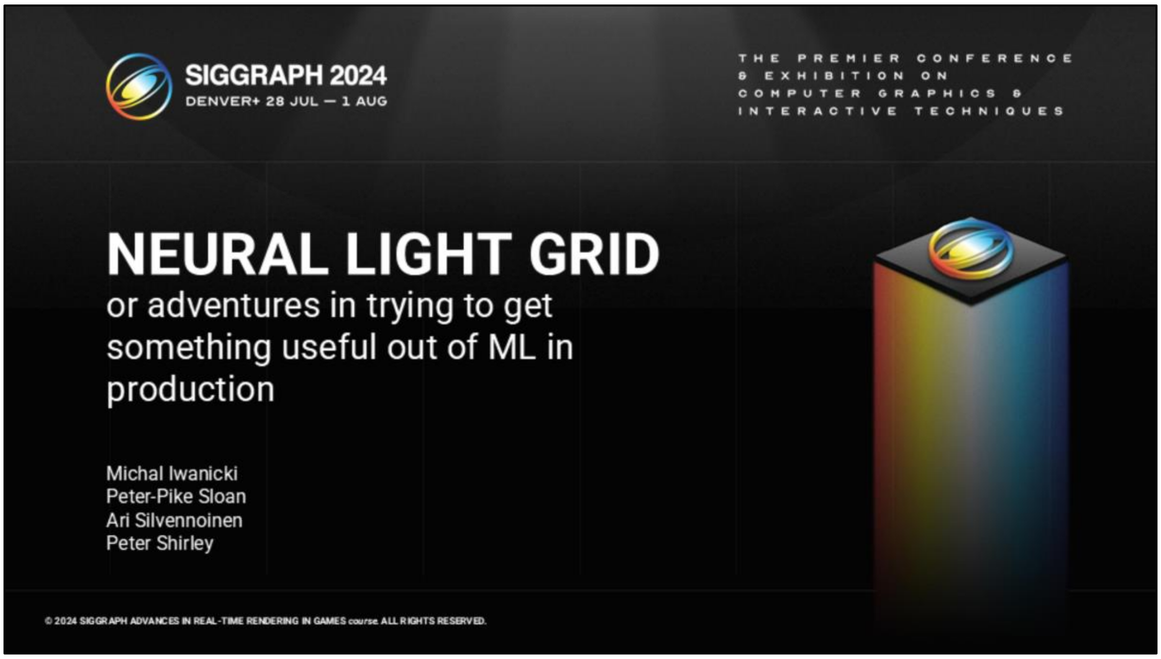


SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

THE PREMIER CONFERENCE & EXHIBITION ON COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES

NEURAL LIGHT GRID

or adventures in trying to get something useful out of ML in production

Michal Iwanicki
Peter-Pike Sloan
Ari Silvennoinen
Peter Shirley

Hi everyone, thanks for the intro Natasha.
My talk today is titled “Neural Light Grid”.
There was some smarter subtitle initially, but then I changed it to “adventures in trying to get something useful out of ML in production”, because this talk is just about the final solution as it is about everything that I tried and didn’t work.
Many people helped me along the way, at some point we had a more formal paper describing these ideas and I wanted to acknowledge Peter-Pike, Ari and Pete who worked on that too.

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

INTRODUCTION

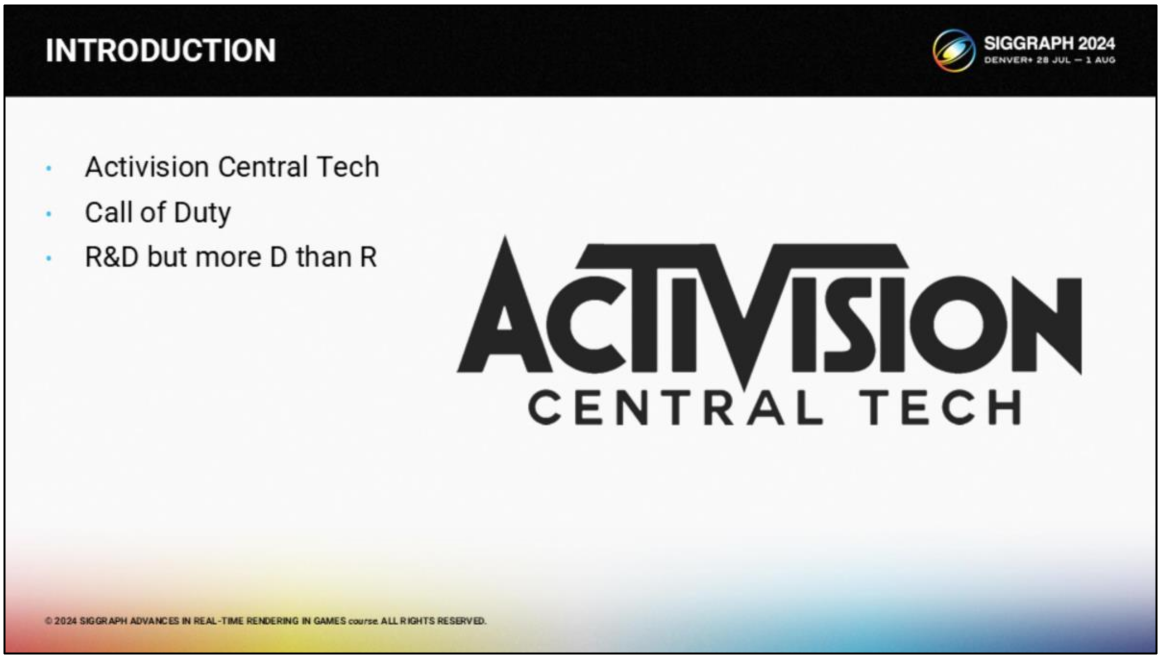
SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

MICHAŁ IWANICKI

ENGINEER

- CD Projekt RED
- Lionhead
- Naughty Dog
- Activision
- michal.iwanicki@activision.com

I'll skip my introduction, as Natasha already did this



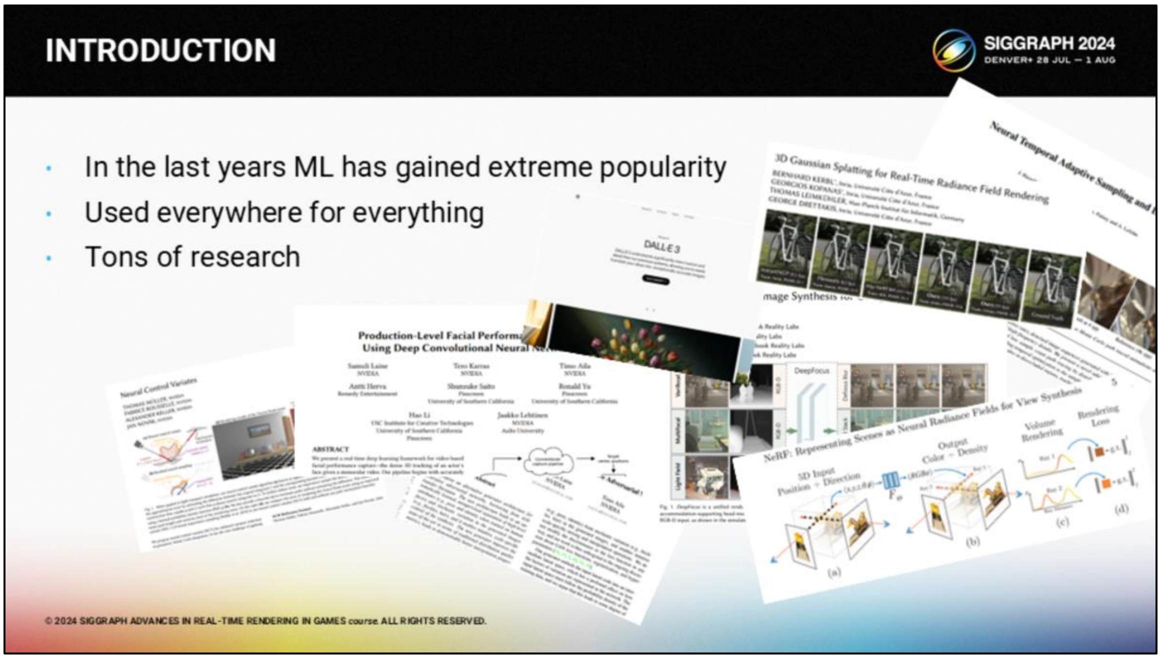
INTRODUCTION

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Activision Central Tech
- Call of Duty
- R&D but more D than R

ACTIVISION
CENTRAL TECH

I work for Activision, in a Central Technology group. We focus mostly on the Call of Duty franchise, and we help all our the studios achieve their goals.
We're sort-of an R&D group, but it's mostly the 'D' part, we just have a bit more flexibility and can sometimes work on a slightly further reaching projects, but it's not really a typical research group, we're very much involved in day-to-day production



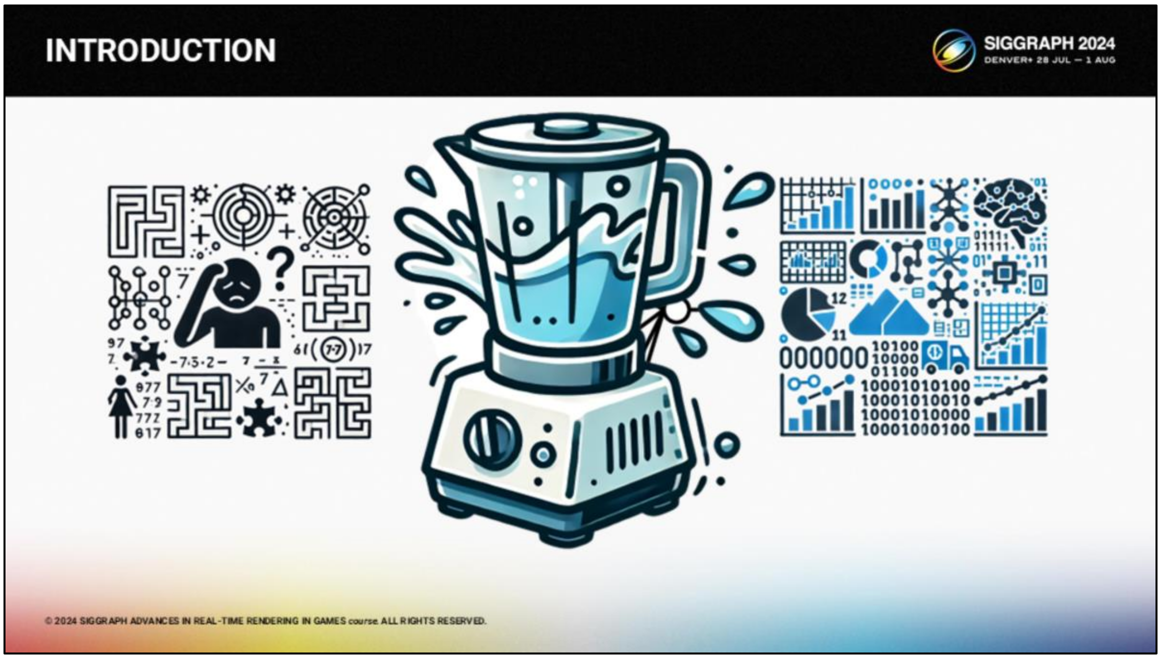
INTRODUCTION

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- In the last years ML has gained extreme popularity
- Used everywhere for everything
- Tons of research

Neural Control Variates
Production-Level Facial Performance Capture Using Deep Convolutional Neural Networks
DALL·E 3
3D Gaussian Splatting for Real-Time Radiance Field Rendering
Neural Temporal Adaptive Sampling and Denoising
DeepFocus
NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis

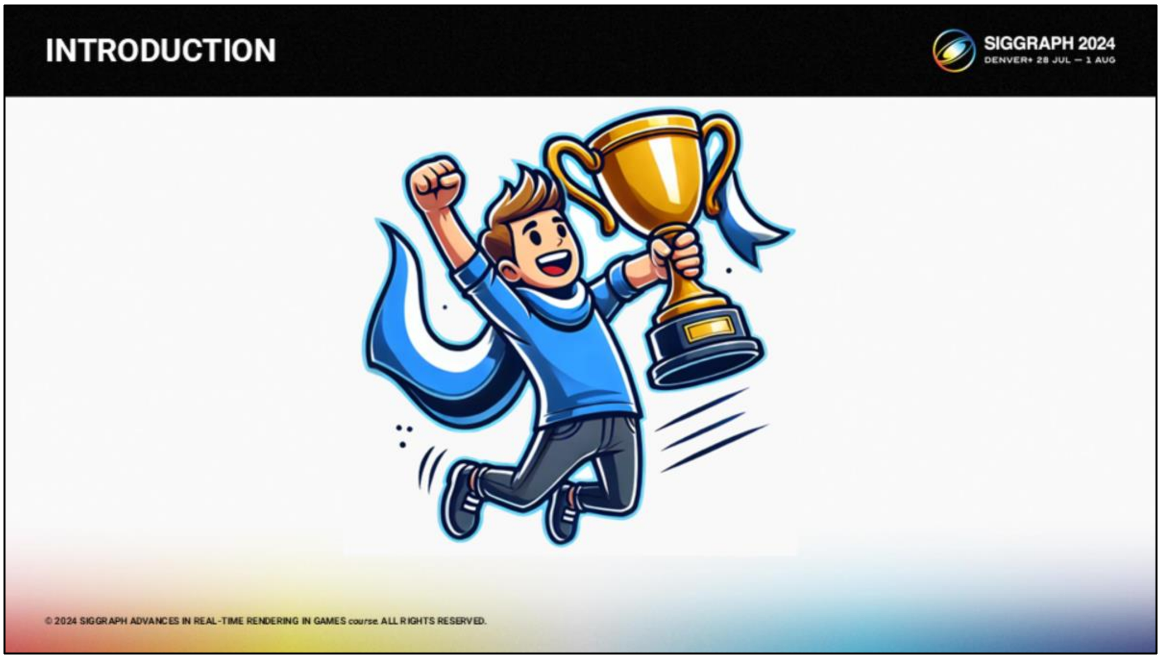
Ok, so what is this talk about?
As I'm sure you've noticed, in the last few years there have been an explosion of different applications of machine learning. It's used pretty much everywhere, for absolutely everything, there are thousands of paper coming out every year. Probably sth like 90% of papers at SIGGRAPH this year have ML in them in some form.



INTRODUCTION

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

And if you look at these papers – it all looks super easy!
You take some hard problem, pick a random neural network architecture, throw in tons of data for good measure, do a brr-brr with a blender...

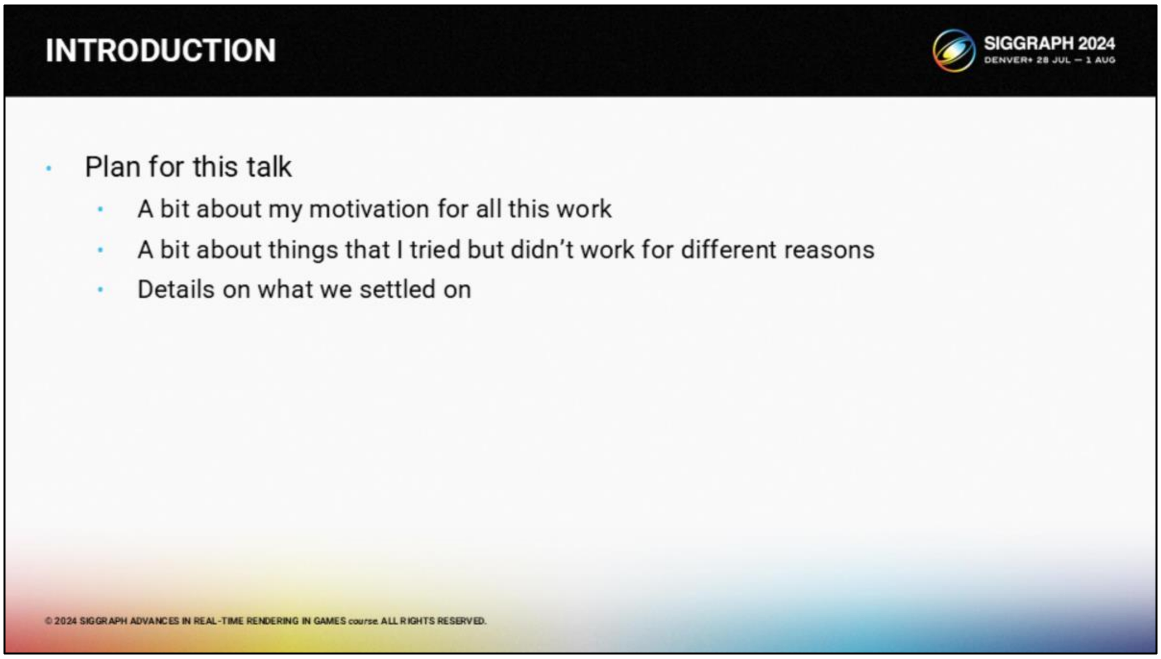


INTRODUCTION

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

… and you get a spectacular success, everything works great.

And, of course I loved the idea!
Just to give you a bit of a spoiler of the rest of the talk: it was not quite like that...
But we'll get to that



INTRODUCTION

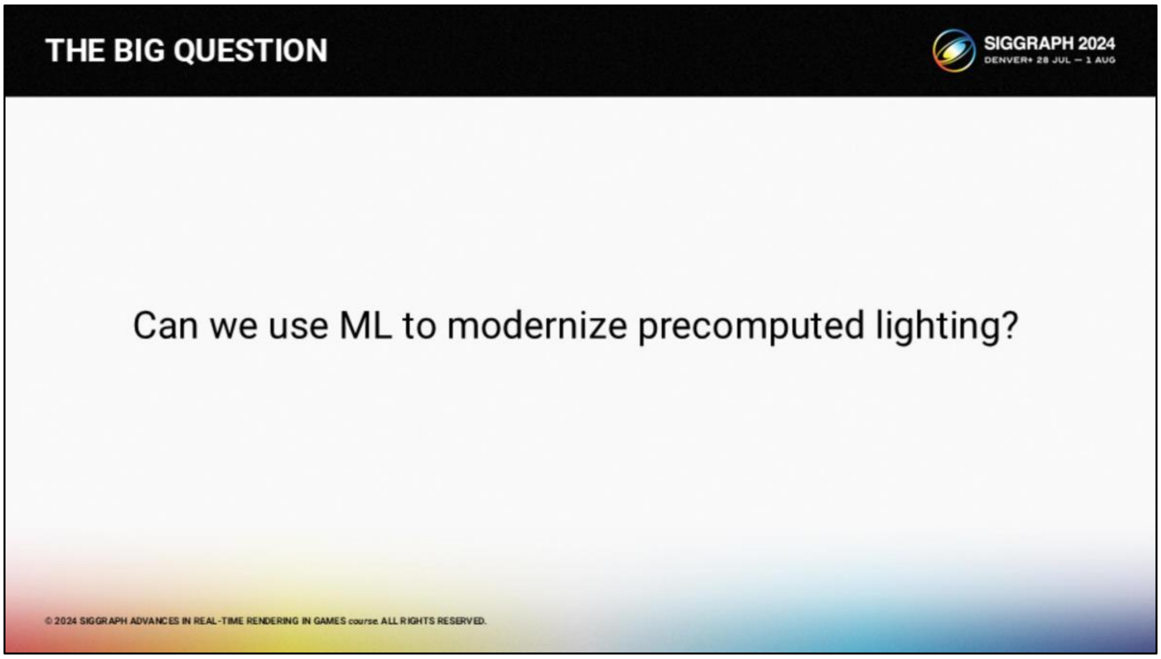
SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Plan for this talk
  - A bit about my motivation for all this work
  - A bit about things that I tried but didn't work for different reasons
  - Details on what we settled on

The plan for the talk is to cover a bit of what I wanted to achieve, go over a couple most interesting things that I tried and that didn't work, and then go over details of what I settled on and what shipped.

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

THE PLAN



THE BIG QUESTION

SIGGRAPH 2024 DENVER+ 28 JUL — 1 AUG

Can we use ML to modernize precomputed lighting?

Since I did a lot of work on precomputed lighting, I was naturally looking for some application in that area.
The big question that I asked myself was: Can we use ML to modernize precomputed lighting.

Just for some background: if you’re not familiar with Call Of Duty, it’s a first-person action game and we heavily rely on precomputed lighting.
We mainly use a combination of lightmaps, and various volumetric represenations to provide the diffuse component of indirect illumination.

Precomputed lighting can provide us really high quality at very little runtime cost, because we pay that cost upfront, during development time, when we do all the heavy calculations and store the results. At runtime we only look that information up

Copyright Activision Publishing
*Not Representative of Actual Gameplay
CALL of DUTY
WARZONE

And while there are real time global illumination solutions and they are becoming more widespread, they all come with non-trivial costs. Call of Duty titles run at 60Hz, and they run at wide range of hardware, from mobile fo high-end PCs, and these systems don't scale well to the lower-end platorms.
We're ok with the set of tradeoffs that precomputating lighting comes with, but we would like to take it to the next level – for instance get to resolutions that would take too much memory with current approaches. And we were hoping modern machine learning could give us that.

