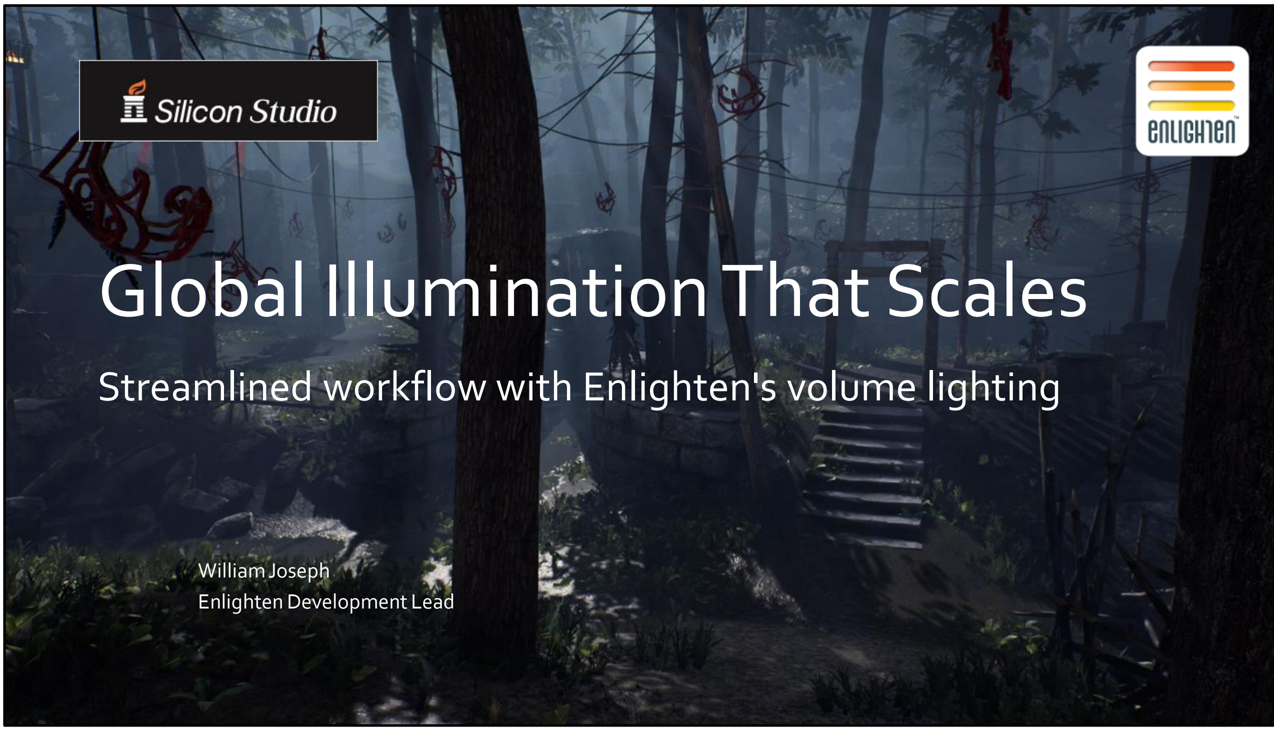


Hello and welcome.
Thanks for coming to this session today.
My name is Will, and I lead the Enlighten development team at Silicon Studio, in Tokyo, Japan
I'm here today to talk about Enlighten, and I'll be focussing on our solution for volume lighting.

Silicon Studio

Agenda

- A little background
- Brief overview of Enlighten
- Challenges associated with lightmaps and probes
- Deep dive: Enlighten's volume lighting solution
- Future of Enlighten at Silicon Studio

2

Today I'll begin with a little background to provide some context for this talk.
I'll then give a brief overview of Enlighten, for anyone not familiar with it…
… and talk about some of the challenges faced by any GI solution based on lightmaps and probes.
I'll do a deep dive into the design and implementation of Enlighten's volume lighting solution.
Finally I'll share some of our future plans for Enlighten at Silicon Studio.

Silicon Studio

A little background

- Enlighten is middleware that provides global illumination
- Used in many AAA game titles
- Originally developed by Geomerics
- Silicon Studio took over development

3

First, the context:
Enlighten is middleware that provides global illumination, used in many triple-A game titles.
I previously worked on Enlighten at Geomerics, until last year, when Silicon Studio took over the rights to develop, license and support Enlighten

Silicon Studio

Hellblade: Senua's Sacrifice

- Developed by Ninja Theory
- Built with UE4 + Enlighten
- Seamless streamed world
- Enlighten volume lighting

4

One recent game using Enlighten is Hellblade: Senua's Sacrifice, developed by Ninja Theory.
They built the game in Unreal Engine 4 with Enlighten.
Enlighten's volume lighting played a small but key part.

Hellblade’s lighting is very atmospheric

… always changing…

.. and makes great use of shadows and bounced light.

The weather changes dynamically to reflect Senua’s state of mind.
This scene starts with an atmosphere of extreme fear, and then the lighting changes…

.. to give a feeling of calm optimism.

Silicon Studio

Brief Overview of Enlighten

What does Enlighten provide?

This shot shows a scene rendered with Enlighten global illumination.
When Enlighten is turned off...

... We see only the direct lighting.
The areas which are not directly lit appear completely black.

With Enlighten, the shadows are filled by reflected sunlight from the cliff.

Silicon Studio

Enlighten computations

- Enlighten computes only reflected (indirect) lighting
- Not the direct lighting or shadows
- Enlighten computations run asynchronously on the CPU

Enlighten computes only the reflected light, known as **indirect lighting**.
The direct lighting and shadows are provided by the engine.
Enlighten computations run asynchronously on the CPU, leaving your GPU free for other rendering tasks.

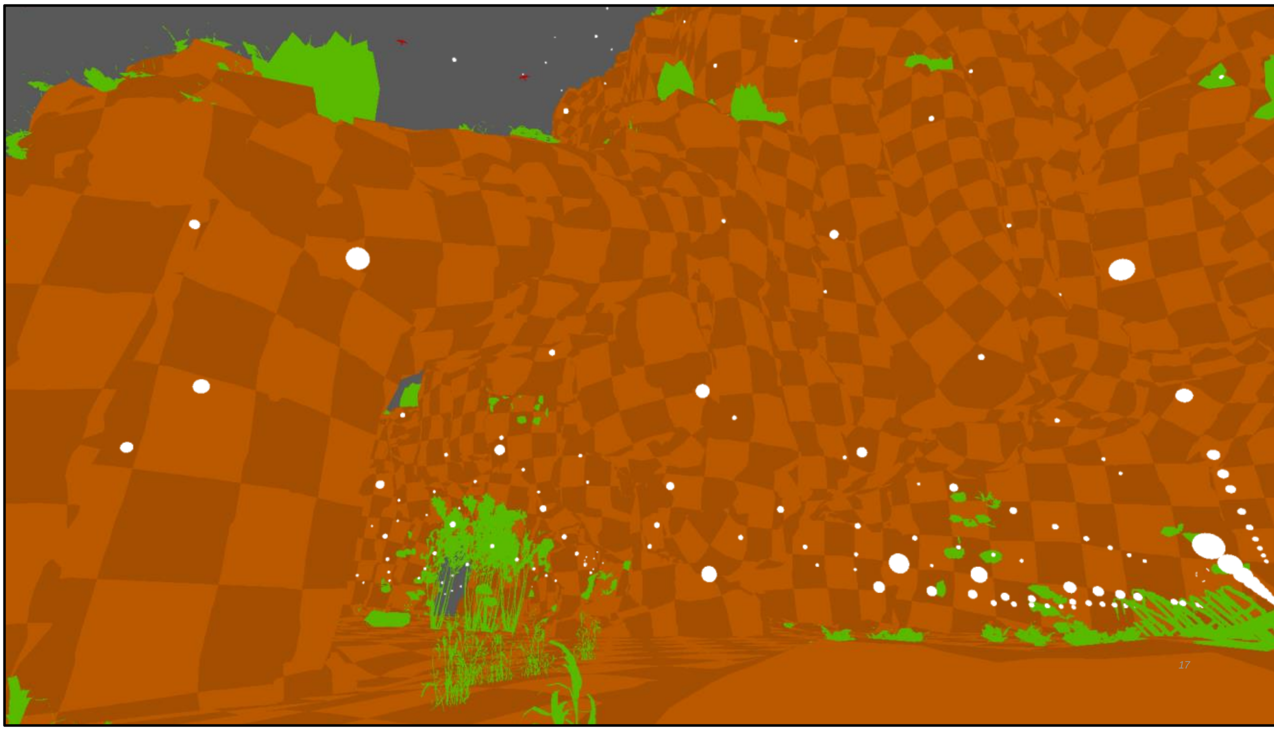
Silicon Studio

Enlighten outputs

- High quality GI with real-time updates in game or editor
- Writes diffuse indirect lighting to **lightmaps** and **probes**
- Also **cubemaps** for use in reflections
- Much lower resolution than traditional baked lighting

Enlighten provides real-time global illumination updates in game, or in your world editor.
Enlighten outputs diffuse indirect lighting to **lightmaps** and **probes**; also **cubemaps** for use in specular reflections, but that's not the focus of this talk.
Enlighten's lightmaps and probes do not include direct lighting and shadows, so we use a much lower resolution than traditional baked lighting.

In this example the area is in shadow, so the lighting is computed by Enlighten.

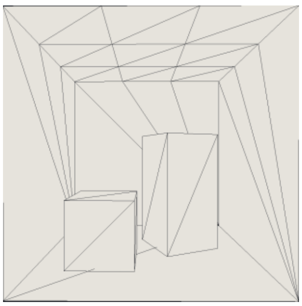


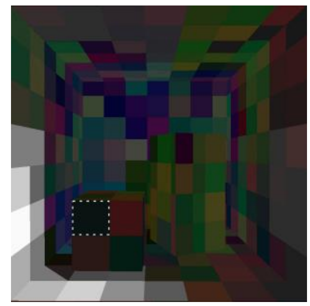
The meshes lit with lightmaps are shown in orange, and the squares show the size of each lightmap pixel.
The meshes lit with probes are shown in green, and the probes are shown in white.
Even this low resolution…

.. is sufficient to capture gradually varying indirect light reflected from large surfaces

Enlighten precomputation

- Divide surfaces into patches, compute form factors

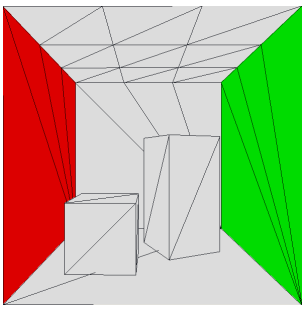


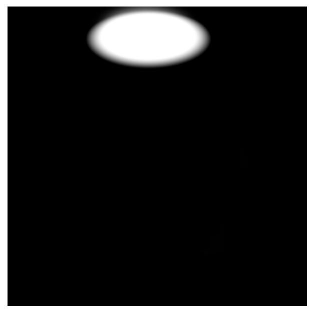


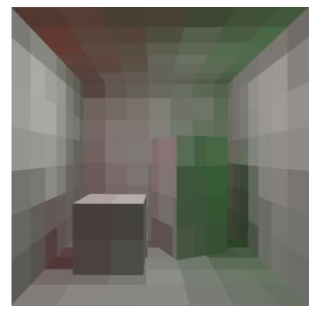
Enlighten uses the **radiosity** method to compute indirect lighting.
We first divide mesh surfaces into patches, and then compute visibility form factors between all patches.
We do this part of the work in an offline precomputation step.

