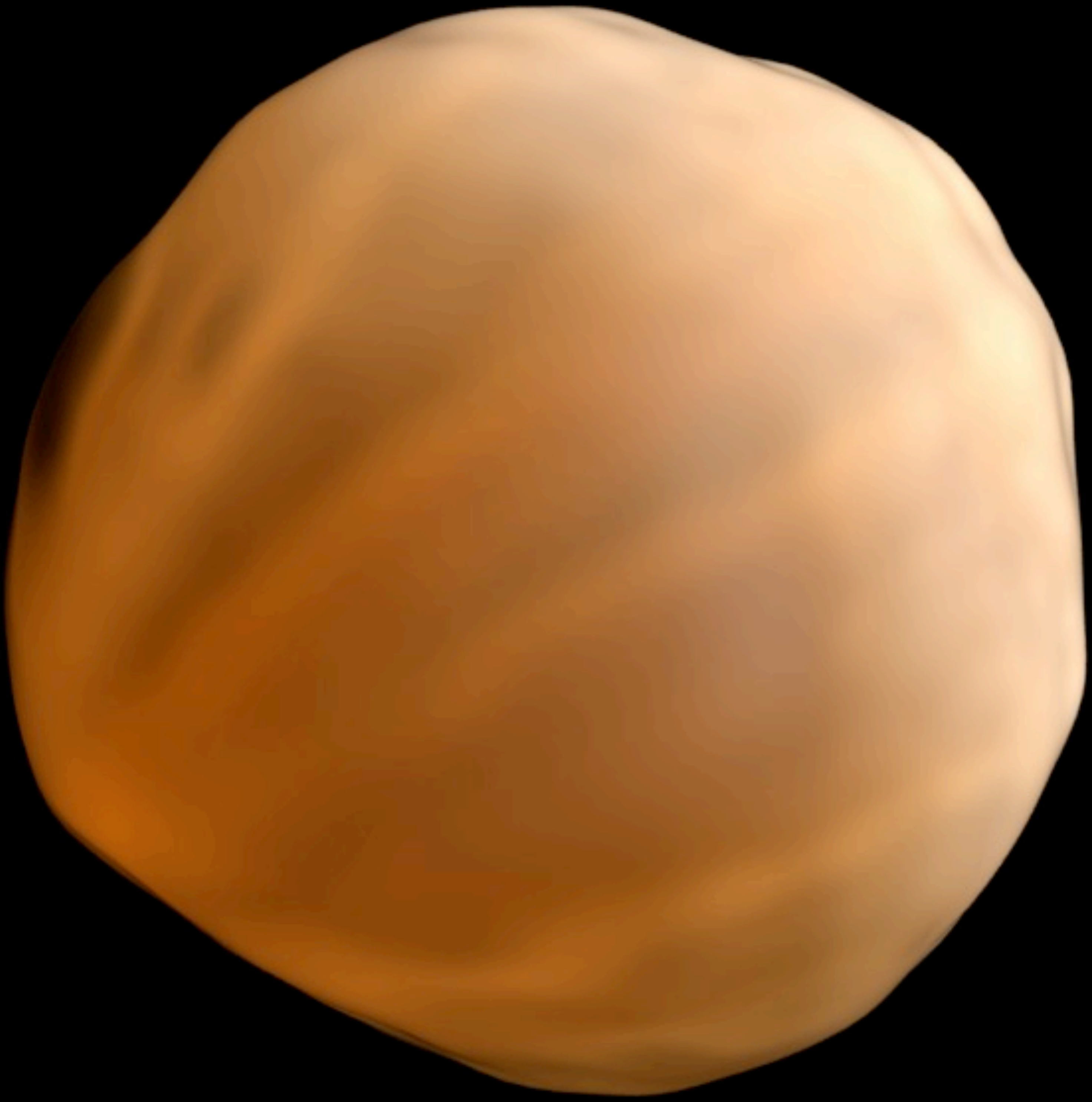
Thursday, 6 September 12

In the rendering step, we traverse this hierarchical Gaussian representation to compute the incident radiance along an eye ray. The goal is to always use the coarsest possible resolution, since we gain speed by using lower levels of the tree.

Here is a simple example of two Gaussians highlighted in blue and red, as well as a coarse approximation of the two in gray. We skip traversal of a subtree if the eye ray does not intersect a conservatively chosen bounding box around the subtree.

And when a subtree is smaller than a single pixel, or if there is a significant amount of attenuation, we use the coarse representative Gaussian as an approximation to the subtree.

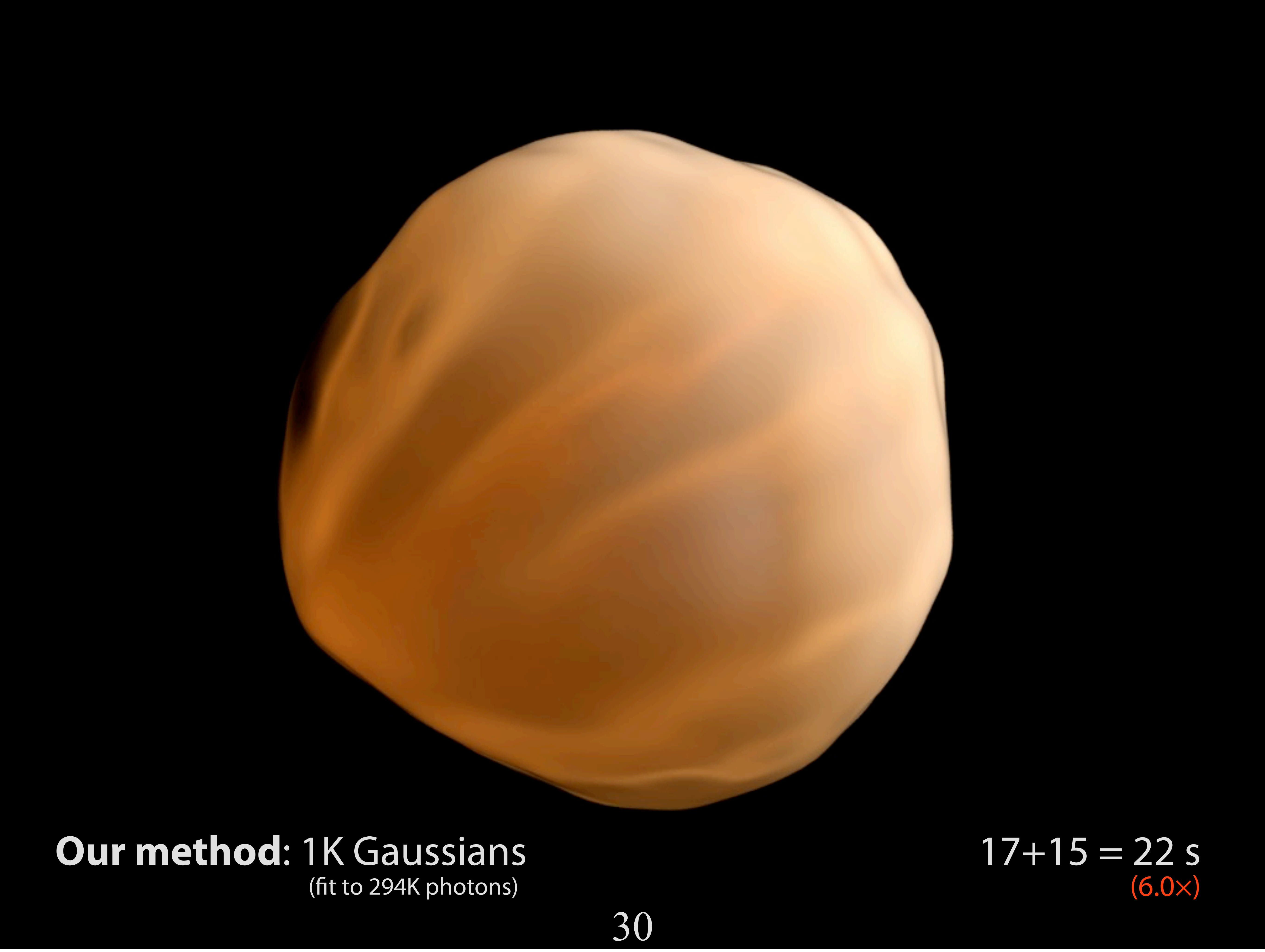
Let’s now take a look at some results we obtained.