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

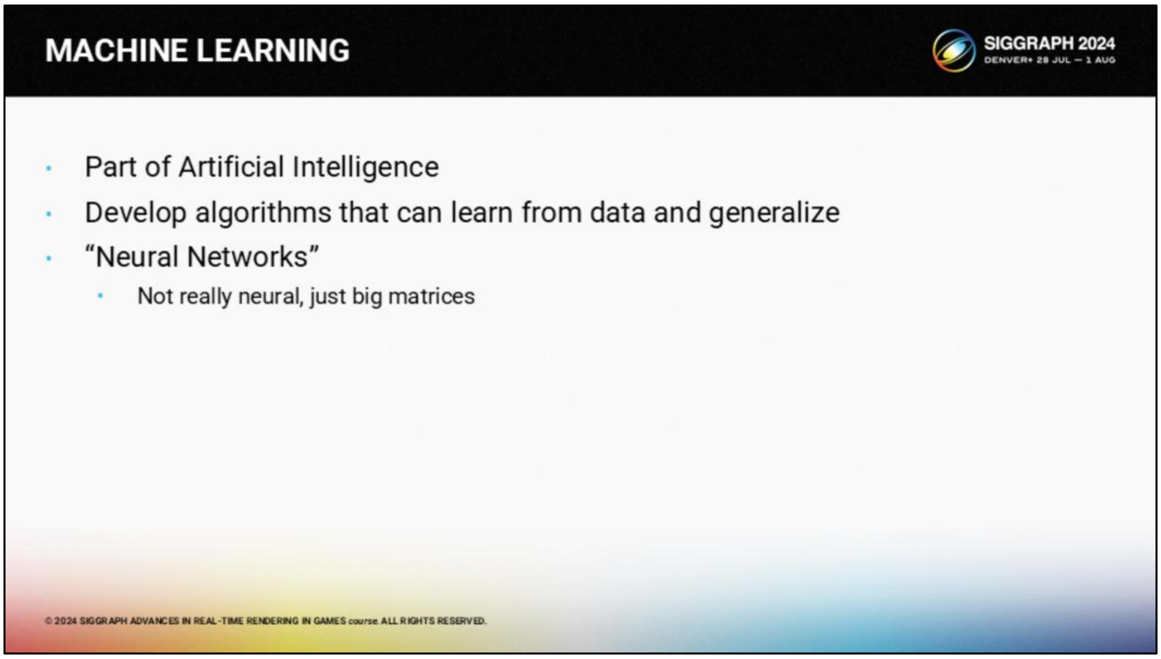
THE PATH
or the things that didn't work

I will start by talking about the approaches we tried, what worked, what didn't, what we learned from them.

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

INTERMISSION
or a quick primer on ML

Actually no, we'll do a very quick technical primer on ML first, just for some background, so we're all on the same page

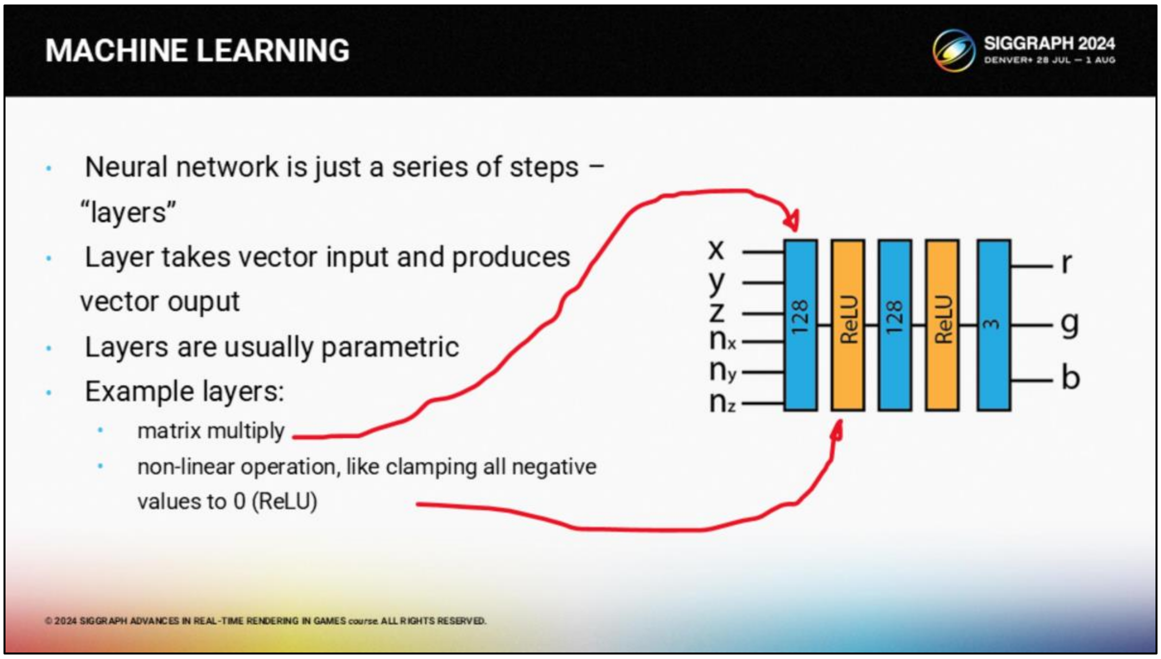


MACHINE LEARNING

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Part of Artificial Intelligence
- Develop algorithms that can learn from data and generalize
- “Neural Networks”
  - Not really neural, just big matrices

Machine learning is a field of study in artificial intelligence. It deals with developing algorithms that can learn from data and generalize to data they haven’t seen before. The most popular type of such algorithm used nowadays in ML are so called “neural networks” – which draw some inspiration from how actual neurons work. But to be honest is a bit of a stretch.

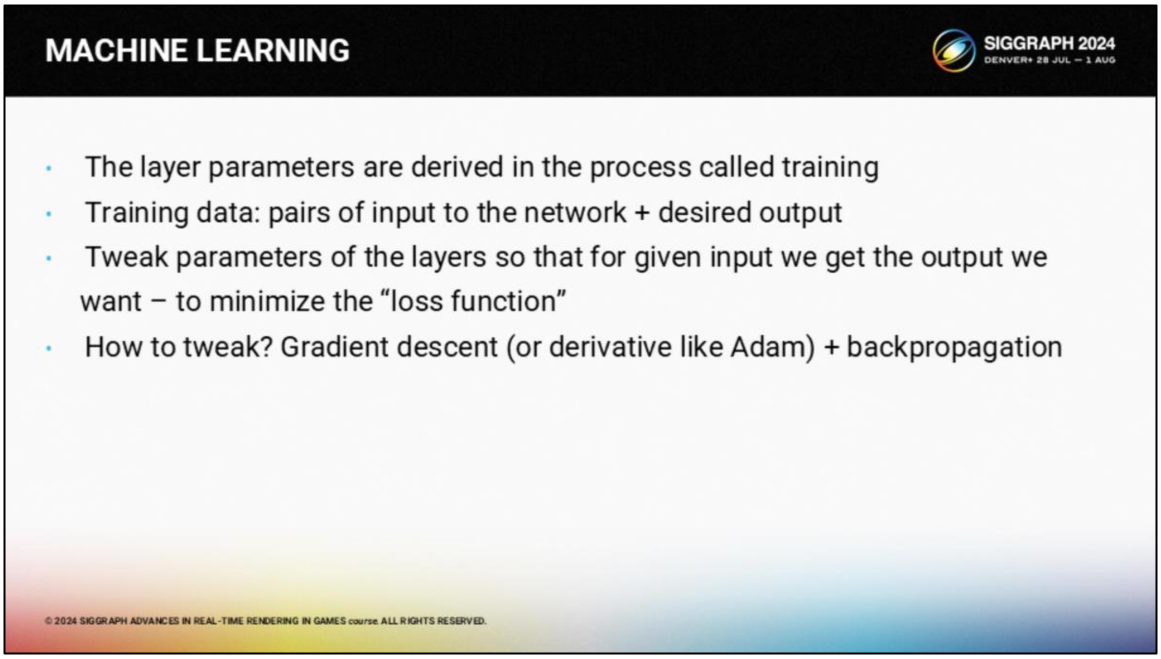


MACHINE LEARNING

- Neural network is just a series of steps – "layers"
- Layer takes vector input and produces vector ouput
- Layers are usually parametric
- Example layers:
  - matrix multiply
  - non-linear operation, like clamping all negative values to 0 (ReLU)

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG
x
y
z
nx
ny
nz
128
ReLU
128
ReLU
3
r
g
b

Under the hood, the neural network is an algorithm composed of a number of processing steps, traditionally called "layers".
Each layer takes some input – some multidimensional vector and produces some output. Usually layers are parametric, so they store some values that control the thing they do.
In the simplest form layer can be a simple matrix multiply – some vector comes in, and gets multiplied by matrix, goes out.
Another type of layer is some non-linear operation: for instance ReLU – rectified linear unit, which is just a fancy name for clamping any negative value to zero.
The structure of the network is usually visualized in these nice schematics. The input is on the left, and then subsequent layers go left to rigth. I will draw layers that do matrix multiples as blue boxes, with the number showing the dimensionality of its output (the input size is implicit from previous layer). Yellow boxes will represent some non-linear operation – in the example on the slide that clamping.
So the network here takes vector with 6 elements, multiplies is by 6x128 matrix, applies the clamping, multiplies by next matrix, does next clamping, and then the next matrix, producing 3 output.
And sometimes such combination of a dot product (=one row of matrix-vector multiply) and non-linear operation is referred to as „neuron"

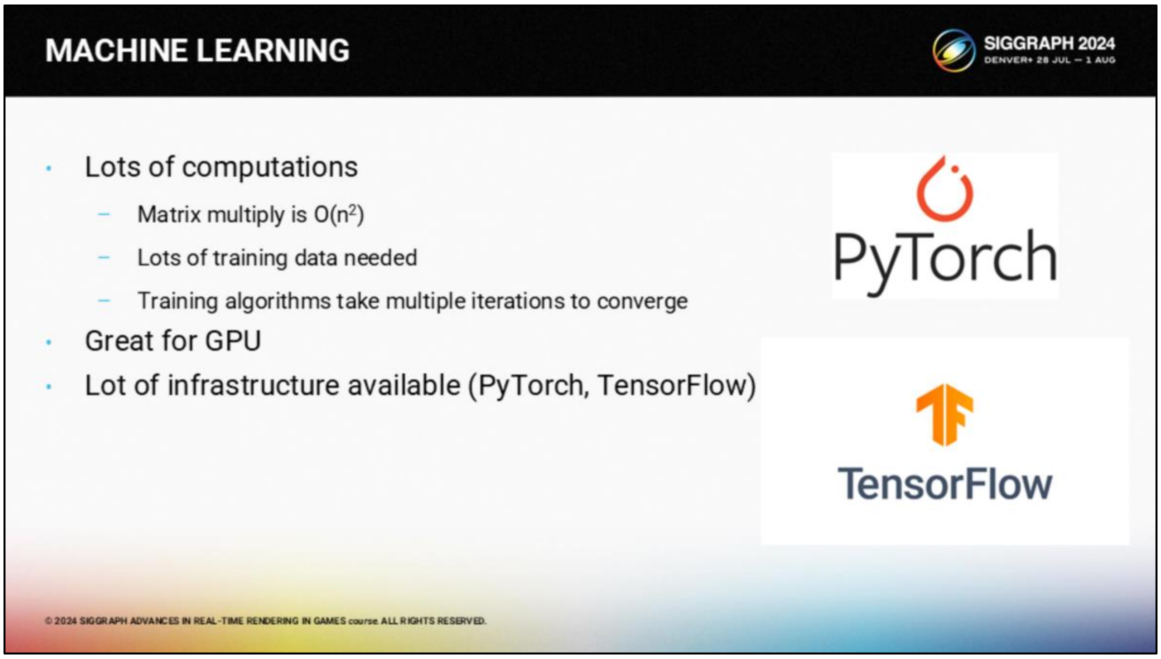


MACHINE LEARNING

SIGGRAPH 2024 DENVER+ 28 JUL – 1 AUG

- The layer parameters are derived in the process called training
- Training data: pairs of input to the network + desired output
- Tweak parameters of the layers so that for given input we get the output we want – to minimize the "loss function"
- How to tweak? Gradient descent (or derivative like Adam) + backpropagation

Importantly, the layer parameters, for instance values in these matrices, derived from the data, in a process called training.
In the simplest case again, we gather a lot of „training data" – pairs of inputs to the network together with the desired outputs.
Then we gradually tweak the parameters of the layers so that the output the network produces for a given input is closer and closer to what we want it to be.
And how close it is is described by „loss function" – for instance a sum of sqaured difference between components – so called L2 loss.
The algorithms that figure out how exactly tweak these parameters are usually some derivative of gradient descent – like Adam. And they need the gradient of the loss function which is produced by „backpropagation" which propagates the derivative of the loss, from the very end to the earlier layers.



MACHINE LEARNING

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Lots of computations
  - Matrix multiply is O(n^2)
  - Lots of training data needed
  - Training algorithms take multiple iterations to converge
- Great for GPU
- Lot of infrastructure available (PyTorch, TensorFlow)

PyTorch

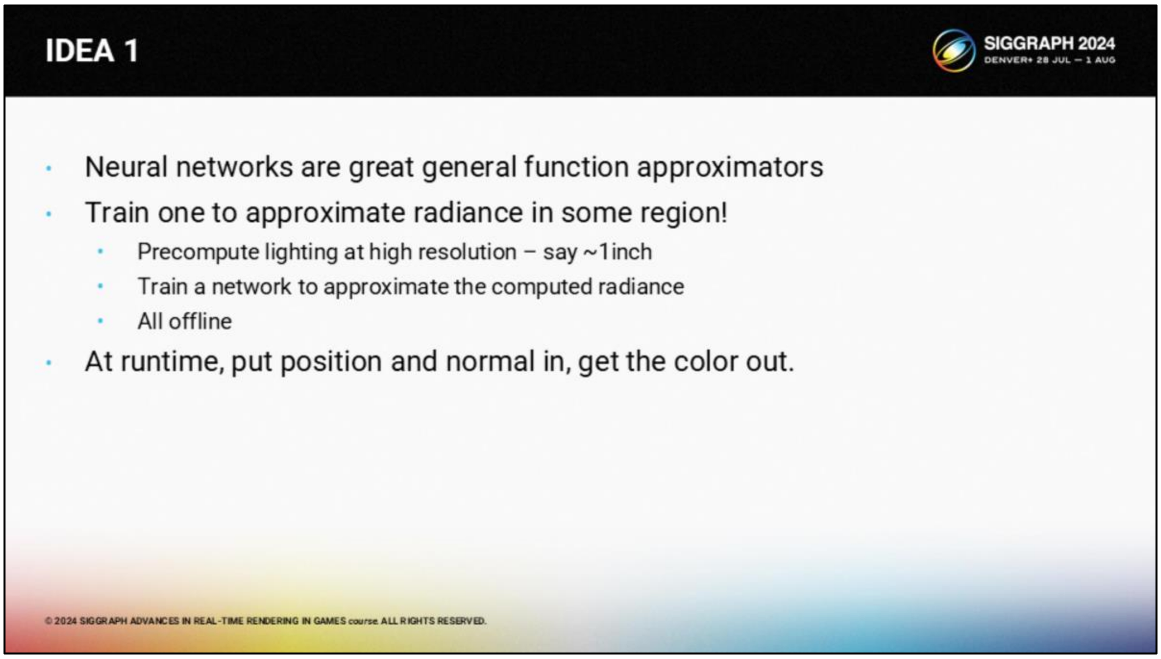
TensorFlow

All this requires tons of computations in general,. The bigger the model is, the more calculations of course, but keep in mind that it grows really quickly.
Matrix multiplies have O(n^2) complexity, so when you increase the layer by the factor of 2, you increase the computation amount by a factor of 4.
Training itself is an iterative process and requires multiple passes to converge.
Because this high computational cost, GPUs are usually used to do the training and evaluation efficiently.
There’s a lot of infrastructure available to help with all this – mainly in a form of Python libraries like PyTorch or Tensorflow.

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

THE PATH
or the things that didn't work

Ok, back to the main topic

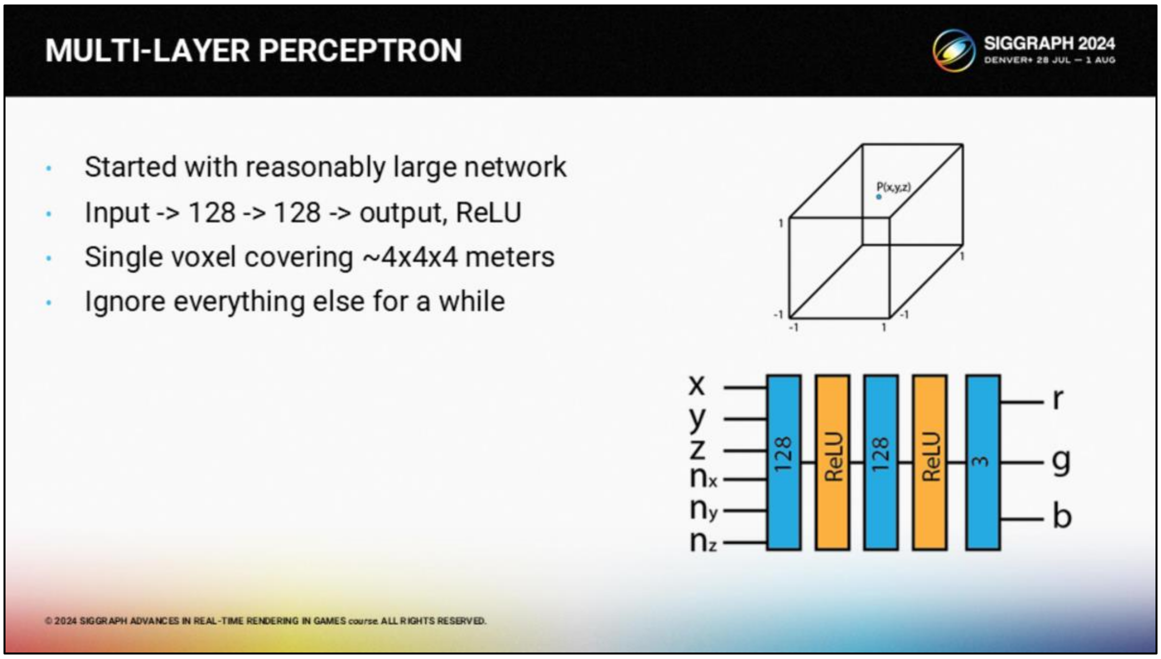


IDEA 1

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Neural networks are great general function approximators
- Train one to approximate radiance in some region!
  - Precompute lighting at high resolution – say ~1inch
  - Train a network to approximate the computed radiance
  - All offline
- At runtime, put position and normal in, get the color out.

Ok, lets put all this to some use!
Neural networks are great function approximators, and they can do it really efficiently – even relatively small networks can approximate complex, multidimensional functions with reasonable accuracy.
So we can try the simplest thing first – use the network to approximate the radiance/ the lighting distribution in some region. You can think of radiance distribution as a function – for every position – you get some radiance.
The process would looks something like that:
We would precompute lighting within that region at some really high resolution, say an inch density – so sth totally impractical to store directly.
Next we would train a network that given a position on the input, would give us back that precomputed lighting on the output
This training would happen offline, and at runtime, when shading the pixel, we put the actual position into the network and get that lighitng back.



