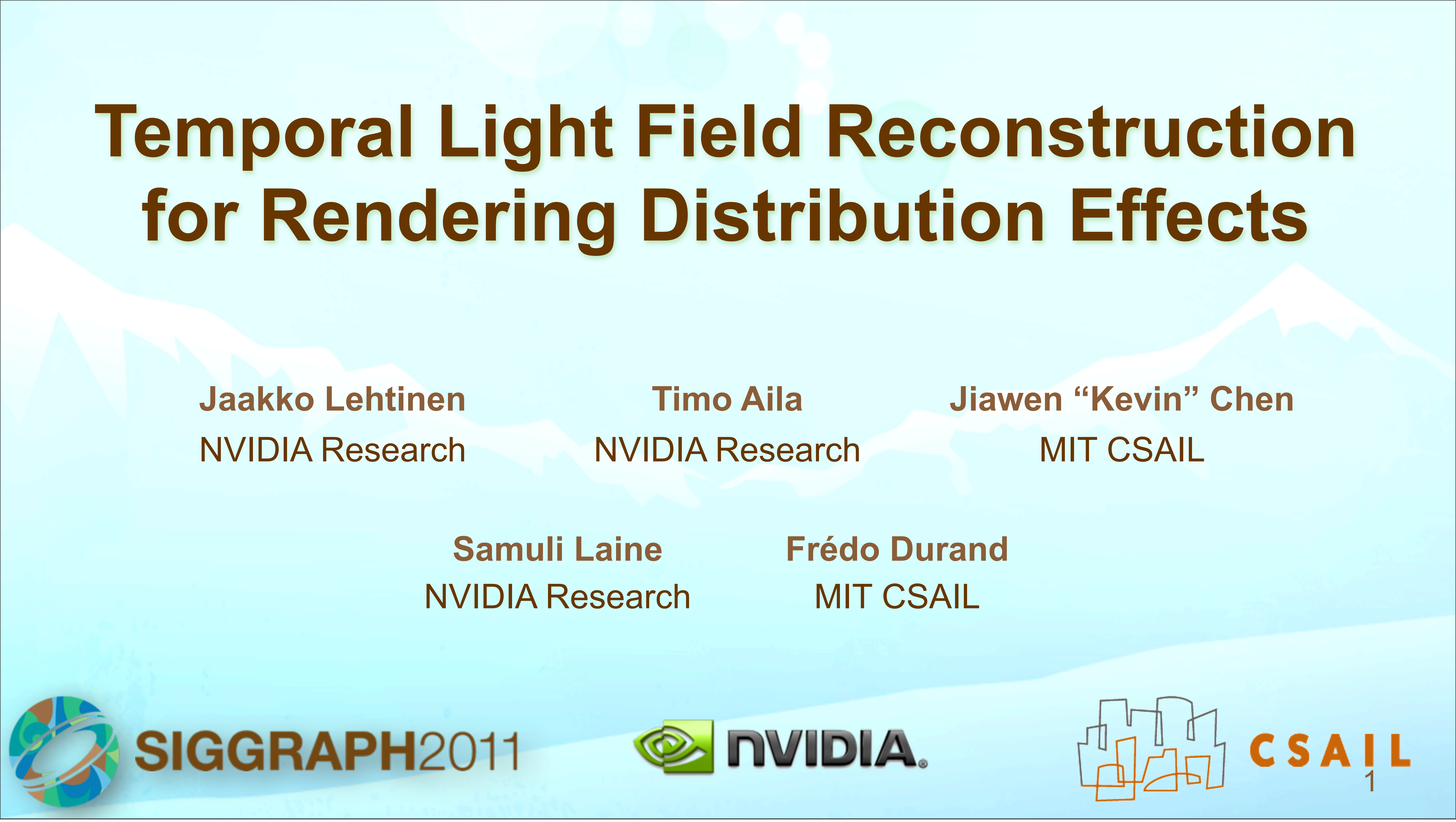


Temporal Light Field Reconstruction for Rendering Distribution Effects

Jaakko Lehtinen
NVIDIA Research

Timo Aila
NVIDIA Research

Jiawen “Kevin” Chen
MIT CSAIL

Samuli Laine
NVIDIA Research

Frédo Durand
MIT CSAIL

SIGGRAPH2011

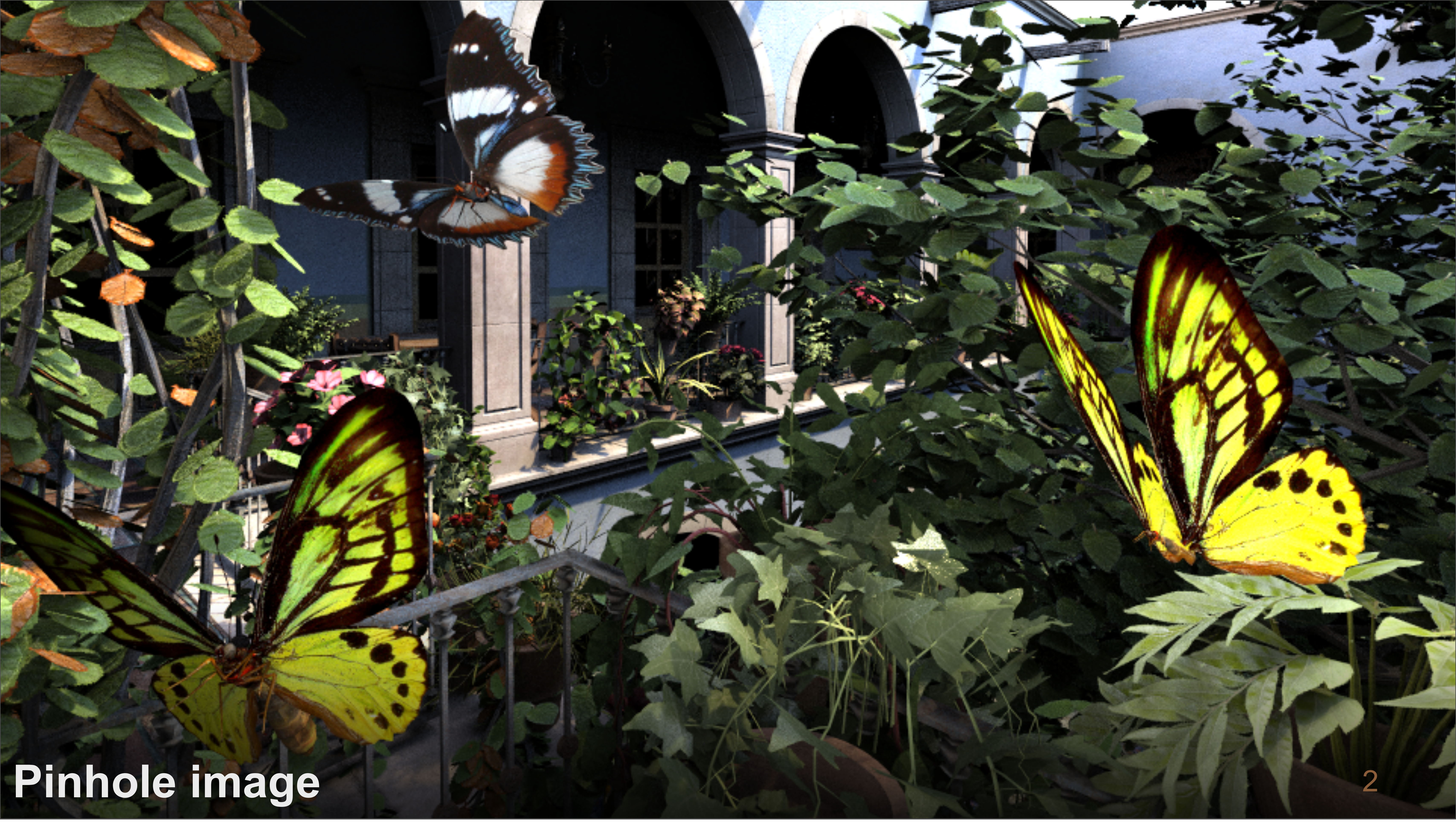
NVIDIA.

CSAIL

Thank you for the introduction.

This talk is about efficiently rendering distribution effects like motion blur, depth of field, and soft shadows.

To illustrate...

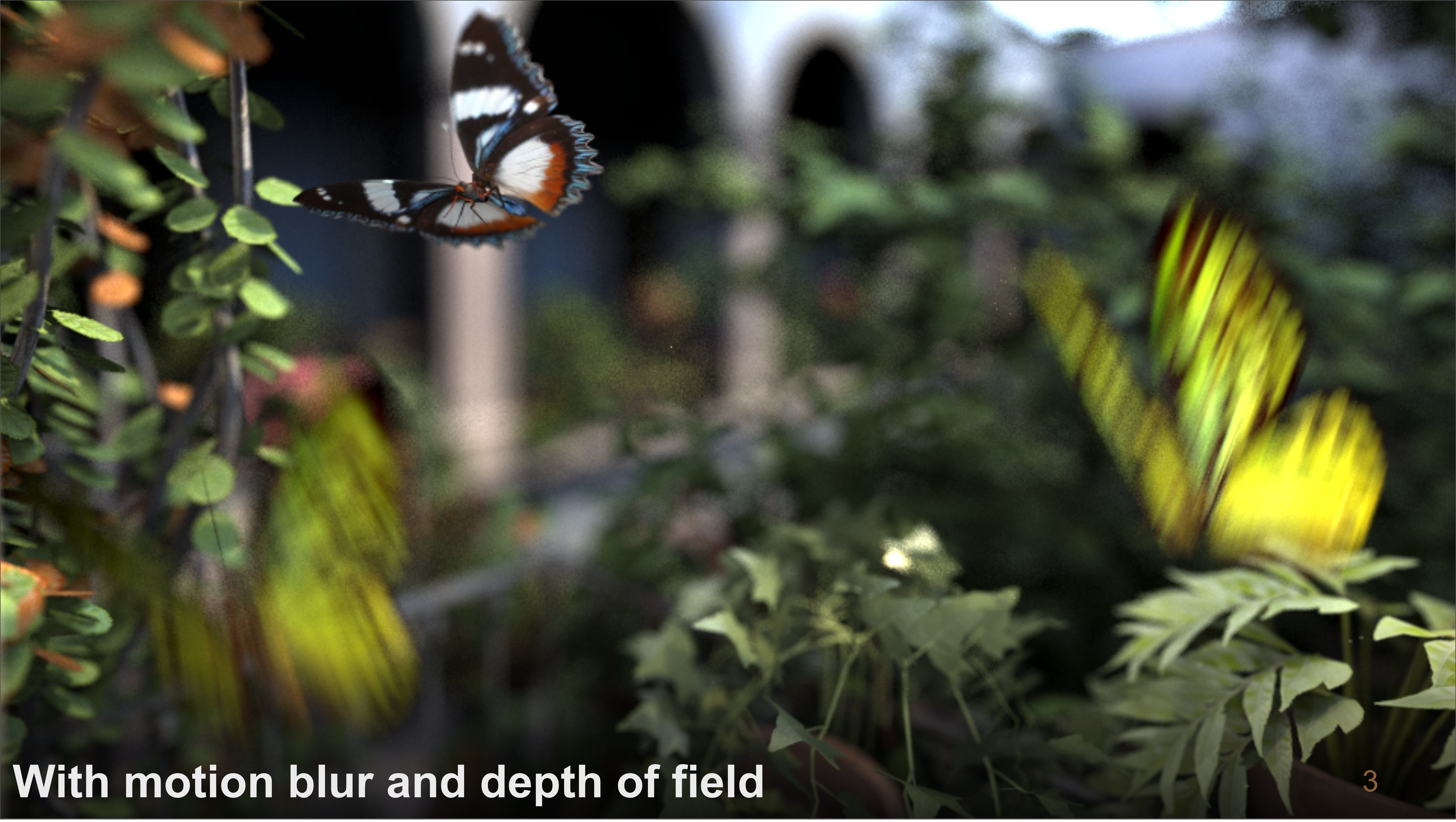


Pinhole image

The image was rendered using the pinhole camera model found in standard ray tracers and rasterizers.

Notice how everything is sharp.

The butterflies that are actually moving appear frozen in time, and both the foreground and background are in focus.



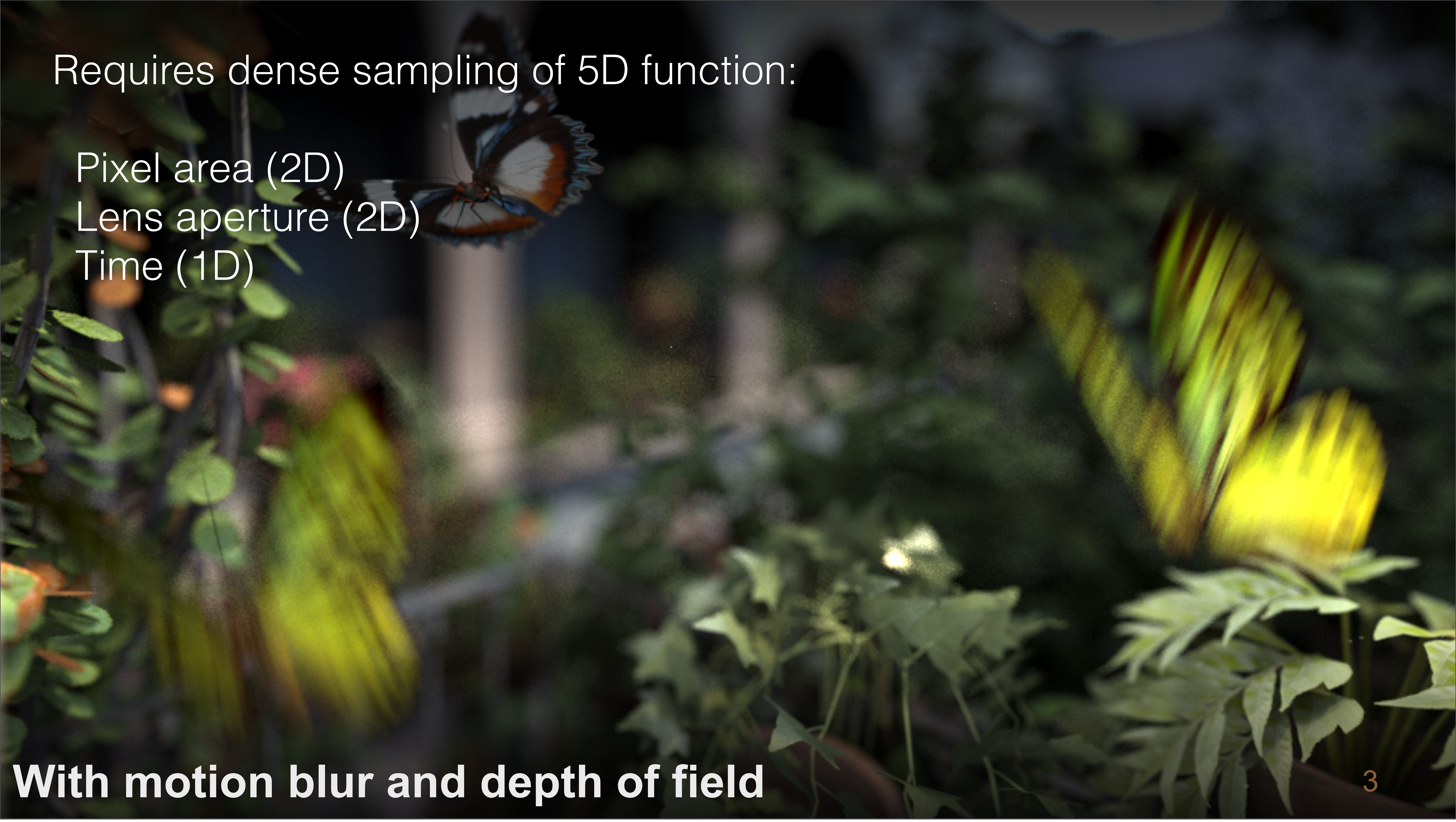
With motion blur and depth of field

This is the same scene rendered with motion blur and depth of field where out of focus objects are blurred.

Distribution effects such as motion blur and depth of field can significantly increase the realism of computer generated images.

The traditional method for rendering these phenomena is to take many samples per pixel.
<animate>
Samples are distributed not only over the pixel area, but also the lens aperture for depth of field, and time for motion blur.

The brute force approach works really well when you take a large number of samples.
However, it is computationally expensive.



Requires dense sampling of 5D function:

- Pixel area (2D)
- Lens aperture (2D)
- Time (1D)

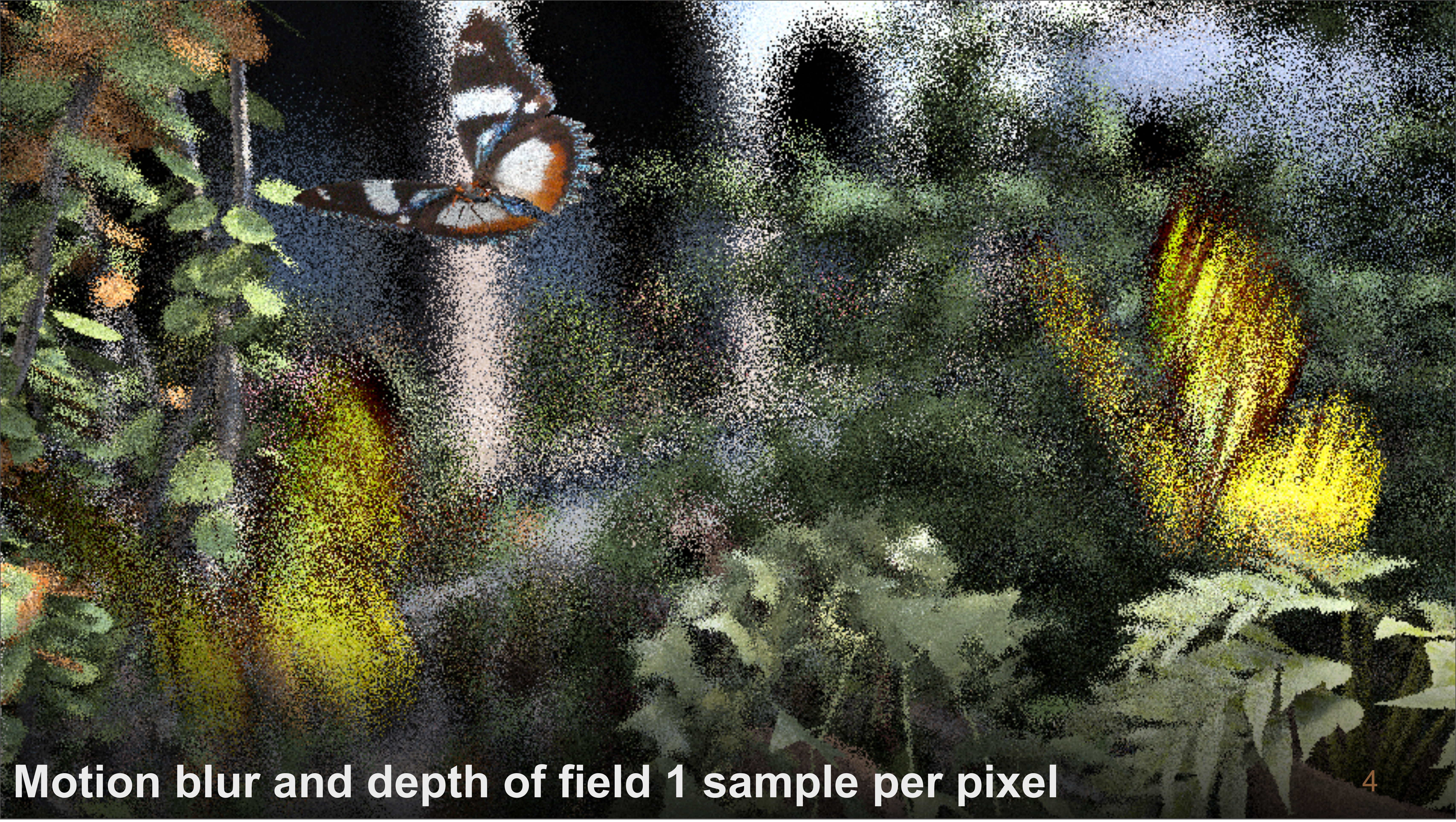
With motion blur and depth of field

This is the same scene rendered with motion blur and depth of field where out of focus objects are blurred.

Distribution effects such as motion blur and depth of field can significantly increase the realism of computer generated images.

The traditional method for rendering these phenomena is to take many samples per pixel.
<animate>
Samples are distributed not only over the pixel area, but also the lens aperture for depth of field, and time for motion blur.

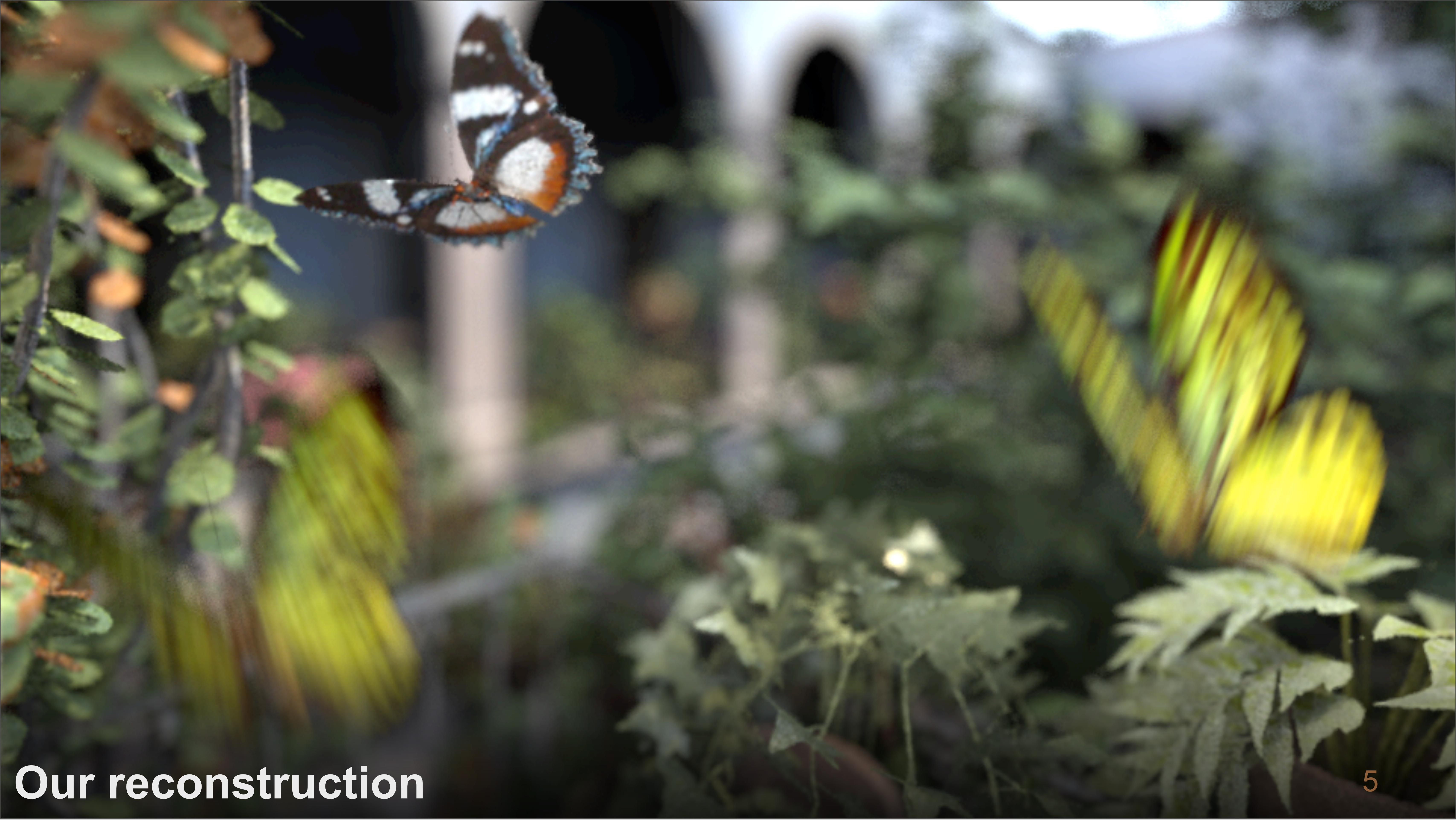
The brute force approach works really well when you take a large number of samples.
However, it is computationally expensive.



Motion blur and depth of field 1 sample per pixel

If you don’t use enough samples, as I show here in the extreme case using only 1 sample per pixel, you get a really noisy image

Our algorithm can take the exact same samples used in this image...



Our reconstruction

and produce this noise-free result.

<pause>

For most of the talk, I’m going to discuss our algorithm in terms of depth of field, so let’s look at how it works

Pinhole camera model

background

object

Let’s begin with the pinhole camera model used in standard raytracing and rasterization.
In this model, we have objects in the scene,
<animate>
along with a virtual sensor plane.

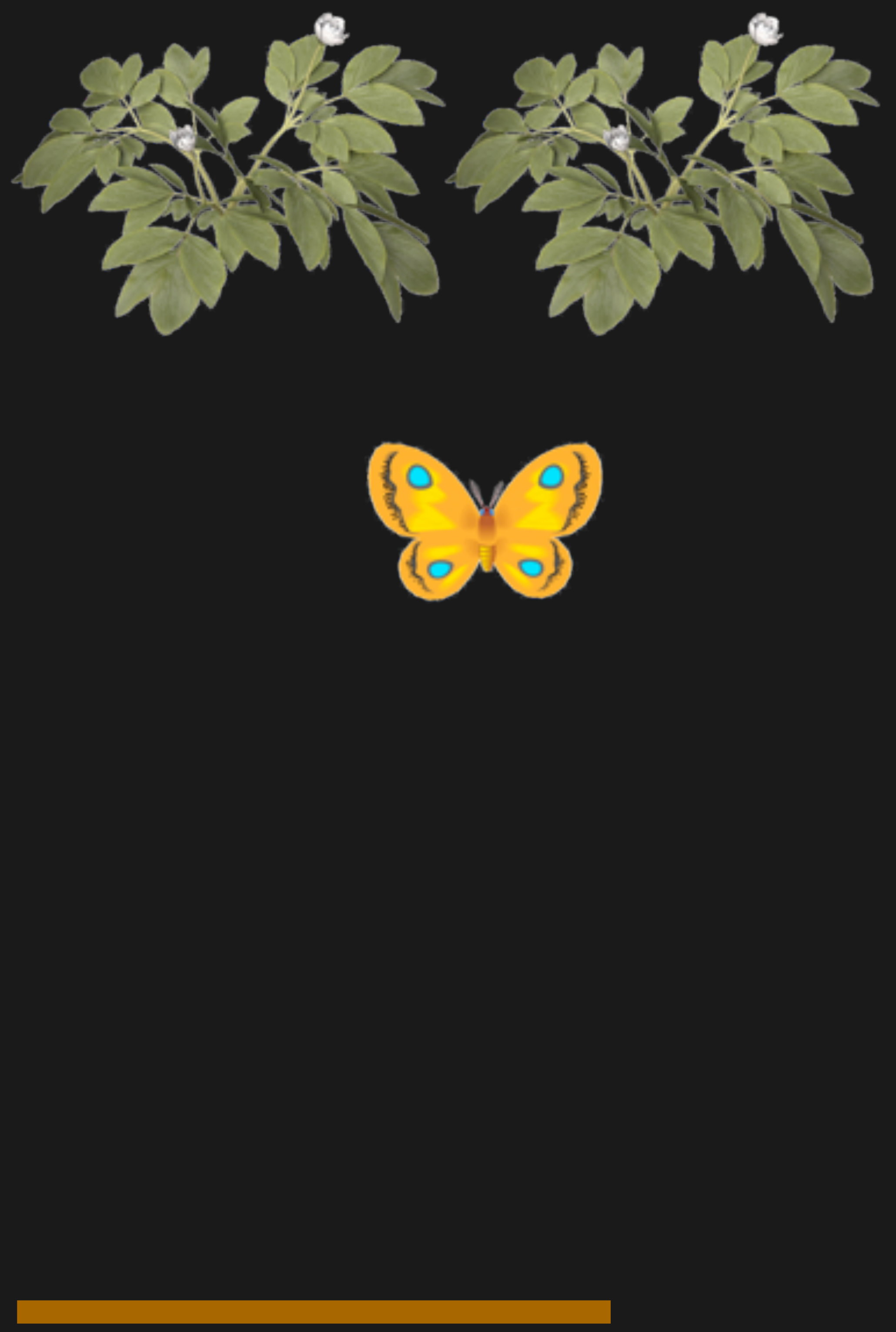
<animate>
We also have a single point in space that we call the pinhole.

With a pinhole camera, all light rays that hit the sensor have to go through the pinhole.
If we consider this pixel, <animate>
it corresponds to a unique ray into the scene, which hits the butterfly. <animate>

Similarly, <animate> this ray hits the background.

And the resulting image is sharp.

Pinhole camera model



background

object

sensor

Let’s begin with the pinhole camera model used in standard raytracing and rasterization.
In this model, we have objects in the scene,
<animate>
along with a virtual sensor plane.

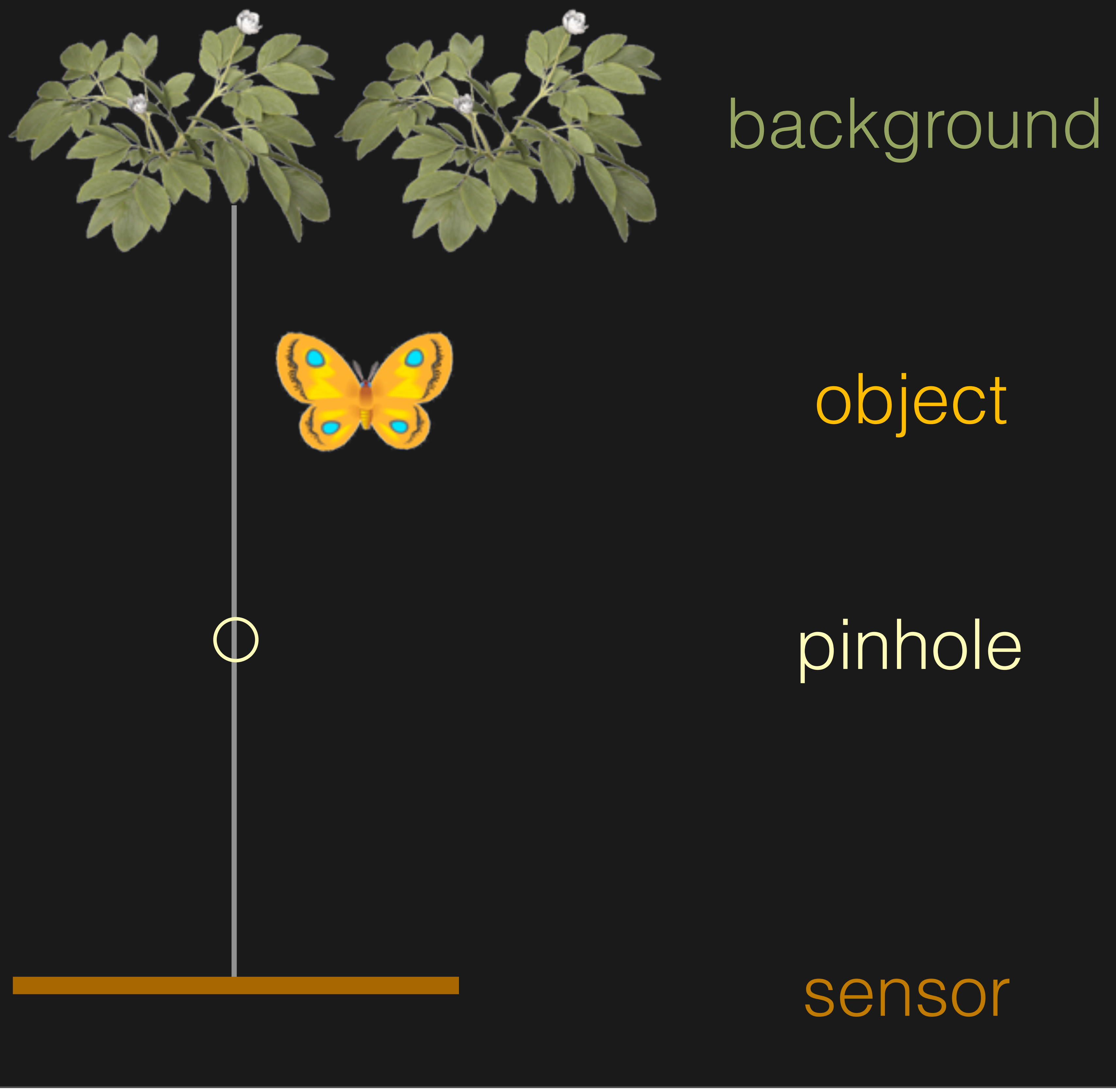
<animate>
We also have a single point in space that we call the pinhole.

With a pinhole camera, all light rays that hit the sensor have to go through the pinhole.
If we consider this pixel, <animate>
it corresponds to a unique ray into the scene, which hits the butterfly. <animate>

Similarly, <animate> this ray hits the background.

And the resulting image is sharp.

Pinhole camera model



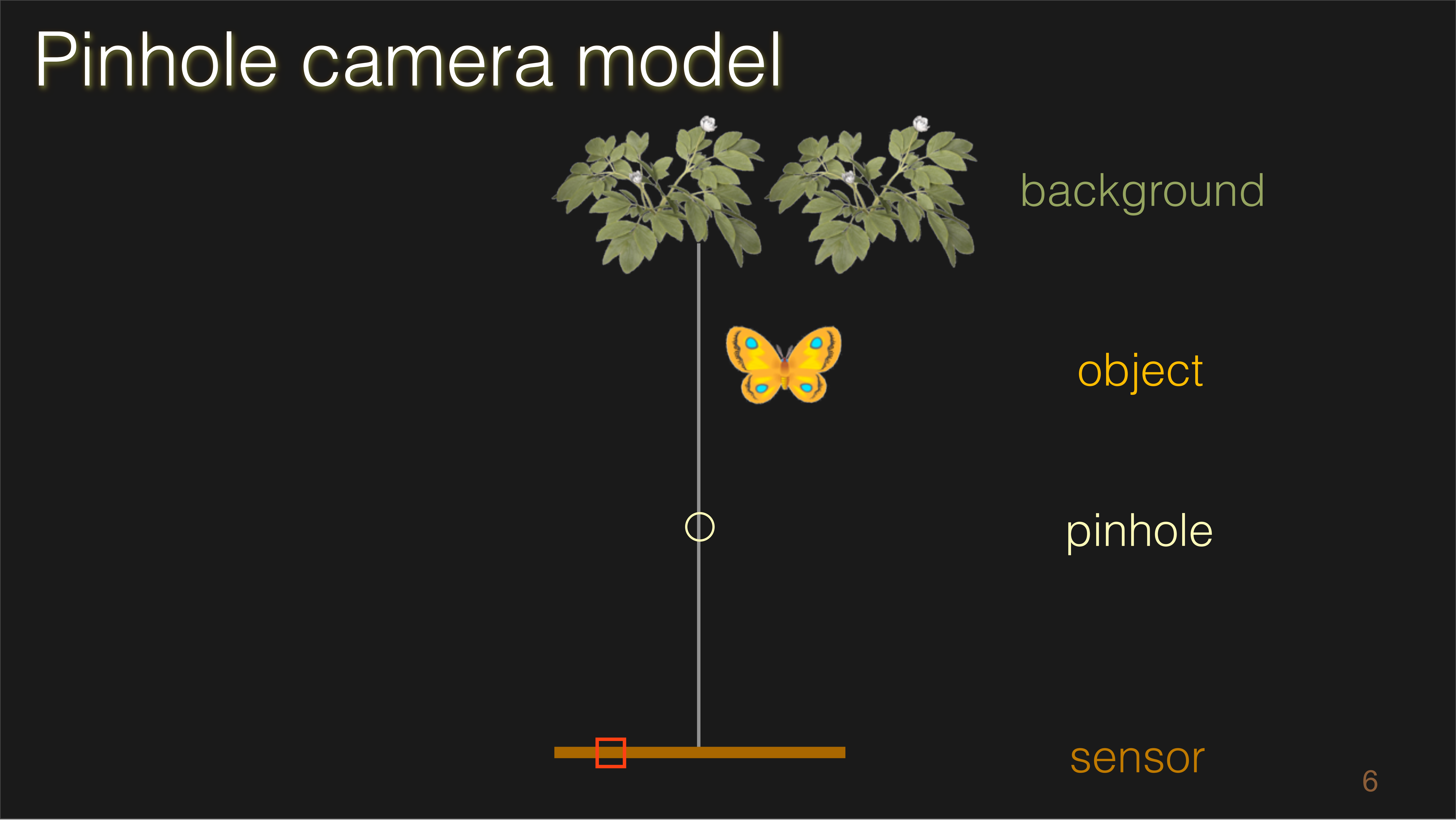
Let's begin with the pinhole camera model used in standard raytracing and rasterization.
In this model, we have objects in the scene,
<animate>
along with a virtual sensor plane.

<animate>
We also have a single point in space that we call the pinhole.

With a pinhole camera, all light rays that hit the sensor have to go through the pinhole.
If we consider this pixel, <animate>
it corresponds to a unique ray into the scene, which hits the butterfly. <animate>

Similarly, <animate> this ray hits the background.

And the resulting image is sharp.



Pinhole camera model

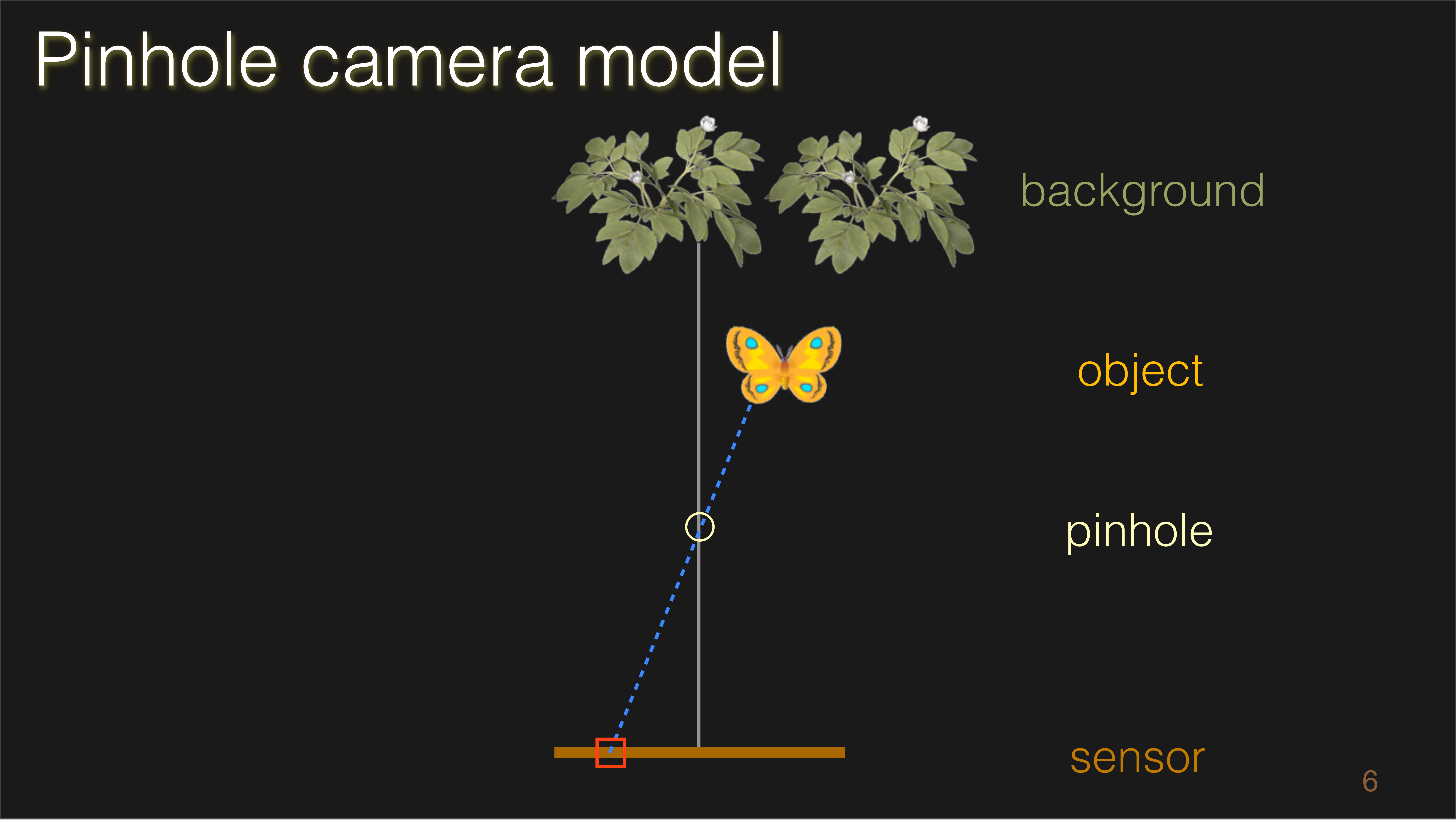
Let’s begin with the pinhole camera model used in standard raytracing and rasterization.
In this model, we have objects in the scene,
<animate>
along with a virtual sensor plane.

<animate>
We also have a single point in space that we call the pinhole.

With a pinhole camera, all light rays that hit the sensor have to go through the pinhole.
If we consider this pixel, <animate>
it corresponds to a unique ray into the scene, which hits the butterfly. <animate>

Similarly, <animate> this ray hits the background.

And the resulting image is sharp.



Pinhole camera model

background

object

pinhole

sensor

6

Let’s begin with the pinhole camera model used in standard raytracing and rasterization.
In this model, we have objects in the scene,
<animate>
along with a virtual sensor plane.

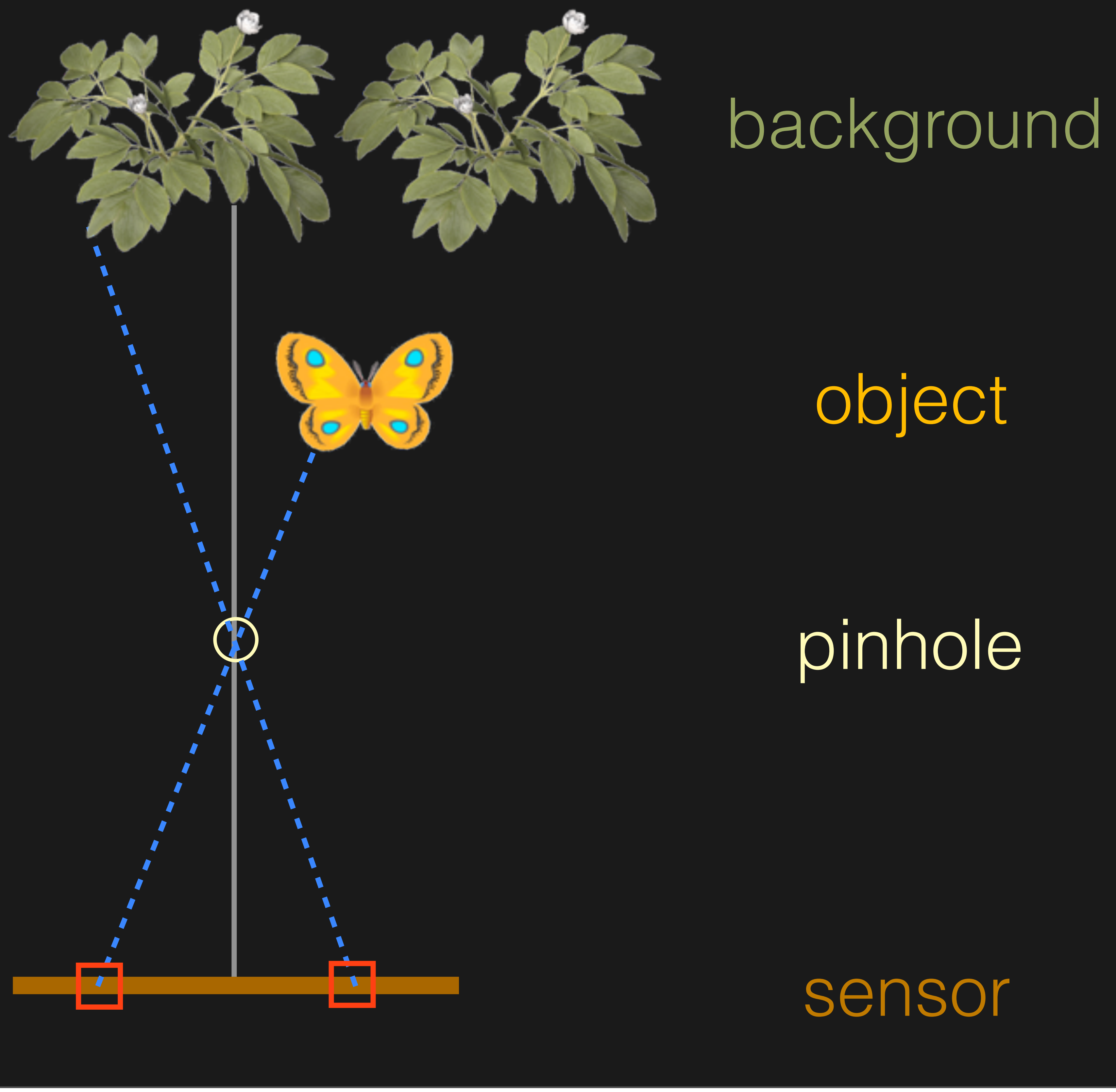
<animate>
We also have a single point in space that we call the pinhole.

With a pinhole camera, all light rays that hit the sensor have to go through the pinhole.
If we consider this pixel, <animate>
it corresponds to a unique ray into the scene, which hits the butterfly. <animate>

Similarly, <animate> this ray hits the background.

And the resulting image is sharp.

Pinhole camera model



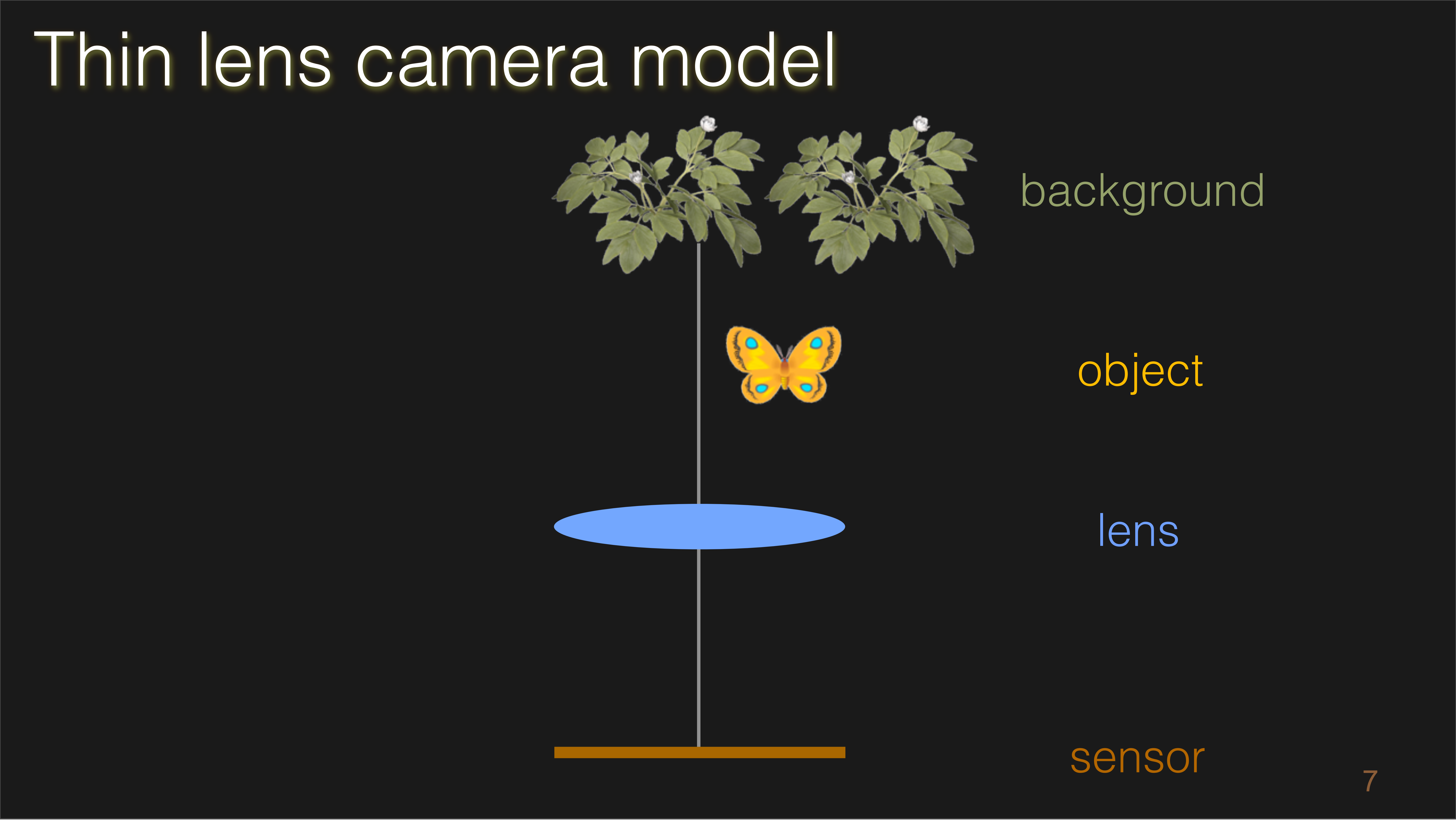
Let’s begin with the pinhole camera model used in standard raytracing and rasterization.
In this model, we have objects in the scene,
<animate>
along with a virtual sensor plane.

<animate>
We also have a single point in space that we call the pinhole.

With a pinhole camera, all light rays that hit the sensor have to go through the pinhole.
If we consider this pixel, <animate>
it corresponds to a unique ray into the scene, which hits the butterfly. <animate>

Similarly, <animate> this ray hits the background.

And the resulting image is sharp.



Thin lens camera model

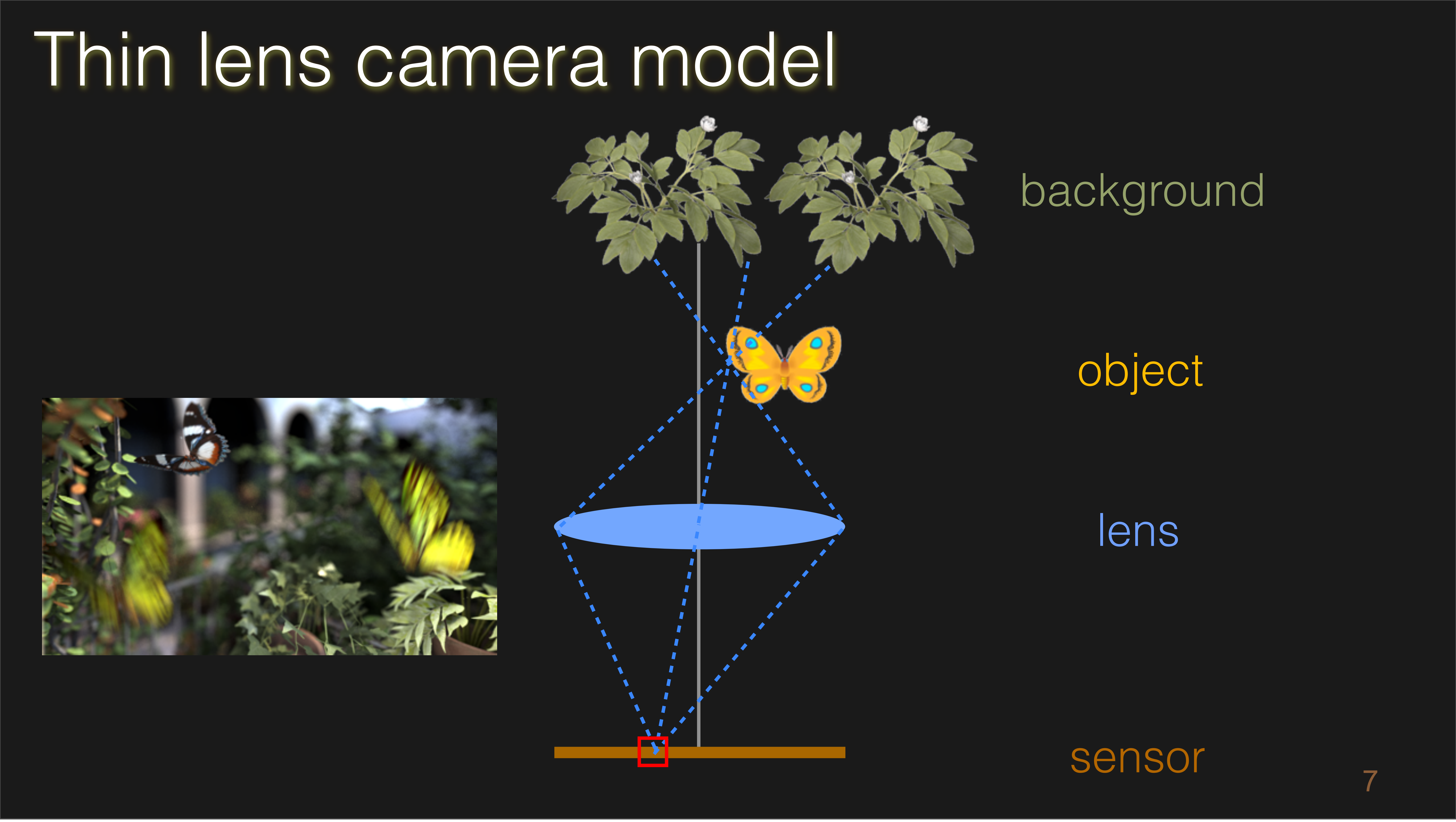
background

object

lens

sensor

A more realistic camera model replaces the pinhole with a thin lens.
A lens gathers light and focuses it onto the sensor.
As a result, a single pixel <animate>
can receive light from many different parts of the scene, and we get a blurry background.