BRE: 294K Photons 6+126 = 132 s

This is a rendering of the bumpy sphere scene courtesy of Bruce Walter, rendered using the beam radiance estimate. It uses 294K photons, which is why the features are still relatively blurred out.

The shown times denote the time spent on preprocessing and the actual rendering.



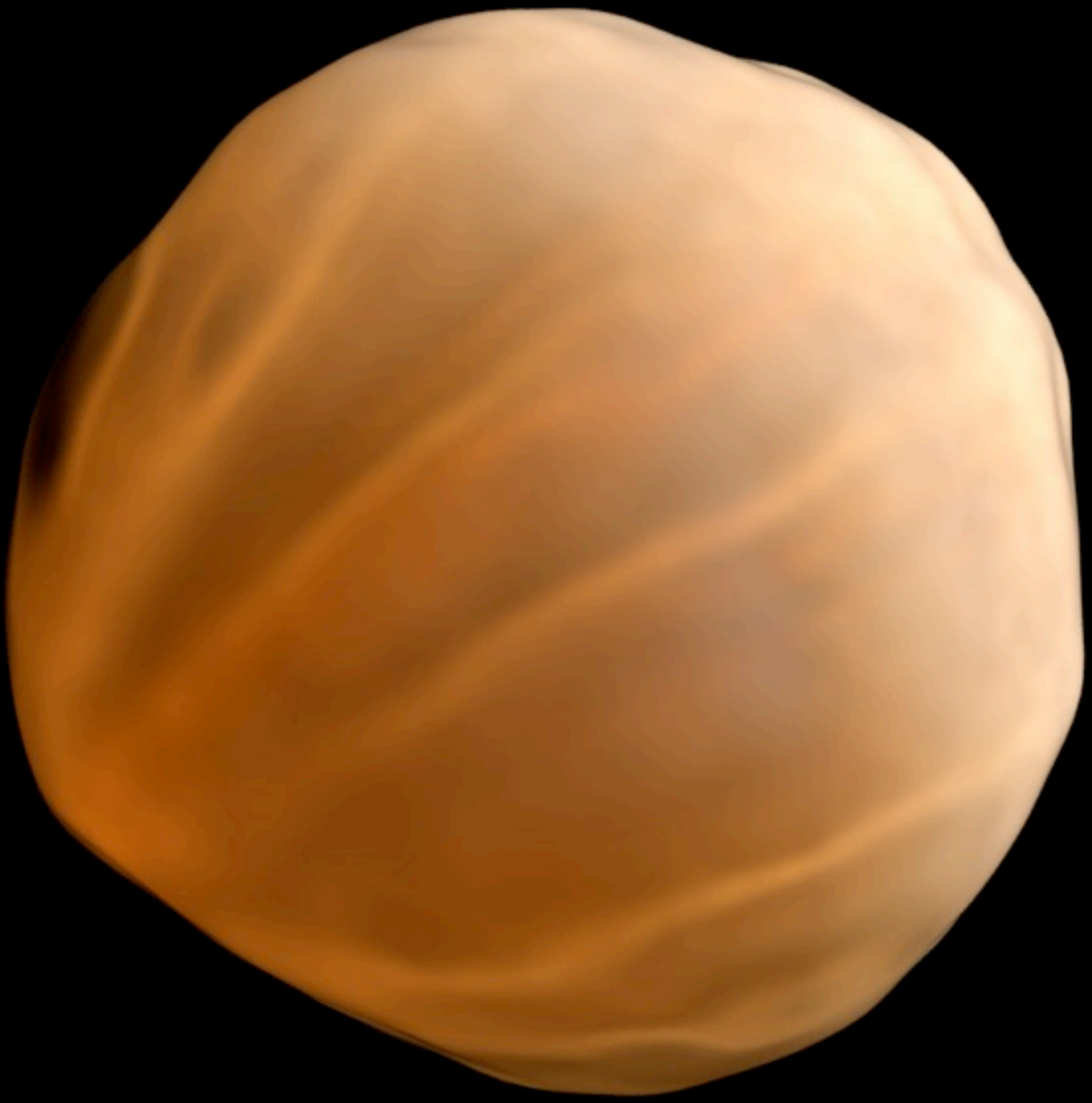
Our method: 1K Gaussians

(fit to 294K photons)