MULTI-LAYER PERCEPTRON

SIGGRAPH 2024 DENVER+ 28 JUL – 1 AUG

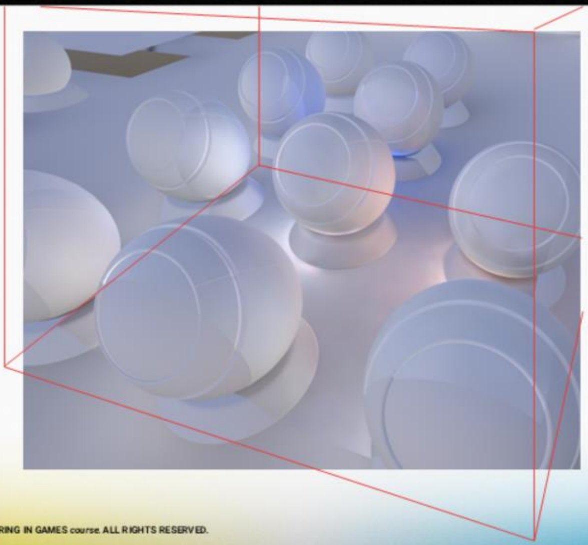
- Started with reasonably large network
- Input -> 128 -> 128 -> output, ReLU
- Single voxel covering ~4x4x4 meters
- Ignore everything else for a while

P(x,y,z)
1
1
-1
-1
-1
1
x
y
z
nx
ny
nz
128
ReLU
128
ReLU
3
r
g
b

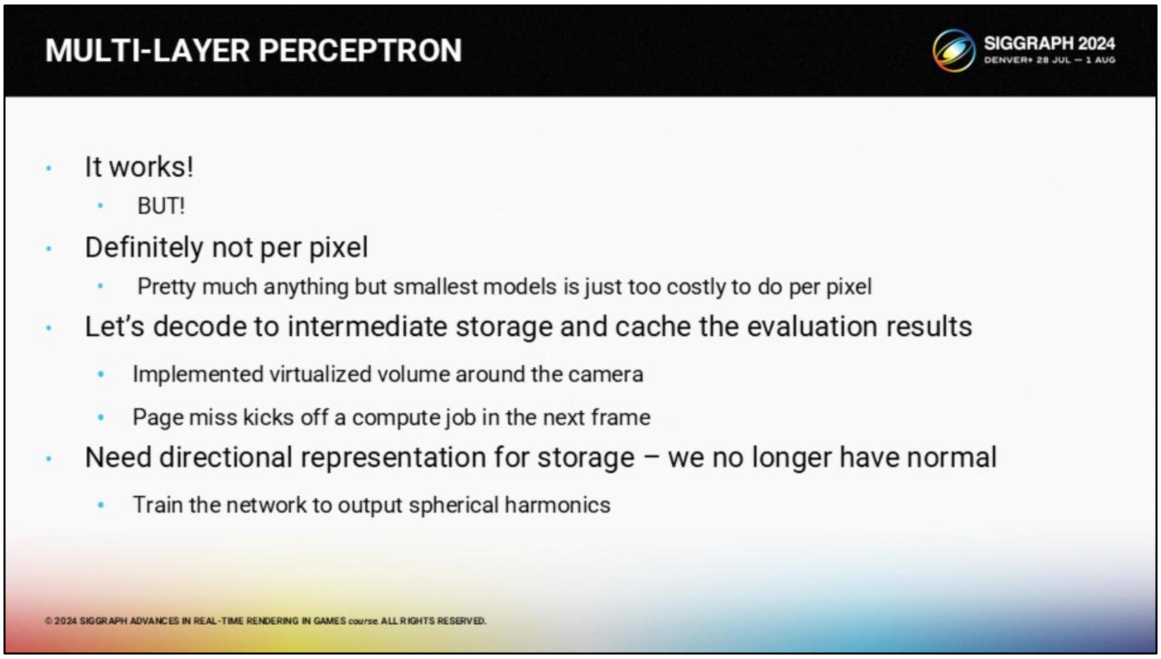
The simplest network you can use for sth like this is a multi-layer perceptron.
It’s just a series of matrix multiplies, interleaved with some nonlinear operations. The input gets multiplied by the first matrix, then gets processed by that non-linear operation and so on.
This is actually the same network that i showed as an example earlier – so it takes 6 values in, does matrix multiply, then clamp to 0, matrix mulityply, another clamp, another matrix multiply and so on. At the output we get 3 values – rgb color.
To start, our network will reconstruct the lighting only within some small region of space, a voxel. As the input we will pass the xyz position, normalized to -1 to 1 range and a direction in which we want to query the lighitng.
For todays standards this is a tiny model. And even like this, that middle section is 128x128 matrix multiply that’s there is already 16k multiplies – so you might already see some problems.

MULTI-LAYER PERCEPTRON

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG



And this is the result.
Just so you know what you're looking at: this is our test scene, it's just the secondary lighting, the scene is generally lightmapped, except that square section in the middle of the image which
Is this voxel from the previous slide, these section uses the neural network that was trained offline. This this very high-resolution precomputed lighting "baked" into the coefficients of these matrices from the previous slide.
And tbh, looks pretty good! We get this really detailed lighting, no real artifacts.
Definitely good enough to keep working on it!

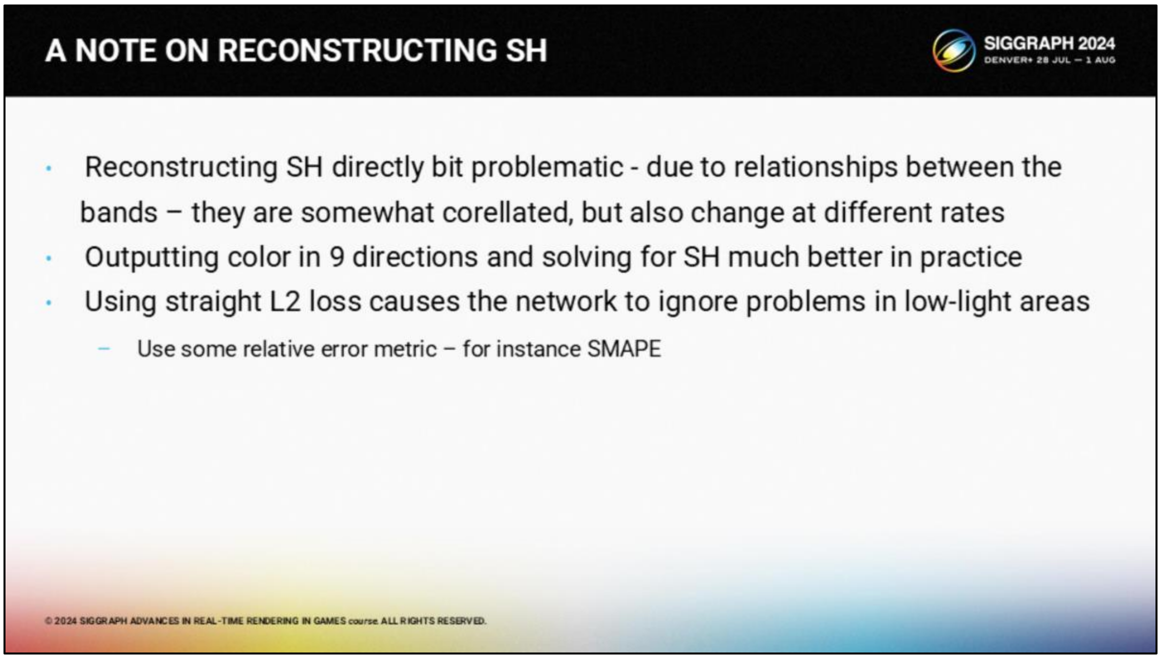


MULTI-LAYER PERCEPTRON

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- It works!
  - BUT!
- Definitely not per pixel
  - Pretty much anything but smallest models is just too costly to do per pixel
- Let's decode to intermediate storage and cache the evaluation results
  - Implemented virtualized volume around the camera
  - Page miss kicks off a compute job in the next frame
- Need directional representation for storage – we no longer have normal
  - Train the network to output spherical harmonics

First problem is of course obvious, there's no way we can do it per pixel. I couldn't dig out the actual numbers for that particular test, but on PS4 where I was testing this it was lots of milliseconds per frame something completely impractical.
So even though we used really small network for modern standards, it's unusable if you try to evaluate it per-pixel on anything but the latest high-end hardware, so we need to try to work around these limitations.
Ok, if not per pixel, maybe we could do a two-stage process – first evaluate the network to some intermediate representation, at a resolution somewhat lower than per-pixel, and then look that up per-pixel. This would let us to share some of the evaluation costs between the pixels.
We implemented this as virtualized volume, it was composed of 4x4x4 texel pages. If during shading a non-existing page was accessed, it would kick off a compute job to evaluate the network for that page in the next frame.
For this we cannot output just RGB any more, because we don't know the surface normal when evaluating the network. But that's easy – since we're working with diffuse lighting, we can train the network to output spherical harmonics.



A NOTE ON RECONSTRUCTING SH

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Reconstructing SH directly bit problematic - due to relationships between the bands – they are somewhat corellated, but also change at different rates
- Outputting color in 9 directions and solving for SH much better in practice
- Using straight L2 loss causes the network to ignore problems in low-light areas
  - Use some relative error metric – for instance SMAPE

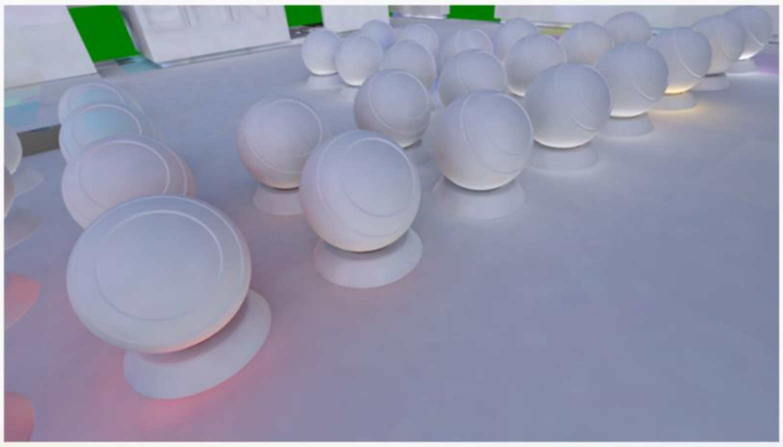
There's a small caveat here, outputting just straight SH can generate subpar results, because different coefficients have different ranges, so if you ever want to play with sth similar, I found that thing that works best is just outputting a diffuse RGB in 9 different direction, and converting this to SH – it's a bijection, so 1:1 mapping, and you do it with a matrix multiply.
.

MULTI-LAYER PERCEPTRON

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

Here are some screenshots from these prototypes. The scene is divided into voxels and we did that baking and training process for all the voxels, and rendered using that virtualized volume.
As you can see it generally looked pretty good and worked pretty well!
We could decide decide how many bricks to fill every frame, so we can do perf budgeting. Generating 128 bricks a frame was enough to avoid popping, and would take aroung 1ms on ps4. The per-pixel lookup was now super-cheap, as it was just a couple of texture lookups and an SH eval.

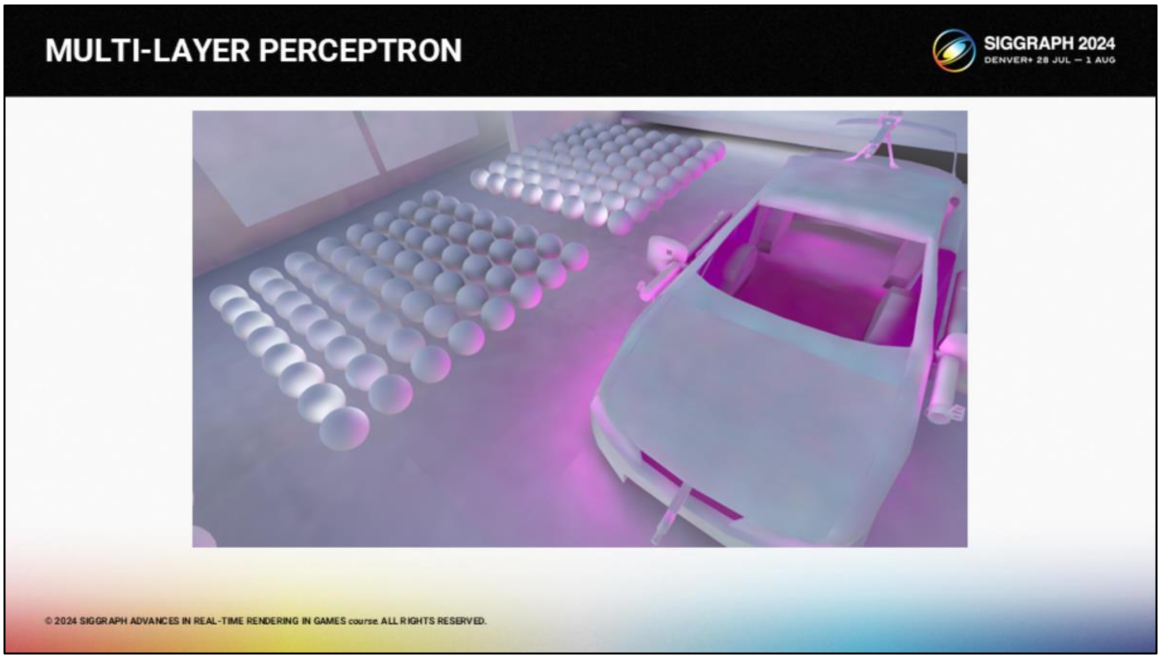
MULTI-LAYER PERCEPTRON



Of course there were some small problems here and there:

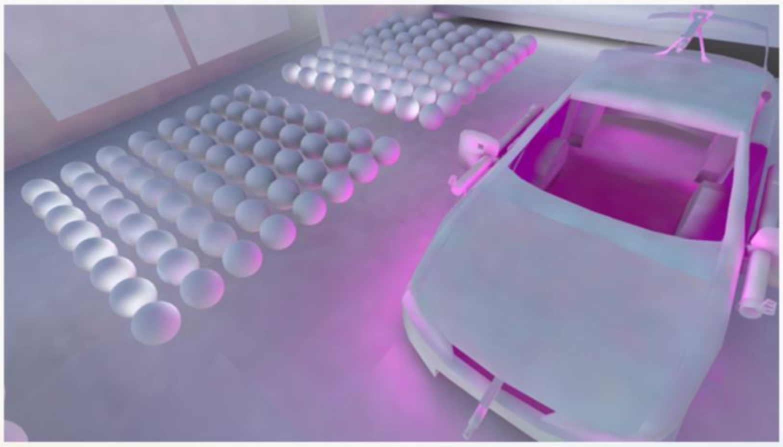
Because of how the scene was divided into voxels there were some discontinuities on the boundaries, which you can see here and there on these images.

Then, things in the distance would generate too many bricks, so we dealt with this by introducing cascaded volume. This in turn exposed some problems with light leaking through the geometry when voxels became too large in the distance and also generated some discontinuities between the cascades.

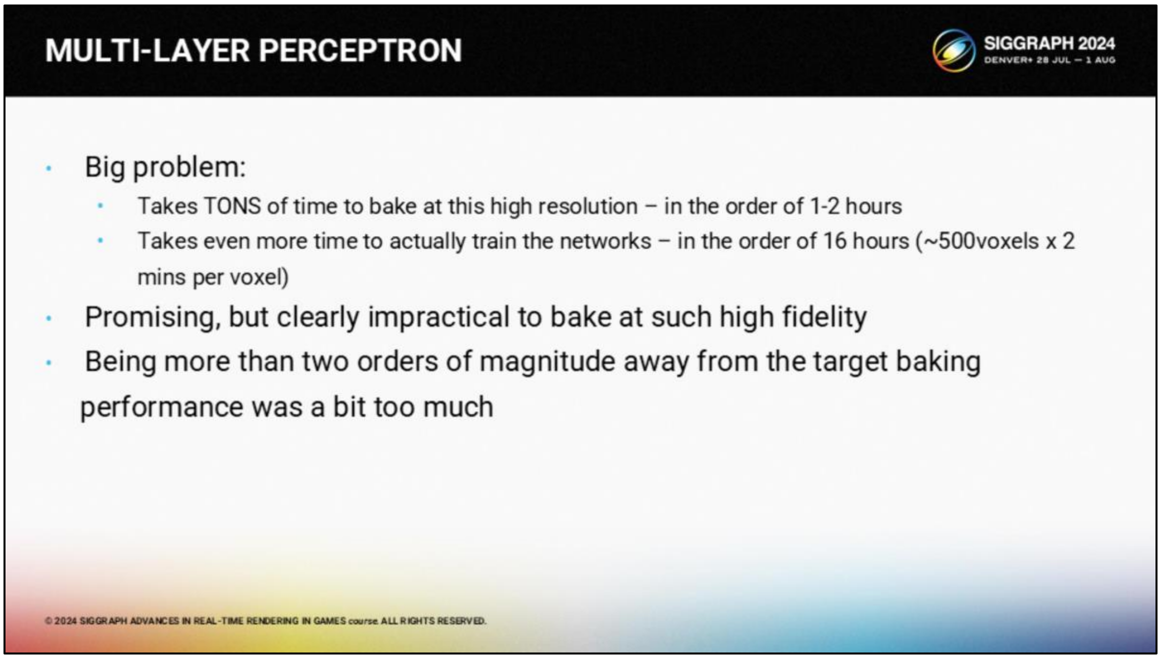


MULTI-LAYER PERCEPTRON

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG



There’s also quite a bit bit of a noise from not fully converged training, there is some noise from the bake but generally nothing that couldnt be be solved with some good, old fashioned engineering.
So we have this nice, high resolution lighting, everything runs at 60hz, even on a 12 year old hardware – where’s the catch?



MULTI-LAYER PERCEPTRON

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

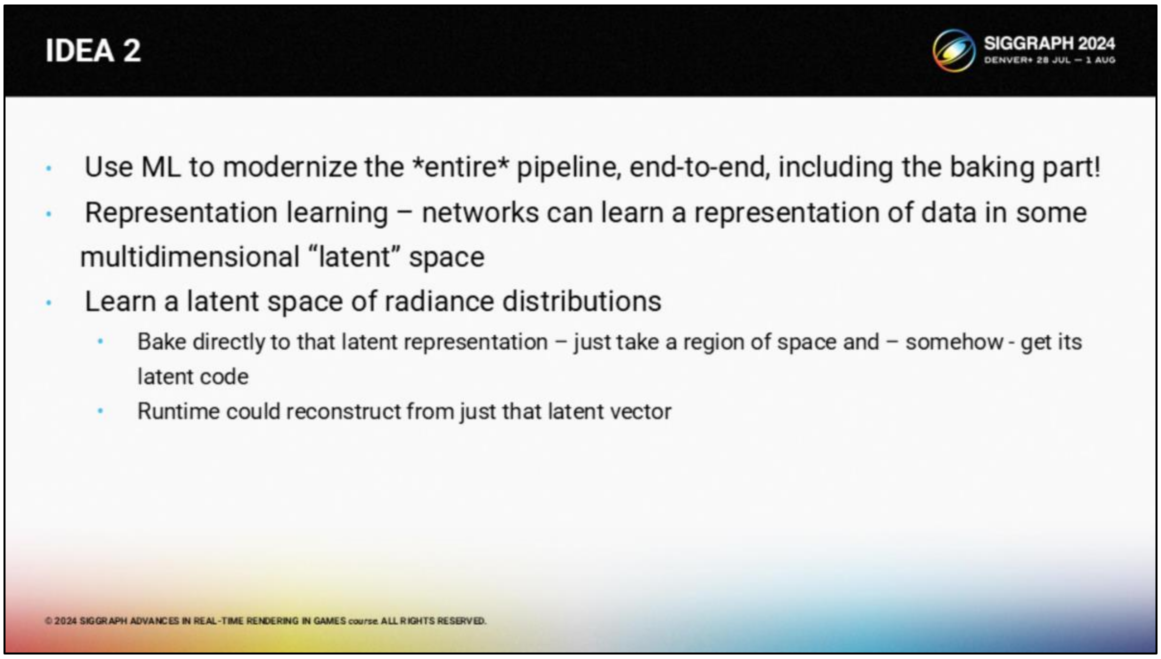
- Big problem:
  - Takes TONS of time to bake at this high resolution – in the order of 1-2 hours
  - Takes even more time to actually train the networks – in the order of 16 hours (~500voxels x 2 mins per voxel)
- Promising, but clearly impractical to bake at such high fidelity
- Being more than two orders of magnitude away from the target baking performance was a bit too much

Well the catch is that we went a bit too crazy with the amount of precomputations.

First there's the baking part. We have a really high performance baking tool, but to bake the lighting at this 1 inch resolution for that small test scene took between 1 and 2 hours, on my 32 core i9 (we bake on the CPU)

What's even worse: it took around 16h to train all these networks – on a pretty decent Titan RTX - and it's just a tiny test scene, that normally takes like 20 seconds to bake lighting.
The scene is really small - there is around 500 voxels there, and each voxel take only around 2 minutes to train – but it just all very quickly adds up.

So even tough it looked really promising, it quickly became clear, that there's no way we can afford bake times like this in production. I'm sure things could be massively optimized in that pipeline, but even it we made it an order of magnitude faster, it would still be an order or magnitude, if not more, too slow
It was clearly not the way.