Thin lens camera model

background

object

lens

sensor

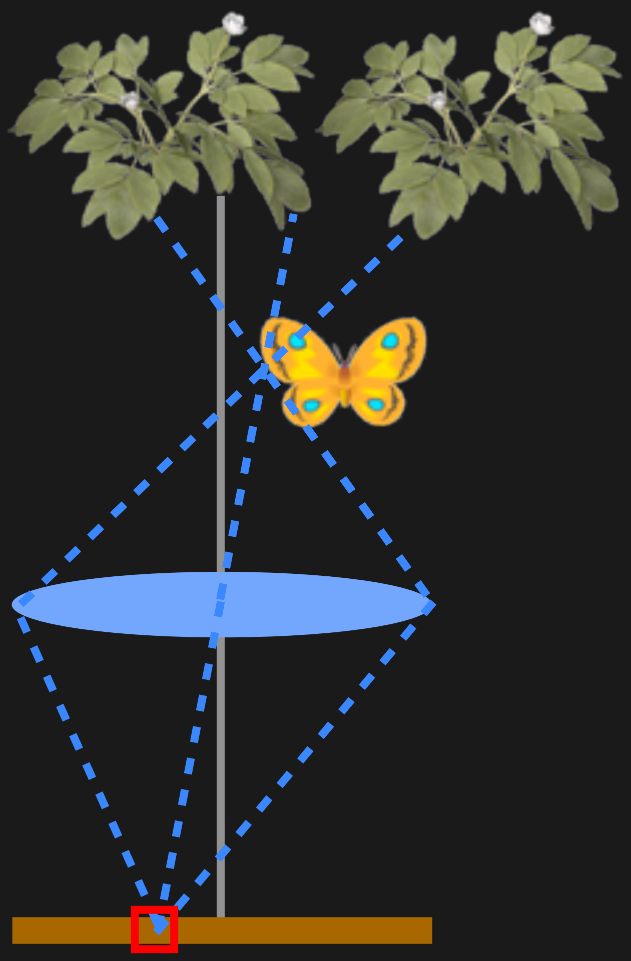
A more realistic camera model replaces the pinhole with a thin lens.

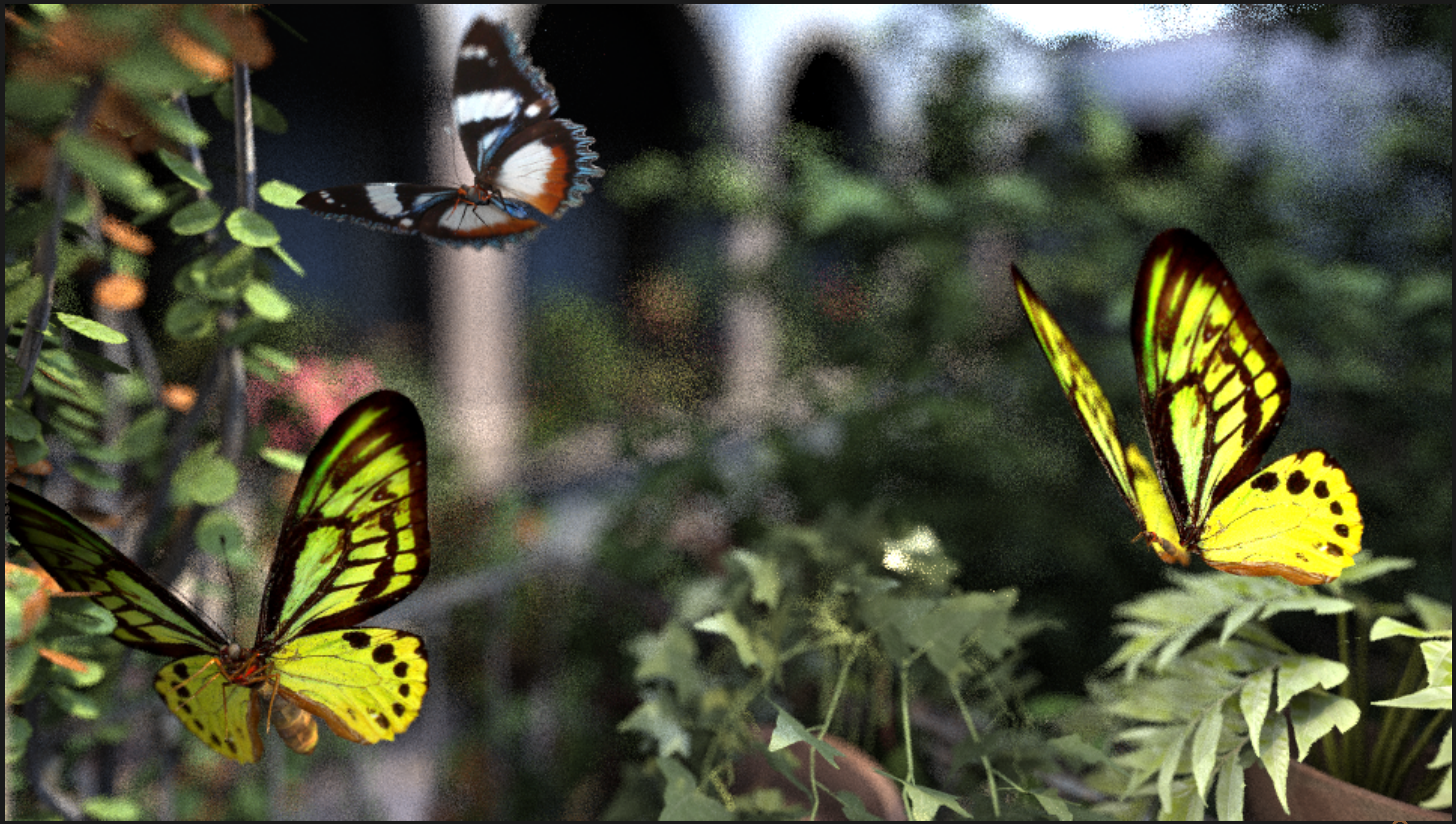
A lens gathers light and focuses it onto the sensor.

As a result, a single pixel <animate>

can receive light from many different parts of the scene, and we get a blurry background.

Depth of field



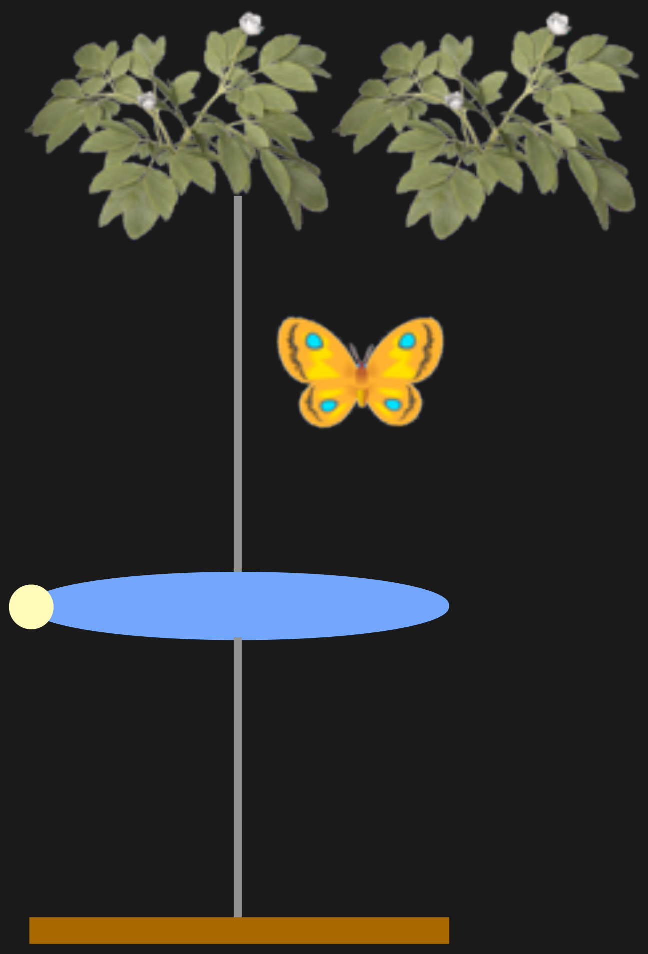


Depth of field, where out-of-focus parts of the scene are blurry,
is due to the apparent motion of objects based on the location at which the ray intersects the lens.

You can think of it as rendering a sharp pinhole image for every point on the lens,
and then taking their average.

For instance, this image... <next>

Depth of field



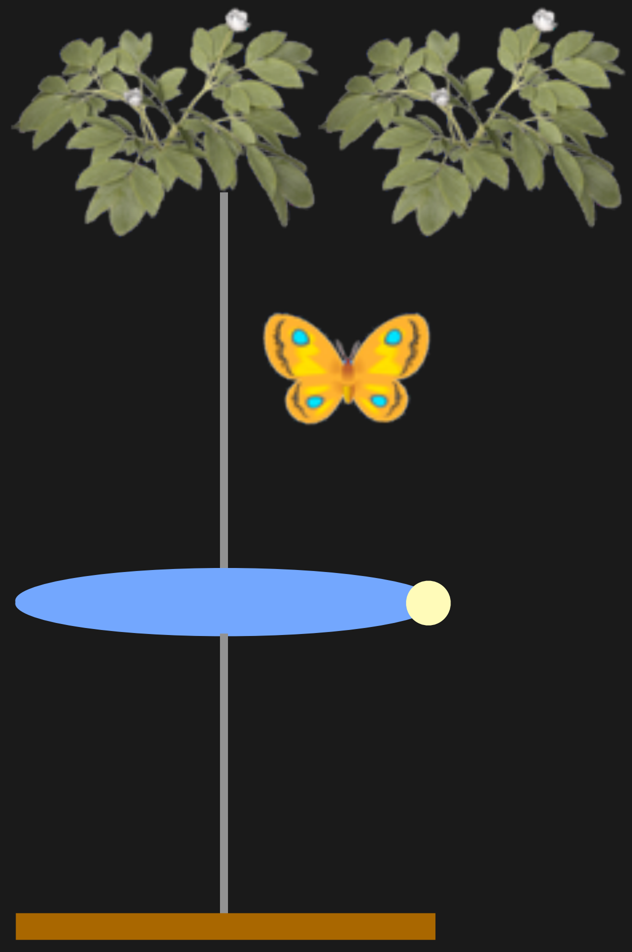
is formed by the set of all rays arriving on the left edge of the lens.

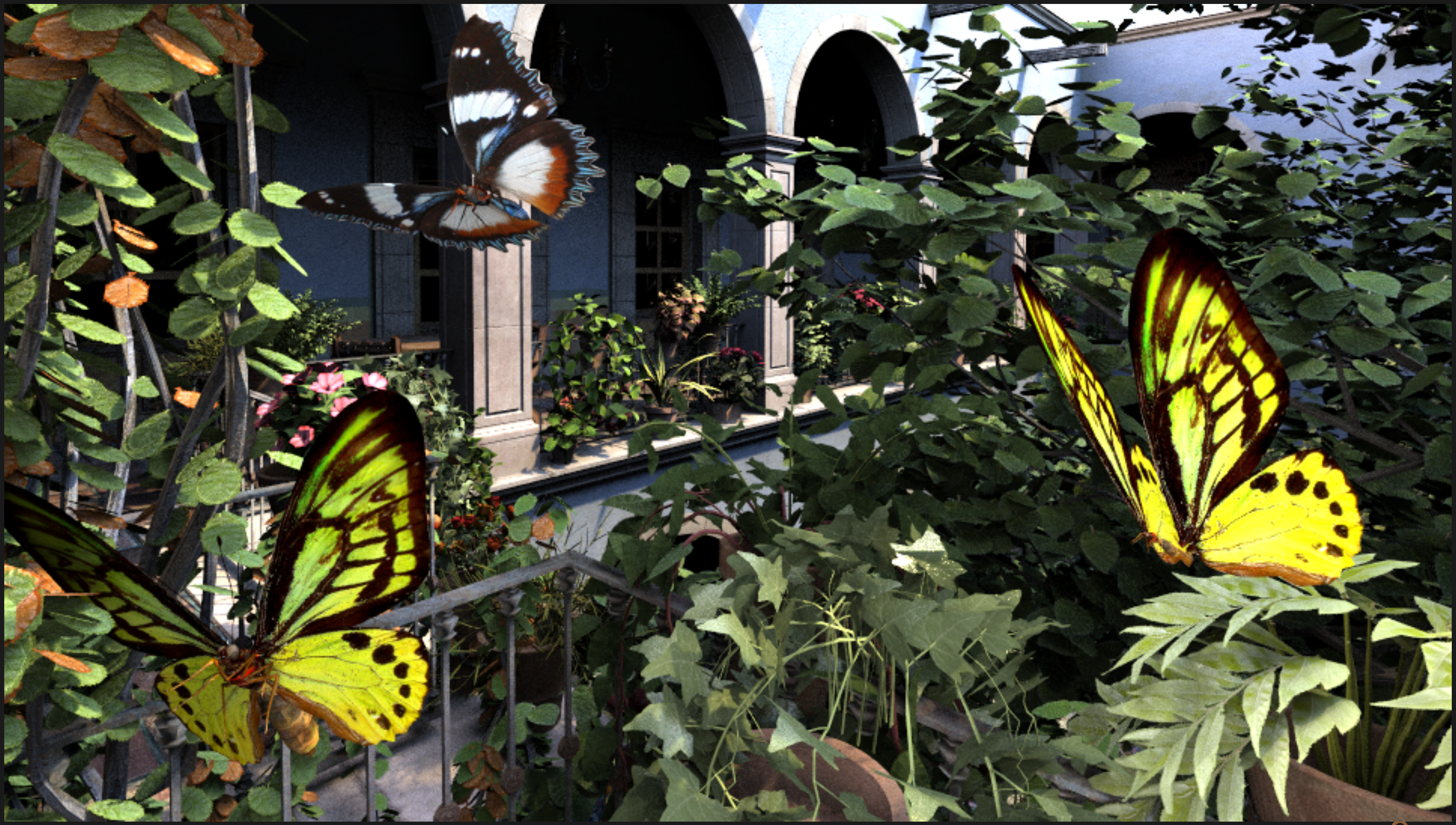
As we move our pinhole camera over the lens,
<animate>
The out-of-focus background appears to shift,
while the in-focus butterflies don’t move at all.

If we average these pinhole images together, <animate>
we get the nice image with in-focus butterflies and out-of-focus background.

Let’s take a different perspective on how defocus blur works.

Depth of field





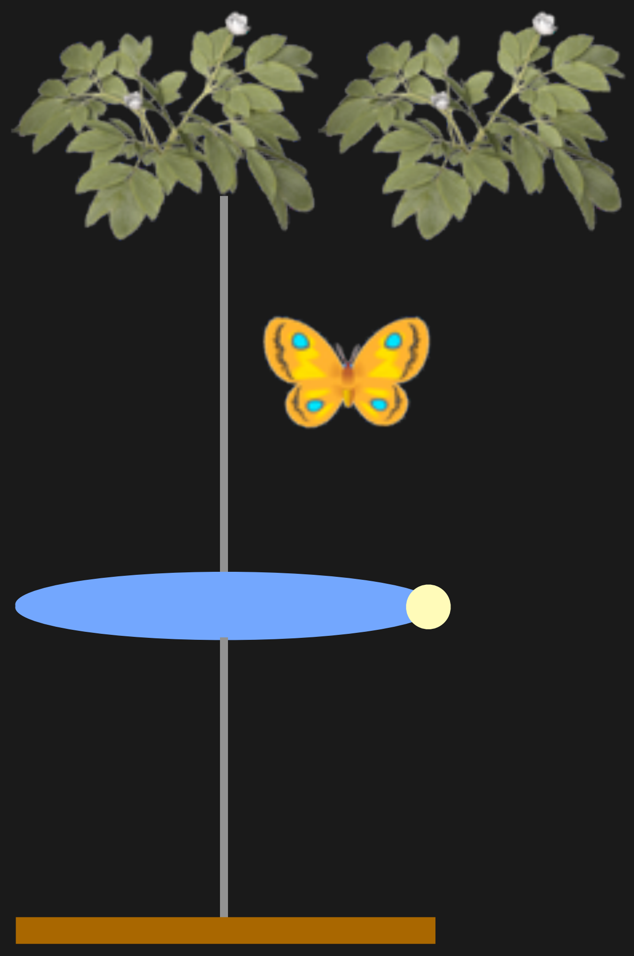
is formed by the set of all rays arriving on the left edge of the lens.

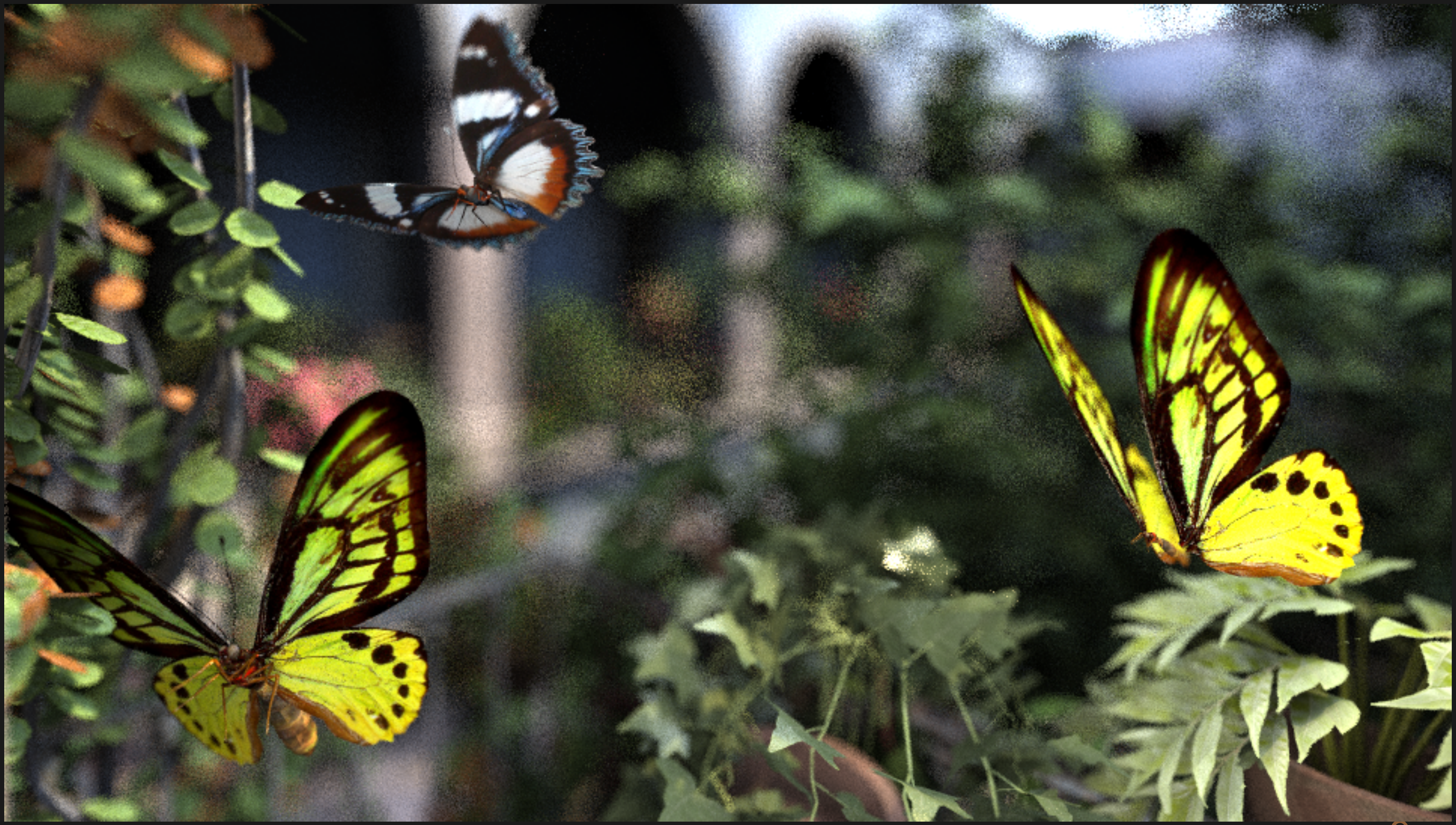
As we move our pinhole camera over the lens,
<animate>
The out-of-focus background appears to shift,
while the in-focus butterflies don’t move at all.

If we average these pinhole images together, <animate>
we get the nice image with in-focus butterflies and out-of-focus background.

Let’s take a different perspective on how defocus blur works.

Depth of field





is formed by the set of all rays arriving on the left edge of the lens.

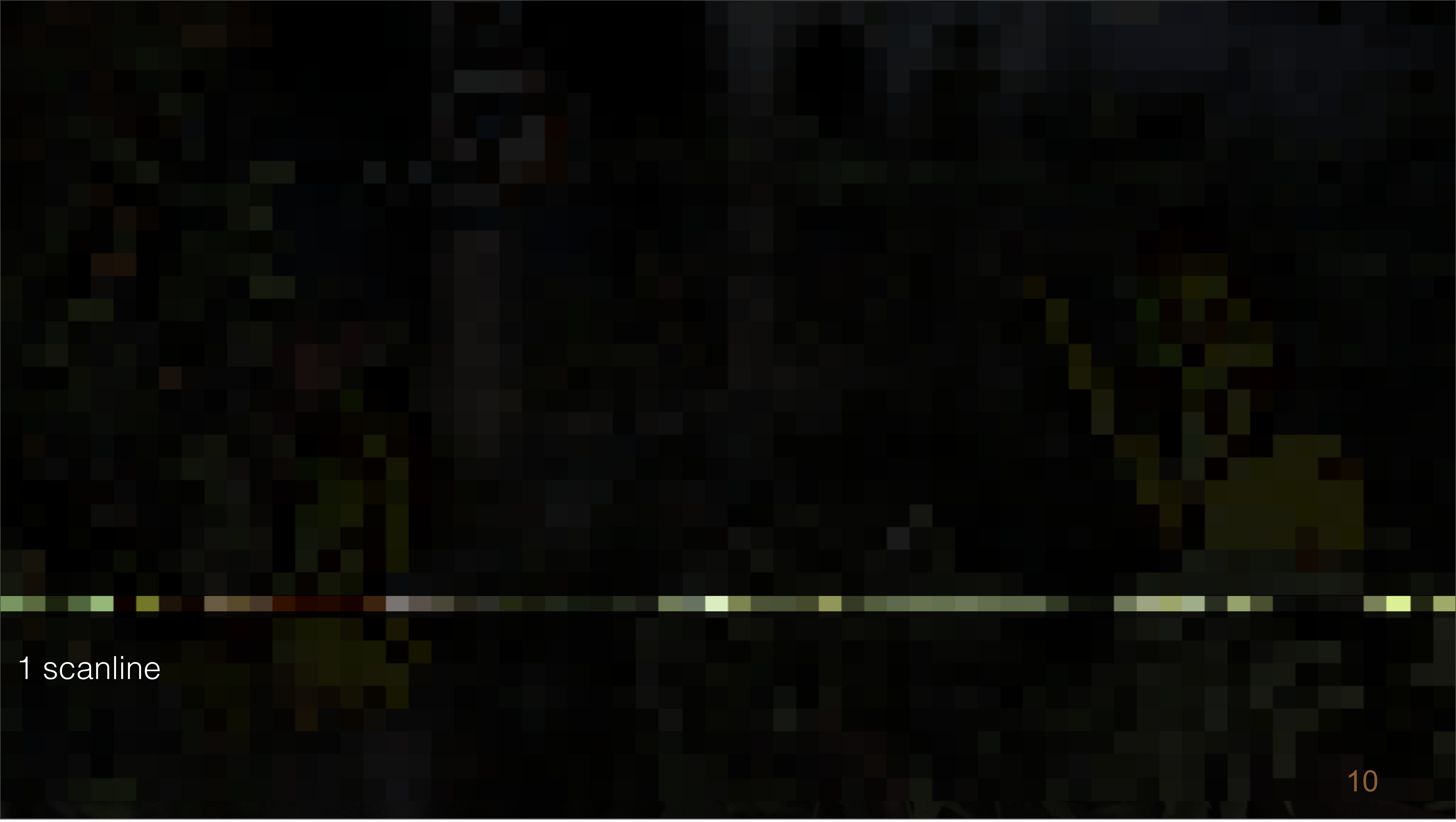
As we move our pinhole camera over the lens,
<animate>
The out-of-focus background appears to shift,
while the in-focus butterflies don’t move at all.

If we average these pinhole images together, <animate>
we get the nice image with in-focus butterflies and out-of-focus background.

Let’s take a different perspective on how defocus blur works.

Let’s take a look at just one scanline of our scene <animate>
and concentrate on this region <animate>

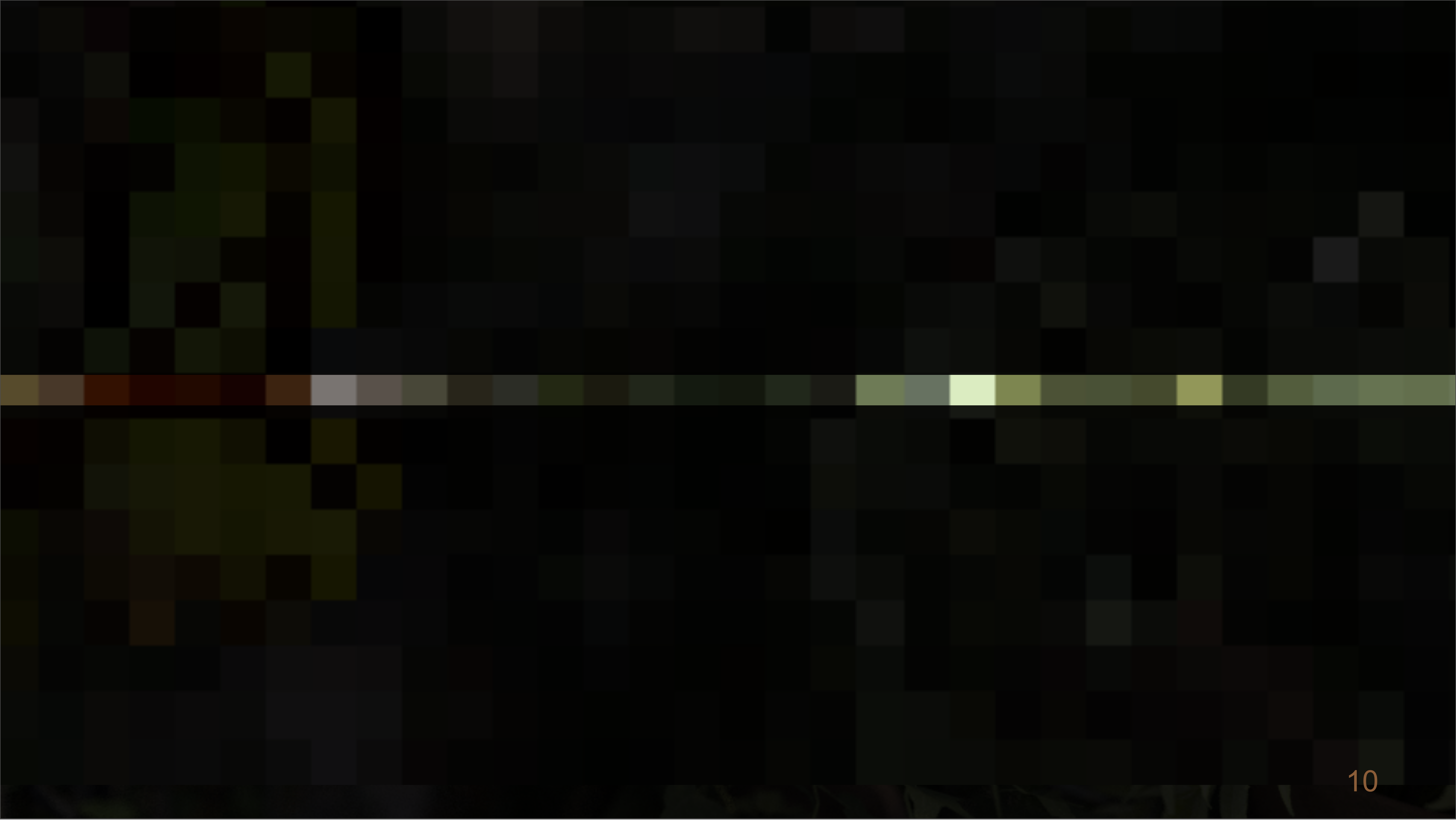
Imagine that instead of recording only the radiance of the ray arriving at each pixel,
we also write down where on the lens the ray hit.
<next>



1 scanline

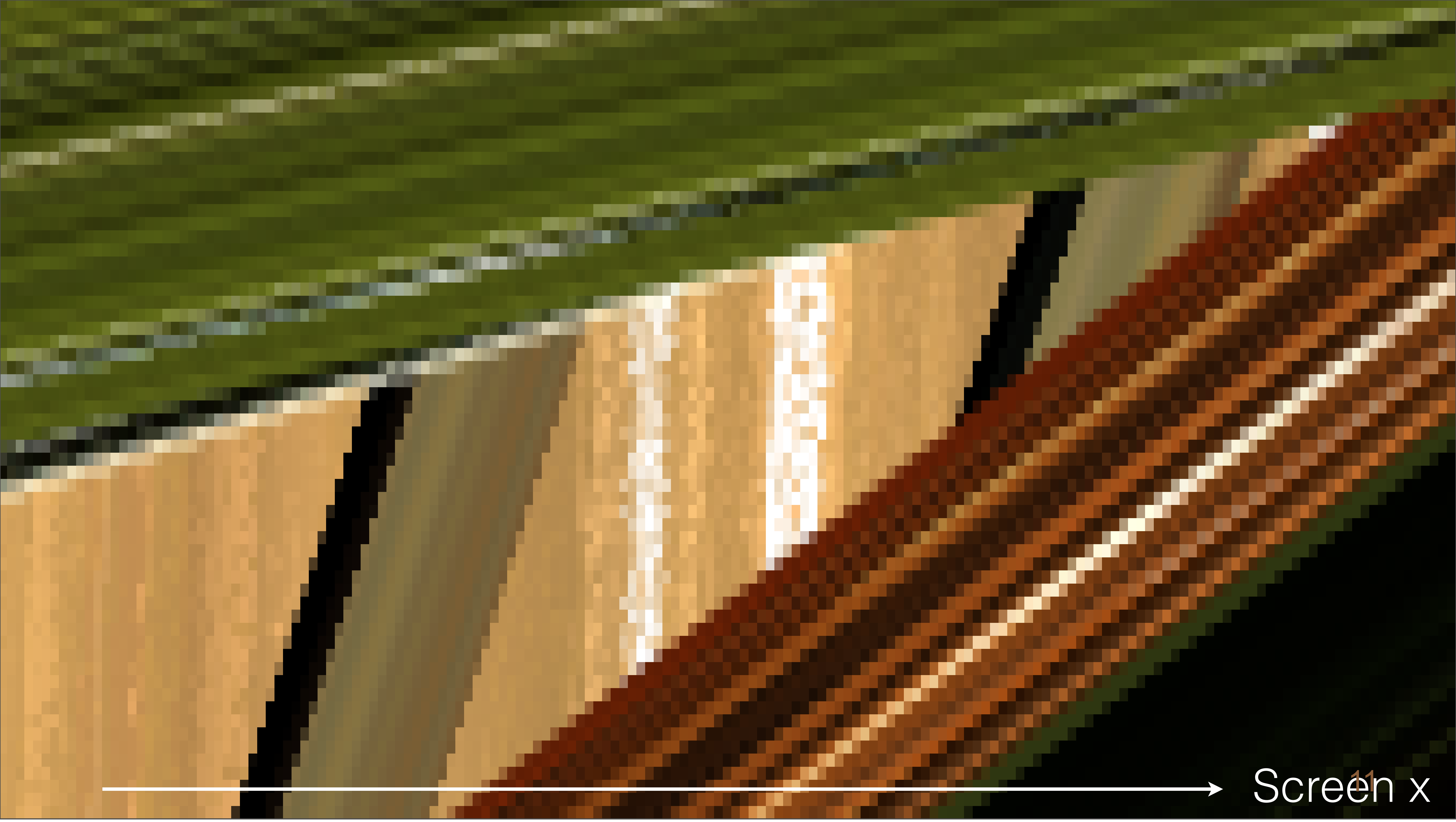
Let’s take a look at just one scanline of our scene <animate>
and concentrate on this region <animate>

Imagine that instead of recording only the radiance of the ray arriving at each pixel,
we also write down where on the lens the ray hit.
<next>



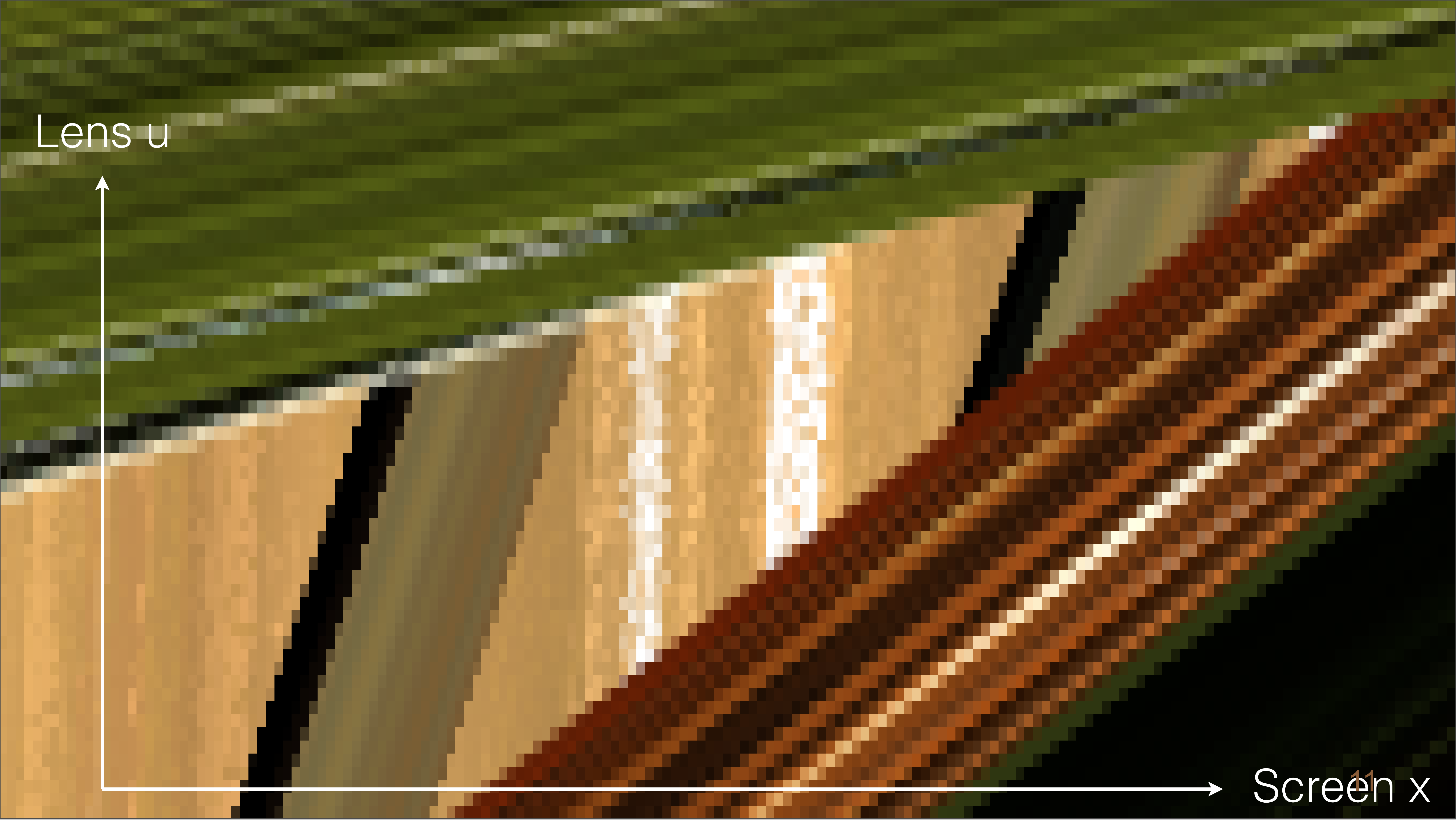
Let’s take a look at just one scanline of our scene <animate>
and concentrate on this region <animate>

Imagine that instead of recording only the radiance of the ray arriving at each pixel,
we also write down where on the lens the ray hit.
<next>



We’d get a 2D diagram like this:

The horizontal axis still corresponds to pixels across the screen.
<animate>
But we added a vertical axis that shows the lens coordinate of each ray.

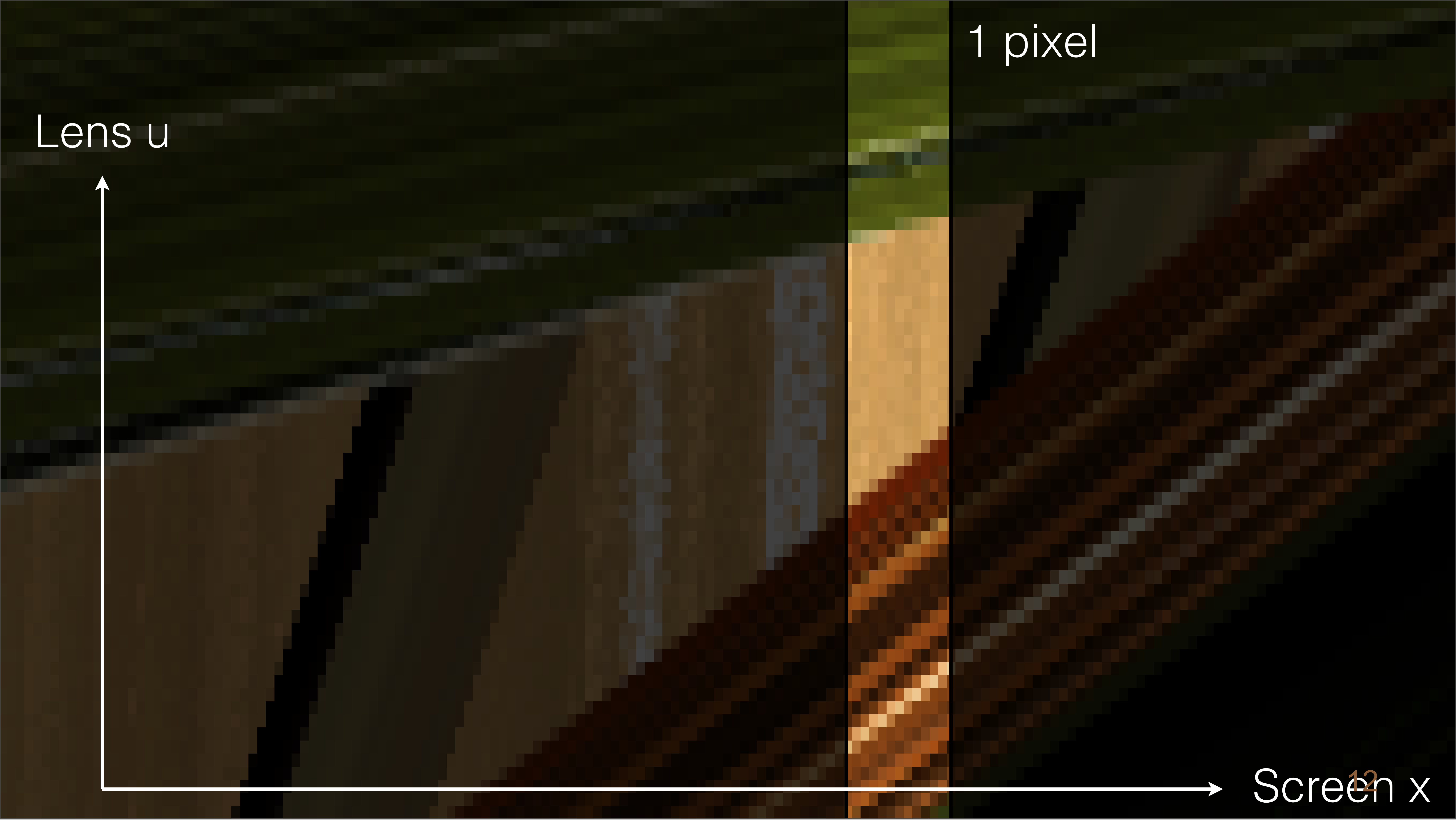


We’d get a 2D diagram like this:

The horizontal axis still corresponds to pixels across the screen.

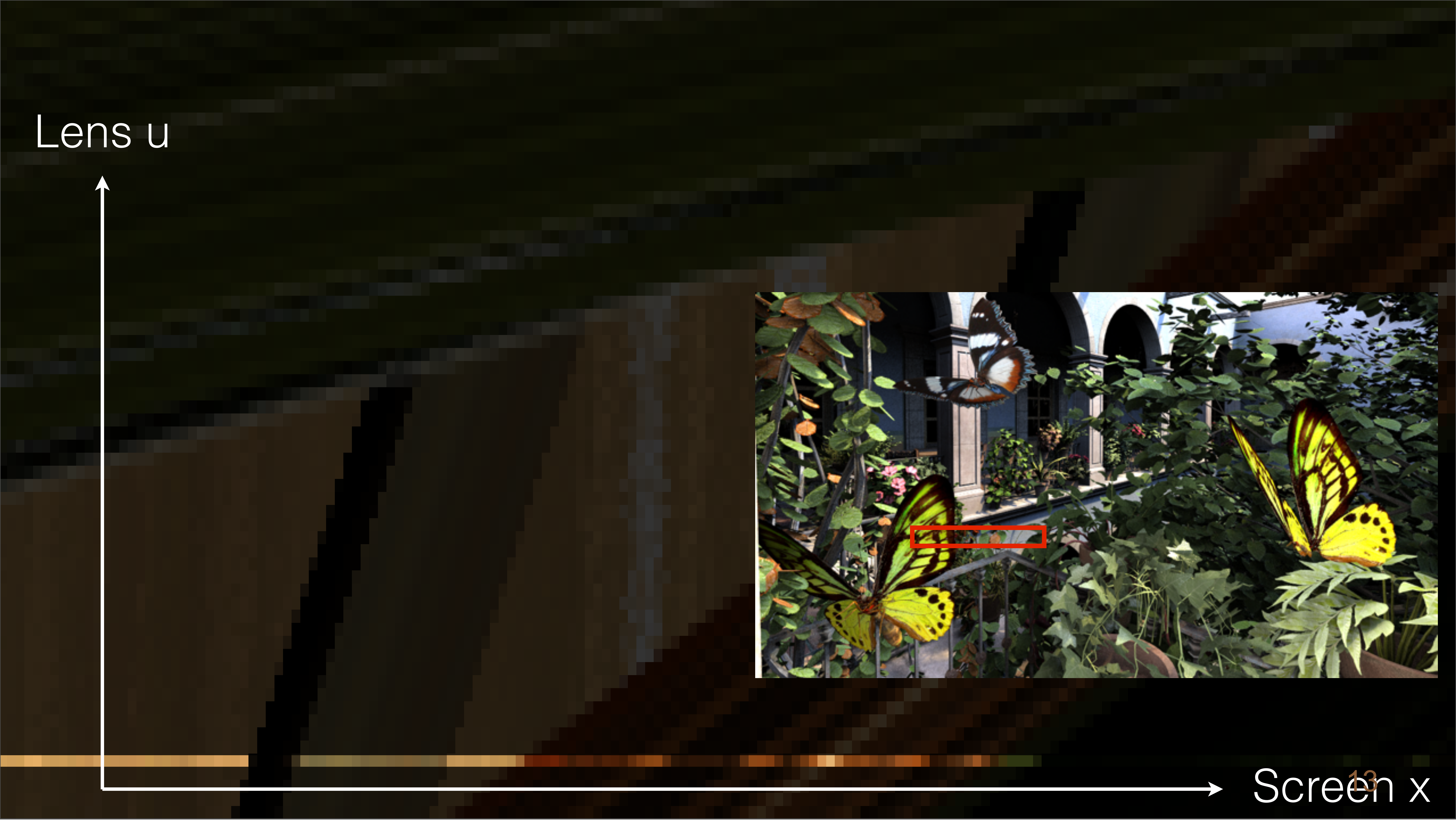
<animate>

But we added a vertical axis that shows the lens coordinate of each ray.



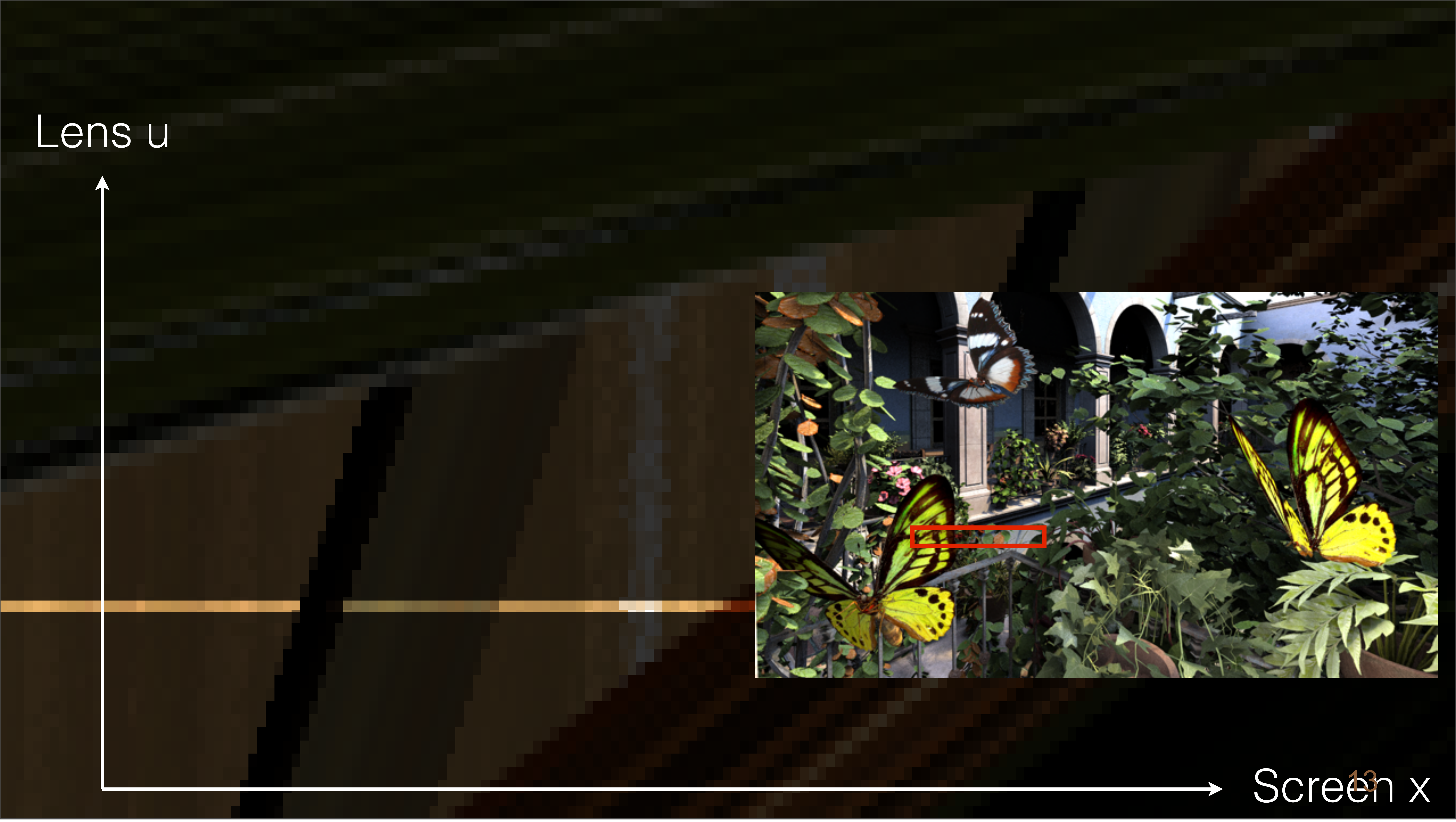
Each column in this diagram corresponds to a pixel,

and each row corresponds
<next>



to one of the pinhole images from a particular position on the lens that I showed earlier.

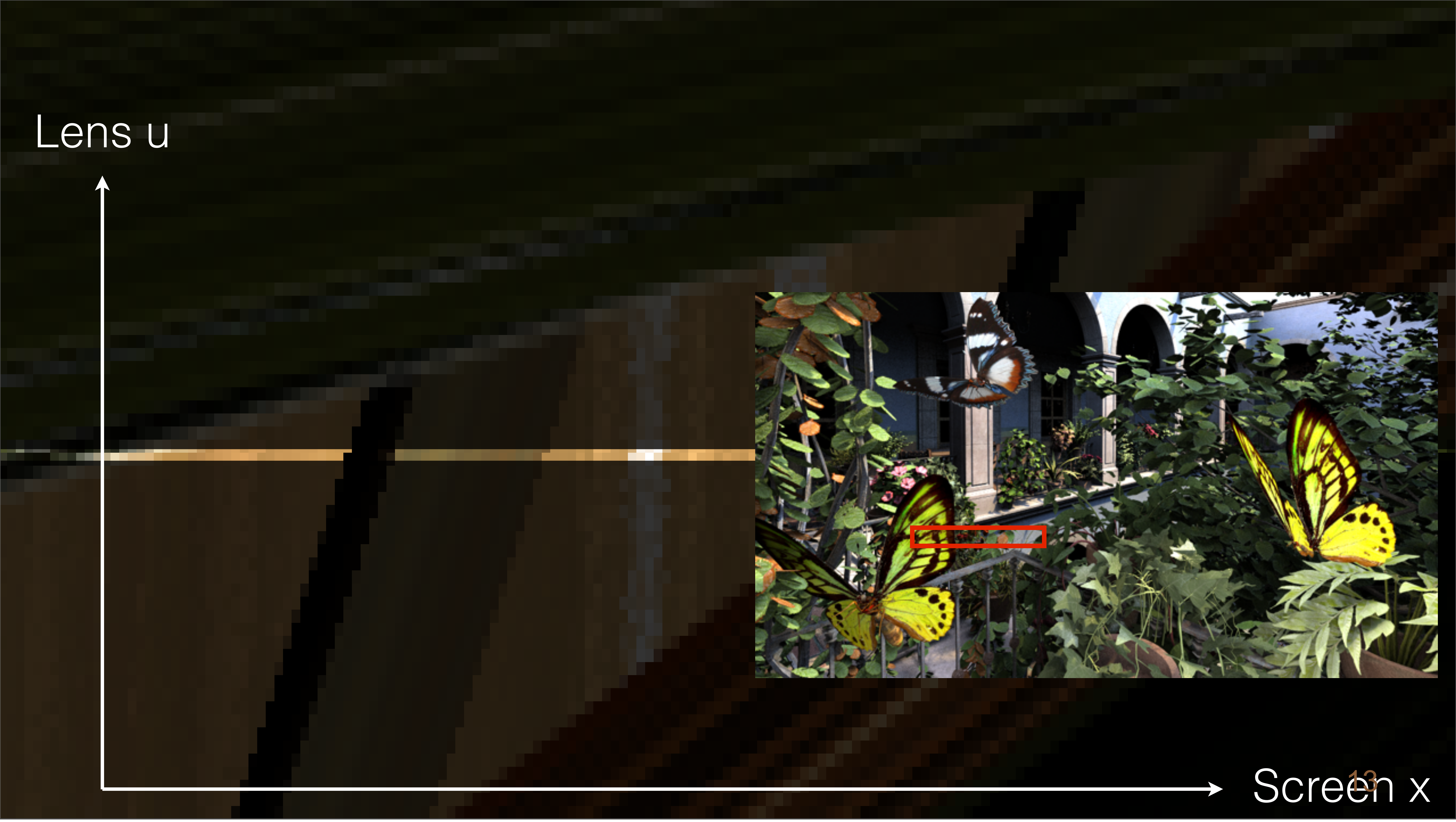
And as we move <animate> over the lens, we get various pinhole images.
<animate>
<animate>



Friday, August 26, 11

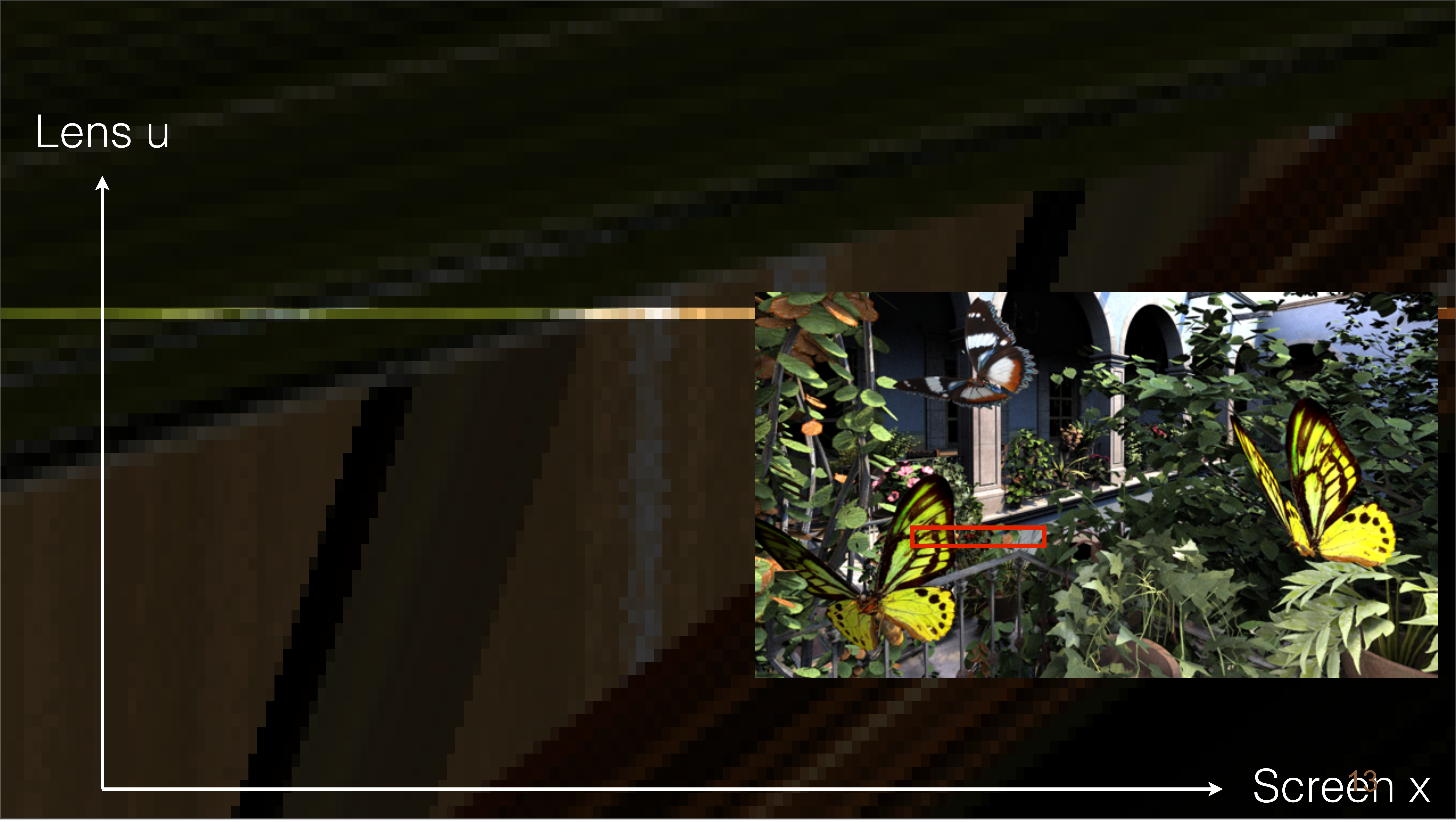
to one of the pinhole images from a particular position on the lens that I showed earlier.