17+15 = 22 s

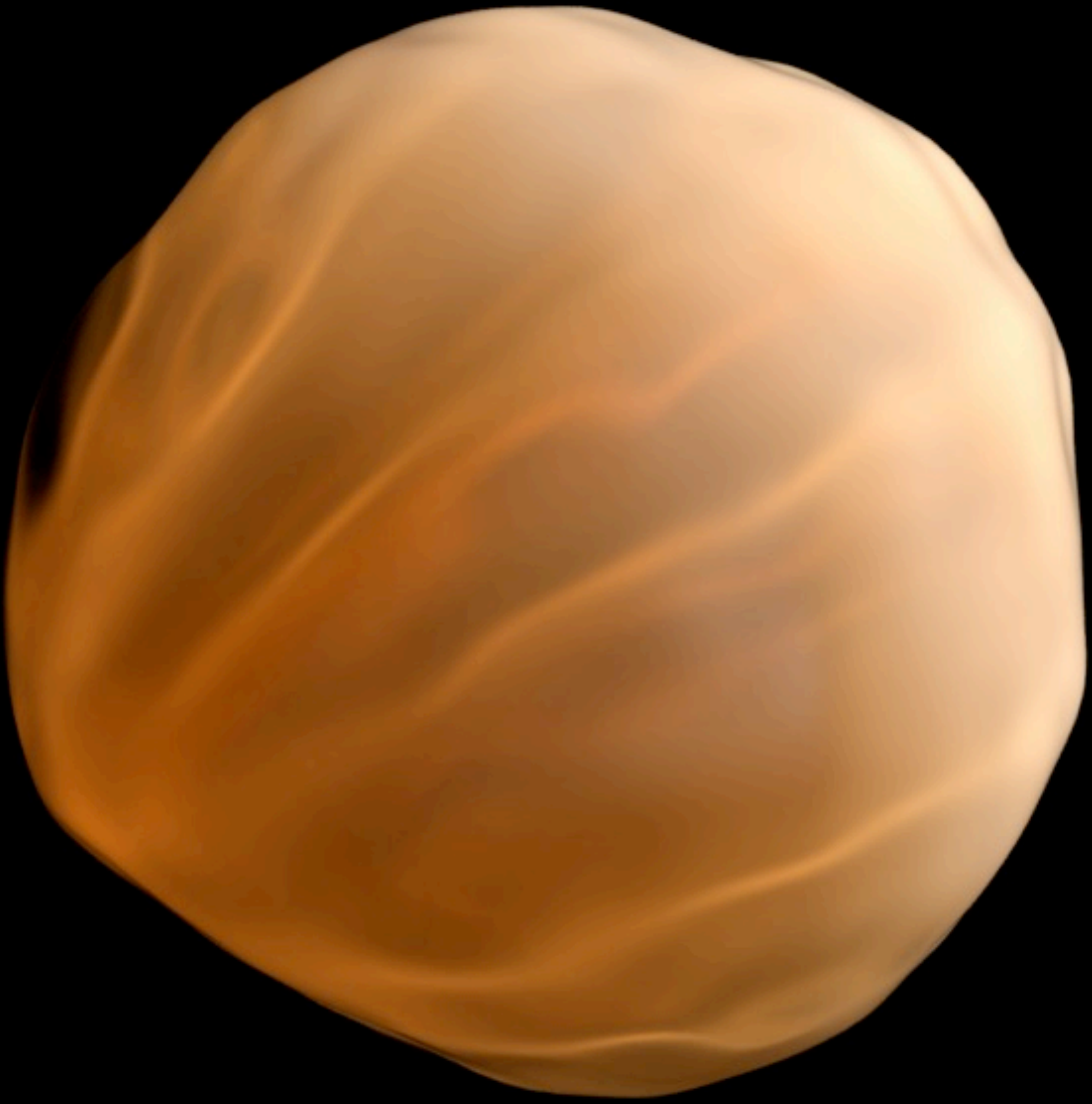
(6.0×)

When we run our method so that it fits to exactly the same amount of photons, we obtain a much crisper rendering, and we can do this in a fraction of the time.



BRE: 1M Photons 23+192 = 215 s

Here is BRE rendering with 1 million photons.



Our method: 4K Gaussians

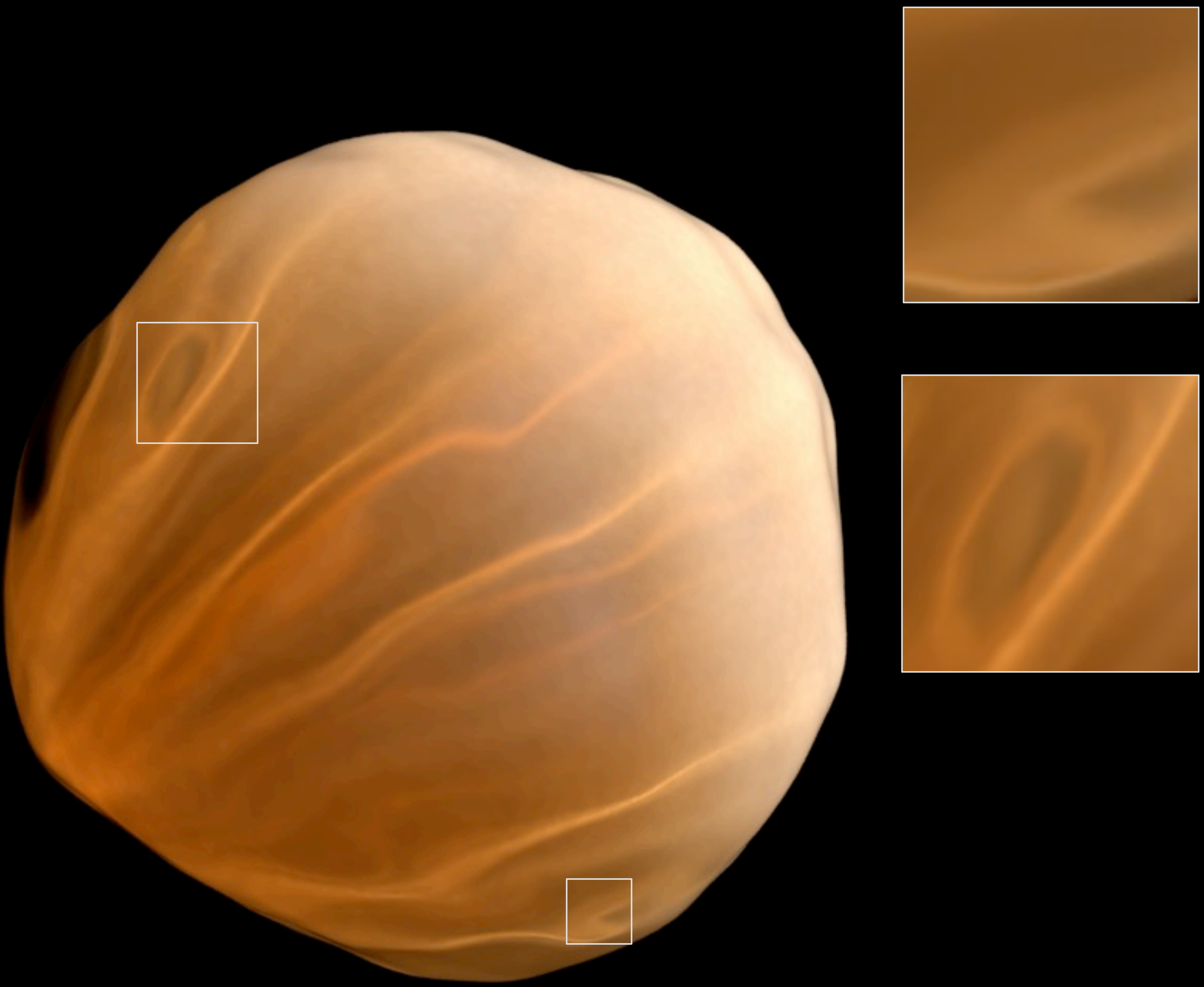
(fit to 1M photons)

35+24 = 59 s

(3.6×)

And this is a rendering of our method with 4K gaussians fit to 1M photons. Even though the preprocessing takes longer than in volumetric photon mapping, our method is faster overall, and we see an improvement in the quality.