IDEA 2

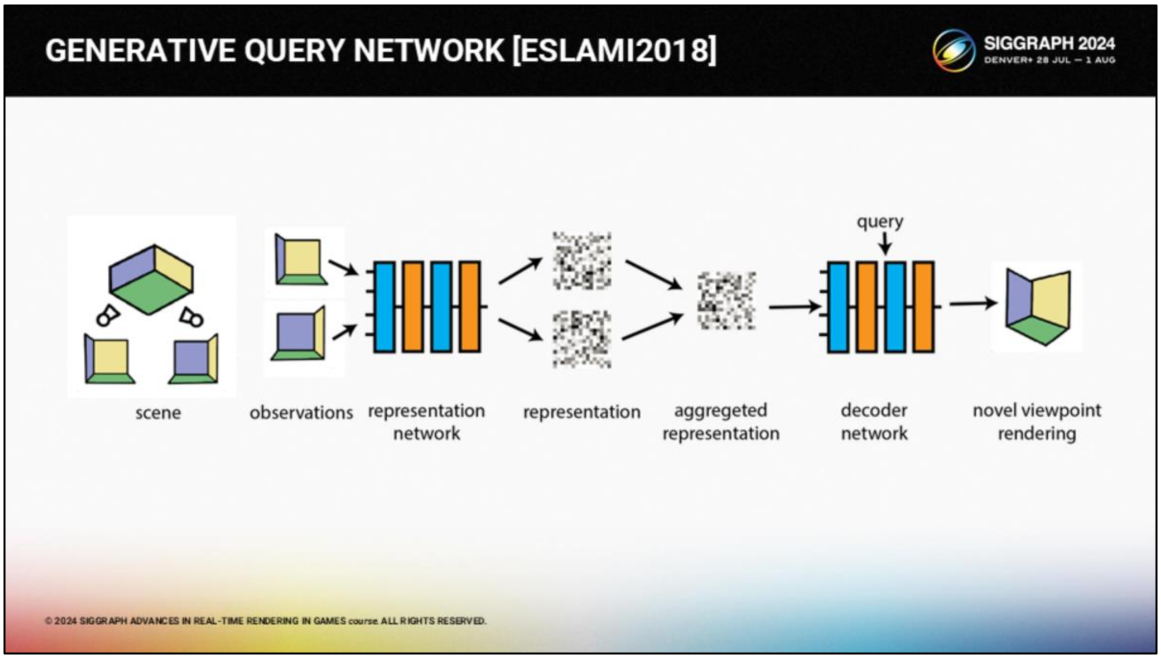
SIGGRAPH 2024 DENVER+ 28 JUL – 1 AUG

- Use ML to modernize the *entire* pipeline, end-to-end, including the baking part!
- Representation learning – networks can learn a representation of data in some multidimensional "latent" space
- Learn a latent space of radiance distributions
  - Bake directly to that latent representation – just take a region of space and – somehow - get its latent code
  - Runtime could reconstruct from just that latent vector

So if the baking time is big part of the problem, maybe we could modernize the entire pipeline, end-to-end, including the baking, with ML?

In machine learning, there is a concept called representation learning – which means that the data is transformed and represented in some multidimentional so-called „latent" space. The points in this latent space have some intrinsic meaning to the system, for instance similar objects – for however you define similar – end up being close in the latent space.

So grand idea number 2: learn a latent space of radiance distributions – so we can represent the lighting in some area just with a single, multidimensional vector. We have all the super nicely lit past Call Of Duty maps, so tons of data to derive such space. And once we have that, instead of baking the lighting within a voxel, training the network etc like before – we would just need to efficiently find the latent represenation of a voxel.
The runtime could take that latent vector and generate the high resolustion radiance at any point from that.



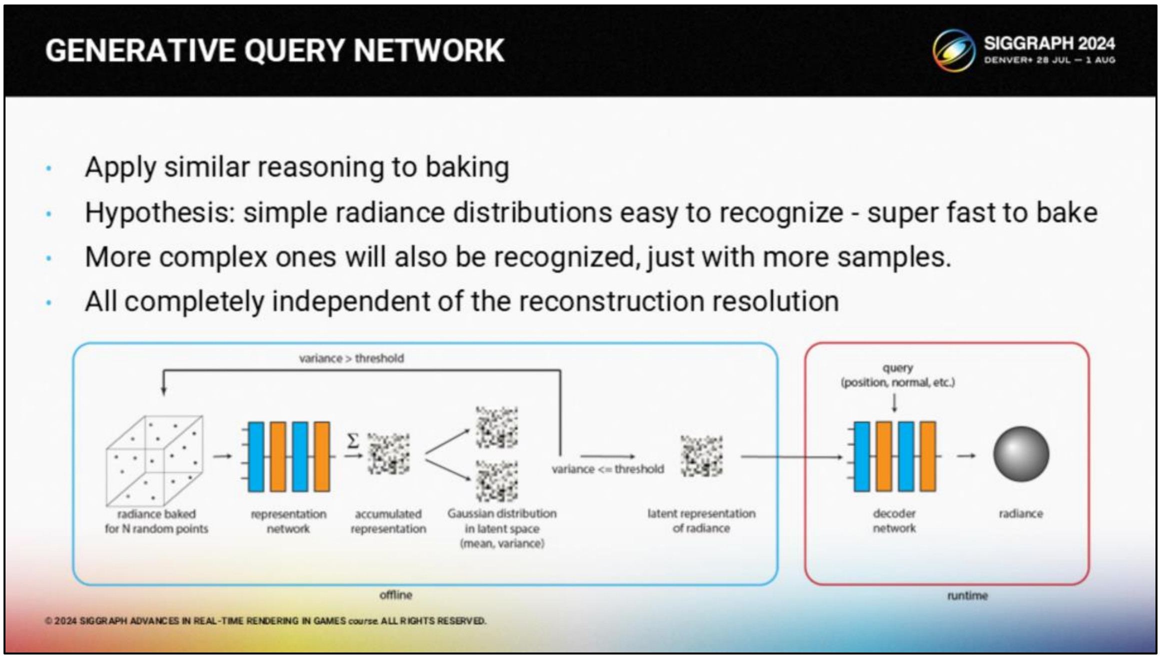
GENERATIVE QUERY NETWORK [ESLAMI2018]

In 2018 there was this great paper from DeepMind, “Neural scene representation and rendering”, which introduced Generative Query Networks.

The general idea was that the system was given a scene and some number of observations of that scene – just renderings from some number of viepoints. These observations were passed through a network 1 – representation network to form get their latent representation – one for each observation.

Then these were combined – just added in practice – to form an aggregated representation. And from that, network number two - decoder network could perform novel view synthesis, render from new positions

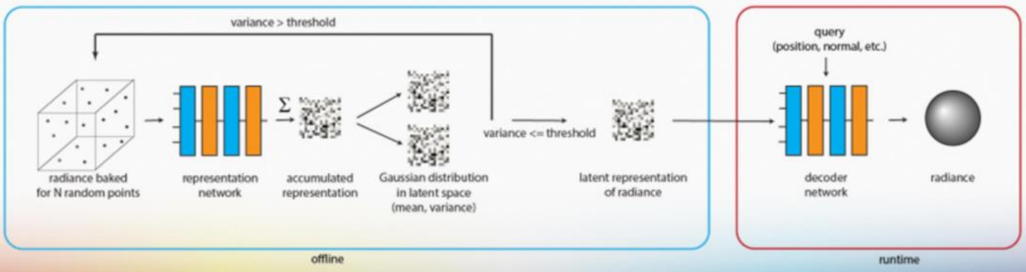
The great thing was that the aggregated representation was probabilistic, it was a Gaussian distribution in the latent space. So the system had a notion of how certain it was about the structure of the scene – if the provided observations were NOT enough to accurately reason about the scene, the variance of that distribution would be large. But additional observations could be provided and accumulated with the earlier ones to get a representation with smaller variance – something that could be done iteratively.



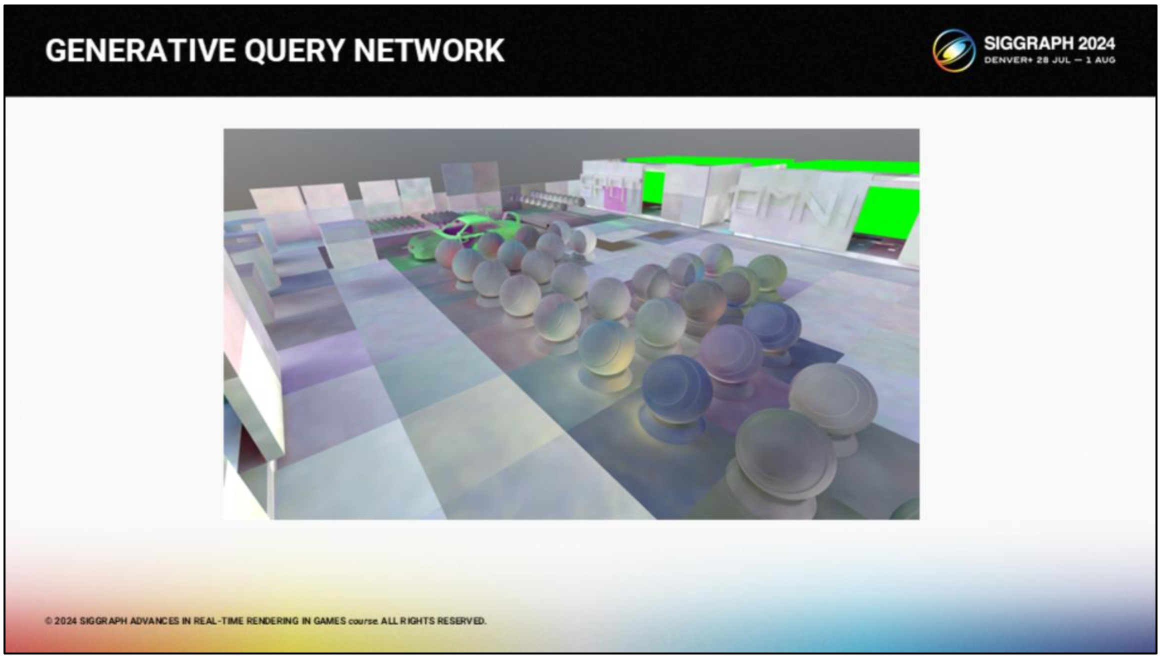
GENERATIVE QUERY NETWORK

SIGGRAPH 2024 DENVER+ 28 JUL – 1 AUG

- Apply similar reasoning to baking
- Hypothesis: simple radiance distributions easy to recognize - super fast to bake
- More complex ones will also be recognized, just with more samples.
- All completely independent of the reconstruction resolution

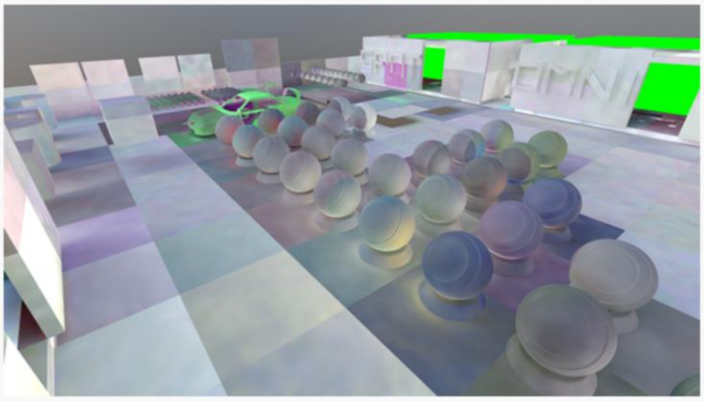

variance > threshold
radiance baked for N random points
representation network
Σ
accumulated representation
Gaussian distribution in latent space (mean, variance)
variance <= threshold
latent representation of radiance
offline
query (position, normal, etc.)
decoder network
radiance
runtime

This general scheme was something that could be potentially used to bake high-resolution lighting, in an iterative way:
say we had the network trained to produce a latent representation of the lightgin from some number of points – like on the left of the diagram – we would bake the lighting in those points and the network gives us a latent representation The representation would be a gaussian distribution, so mean and variance.
 If these points were enough to reason about the radiance distribution, the resulting variance would be small, and we would be done. But if the variance was large, we could just bake more points, go through the diagam again -provide more observations and get a tighter distribution
Once we're happy, we save the representation, just the mean of the Gaussian, and at runtime would use it together with a general decoder network to go from latent representation to radiance at any point.
So the whole network for the voxel from the previous system is now getting replaced with a *single* latent description of that voxel and a decoder network that is shared between all the voxels.

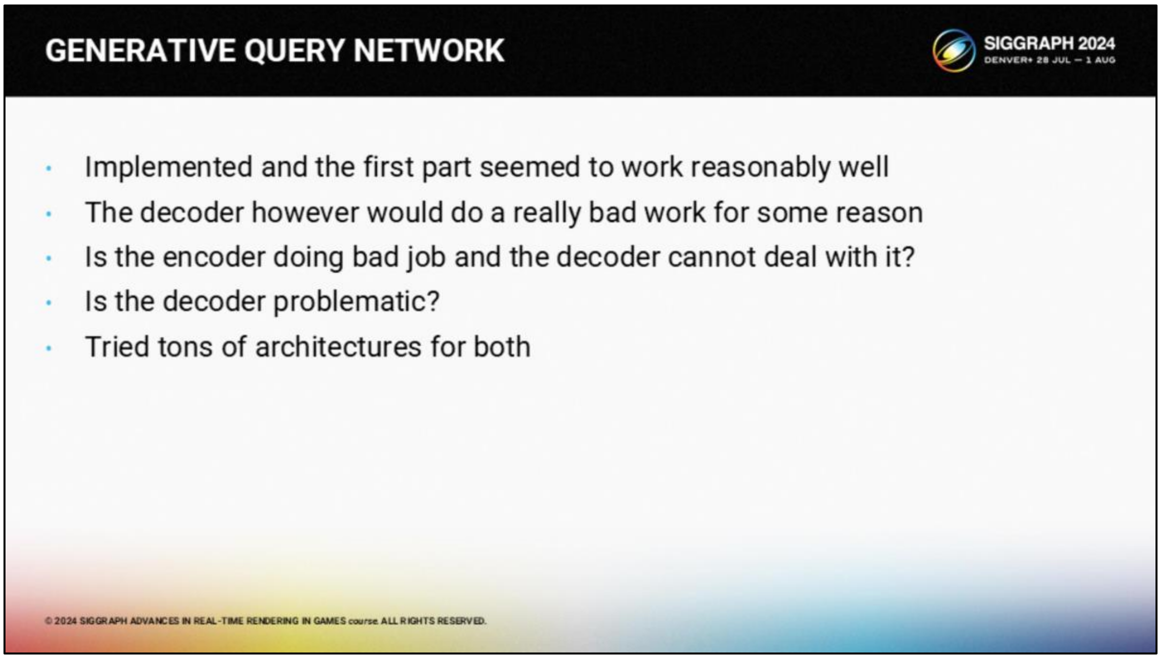


GENERATIVE QUERY NETWORK

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG



We implemented all this, but this is what it looked like… hmm… not quite the result I was hoping for.
I mean there are areas that kind-of-maybe look somewhat along the lines of doing something reasonable, but generally no, it’s a big mess

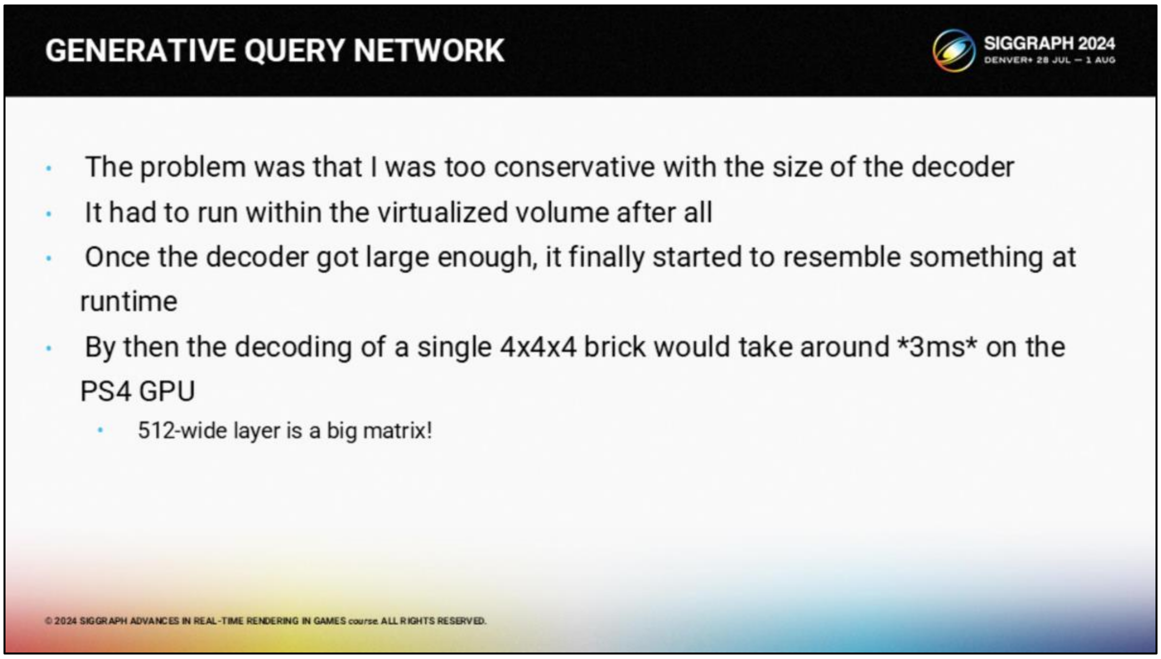


GENERATIVE QUERY NETWORK

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Implemented and the first part seemed to work reasonably well
- The decoder however would do a really bad work for some reason
- Is the encoder doing bad job and the decoder cannot deal with it?
- Is the decoder problematic?
- Tried tons of architectures for both

When investigating the insides of the system the encoder part was working reasonably well and behaving as expected – for simpler lighting distributions – like freespace - it was enough to just bake a single set of points to get representation with low variance. For areas with complex geometry, it needed more samples.
But then the decoder just couldn't generate anything reasonable out of it.
So is the encoder not doing the right thing? Maybe the decoder should use different architecture? Maybe we need more data? Maybe it's something else entirely? That's the kind of questions you ask yourself, and the problem is that it's tricky to know for sure

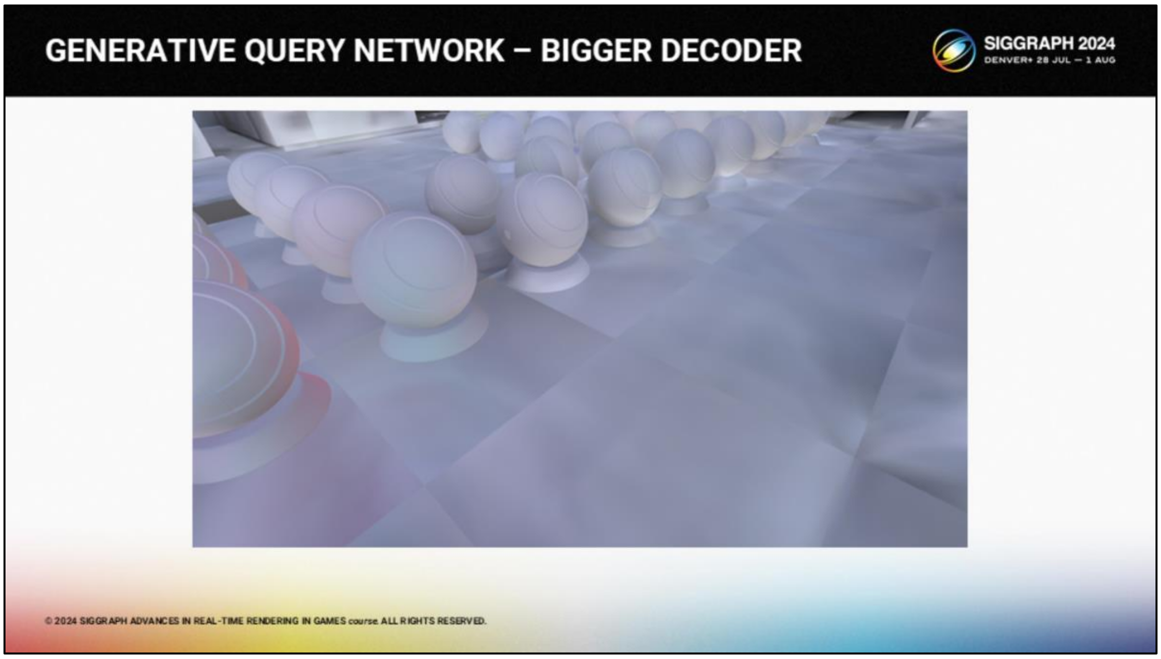


GENERATIVE QUERY NETWORK

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

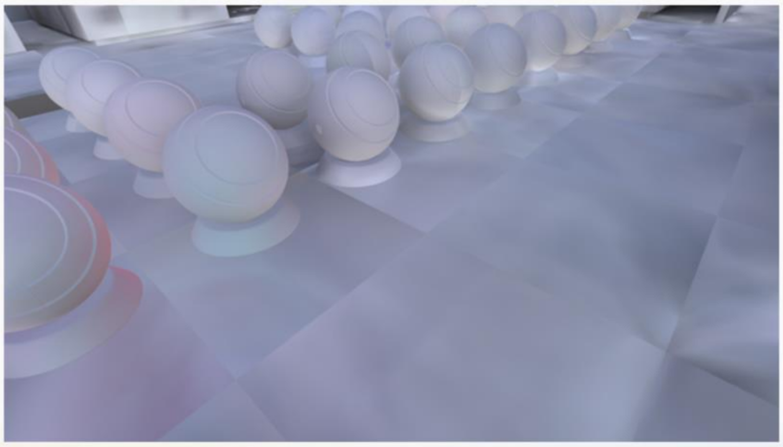
- The problem was that I was too conservative with the size of the decoder
- It had to run within the virtualized volume after all
- Once the decoder got large enough, it finally started to resemble something at runtime
- By then the decoding of a single 4x4x4 brick would take around *3ms* on the PS4 GPU
  - 512-wide layer is a big matrix!