And as we move <animate> over the lens, we get various pinhole images.
<animate>
<animate>



to one of the pinhole images from a particular position on the lens that I showed earlier.

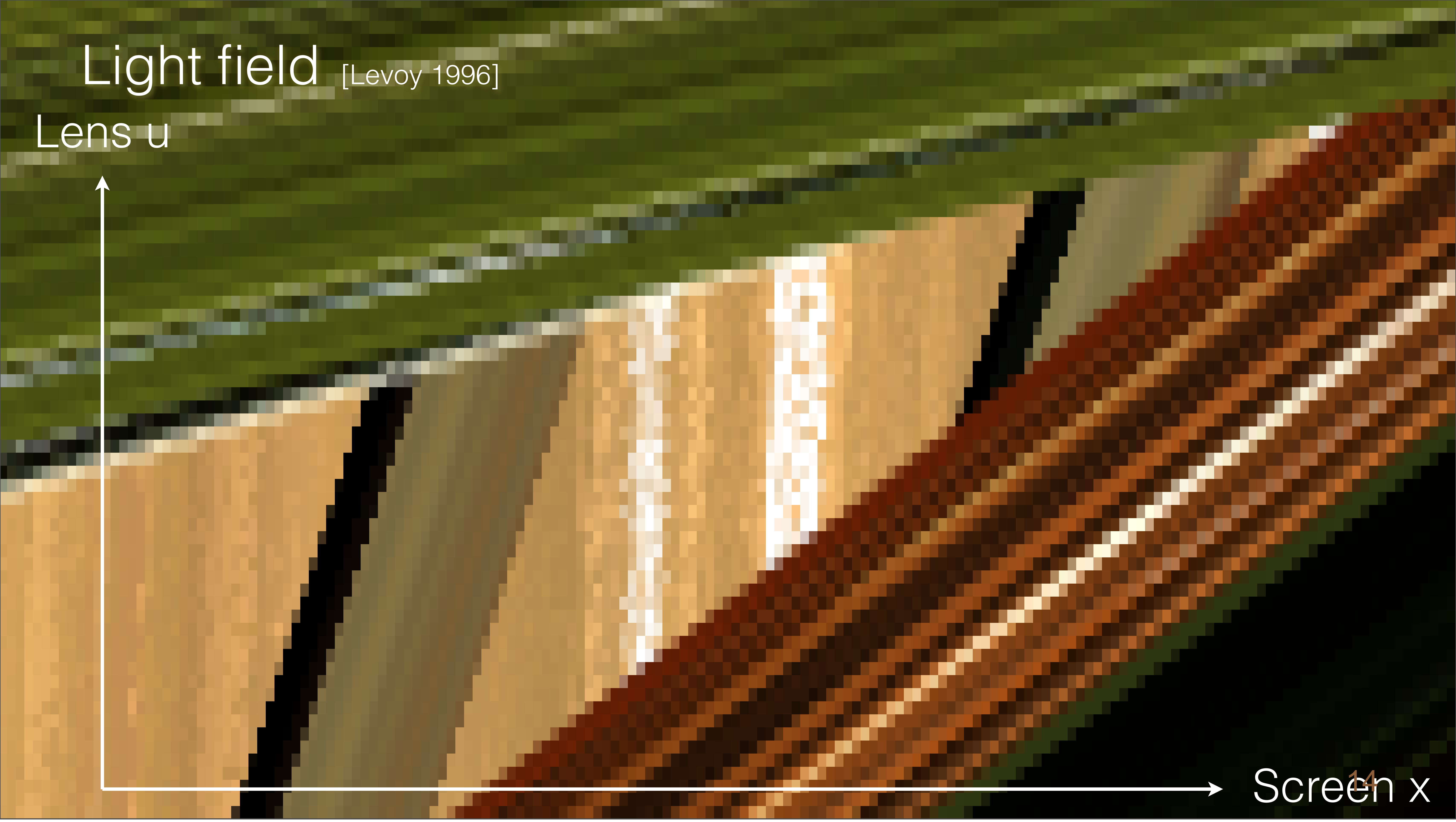
And as we move <animate> over the lens, we get various pinhole images.
<animate>
<animate>



to one of the pinhole images from a particular position on the lens that I showed earlier.

And as we move <animate> over the lens, we get various pinhole images.
<animate>
<animate>

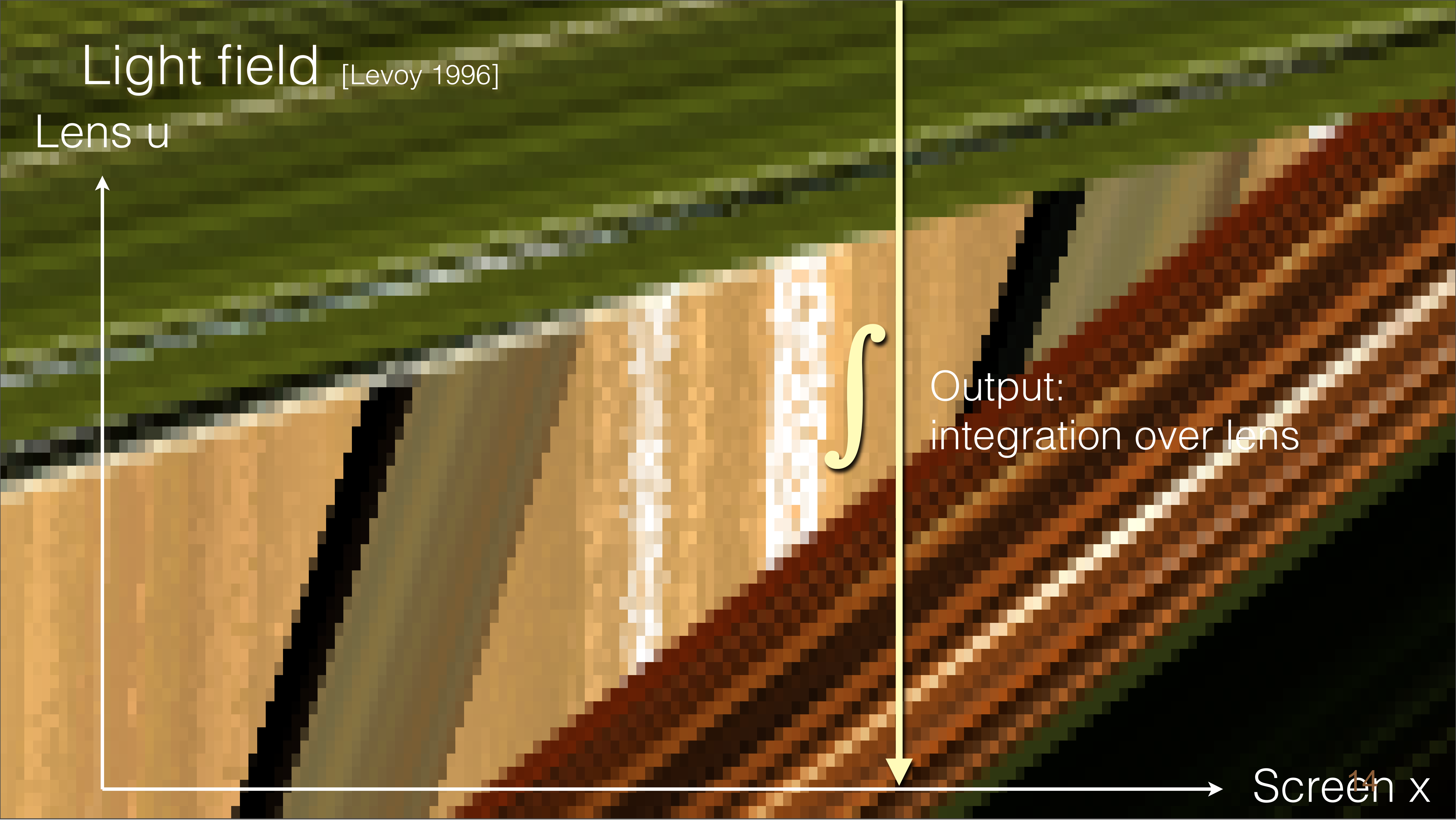
Light field [Levoy 1996]



This function of x and u is called the light field

<animate>

and the output image with blurry defocus is the result of integrating down the vertical dimension: in other words, over the aperture of the lens.



Light field [Levoy 1996]

Lens u

∫

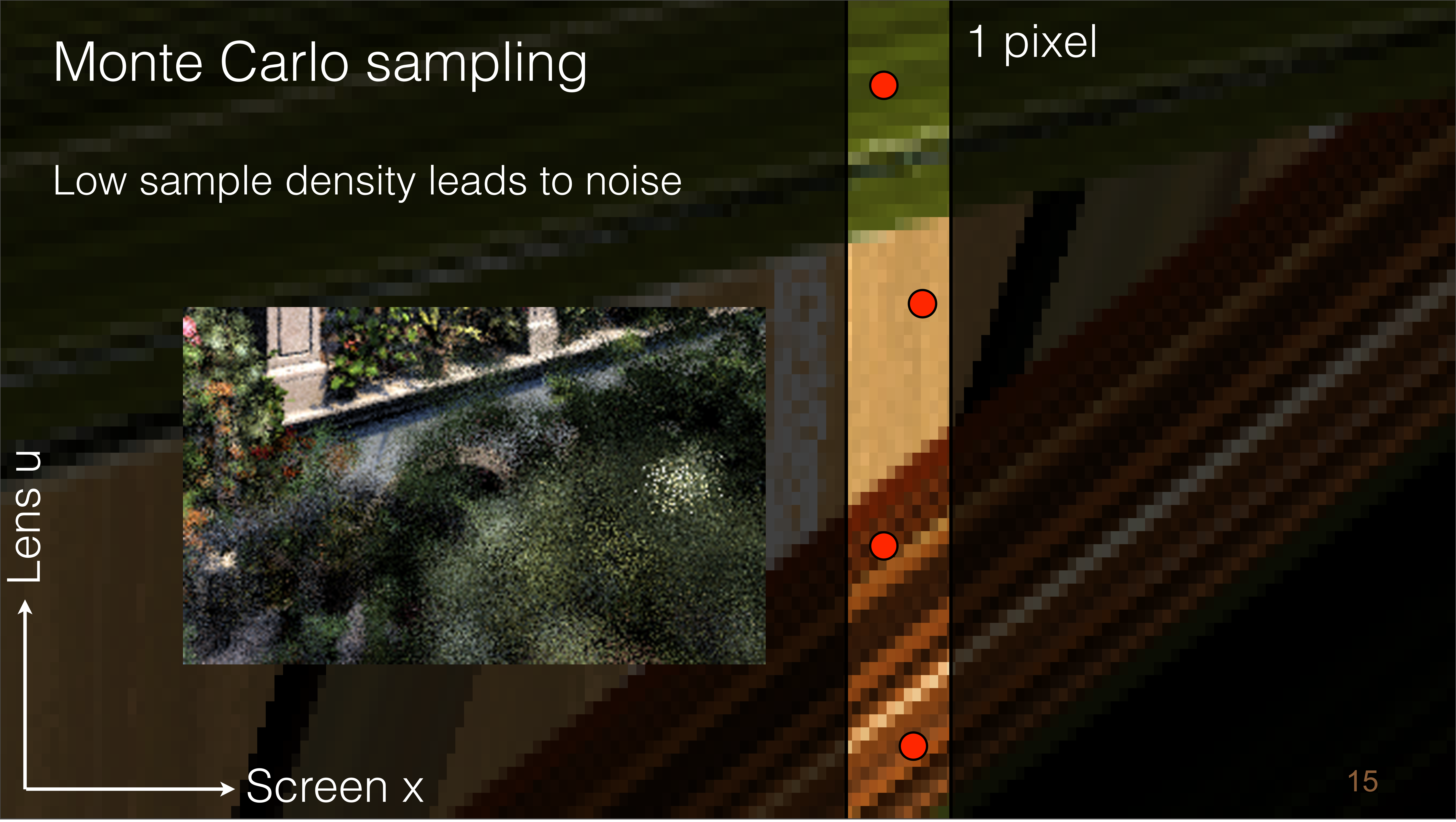
Output:
integration over lens

Screen x

This function of x and u is called the light field
<animate>
and the output image with blurry defocus is the result of integrating down the vertical dimension: in other words, over the aperture of the lens.

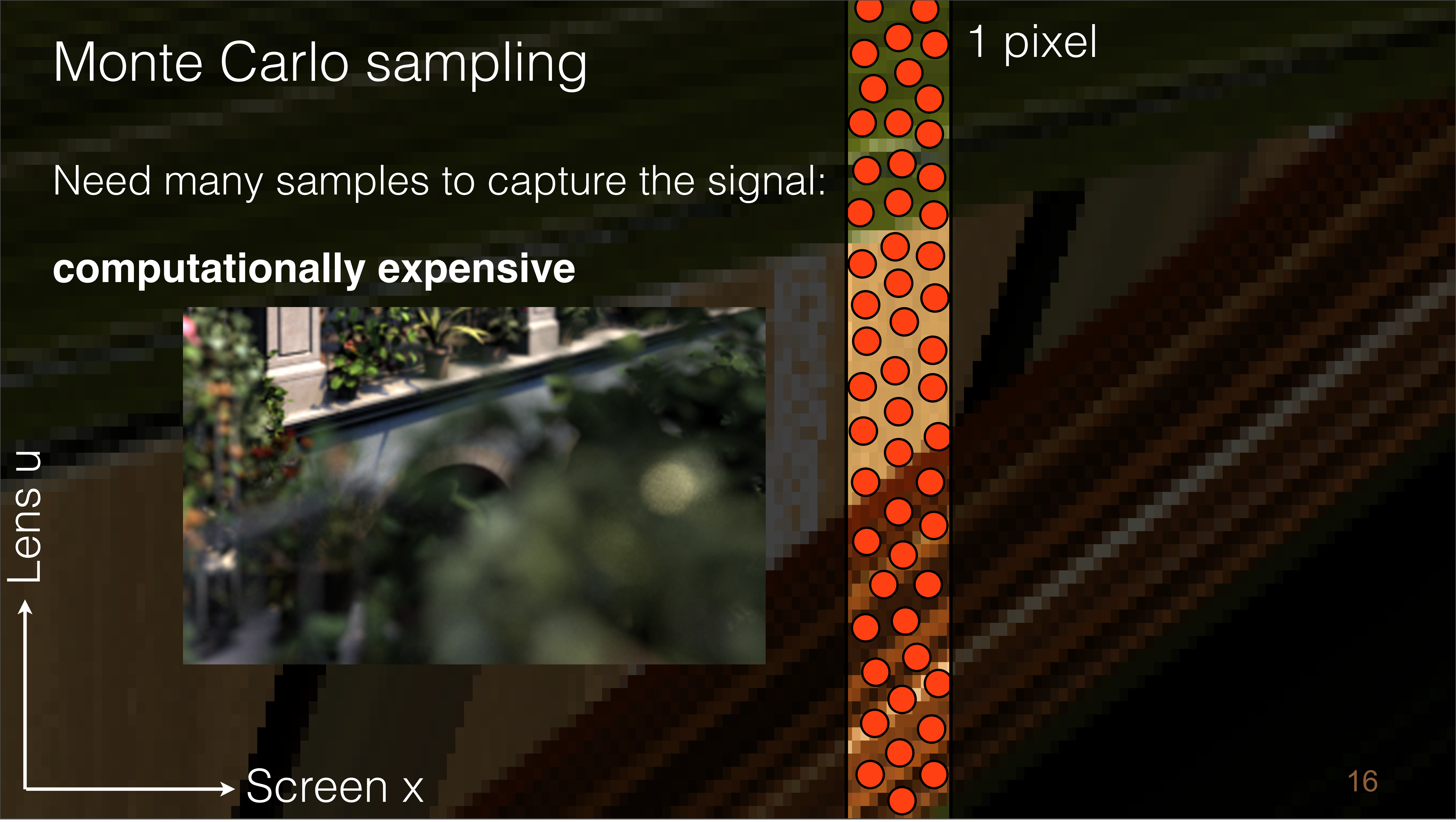
Monte Carlo sampling

Low sample density leads to noise



The standard Monte-Carlo algorithm estimates this integral by taking samples of this function and summing their radiances down a column.

But if we only have say, 4 samples, we get a pretty noisy estimate.



Monte Carlo sampling

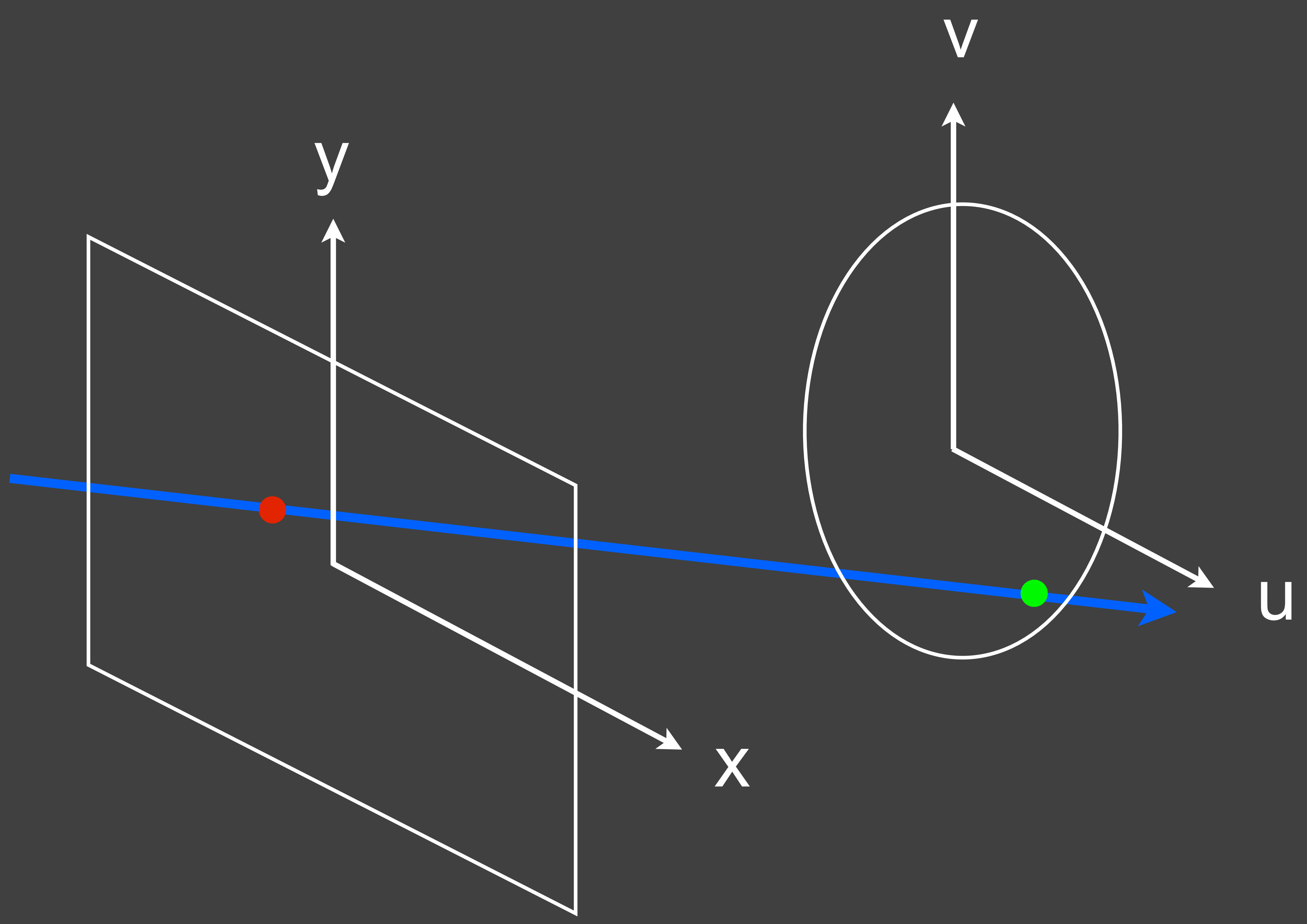
Need many samples to capture the signal:

computationally expensive

With many samples, we get a much better estimate.
But sampling is expensive!
Each sample corresponds to shooting a ray, intersecting it with potentially moving geometry, and shading.

<pause>
Ok, so that was a look at the light field in the case of depth of field in flatland.

Temporal light fields



For a 3D scene, we integrate over two lens coordinates, u and v, to produce a 2D image.
So the traditional light field is a 4D function.

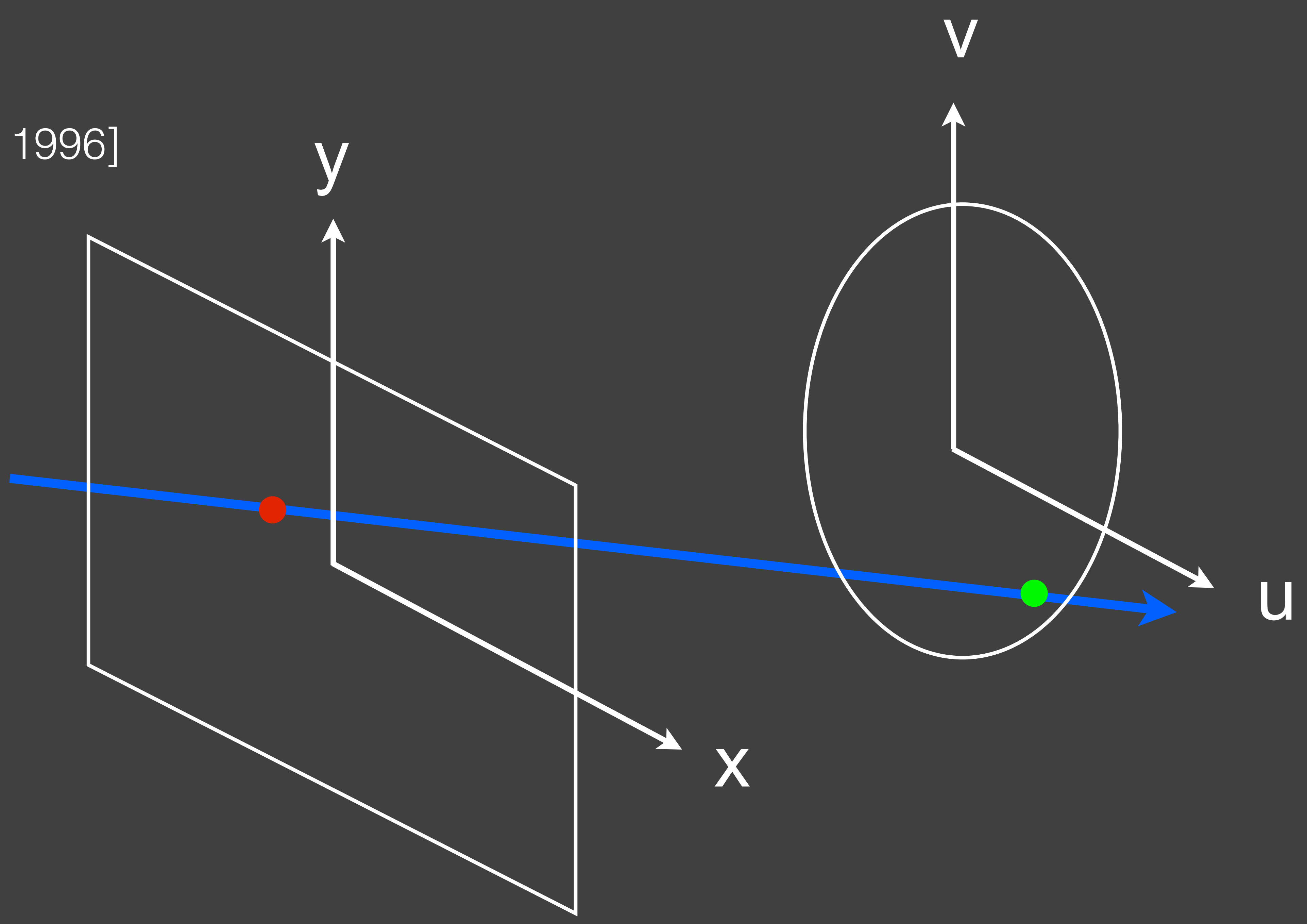
<animate>

In our case, we also deal with moving geometry, so our *temporal* light field is a 5D function <animate>
and we integrate over the u, v and t dimensions.

Temporal light fields

Traditional light field is 4D [Levoy 1996]

x,y over sensor (2D)
u,v over lens (2D)



For a 3D scene, we integrate over two lens coordinates, u and v, to produce a 2D image.
So the traditional light field is a 4D function.

<animate>

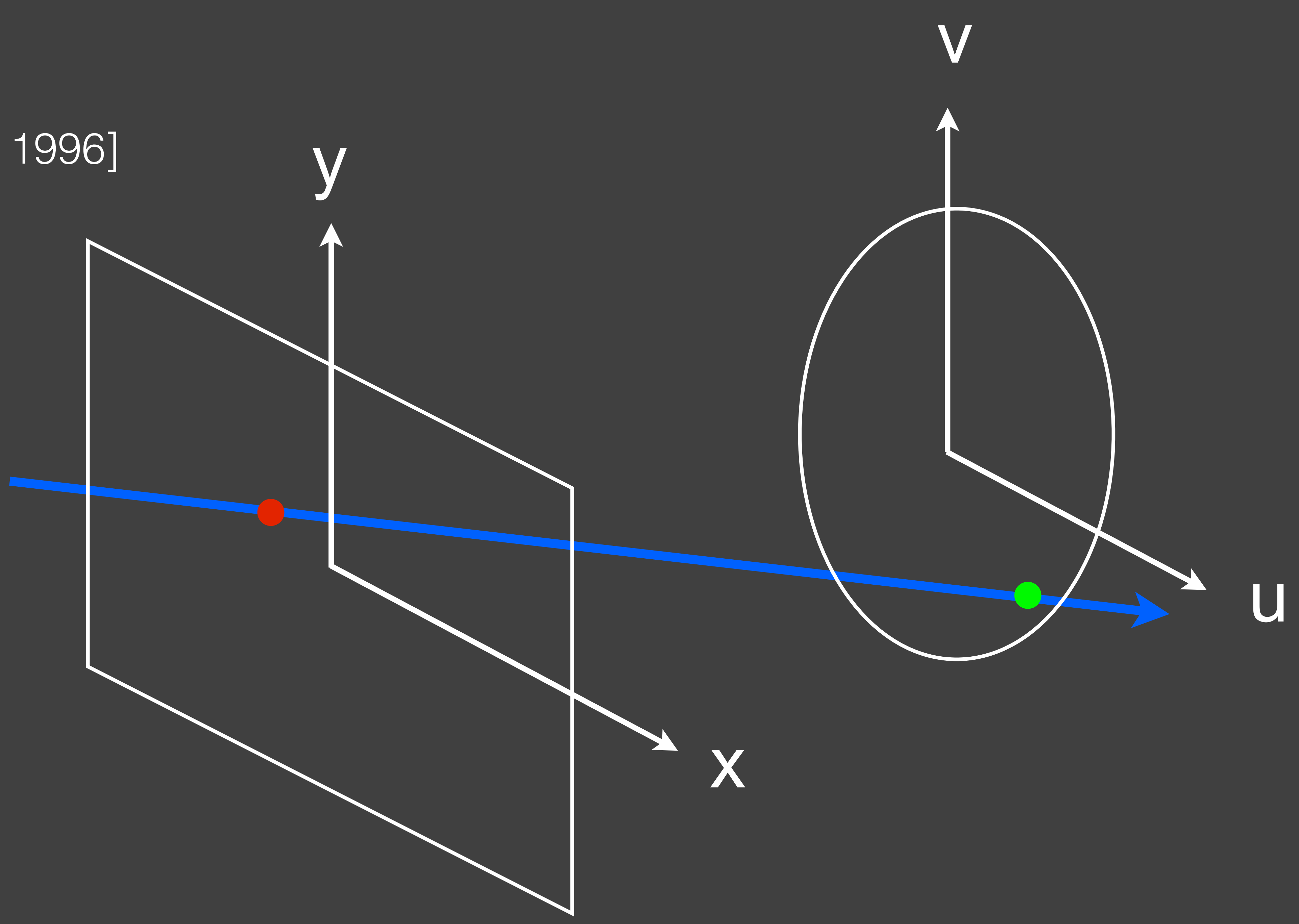
In our case, we also deal with moving geometry, so our *temporal* light field is a 5D function <animate>
and we integrate over the u, v and t dimensions.

Temporal light fields

Traditional light field is 4D [Levoy 1996]

x,y over sensor (2D)
u,v over lens (2D)

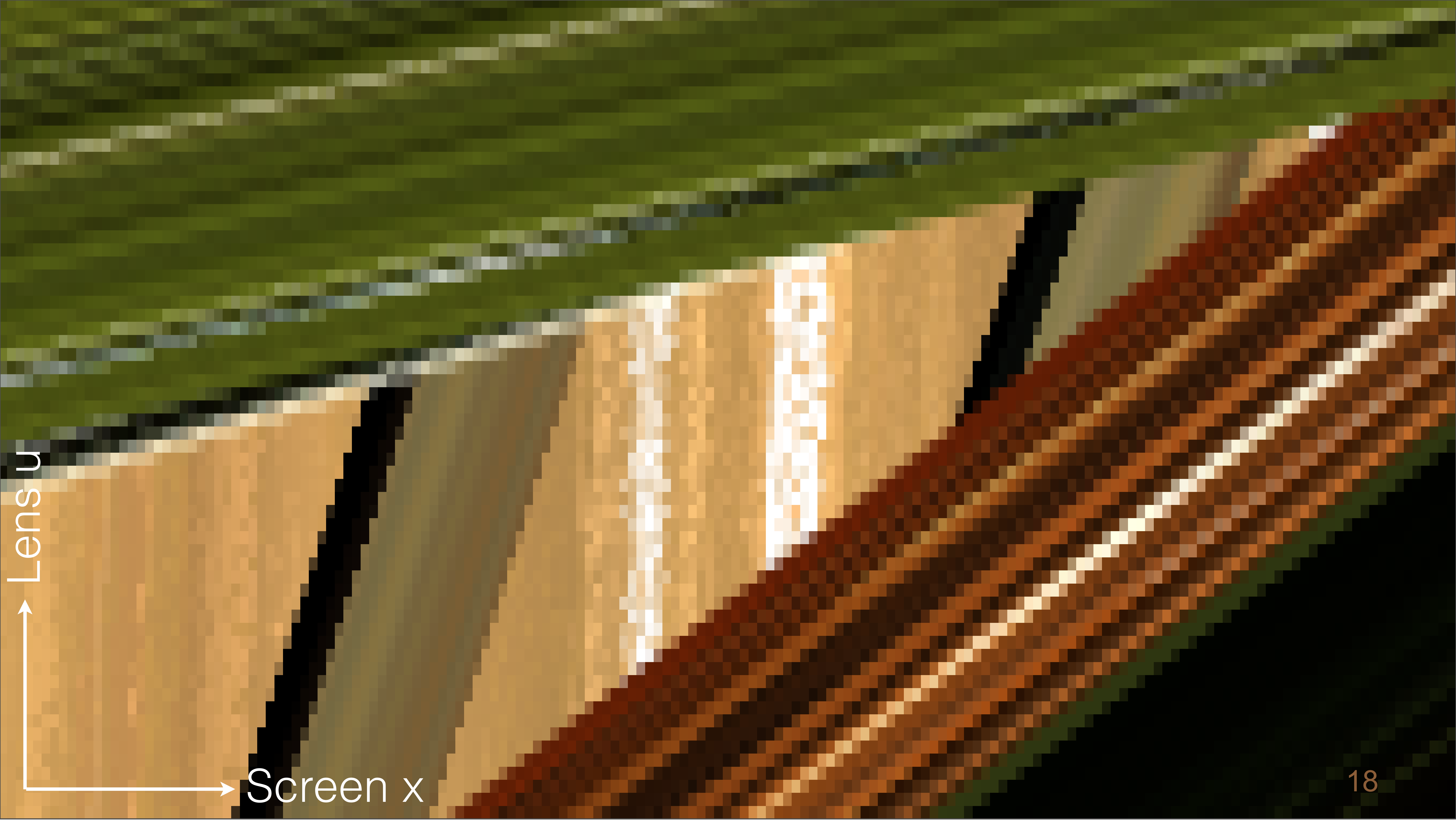
Add time dimension for moving geometry (5D)



For a 3D scene, we integrate over two lens coordinates, u and v, to produce a 2D image.
So the traditional light field is a 4D function.

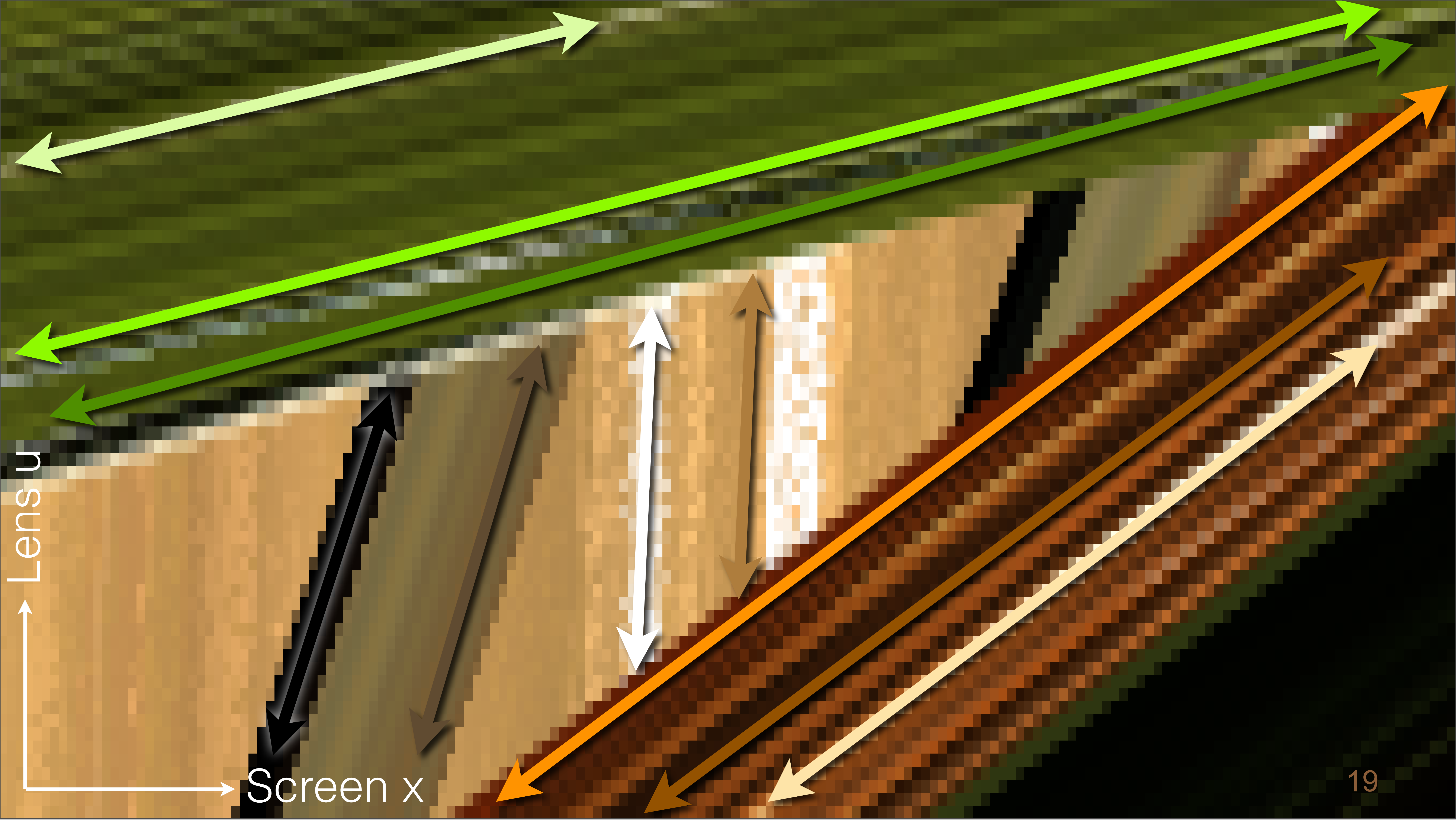
<animate>

In our case, we also deal with moving geometry, so our *temporal* light field is a 5D function <animate>
and we integrate over the u, v and t dimensions.



Let’s go back to our example with the 2D x-u light field.

It’s pretty obvious that it’s not some random signal: it’s very structured.



In particular, the function is approximately constant along these lines of various slopes.

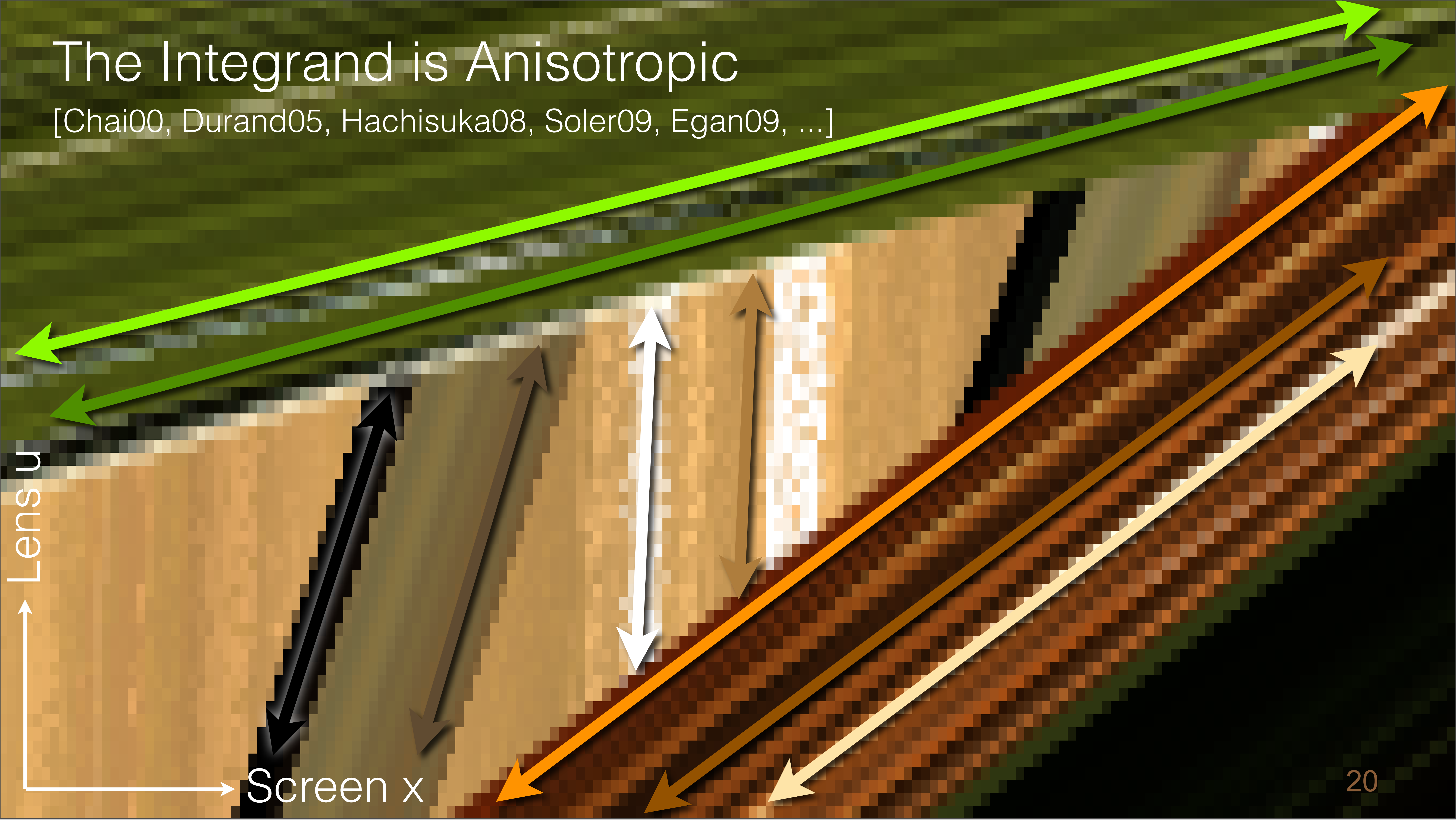
Each line corresponds to a point on a 3D object and shows how its image moves as you change the viewpoint over the lens.

In-focus points are vertical: they don’t move as you change the viewpoint.
More slanted lines are farther out of focus.

If you think about it, that makes sense: we are integrating down columns.
The more slanted the line, the more it gets smeared horizontally over the image when we integrate.

The Integrand is Anisotropic

[Chai00, Durand05, Hachisuka08, Soler09, Egan09, ...]



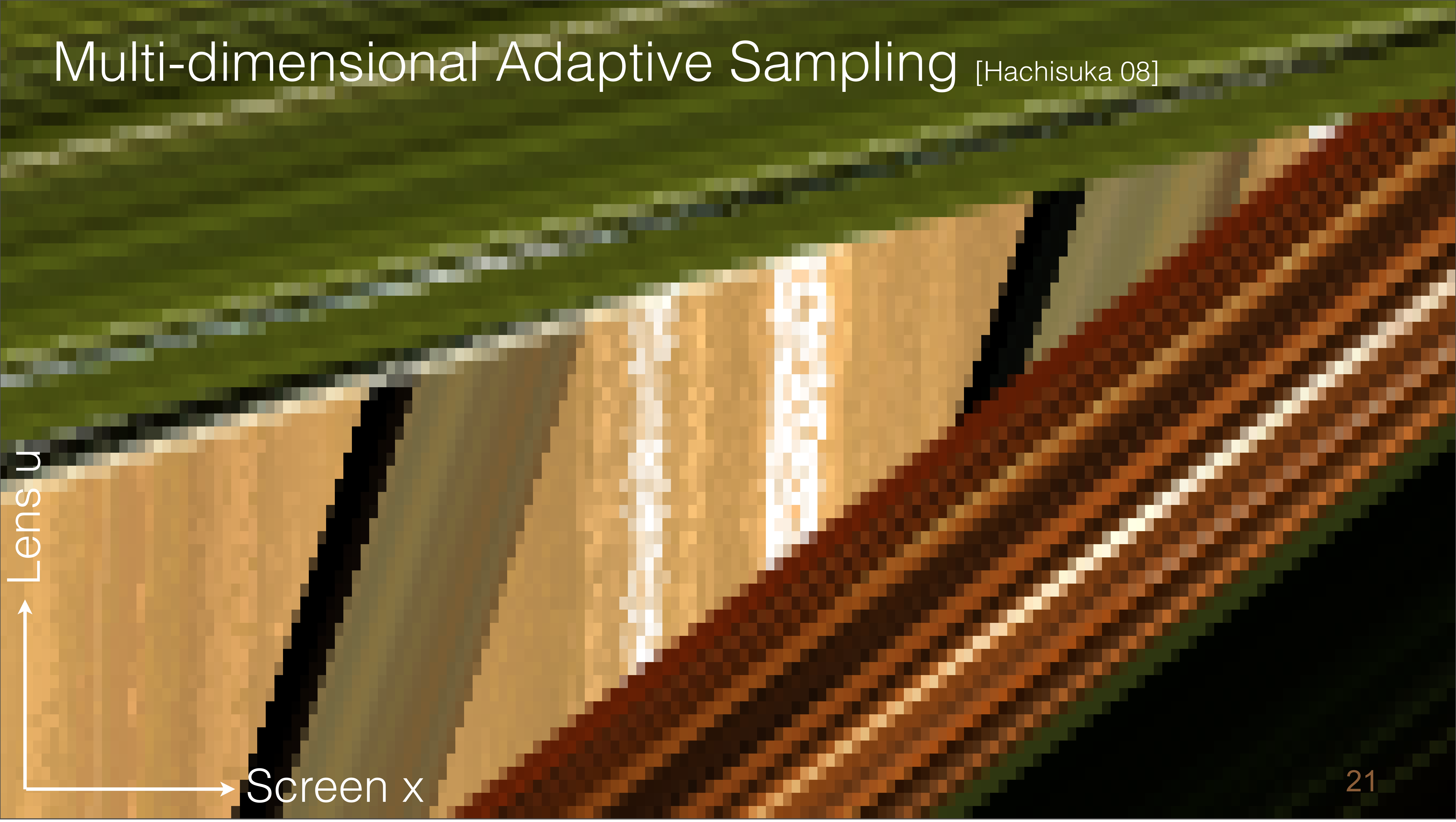
The structured nature of the light field suggests that we should be able to get away with taking only a small number of samples and reconstructing the signal knowing this structure.

Indeed, we weren't the first ones to do this.

A number of researchers have observed that the light field is highly anisotropic and took advantage of this to accelerate rendering.

Let's take a close look at what they do.

Multi-dimensional Adaptive Sampling [Hachisuka 08]

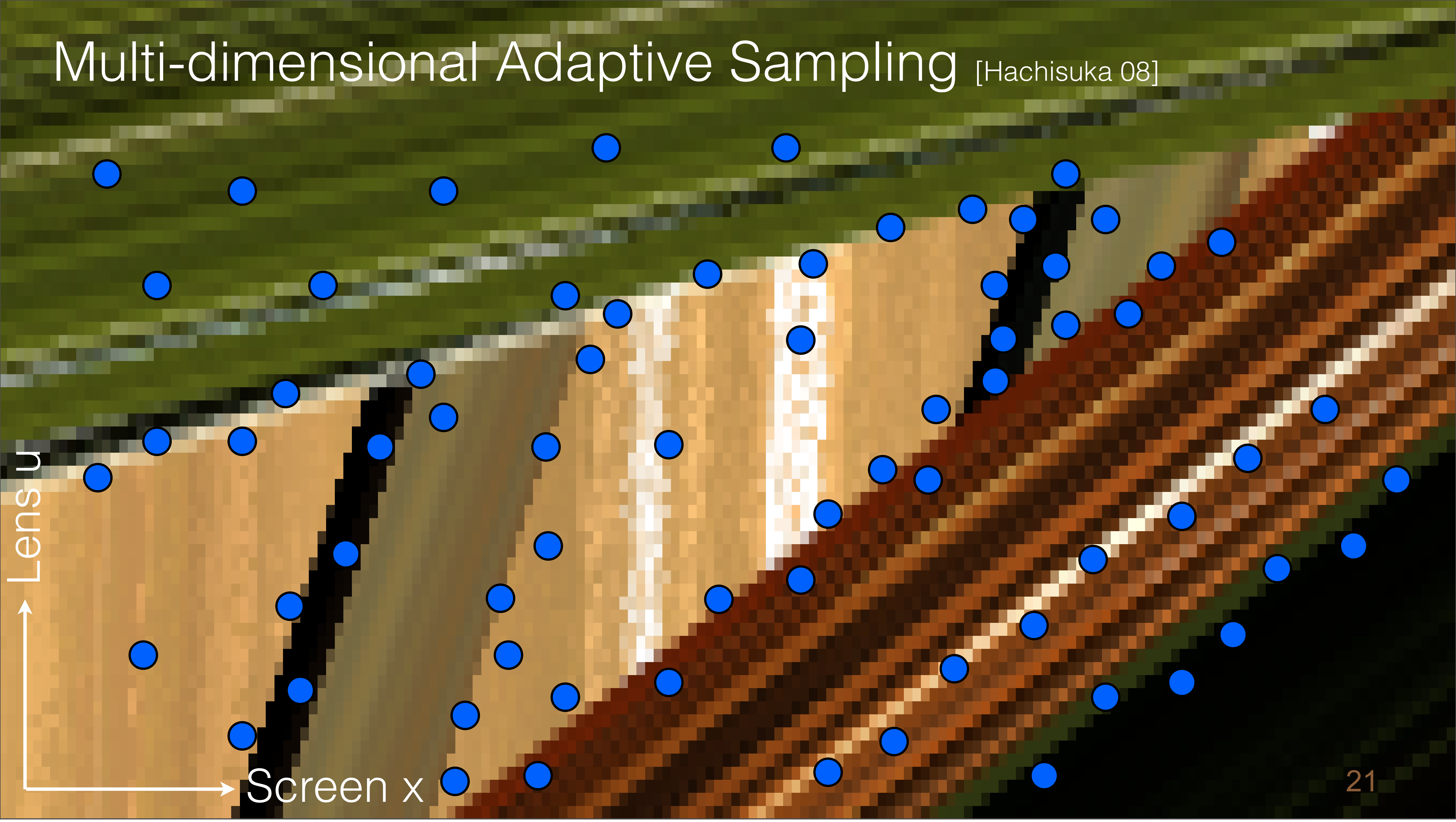


Multi-dimensional adaptive sampling, by Hachisuka and colleagues,
take a data driven approach to determining this structure.

They perform adaptive sampling <animate>,
which places samples near high contrast boundaries.

From these sample radiances, <animate>
they can deduce the local anisotropy in the light field <animate>

Unfortunately, because the approach is data-driven, texture and noise can throw off the estimator.
<shortpause>
And, their method does not explicitly handle visibility.



Multi-dimensional Adaptive Sampling [Hachisuka 08]

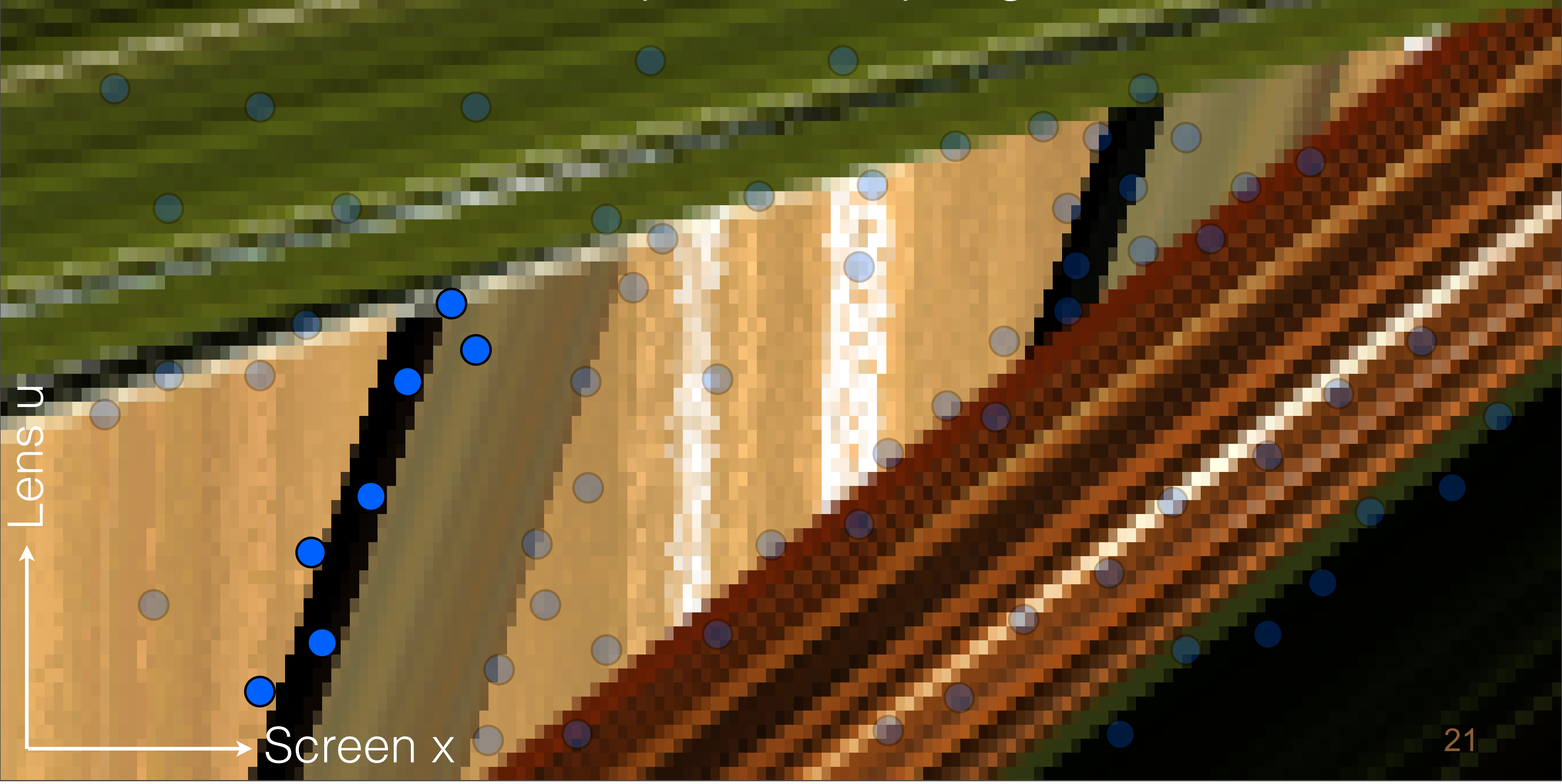
Multi-dimensional adaptive sampling, by Hachisuka and colleagues,
take a data driven approach to determining this structure.

They perform adaptive sampling <animate>,
which places samples near high contrast boundaries.

From these sample radiances, <animate>
they can deduce the local anisotropy in the light field <animate>

Unfortunately, because the approach is data-driven, texture and noise can throw off the estimator.
<shortpause>
And, their method does not explicitly handle visibility.