Enlighten runtime update

- Compute lighting in real-time: can change lighting in game



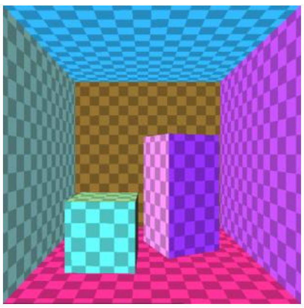
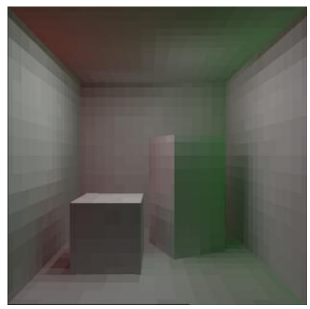
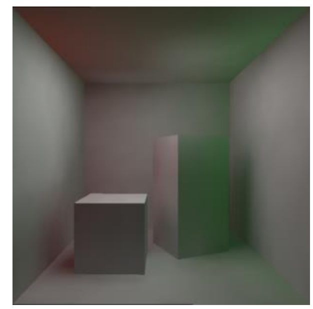




Enlighten **doesn’t** compute the lighting in advance.
Based on the light sources provided at run-time, we compute the illumination of each patch after multiple light bounces.
This part of the computation runs in real-time, so you can instantly see changes to the lighting.

Output for rendering

- Use form factors to update lightmaps in real time

We also precompute form factors to patches for each lightmap pixel.
We use these form factors to update the illumination at runtime for each pixel.
When rendering, we sample indirect lighting from the lightmap.

We use the same method to precompute form factors and compute illumination for each **probe**.

Silicon Studio

ENLIGHTEN™

Precomputation trade-off

- Precomputation includes only static meshes
- Moving meshes are excluded: they don’t reflect light
- Excluded meshes still have great lighting:
  - indirect lighting from probes
  - direct lighting and shadows
  - specular reflections and ambient occlusion

Only static meshes can be included in the precomputation, so moving meshes are excluded.
Excluded meshes don’t reflect light, but still get high quality indirect lighting from probes.
Both included and excluded meshes are affected as normal by direct lighting and shadows, specular reflections and ambient occlusion.

Silicon Studio

Summary: Enlighten overview

- Provides indirect lighting part of GI
- Real-time updates in game and world editor
- Lightmaps and probes have minimal GPU overhead
- Precomputation: only static meshes reflect light

In summary:
Enlighten provides real-time indirect lighting updates in game and in your world editor.
Enlighten uses lightmaps and probes to enable efficient rendering with minimal GPU overhead.
Lastly, only static meshes can be included in the precomputation and reflect light.

Lightmaps and Probes

So now let's talk about lightmaps and probes.

Silicon Studio

Efficient rendering

- Lightmaps and probes are widely used
- Inexpensive lighting in forward shading pass
- Each technique has its own challenges

These are an integral part of any precomputed GI solution, for good reasons. They make it simple and cheap to sample indirect lighting in a forward rendering pass.
I'm going to talk about some of the challenges commonly associated with each technique.

ENLIGHTEN

Lightmaps

- 2D representation of lighting across a surface
- Applied to surfaces using UV mapping
- High fidelity lighting for large static meshes
- Challenge: good lighting with minimal lightmap size

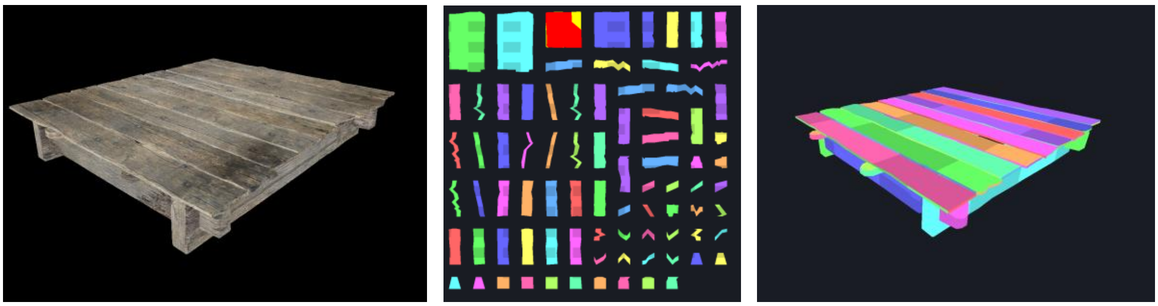
Lightmaps are a 2D representation of lighting, applied to surfaces using UV mapping. They can provide high fidelity lighting for large static surfaces, such as building walls, or a terrain heightmap.

To keep the cost reasonable, we need to produce a UV unwrap which minimizes the number of pixels in the lightmap, while also providing accurate lighting.

Silicon Studio

UV unwrapping

- Doing this by hand is difficult and time consuming



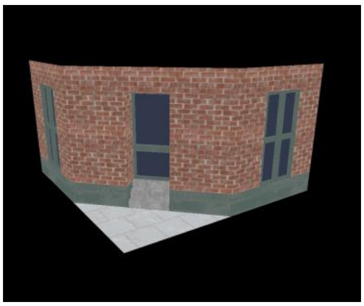
27

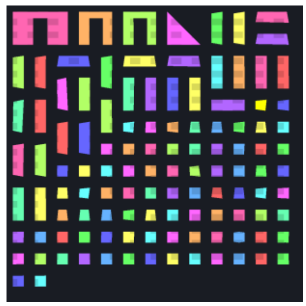
Here's an example of the lightmap UV layout for a typical mesh.
On the right, the mesh is colored to show how surfaces are mapped to lightmap UV space.
For a moderately complex mesh, it can be a challenge to produce an efficient lightmap UV mapping,
.. and doing this by hand is difficult and time consuming for even the most practiced artist.

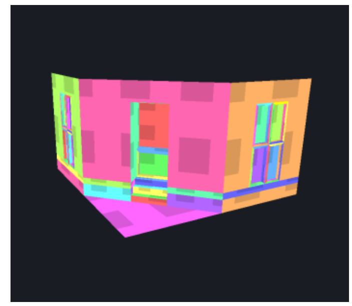
Silicon Studio

Wasteful unwrap

- 1 m^2 resolution: lightmap with 728 pixels







This building mesh has a lighting resolution of one pixel per square meter.
This means that its lightmap UVs are laid out so that each pixel covers at least one square meter in world space.
Because the small parts of the mesh are all unwrapped separately, the resulting lightmap uses more than 700 pixels.

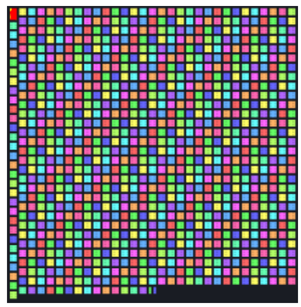
Automatic Simplification

- 1 m^2 resolution: lightmap with 256 pixels

Enlighten automatically produces a much more efficient unwrap, for the same lighting resolution.
It can automatically simplify the UV layout to use fewer lightmap pixels.
This version of the lightmap uses around 250 pixels.

Complex organic meshes

- Lightmaps not a good solution: 15,624 pixels



Creating good lightmap UVs is even more difficult with a very complex organic mesh like this tree.
Automatic simplification can't do much in this scenario.
For the same lighting resolution, the lightmap has more than 15,000 pixels.
Lightmaps just aren't a good solution for this type of mesh.

Silicon Studio

Static surfaces only!

- Lightmaps provide lighting only for static surfaces
- Can't use lightmaps in many situations:
  - moving meshes
  - particles
  - volumetric fog

Lightmaps have some other fundamental limitations.
We can't use lightmaps to light meshes that move or deform at run-time.
.. or for lighting volumetric effects like fog.
For this..

Probes

- 3D representation of diffuse lighting across a volume
- Sample from nearby probes to get lighting at a point
- Lighting for moving meshes and volumetric effects
- Alternative to lightmaps for any mesh – no lightmap UVs!

.. we turn to **Probes**.
We place probes in the world to provide diffuse lighting across a 3D volume of space.
To get the lighting at any point in the volume we can sample from nearby probes.
Probes are typically used to provide lighting for moving meshes and volumetric effects.
We can also use probes to light any mesh that could be lit using a lightmap,
…which saves effort because we don't need to unwrap lightmap UVs.

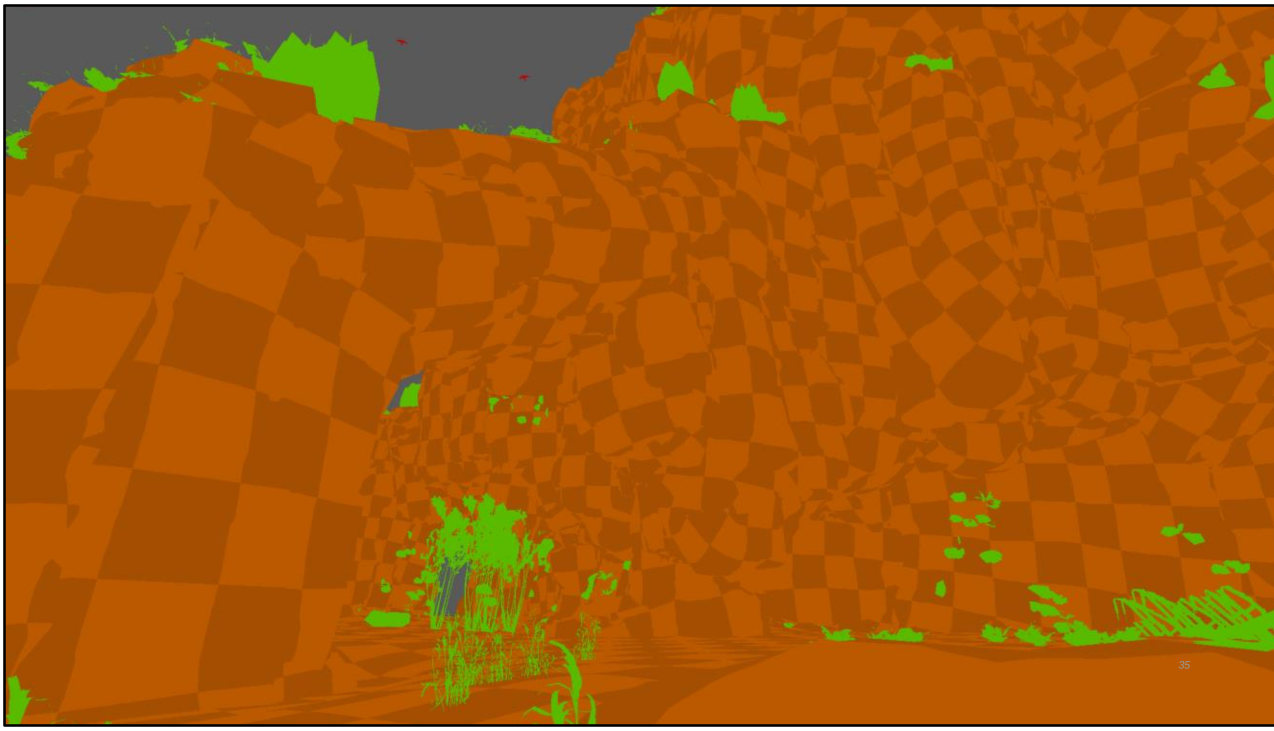
Silicon Studio

Per-mesh sampling

- Sample probes only once for each mesh
- Simple to implement
- Minimal shading cost

To compute the lighting, a common approach is to sample from nearby probes once per mesh.
This is simple to implement and inexpensive to render.

Here’s the same scene as before.



The meshes shown in green are each lit with a single sample from nearby probes.
Each piece of foliage is a separate mesh.

Even with only a single sample per mesh, the indirect lighting for the smaller meshes matches the surrounding areas.

In the final image, the lighting for the smaller meshes looks great.