So what was the problem?
Well, I was just way too conservative with the size of everything – in the end, I still wanted it to work in real-time, so I was keeping the decoder size pretty moderate – with 128-wider layers, or when I was feeling brave I would go to 192- or 256-wide layers..
It turned out that to decode anything even remotely reasonable we need much much more expressive power in the decoder. It had to be larger, trained on more data and generally more flexible.
However even a still fairly simple architecture with 6 layers, 512-wide would take 3 milliseconds to decode a single 4x4x4 brick of our virtualized volume, decoding 128 that was needed for visual stability would be over 300ms. Ups.

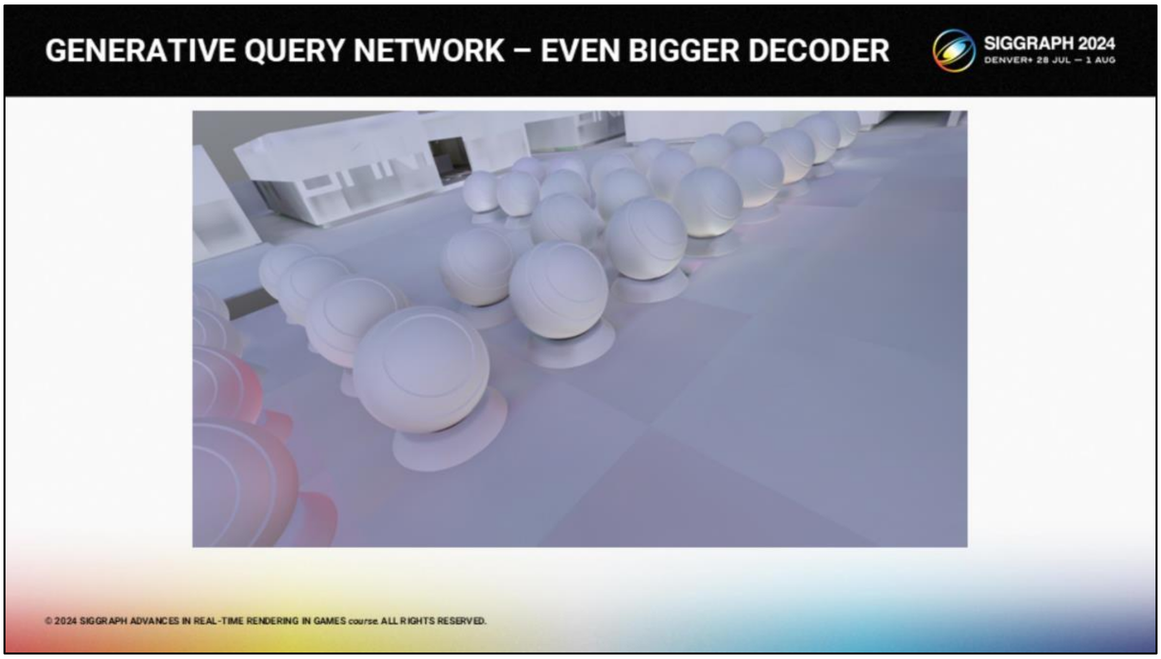


GENERATIVE QUERY NETWORK – BIGGER DECODER

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

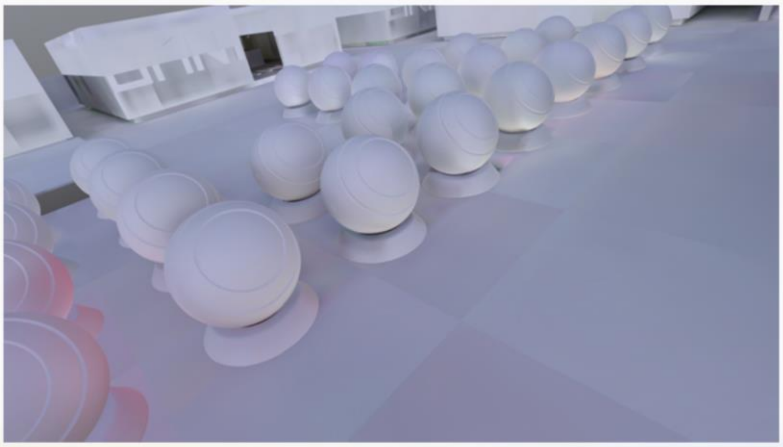


And it didn’t even look good - this is what it looked like with 6 layers 512 wide – it *barely* starts to resemble anything useful

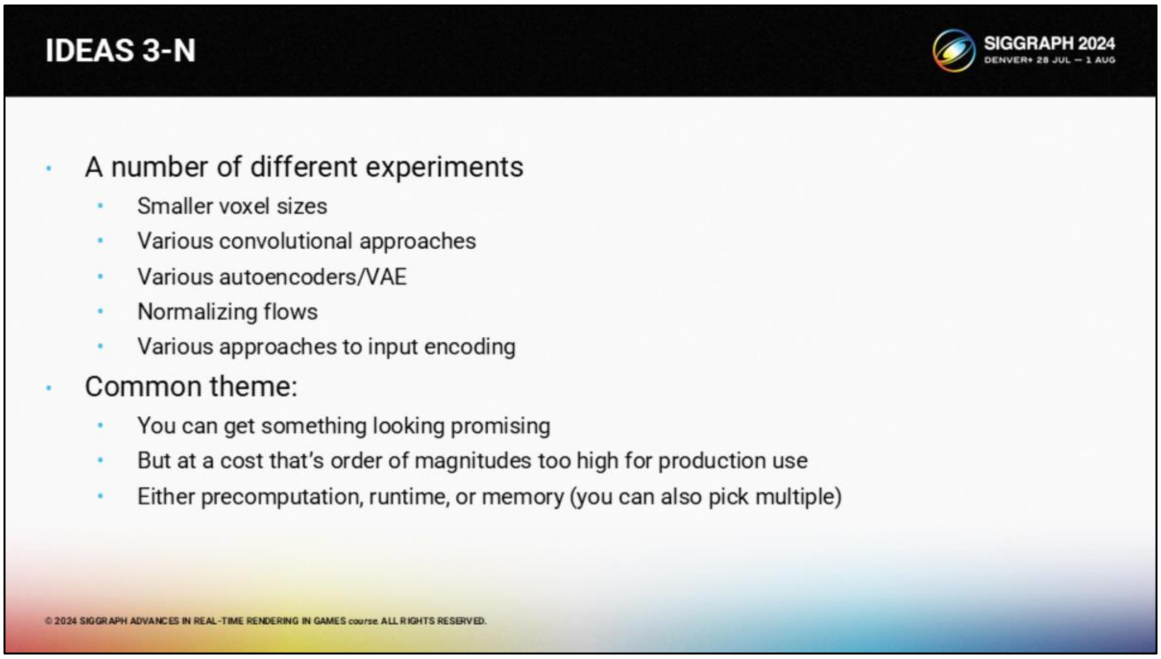


GENERATIVE QUERY NETWORK – EVEN BIGGER DECODER

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG



This is even larger decoder, i think 768 or 1024 wide layers. Is does look slightly better, but is still far far from the quality that we're after, even though it takes around 10ms to decode 4x4x4 brick

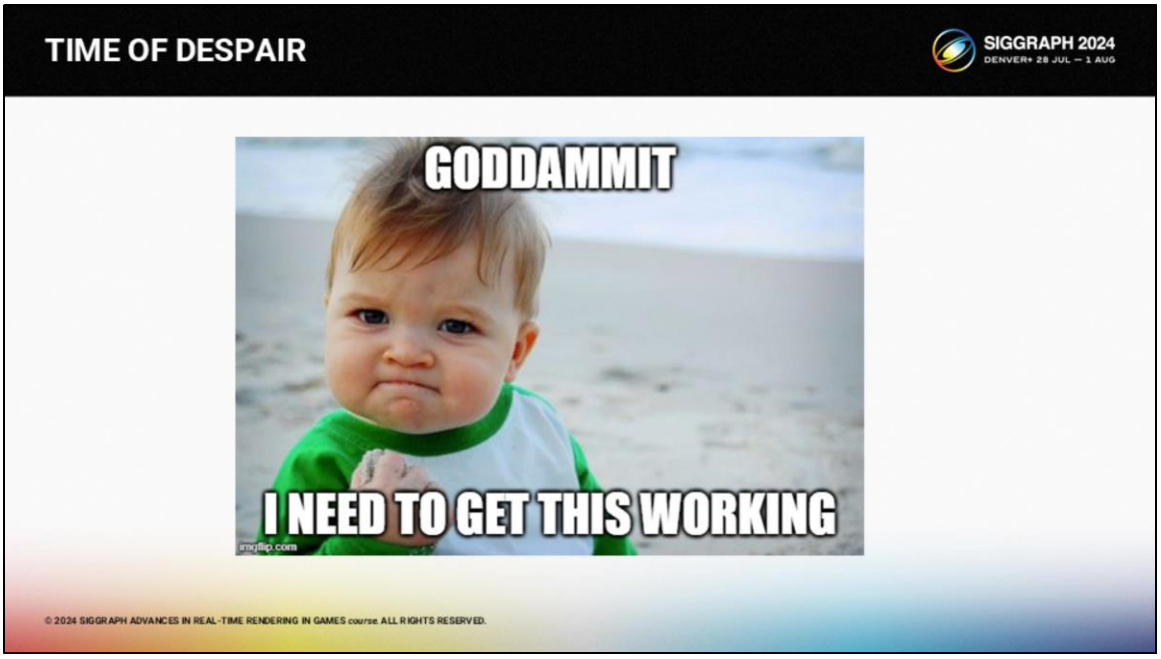


IDEAS 3-N

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

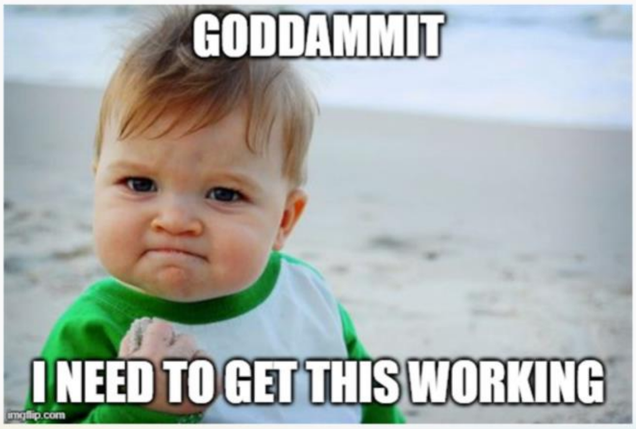
- A number of different experiments
  - Smaller voxel sizes
  - Various convolutional approaches
  - Various autoencoders/VAE
  - Normalizing flows
  - Various approaches to input encoding
- Common theme:
  - You can get something looking promising
  - But at a cost that's order of magnitudes too high for production use
  - Either precomputation, runtime, or memory (you can also pick multiple)

I spent way too much time trying to figure out some better solution. I experimented with smaller voxels, which were somewhat behaving better, but at a cost of much higher memory consumption and problems with aliasing in the distance. I tried tons of different architectures for the decoder, convolutional approaches, LSTM, transformers, normalizing flows, I analyzed how activation layers or input encoding affects the result – but generally the common theme was: if anything looks even remotely promising, it's already way too costly either on the precomputation side, on runtime side or memory wise – or sometimes all three. So at some point, I put this idea on the backburner.

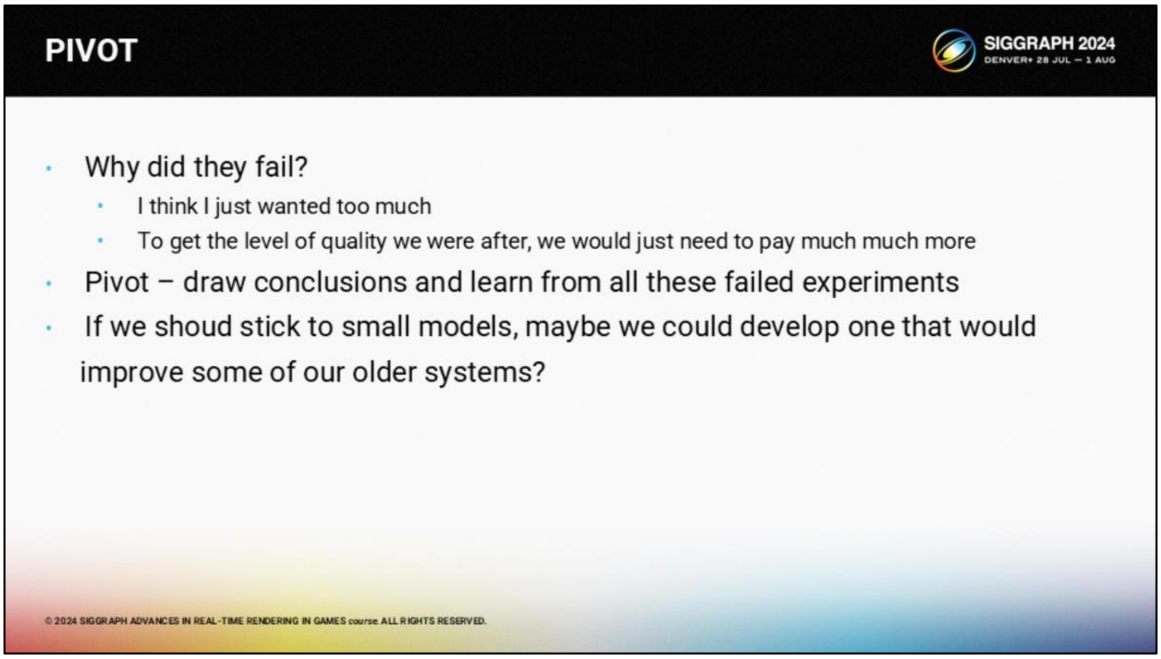


TIME OF DESPAIR

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG



But goddamit, all these neural networks work so well for everyone! There has to be something there, I need to get something out of that!


PIVOT

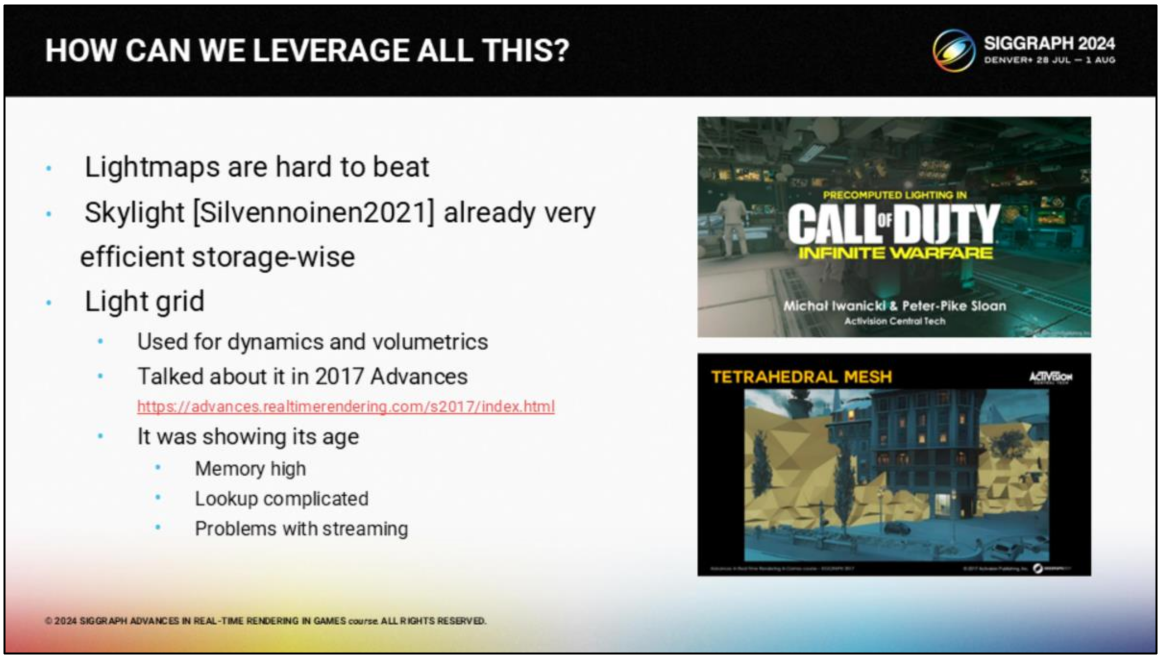
SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Why did they fail?
  - I think I just wanted too much
  - To get the level of quality we were after, we would just need to pay much much more
- Pivot – draw conclusions and learn from all these failed experiments
- If we shoud stick to small models, maybe we could develop one that would improve some of our older systems?

Ok, lets take a step back and try to think what these failed attempts had in common.

My conclusion was that I was expecting too much from them. I wanted to reconstruct complex, detailed signal from super compressed representation. Yes, neural networks can generate complex data – look at DALLe, Stable Diffusion etc – but the cost of evaluting these model is enormous compared to what we can do at real-time rates, it's orders of magnitude away. Smaller models have much much more moderate results. Super-resolution using deep learning is a good example here – the managable models do a decent job doing upscaling by a factor of 2 – generating 16 pixels from 4 – and even they are pretty large.

But I learned quite a lot during all this exploration, so maybe there is something that we could use. Everything seems to indicate that we should stick to smaller models so mayber we could develop one that would improve some of the systems we curretly use?



HOW CAN WE LEVERAGE ALL THIS?

SIGGRAPH 2024 DENVER+ 28 JUL – 1 AUG

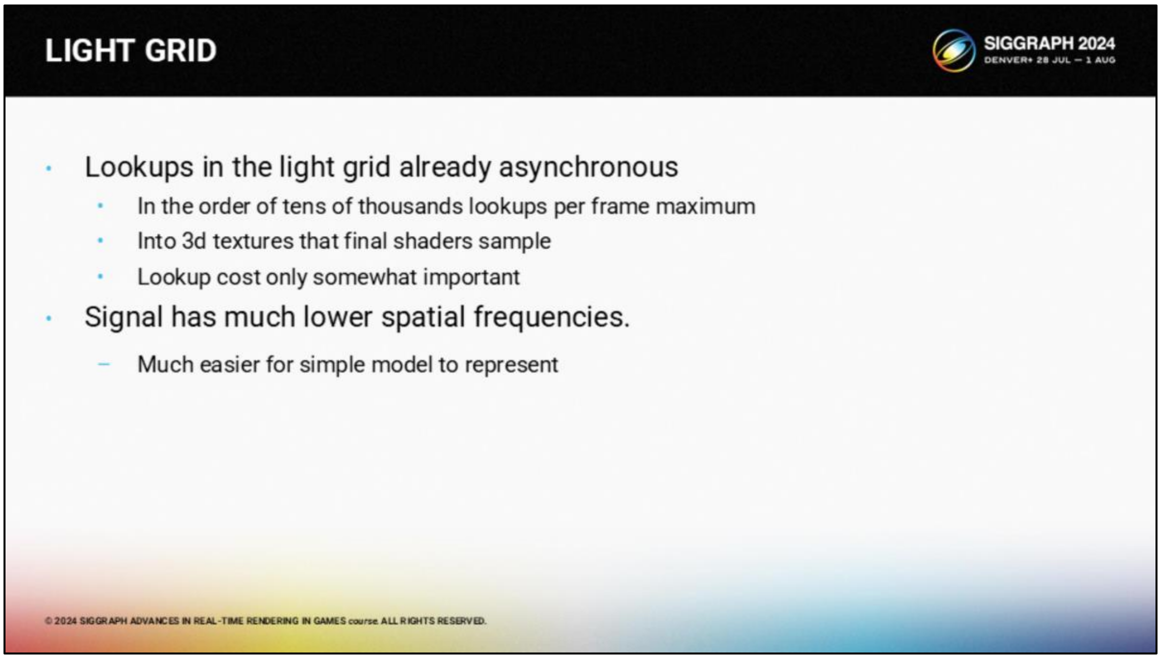
- Lightmaps are hard to beat
- Skylight [Silvennoinen2021] already very efficient storage-wise
- Light grid
  - Used for dynamics and volumetrics
  - Talked about it in 2017 Advances https://advances.realtimerendering.com/s2017/index.html
  - It was showing its age
    - Memory high
    - Lookup complicated
    - Problems with streaming

TETRAHEDRAL MESH
ACTIVISION

Just to give you a brief overview of what we use in the game:

we have lightmaps both 2d and 3d, which are really hard to beat – because they are super
we have skylight which is our internal name for a volumetric system used in the distance,a lower LOD for lighitng, based on moving basis decomposition that Ari talked about at EGSR few years ago,
And we have lightgid – it's a variation of an irradiance volume used to provide lighting for dynamic objects, volumetric effects etc
you can just query it at arbitrary point in space to get the radiance there.

We talked about it in this course in 2017. It was based on a tetrahedral mesh connecting probes and it was starting to show its age. It was using quite a bit of memory, The way it dealt with rejecting unwanted probes, to avoid leaking, was a bit problematic, because it wasn't c0 continuous. It was also difficult to deal with it on large maps and so on. Long story short it was definitely something that could be improved.