Multi-dimensional Adaptive Sampling [Hachisuka 08]

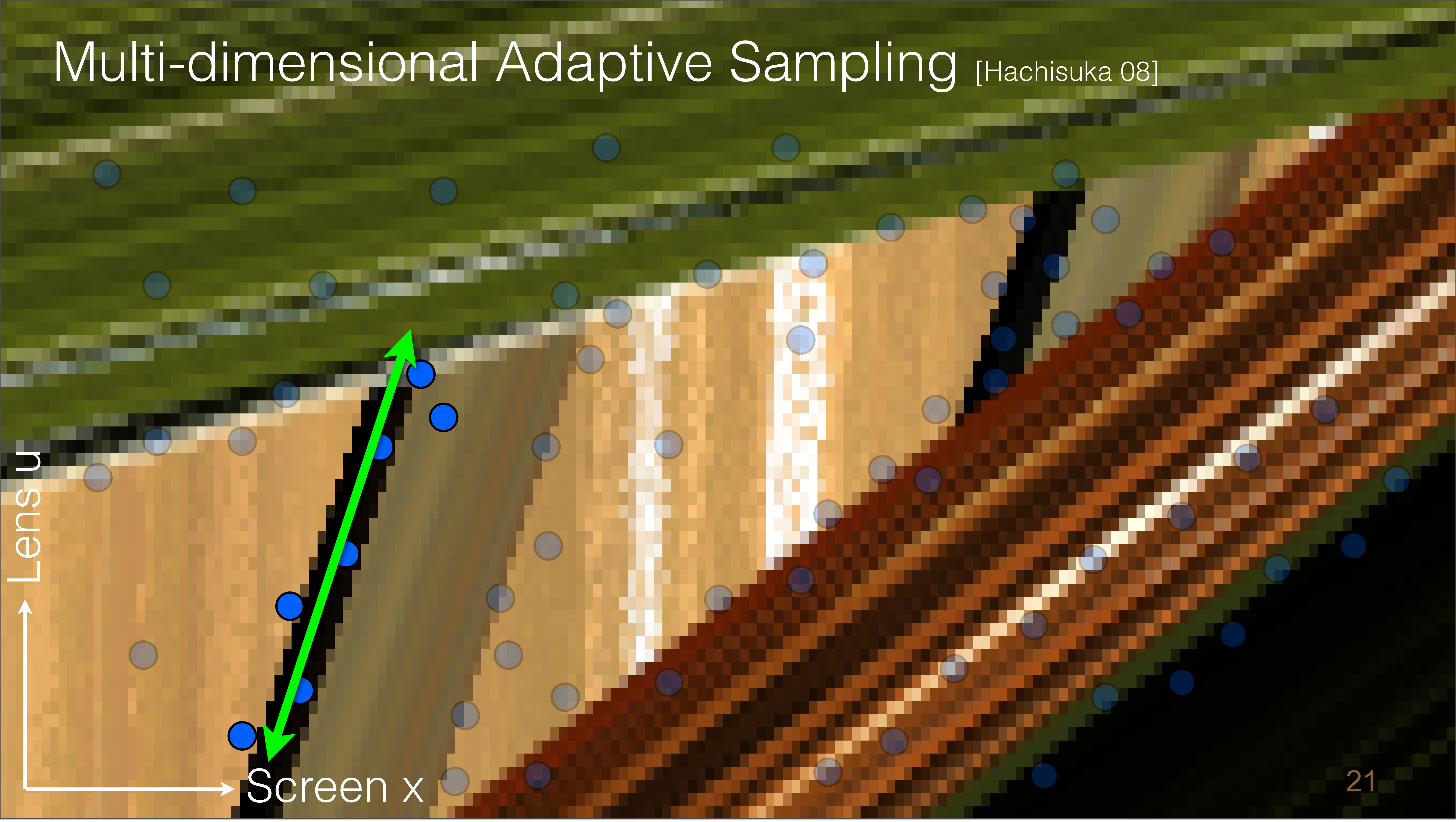


Multi-dimensional adaptive sampling, by Hachisuka and colleagues,
take a data driven approach to determining this structure.

They perform adaptive sampling <animate>,
which places samples near high contrast boundaries.

From these sample radiances, <animate>
they can deduce the local anisotropy in the light field <animate>

Unfortunately, because the approach is data-driven, texture and noise can throw off the estimator.
<shortpause>
And, their method does not explicitly handle visibility.



Multi-dimensional Adaptive Sampling [Hachisuka 08]

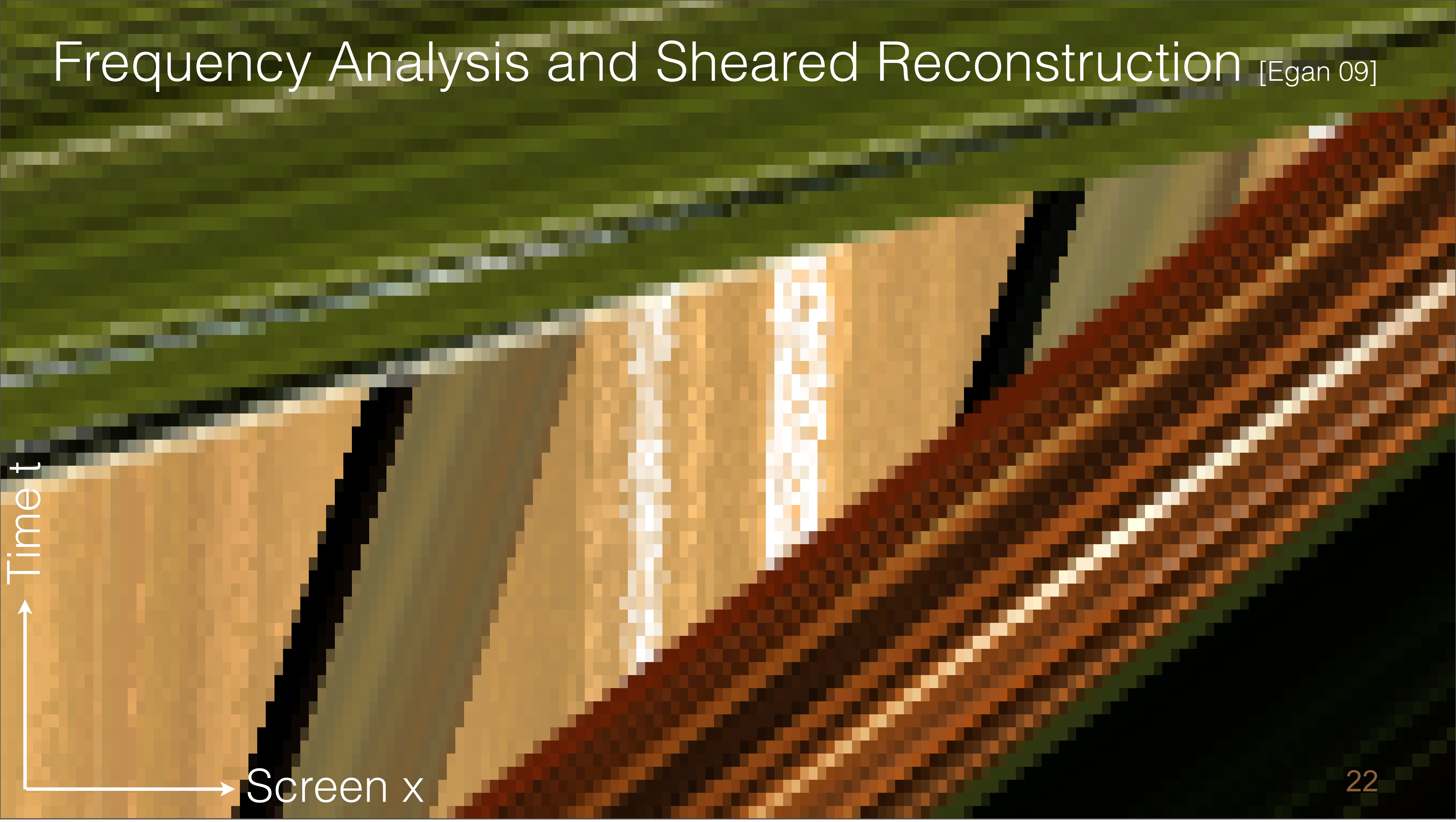
Multi-dimensional adaptive sampling, by Hachisuka and colleagues,
take a data driven approach to determining this structure.

They perform adaptive sampling <animate>,
which places samples near high contrast boundaries.

From these sample radiances, <animate>
they can deduce the local anisotropy in the light field <animate>

Unfortunately, because the approach is data-driven, texture and noise can throw off the estimator.
<shortpause>
And, their method does not explicitly handle visibility.

Frequency Analysis and Sheared Reconstruction [Egan 09]



More recently, Kevin Egan and colleagues applied Fourier analysis to directly compute the slopes of the temporal light field for the case of motion blur.

Like MDAS, they start with a sparse sampling of the scene. <animate>
But now, each sample comes equipped with a velocity vector, or slope <animate>

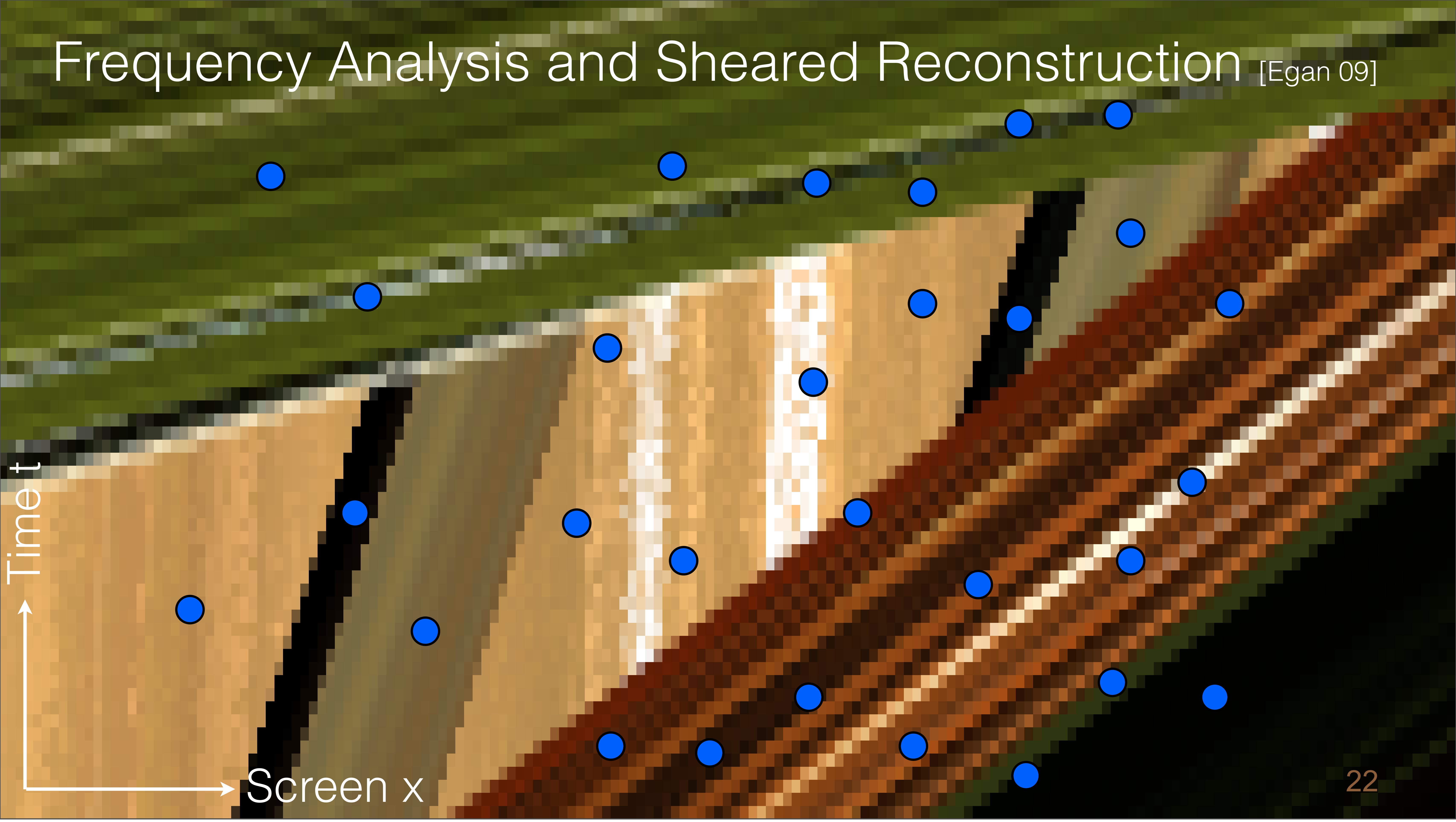
To produce the output image,
at each *pixel*, <animate>
they fit an anisotropic reconstruction kernel to the input slopes within that pixel combined with the frequency response of integrating over the shutter interval.

However, because their method operates on a per-pixel basis, <animate>
it doesn't work as well in regions with complex motion or visibility, where a filter with a single direction of anisotropy is insufficient.

Like Egan's algorithm, our method also computes and takes advantage of per-sample slopes.
However, we explicitly decouple the reconstruction and integration steps, which lets us handle these difficult regions.

Let me walk you through an overview of how our algorithm works.

Frequency Analysis and Sheared Reconstruction [Egan 09]



More recently, Kevin Egan and colleagues applied Fourier analysis to directly compute the slopes of the temporal light field for the case of motion blur.

Like MDAS, they start with a sparse sampling of the scene. <animate>
But now, each sample comes equipped with a velocity vector, or slope <animate>

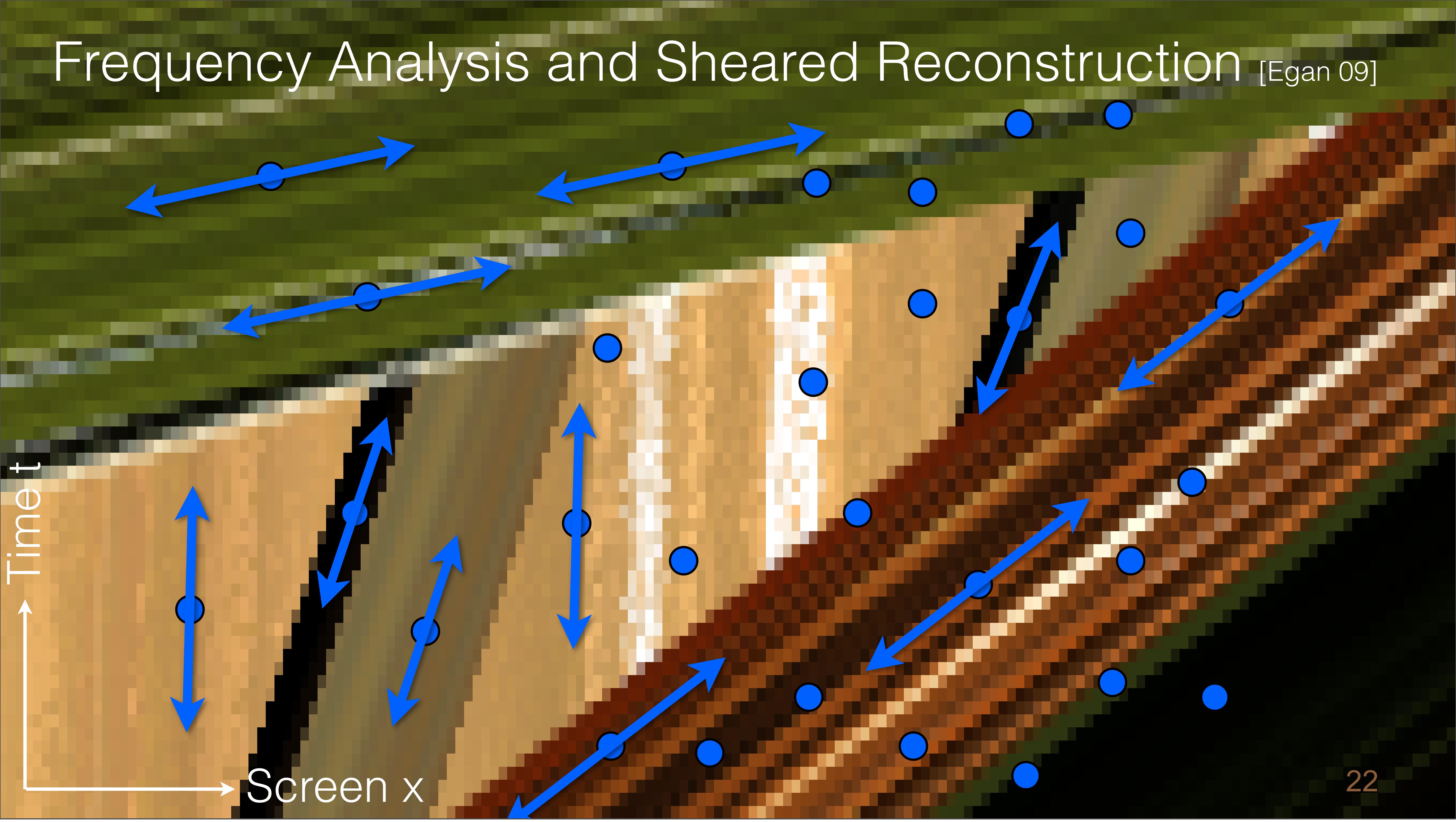
To produce the output image,
at each *pixel*, <animate>
they fit an anisotropic reconstruction kernel to the input slopes within that pixel combined with the frequency response of integrating over the shutter interval.

However, because their method operates on a per-pixel basis, <animate>
it doesn't work as well in regions with complex motion or visibility, where a filter with a single direction of anisotropy is insufficient.

Like Egan's algorithm, our method also computes and takes advantage of per-sample slopes.
However, we explicitly decouple the reconstruction and integration steps, which lets us handle these difficult regions.

Let me walk you through an overview of how our algorithm works.

Frequency Analysis and Sheared Reconstruction [Egan 09]



More recently, Kevin Egan and colleagues applied Fourier analysis to directly compute the slopes of the temporal light field for the case of motion blur.

Like MDAS, they start with a sparse sampling of the scene. <animate>
But now, each sample comes equipped with a velocity vector, or slope <animate>

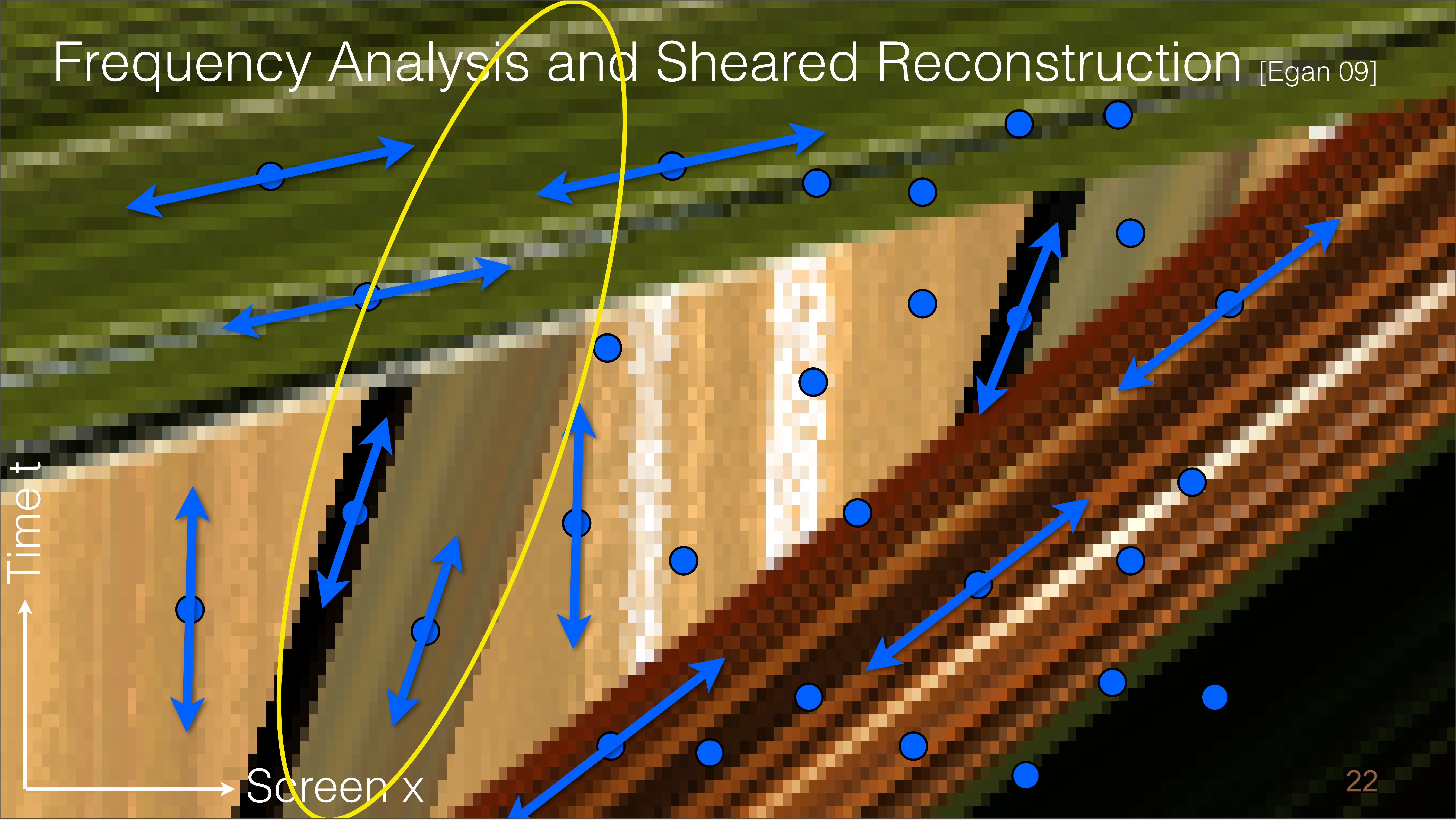
To produce the output image,
at each *pixel*, <animate>
they fit an anisotropic reconstruction kernel to the input slopes within that pixel combined with the frequency response of integrating over the shutter interval.

However, because their method operates on a per-pixel basis, <animate>
it doesn't work as well in regions with complex motion or visibility, where a filter with a single direction of anisotropy is insufficient.

Like Egan's algorithm, our method also computes and takes advantage of per-sample slopes.
However, we explicitly decouple the reconstruction and integration steps, which lets us handle these difficult regions.

Let me walk you through an overview of how our algorithm works.

Frequency Analysis and Sheared Reconstruction [Egan 09]



More recently, Kevin Egan and colleagues applied Fourier analysis to directly compute the slopes of the temporal light field for the case of motion blur.

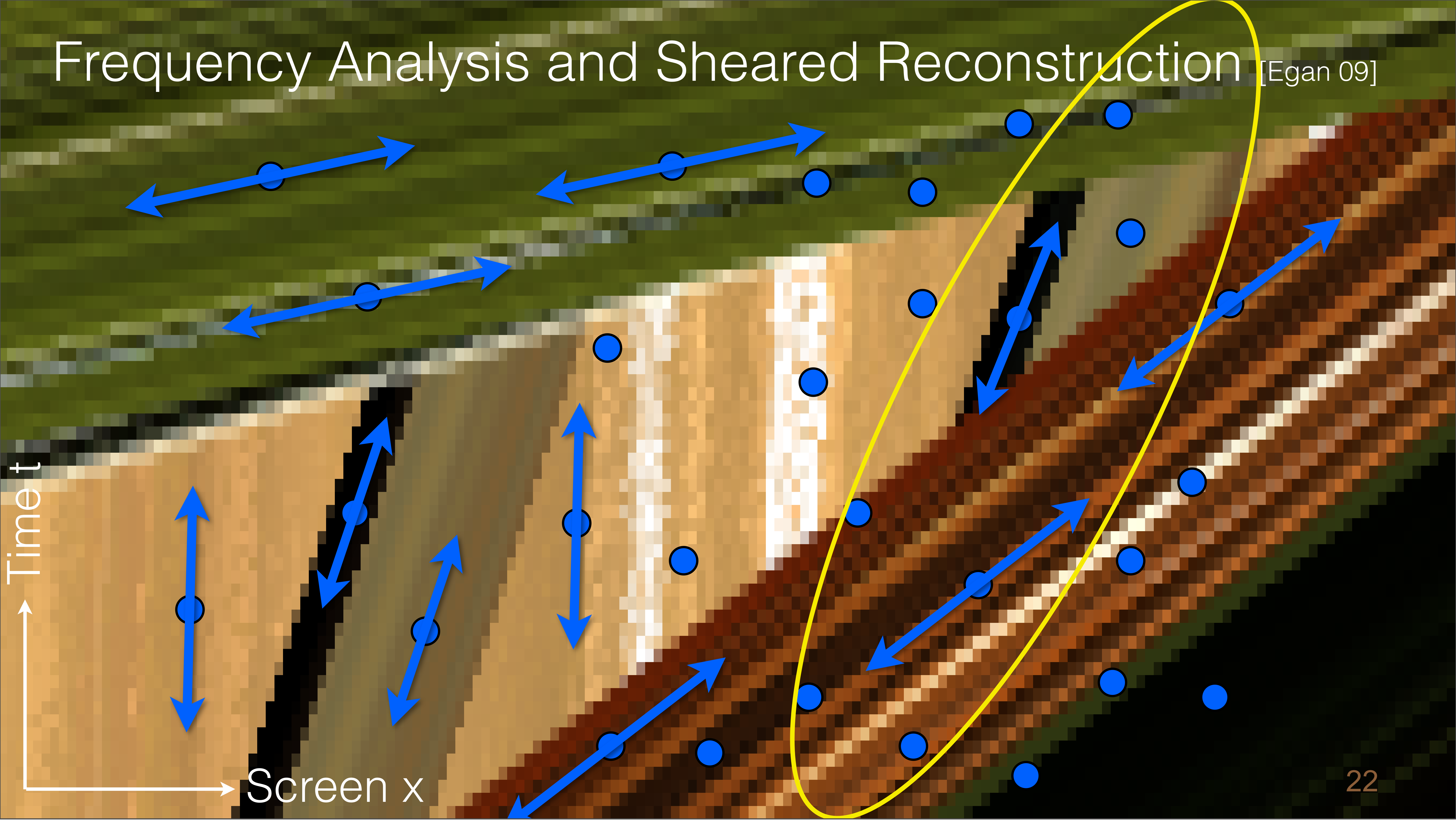
Like MDAS, they start with a sparse sampling of the scene. <animate>
But now, each sample comes equipped with a velocity vector, or slope <animate>

To produce the output image,
at each *pixel*, <animate>
they fit an anisotropic reconstruction kernel to the input slopes within that pixel combined with the frequency response of integrating over the shutter interval.

However, because their method operates on a per-pixel basis, <animate>
it doesn't work as well in regions with complex motion or visibility, where a filter with a single direction of anisotropy is insufficient.

Like Egan's algorithm, our method also computes and takes advantage of per-sample slopes.
However, we explicitly decouple the reconstruction and integration steps, which lets us handle these difficult regions.

Let me walk you through an overview of how our algorithm works.



Frequency Analysis and Sheared Reconstruction [Egan 09]

More recently, Kevin Egan and colleagues applied Fourier analysis to directly compute the slopes of the temporal light field for the case of motion blur.

Like MDAS, they start with a sparse sampling of the scene. <animate>
But now, each sample comes equipped with a velocity vector, or slope <animate>

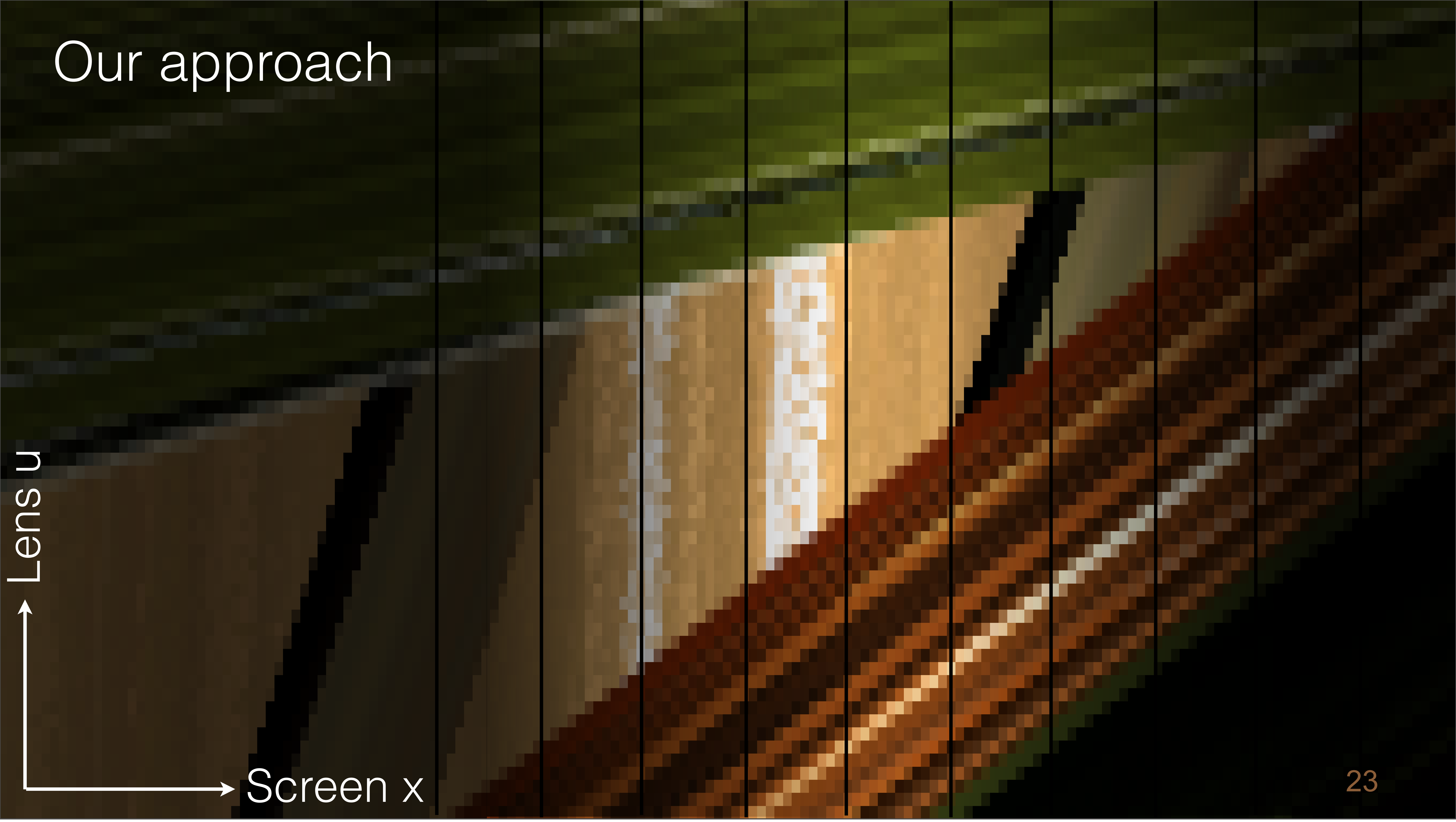
To produce the output image,
at each *pixel*, <animate>
they fit an anisotropic reconstruction kernel to the input slopes within that pixel combined with the frequency response of integrating over the shutter interval.

However, because their method operates on a per-pixel basis, <animate>
it doesn't work as well in regions with complex motion or visibility, where a filter with a single direction of anisotropy is insufficient.

Like Egan's algorithm, our method also computes and takes advantage of per-sample slopes.
However, we explicitly decouple the reconstruction and integration steps, which lets us handle these difficult regions.

Let me walk you through an overview of how our algorithm works.

Our approach

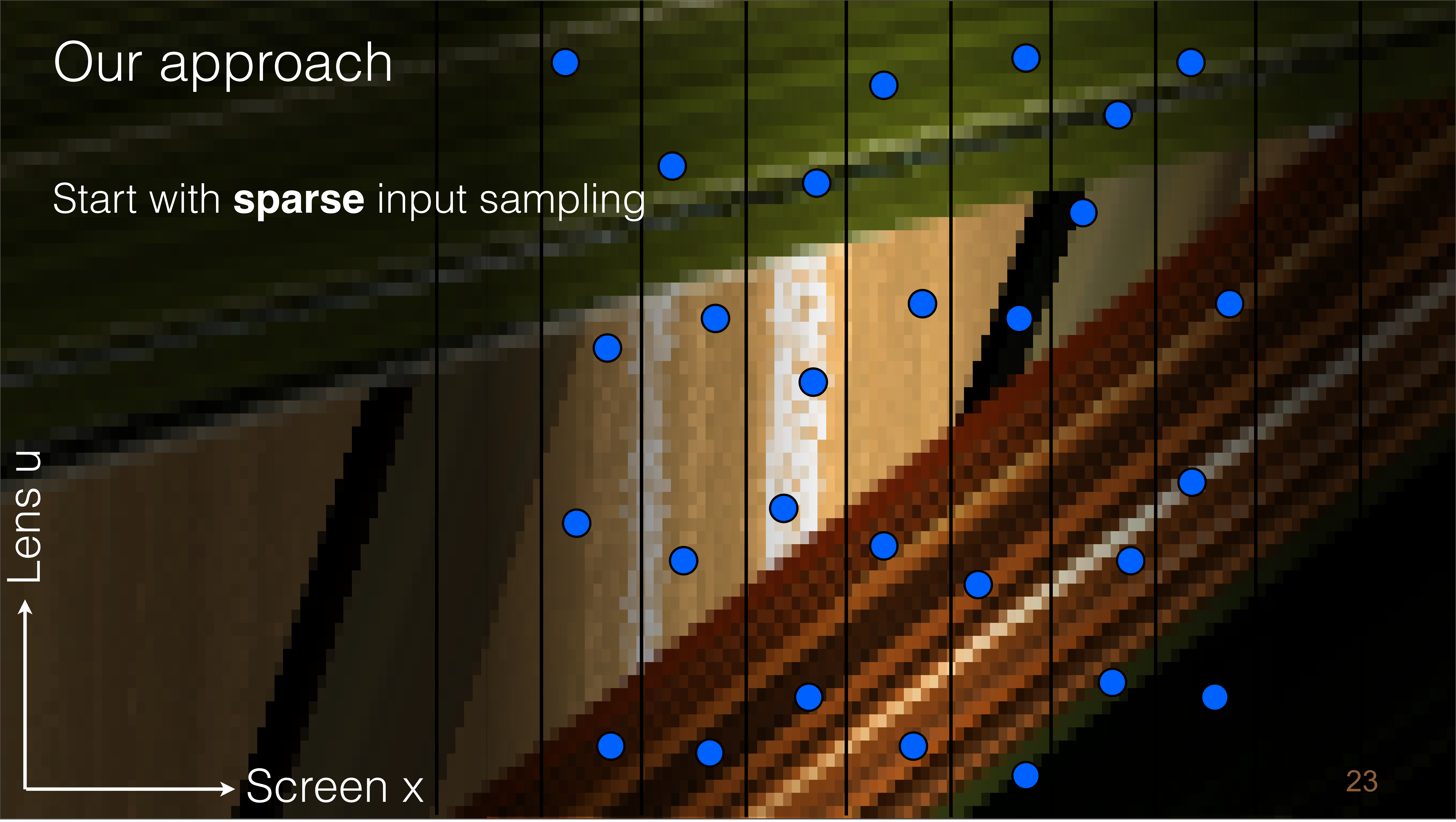


Like MDAS and sheared reconstruction, we also take as input a sparse stochastic sampling of the light field,
<animate>
visualized by these blue circles.

Note that we only have information at the blue circles.
The light field shown in the background is for illustration only.

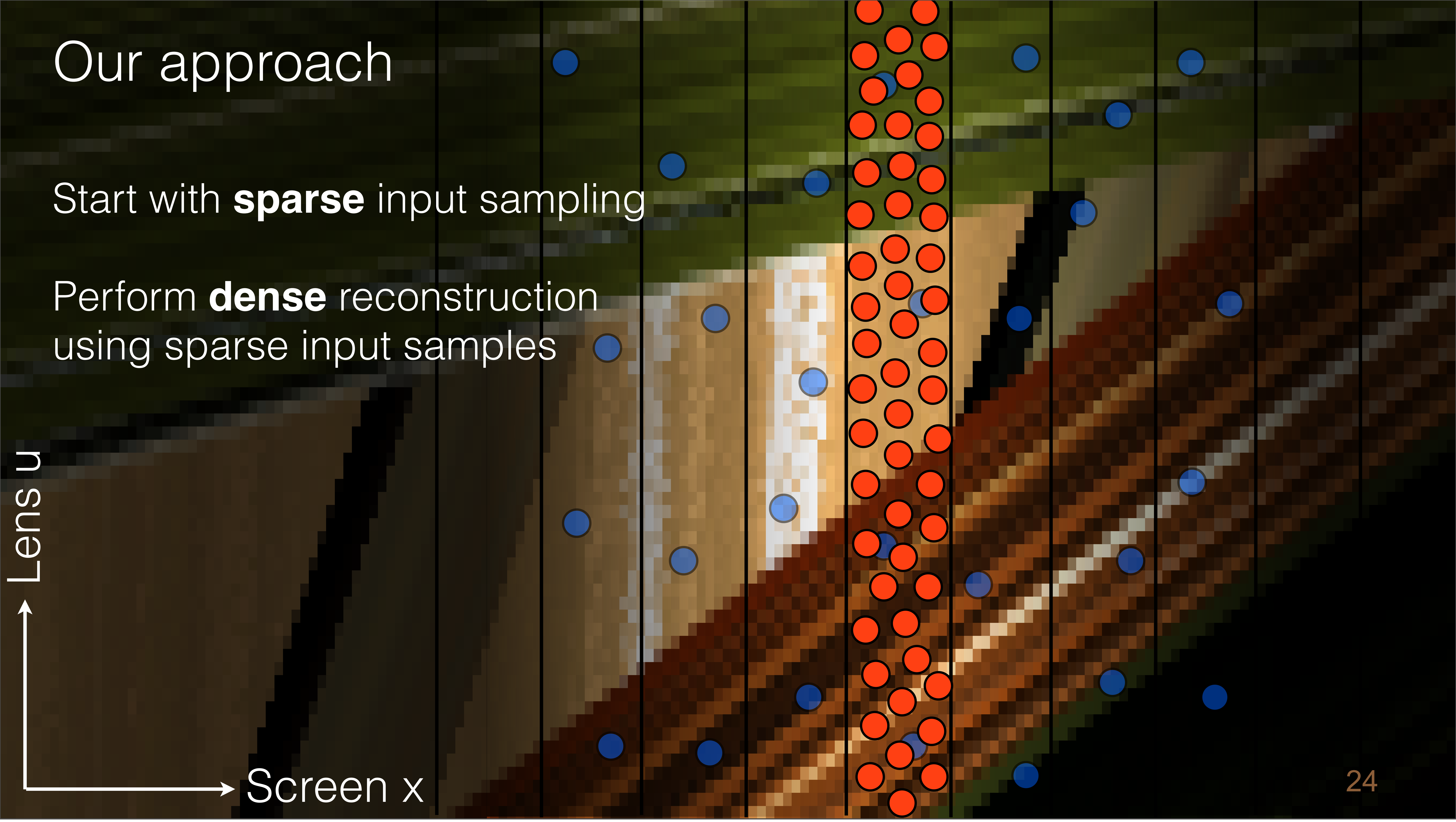
Our approach

Start with **sparse** input sampling



Like MDAS and sheared reconstruction, we also take as input a sparse stochastic sampling of the light field,
<animate>
visualized by these blue circles.

Note that we only have information at the blue circles.
The light field shown in the background is for illustration only.



Our approach

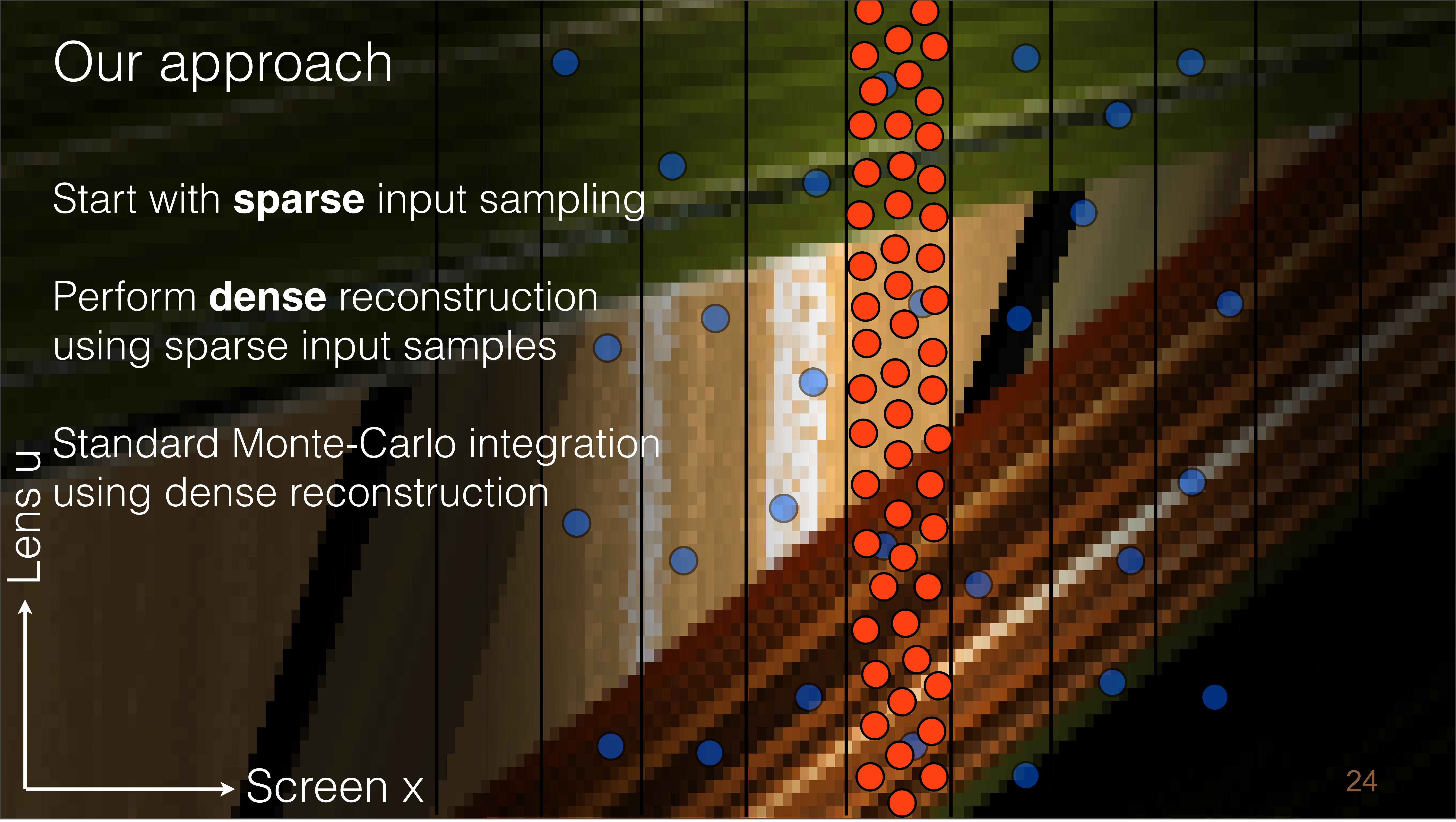
Start with **sparse** input sampling

Perform **dense** reconstruction using sparse input samples

We want to reconstruct the radiance at a dense set of locations,
visualized using the red circles,

and we want to do this,
using information from the sparse blue samples only.

<animate>
The last step is just standard Monte Carlo integration over the dense reconstructed radiances.



Our approach

Start with **sparse** input sampling

Perform **dense** reconstruction using sparse input samples

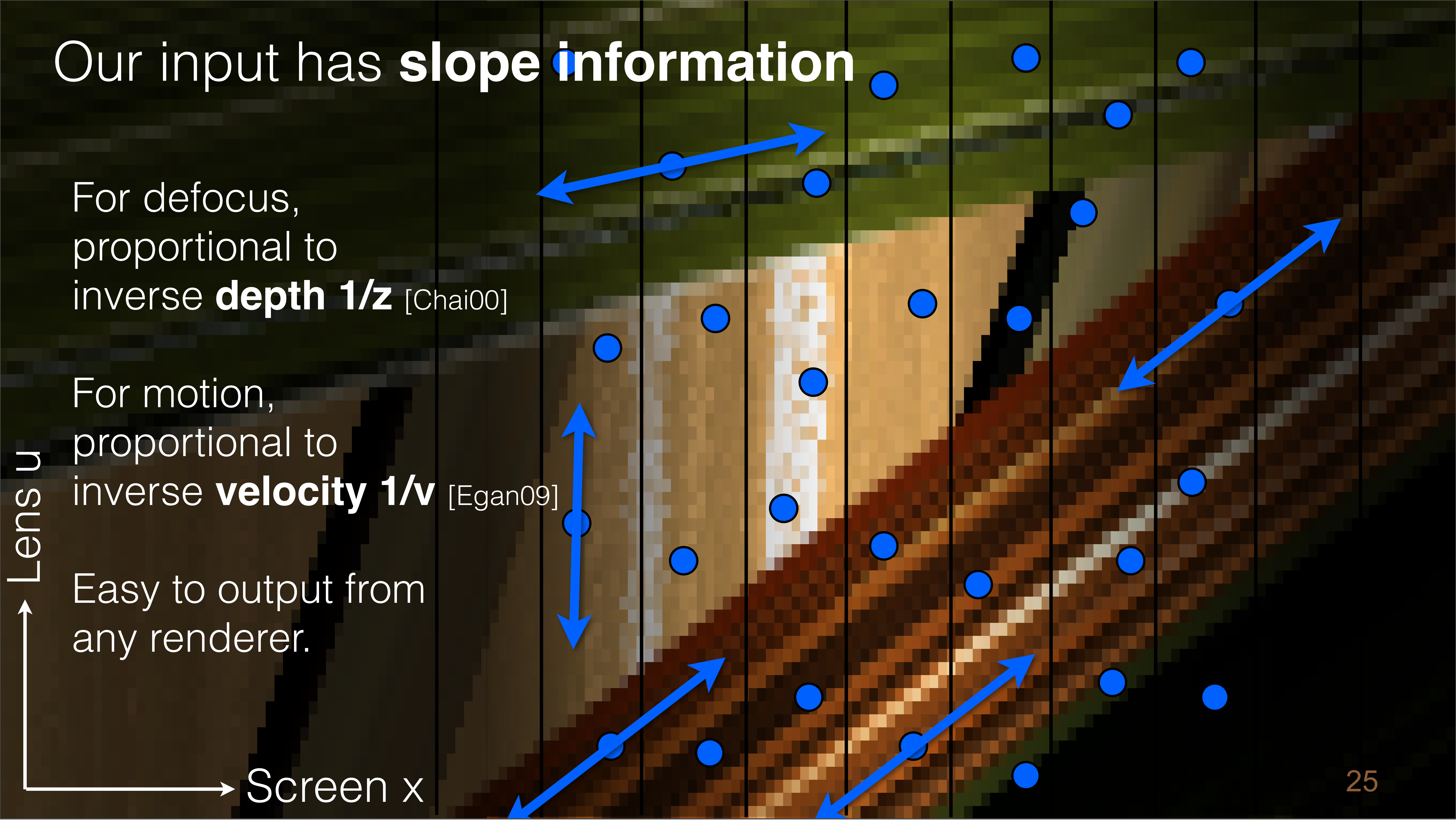
Standard Monte-Carlo integration using dense reconstruction

We want to reconstruct the radiance at a dense set of locations,
visualized using the red circles,

and we want to do this,
using information from the sparse blue samples only.

<animate>
The last step is just standard Monte Carlo integration over the dense reconstructed radiances.

Our input has **slope information**



For defocus, proportional to inverse **depth 1/z** [Chai00]

For motion, proportional to inverse **velocity 1/v** [Egan09]

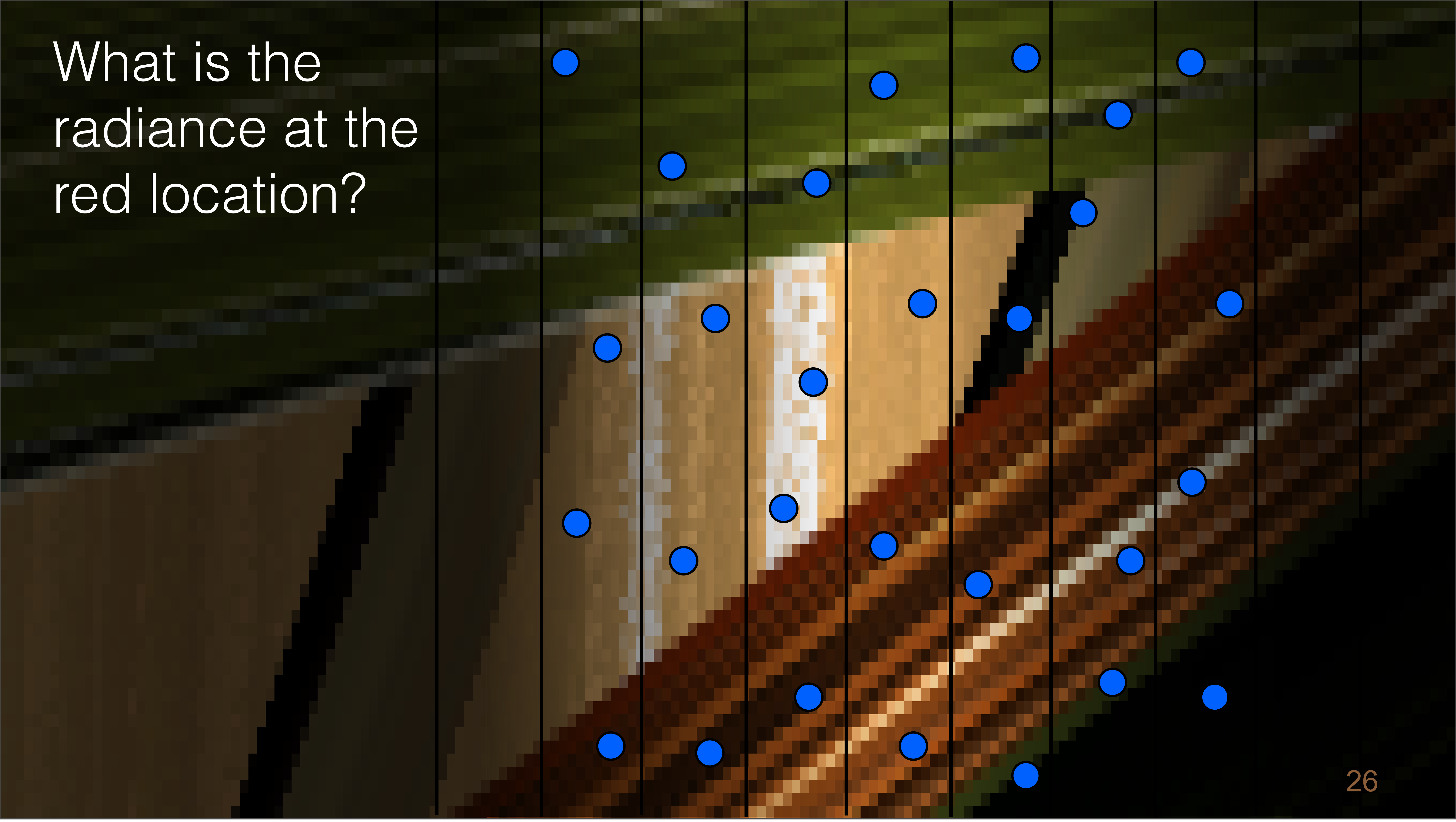
Easy to output from any renderer.

Like sheared reconstruction, our input samples come equipped with radiance and a slope.

These per-sample slopes depend only on the geometry of the scene.
For defocus, it’s proportional to inverse depth,
and for motion, it’s proportional to inverse velocity.

These are easy to output from any renderer.

What is the radiance at the red location?

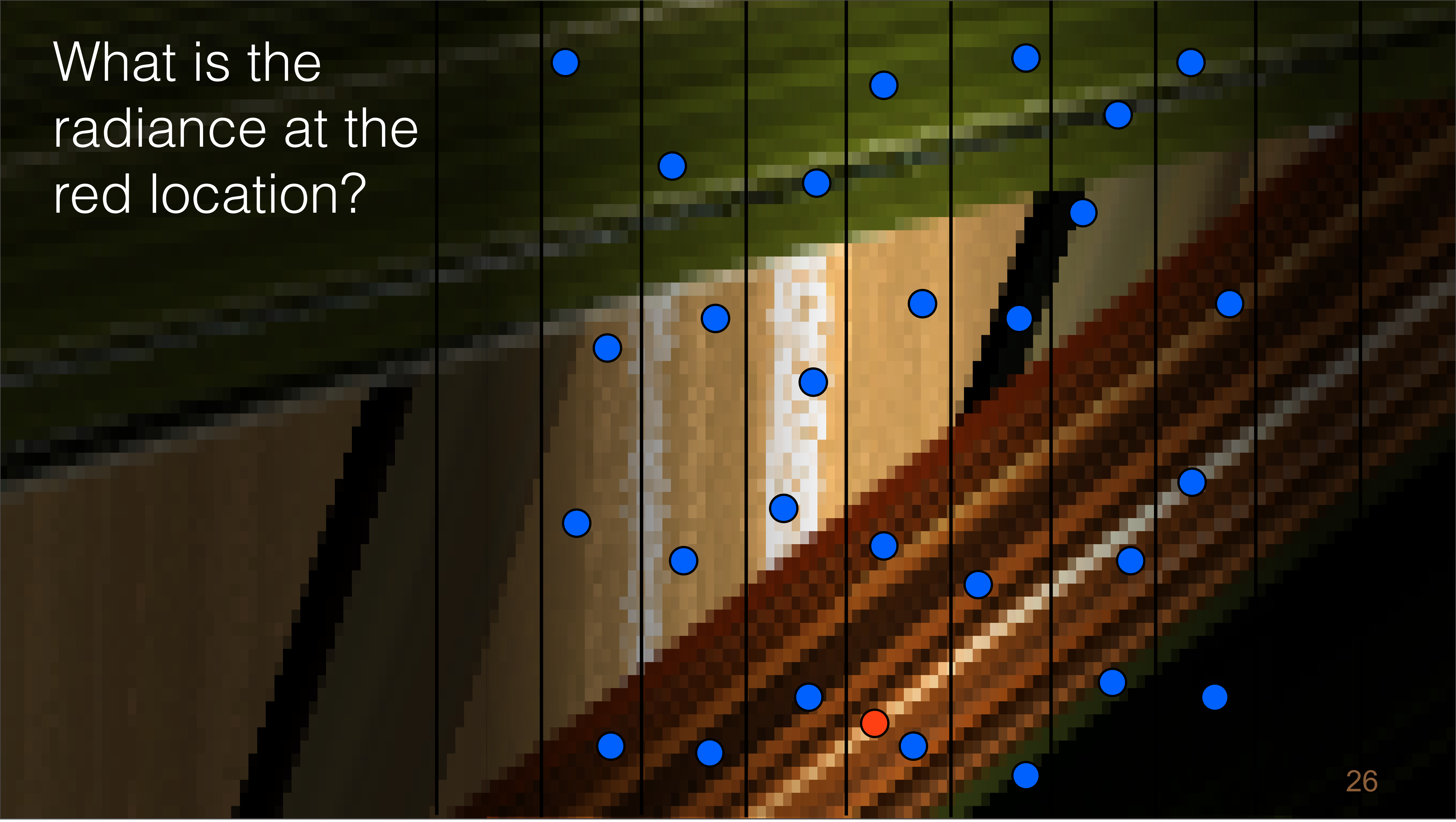


The central step of our method is to reconstruct the radiance at an arbitrary location.

For example, what is the radiance at this red location <animate>?

Well, we can reproject <animate>
from this input sample using its slope.
But...