Silicon Studio

Probe lighting for large meshes

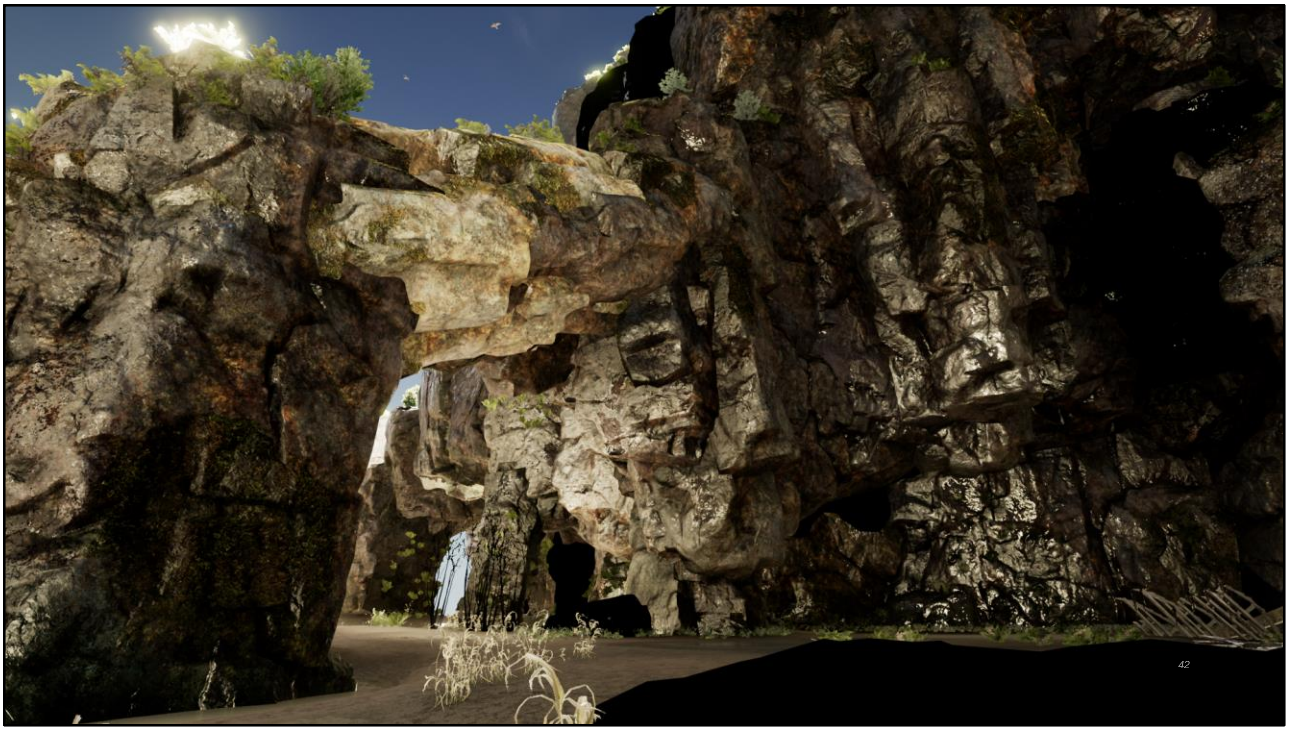
- Challenge: accurate lighting for larger meshes
- One sample per mesh?

One sample per mesh might give good enough lighting for small meshes, but we also need accurate lighting for larger meshes.
Is one sample per mesh going to be enough?

This is the same scene, but now the larger meshes…

.. shown in yellow are also lit each with a single sample.

The lighting appears obviously incorrect when there is large variation in the lighting around the mesh.
Meshes which sample lighting from an invalid location appear black.
This can happen when a large mesh is embedded in the terrain.



These problems are obvious in the final image.

Silicon Studio

Cost of computing samples

- Find nearby probes; compute weights; interpolate lighting
- Practical for small scenes, one sample per mesh
- Challenge: many thousands of meshes
- Challenge: re-interpolate everything when lighting changes

To get the lighting for each sample, we:

- Find nearby probes to sample from,
- find an interpolation weight for each probe,
- and then compute the weighted average.

With carefully optimized CPU code we can afford a few tens of thousands of samples per frame.

Todays games draw hundreds of thousands of meshes per frame, using instanced rendering.

With Enlighten we need to re-interpolate every sample when lighting changes.

This poses a challenge even when we sample just once per mesh.

enlighten

Placing probes efficiently

- Challenge: how to distribute probes efficiently
- Artist can place each probe by hand
- Completely impractical in large world

Another challenge:
We need to place probes throughout the scene, to provide lighting for moving meshes and volumetric effects.
Ideally we would cover a large area using few probes.
This work is often done by an artist placing each probe by hand..
… but this is not practical in a larger world.

enlighten

Summary: lightmaps and probes

- Difficult to unwrap lightmap UVs for a complex mesh
- One probe sample is not enough for a large mesh
- Sampling lighting from probes can be expensive
- Not practical to distribute probes efficiently by hand

While lightmaps and probes provide efficient rendering, each technique poses challenges:
It’s very difficult to produce good lightmap UVs for a complex mesh.
If the mesh is large, a single probe lighting sample is not enough to provide good lighting.
We can only afford to compute a limited number of probe lighting samples on the CPU.
And, distributing probes throughout the world by hand is not practical in a large scene.

Volume Lighting Vision

We set out to address these challenges with our volume lighting solution.
So, what would a good solution look like?

These trees are indirectly lit using probes, with one sample per mesh.
Each tree looks flat.

With one sample per screen pixel, the occlusion from the leaves brings out the shape of the tree.

This is the lighting with one sample per mesh; there is no variation across each mesh.

.. but with one sample per pixel, there is much more variation in the indirect lighting.

Silicon Studio

ENLIGHTEN™

Required: per-pixel sampling

- High fidelity lighting for large meshes
- Complex lighting variation within a mesh
- Consistent lighting across adjacent meshes
- Ideal: one sample per screen pixel

This gives us the first requirement:
To provide a high fidelity alternative to lightmaps, we need complex lighting variation within a mesh, and consistent lighting between adjacent meshes.
We need more than one sample per mesh; our target is one sample per screen pixel!

Required: scalable rendering

- Many thousands of meshes
- Hundreds of thousands of GPU particles
- Lighting across huge volumes
- Cost must scale predictably with world size and complexity

The next requirement:
For massive game worlds, the cost of rendering must scale well.
It must handle many thousands of meshes, both static and moving, and hundreds of thousands of GPU particles.
.. provide lighting across a huge volume of space
.. and the rendering cost must scale predictably with the size and complexity of the world.

Silicon Studio

Required: minimal GPU time

- GPU is busy with other, more interesting problems
- Minimal GPU cost – in same ballpark as lightmaps
- Usable on low end hardware

We also want to limit the GPU cost to within the same ballpark as for 2D lightmaps.
I know you would rather spend your limited GPU time on more interesting problems!

Silicon Studio

Required: scalable authoring

- Placing probes is difficult and time consuming
- Artist time can be better spent!
- Provide control of location and resolution
- Artist chooses where to spend the lighting budget

An important requirement:
We must eliminate the manual work required to distribute probes in a large world.
The artist's time is too valuable to spend doing this!
While we can automate much of the process, we don't want to take too much control away from the artist.
The artist has a limited budget for lighting data, and they want to allocate a larger part of it to the important areas of the world that the player will spend more time in.

Silicon Studio

Required: real-time updates

- When lighting changes:
  - Compute lighting for every probe
  - Interpolate lighting for every sample
- Minimize the number of probes updated
- Minimize the cost of interpolating lighting

On top of all this, we have to handle real-time lighting updates, which poses some unique challenges.
When the lighting changes we must compute new lighting for every probe, and interpolate lighting for every sample.
To make this practical we have to minimize both the number of probes that must be updated, and the cost of interpolating lighting.

Silicon Studio

Summary: Volume lighting vision

- High fidelity probe lighting for large meshes
- Fast sampling for millions of meshes
- Efficient automatic distribution of probes
- Replace lightmaps for problematic types of mesh

So, our volume lighting vision gave us these requirements.
It should provide the same high fidelity lighting as 2D lightmaps, with variation across a mesh.
It should be practical to use with millions of meshes and with constantly changing lighting.
To enable ever larger game worlds, it should automatically place probes throughout the world.
Such a solution could replace lightmaps for large meshes that are difficult to unwrap.

Silicon Studio

Volume Lighting Solution

So how does Enlighten's volume lighting solution fulfil all these requirements?

Silicon Studio

Where to Place Probes

- With increasing resolution and more probes:
  - more accurate lighting
  - higher cost of updates
- Required resolution depends on properties of lighting

58

The choice we make about where to place probes is important because it determines both how accurately we can reconstruct the lighting, and the cost of updates.
The resolution we need depends on the properties of the lighting we want to reconstruct.

Properties of direct and indirect lighting

- Direct light is emitted by bright point sources
  - Quickly varying intensity
  - Sharp shadow transitions
- Indirect light is emitted by surfaces with a large area
  - Slowly varying intensity
  - Soft shadow transitions

Direct light is emitted by bright point sources, so we expect to see quickly varying intensity and sharp shadow transitions.
Indirect light is emitted by surfaces with a large area and is generally less intense, so we expect to see slowly varying intensity and soft shadow transitions.

Low resolution indirect lighting

- Indirect lighting requires fewer spatial samples
- Use dedicated probes for indirect lighting
- Place probes for indirect lighting at lower density

Remember that Enlighten computes only indirect lighting, so we can ignore direct lighting and shadows.
Our indirect lighting can be reconstructed with good enough quality using fewer probes than would be required for direct lighting.
We use a separate set of probes for indirect lighting, and distribute them across the world with much lower density.

ENLIGHTEN

Adaptive resolution indirect lighting

- Further from surface, rate of intensity change is smaller
- Less information required to reconstruct lighting
- Place more probes close to surfaces; fewer in open space

Another useful observation is that the rate of change of light intensity becomes smaller as the distance to the emitting surface increases.
When the rate of change in intensity is smaller, we can reconstruct the lighting with less information.
Based on this observation, we need to place more probes close to surfaces than out in open space.

Silicon Studio

Multi-resolution indirect lighting

- Scale to very large open environments
- Reduce lighting resolution for distant areas: LODs
- Place probes at multiple resolutions
- Choose resolution at runtime

The number of probes for which we need to compute lighting, increases in proportion to the visible area of the scene.
In a large open environment where the entire scene is visible, it wouldn't be practical to compute lighting for every probe in the world.
We need a way to reduce lighting resolution for distant areas, in the same way as we would reduce mesh detail using multiple LODs.
To enable this we need to place multiple sets of probes, each with successively lower resolution.
Then at runtime we can choose which set of probes to sample from based on distance to the viewer.

Summary: Where to Place Probes

- Indirect lighting only: low density of probes
- Increase density close to surfaces
- Multiple levels of detail

So we want to place probes at a low density throughout the world, with higher density close to surfaces.
We also want to our distribution of probes to provide more than one level of detail.