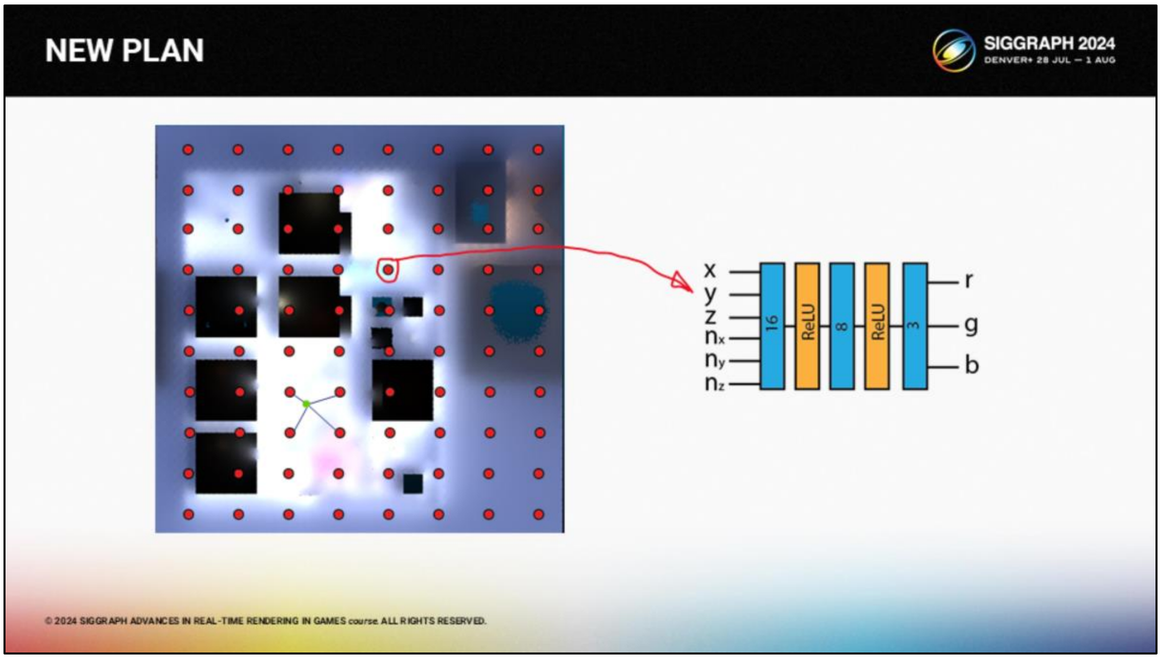
LIGHT GRID

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Lookups in the light grid already asynchronous
  - In the order of tens of thousands lookups per frame maximum
  - Into 3d textures that final shaders sample
  - Lookup cost only somewhat important
- Signal has much lower spatial frequencies.
  - Much easier for simple model to represent

Our lightgrid is sampled asynchronously, into 3d texture which were then used during actual rendering. Usually during a frame, we perform in the order of several tens of thousands lookups – for characters, particles, volumetric effects etc. The cost of the lookup is only somewhat important, as we don;t do that many.

But what is really important in the context of using ML for that, the spectral content of the signal is much lower. Since we resample it anyway, we dont store super high frequency details in the lightgrid. So we can hope it will be much easier to represent it with a model we can afford.

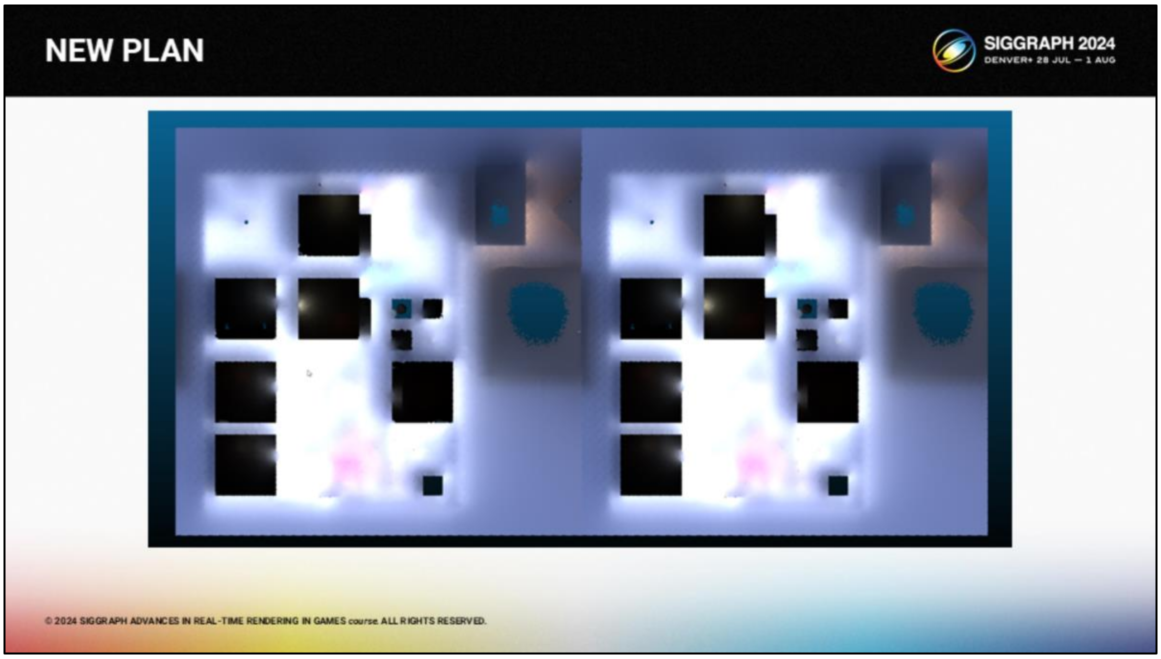


NEW PLAN

So we drew some conclustions for the earlier lessons and came up with a model like this
This is a cross section of the test map, showing radiance acorss some flat plane, viewed from above. This can give you some idea of the signal we're trying to encode. You can see the bright blue outside areas, some darker insides, light falling through doorways etc.

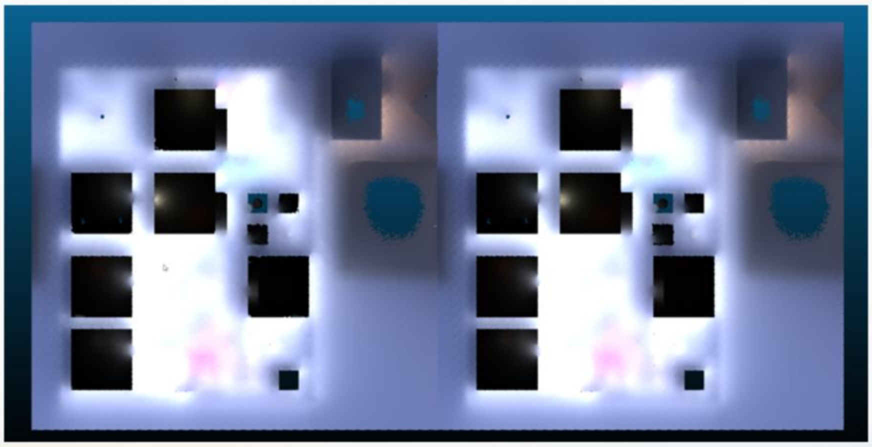
We create a grid of nodes across that, somewhat like probes
Since all these earlier experiments ehibited these hard edge on the voxel boundaries, wa wanted to incorporate a smooth interpolation from the very beginning to get rid of these.
Each of these networks is supposed to reconstruct the lighting around it – takes in the local position and direction as an input and returns the lighting. And because each point is influences by 4 of these, we evaluate 4 and interpolate the results.

So we took that whole lighting for the entire volume of the level and train all these network to output that baked lighting…

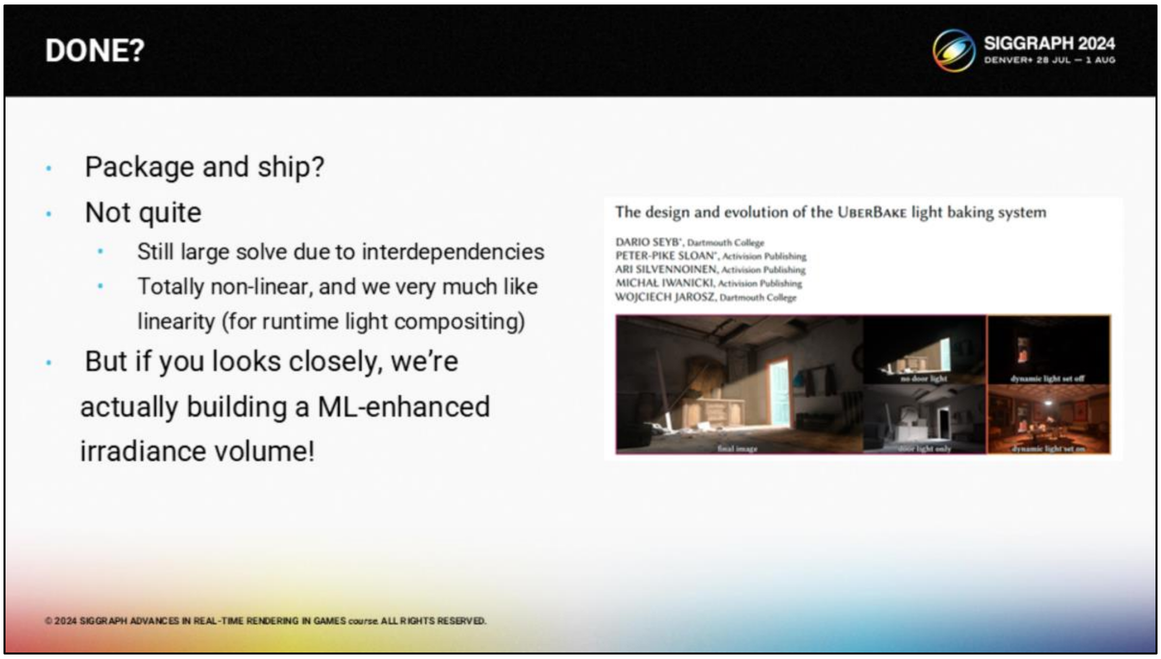


NEW PLAN

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG



And this is what came out of - reconstruction on the left, reference on the right.
As you can see, they are really, really close, neural version can represent the lighting discontinuities at the walls really well. The bright lighting stops nicely at the edges, there's no leaking.
Again now the denisty of this grid is much closer to the frequencies of the signal we want to approximate. We can do much better job with the reconstruction because we simply don't need to hallucinate that much.



DONE?

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Package and ship?
- Not quite
  - Still large solve due to interdependencies
  - Totally non-linear, and we very much like linearity (for runtime light compositing)
- But if you looks closely, we're actually building a ML-enhanced irradiance volume!

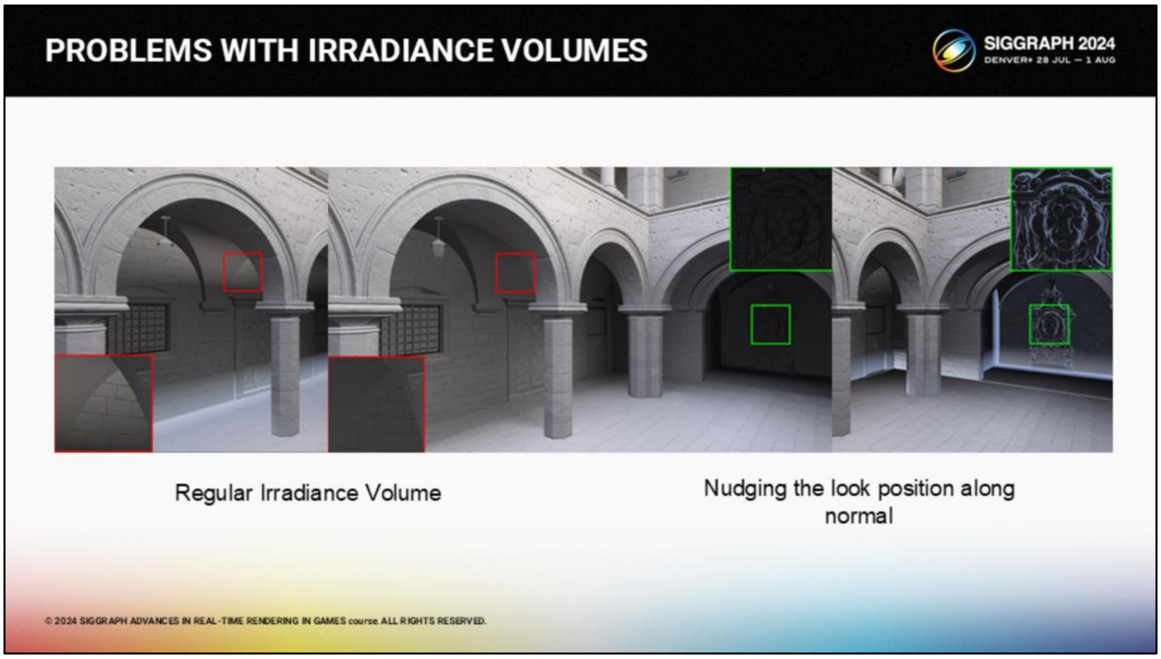
The design and evolution of the UberBake light baking system

DARIO SEYB*, Dartmouth College
PETER-PIKE SLOAN*, Activision Publishing
ARI SILVENNOINEN, Activision Publishing
MICHAŁ IWANICKI, Activision Publishing
WOJCIECH JAROSZ, Dartmouth College

So is that it? Are we done?
Of course not! We're talking about replacing a system that we had working for over 6 years, with tons of tentacles everywhere.
For instance, we very much rely on the fact that our lighting representation is linear – to do runtime compositing – we talked about it in the UberBake paper in 2020. We cannot really use networks, as the representation because they are non-linear and cannot be easily composed.

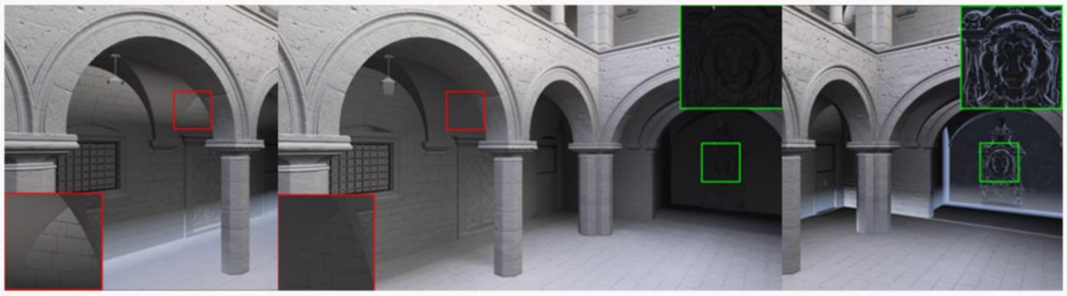
If you look closly at this last prototype it at it's heart, still an irradiance volume - modernized, ML-powered but still irradiance volume - you have a grid, you interpolate between the nodes, but the nodes are networks instead of just irradiance probes.
So what would we need to do to actually still use the irradiance probes but get some of the benefits of neural representation?

btw, Pete Shirley, my boss worked wit me on all this as well, is one of the authors of the original irradiance volume paper, he was very happy to go back to this topic after 30 years.



PROBLEMS WITH IRRADIANCE VOLUMES

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

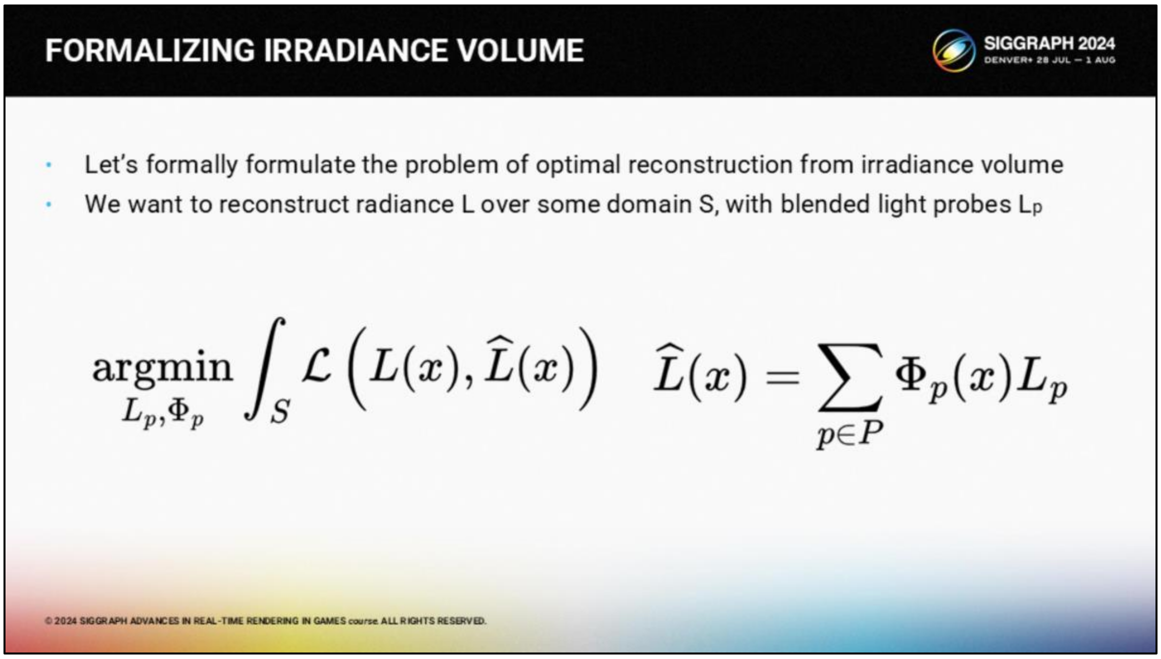


Regular Irradiance Volume

Nudging the look position along normal

We've been traditionally thinking about reconstructing lighting with irradiance volume, as simply blending/interpolating between the stored values. This is how it was originally described, and this is how it is most often approached.
But in this form irradiance volumes have some serious drawbacks, more importantly they dont care about geometry and just interpolate lighting through it, generating leaking like here
And solutoins for these problems are pretty ad hoc, we add some visibility information to the probes, to discard ones behind walls, we nudge the lookup position (like on the right), but its all pretty error prone

What we want it sth like this, a better, unified way to reconstruct the correct lighting.



FORMALIZING IRRADIANCE VOLUME

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Let's formally formulate the problem of optimal reconstruction from irradiance volume
- We want to reconstruct radiance L over some domain S, with blended light probes L_p

$$\underset{L_p,\Phi_p}{\operatorname{argmin}} \int_S \mathcal{L}\left(L(x), \widehat{L}(x)\right) \quad \widehat{L}(x) = \sum_{p\in P} \Phi_p(x) L_p$$

And there's actually a simple way to do this. We just need to properly formulate this reconsstruction with the irradiance volume.

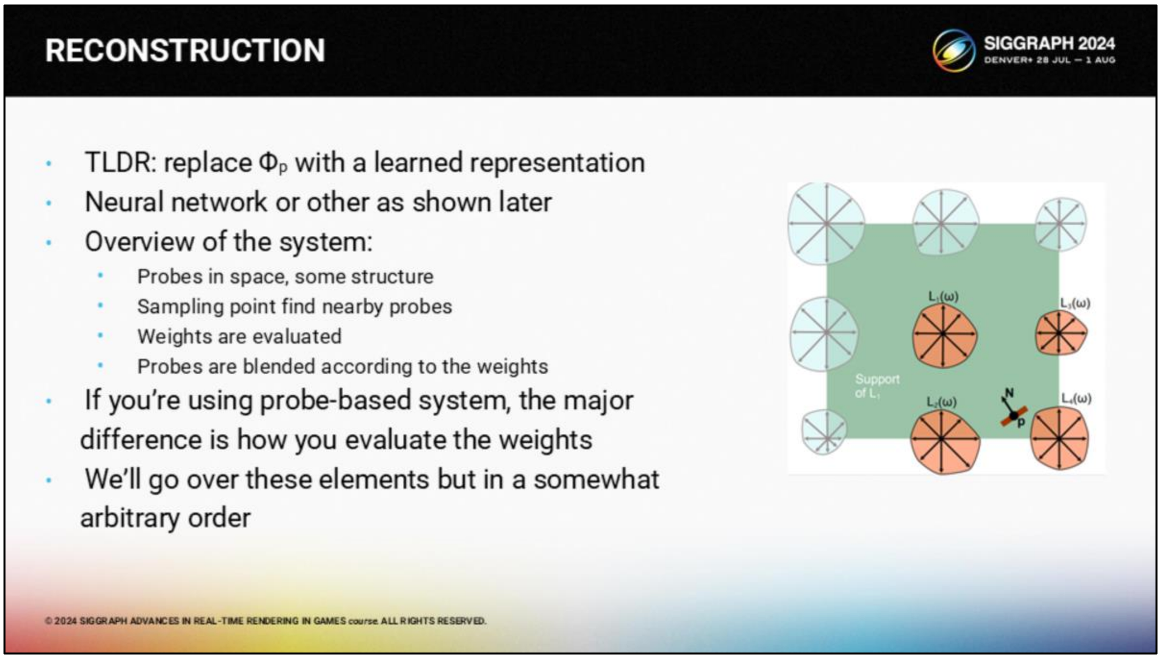
So formally speaking:
we have some domain S, and we want to approximate the radiance L over this domain with the approximated radiance Lhat. The Lhat is a weighted sum of some number of probes, Lp each weighted by some weighing function Phi, that varies spatially.