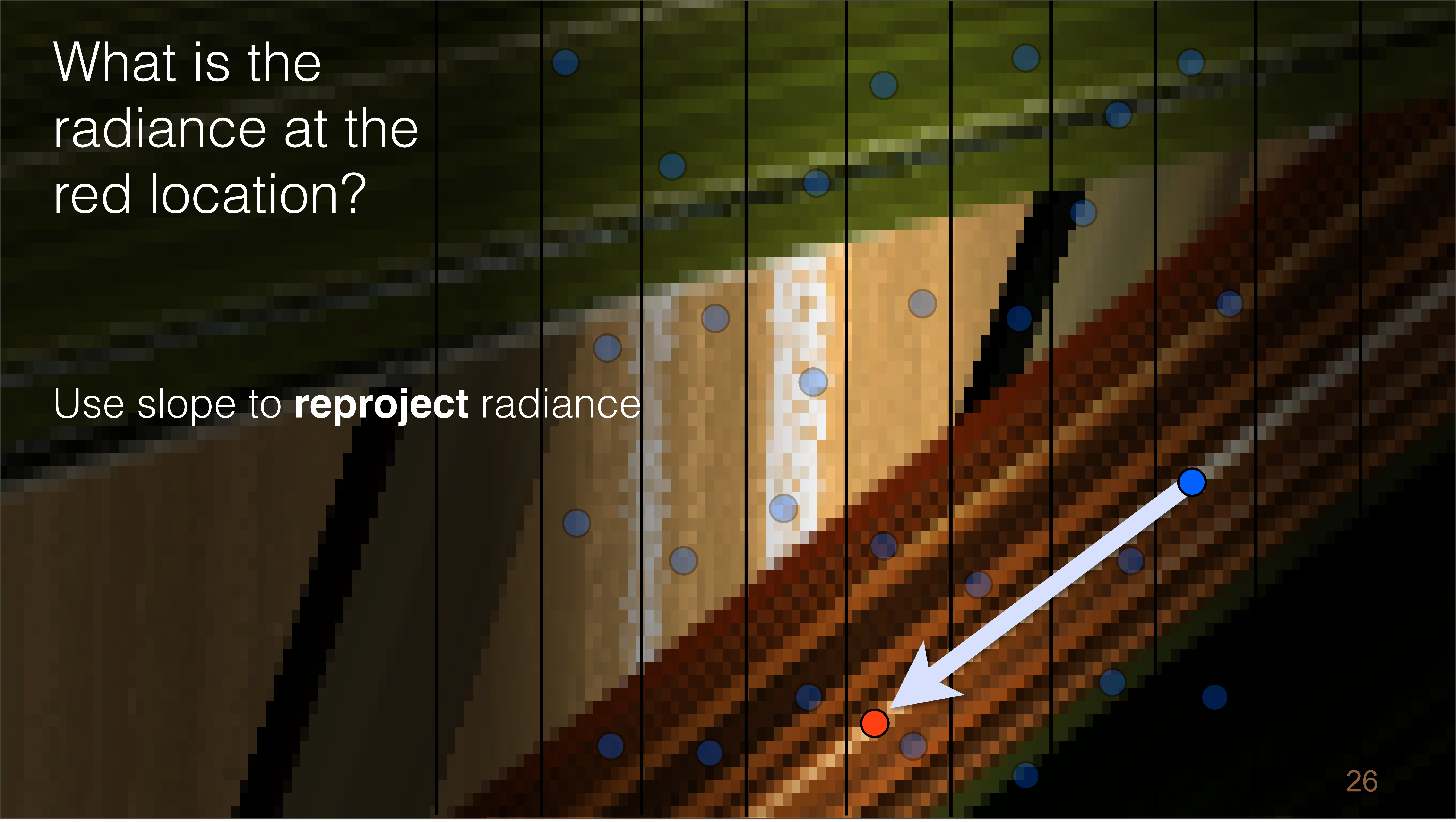
What is the radiance at the red location?



The central step of our method is to reconstruct the radiance at an arbitrary location.

For example, what is the radiance at this red location <animate>?

Well, we can reproject <animate>
from this input sample using its slope.
But...



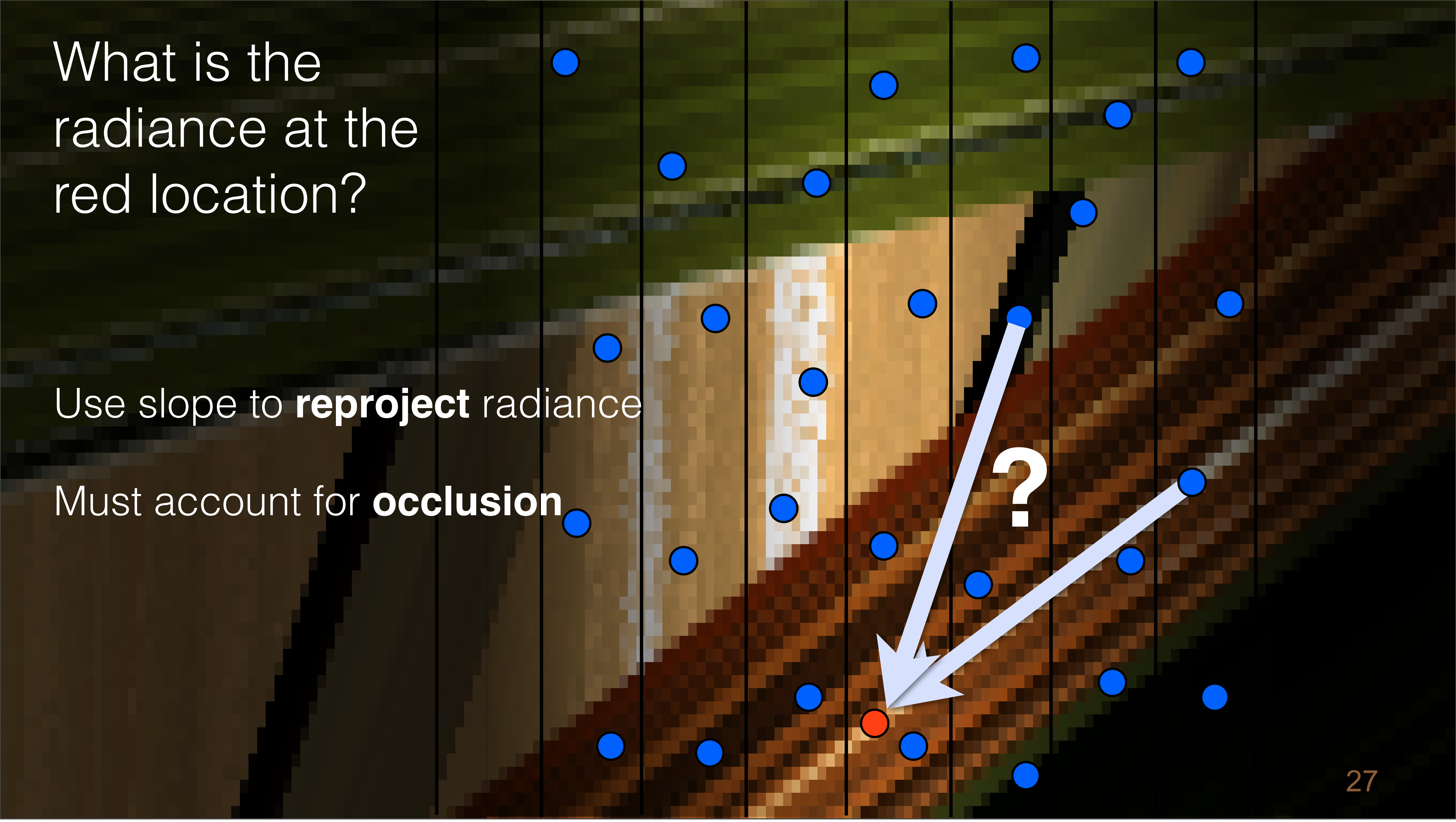
What is the radiance at the red location?

Use slope to **reproject** radiance

The central step of our method is to reconstruct the radiance at an arbitrary location.

For example, what is the radiance at this red location <animate>?

Well, we can reproject <animate>
from this input sample using its slope.
But...



What is the radiance at the red location?

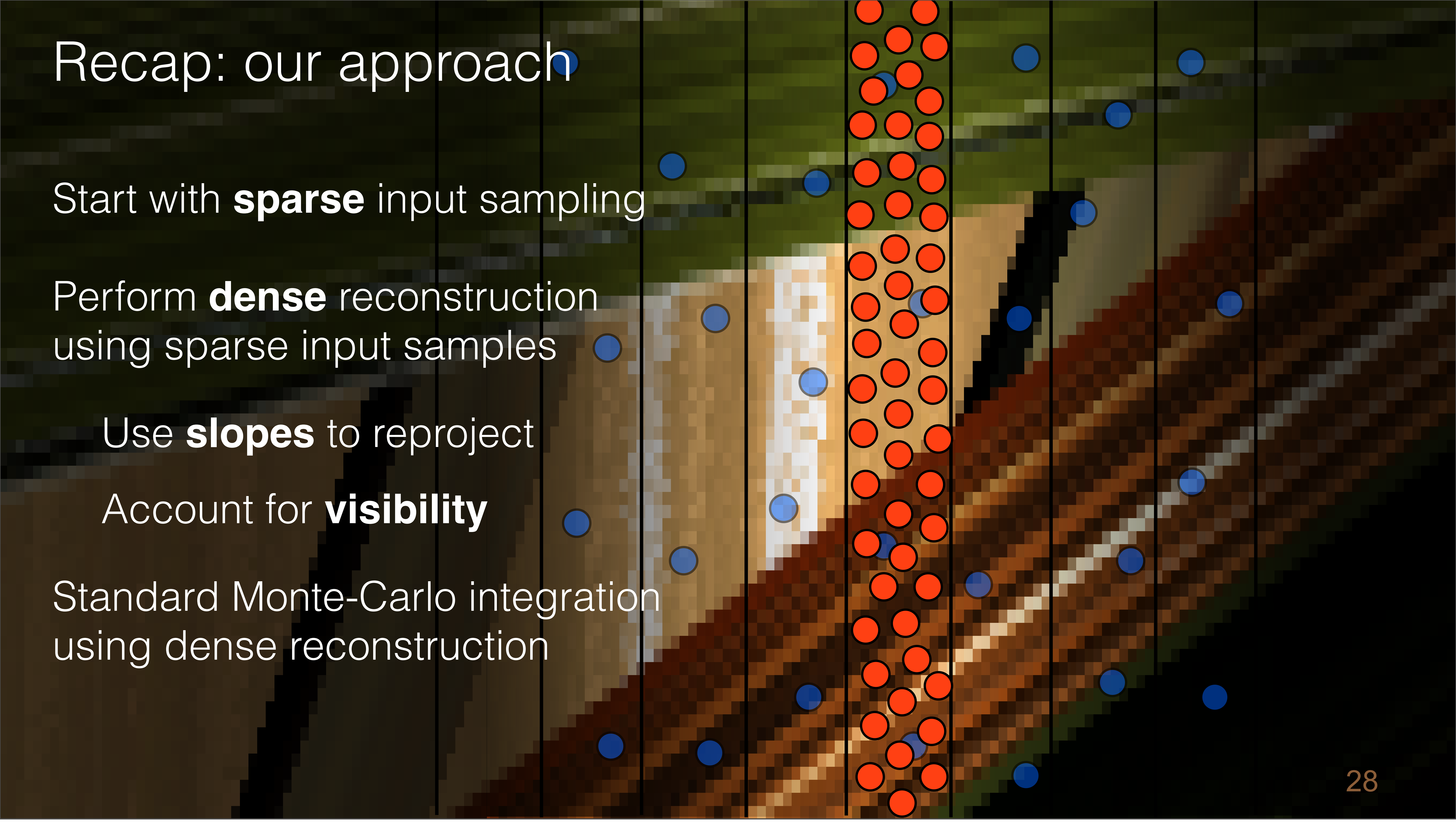
Use slope to **reproject** radiance

Must account for **occlusion**

we can also reproject from this one.

Their slopes both point to the same red location!
But if you look at the light field, the first sample is clearly on top.

Therefore, our reprojection must also account for occlusion.
Determining visibility is a key ingredient to our algorithm
and I’ll come back to it in more detail in a little bit.



Recap: our approach

Start with **sparse** input sampling

Perform **dense** reconstruction using sparse input samples

- Use **slopes** to reproject
- Account for **visibility**

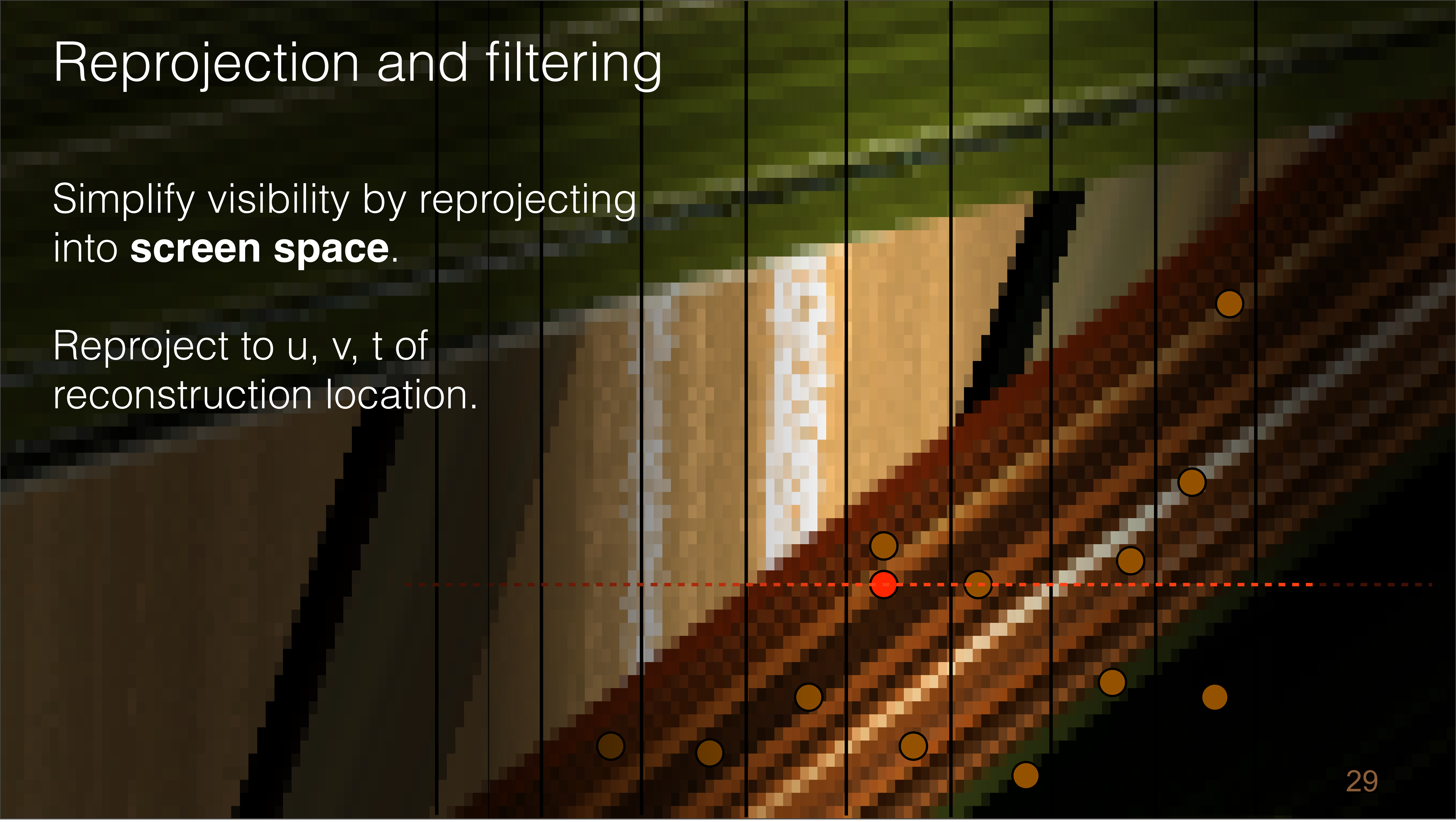
Standard Monte-Carlo integration using dense reconstruction

To recap, with our approach, we start with a sparse input sampling of the light field that comes with slopes

And we want to perform a dense reconstruction using this information.
To do this, we use the slopes attached to the sparse input to reproject them, while taking into account visibility.

Finally, we perform standard Monte-Carlo integration over the dense reconstruction.

Let’s look at how reprojection and visibility works in more detail.



Reprojection and filtering

Simplify visibility by reprojecting into **screen space**.

Reproject to u, v, t of reconstruction location.

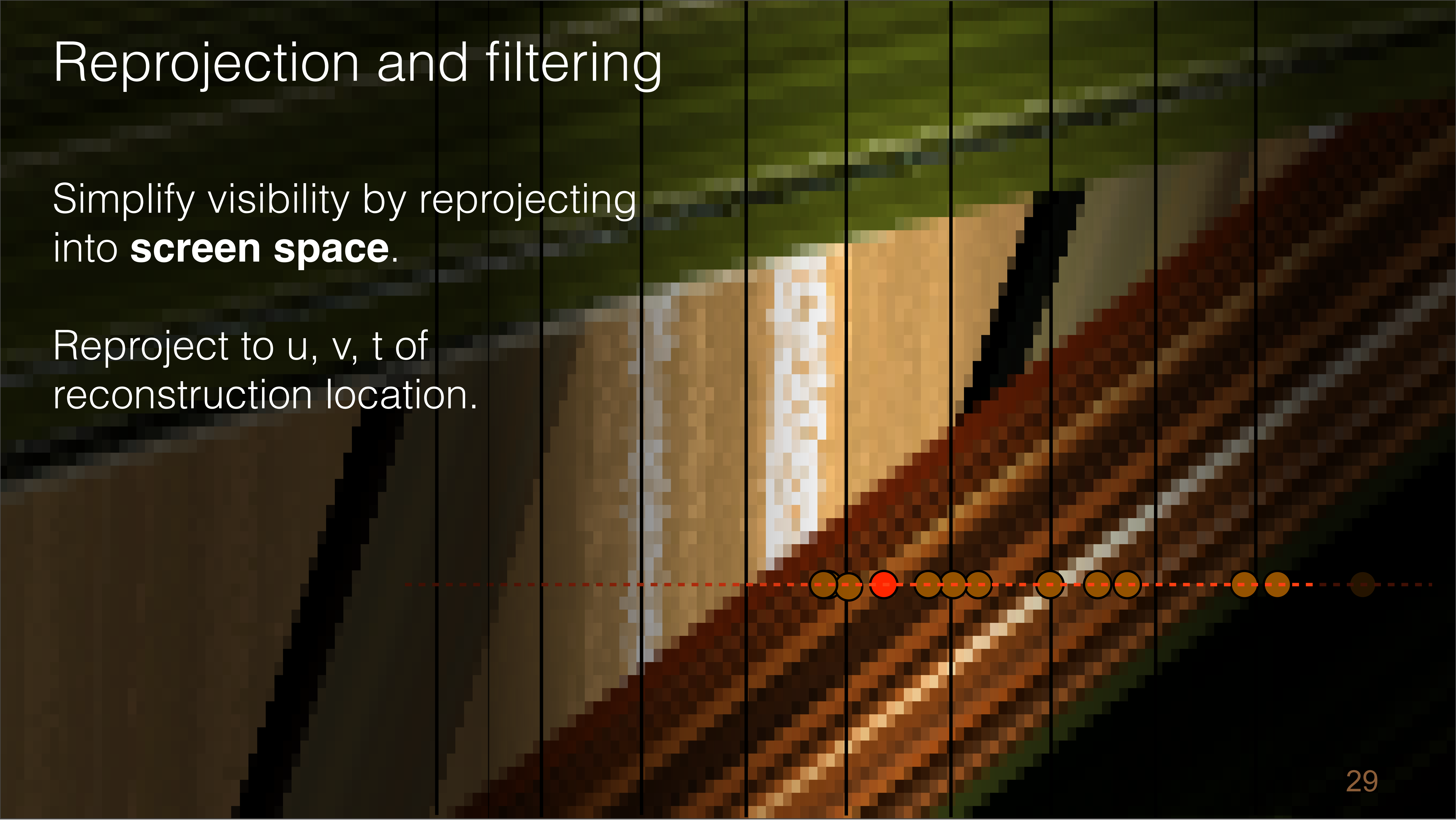
To simplify the visibility problem later, we re-project the input samples into 2D screen space.

This just means taking the input samples,
<animate>
and following their slopes until they reach the same lens and time coordinate as the reconstruction location.

Then, after we account for visibility,
<animate>
all we need to do is compute a 2D pixel filter over the visible samples and we're done!

<pause>
I want to point out that with a 5D temporal light field, the input sampling is *extremely* sparse.
However, because the function is approximately constant along these trajectories
we can essentially reconstruct at an arbitrary resolution.

Now let's look at how we solve visibility.



Reprojection and filtering

Simplify visibility by reprojecting into **screen space**.

Reproject to u, v, t of reconstruction location.

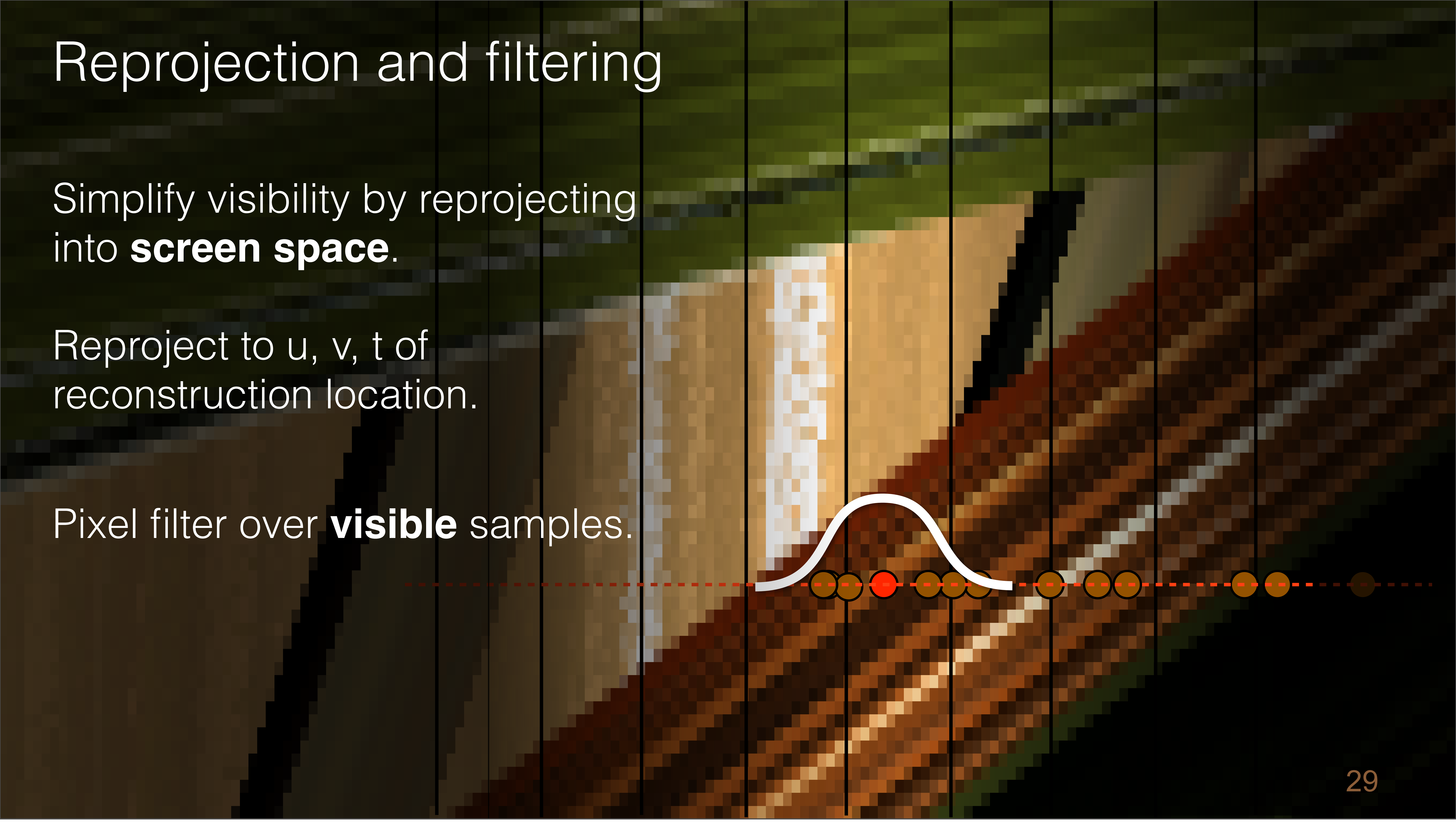
To simplify the visibility problem later, we re-project the input samples into 2D screen space.

This just means taking the input samples,
<animate>
and following their slopes until they reach the same lens and time coordinate as the reconstruction location.

Then, after we account for visibility,
<animate>
all we need to do is compute a 2D pixel filter over the visible samples and we’re done!

<pause>
I want to point out that with a 5D temporal light field, the input sampling is *extremely* sparse.
However, because the function is approximately constant along these trajectories
we can essentially reconstruct at an arbitrary resolution.

Now let’s look at how we solve visibility.



Reprojection and filtering

Simplify visibility by reprojecting into **screen space**.

Reproject to u, v, t of reconstruction location.

Pixel filter over **visible** samples.

To simplify the visibility problem later, we re-project the input samples into 2D screen space.

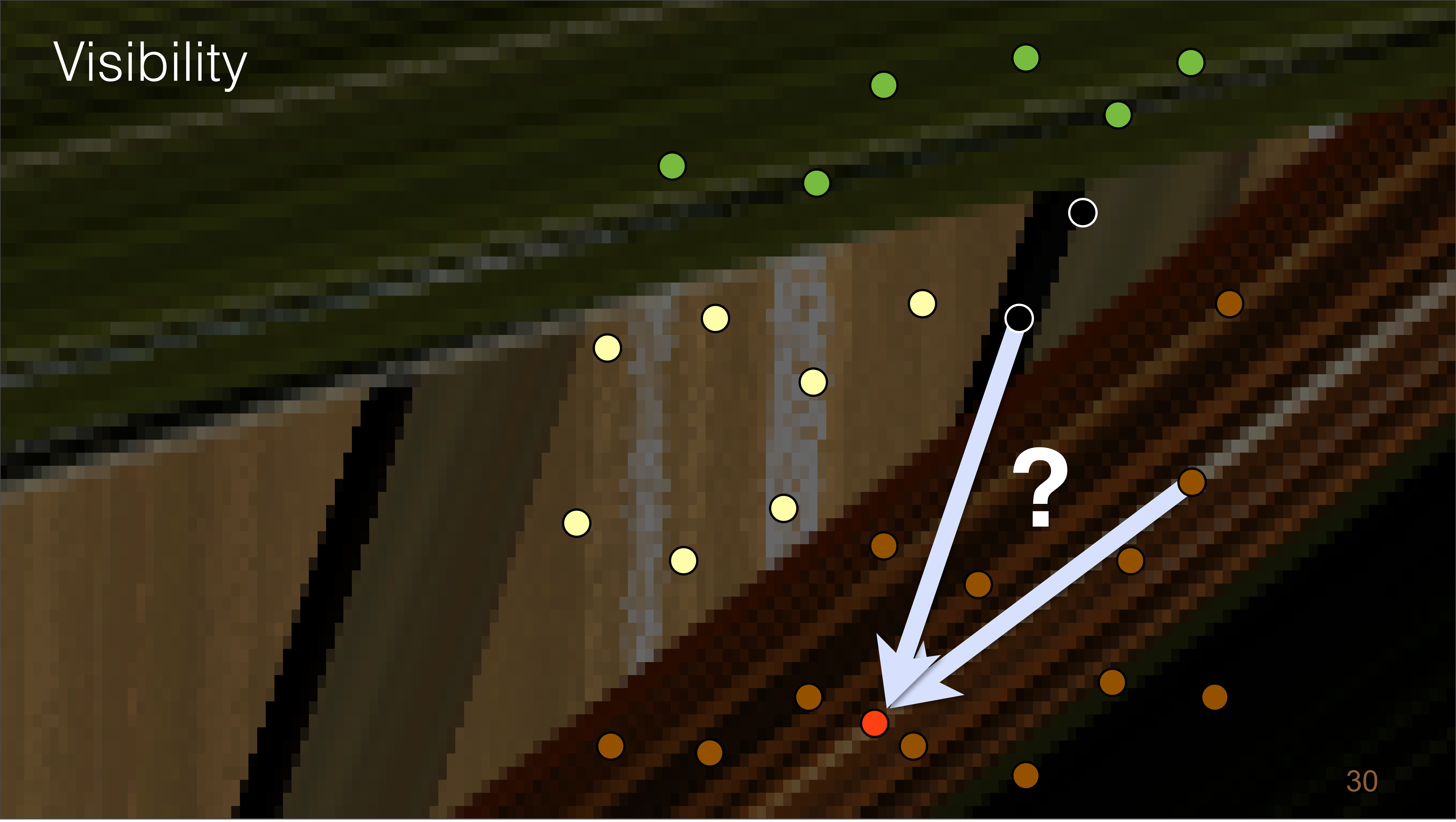
This just means taking the input samples,
<animate>
and following their slopes until they reach the same lens and time coordinate as the reconstruction location.

Then, after we account for visibility,
<animate>
all we need to do is compute a 2D pixel filter over the visible samples and we’re done!

<pause>
I want to point out that with a 5D temporal light field, the input sampling is *extremely* sparse.
However, because the function is approximately constant along these trajectories
we can essentially reconstruct at an arbitrary resolution.

Now let’s look at how we solve visibility.

Visibility

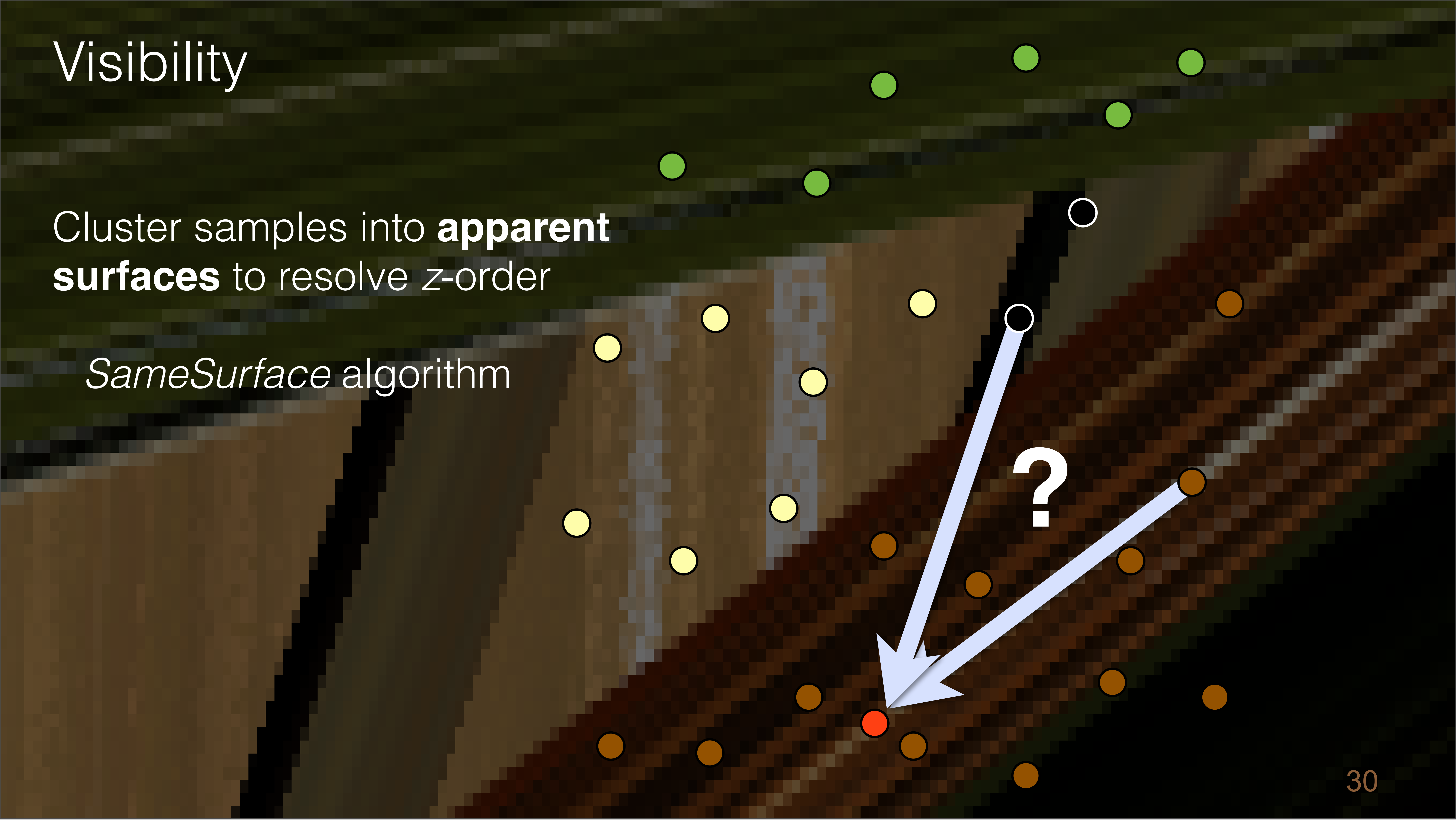


To determine the radiance at a reconstruction location, we have two related issues.
Remember that all we have as input is a cloud of point samples with slopes.
They effectively have zero area.

<animate>
Therefore, to figure out what’s in front of what, we first need to cluster point samples into apparent surfaces.
For this, we have geometric algorithm that we call SameSurface.

<animate>
Once we have apparent surfaces, the other issue is determining coverage:
What is the front-most apparent surface that covers the reconstruction location, if any.

For this, we carefully search the neighborhood of the reconstruction location,
making sure that real holes are not covered.



Visibility

Cluster samples into **apparent surfaces** to resolve *z*-order

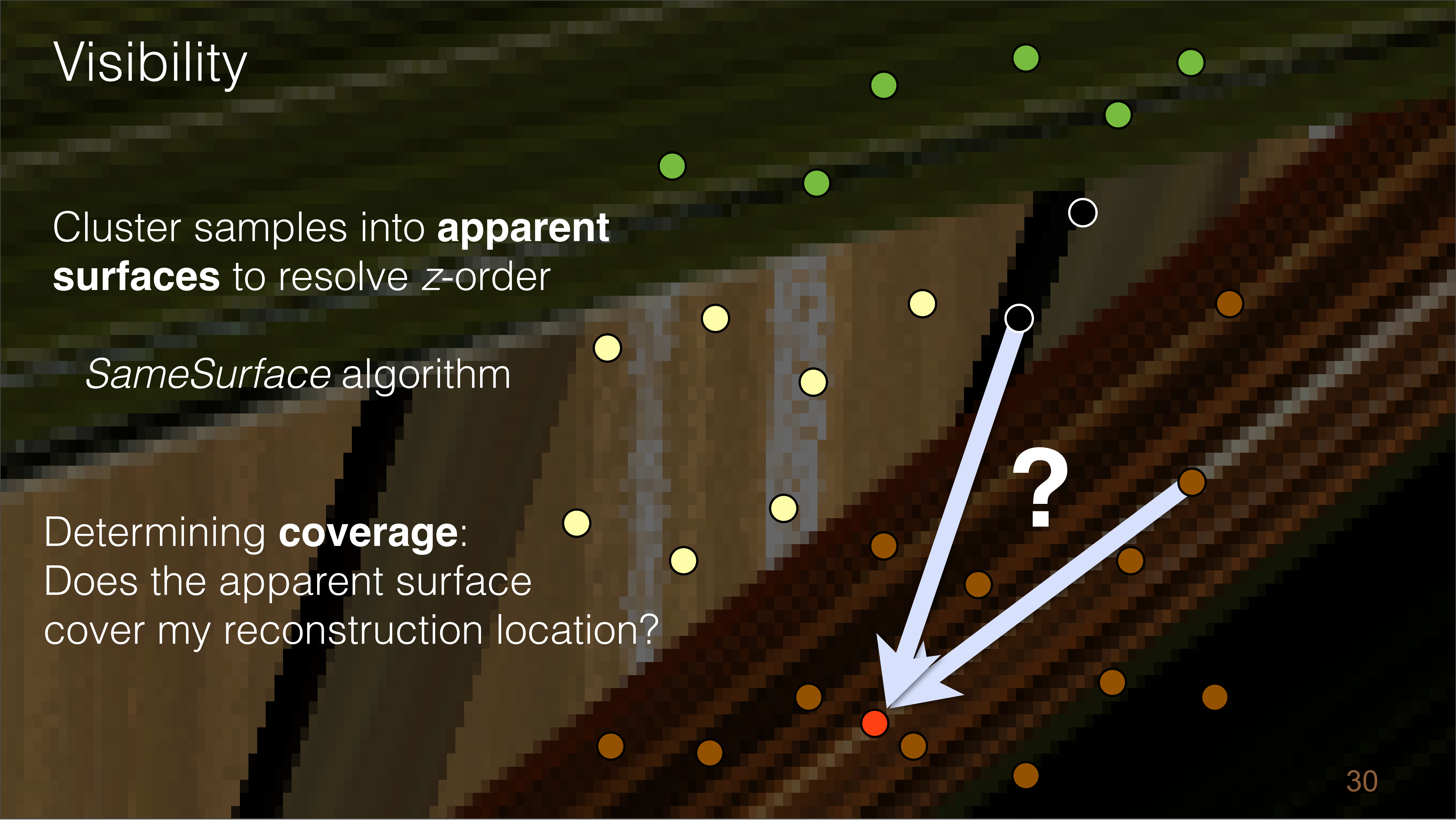
SameSurface algorithm

To determine the radiance at a reconstruction location, we have two related issues.
Remember that all we have as input is a cloud of point samples with slopes.
They effectively have zero area.

<animate>
Therefore, to figure out what’s in front of what, we first need to cluster point samples into apparent surfaces.
For this, we have geometric algorithm that we call SameSurface.

<animate>
Once we have apparent surfaces, the other issue is determining coverage:
What is the front-most apparent surface that covers the reconstruction location, if any.

For this, we carefully search the neighborhood of the reconstruction location,
making sure that real holes are not covered.



Visibility

Cluster samples into **apparent surfaces** to resolve *z*-order

SameSurface algorithm

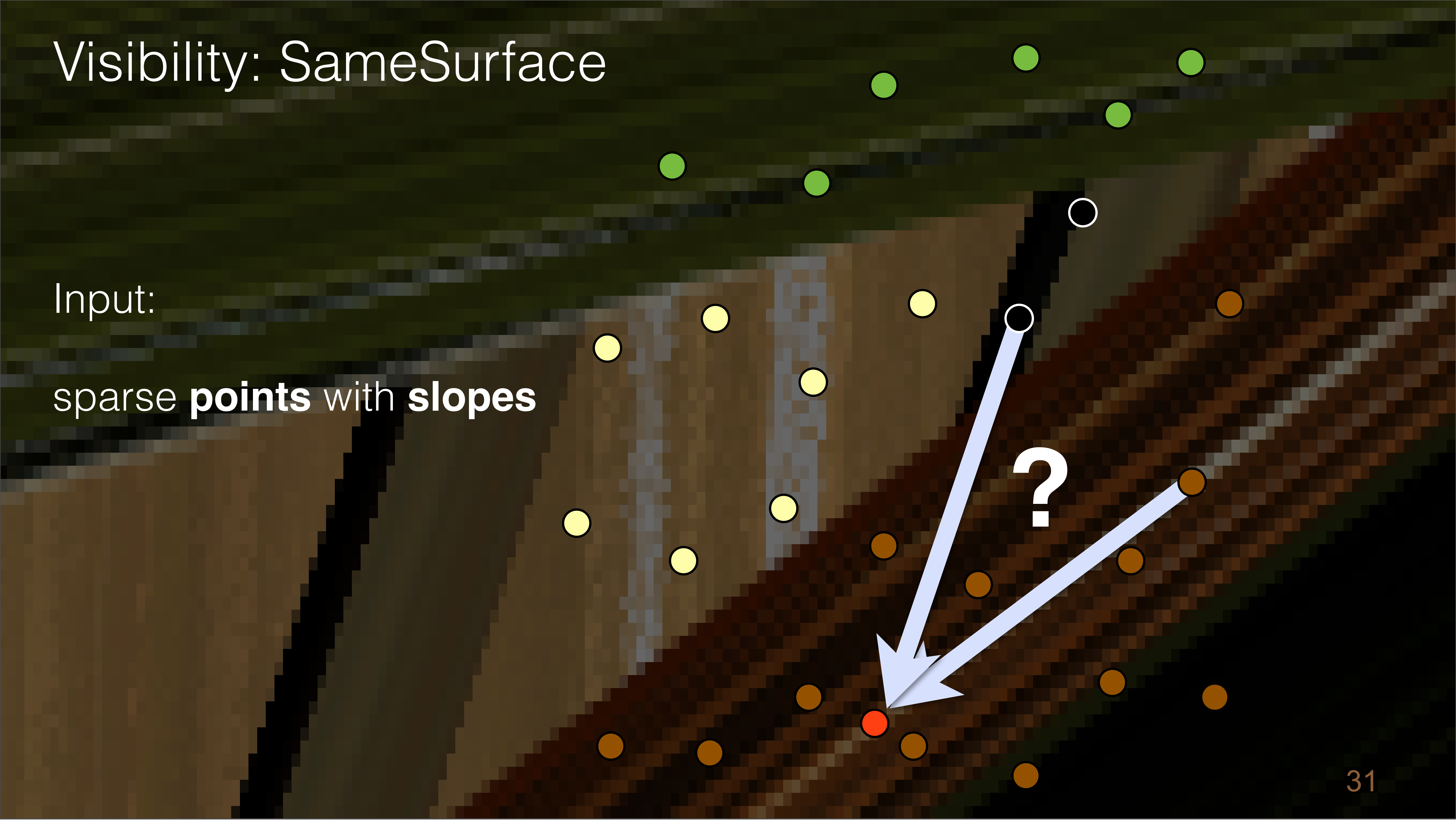
Determining **coverage**:
Does the apparent surface cover my reconstruction location?

To determine the radiance at a reconstruction location, we have two related issues.
Remember that all we have as input is a cloud of point samples with slopes.
They effectively have zero area.

<animate>
Therefore, to figure out what’s in front of what, we first need to cluster point samples into apparent surfaces.
For this, we have geometric algorithm that we call SameSurface.

<animate>
Once we have apparent surfaces, the other issue is determining coverage:
What is the front-most apparent surface that covers the reconstruction location, if any.

For this, we carefully search the neighborhood of the reconstruction location,
making sure that real holes are not covered.



Visibility: SameSurface

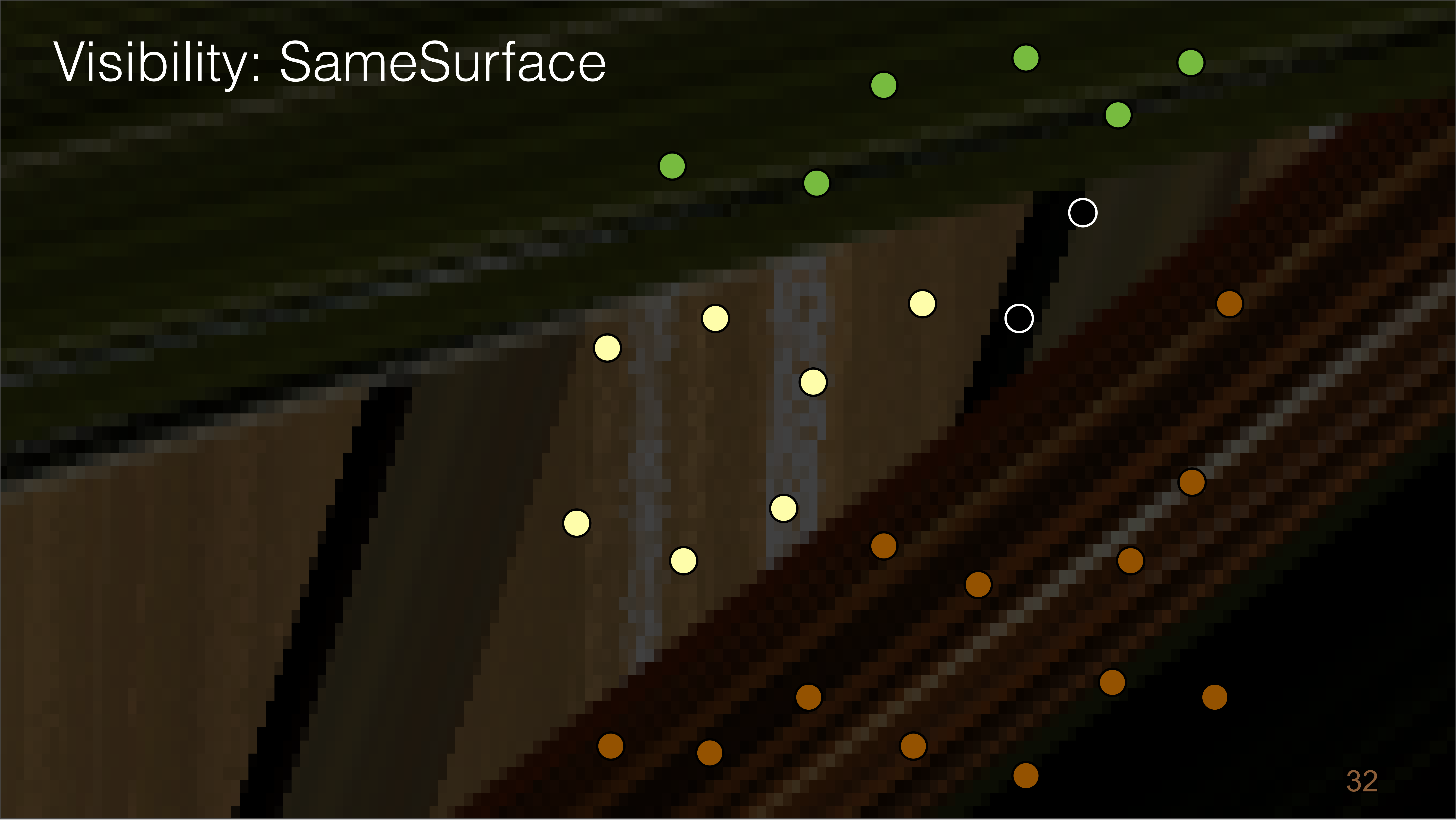
Input:

sparse **points** with **slopes**

Once again, recall that all we have, as input,

are point samples with slopes

Visibility: SameSurface



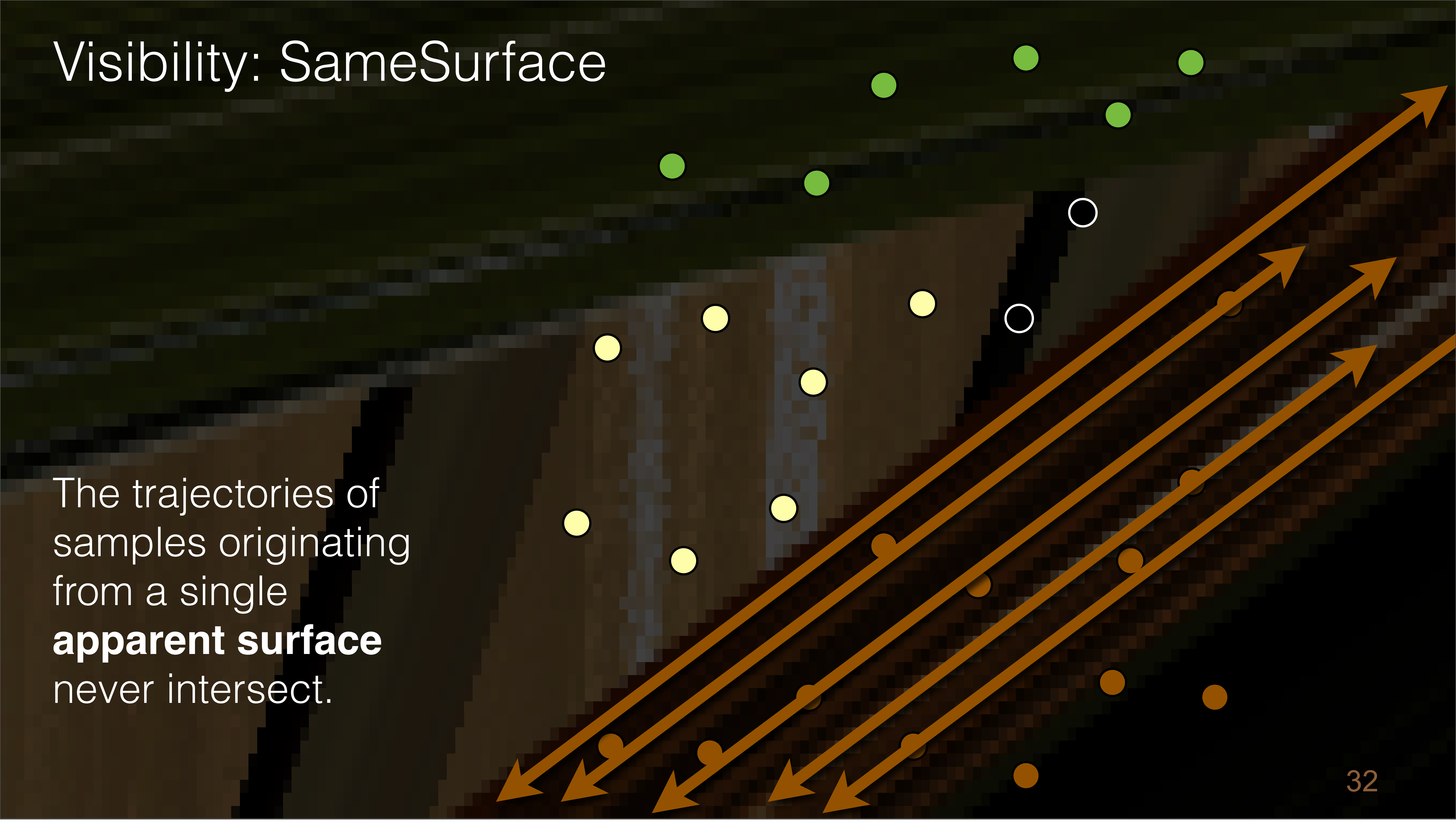
But observe that <animate>

if a set of samples don’t move relative to each other under viewpoint changes in such a way that they occlude on screen,

then their trajectories do not intersect in the light field.

Therefore, we can treat this set of samples as a single *apparent surface*.

So these samples form an apparent surface.
<animate>
And these..
<animate>
and these...
<animate>
<next>



Visibility: SameSurface

The trajectories of samples originating from a single **apparent surface** never intersect.

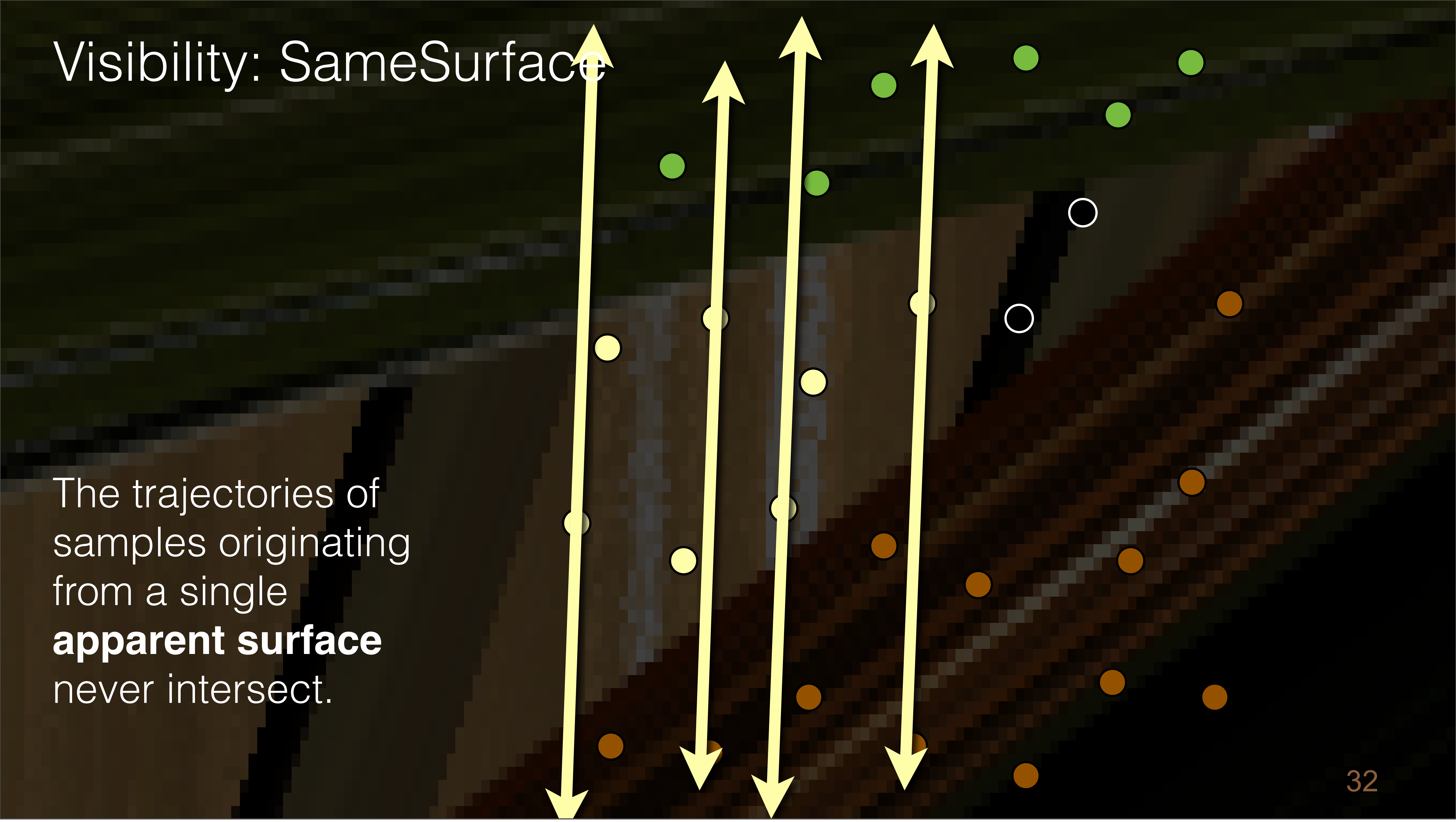
But observe that <animate>

if a set of samples don’t move relative to each other under viewpoint changes in such a way that they occlude on screen,

then their trajectories do not intersect in the light field.

Therefore, we can treat this set of samples as a single *apparent surface*.

So these samples form an apparent surface.
<animate>
And these..
<animate>
and these...
<animate>
<next>



Visibility: SameSurface

The trajectories of samples originating from a single **apparent surface** never intersect.