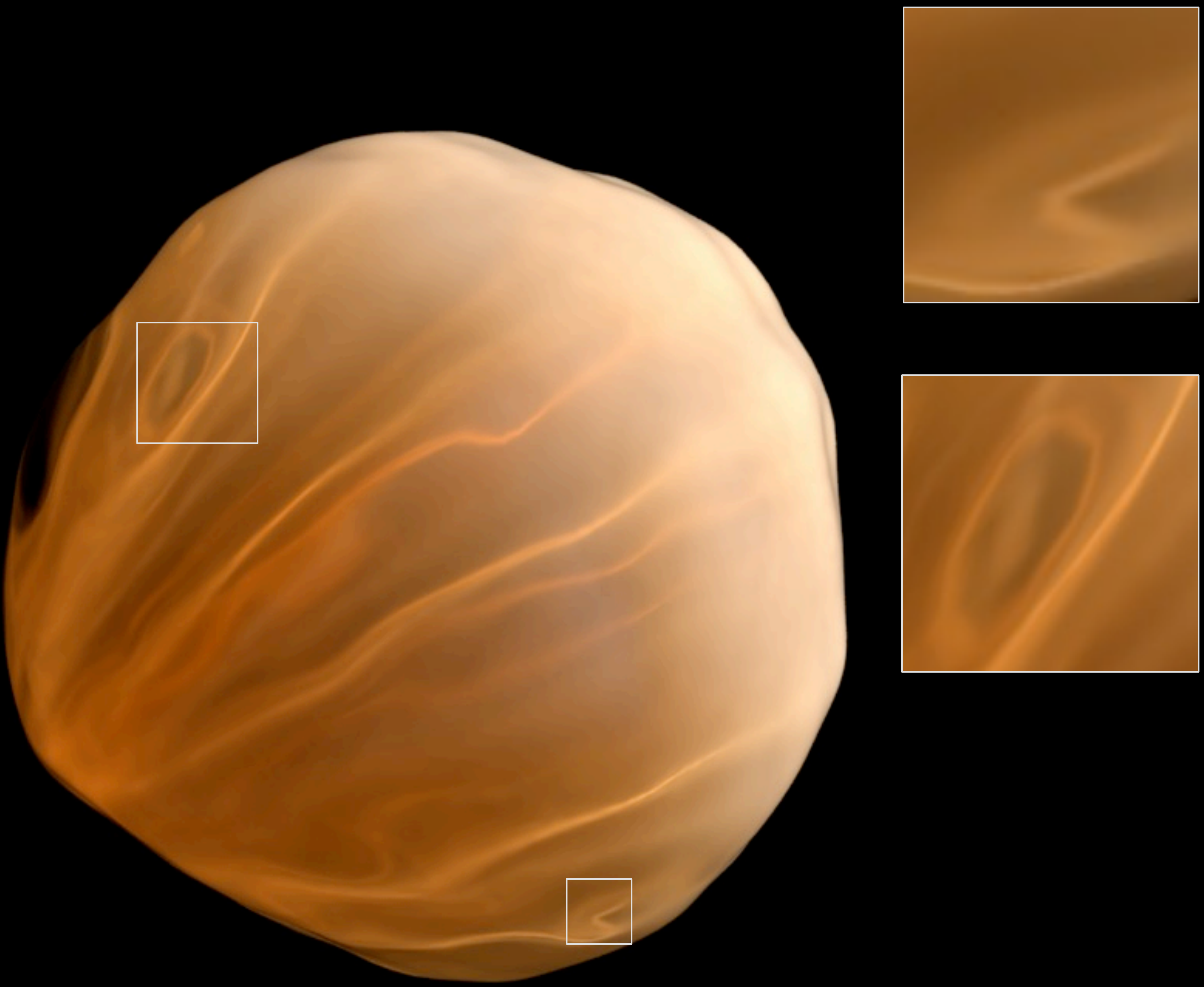
In a way, our method could be seen to act as a smart noise reduction and sharpening filter.



BRE: 18M Photons

507+609 = 1116 s

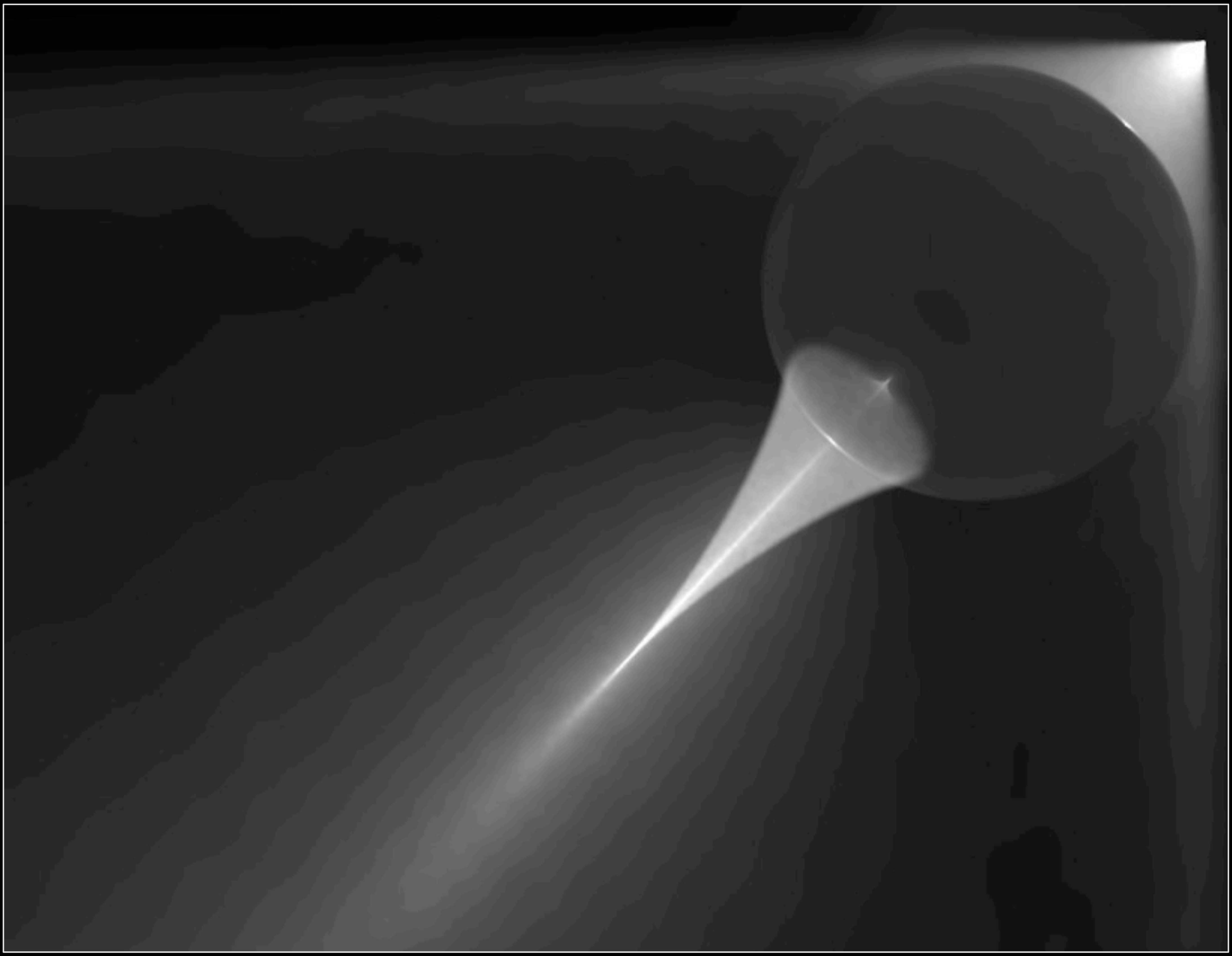
Here is BRE rendering with as much as 18 million photons.