We want the difference between the two, indicated here with the Loss function symbol L, integrated over the entire domain to be minimized, by figuring out the radiance stored in the probes, Lp, and the weighing functions.

In the typical irradiance volume the the weighting functions phi, are just trilinear interpolation function, fixed. You can try to incorporate visibility into these, like in our old system, or DDGI, but it doesn't really mean that you minimize the reconstruction error, or at least not in any principled way.

But we can do a proper thing and actually find weighting functions that give us the optimizal reconstruction.

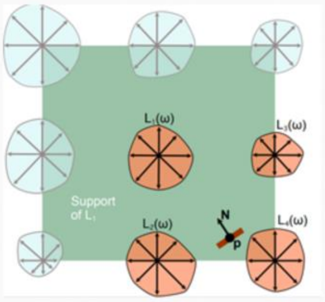
If you were to remember only one thing from that presentation, try to remember that slide, it’s easily the most important one – don’t treat the reconstructing lighting from probe as some ad-hoc blending, but as an actual minimization problem



RECONSTRUCTION

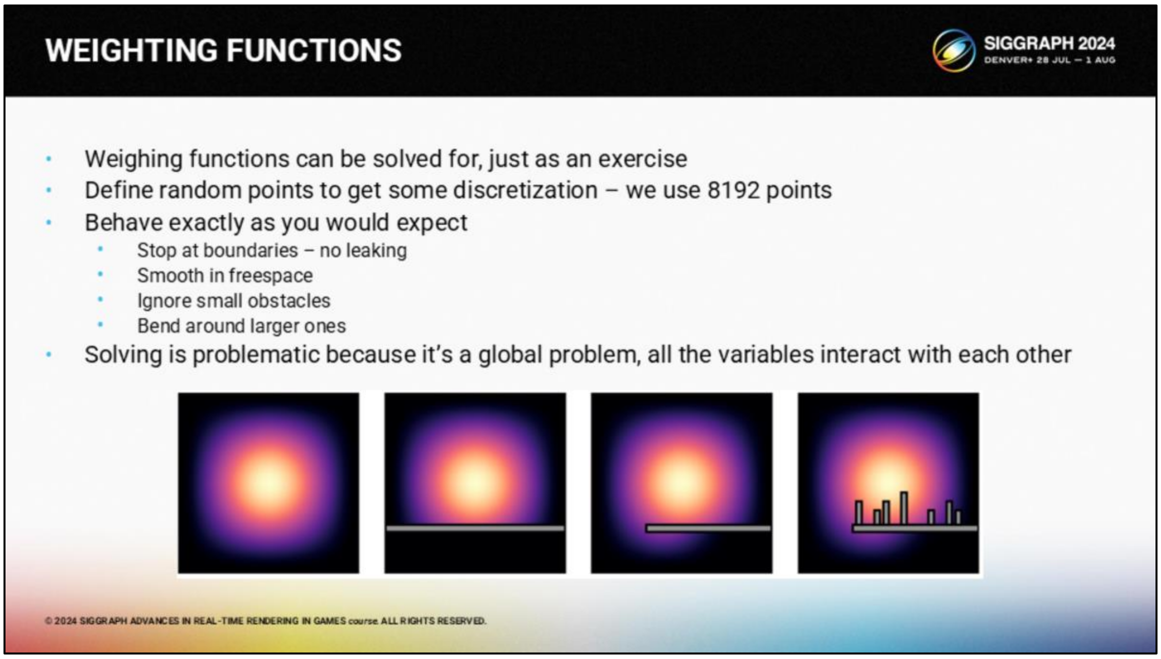
SIGGRAPH 2024 DENVER+ 28 JUL — 1 AUG

- TLDR: replace Φ_p with a learned representation
- Neural network or other as shown later
- Overview of the system:
  - Probes in space, some structure
  - Sampling point find nearby probes
  - Weights are evaluated
  - Probes are blended according to the weights
- If you're using probe-based system, the major difference is how you evaluate the weights
- We'll go over these elements but in a somewhat arbitrary order



TLDR of our solution: we use proper weighting functions.

So the solution we ended up with is a modern version if irradiance volume - bunch of probes in some data structure and a representation of their weighting functions. When sample lighting, we find which probes influence the sampling position, we evaluate their weighting functions and blend them accordingly.
The solution that we actually ended up with and shipped was, as you can see much more modest compared to what we started with.
We will now go over these individual elements in somewhat random order.



WEIGHTING FUNCTIONS

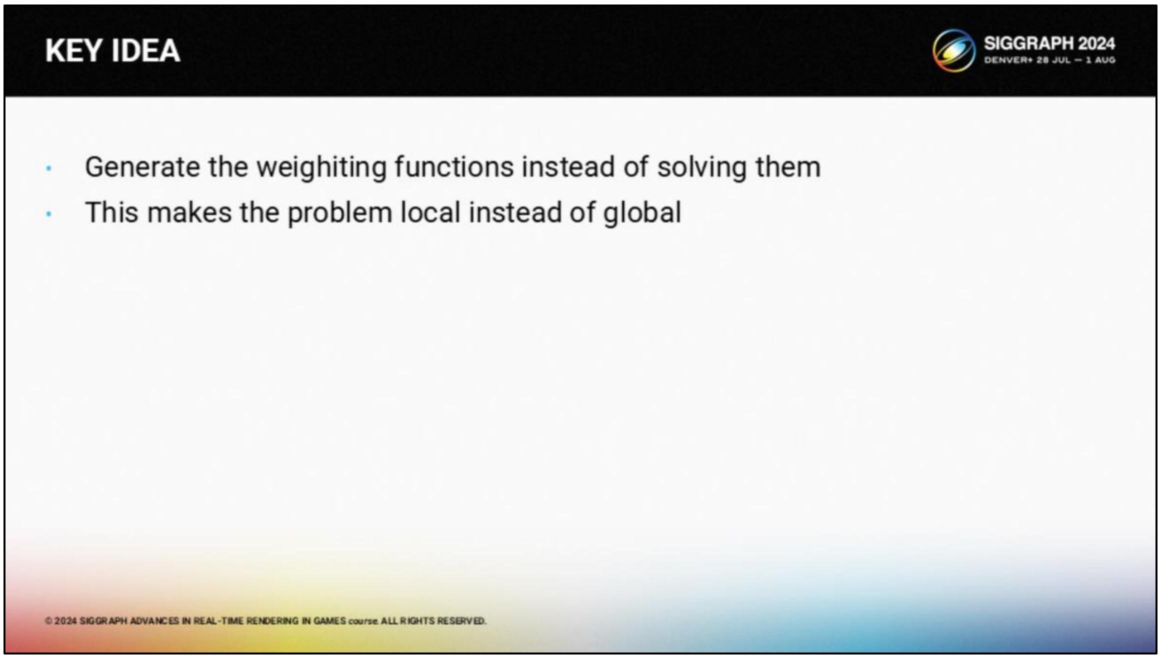
SIGGRAPH 2024 DENVER+ 28 JUL – 1 AUG

- Weighing functions can be solved for, just as an exercise
- Define random points to get some discretization – we use 8192 points
- Behave exactly as you would expect
  - Stop at boundaries – no leaking
  - Smooth in freespace
  - Ignore small obstacles
  - Bend around larger ones
- Solving is problematic because it’s a global problem, all the variables interact with each other

First lets look at the weighting functions. If we describe them in some parametric form we can solve for them, based on the minimization problem from two slides before. The simplest way is to just discretize these functions - point sample them - and use some gradient descent algorithm to minimize the error, just L2 in the simplest case, We generate 8192 random points in the space around each probe, this give a good coverage of space, while still being manageable.

And if we then visualize functions they behave pretty much exactly as you would expect – in free space they just look like a simple interpolation kernel, if there’s some boundary, a wall for instance, they abruptly stop on it. They pretty much ignore small obstacles, but smoothly wrap around large one. They are continuous in free space.

Solving for these functions is the proper thing to do. It is however a global problem – because all the functions overlap and interact with each other, everything needs to be solved for simultaneously, creating one, possibly very large system – and as you might expect, it makes it pretty slow, too slow for production.

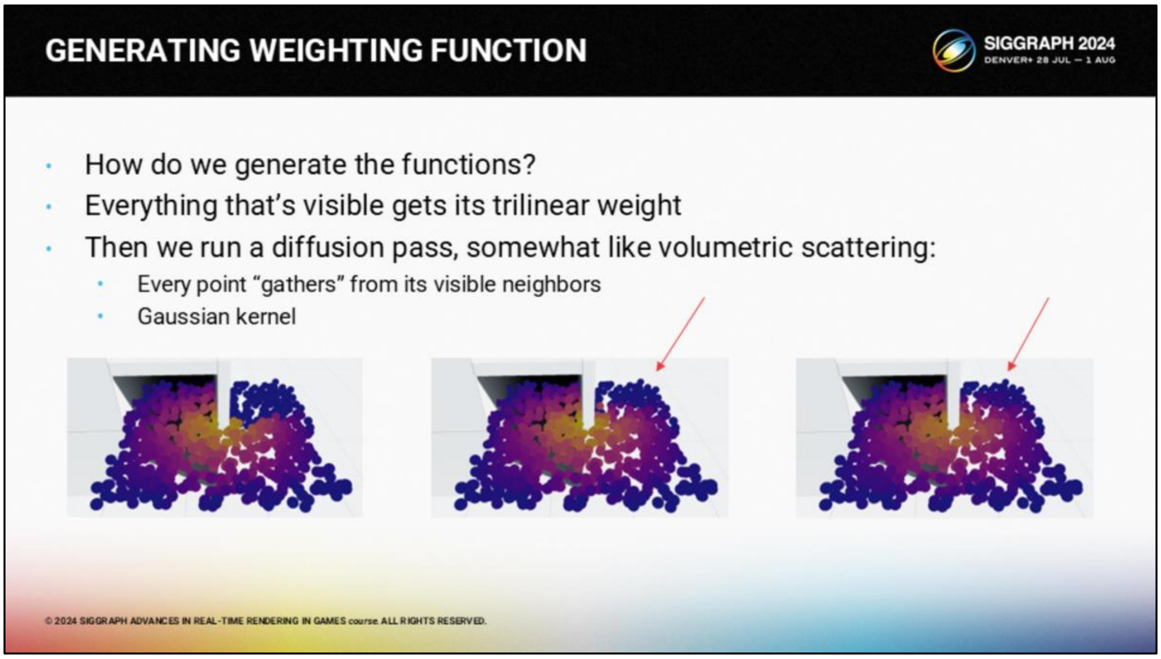


KEY IDEA

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Generate the weighiting functions instead of solving them
- This makes the problem local instead of global

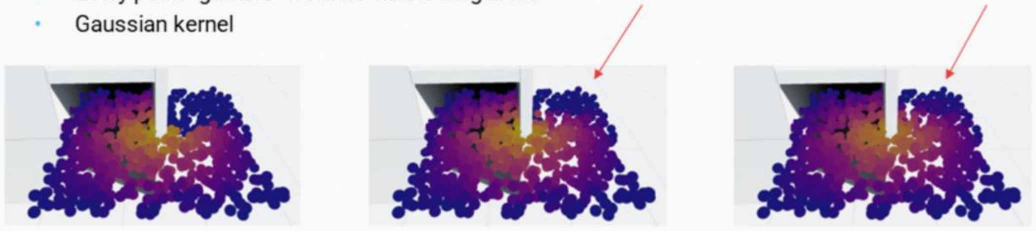
So instead of solving them, we generate them, one by one, completely independently, but in a way that gives us results similar to what they are when solved for



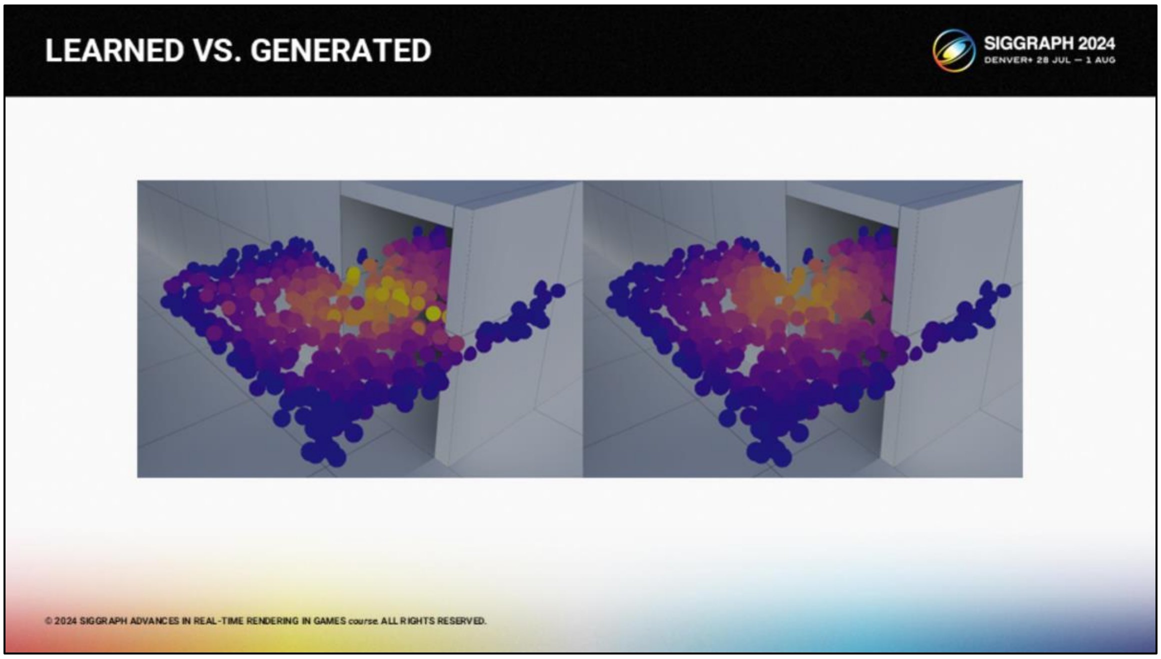
GENERATING WEIGHTING FUNCTION

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- How do we generate the functions?
- Everything that's visible gets its trilinear weight
- Then we run a diffusion pass, somewhat like volumetric scattering:
  - Every point "gathers" from its visible neighbors
  - Gaussian kernel

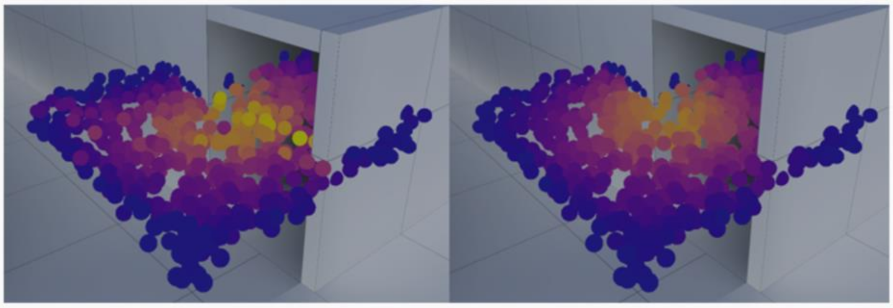


The process somewhat similar to volumetric scattering. We start from the probe position in the center. Every point of our discretization that's visible from that center position gets a weight that's equal to the trilinear interpolation weight. Next we run 2 "diffusion" passes where each point that hasn't received any contribution yet looks at its closest neighbors and gathers their weight, with some hand-tweaked Gaussian falloff. You can see the process on some example case. This visualization will be used in the next slides too, this is how we debug display the discretized weighting function: these are just the points from the discretization, but only ones on some plane, so it's easier to analyze. In the first step we get this initial radial pattern from the center, but it's occluded on the back-right by a piece of wall. In the next two steps that function gradually creeps over the edge and smooths out.

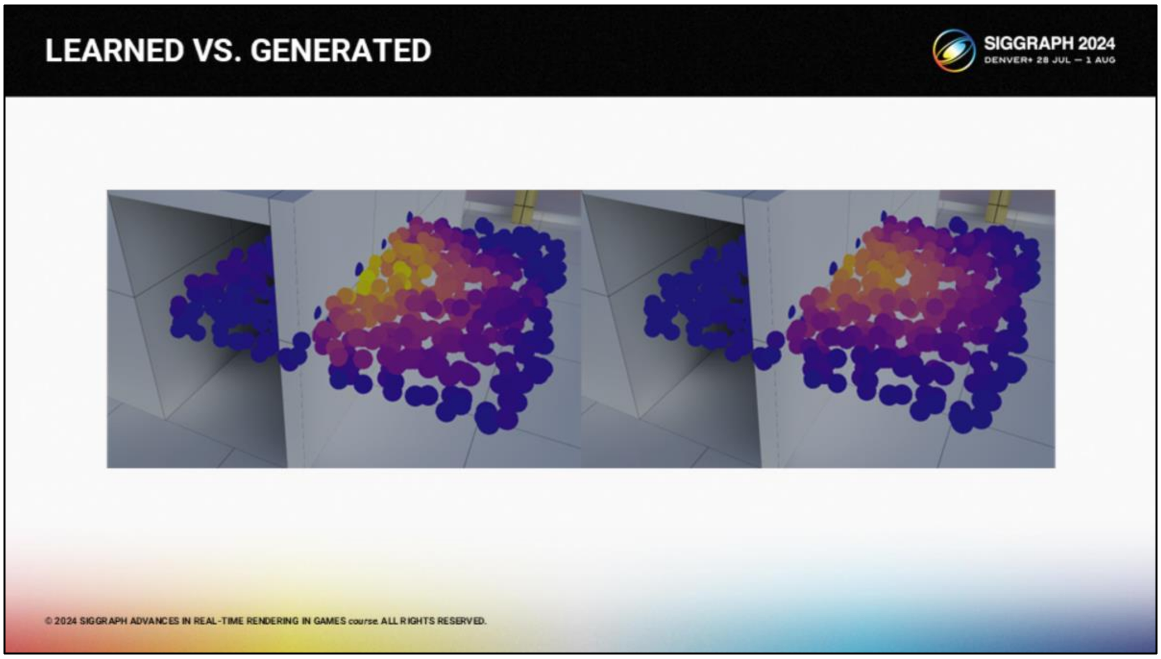


LEARNED VS. GENERATED

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

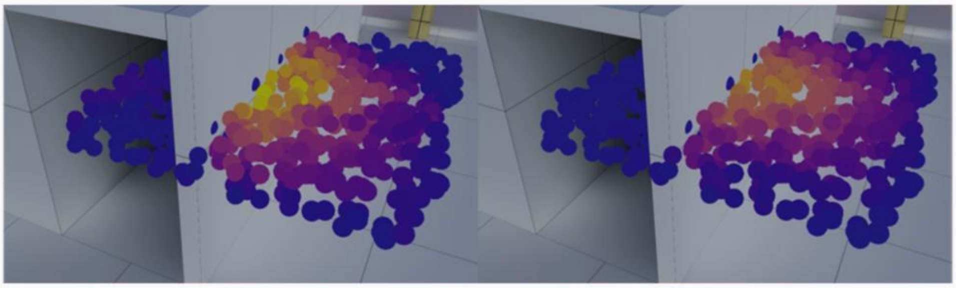


This is a bunch of examples of the generate functions vs the learned ones – learned ones on the left, generated on the right.