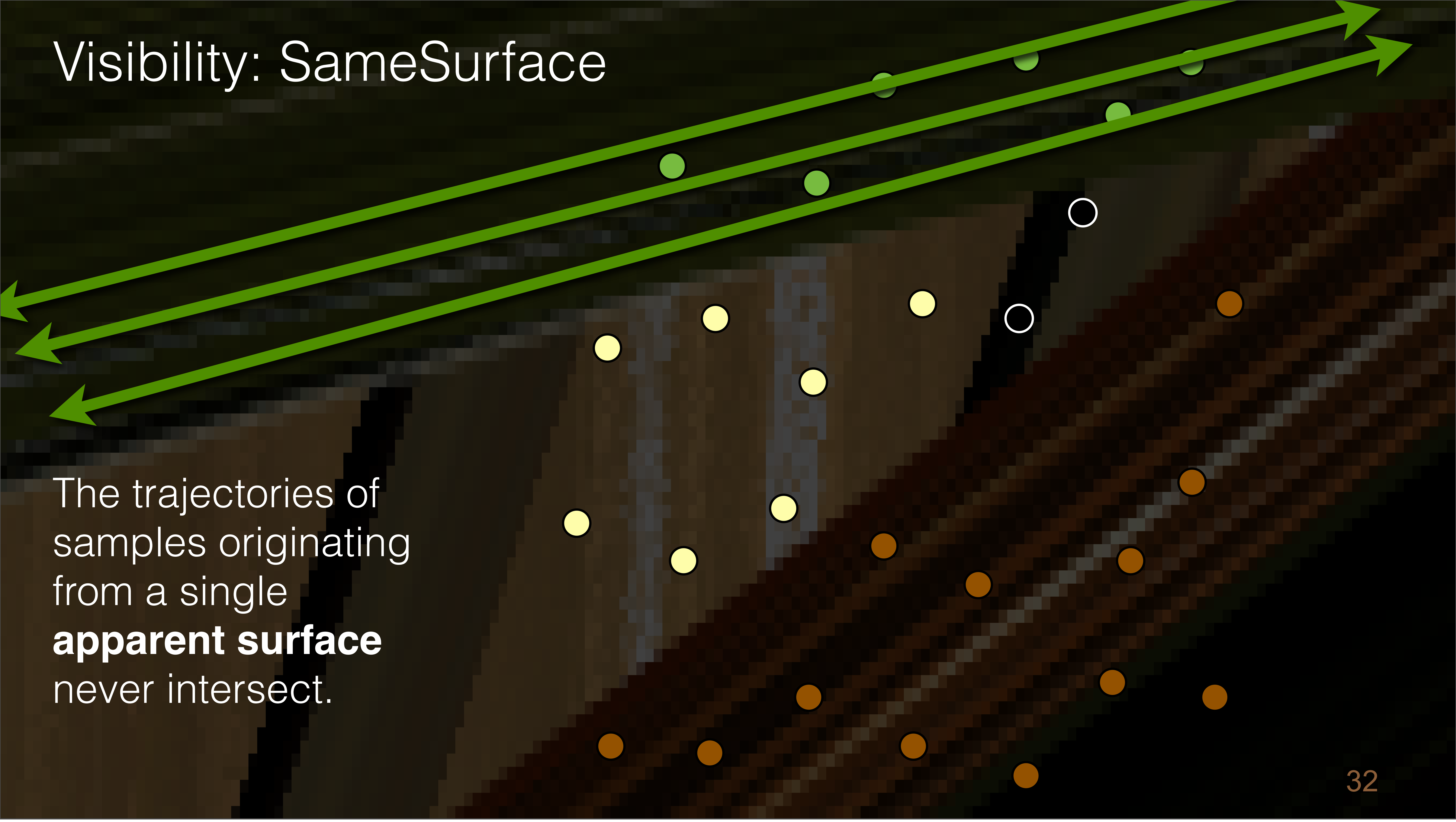
But observe that <animate>

if a set of samples don’t move relative to each other under viewpoint changes in such a way that they occlude on screen,

then their trajectories do not intersect in the light field.

Therefore, we can treat this set of samples as a single *apparent surface*.

So these samples form an apparent surface.
<animate>
And these..
<animate>
and these...
<animate>
<next>



Visibility: SameSurface

The trajectories of samples originating from a single **apparent surface** never intersect.

But observe that <animate>

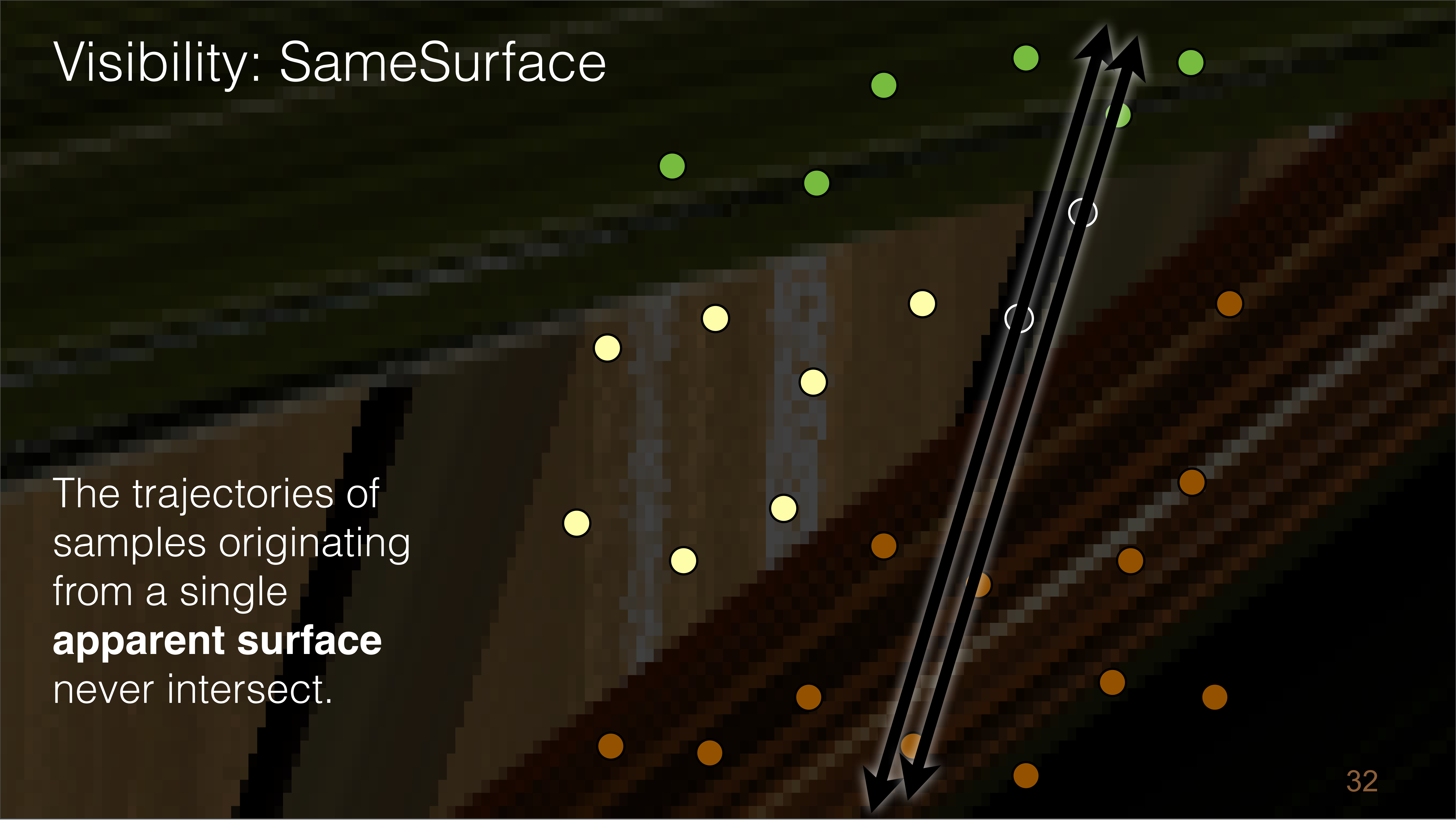
if a set of samples don’t move relative to each other under viewpoint changes in such a way that they occlude on screen,

then their trajectories do not intersect in the light field.

Therefore, we can treat this set of samples as a single *apparent surface*.

So these samples form an apparent surface.
<animate>
And these..
<animate>
and these...
<animate>
<next>

Visibility: SameSurface



The trajectories of samples originating from a single **apparent surface** never intersect.

But observe that <animate>

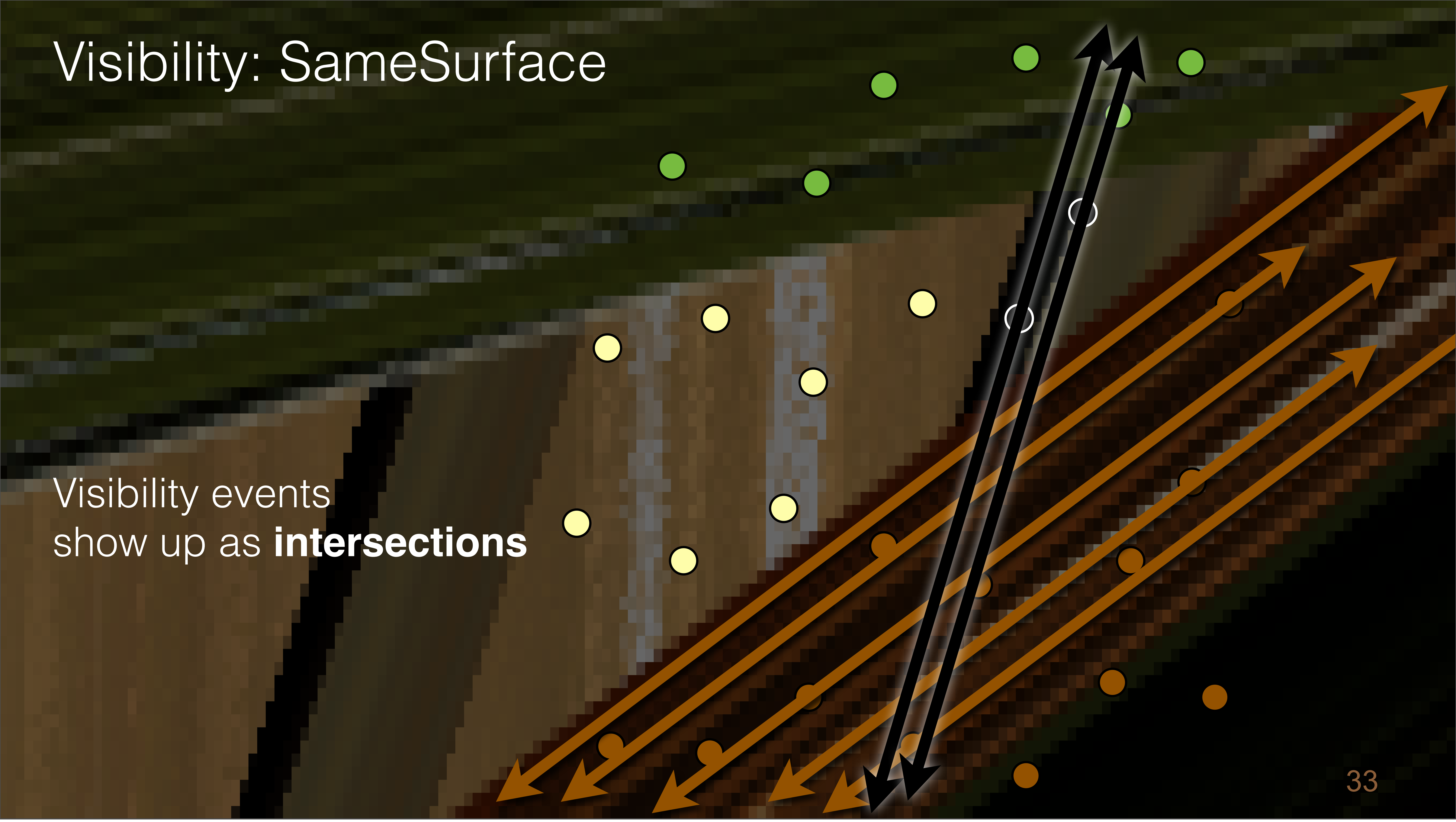
if a set of samples don’t move relative to each other under viewpoint changes in such a way that they occlude on screen,

then their trajectories do not intersect in the light field.

Therefore, we can treat this set of samples as a single *apparent surface*.

So these samples form an apparent surface.
<animate>
And these..
<animate>
and these...
<animate>
<next>

Visibility: SameSurface



Visibility events show up as **intersections**

Here, these two sets of trajectories intersect,

therefore, they must come from two distinct apparent surfaces.

And, since slope corresponds to inverse depth, we know which is in front.

<pause>

Ok, now that we know how to cluster samples into apparent surfaces, and sort them by depth,
we need to determine whether the samples that form the apparent surface actually covers a reconstruction location.

Visibility: Coverage

Here’s how we determine coverage in the case of two overlapping apparent surfaces.
The extension to additional apparent surfaces is straightforward.

<animate>
These are samples from two apparent surfaces after reprojection:
they are in screen space.

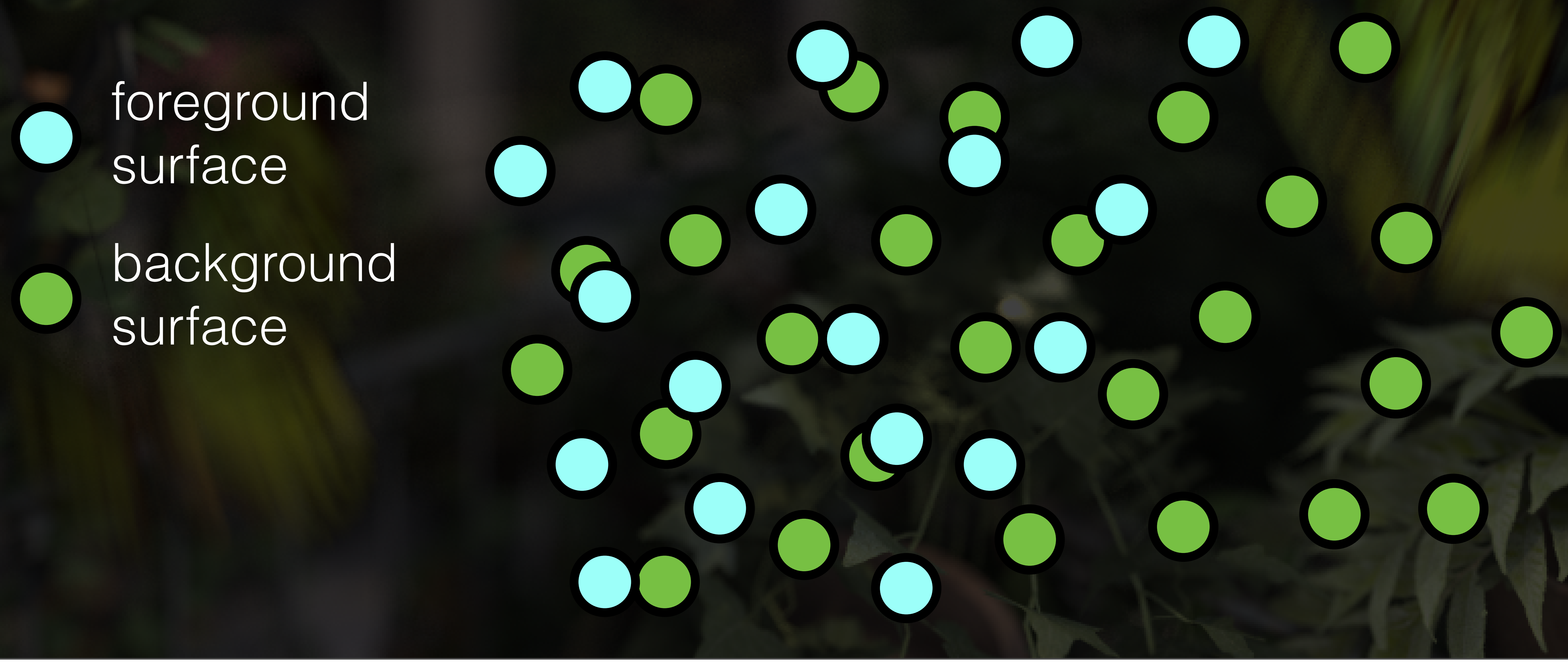
The question is, <animate>
at the reconstruction location shown in red,
does the foreground apparent surface cover the one in the background one?

What we do is search a screen space neighborhood of some radius R, <animate>
and check if there exists a triplet of foreground samples that spans a triangle <animate> containing the reconstruction location.
If there is no such triangle, we go on the next apparent surface.

Note that this is just a simple loop over the apparent surfaces and samples: there is no actual geometry.

Visibility: Coverage

Does foreground apparent surface cover reconstruction location?



● foreground surface

● background surface

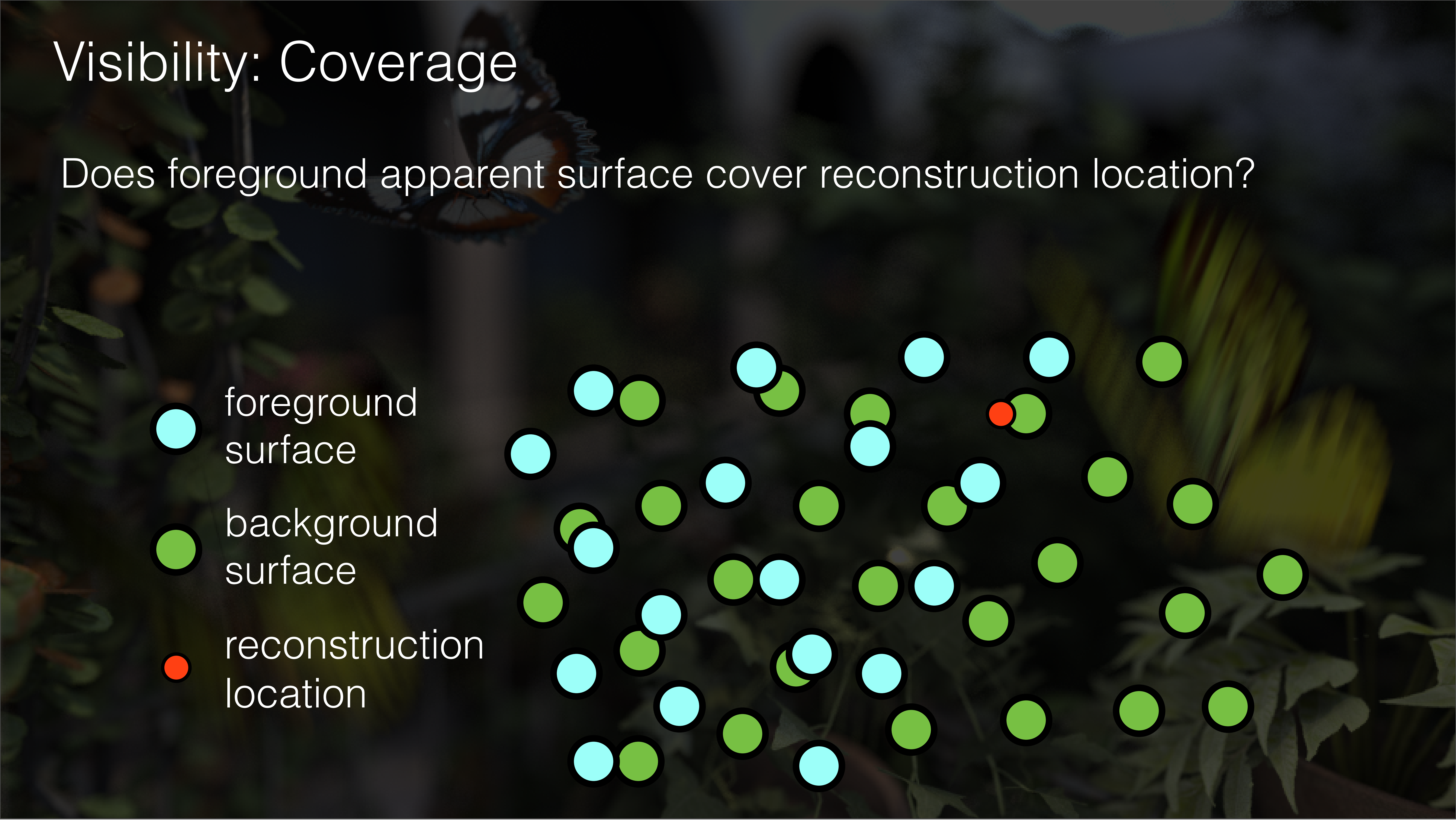
Here’s how we determine coverage in the case of two overlapping apparent surfaces.
The extension to additional apparent surfaces is straightforward.

<animate>
These are samples from two apparent surfaces after reprojection:
they are in screen space.

The question is, <animate>
at the reconstruction location shown in red,
does the foreground apparent surface cover the one in the background one?

What we do is search a screen space neighborhood of some radius R, <animate>
and check if there exists a triplet of foreground samples that spans a triangle <animate> containing the reconstruction location.
If there is no such triangle, we go on the next apparent surface.

Note that this is just a simple loop over the apparent surfaces and samples: there is no actual geometry.



Visibility: Coverage

Does foreground apparent surface cover reconstruction location?

● foreground surface

● background surface

● reconstruction location

Here’s how we determine coverage in the case of two overlapping apparent surfaces.
The extension to additional apparent surfaces is straightforward.

<animate>
These are samples from two apparent surfaces after reprojection:
they are in screen space.

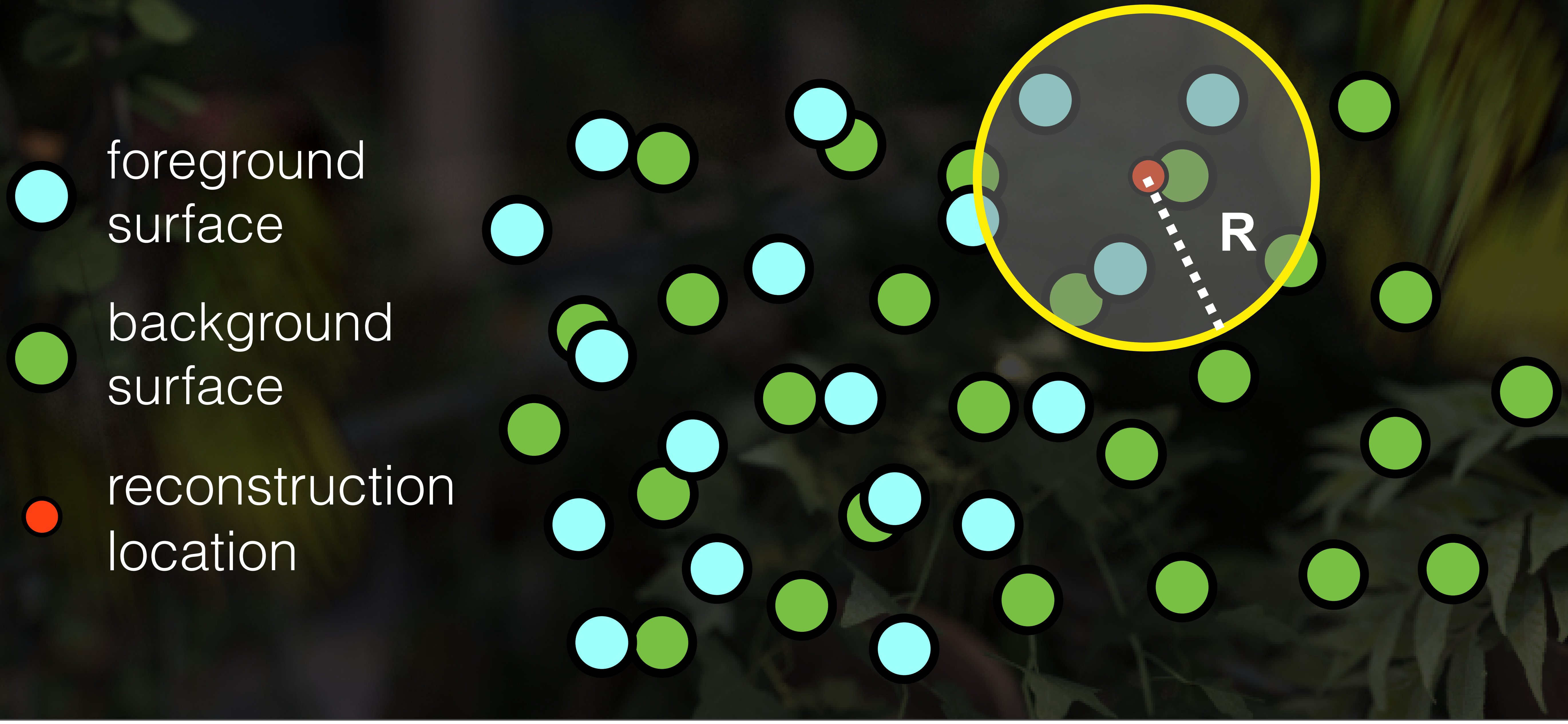
The question is, <animate>
at the reconstruction location shown in red,
does the foreground apparent surface cover the one in the background one?

What we do is search a screen space neighborhood of some radius R, <animate>
and check if there exists a triplet of foreground samples that spans a triangle <animate> containing the reconstruction location.
If there is no such triangle, we go on the next apparent surface.

Note that this is just a simple loop over the apparent surfaces and samples: there is no actual geometry.

Visibility: Coverage

Does foreground apparent surface cover reconstruction location?



● foreground surface

● background surface

● reconstruction location

Here's how we determine coverage in the case of two overlapping apparent surfaces.
The extension to additional apparent surfaces is straightforward.

<animate>
These are samples from two apparent surfaces after reprojection:
they are in screen space.

The question is, <animate>
at the reconstruction location shown in red,
does the foreground apparent surface cover the one in the background one?

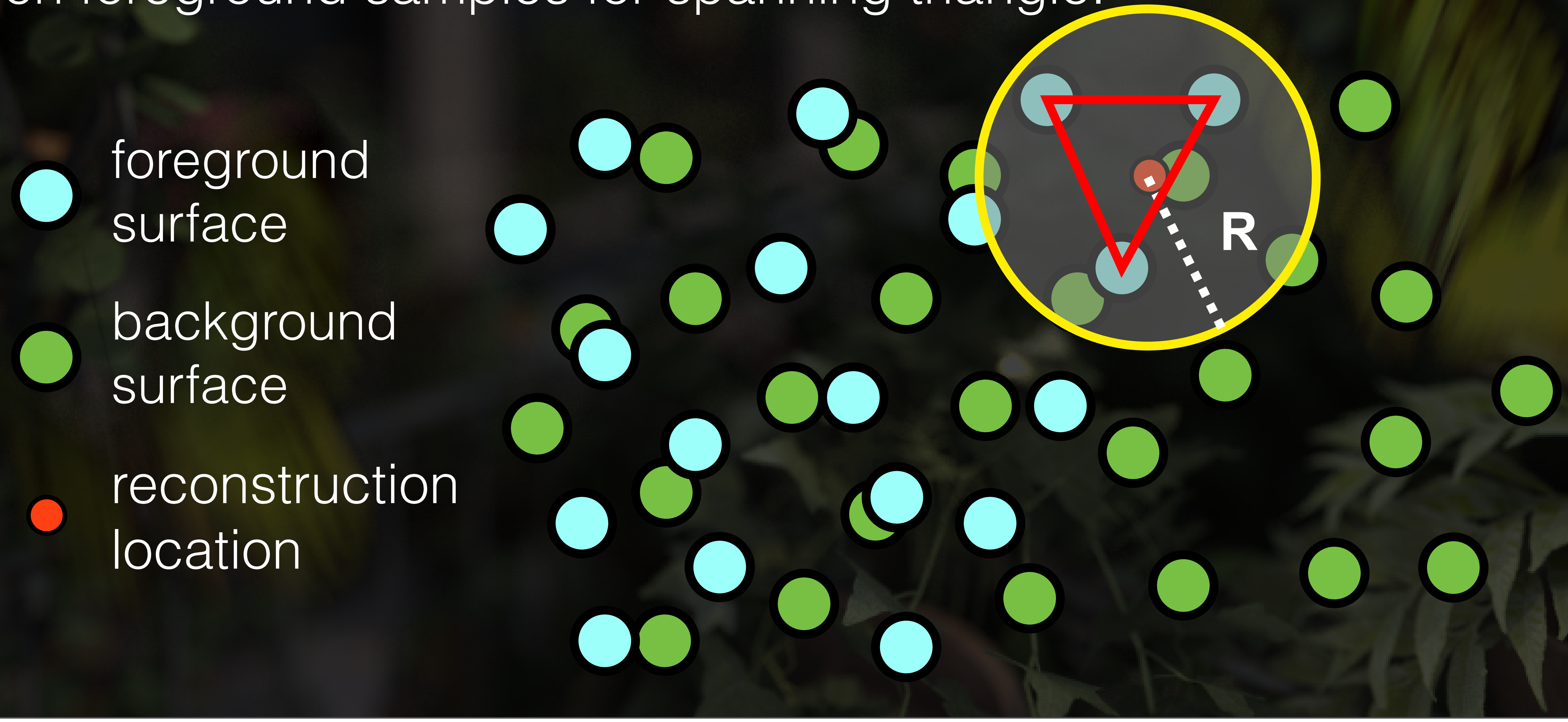
What we do is search a screen space neighborhood of some radius R, <animate>
and check if there exists a triplet of foreground samples that spans a triangle <animate> containing the reconstruction location.
If there is no such triangle, we go on the next apparent surface.

Note that this is just a simple loop over the apparent surfaces and samples: there is no actual geometry.

Visibility: Coverage

Does foreground apparent surface cover reconstruction location?

Search foreground samples for spanning triangle.



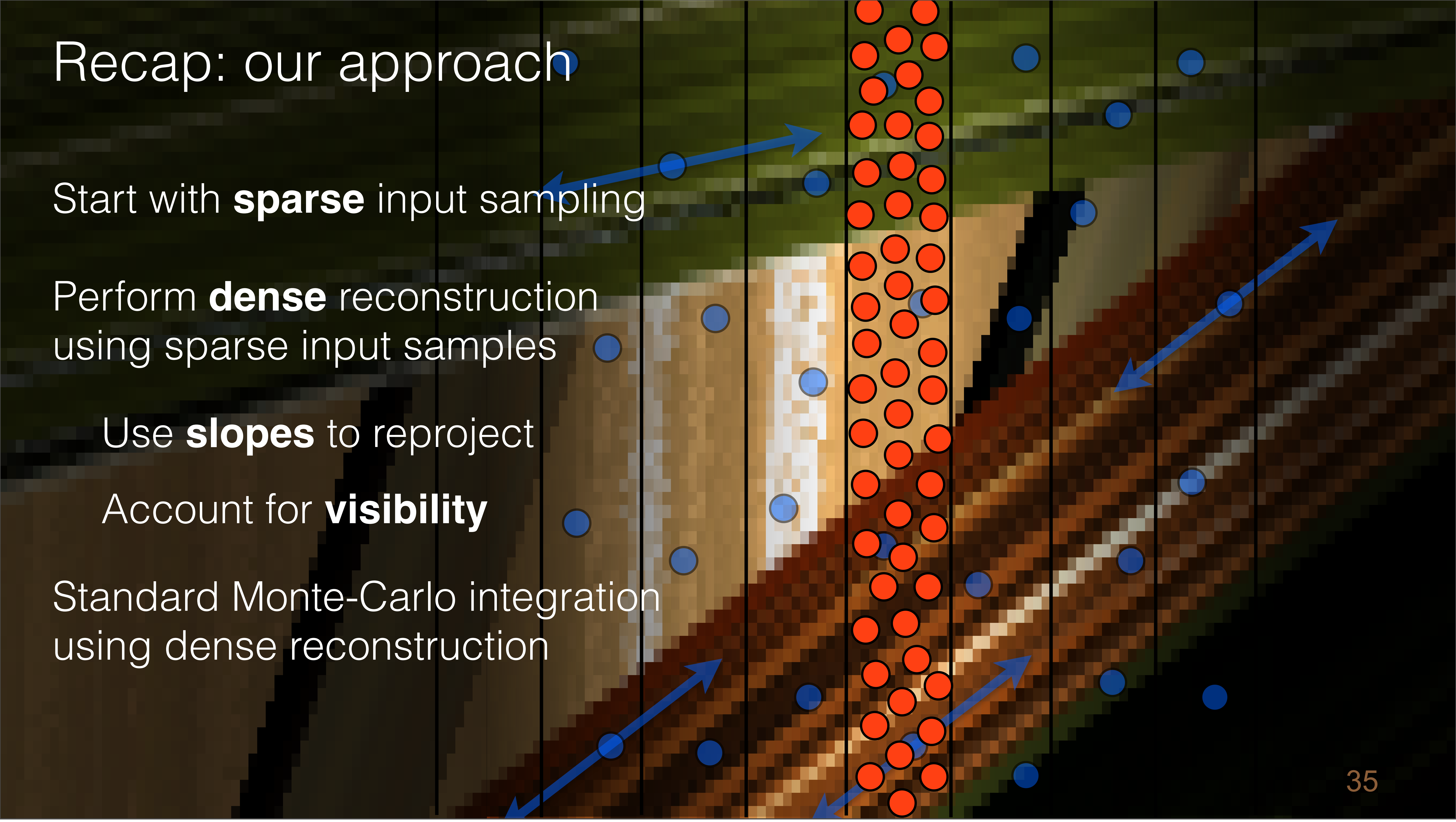
Here's how we determine coverage in the case of two overlapping apparent surfaces.
The extension to additional apparent surfaces is straightforward.

<animate>
These are samples from two apparent surfaces after reprojection:
they are in screen space.

The question is, <animate>
at the reconstruction location shown in red,
does the foreground apparent surface cover the one in the background one?

What we do is search a screen space neighborhood of some radius R, <animate>
and check if there exists a triplet of foreground samples that spans a triangle <animate> containing the reconstruction location.
If there is no such triangle, we go on the next apparent surface.

Note that this is just a simple loop over the apparent surfaces and samples: there is no actual geometry.



Recap: our approach

Start with **sparse** input sampling

Perform **dense** reconstruction using sparse input samples

- Use **slopes** to reproject
- Account for **visibility**

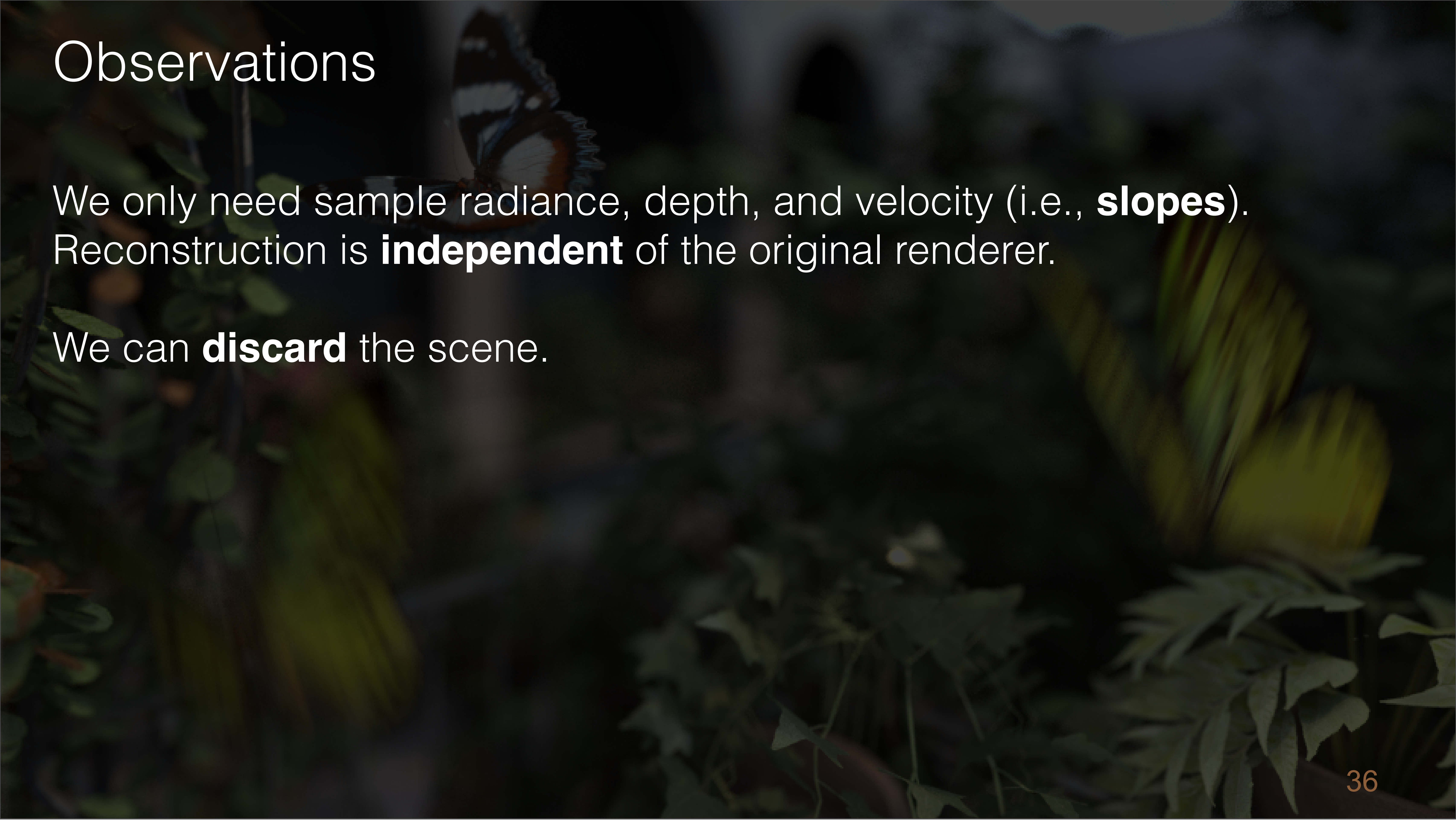
Standard Monte-Carlo integration using dense reconstruction

Ok, now we have all the pieces.

We start with a sparse sampling of the scene, where each sample comes with an attached radiance and slope.
We perform a dense reconstruction at many locations per pixel.

To determine the radiance at each reconstruction location,
we reproject the input samples to the lens and time coordinate of the reconstruction location,
cluster them into apparent surfaces based on their slopes,
and determine coverage by searching for a spanning triangle within the expected dispersion radius.

Before I go on to show results, allow me to make a few observations.



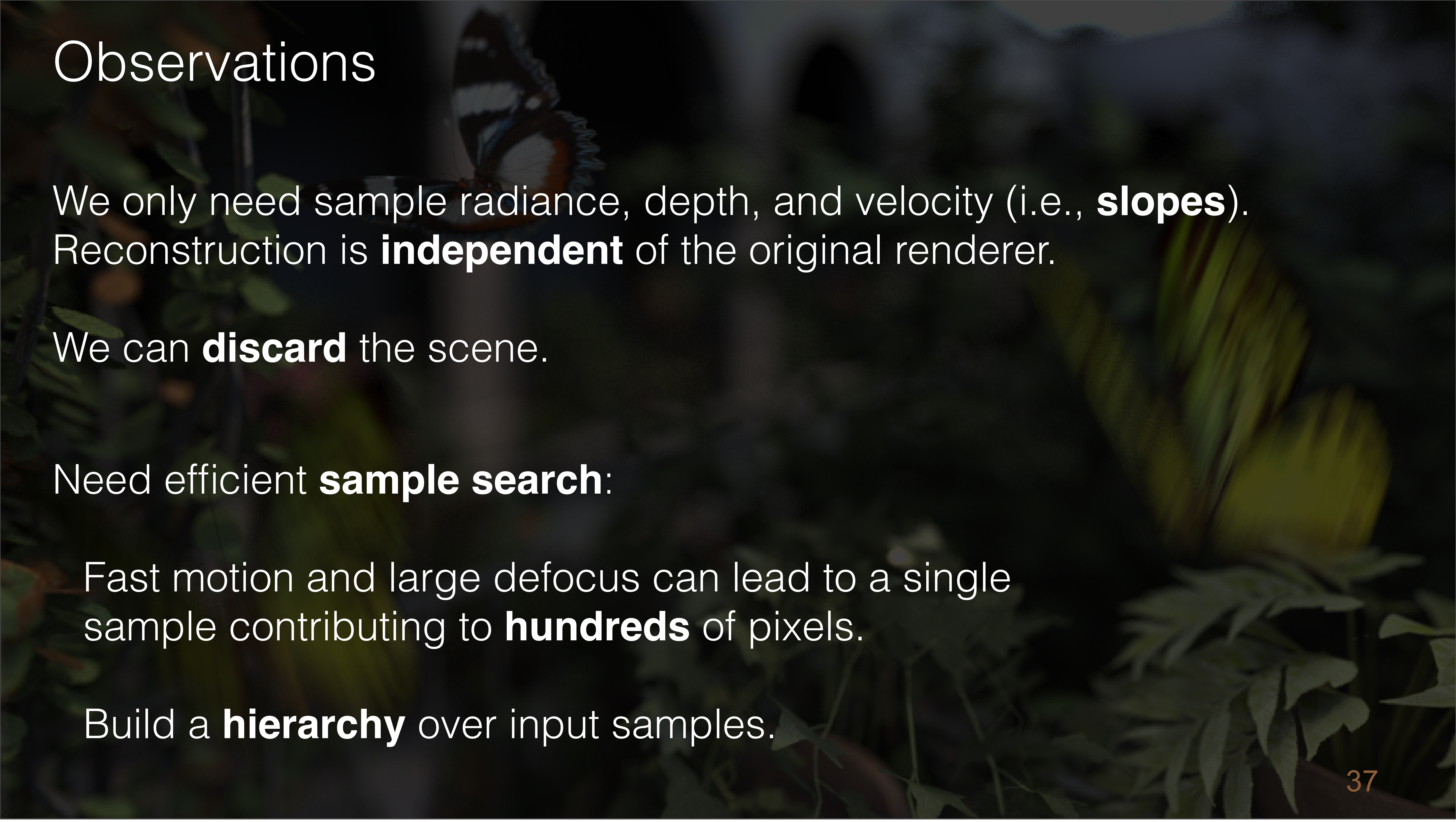
Observations

We only need sample radiance, depth, and velocity (i.e., **slopes**).
Reconstruction is **independent** of the original renderer.

We can **discard** the scene.

I stressed several times that we only need the sample radiances, depths, and velocities (in other words, slopes).
This makes our reconstruction algorithm independent of which renderer we use.

In fact, besides camera parameters, we don’t even need the scene.



Observations

We only need sample radiance, depth, and velocity (i.e., **slopes**). Reconstruction is **independent** of the original renderer.

We can **discard** the scene.

Need efficient **sample search**:

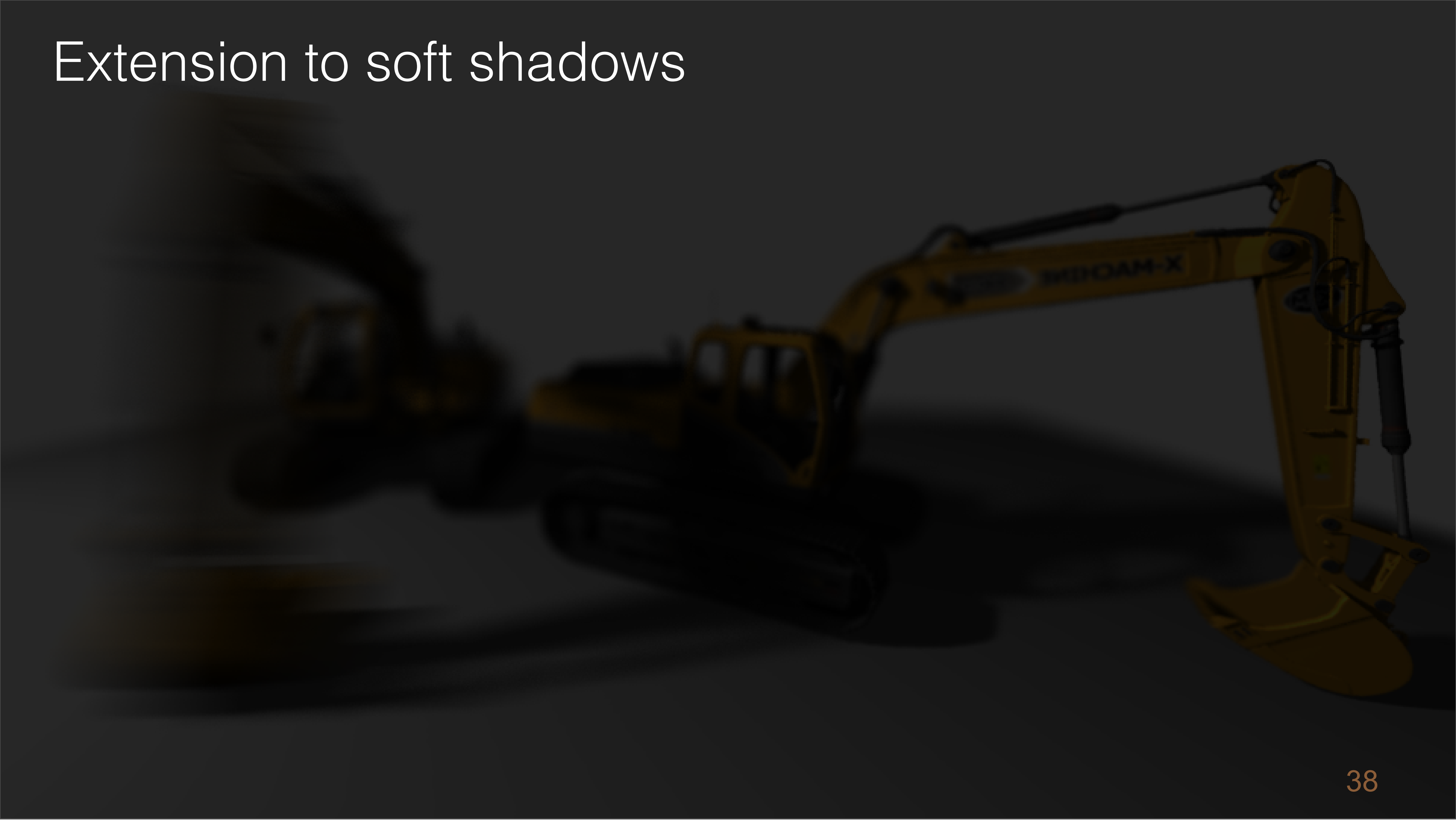
Fast motion and large defocus can lead to a single sample contributing to **hundreds** of pixels.

Build a **hierarchy** over input samples.

I also skipped over the fact that *for* our method to outperform brute force sampling,
we need to be able to efficiently search for all the reprojected samples within the radius R.

This is challenging because with fast motion and large defocus blurs, a single sample can be smeared across large parts of the screen.

For this, we used a motion bounding volume hierarchy.
The details on the hierarchy are in the paper.



Extension to soft shadows

Our method is not limited to temporal light fields inside cameras.

<animate>

In particular, notice that the geometry between an area light source and shadow receiver is very similar to the geometry between a lens and a sensor.

When rendering soft shadows cast by an extended light source, we need to integrate over the area of the light source to determine what fraction of the light is visible at the receiver.

To use our method, we first render a sparse, stochastic, *shadow* light field:

for a set of rays originating on the light, what is the distance to the closest object in the scene?

<animate>

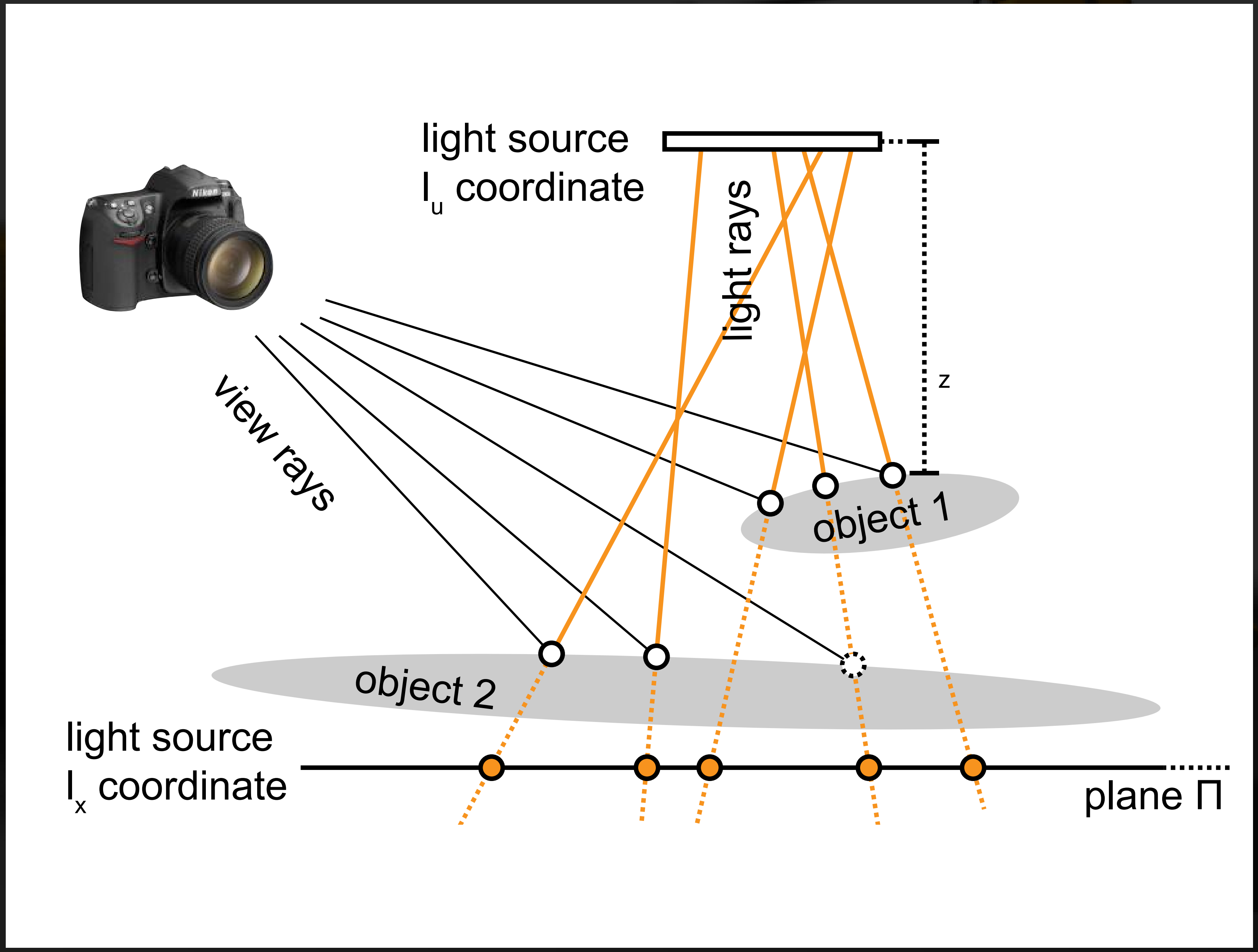
Then, when we render, we can reconstruct the depth function at a high resolution and *compare depths* to determine how much of the area light is visible.

<animate>

Extension to soft shadows

An **area light** is very much like a **lens**.

lens ~ light, sensor ~ virtual plane



Our method is not limited to temporal light fields inside cameras.

<animate>

In particular, notice that the geometry between an area light source and shadow receiver is very similar to the geometry between a lens and a sensor.

When rendering soft shadows cast by an extended light source, we need to integrate over the area of the light source to determine what fraction of the light is visible at the receiver.

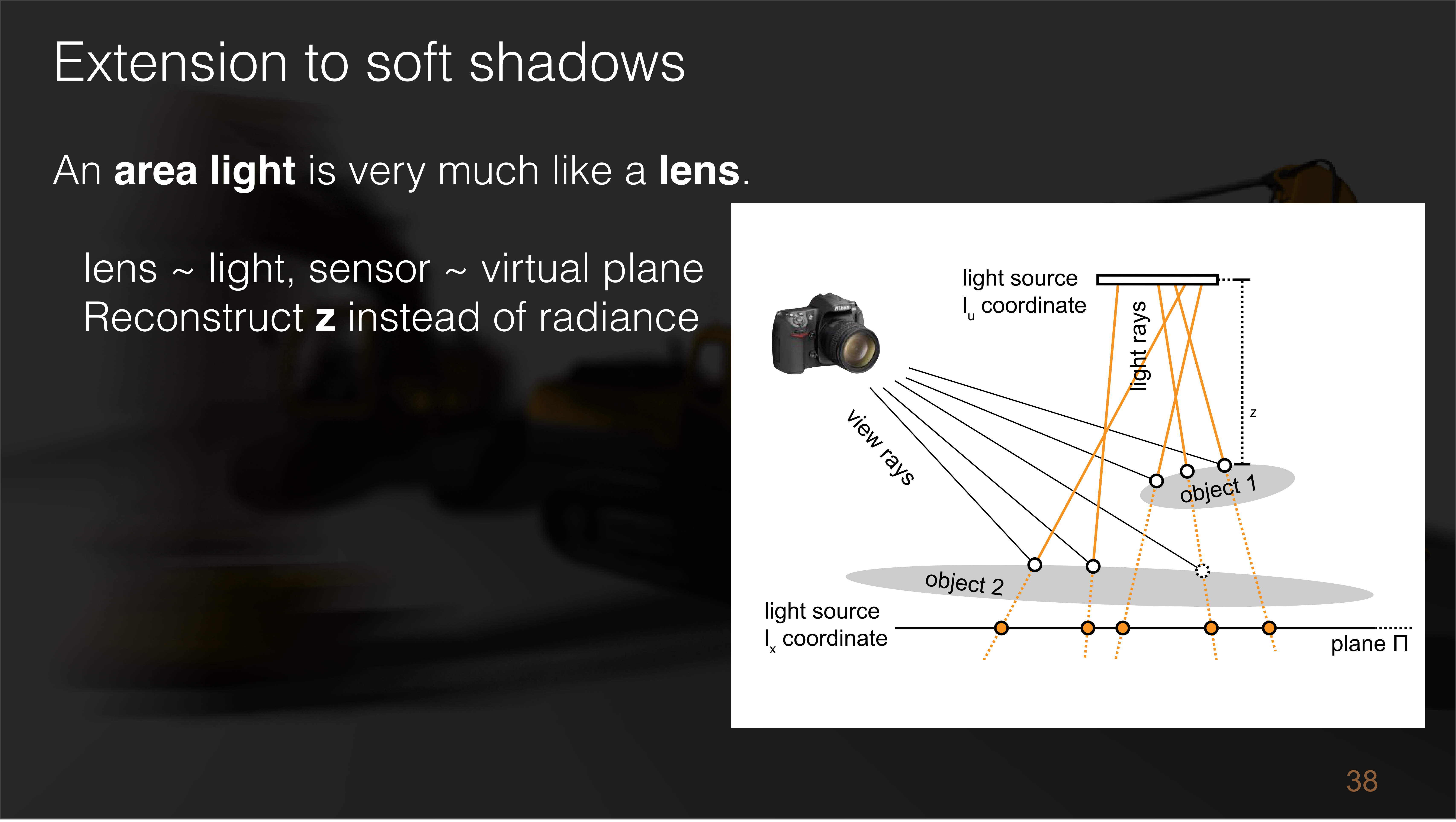
To use our method, we first render a sparse, stochastic, *shadow* light field:

for a set of rays originating on the light, what is the distance to the closest object in the scene?