Silicon Studio

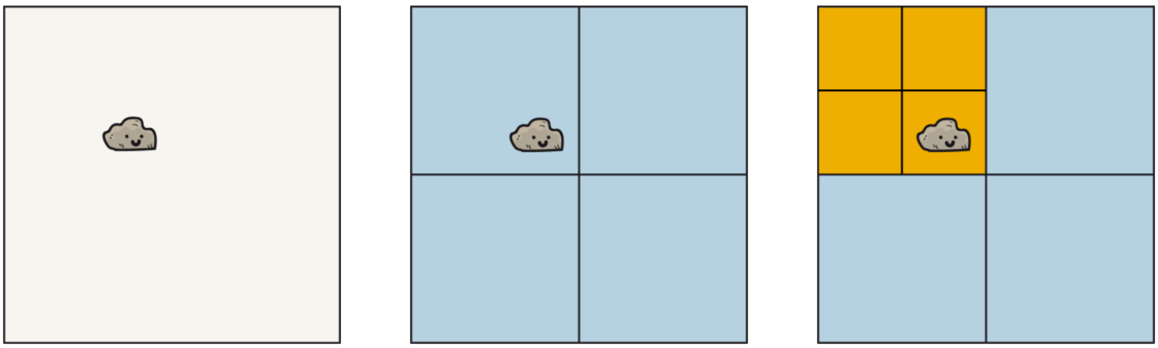
Automatically Placing Probes

How do we do this automatically, while keeping the right amount of control for the artist?

Silicon Studio

Adaptive subdivision

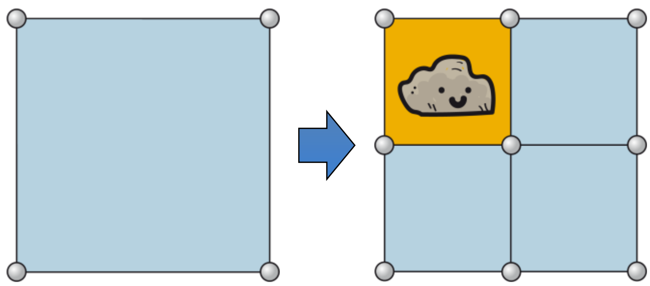
- Recursively subdivide volume around surfaces



To produce our initial distribution of probes, we start with a single large voxel, if it’s close to surfaces, subdivide it, then repeat for each of the eight child voxels.
We keep subdividing until we reach the minimum voxel size specified by the artist.
This process produces an octree structure.
The examples here show a 2D quadtree, limited to three levels with each shown in a different color, but the process is exactly the same with an octree in 3D.

Placing Probes

- Place probes at voxel corners
- Subdivide: 8 child voxels = 3x3x3 grid of probes



To distribute probes with varying density across the world, we can place one at the corners of each voxel.
Each subdivided voxel has eight children, so this results in a 3x3x3 grid of probes for each subdivided voxel.

Silicon Studio

Subdivision heuristic

- Fire rays in all directions to find distance to surface
- Use **harmonic mean** distance
  - ***N*** total number of rays fired
  - d_i distance to surface along *i*th ray
- Subdivide if closer than threshold

$$HM = \frac{N}{\sum_{i}^{N} \frac{1}{d_i}}$$

Source: [Tatarchuk2005]

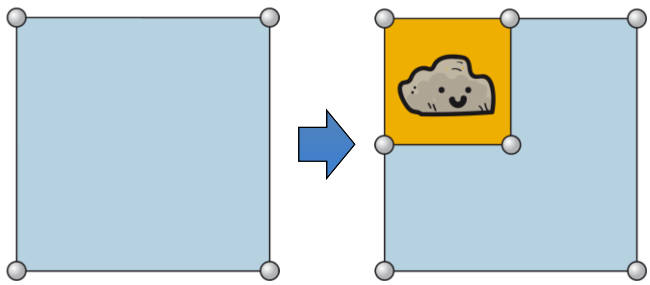
We use a simple heuristic to determine whether a voxel is close to surfaces.
To find the distance to surfaces, we fire a bunch of rays in all directions from the corners of each voxel.
We subdivide the voxel if any of its corners are within a predefined threshold distance to geometry.
We take the **harmonic mean** of the intersection distance for all rays…
… this gives us a more meaningful average distance than the **arithmetic mean**.
This subdivision process is based on the adaptive subdivision method proposed by Natalya Tatarchuk at GDC Europe 2005.

Silicon Studio

ENLIGHTEN

Placing fewer probes

- Minimize the number of probes created
- Place probes only for child voxels close to surfaces

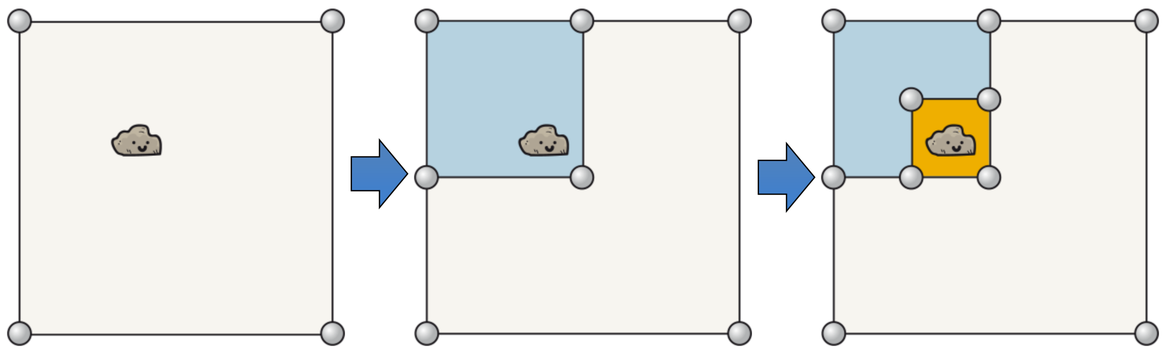


68

To minimize the total number of probes created, we chose a slightly different scheme.
When we subdivide a voxel, we record which child voxels are close to surfaces.
We only place probes at the corners of these child voxels.

Placing fewer probes

- Fewer probes to cover the same volume



With this scheme, we need fewer probes to cover the same volume.

Silicon Studio

ENLIGHTEN

Giving control to the artist

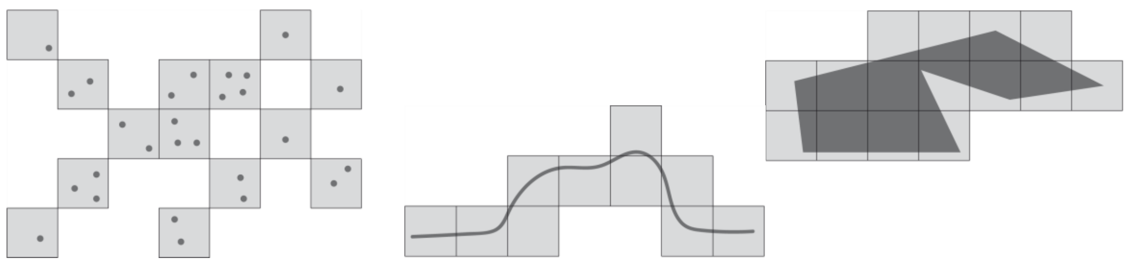
- Automatically place probes only where needed
- Limit resolution in specified areas
- Use **voxel bitmaps** to control location and resolution

How can we control the location and resolution of probes placed with this method?
We use voxel bitmaps to control both.

Silicon Studio

Voxel bitmaps

- Each voxel is either occupied or empty
- Easy to convert point clouds, surfaces or volumes



A voxel bitmap is a grid of voxels, with some marked as occupied.
We can easily convert any primitive to a voxel bitmap, including point clouds, surfaces and volumes.

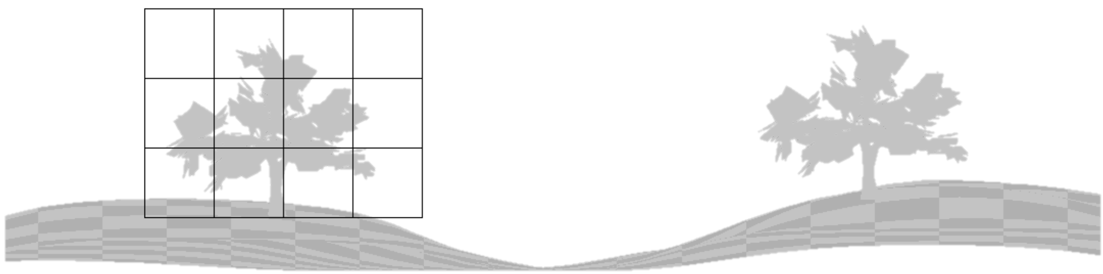
Silicon Studio

Voxel bitmaps example

Voxel bitmaps aren’t particularly usable for an artist, so how can we expose the controls in a more friendly way?

The terrain is lit using a lightmap, and the artist chooses to light the trees using probes.
The trees are static meshes, so we need to place probes around them to provide lighting.

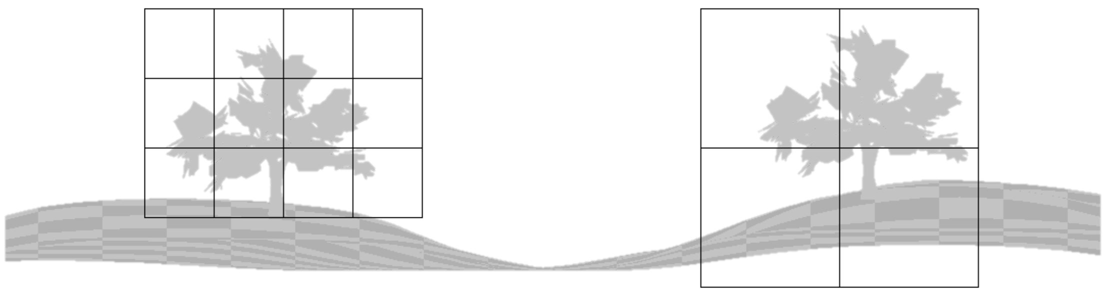
Voxels occupied by mesh



So we automatically add the trees to the voxel bitmap.
Here are the voxels occupied by the tree on the left.

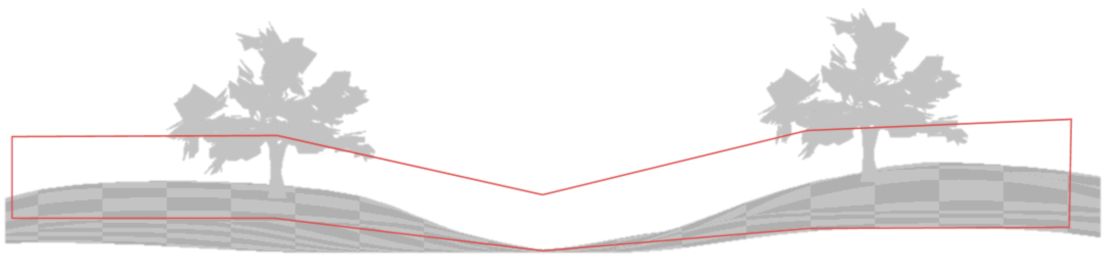
Silicon Studio

Per-mesh voxel resolution



The artist can choose a different lighting resolution for each mesh.
They choose to use half the resolution for the tree on the right, so we reduce its voxel resolution accordingly.

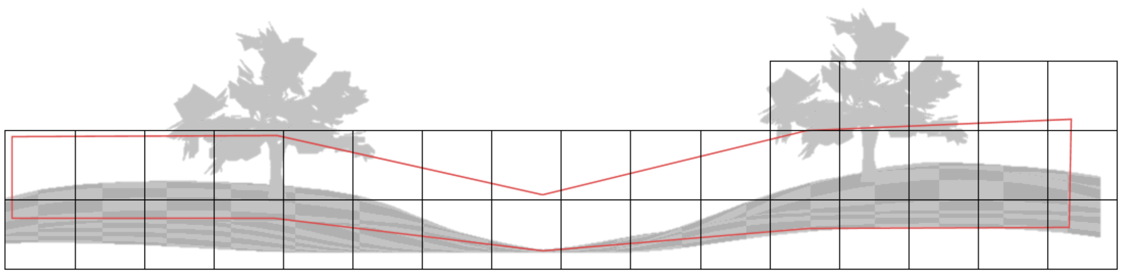
Volume for moving meshes



In this example we don't know in advance where moving meshes will be, so we ask the artist to indicate this by placing a volume.
This volume covers where the player can walk on the terrain.