Our method: 64K Gaussians
(fit to 18M photons)

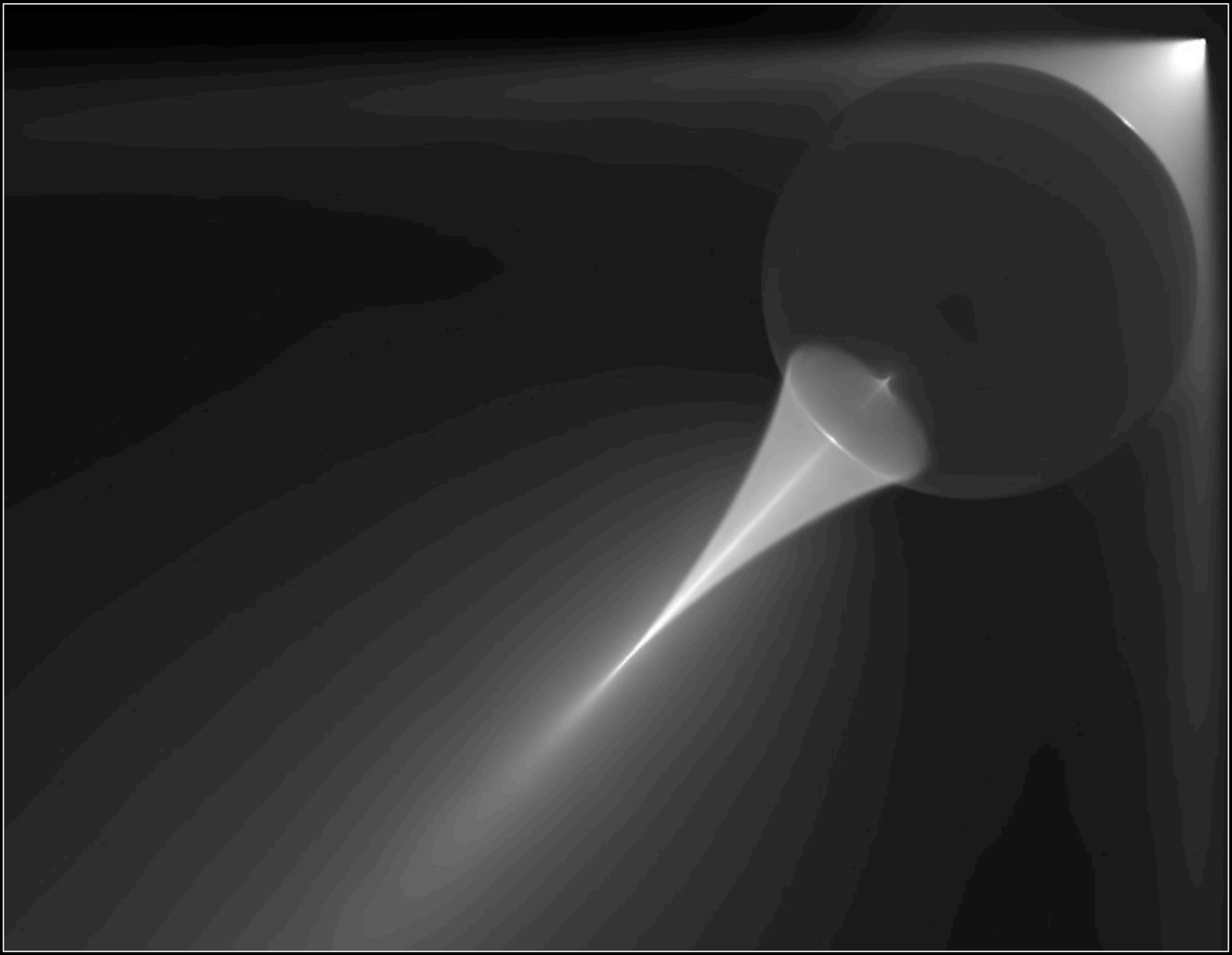
868+66 = 934 s
(1.2×)

and the corresponding rendering using our method. Even at this resolution, the sharpening effect still enhances features noticably.