<animate>

Then, when we render, we can reconstruct the depth function at a high resolution and *compare depths* to determine how much of the area light is visible.

<animate>



Extension to soft shadows

An **area light** is very much like a **lens**.

lens ~ light, sensor ~ virtual plane
Reconstruct **z** instead of radiance

Our method is not limited to temporal light fields inside cameras.

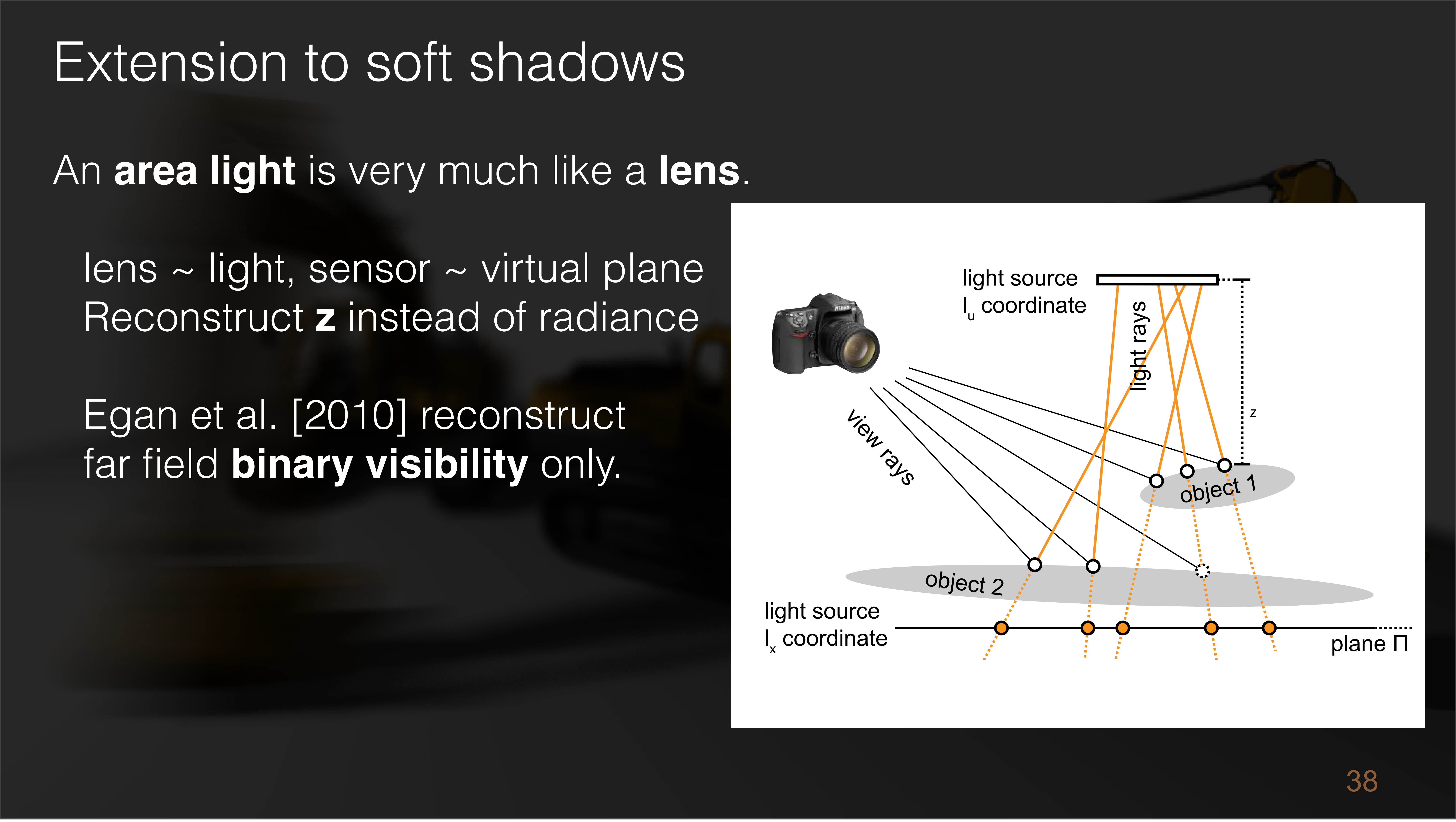
<animate>
In particular, notice that the geometry between an area light source and shadow receiver is very similar to the geometry between a lens and a sensor.
When rendering soft shadows cast by an extended light source, we need to integrate over the area of the light source to determine what fraction of the light is visible at the receiver.

To use our method, we first render a sparse, stochastic, *shadow* light field:

for a set of rays originating on the light, what is the distance to the closest object in the scene?

<animate>
Then, when we render, we can reconstruct the depth function at a high resolution and *compare depths* to determine how much of the area light is visible.

<animate>



Extension to soft shadows

An **area light** is very much like a **lens**.

lens ~ light, sensor ~ virtual plane
Reconstruct **z** instead of radiance

Egan et al. [2010] reconstruct
far field **binary visibility** only.

38

Our method is not limited to temporal light fields inside cameras.

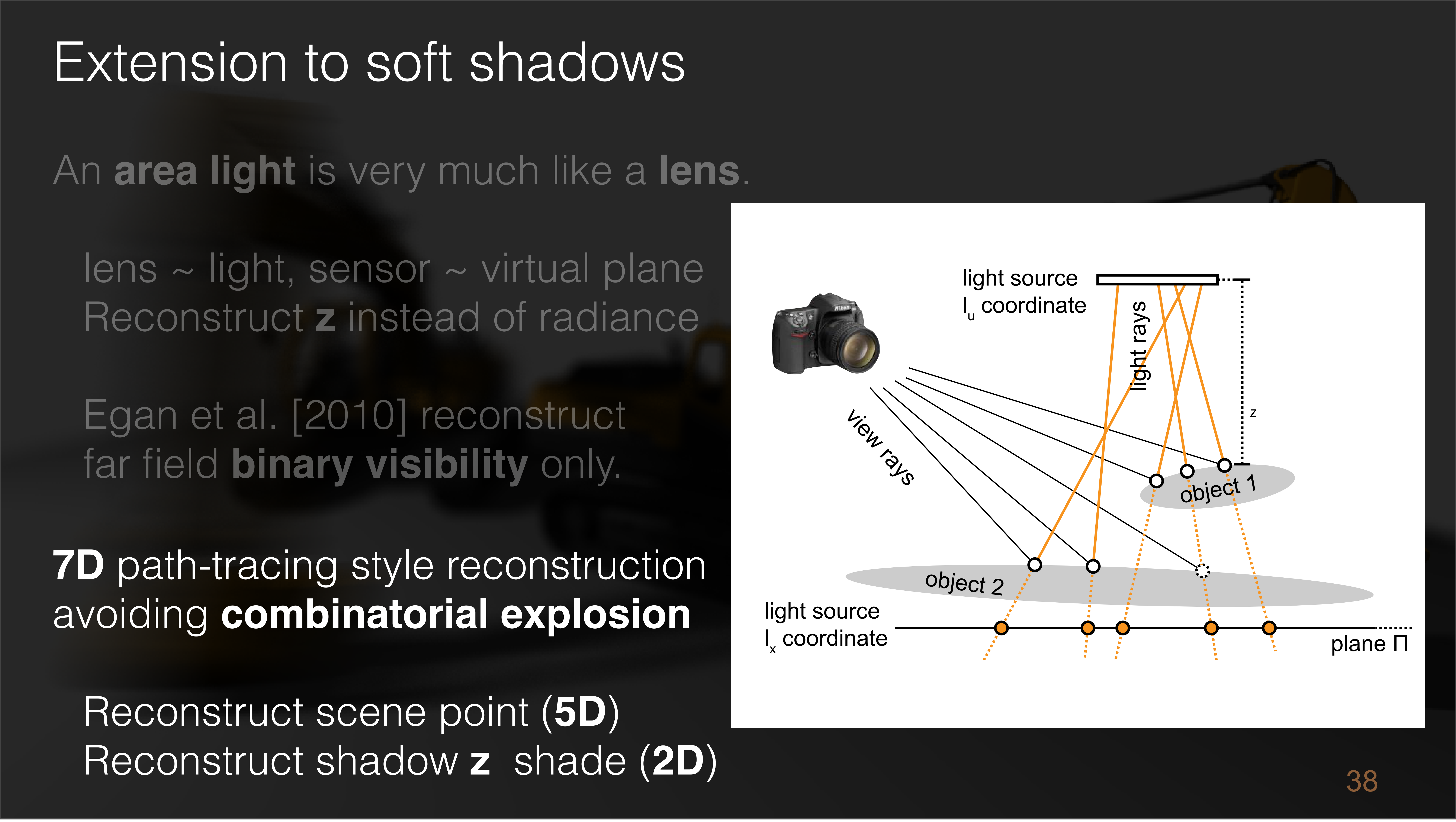
<animate>
In particular, notice that the geometry between an area light source and shadow receiver is very similar to the geometry between a lens and a sensor.
When rendering soft shadows cast by an extended light source, we need to integrate over the area of the light source to determine what fraction of the light is visible at the receiver.

To use our method, we first render a sparse, stochastic, *shadow* light field:

for a set of rays originating on the light, what is the distance to the closest object in the scene?

<animate>
Then, when we render, we can reconstruct the depth function at a high resolution and *compare depths* to determine how much of the area light is visible.

<animate>



Extension to soft shadows

An **area light** is very much like a **lens**.

lens ~ light, sensor ~ virtual plane
Reconstruct **z** instead of radiance

Egan et al. [2010] reconstruct
far field **binary visibility** only.

7D path-tracing style reconstruction
avoiding **combinatorial explosion**

Reconstruct scene point (**5D**)
Reconstruct shadow **z** shade (**2D**)

Our method is not limited to temporal light fields inside cameras.

<animate>
In particular, notice that the geometry between an area light source and shadow receiver is very similar to the geometry between a lens and a sensor.
When rendering soft shadows cast by an extended light source, we need to integrate over the area of the light source to determine what fraction of the light is visible at the receiver.

To use our method, we first render a sparse, stochastic, *shadow* light field:

for a set of rays originating on the light, what is the distance to the closest object in the scene?

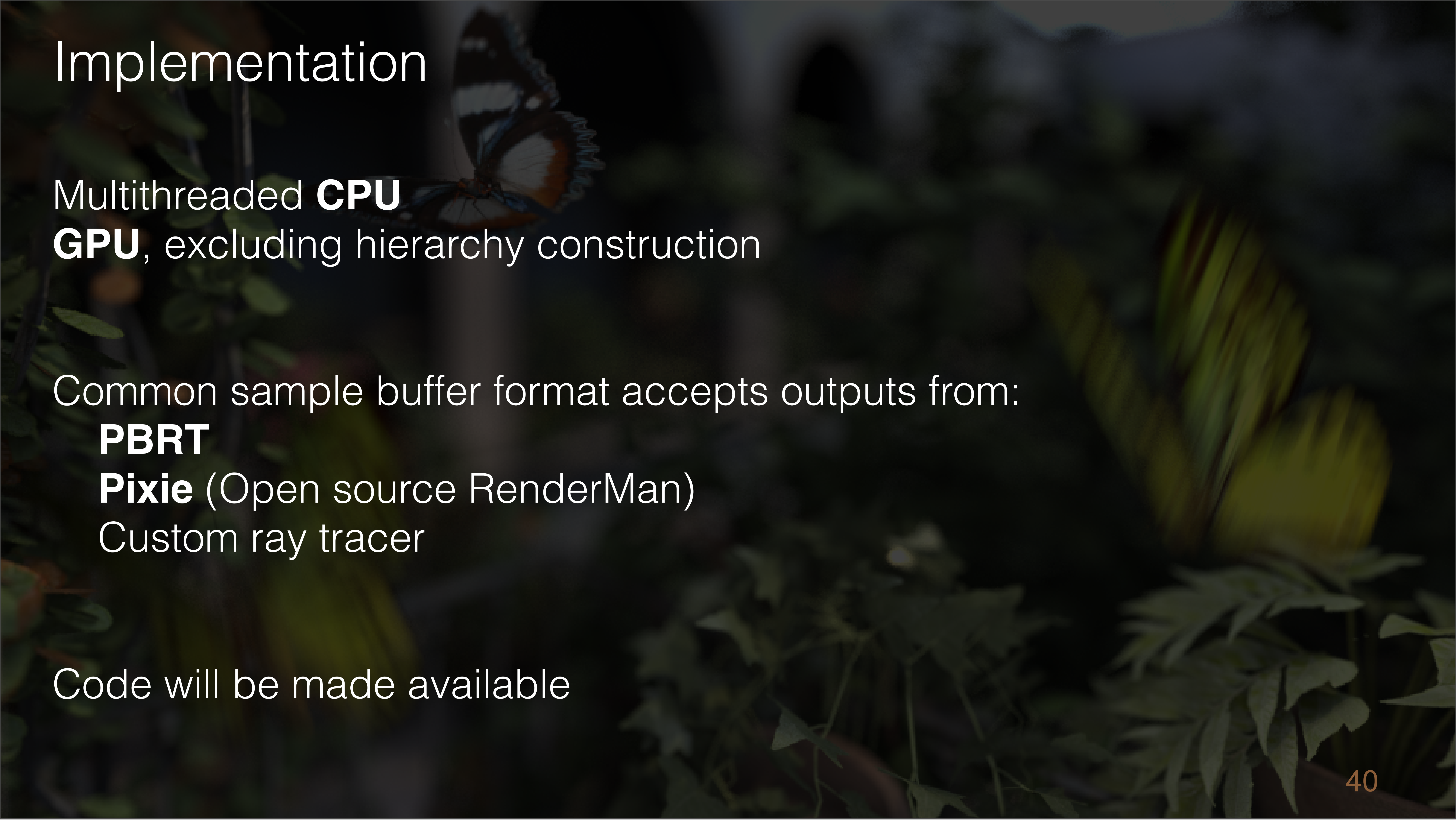
<animate>
Then, when we render, we can reconstruct the depth function at a high resolution and *compare depths* to determine how much of the area light is visible.

<animate>

Results

39

Now let’s look at some results.



Implementation

Multithreaded **CPU**
GPU, excluding hierarchy construction

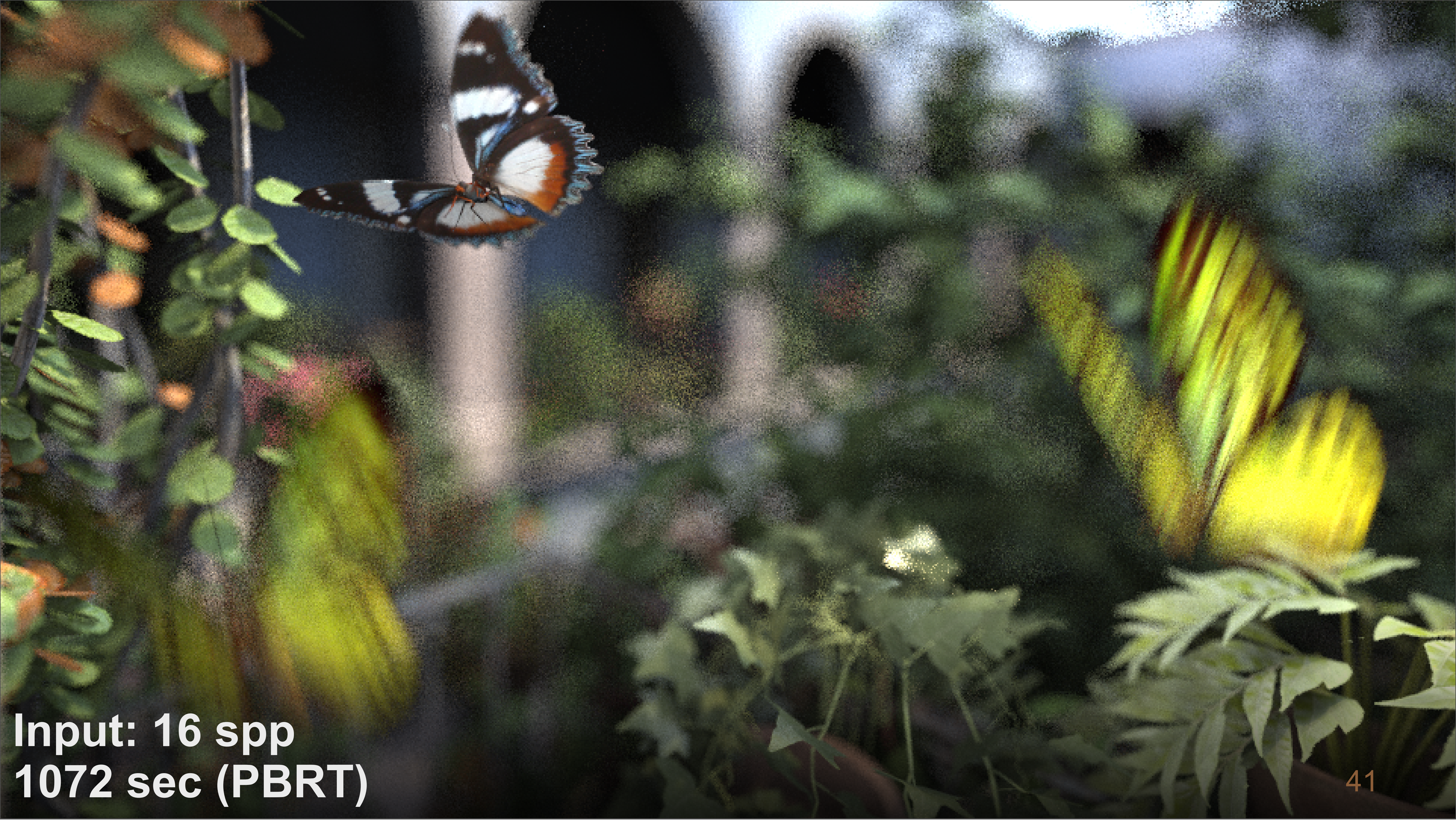
Common sample buffer format accepts outputs from:
- **PBRT**
- **Pixie** (Open source RenderMan)
- Custom ray tracer

Code will be made available

We implemented two versions of our algorithm, a multithreaded CPU version
and a GPU version that does everything except building the hierarchy.
The timings I give will be from the GPU version.

As I mentioned before, we don’t even need the scene anymore,
Our common sample buffer format lets us use outputs from PBRT, Pixie, and a custom ray tracer which we used to do soft shadows.

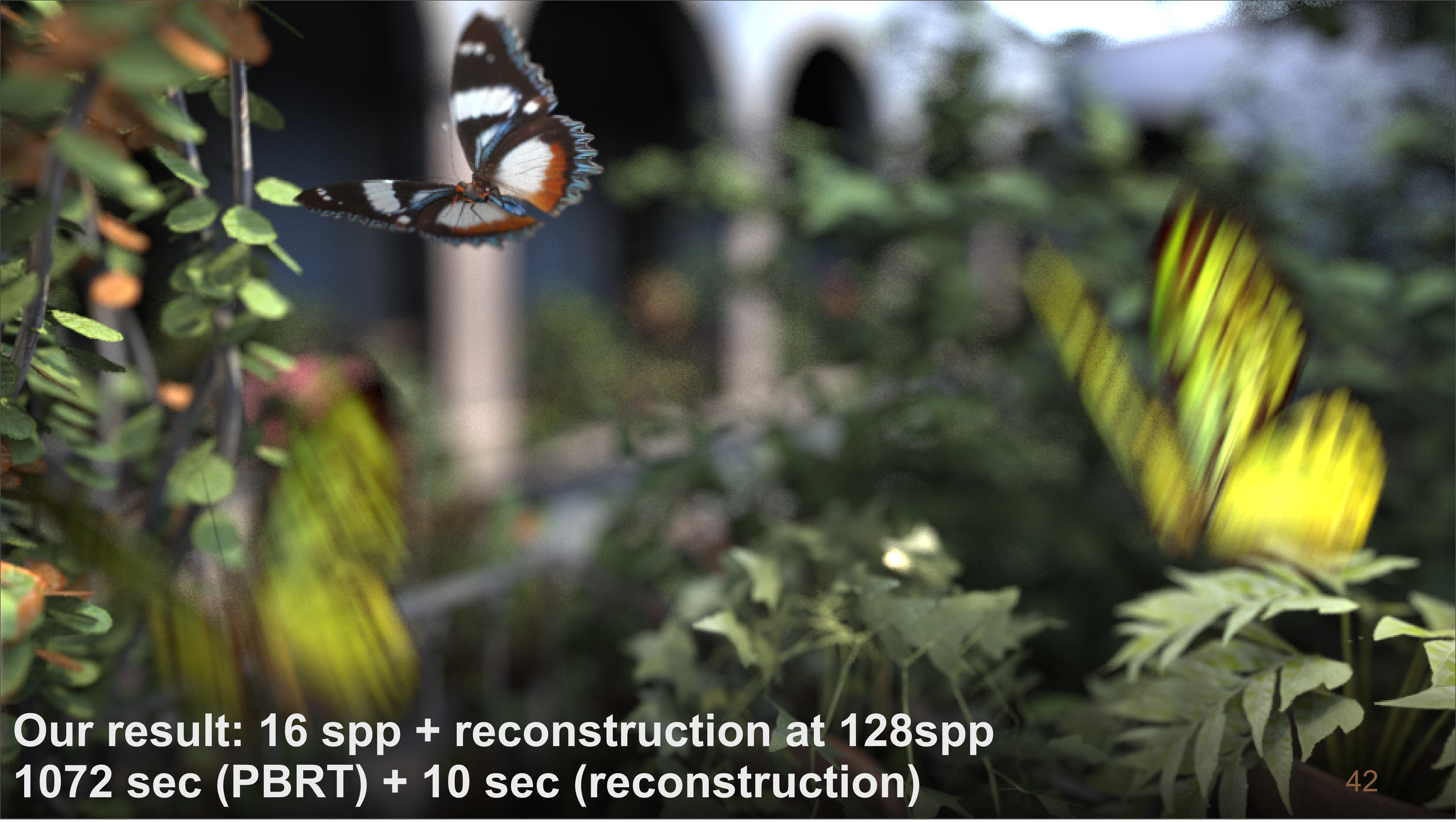
I also want to mention that our code will be made available on the web.



Input: 16 spp

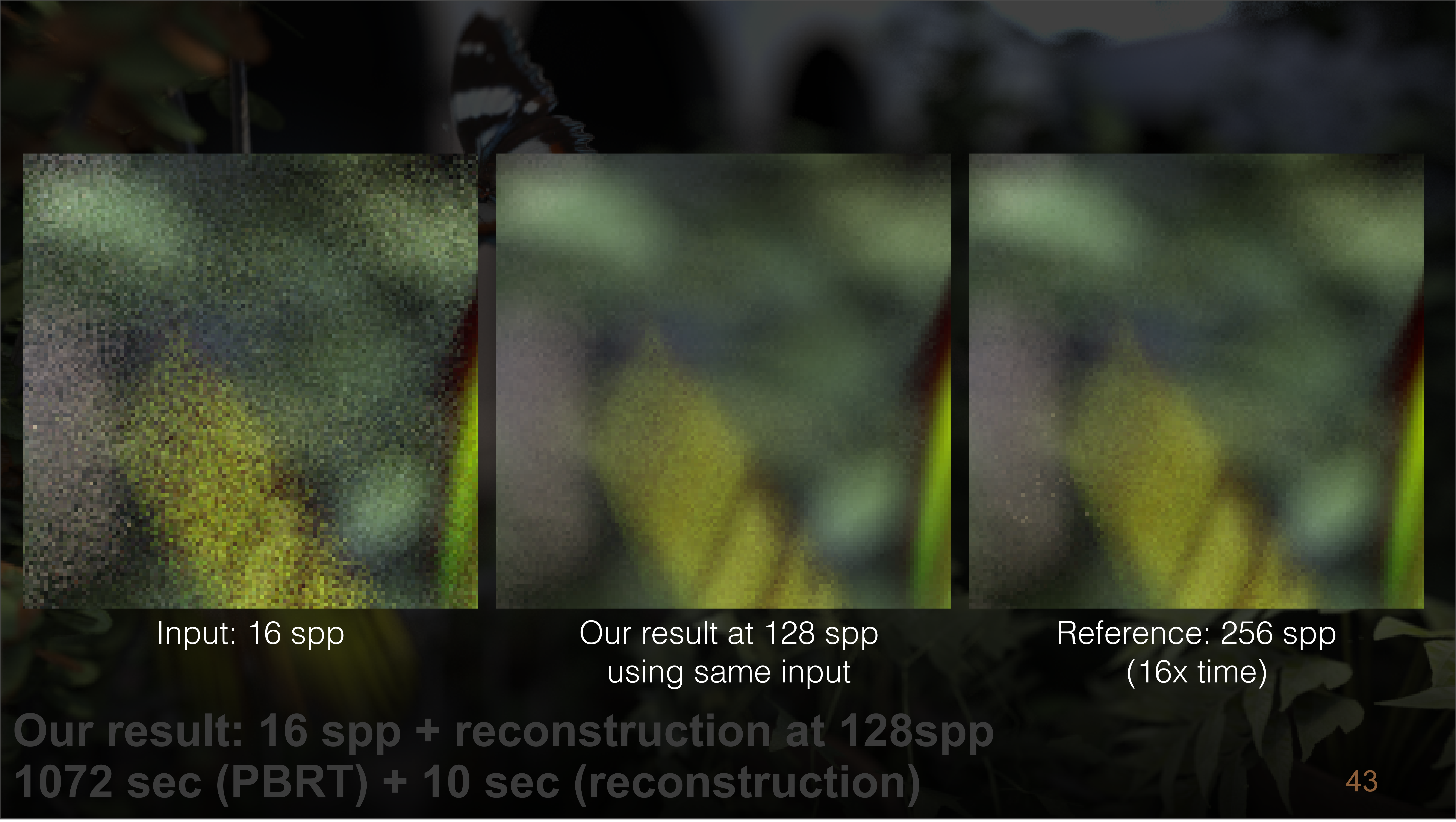
1072 sec (PBRT)

This is the butterfly scene I showed earlier at 16 samples per pixel.



Our result: 16 spp + reconstruction at 128spp
1072 sec (PBRT) + 10 sec (reconstruction)

Using the same samples, we reconstructed the light field at 128 samples per pixel using only 10 seconds with our GPU implementation.



Input: 16 spp

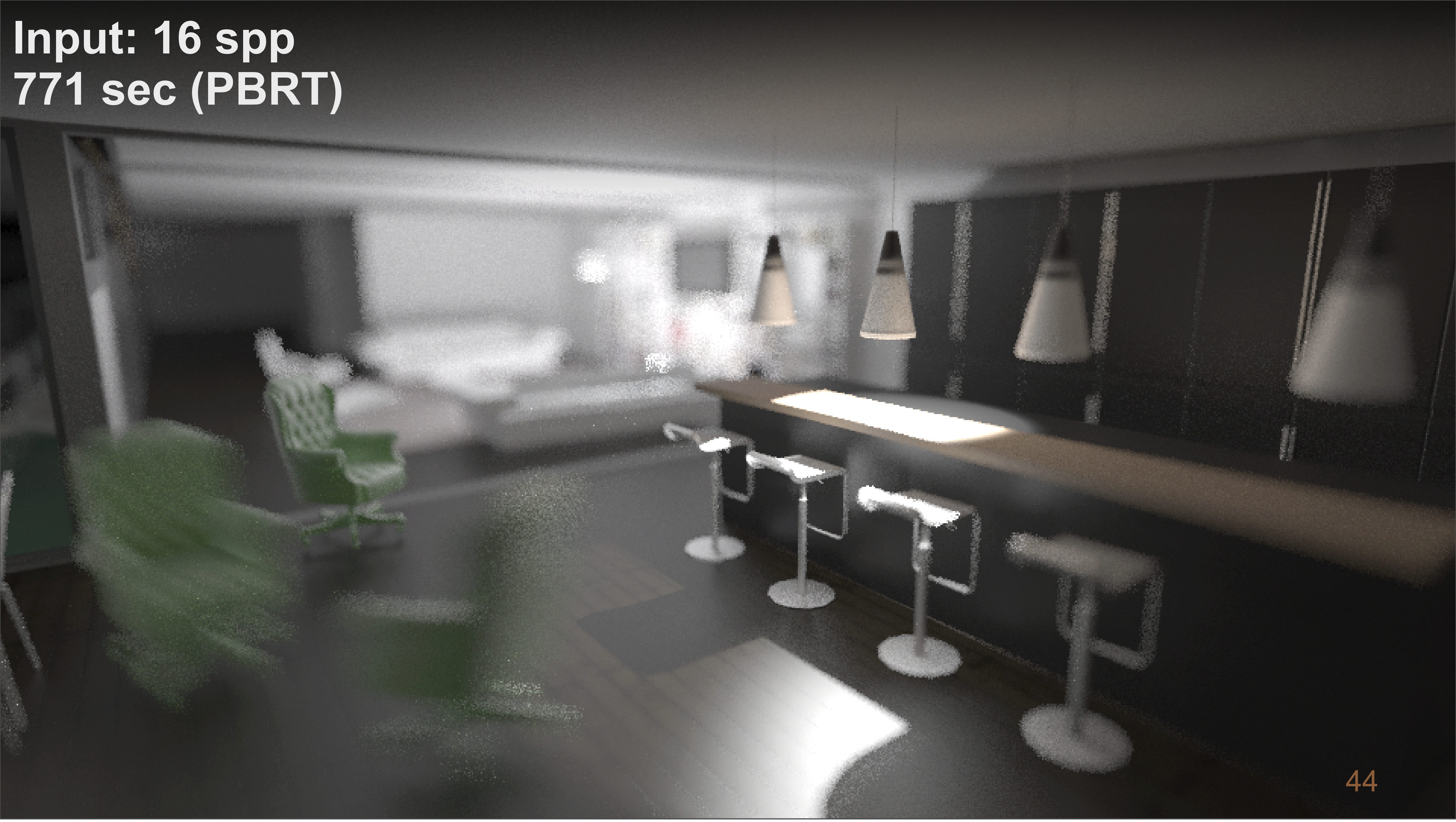
Our result at 128 spp
using same input

Reference: 256 spp
(16x time)

**Our result: 16 spp + reconstruction at 128spp
1072 sec (PBRT) + 10 sec (reconstruction)**

Here’s a closeup to compare.
Notice the noise in the blurry background and moving butterfly when using 16 samples per pixel.

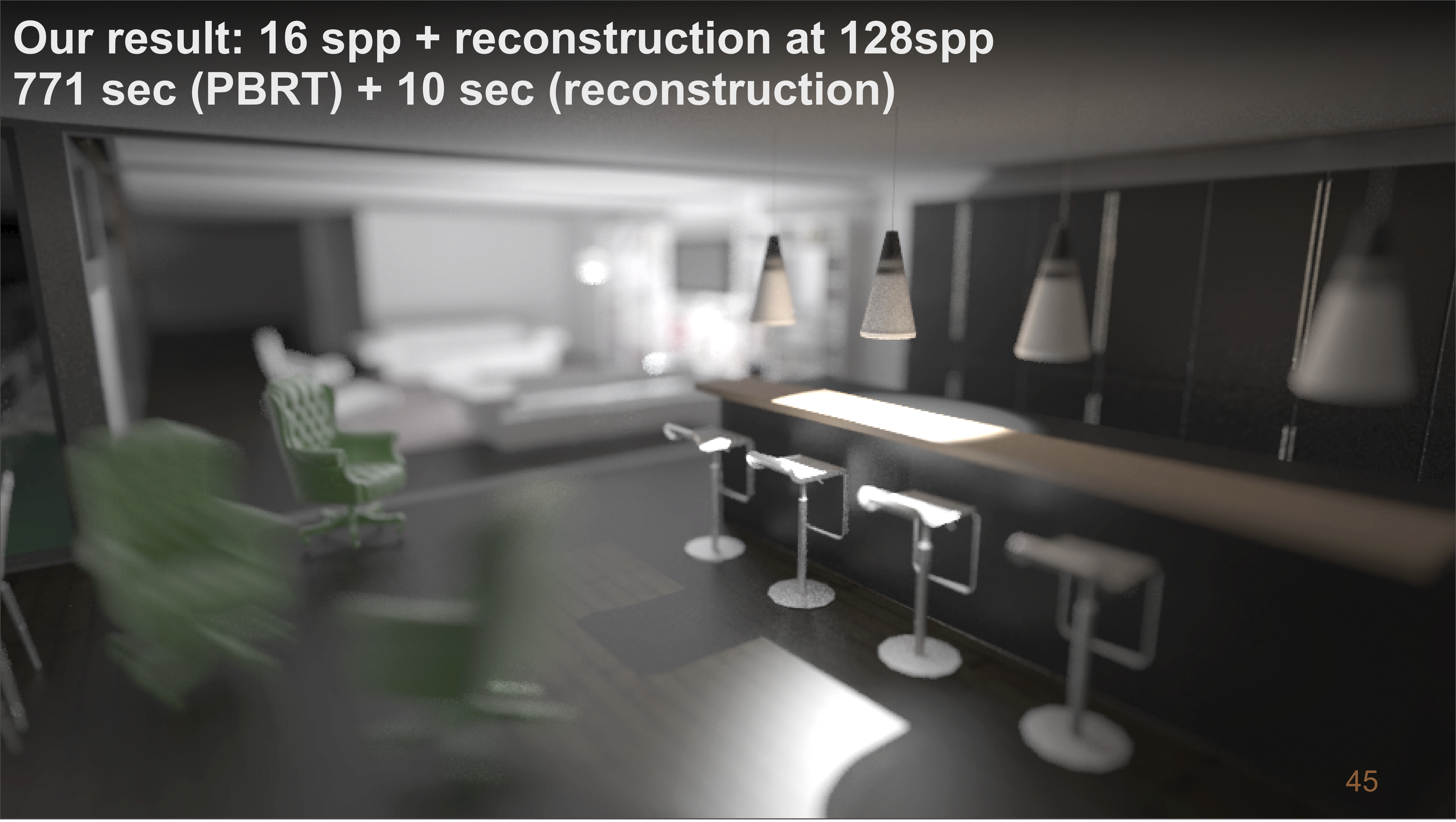
Our reconstruction at 128 spp matches the reference quite well and uses only 1/16 the time.



Input: 16 spp
771 sec (PBRT)

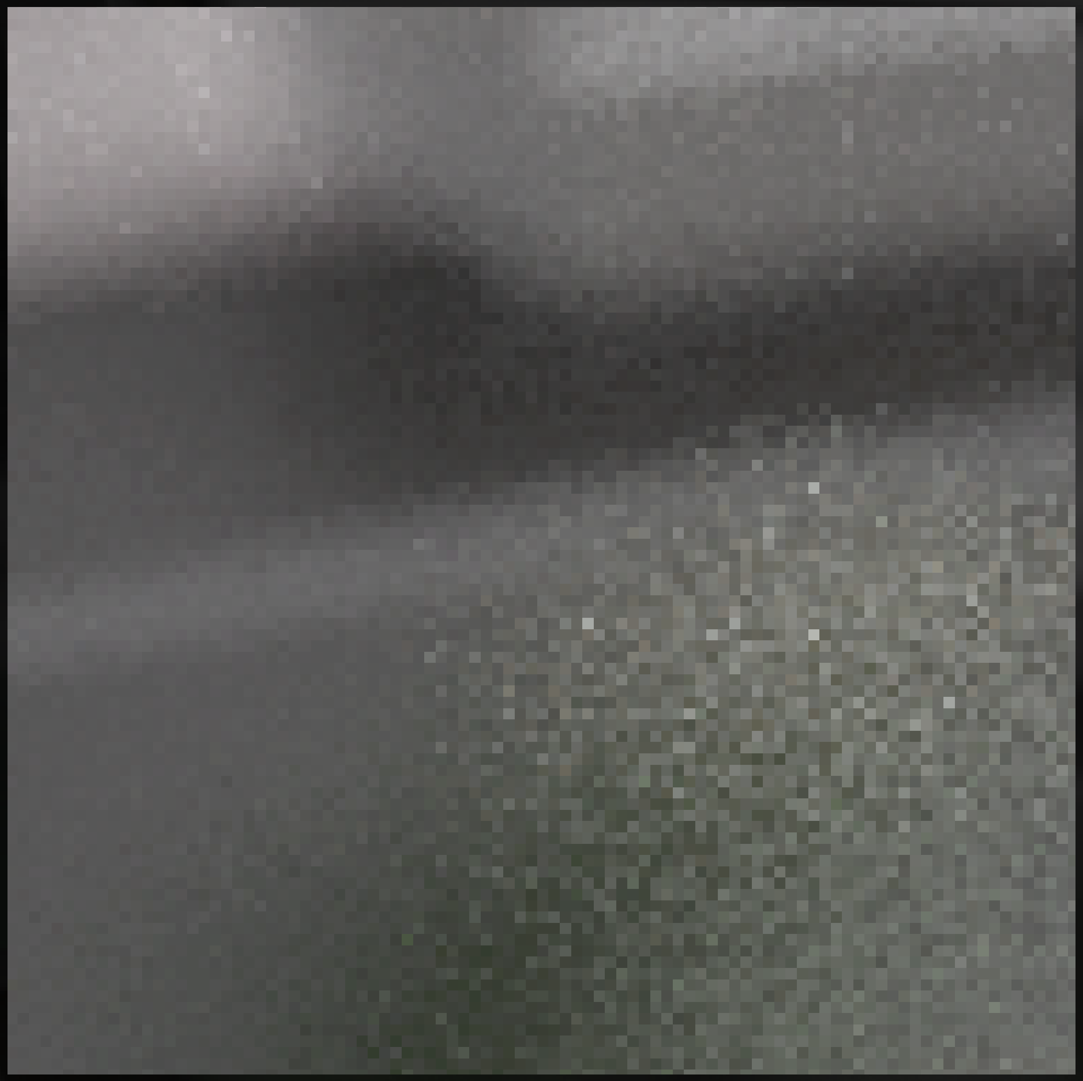
Here’s another scene from PBRT, to which we added defocus and moving chairs.

Our result: 16 spp + reconstruction at 128spp
771 sec (PBRT) + 10 sec (reconstruction)

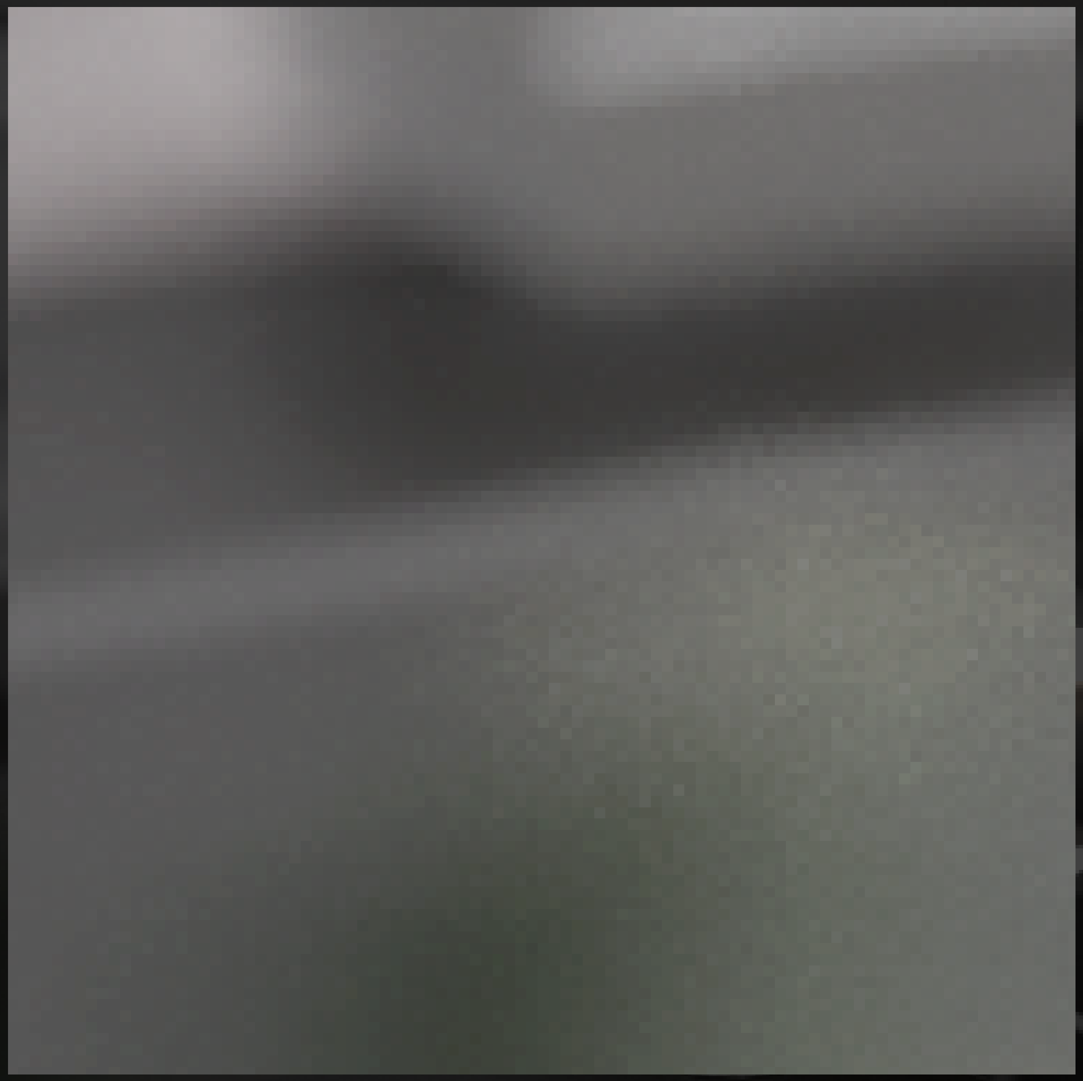


And our results.

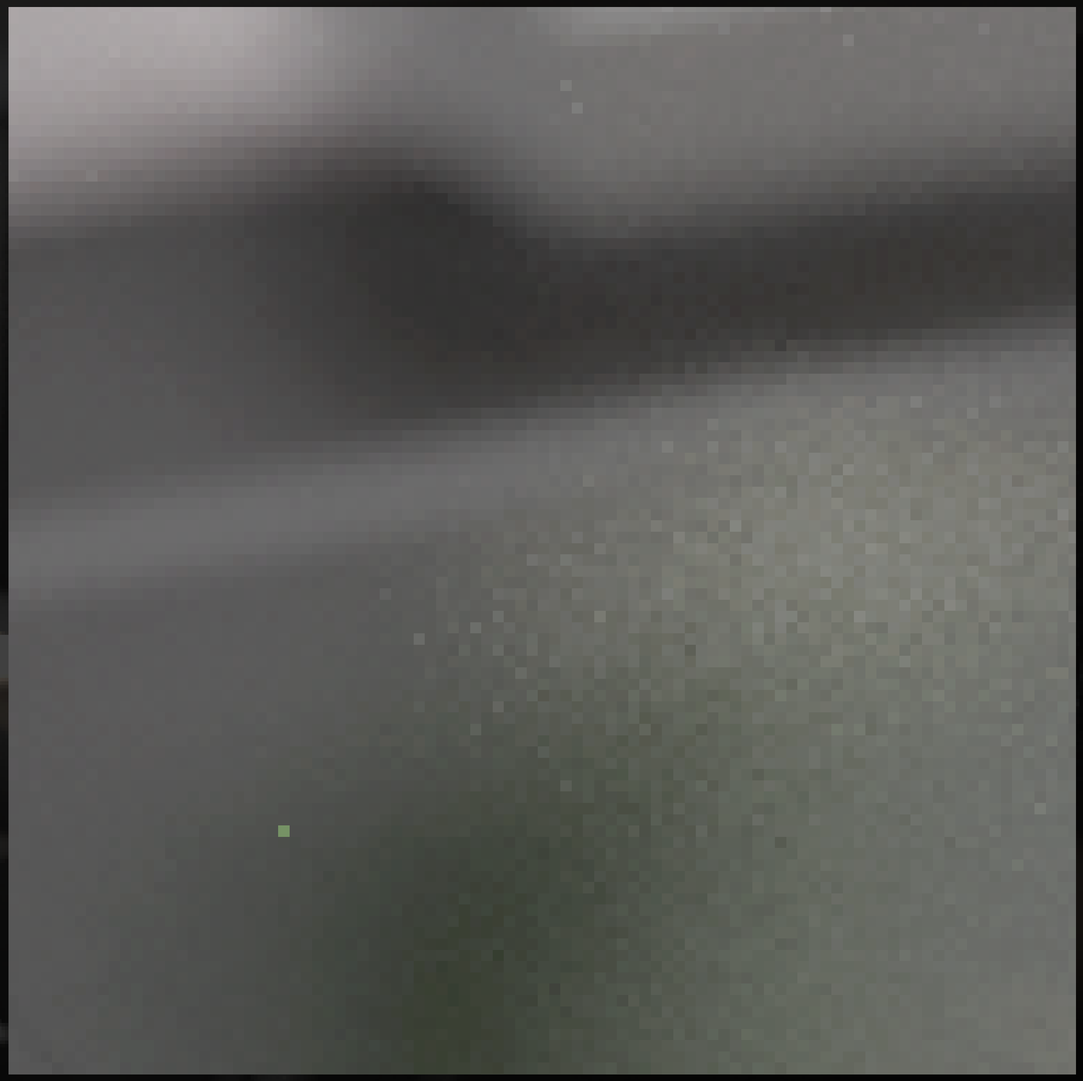
Our result: 16 spp + reconstruction at 128spp
771 sec (PBRT) + 10 sec (reconstruction)



Input: 16 spp



Our result at 128 spp
using same input

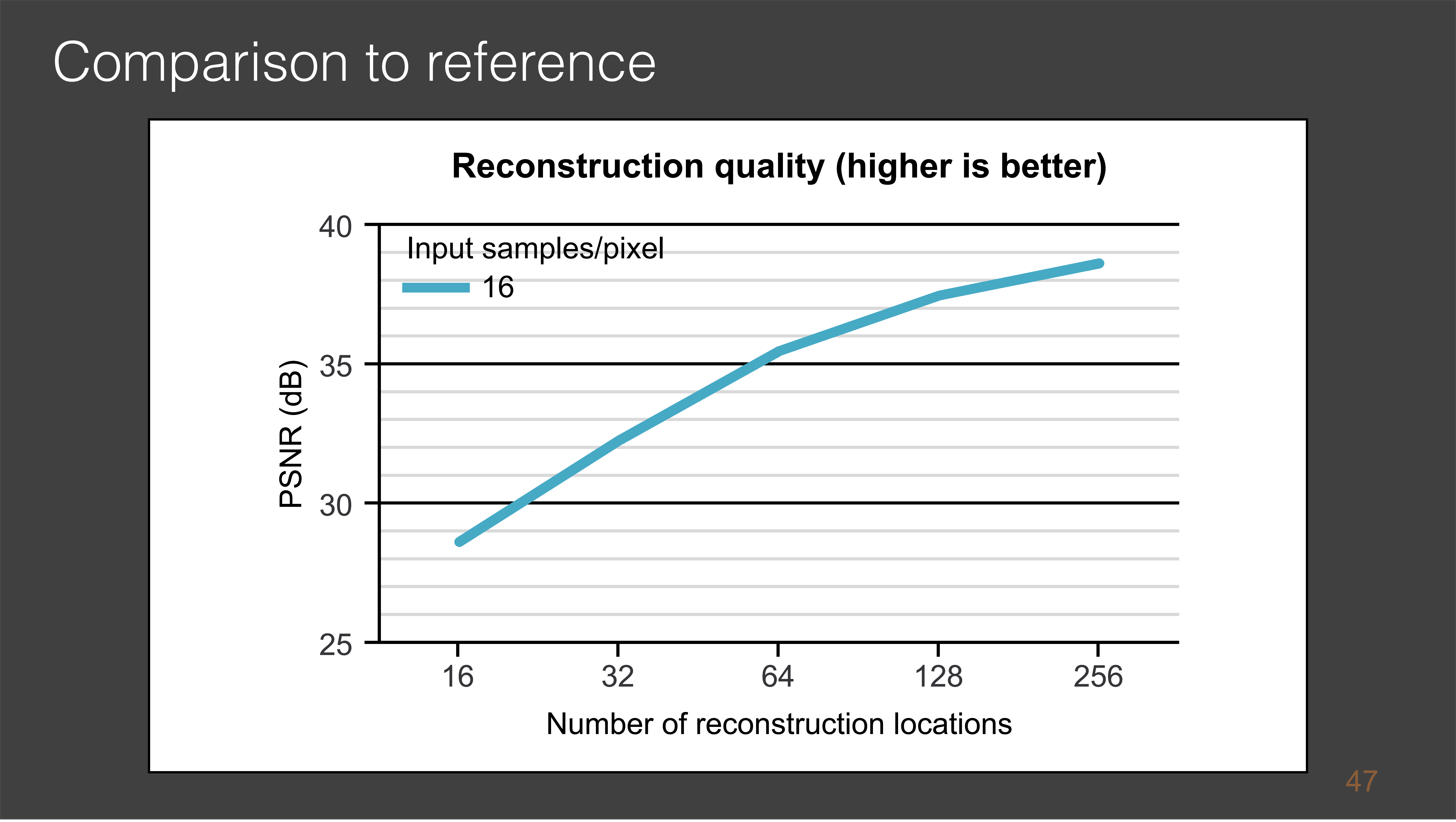


Reference: 256 spp
(16x time)

Looking at a closeup of the moving chairs, at 16 samples per pixel, the moving chair is quite noisy

If you look carefully, you might notice that our 128 samples per pixel reconstruction is actually a bit smoother than the 256 samples per pixel reference image.

This is due to the fact that each of our reconstruction samples can gather from a larger number of input samples than brute force does.



Comparison to reference

We also quantitatively evaluated the quality of our reconstruction against a reference image rendered at 1024 samples per pixel.

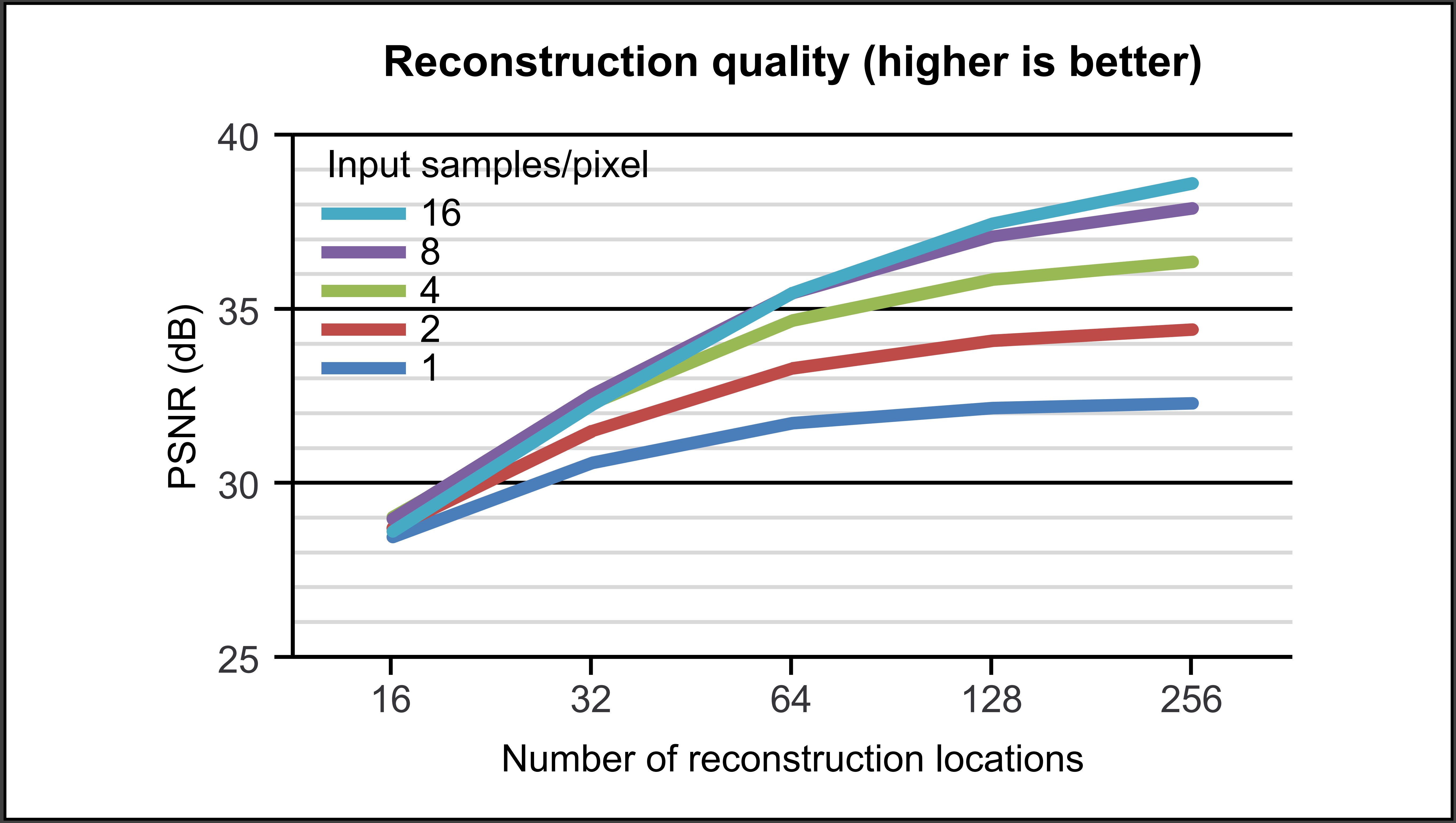
This graph plots our peak signal to noise ratio with higher PSNR being better.
Starting from 16 samples per pixel as input, the quality of our reconstruction gets better as we increase the number of reconstruction locations.

We can also vary the number of input samples <animate>
Each curve in the plot here plots a different number of input samples and plots the PSNR as we increase the number of reconstruction locations.

Using 16 samples per pixel appears to be more than sufficient to achieve a PSNR in the high 30 dBs, which is indistinguishable from the reference.

And even starting from 1 sample per pixel, we can do better than 30 dBs.