Silicon Studio

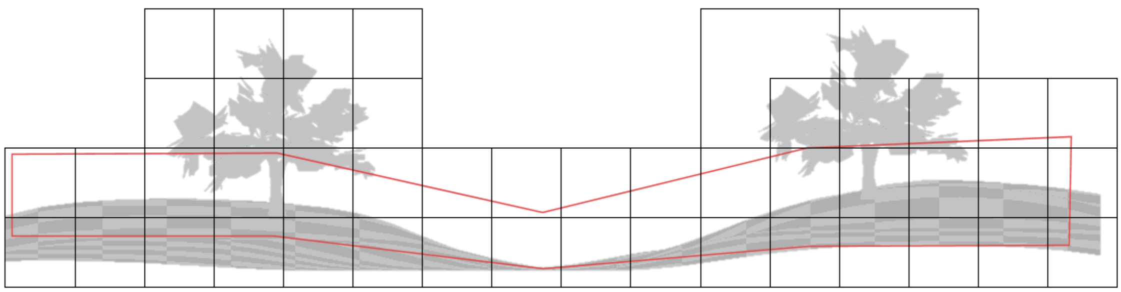
Voxels occupied by volume



We add the volume to the voxel bitmap.

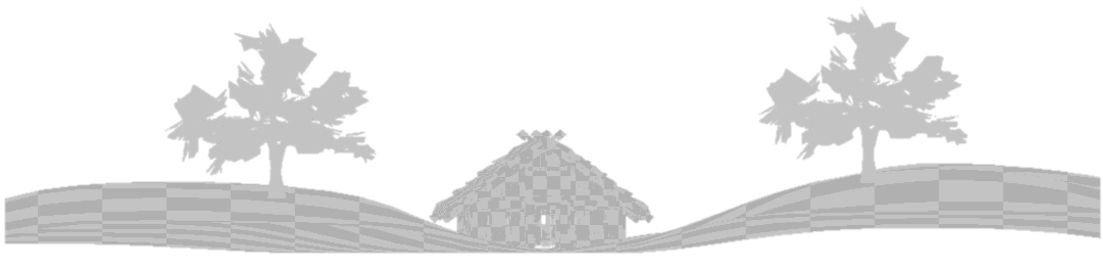
Silicon Studio

Voxels occupied by meshes and volume



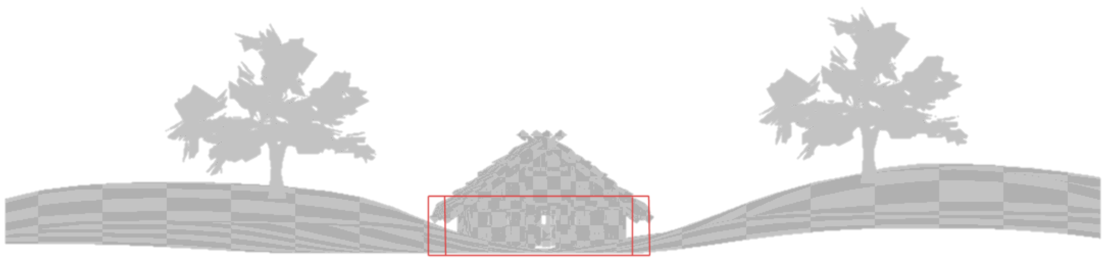
And these are the combined voxels for both trees and the volume.
The larger and smaller voxels are aligned.

Increase resolution for interior



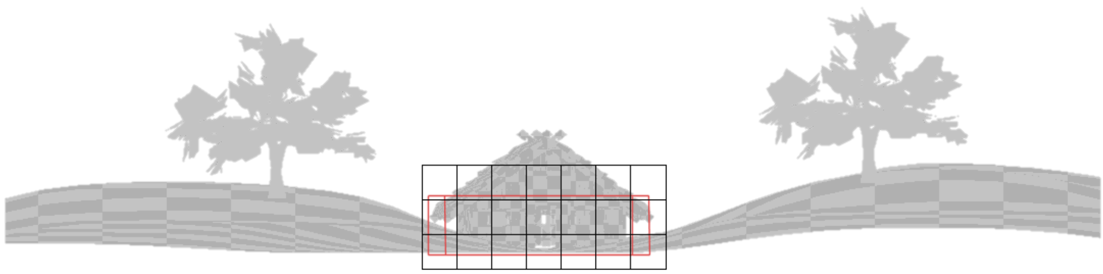
The artist can also choose a different lighting resolution for each volume.
To increase the density of probes to cover the interior of this house…

High resolution volume covers interior



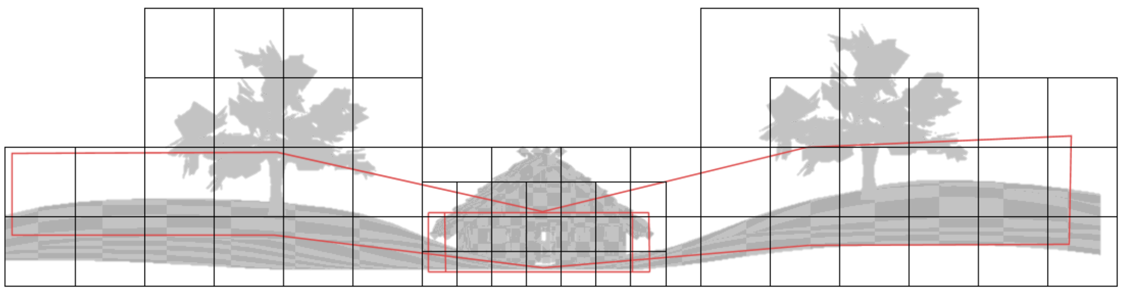
They can place a high resolution volume over the interior…

Voxels occupied by high resolution volume



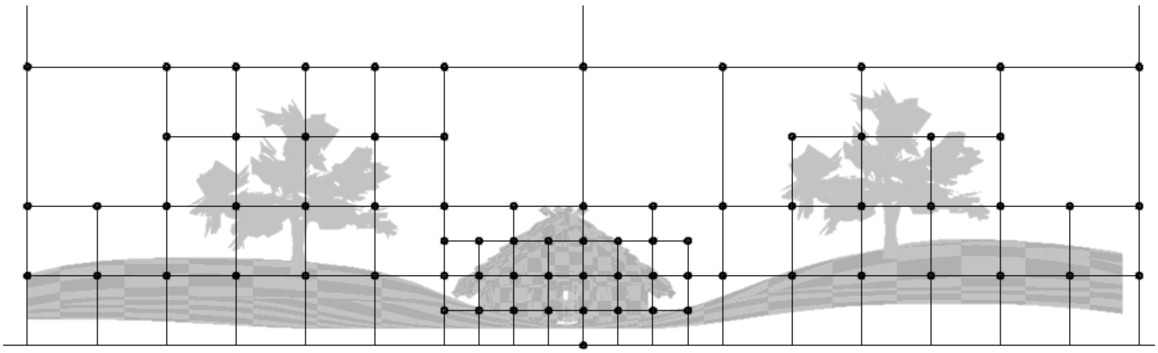
We add the high resolution voxels to the bitmap.

All occupied voxels combined



And these are the combined voxels for both trees and both volumes.
This tells us where to place probes.

Distribute probes based on voxel bitmap



When we distribute probes in the world, we use the voxel bitmap to control the location and resolution.
We subdivide the octree only in areas close to surfaces that are also occupied by voxels in the bitmap.
In the resulting distribution of probes, the density depends on both distance to surfaces and the resolution specified by the artist.

Silicon Studio

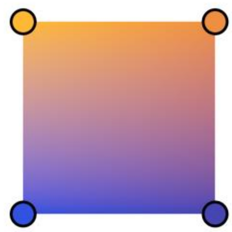
Interpolating Probe Lighting

Now that we have some probes, we need an inexpensive way to interpolate a sensible lighting value at any point in the world

Silicon Studio

Trilinear interpolation

- Linear interpolation in 3D; simple and low cost

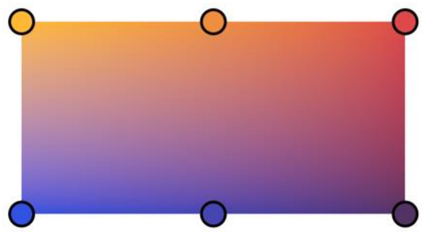


To compute the lighting for a sample point within the volume, we use 3D linear interpolation, known as trilinear.
The example shows the result of 2D bilinear interpolation – this is like a slice through the 3D volume.
To do this we just need to know the voxel containing our sample, and the lighting values of the probes at each of the voxel corners.
This is simple to implement, and helps us to minimize the cost of each sample.

Silicon Studio

Seamless lighting?

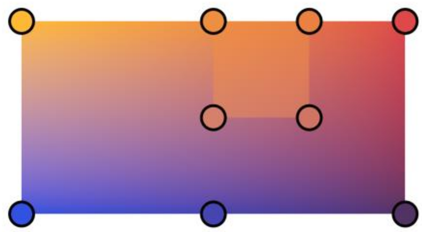
- Seamless interpolation between voxels of the same size



We want to sample probes once per screen pixel, so we need seamless lighting across the volume.
With this method we have seamless lighting between adjacent voxels of the same size.

Seams between voxels

- Seams between voxels of different sizes

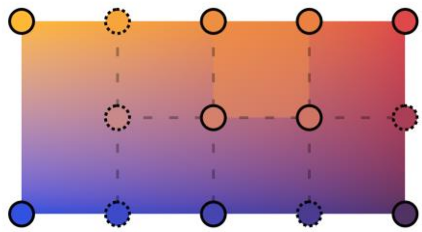


However, there is a seam in the lighting at the boundary between two voxels of different sizes.
How can we fix this without making interpolation more complicated and expensive?

Silicon Studio

Fix seams with virtual probes

- Add **virtual voxels** and **virtual probes** around smaller voxel



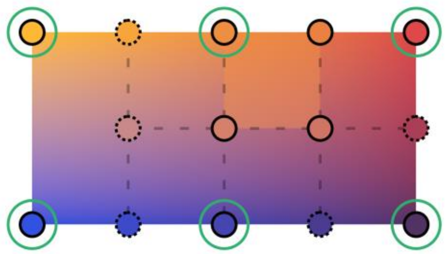
We add voxels adjacent to the smaller voxel until its neighbors are all the same size.
We call these **virtual** voxels.
We add virtual probes at the corners of these virtual voxels, as shown.

Silicon Studio

enlighten

Reconstruct virtual probes at runtime

- Interpolate from probes in the parent voxel



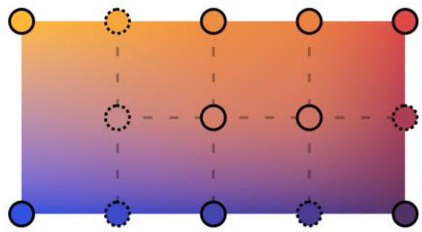
We reconstruct the lighting for a virtual probe at run-time by interpolating from probes in its parent voxel, circled in green.

Silicon Studio

ENLIGHTEN™

Seamless lighting!