BRE: 4M Photons 89 + 638 = 727 s

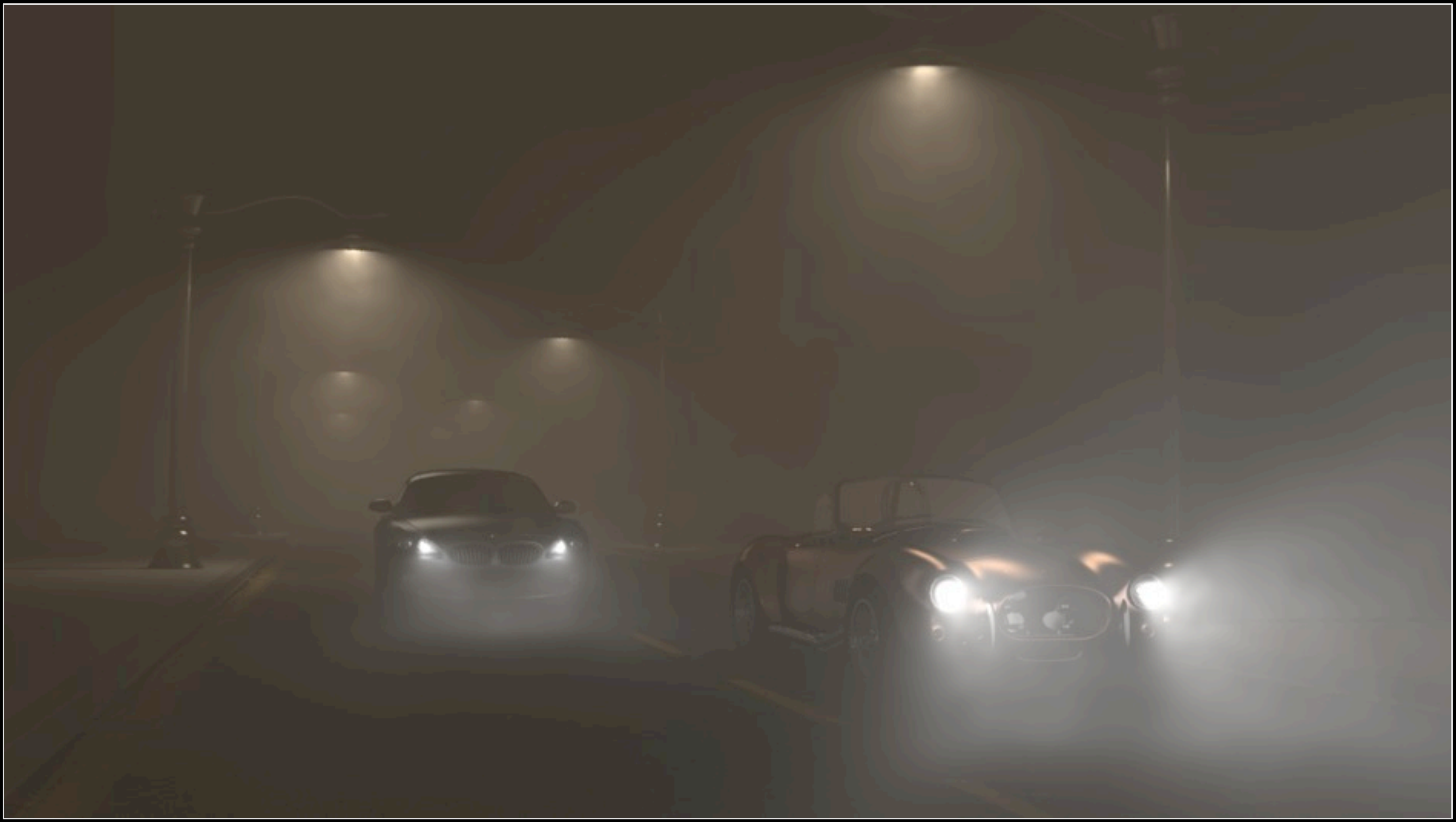
Here is BRE rendering of the sphere caustic we saw earlier, using 4 million photons.



Our method: 16K Gaussians

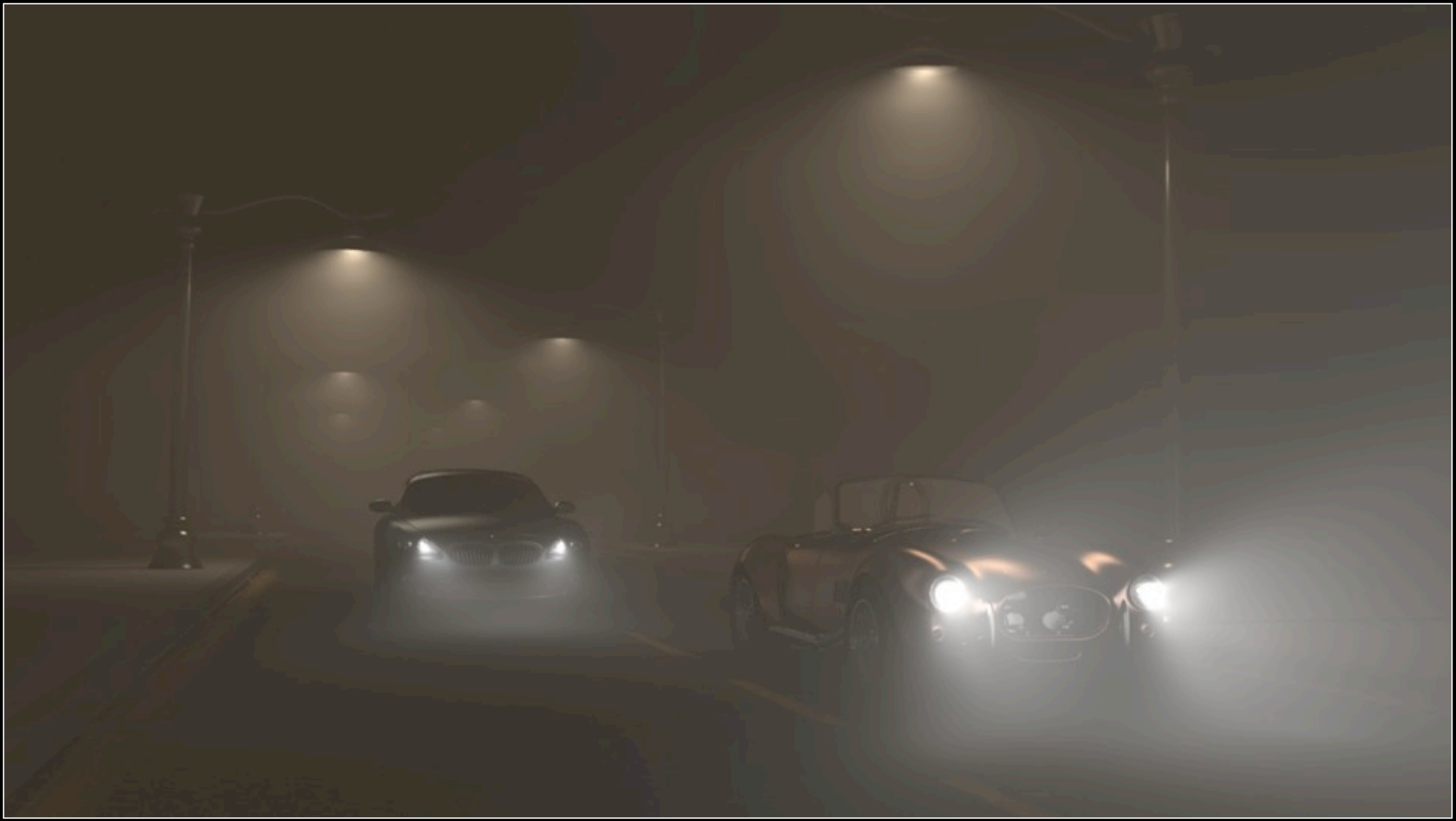
330 + 127 = 457 s
(1.6×)

Here is rendering using our method and 16 thousand Gaussians, and you can really see the noise reduction. The use of anisotropy is really valuable in scenes like this.



BRE: 917K Photons 13+268 = 281 s

And finally, here is a rendering of the cars scene, with about a million photons.