Comparison to reference



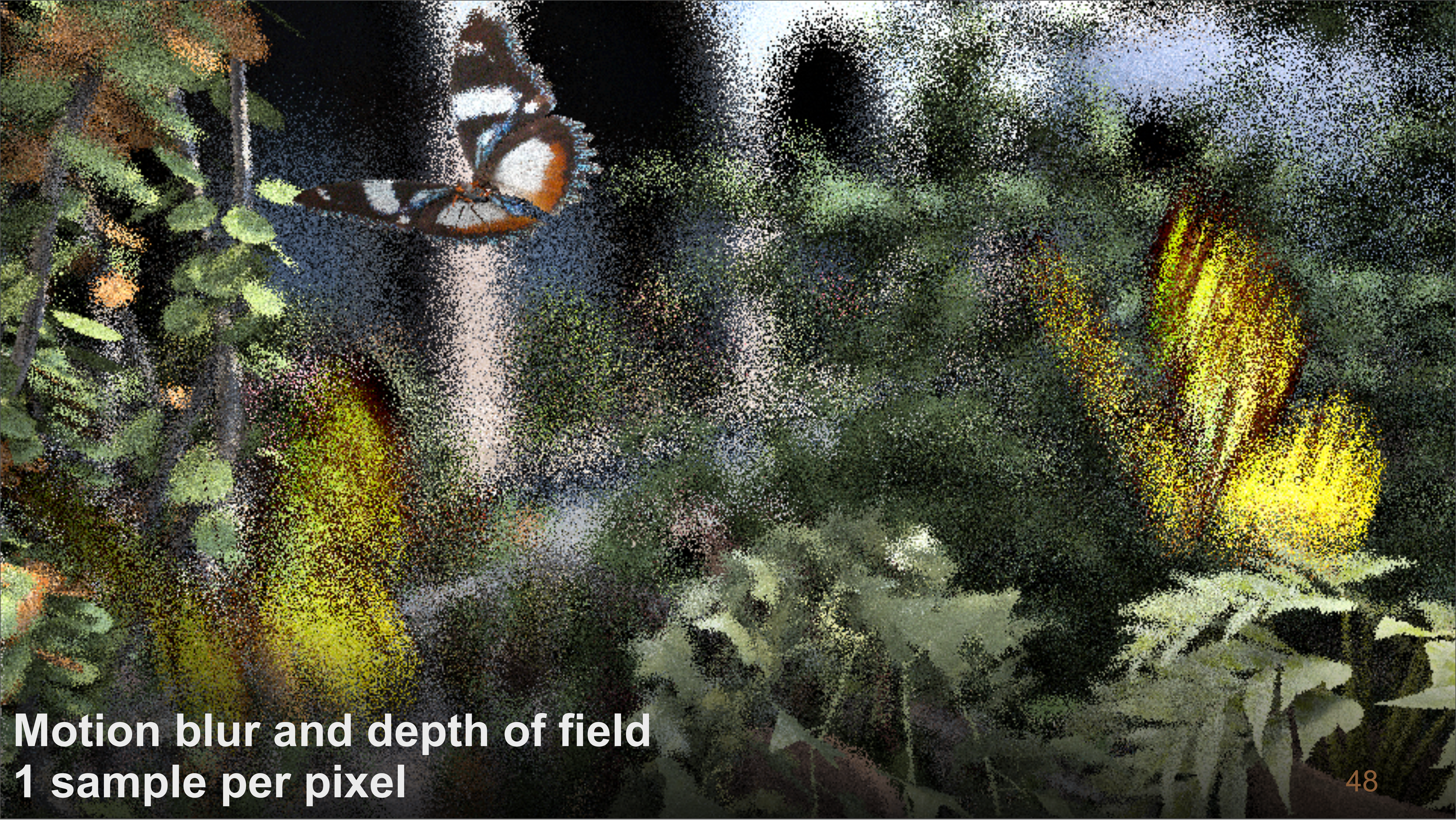
We also quantitatively evaluated the quality of our reconstruction against a reference image rendered at 1024 samples per pixel.

This graph plots our peak signal to noise ratio with higher PSNR being better.
Starting from 16 samples per pixel as input, the quality of our reconstruction gets better as we increase the number of reconstruction locations.

We can also vary the number of input samples <animate>
Each curve in the plot here plots a different number of input samples and plots the PSNR as we increase the number of reconstruction locations.

Using 16 samples per pixel appears to be more than sufficient to achieve a PSNR in the high 30 dBs, which is indistinguishable from the reference.

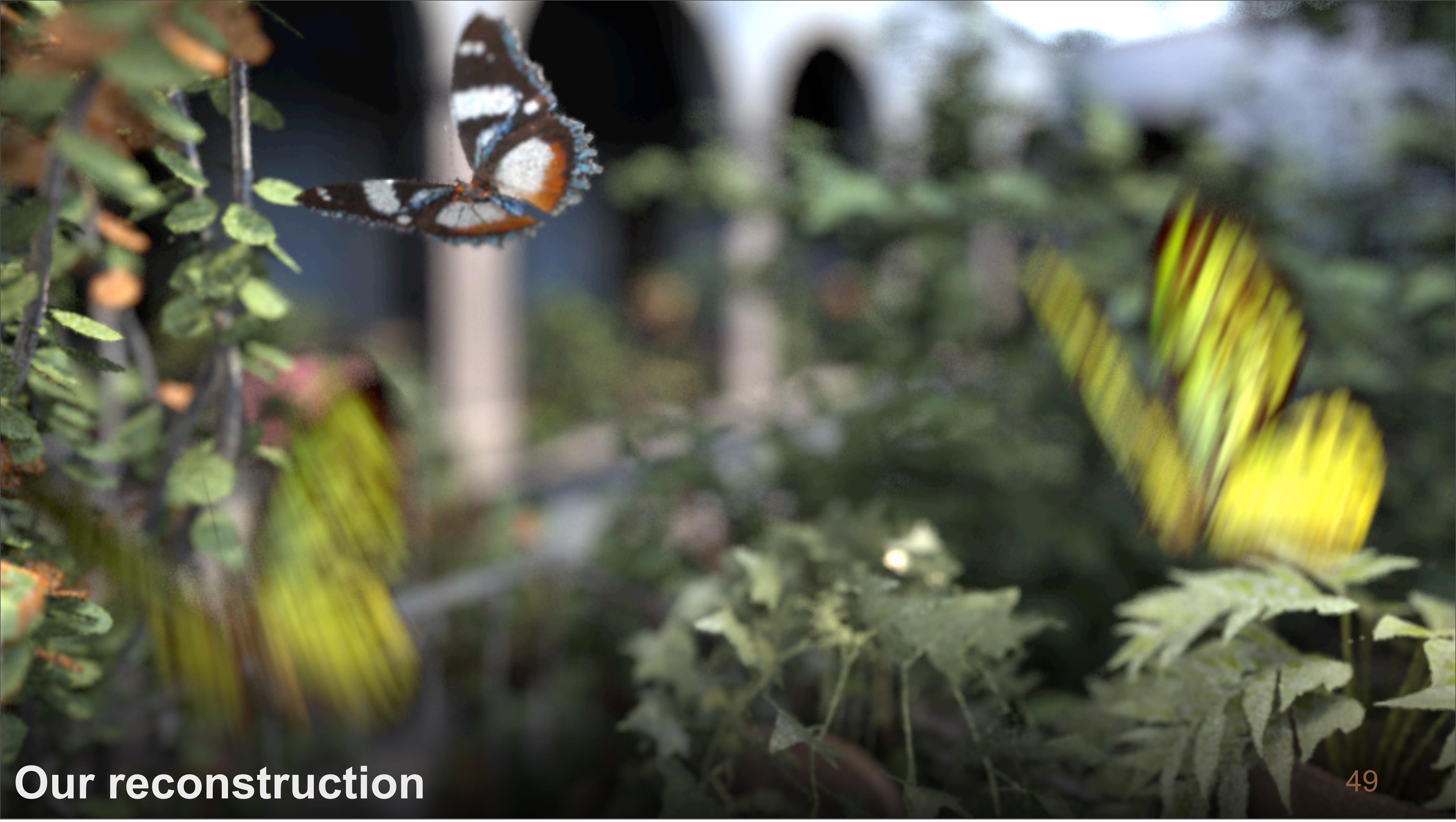
And even starting from 1 sample per pixel, we can do better than 30 dBs.



Motion blur and depth of field
1 sample per pixel

To show you an extreme case, 1 sample per pixel using box reconstruction is really noisy.

Using the exact same samples as in this image,

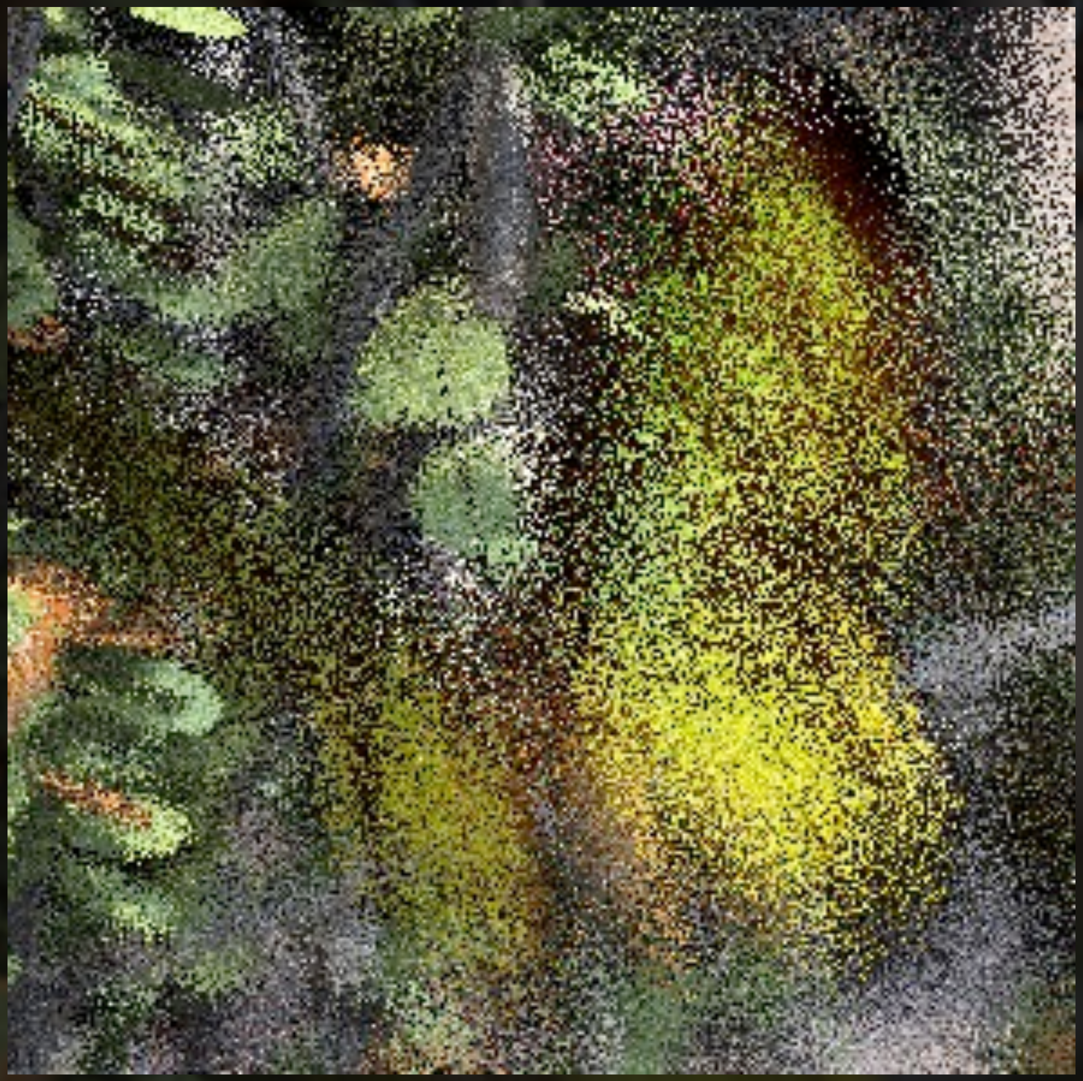


Our reconstruction

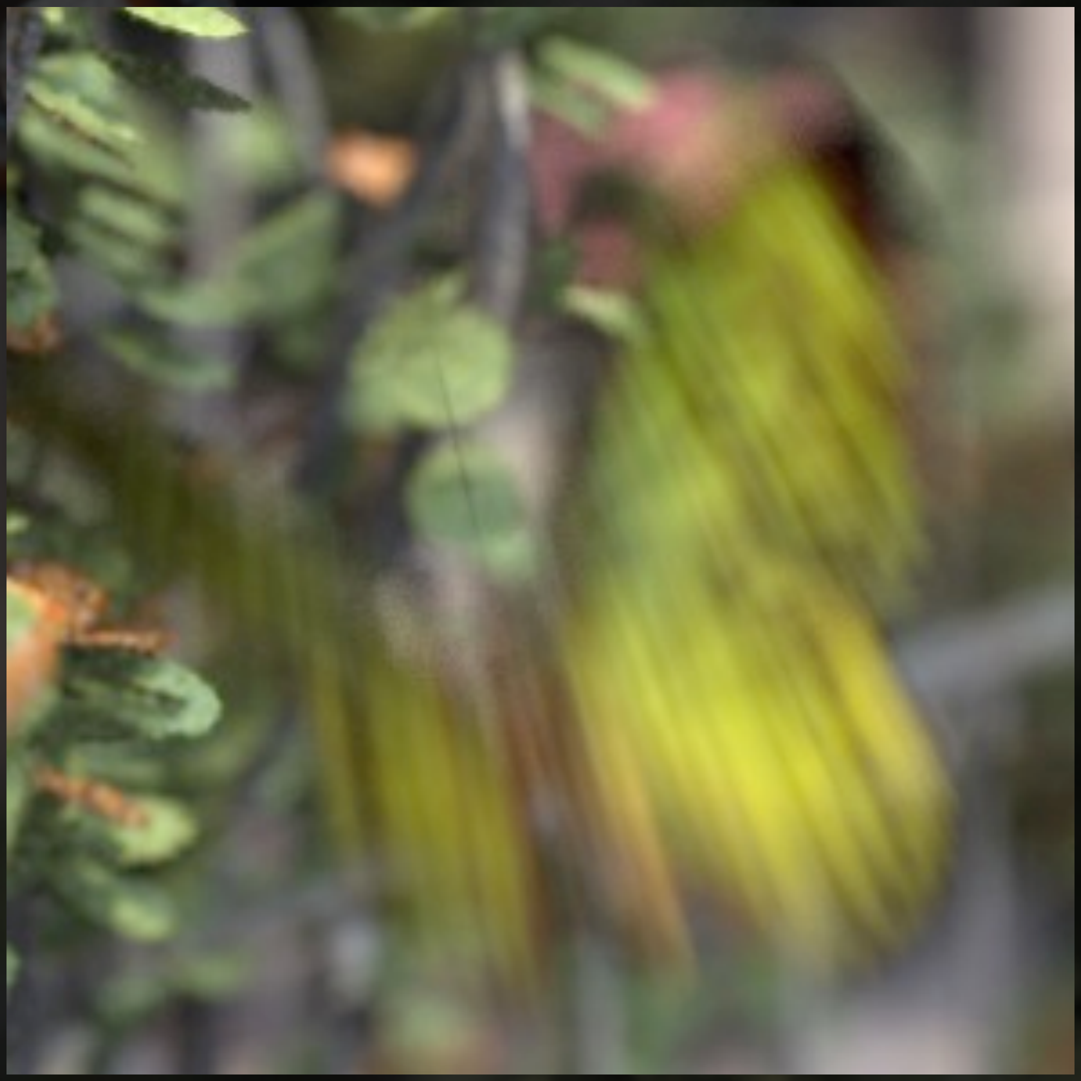
our algorithm can produce this high quality result.

Here’s a closeup.

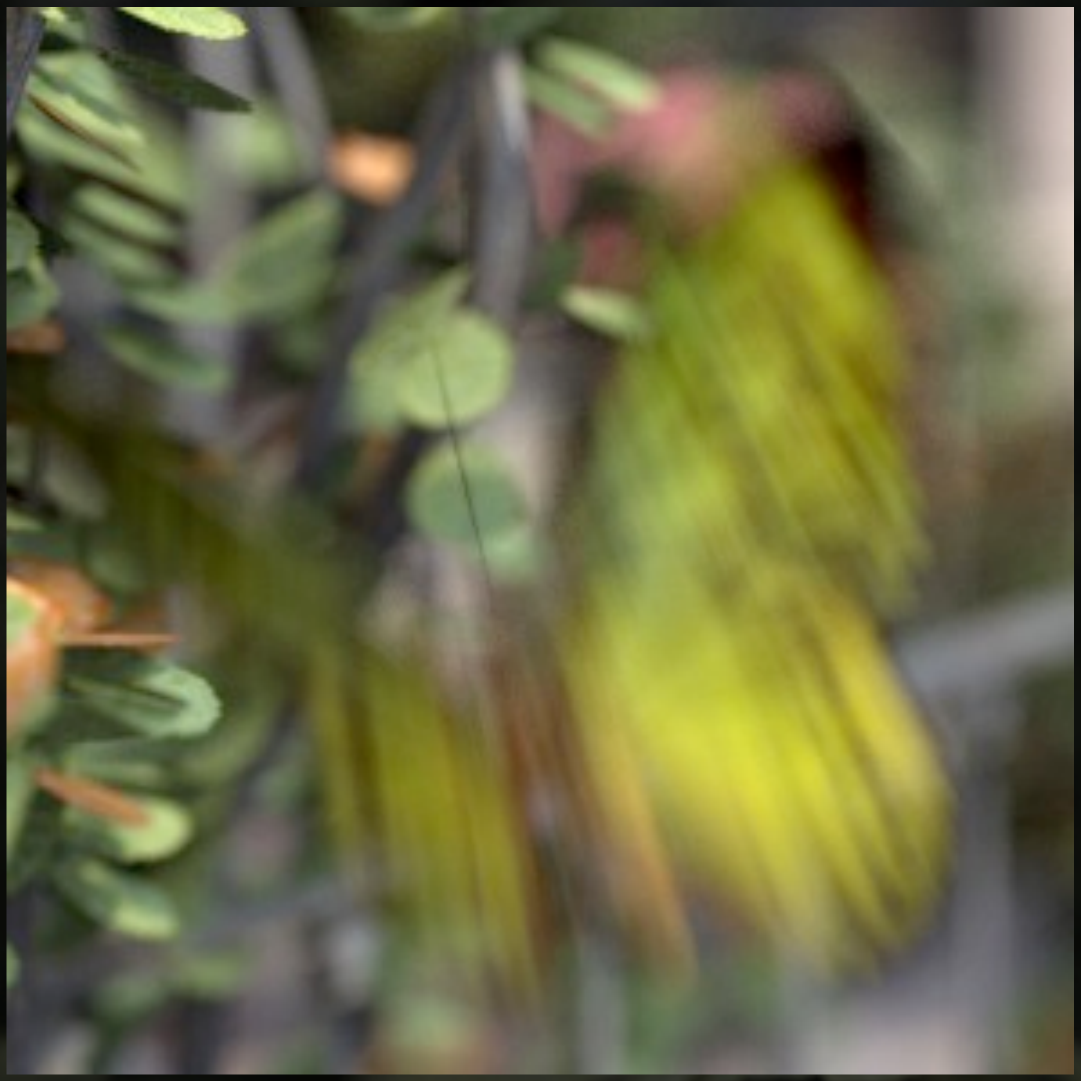
Our reconstruction



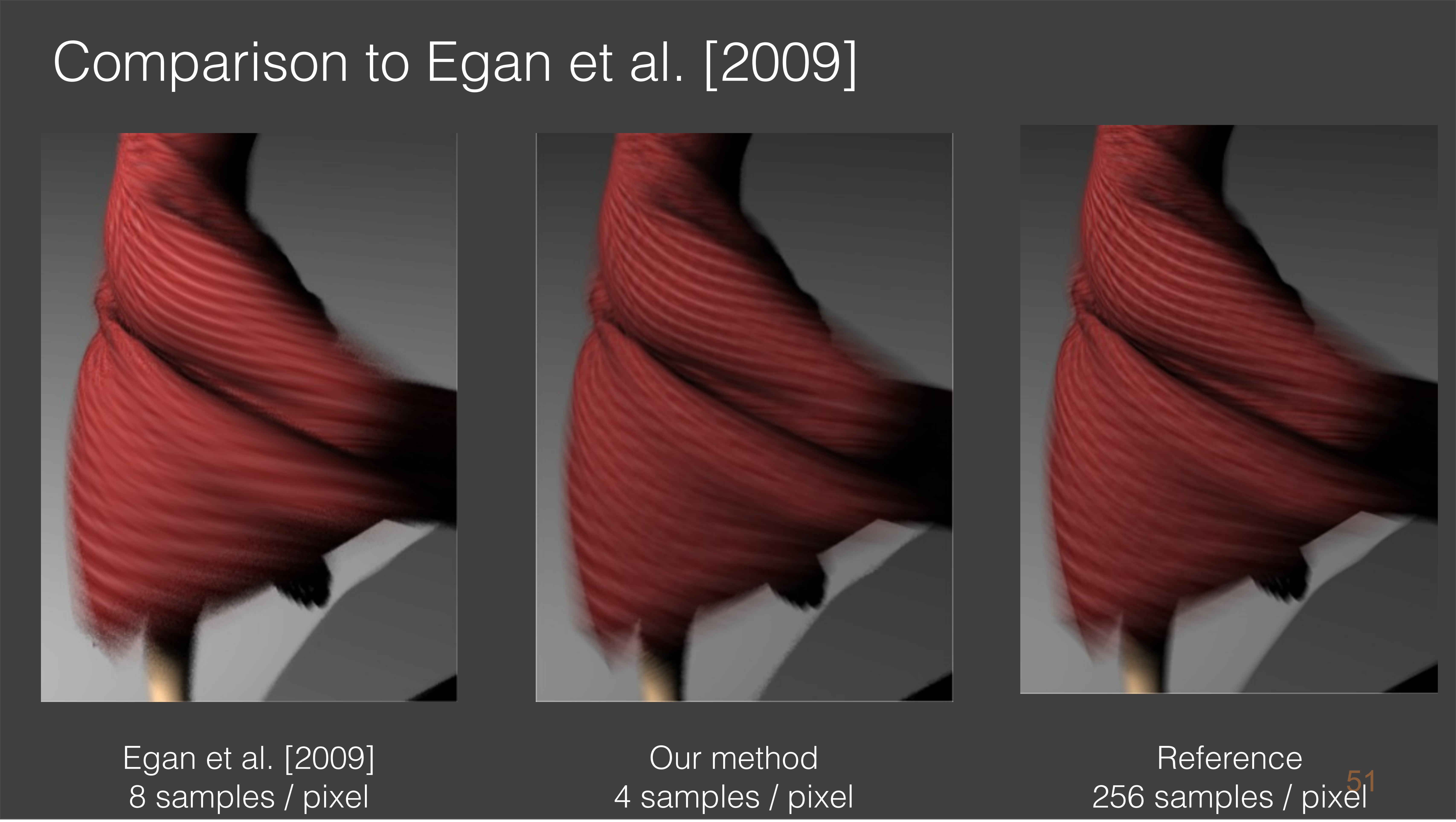
Input: 1 spp



Our result: 1 spp -> 128 spp



Reference 256 spp
(256x time)



Comparison to Egan et al. [2009]

Egan et al. [2009]
8 samples / pixel

Our method
4 samples / pixel

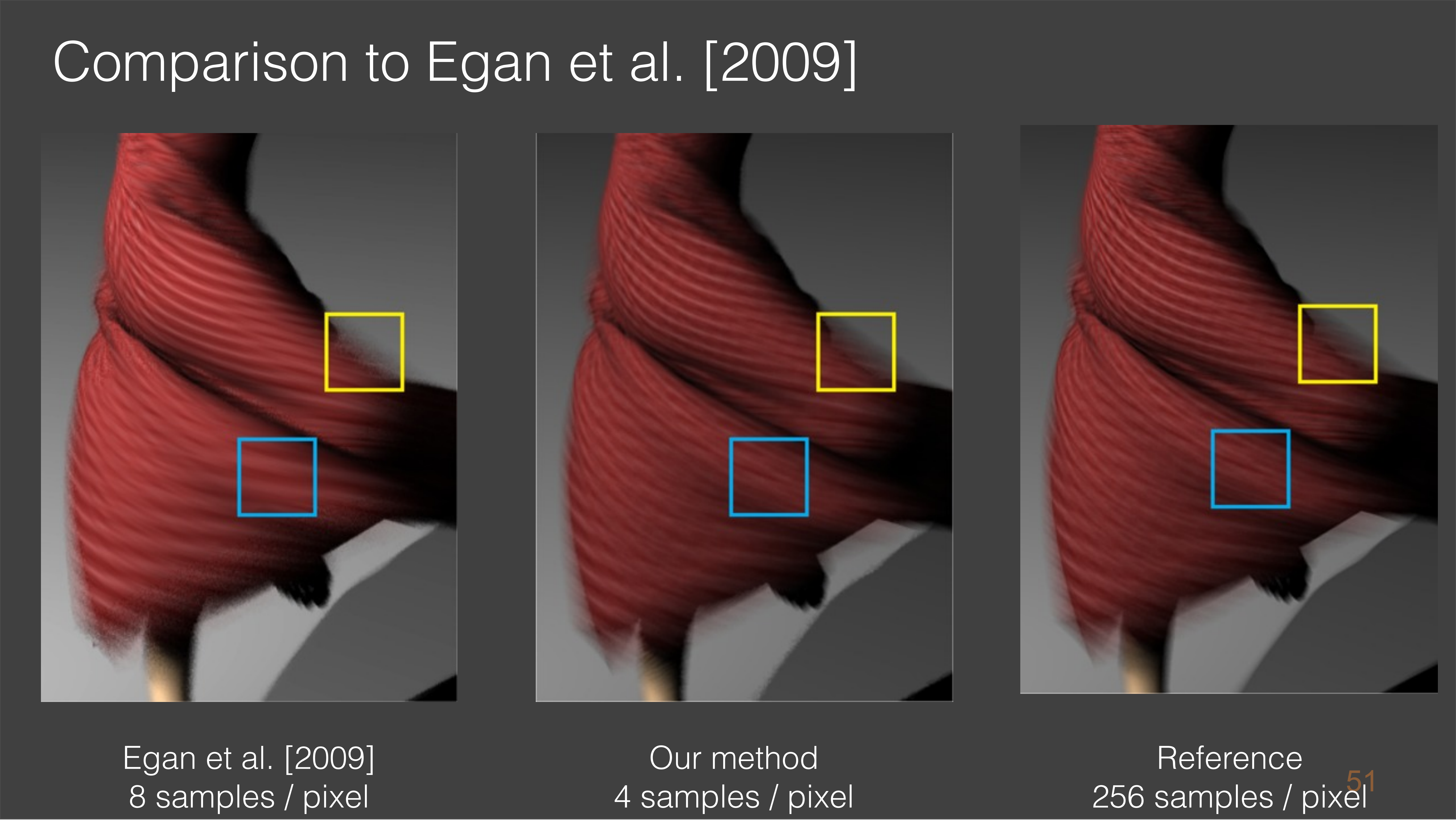
Reference
256 samples / pixel

In addition to a high quality reference, we also compared our method to the sheared reconstruction method by Egan and colleagues for the case of motion blur.

<animate>

As I pointed out earlier, in regions of complex motion and visibility, particularly at boundaries,
their method resorts to brute force sampling, which results in some noise, whereas our reconstruction remains smooth.

Here’s a closeup.



Comparison to Egan et al. [2009]

Egan et al. [2009]
8 samples / pixel

Our method
4 samples / pixel

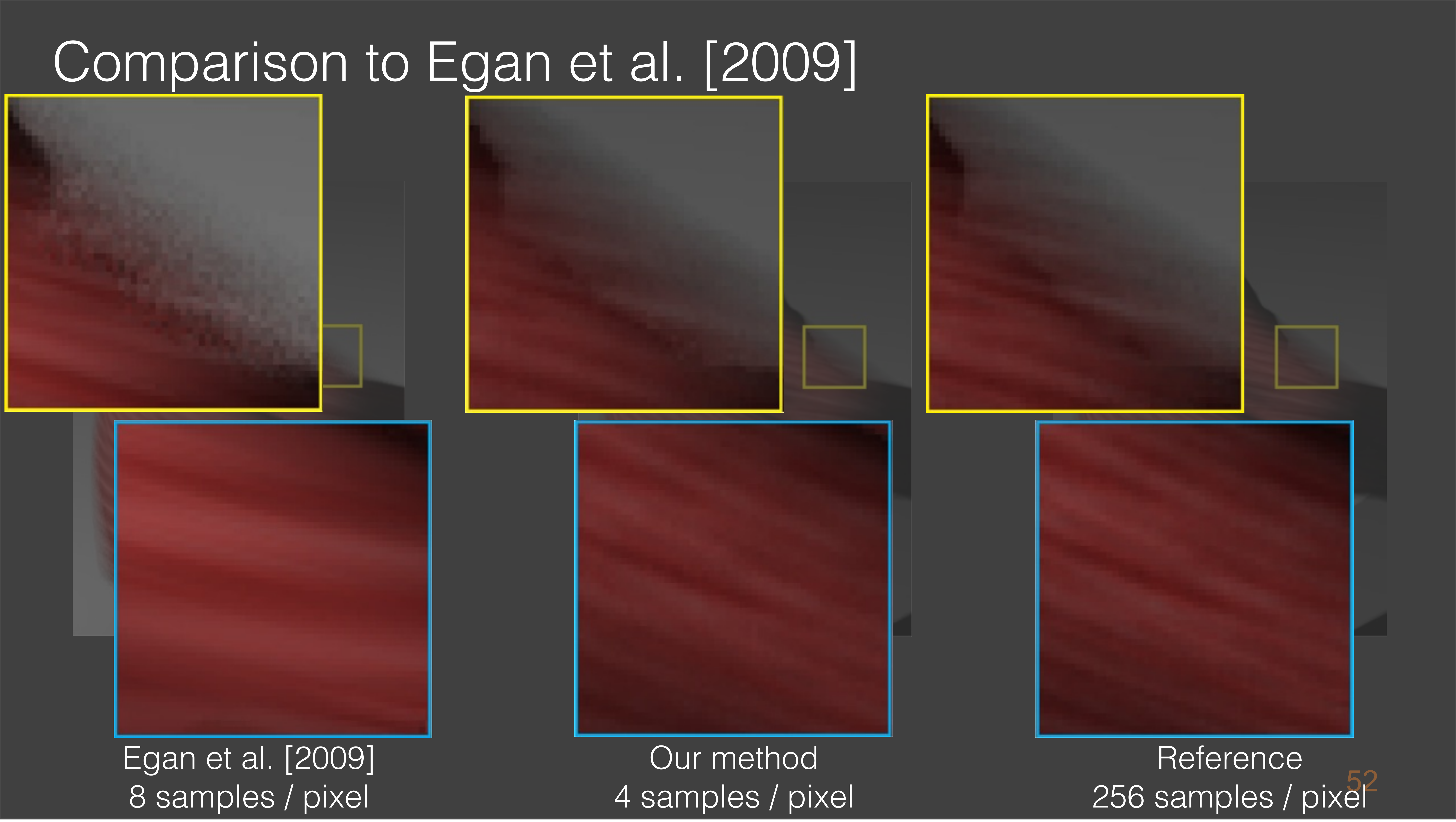
Reference
256 samples / pixel

In addition to a high quality reference, we also compared our method to the sheared reconstruction method by Egan and colleagues for the case of motion blur.

<animate>

As I pointed out earlier, in regions of complex motion and visibility, particularly at boundaries,
their method resorts to brute force sampling, which results in some noise, whereas our reconstruction remains smooth.

Here’s a closeup.



Comparison to Egan et al. [2009]

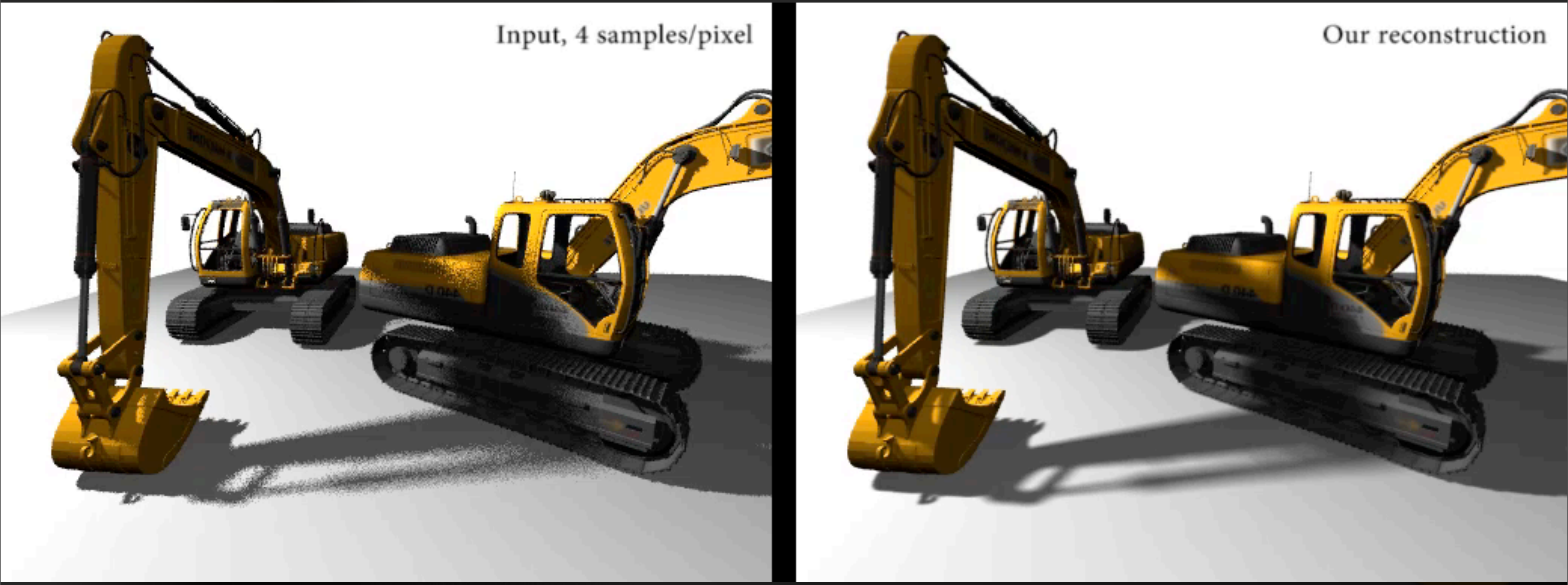
Egan et al. [2009]
8 samples / pixel

Our method
4 samples / pixel

Reference
256 samples / pixel

Notice the noise near the boundary of the dress with the background, and how our method better reconstructs the texture of the dress.

Soft shadows, 4 spp

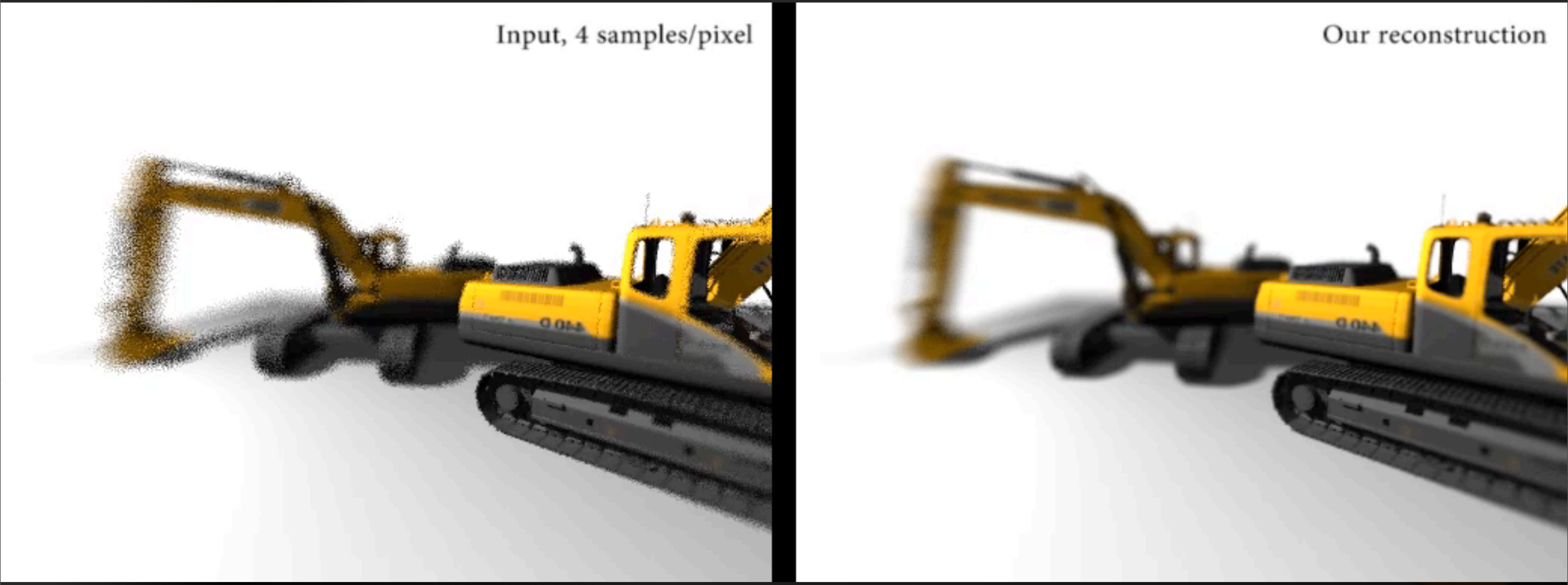


To demonstrate that our reconstruction is stable under animation, we rendered some videos.

This scene has soft shadows only.
The input with 4 samples of the area light per pixel is on the left.
Notice the noise in all the shadows, especially on the ground.

Our reconstruction is on the right, with no noise.

7D soft shadows with motion and defocus, 4 spp



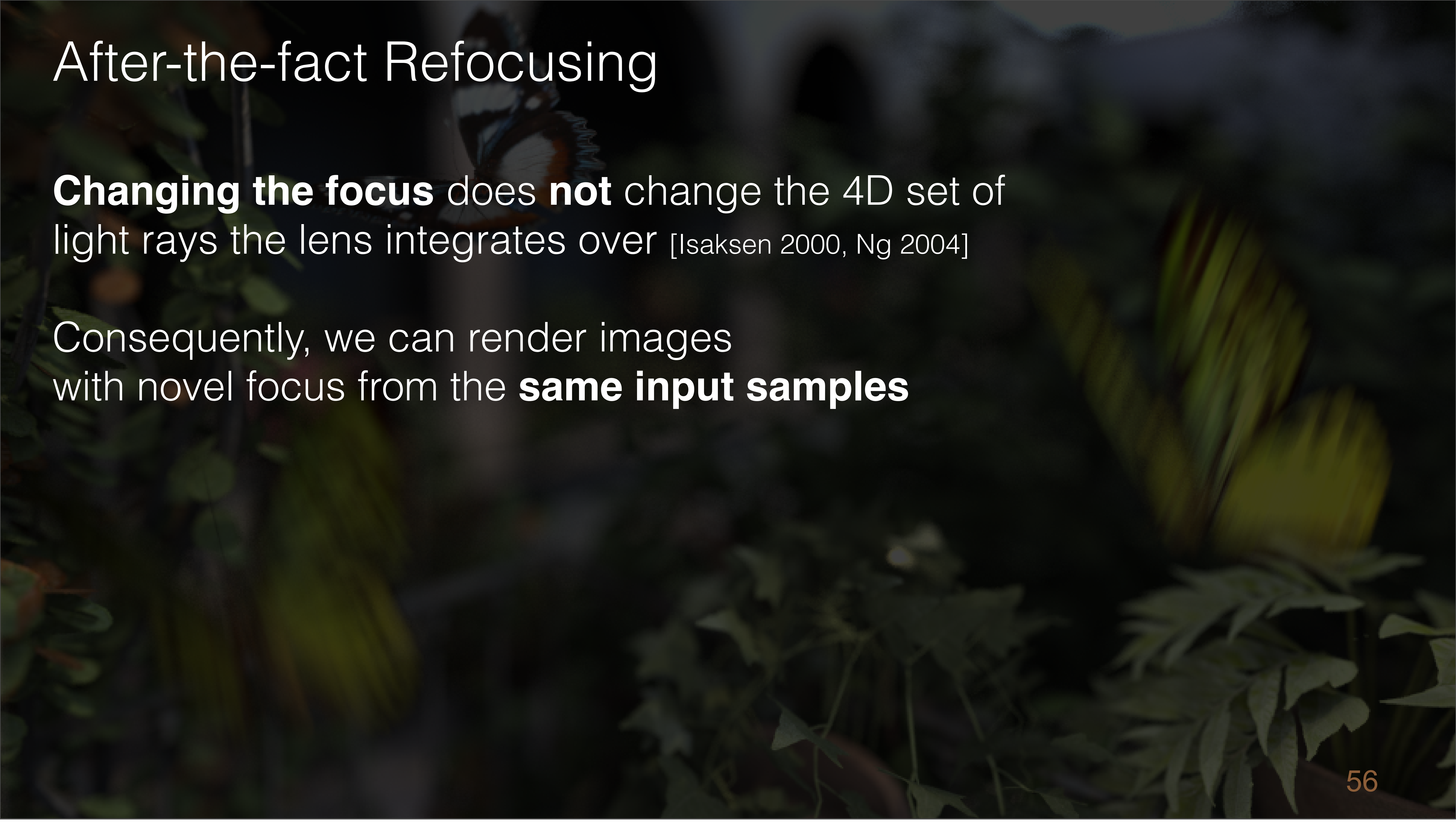
And here’s a video of our 7D reconstruction with simultaneous motion blur, defocus, and soft shadows,
all starting from 4 samples per pixel.

The scene is illuminated by an area light.
The excavator in the back is out of focus and its arm is motion blurred.
Notice how the arm becomes sharp when it rotates through the focal plane and the motion happens to be orthogonal to the image plane.

With 4 samples per pixel, the input is quite noisy.
And our 7D reconstruction is on the right, with smooth blur and shadows.

Goodies

Finally, as a bonus, we have one other feature.



After-the-fact Refocusing

Changing the focus does **not** change the 4D set of light rays the lens integrates over [Isaksen 2000, Ng 2004]

Consequently, we can render images with novel focus from the **same input samples**

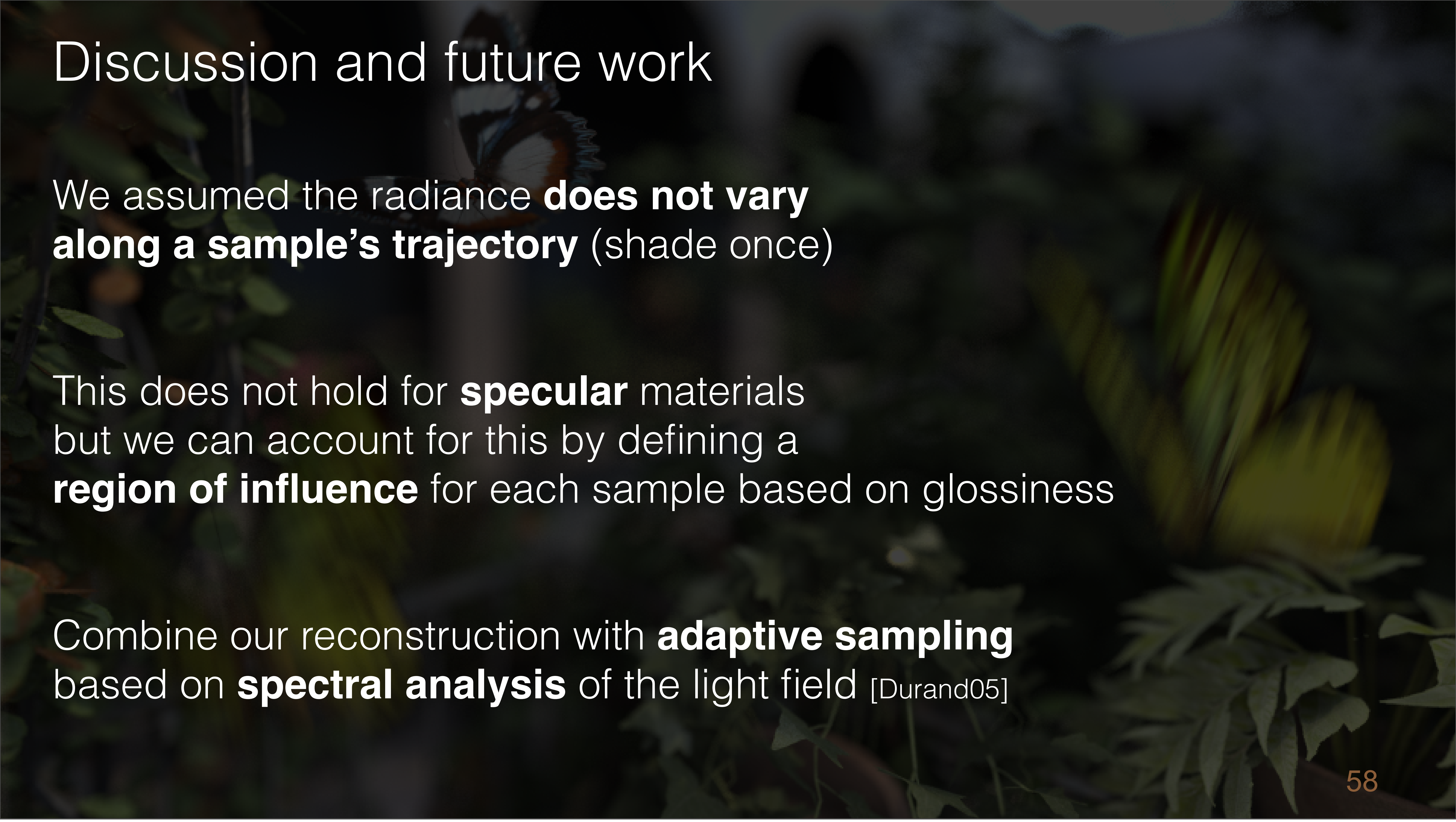
It’s well known that changing a camera’s focal plane does *not* change the light field inside the camera.

It’s simply integrating over a different slice.

Our reconstruction trivially lets us refocus after sampling.

In this sequence, we pull the focus from the background all the way to the foreground foliage.

Notice that our reconstruction is stable despite the fact that every frame in this animation was computed independently from the *exact* same set of input samples



Discussion and future work

We assumed the radiance **does not vary along a sample’s trajectory** (shade once)

This does not hold for **specular** materials
but we can account for this by defining a
region of influence for each sample based on glossiness

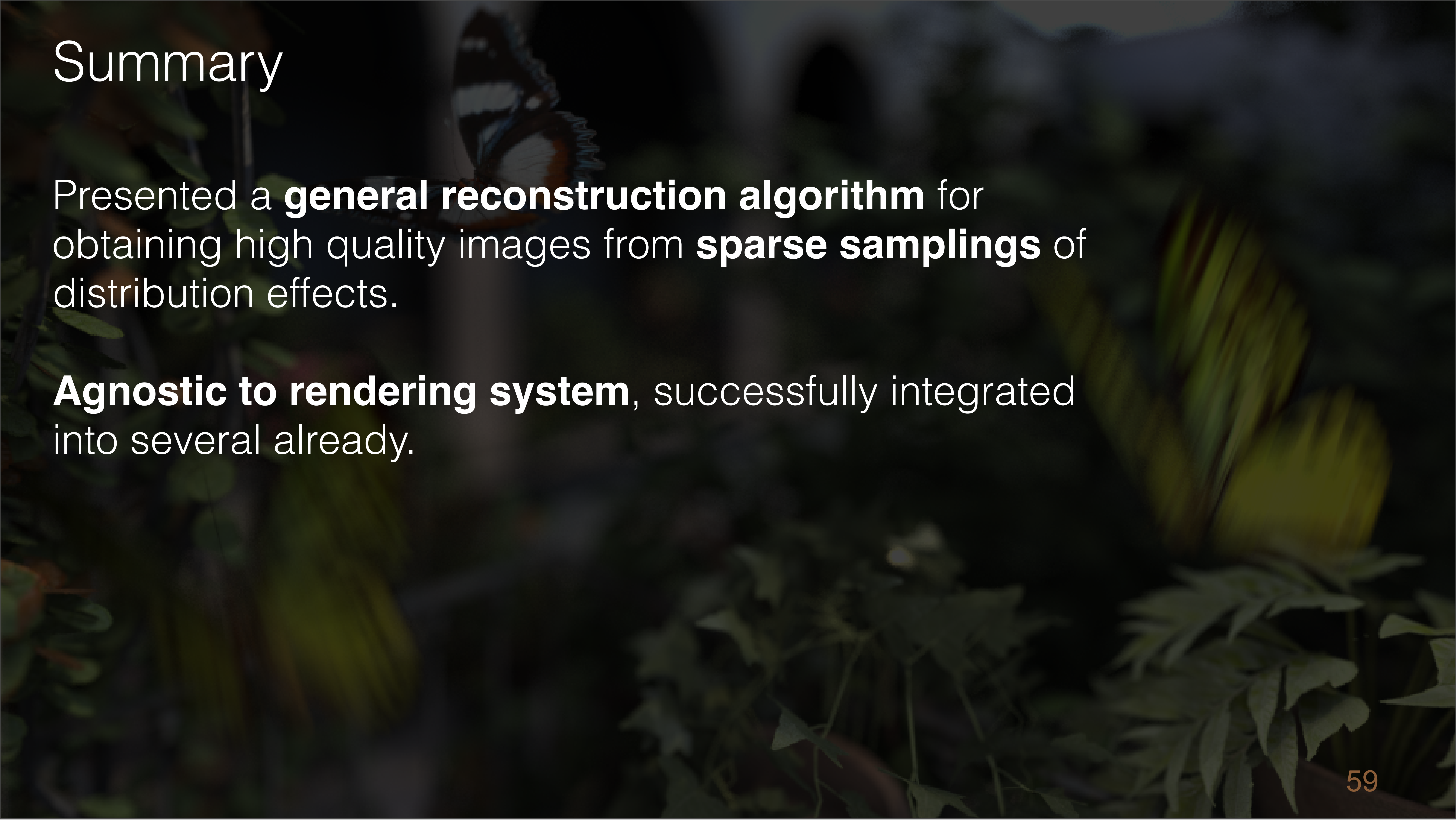
Combine our reconstruction with **adaptive sampling**
based on **spectral analysis** of the light field [Durand05]

Our system does have a few limitations, which I will discuss here.

One fundamental assumption that we made was that radiance does not vary across a sample’s trajectory.
This is the classic shade-once assumption, which does not hold for glossy or specular objects.

The right way to handle this, which we want to tackle in future work,
is to adjust the region of influence of input samples based on their glossiness.

To make our sampling more efficient, we may want to integrate our reconstruction algorithm
into an adaptive sampling system that accounts for the spectral information in the light field.



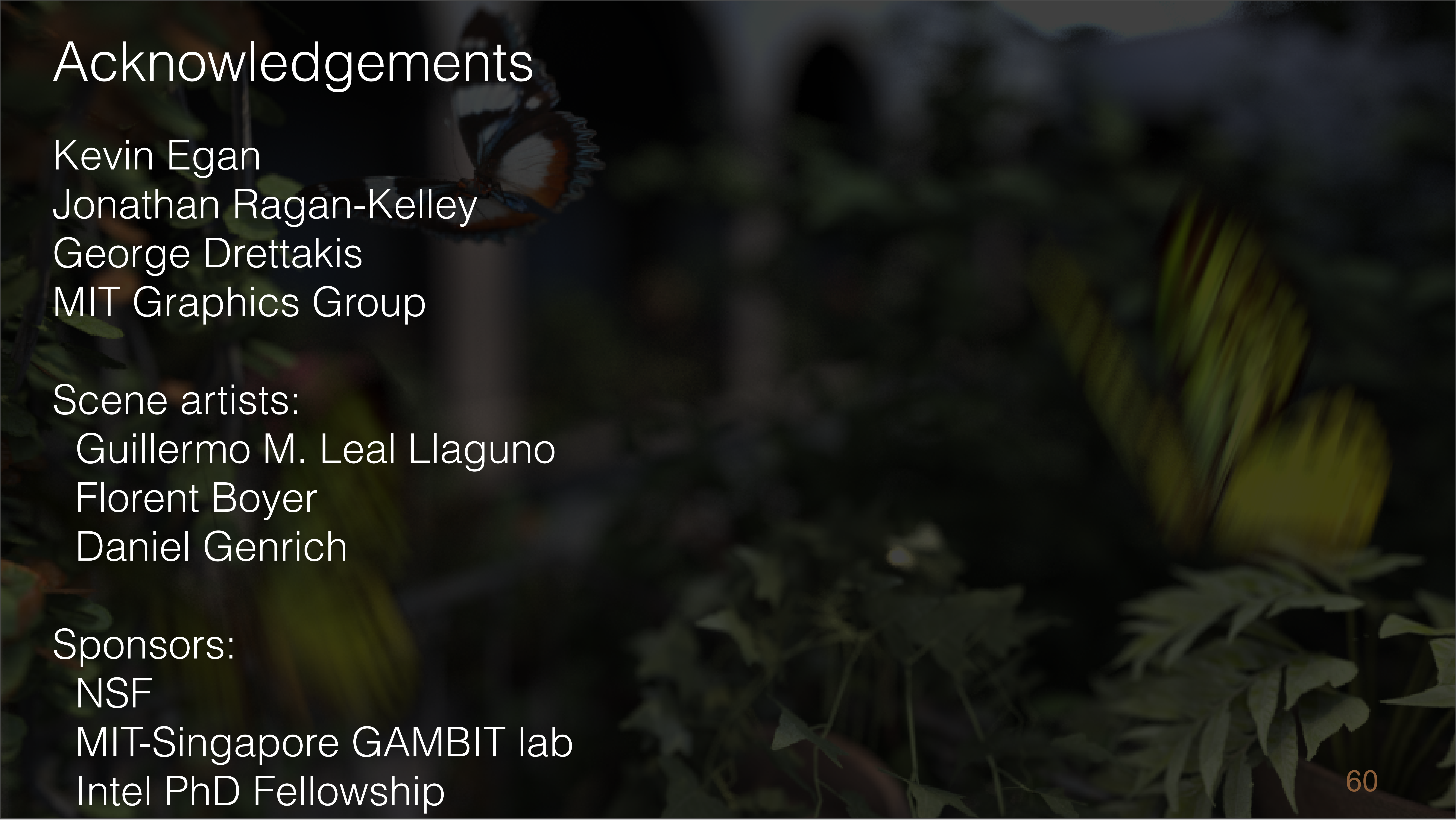
Summary

Presented a **general reconstruction algorithm** for obtaining high quality images from **sparse samplings** of distribution effects.

Agnostic to rendering system, successfully integrated into several already.

To summarize, we’ve presented a general reconstruction algorithm
for obtaining high quality images from sparse stochastic samplings of distribution effects.

Our approach is agnostic to the underlying rendering system
and has been successfully integrated with a raytracer, stochastic rasterizer, and renderman system.



Acknowledgements

Kevin Egan
Jonathan Ragan-Kelley
George Drettakis
MIT Graphics Group

Scene artists:
- Guillermo M. Leal Llaguno
- Florent Boyer
- Daniel Genrich

Sponsors:
- NSF
- MIT-Singapore GAMBIT lab
- Intel PhD Fellowship

I’d like to acknowledge the following individuals, the scene artists, and our sponsors for all their help.

Thank you for your attention and I’ll take any questions.