- Seamless lighting between virtual voxel and parent voxel



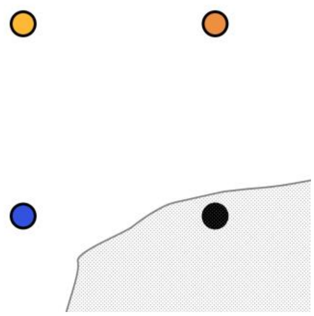
The lighting for a virtual probe now matches the interpolated lighting result for a sample in the parent voxel at the same location.
This give us seamless lighting between smaller and larger voxels.
However, there’s still another interpolation problem to solve…

Silicon Studio

ENLIGHTEN™

Probes outside the world

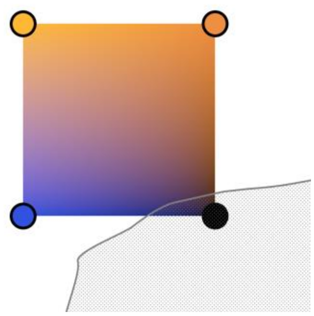
- Probes placed outside the world marked as **invalid**



Some probes might be placed outside the world.
Here, the black probe is embedded in a wall.
We automatically detect probes that see back-faces and mark them as **invalid**.

Sampling invalid probes

- Trilinear interpolation might sample from invalid probes



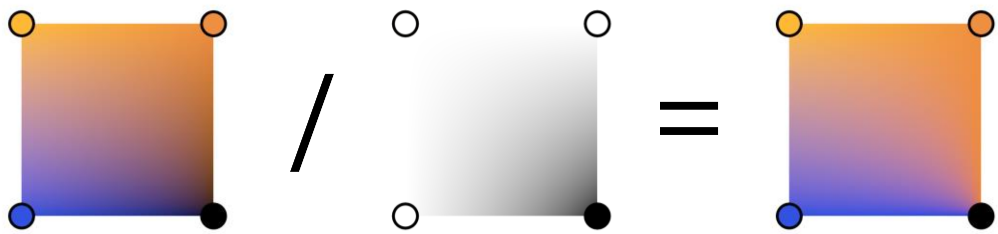
We don't want to sample lighting from these invalid probes, but our simple interpolation method means we can’t avoid it.
Invalid probes have all-zero lighting values, which is visible as dark areas in the lighting.

Silicon Studio

ENLIGHTEN

Compensate for invalid probes

- 1-bit weight channel: valid = 1, invalid = 0

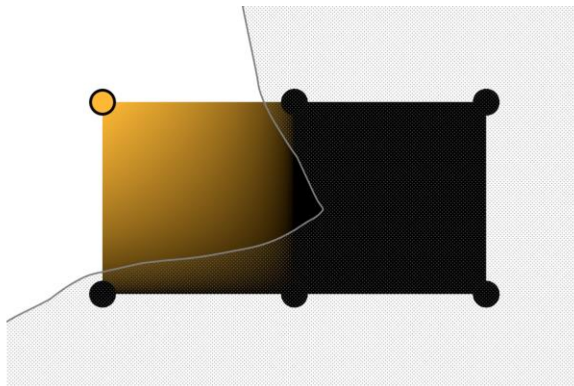


92

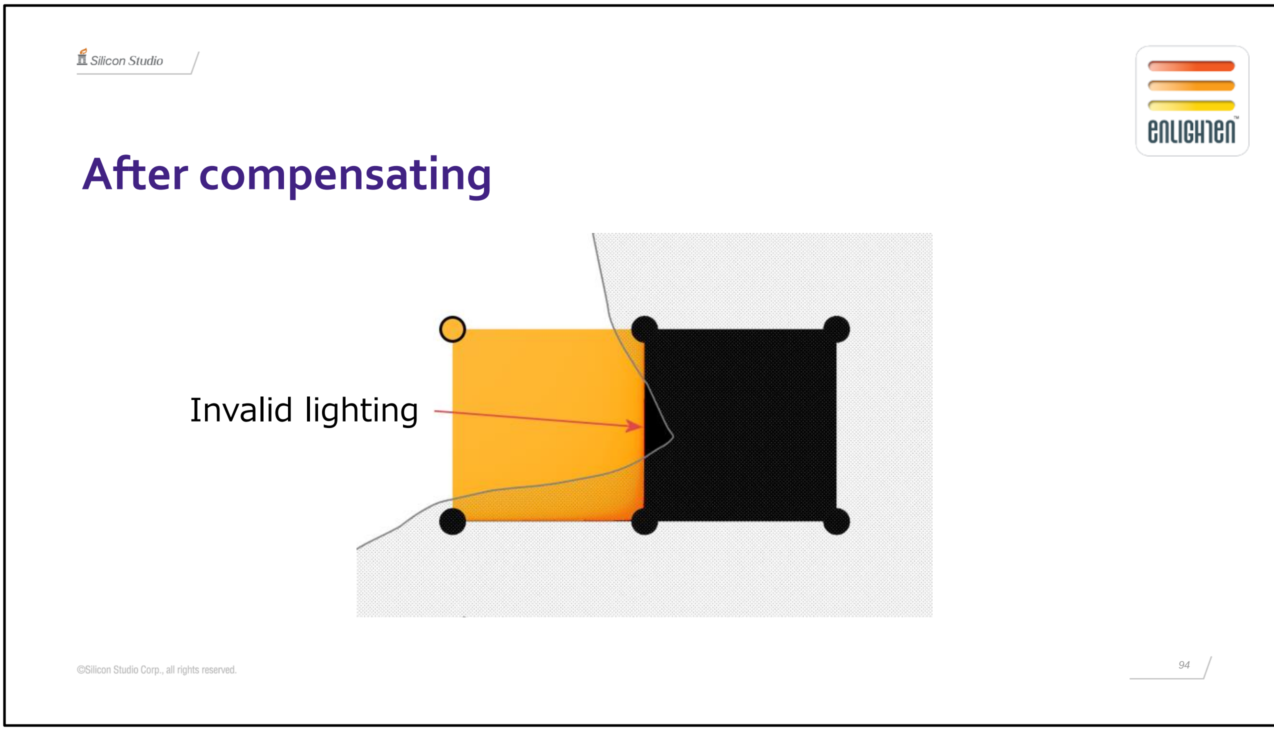
To solve this, we augment the probes with a one-bit **weight** channel, which contains one for valid probes and zero for invalid probes.
After interpolating both the lighting and the weight, we divide by the weight to normalize the result.

Silicon Studio

Many invalid probes



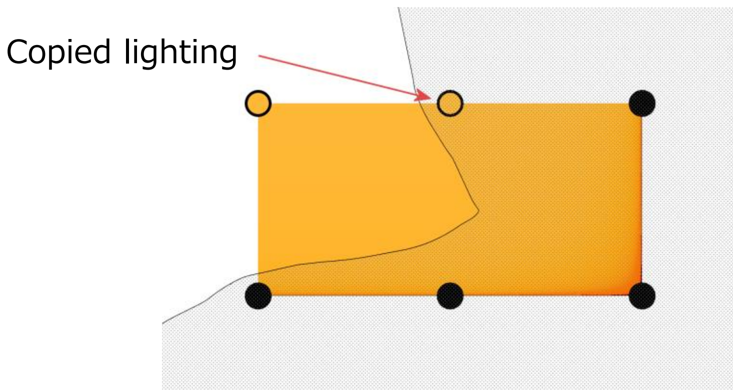
In the extreme case here, all but one probe are invalid.
This is the result of interpolation without compensating for invalid probes.



After we divide by the weight channel to compensate for the invalid probes, there is still a problem.
When both of the probes on the same edge are invalid, the lighting for samples at the edge is invalid.

ENLIGHTEN

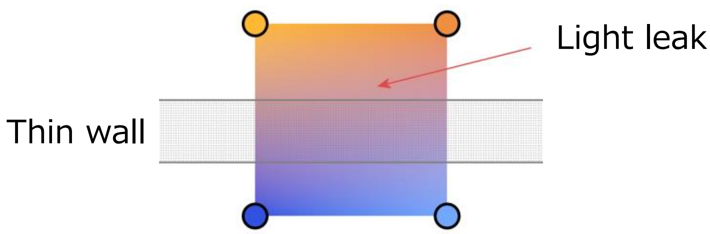
Copy from adjacent probe



Fortunately, we can automatically detect and fix this problem.
We choose one of the probes of the invalid edge and copy its lighting from an adjacent valid probe.

Light leaks through walls

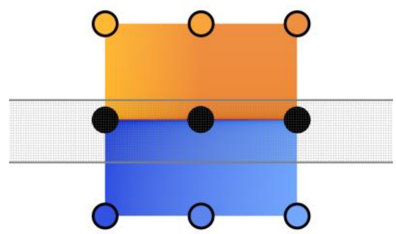
- Where a voxel spans disconnected areas



We also have to deal with the problem of light leaking through thin walls.
When probes are placed in regular grids, we see this type of **light leak** where a voxel spans disconnected areas.
The adaptive placement process places more probes close to walls, which gives better results than a regular grid of probes.
Even with adaptive placement, light leaks can still occur, if the probe resolution is insufficient.

Fixing light leaks

- Increase resolution in problem area



The artist increases the probe resolution in the problem area to prevent the leak. We use the weight channel to compensate for the invalid probes embedded within the wall.

Silicon Studio

Sampling Probes in a Shader

We now have seamless lighting with only simple trilinear interpolation.
How do we sample lighting from probes in our shader?

Silicon Studio

Hardware accelerated interpolation

- Sample from 3D texture in same way as lightmaps
- Enables millions of probe samples per frame
- Custom interpolation shader: load 8 probes?
- Xbox One: one filtered sample is up to 4 times faster

In a pixel shader or a compute shader, we can sample lighting from a 3D texture in the same way we would sample a 2D lightmap texture.
If we can stick to trilinear interpolation between eight probes, we can use use GPU texture mapping hardware to cheaply compute millions of probe lighting samples per frame – this would be impossible on the CPU.
If we used a custom interpolation shader, we have to load the data for each probe we want to interpolate from, before we can do any arithmetic.
In a quick test on the Xbox One, a single trilinear volume texture sample was up to 4 times faster than eight separate loads.

Silicon Studio

ENLIGHTEN

3D virtual texture

- 3D texture represents constant resolution regular grid
- Need seamless interpolation across large volume
- Not enough memory for 3D texture covering this volume
- Use a 3D **virtual texture** instead

  [Barret2008] [Mittring2008]

A regular 3D texture with a high enough resolution to provide seamless interpolation for the entire world would require gigabytes of video memory.
Instead we use a 3D **virtual texture** to efficiently cover both areas with a high density of probes, and large areas of empty space containing very few probes.
For more about virtual textures, refer to Sean Barret's GDC 2008 talk, and Martin Mittring's talk from SIGGRAPH 2008

Virtual textures

- Represents a regular texture too large to keep in memory
- Divide regular texture into fixed size **tiles**
- Only a subset of tiles resident in **tile cache**
- Sample **indirection texture** to find tile at position in world

A **virtual texture** represents a regular texture which is too large to completely load into memory.
We first divide the original texture into a number of fixed size **tiles**, and load a subset of tiles into the **tile cache**.
Each texel of the **indirection texture** contains a pointer to a tile in the tile cache.
We sample the indirection texture to find which tile covers a given position in the world.