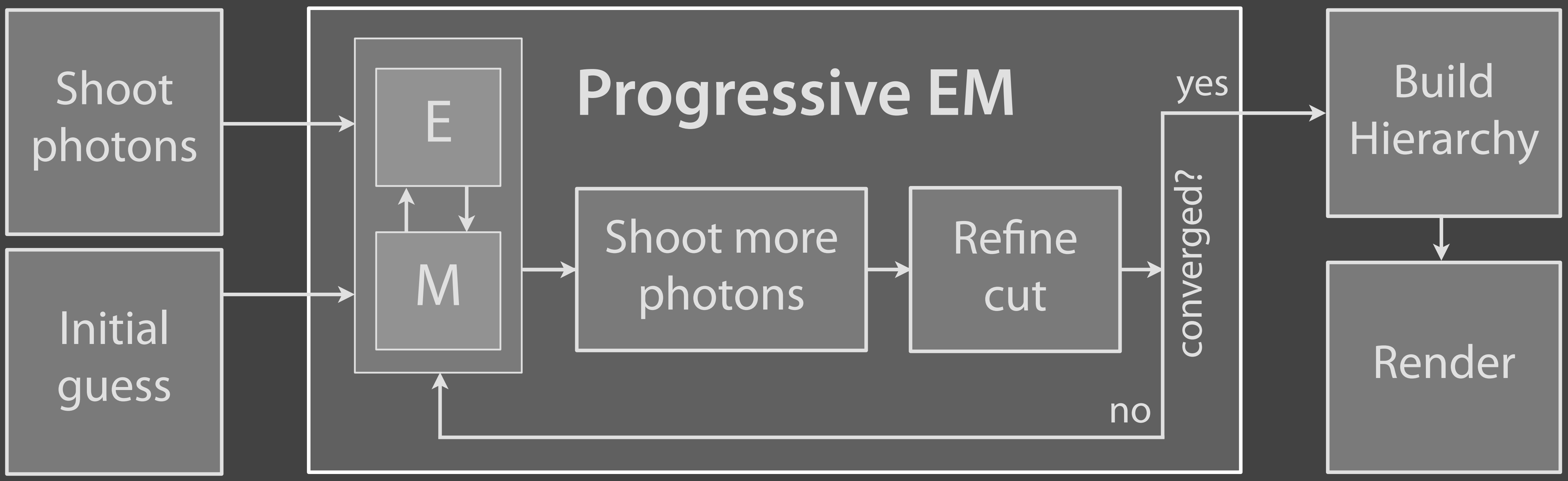
Our method: 4K Gaussians

54 + 71 = 125 s
(2.2×)

.. and the corresponding rendering using our methods.

To summarize these results: our method can obtain better quality through intelligent filtering of the data, while at the same time using a significantly smaller number of terms in its internal radiance representation. Because we're using so few terms, rendering is much faster, which more than makes up for the more involved preprocessing.

Temporal Coherence



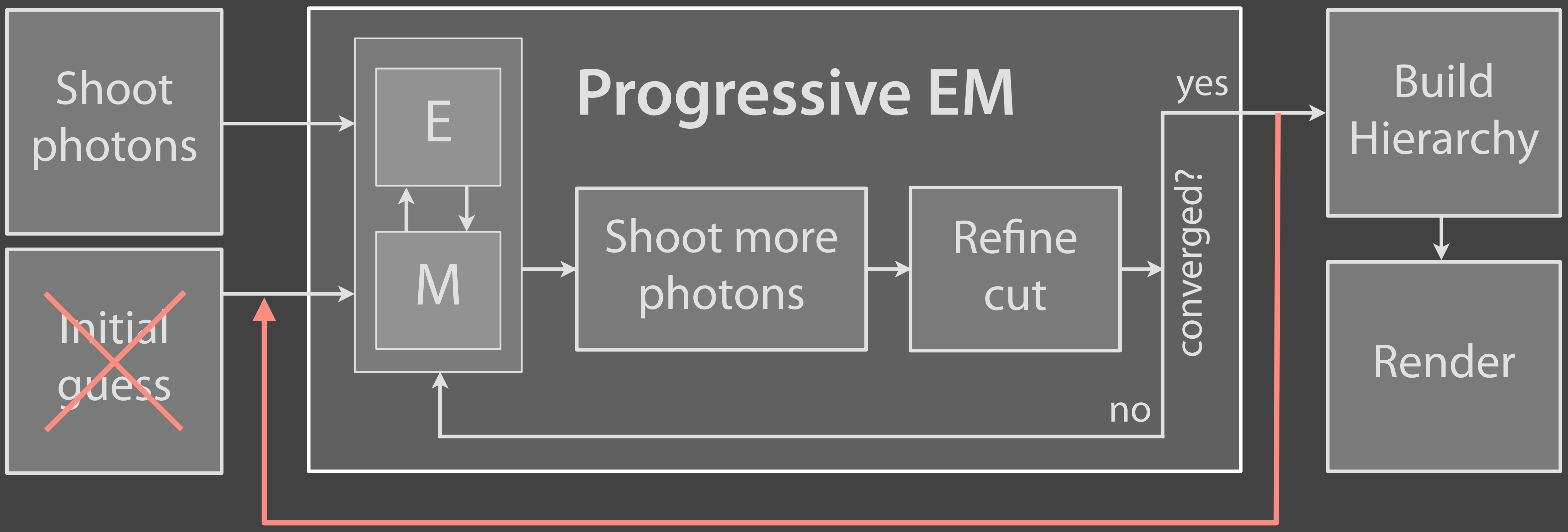
- Feed the result of the current frame into the next one

Thursday, 6 September 12

There are two extensions of our method.

One applies when rendering animations involving volumetric lighting. Usually, in these kinds of scenes, the radiance distribution will only undergo relatively small changes from frame to frame. We can make use of this property by passing [CLICK] the solution of one frame as the initial guess of the next frame . Not only does this accelerate the fitting considerably: it also leads to a smooth GMM that doesn't suffer from temporal noise.

Temporal Coherence



- Feed the result of the current frame into the next one

  ⟶ Faster fitting, no temporal noise

Thursday, 6 September 12

There are two extensions of our method.

One applies when rendering animations involving volumetric lighting. Usually, in these kinds of scenes, the radiance distribution will only undergo relatively small changes from frame to frame. We can make use of this property by passing [CLICK] the solution of one frame as the initial guess of the next frame . Not only does this accelerate the fitting considerably: it also leads to a smooth GMM that doesn’t suffer from temporal noise.

Scene 1: BUMPYSPHERE

Volume caustics from a rotating light source

Thursday, 6 September 12

Here is a quick demo of this, showing the bumpy sphere scene with a rotating light source.

In the BRE rendering, you can clearly see temporal noise from the photon tracing step. Using our method without temporal coherence, the rendering is also quite noisy, but now it’s the parameters of the Gaussians that are affected by noise. But if we re-use the previous frame’s solution as an initial guess, then this noise goes away completely.