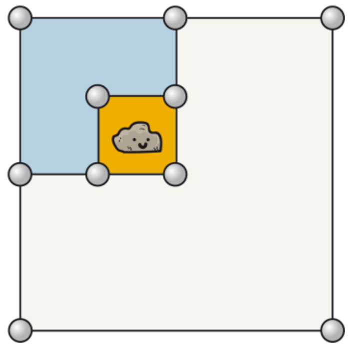
Silicon Studio

Choosing the tile size

- Indirection texture size = original texture size / tile size
- Larger tiles -> smaller indirection texture
- Smaller tiles can adapt more tightly to resolution changes
- We chose relatively small 3x3x3 tiles

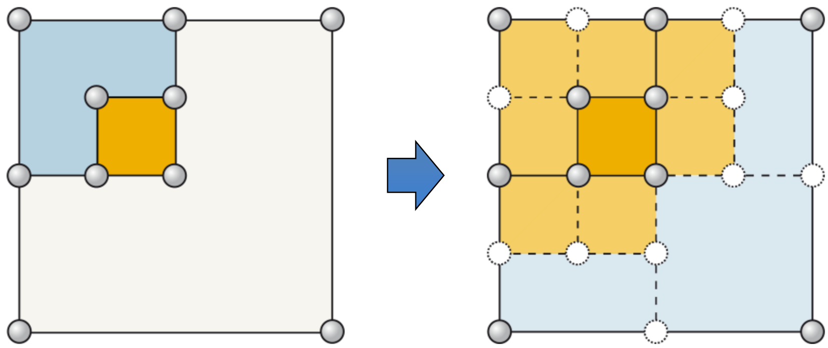
Our choice of tile size for the 3D virtual texture is important.
If we choose a large tile size, we can cover the same volume of space with a smaller indirection texture.
On the other hand, with smaller tiles the texture can adapt more tightly to local changes in resolution.
We chose a relatively small tile size of 3x3x3 because we expect large changes in resolution.

Octree as a virtual texture



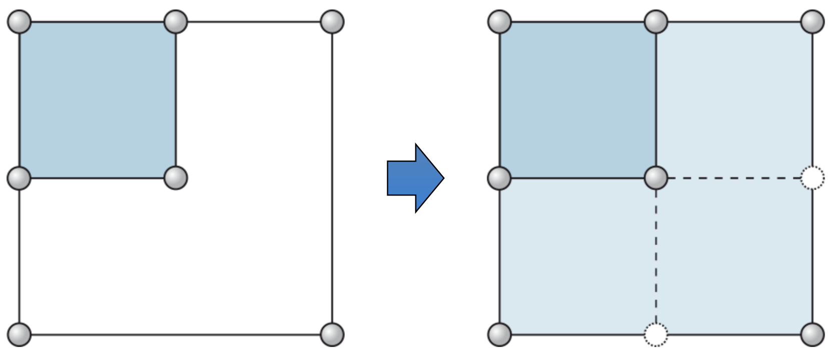
So let's go back to the octree that we created when automatically placing probes.
Each level has successively smaller voxels, colored here with blue and yellow.
I'll walk you through an example of how we build a 3D virtual texture from this octree data.

Add virtual voxels and probes



To enable seamless interpolation of the lighting between smaller and larger voxels, we add virtual voxels and probes

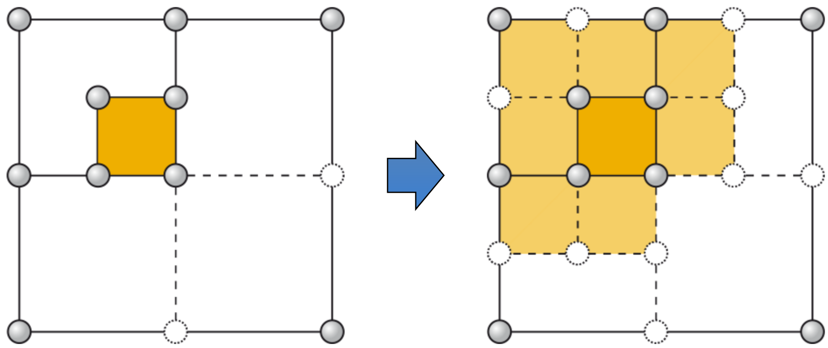
Add virtual voxels and probes



To make this really clear, here are the virtual voxels for the blue level…

Silicon Studio

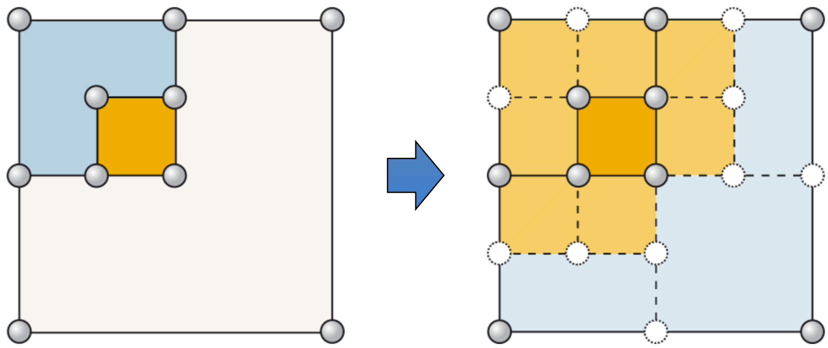
Add virtual voxels and probes



… and for the yellow level

Silicon Studio

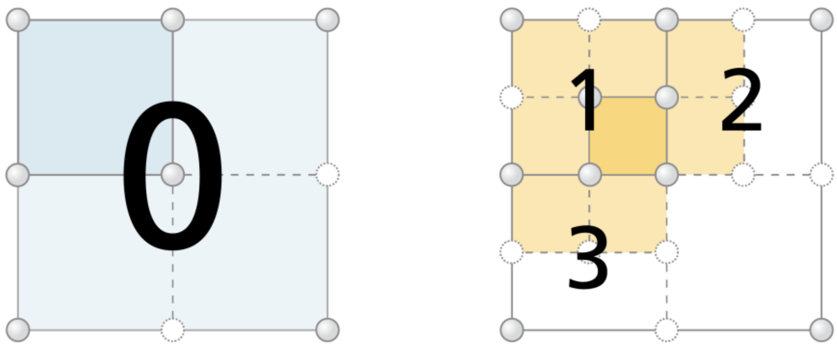
Add virtual voxels and probes



And now both levels combined.

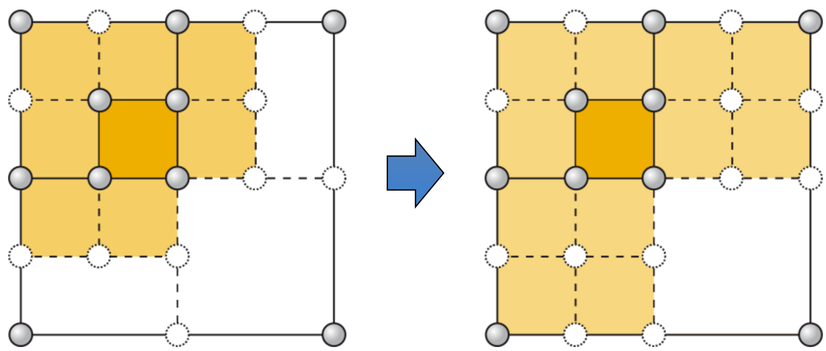
Silicon Studio

Group voxels into tiles



We group voxels into 2x2x2 tiles.
Here, each tile is numbered separately.
Tiles 2 and 3 on the right are not completely filled – and we want to fill them up…

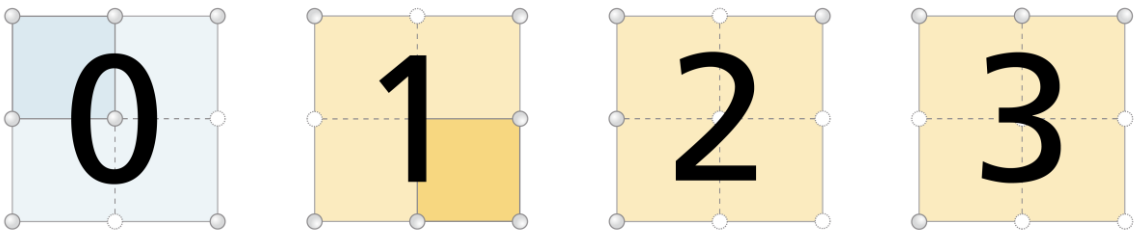
Fill partial tiles with virtual voxels



.. so for these partially empty blocks, we add more virtual voxels and probes, like so.

Final set of tiles

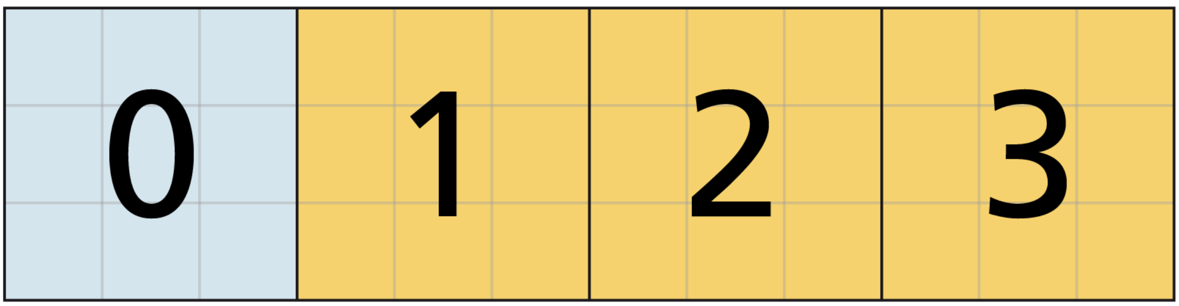
- Each tile's **probes** are a 3x3x3 texel grid



Now we have split the octree into this set of four tiles.
Each tile forms a 3x3x3 grid of probes, which fits perfectly in a virtual texture tile with one probe per texel.

Tiles in tile cache

- Regular 3D texture containing all tiles

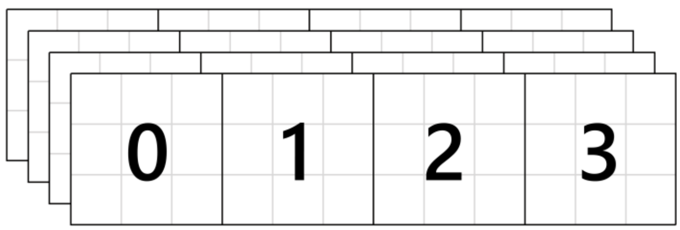


Each tile is a 3x3x3 block of texels in our **tile cache**.
The grid here shows the texels, and each texel represents a single probe.
The **tile cache** is a regular 3D texture containing a pool of virtual texture tiles, to which we can easily add and remove tiles at runtime.

Silicon Studio

Tile data

- 3 sets of 4 SH coefficients in 3 half-float textures
- 8-bit weight texture



112

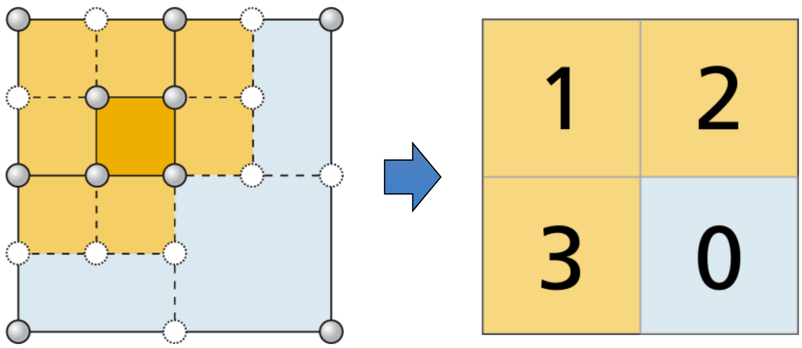
The tile cache is made up of multiple textures with the same tile layout.
We store three sets of four spherical harmonic coefficients per probe, in three separate half-float textures.
The weight channel is stored in a separate 8-bit texture.

Indirection texture

- Flattened representation of octree
- Indirection texel (32bpp):
  - tile's offset within the tile cache
  - tile's octree level
- Sample to find tile covering world-position
- Compute location and sample lighting from tile cache

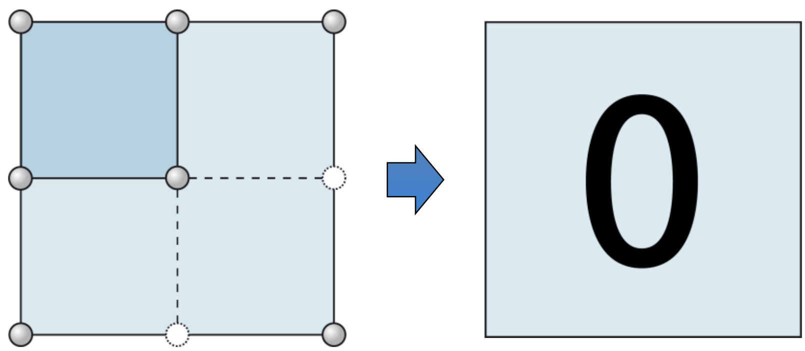
Now we have our probe data in a 3D texture, how do we sample it in a shader?
To enable this, we build an indirection texture that contains a flattened representation of the octree.
We point-sample the indirection texture to get the offset of a tile within the tile cache, and the tile's octree level.
From this we compute the location in the tile cache from which to sample the probe lighting.

Flattened octree



On the left is the octree, and on the right is the corresponding indirection texture.
For this 2D example, the indirection texture has four texels.
The yellow texels point to the smaller tiles, 1, 2 and 3 in the yellow level of the octree.
The bottom right texel, points to the blue tile, zero from the next octree level.

Indirection texture LODs

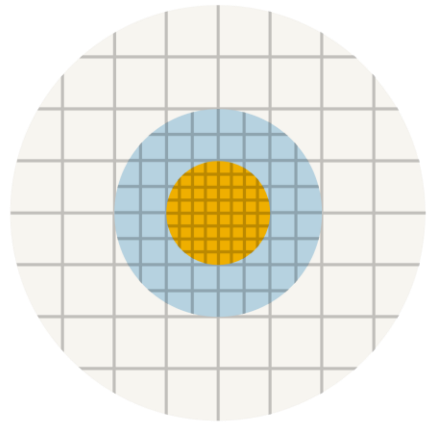


For each level of the octree we maintain a separate LOD level in the indirection texture.
On the left is the blue level of the octree, and on the right is the corresponding indirection texture.
The indirection texture for this level has half the resolution, and points only to tile zero.

Silicon Studio

Per-pixel LOD

- Distance from viewer determines indirection texture LOD
- Reduced lighting resolution for distant areas

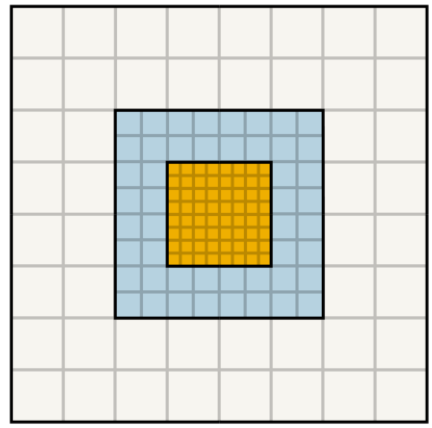


116

When sampling the indirection texture in our shader, we choose a LOD level based on the distance from the viewer.
This gives us reduced lighting resolution for distant areas.

Indirection texture clipmap

- Must limit size of indirection texture
- **3D clipmap** centered on viewer
  [Pantaleev2015]
- High resolution only around viewer



We don't have enough memory to cover the whole world with a single indirection texture at the resolution we need.
Instead, we cover only the area of the world around the viewer, using a **3D clipmap**.
All LOD levels of the clipmap have the same number of texels, and each successive LOD covers twice the area of the world.

Silicon Studio

Summary: Volume Lighting Solution

- Automatic placement with control of resolution
- Trilinear interpolation; fix seams, invalid probes, leaks
- Scalable rendering with 3D virtual texture

Enlighten's volume lighting solution places probes automatically, but still gives the artist complete control.
We use hardware accelerated trilinear interpolation with simple solutions for seams, invalid probes and light leaks.
The 3D virtual texture enables very large scenes with a fixed run-time overhead.

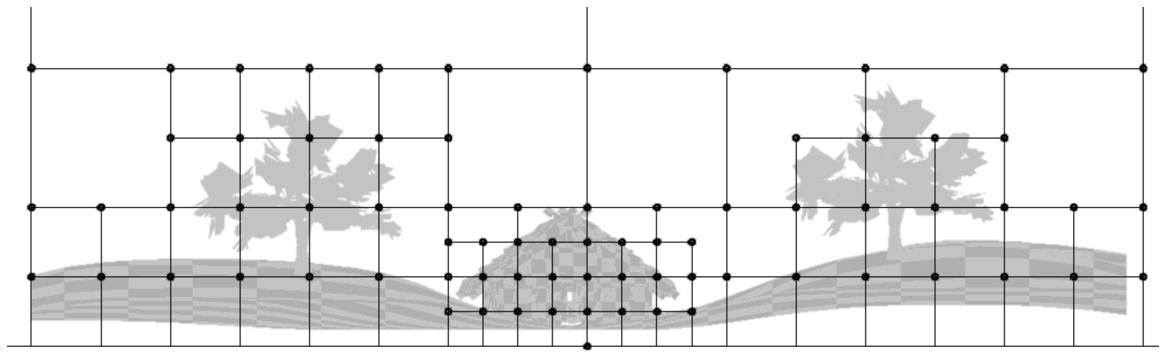
Silicon Studio

One Last Optimization

We have left room for one last optimization.

Silicon Studio

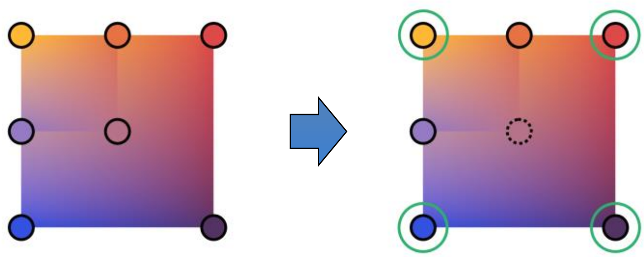
Minimize the number of probes



At runtime we must compute the lighting for each probe using its form factors. This takes time, so we want to minimize the number of probes for which we need to do this.

Identify superfluous probe

- Can be replaced with a **virtual probe**



In this example, the lighting for the probe in the center is almost the same as the interpolated lighting at the same location in its parent voxel.
If we converted the center probe to a virtual probe, the difference would be imperceptible.

Requires final lighting

- Compare lighting values to identify unnecessary probes?
- Enlighten lighting is updated at run-time
- Final lighting values are not known in advance
- Another way?

The problem is, we need the final lighting value for the probe before we can detect this.
We don't know the final lighting values in advance, so we must find another way.

Compare form factors

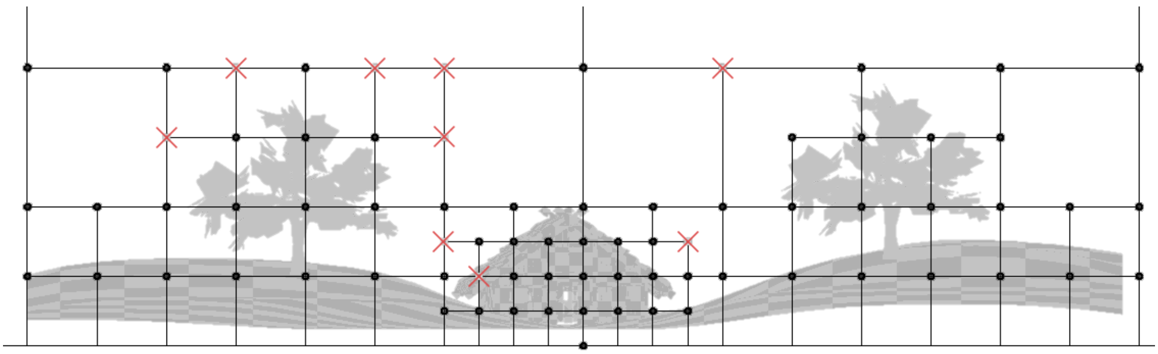
- Enlighten precomputes form factors for each probe
- To identify unnecessary probes:
  - interpolate form factors within parent voxel
  - compare vs precomputed form factors
  - if error is less than threshold, convert to virtual probe

We do have the precomputed form factors for each probe, from which the final lighting will be computed at runtime.
Instead, we interpolate the form factors within the parent voxel and compare the result with the precomputed form factors.
If the error is acceptable we convert the probe to a virtual probe.

Silicon Studio

ENLIGHTEN™

Probes converted to virtual



So we can discard their form factor data for these probes.
Computing the interpolated lighting for virtual probes takes less time than using the form factors.

Volume Lighting Impact

So now we're done with the technical part!
Let's look at the impact Enlighten's volume lighting has in production.

In Hellblade: Senua's Sacrifice, the character of Senua is the centerpiece of the game.
Enlighten's volume lighting provides global illumination for Senua.
The lighting really helps to ground the character in the world.
The automatic placement of probes throughout the world also saved a lot of art time.

The technique provides high fidelity indirect lighting for the extensive forests and vegetation, volumetric fog and translucent particles.

The Hellblade world was built by a very small team, with a single environment artist. Using Enlighten volume lighting helped to streamline their workflow…

.. with the option to use probe lighting instead of lightmaps for complex meshes, they don’t need to worry about lightmap UVs.
Lightmaps are still a good solution or meshes with large flat surfaces like large architectural features and terrain.
Fortunately we can easily mix both techniques within the same world to get the best of both.

With real time updates in the editor, the artist doesn’t need to wait for a bake to iterate on the lighting.
All that artist time can instead be put into creating an awesome visual experience.

All these images were captured from the PS4 version of Hellblade: Senua’s Sacrifice.
The team at Ninja Theory did an amazing job!

Silicon Studio

The Future of Enlighten

- Further streamline artist workflow
- Scale to even more massive worlds
- Simplify integrating Enlighten SDK
- New techniques that complement Enlighten

In future releases of Enlighten we plan many improvements.
We will further streamline the workflow and user experience to make it easy to get the best results with Enlighten.
As game worlds continue to get larger and more detailed we'll ensure Enlighten can scale up to meet the challenge.
We're also working on new documentation and sample code, to simplify integrating the Enlighten SDK into your engine.
Finally, we will continue to research and develop rendering techniques that complement Enlighten.

Where can I get Enlighten?

- Available for Unreal Engine 4
- SDK can be integrated into any engine
- Supports all major platforms

We provide an integration of Enlighten into Unreal Engine 4, and a standalone SDK for integration into your own engine.
Enlighten supports all major platforms, including Nintendo Switch

Please come visit the Silicon Studio booth in the expo hall to find out more.

Special Thanks

Dominic Matthews

Eri Ikuno

Leon O'Reilly

Geomerics

Ryan Jones

Jun Yoshino

Tara Nasupovic

Filippo Ceffa

Silicon Studio

ENLIGHTEN™

william.joseph@siliconstudio.co.jp

https://www.siliconstudio.co.jp/middleware/enlighten/en/

https://www.siliconstudio.co.jp/en/careers/ ←Hiring!

Questions?

References

- [Tatarchuk2005] Natalya Tatarchuk – GDCE 2005 "Irradiance Volumes for Games" Link
- [Barret2008] Sean Barret – GDC 2008 "Sparse Virtual Textures" Link
- [Mittring2008] Martin Mittring – SIGGRAPH 2008 "Advanced Virtual Texture Topics" Link
- [Panataleev2015] Alexey Pantaleev – GTC 2015 "NVIDIA VXGI: Dynamic Global Illumination for Games" Link