Realtime Visualization

GPU-based rasterizer:

- Anisotropic Gaussian splat shader: 30 lines of GLSL
- Gaussian representation is very compact (4096-term GMM requires only ~240KB of storage)
- View-independent precomputation
- Possible applications in games

Live demo:

- Run on an NVidia GeForce 9600M GT graphics card

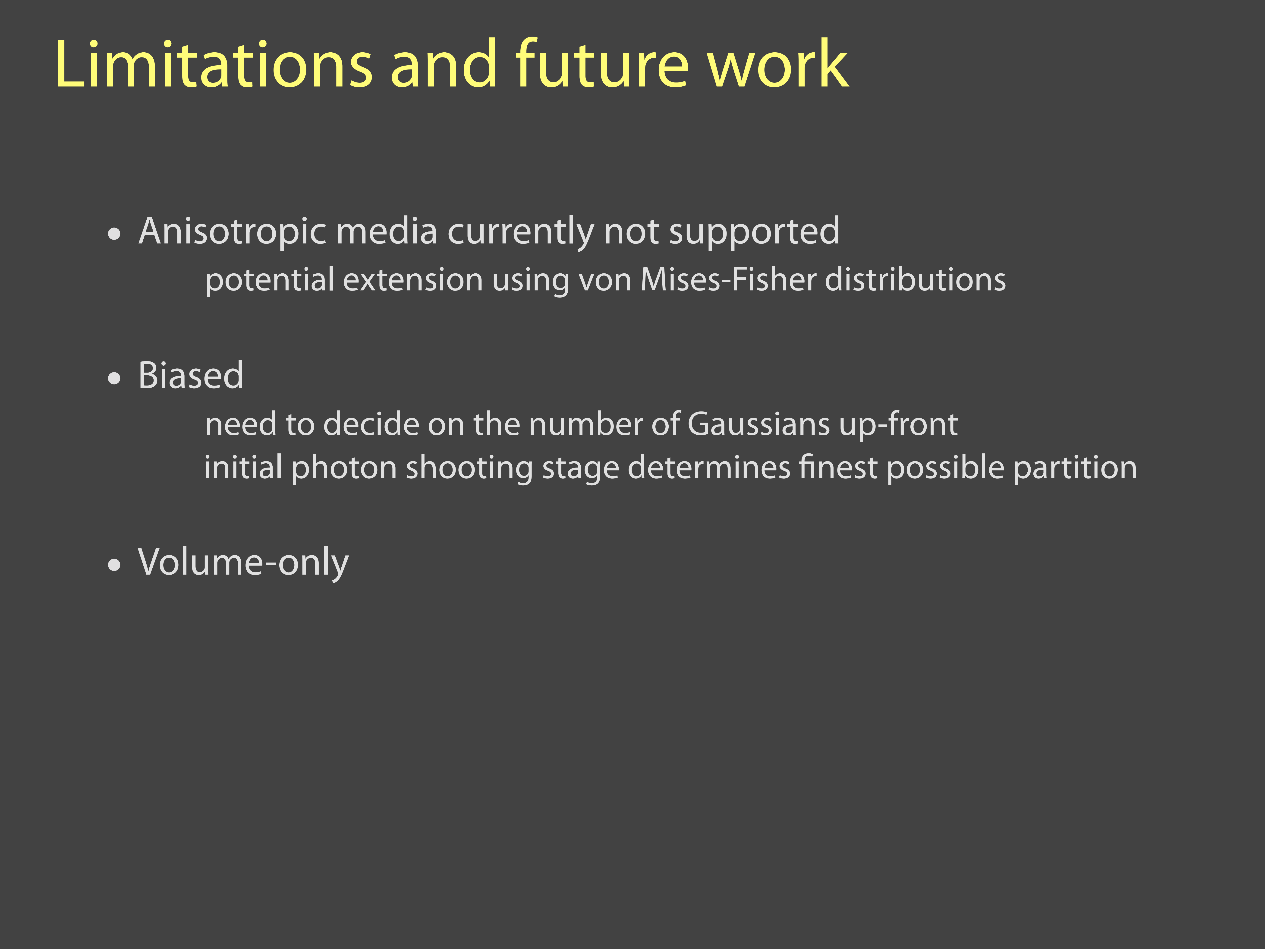
Thursday, 6 September 12

Another application is to use our fitted GMMs in the context of realtime visualization.

It is possible to code up the contribution from an anisotropic Gaussian in a simple GLSL shader so
that large amounts of Gaussians can be drawn at realtime using a splatting approach.

Since the GMM representation only has a tiny storage footprint, it could be of use in Games
where a a view-independent preprocess renders fixed or animated volumetric lighting to files,
which are then drawn interactively while a character moves through a scene.

Next'll show a demo such a visualization -- we will take a look at caustics from a moving water
surface, which is the scene shown in the image on the top here. [[This is running on the



Limitations and future work

- Anisotropic media currently not supported
  - potential extension using von Mises-Fisher distributions
- Biased
  - need to decide on the number of Gaussians up-front
  - initial photon shooting stage determines finest possible partition
- Volume-only

Thursday, 6 September 12

Our method has a few limitations. It currently does not support anisotropic media, since the Gaussian representation does not store the directionality of light. One potential workaround will be to run the algorithm in 5D, by letting it operate on a product of Gaussian and von Mises-Fisher distributions.

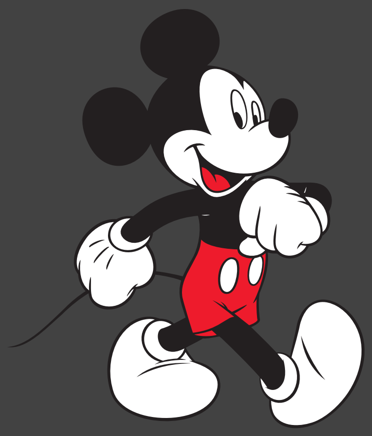
Also, our method is currently statistically biased, since it is necessary to decide upon a number of Gaussians up-front and furthermore, the initial photon shooting stage determines the finest possible partition that is available to EM. Like progressive photon mapping, our method continually shoots photons as the algorithm runs. While this means that the amount of noise will decrease over time, there is a point after which fine details in the solution improved any further. This is in contrast to progressive photon mapping, which can achieve any desired accuracy level when it is given enough time. One possible solution may be to dynamically extend the number of Gaussians as the algorithm runs, but we have not implemented such a scheme.

The final limitation is that, the algorithm of course only works for for volumes, which are

Conclusion

Contributions

- Rendering technique based on parametric density estimation
- Uses a progressive and optimized variant of accelerated EM
- Compact & hierarchical representation of volumetric radiance
- Extensions for temporal coherence and real-time visualization
- General technique, possible applications outside of graphics



Thursday, 6 September 12

To conclude, we have presented an algorithm for rendering volumetric lighting using parametric density estimation and a progressive variant of accelerated EM. Our approach makes volumetric photon mapping practical by fitting a compact gaussian mixture, incorporating level-of-detail, and eliminating temporal flickering. We also presented a way to use this approach for real-time visualization. Most parts of our technique are quite general and could have potential applications outside of graphics as well.

Thanks for listening, and if there are any questions, I'll take them now.