LEARNED VS. GENERATED

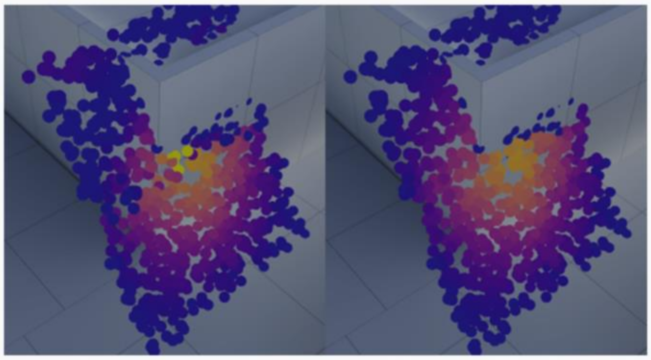
SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

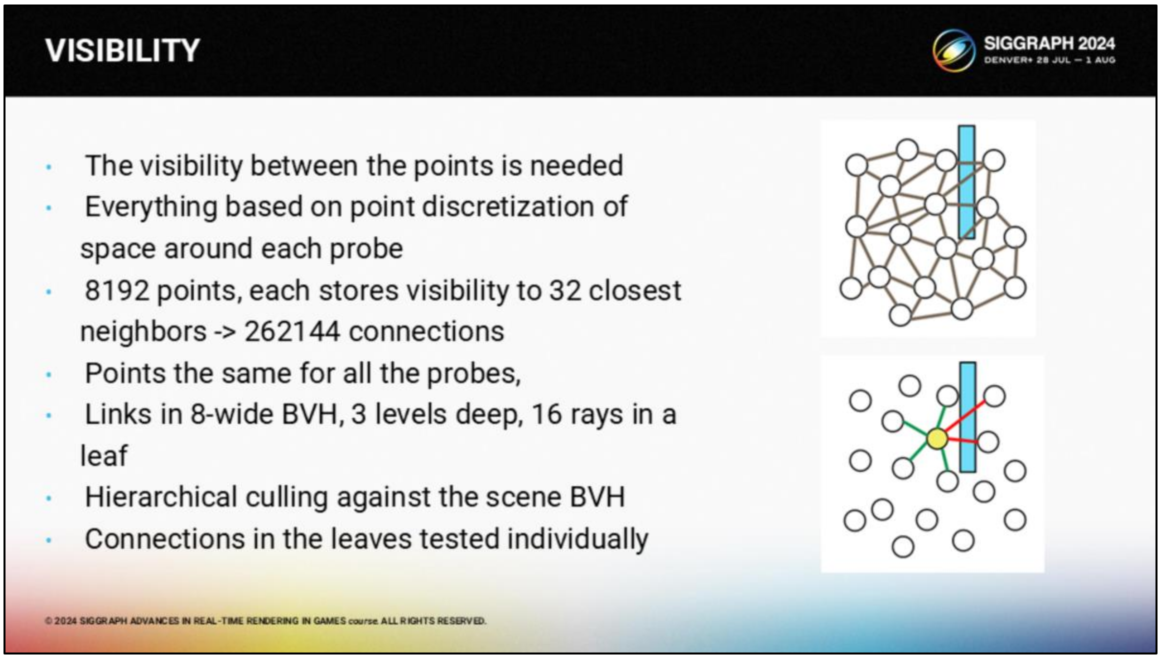


As you can se they are not identical, but they behave in a very similar way.

LEARNED VS. GENERATED

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

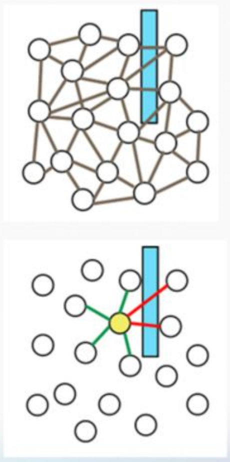




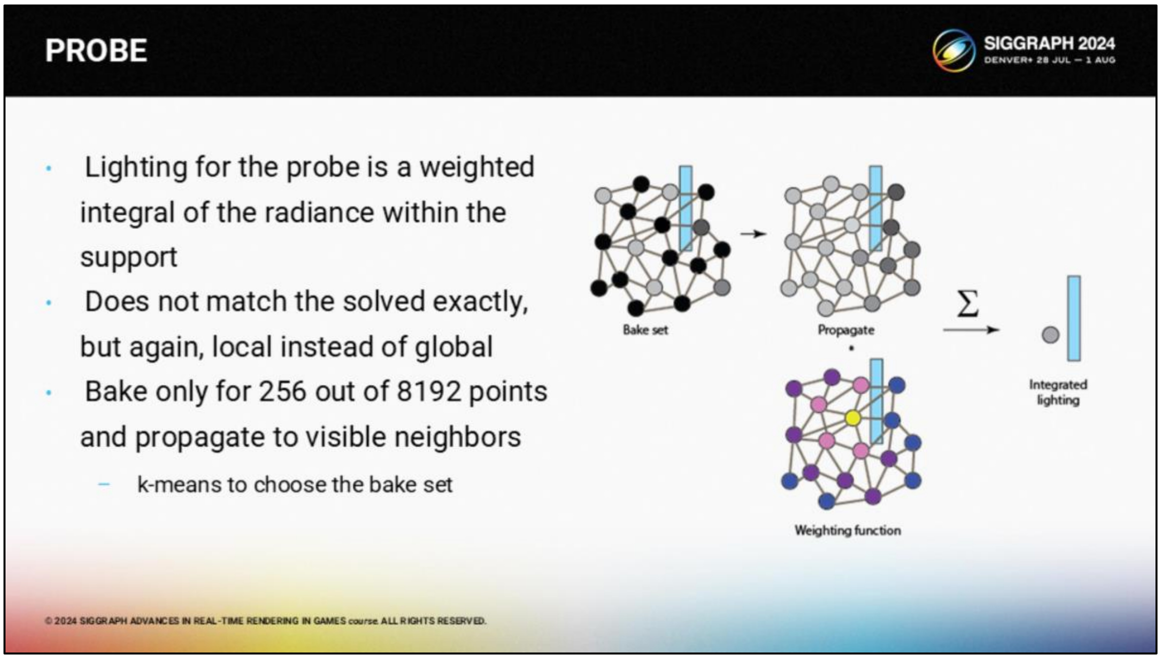
VISIBILITY

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- The visibility between the points is needed
- Everything based on point discretization of space around each probe
- 8192 points, each stores visibility to 32 closest neighbors -> 262144 connections
- Points the same for all the probes,
- Links in 8-wide BVH, 3 levels deep, 16 rays in a leaf
- Hierarchical culling against the scene BVH
- Connections in the leaves tested individually



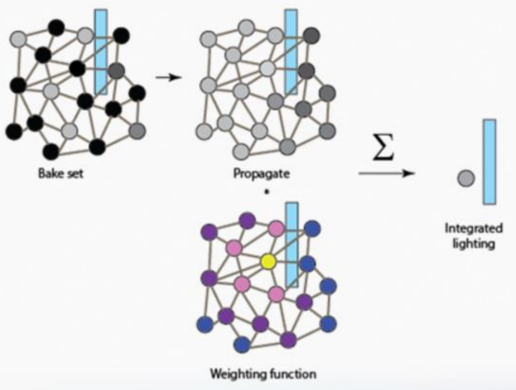
This whole generation process relies on knowing which of the points see which of their neighbors. Since each probe has over 8 thousands points, and there's usually few hundred thousands probes on a map, just naively testing these connections would be prohibitively slow. To speed it up, we do a bunch of things.
First, we use the same pattern of these points for every probe, so we can precompute some things. Next, we only analyze visibility to the closest 32 neighbors for each point. All these numbers btw are just a result of some experiments, they are some balance between the quality of representation and computational cost.
This still gives over 260 thousands connections, so we group them and put these links in a BVH tree. Then, for every probe we do BVH-vs-BVH test against the scene to quickly discard big groups of connections that don't intersect with any of the scene geometry. Only the connections that potentially hit anything are actually tested using regular ray intersection tests. All this allows us to compute all these intersections for a typical scene of around 200k probes in under a minute.



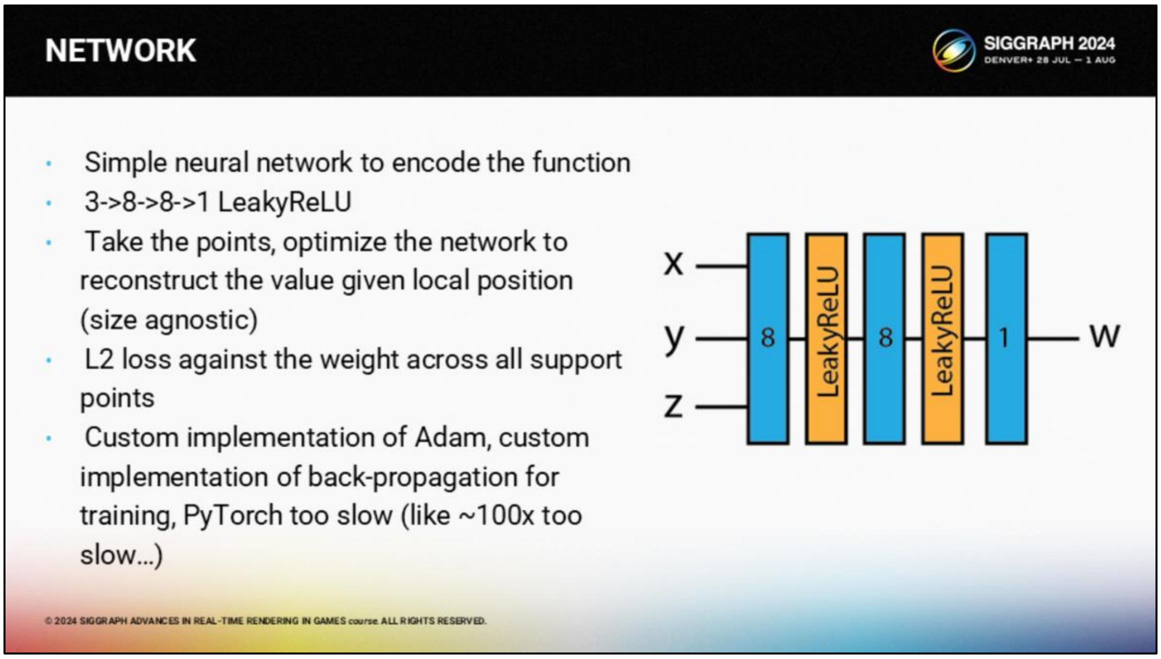
PROBE

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Lighting for the probe is a weighted integral of the radiance within the support
- Does not match the solved exactly, but again, local instead of global
- Bake only for 256 out of 8192 points and propagate to visible neighbors
  - k-means to choose the bake set



The lighting that we store in the probes is also not solved for, for the same reasons. It's instead integrated over the support of the weighting function.
So we first bake a low-quality lighting for a subset of our support points, 256 of them – again 256 just hit the right balance.. Next, using the mutual visibility of the points, we propagate the lighting to the *visible* neighbors of each points. Once every point has its lighting, we multiply it by the value of the weighting function and sum up all the contributions.
This has a big advantage over just point-sampling the radiance – which happens when you render a cubemap at the probe location – because it spatially filters the radiance signal over the support of the probe. This eliminates significant spatial aliasing that is present with point-sampled radiance/visibility (it manifests itself by a large dependence of the probe radiance on its location, small perturbation in probe position can massively change the entire reconstructed signal around it).

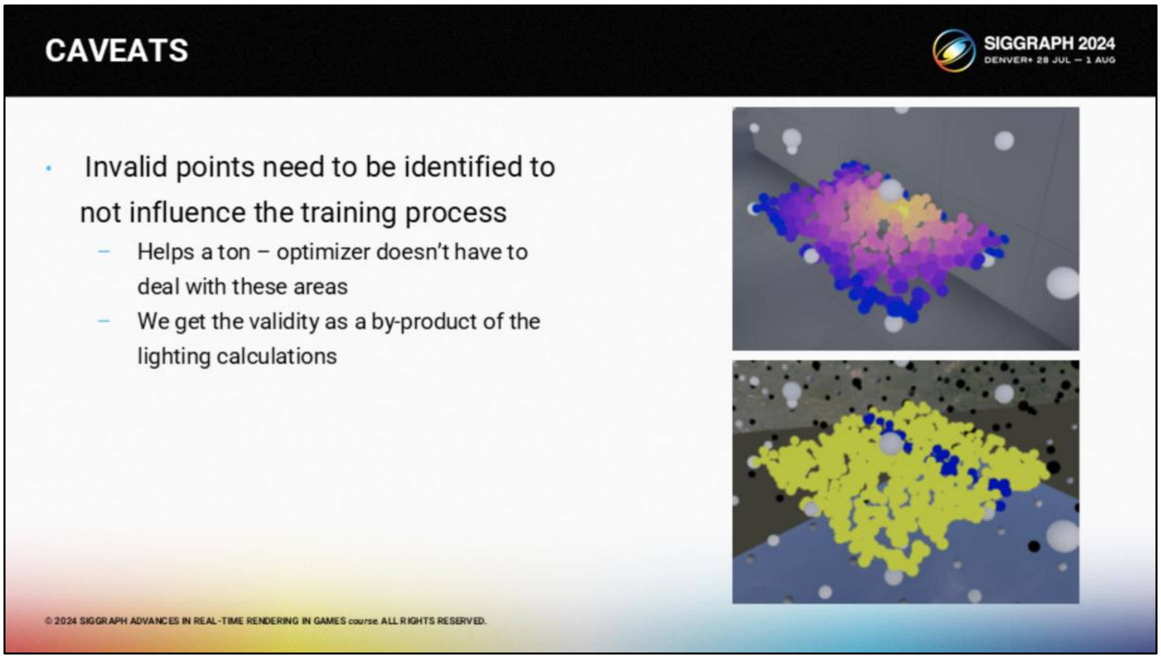


NETWORK

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Simple neural network to encode the function
- 3->8->8->1 LeakyReLU
- Take the points, optimize the network to reconstruct the value given local position (size agnostic)
- L2 loss against the weight across all support points
- Custom implementation of Adam, custom implementation of back-propagation for training, PyTorch too slow (like ~100x too slow...)

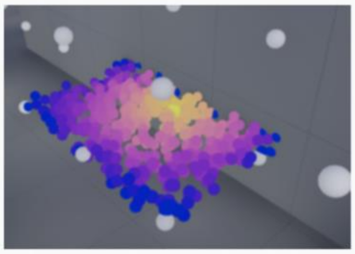
For runtime evaluation, we turn this point representatio of the weighting functions into a very small neural network.
It takes local xyz position as an input, has two hidden layers, 8-wide, and uses LeakyReLU activation – LeakyReLu is just a different non-linear activation function, similar to ReLU but instead of clamping the negative values it multiplies them by a small number – in the simplest term it helps with the training and gives the network a bit more expressive power
The network outputs a single value – the weight at the given position.
For training these, we wrote our own implementation of Adam, and our own back propagation optimized for these small networks, as PyTorch was way too slow in such small, constrained scenario – in our tests the CPU version of PyTorch could be around 100x slower than our custom code, and the GPU PyTorch was only 15% faster than our CPU code (everything measured on AMD 5995WX + RTX 3090).

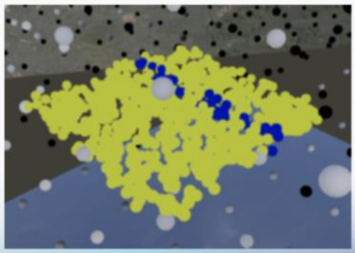


CAVEATS

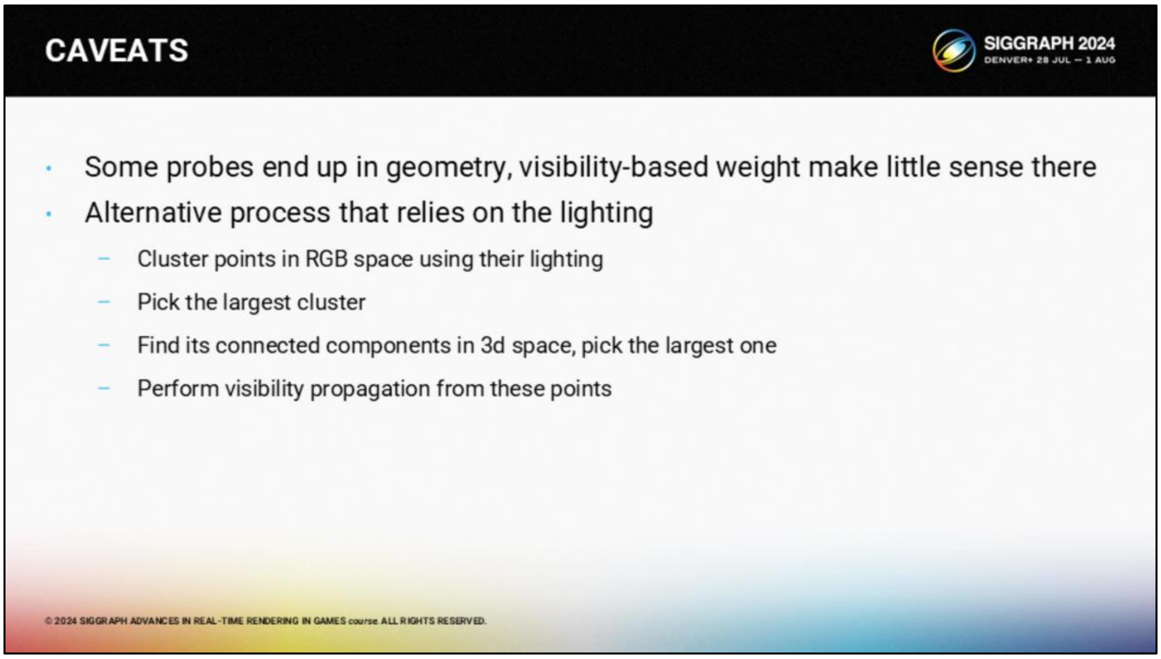
SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Invalid points need to be identified to not influence the training process
  - Helps a ton – optimizer doesn't have to deal with these areas
  - We get the validity as a by-product of the lighting calculations





There is a bunch of details here and there that are important to the process – for instance, many of the support points will fall inside the geometry – and that's actually good. They don't need to contribute to the loss during optimization, and the network is free to do anything it needs in these regions, which makes the training process much more efficient. We get this validity information from our baking pipeline, as a by-product of the lighting calculations.



CAVEATS

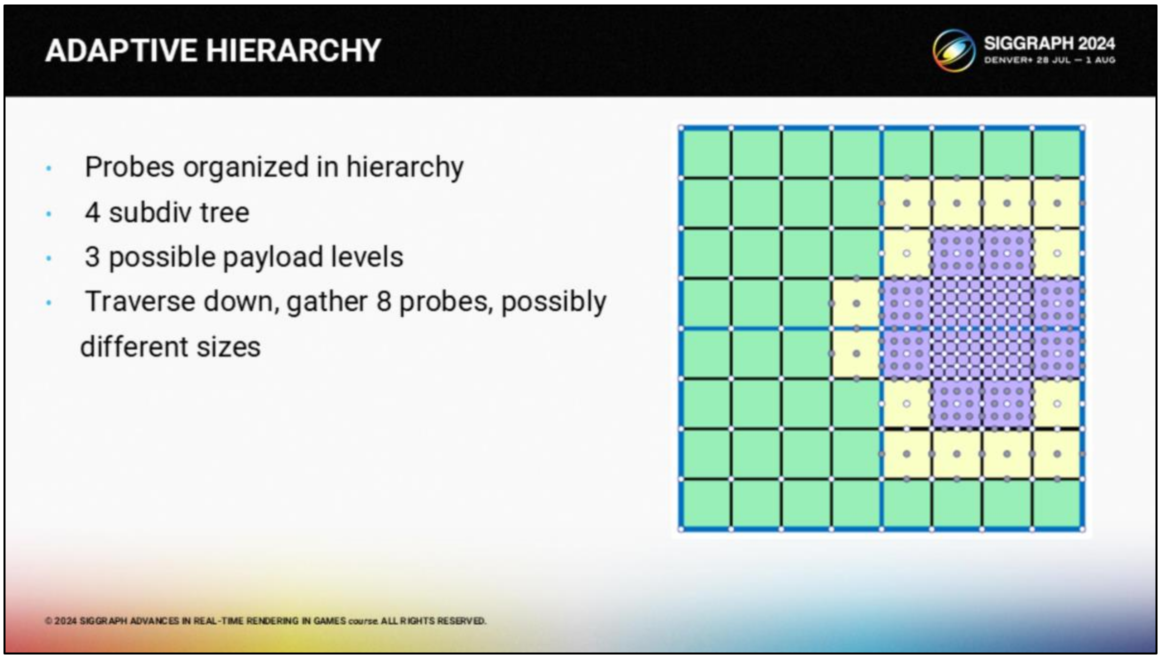
SIGGRAPH 2024 DENVER+ 28 JUL — 1 AUG

- Some probes end up in geometry, visibility-based weight make little sense there
- Alternative process that relies on the lighting
  - Cluster points in RGB space using their lighting
  - Pick the largest cluster
  - Find its connected components in 3d space, pick the largest one
  - Perform visibility propagation from these points

Another thing worth mentioning are situations when the center position, the location of the probe, ends up inside the geometry. The generated weighting function in such case is incorrect, all empty. I won't go into details of how to deal with it, but we simply generate the weightig function slightly differently

We start by generating the lighitng for all the support points. We then perform clustering of them in RGB space – to group areas that have similar brightness, to ensure that if there's some hard discontinuity, the generated weighting function wont bleed over. We pick the largest cluster and find its connected component in 3d space – as even a continous chunk in RGB can be disjoint spatially. We take the largest connected component and every point in it gets an initial weight equal to the trilinear kernel. Next we proceed with the spatial diffusion process ass before.

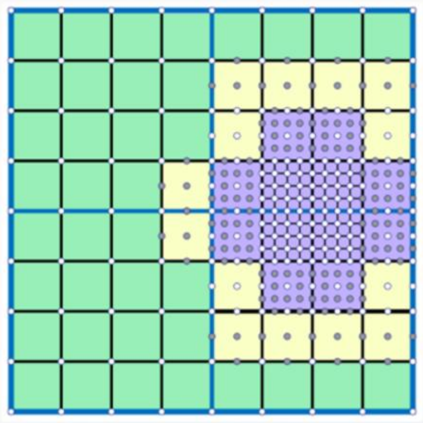
Lstory short it all naturally fits into the whole framework, and doesn't really pose any problems. We don't need to nudge the probe or hack it in any way, it all just relies on the weighting function



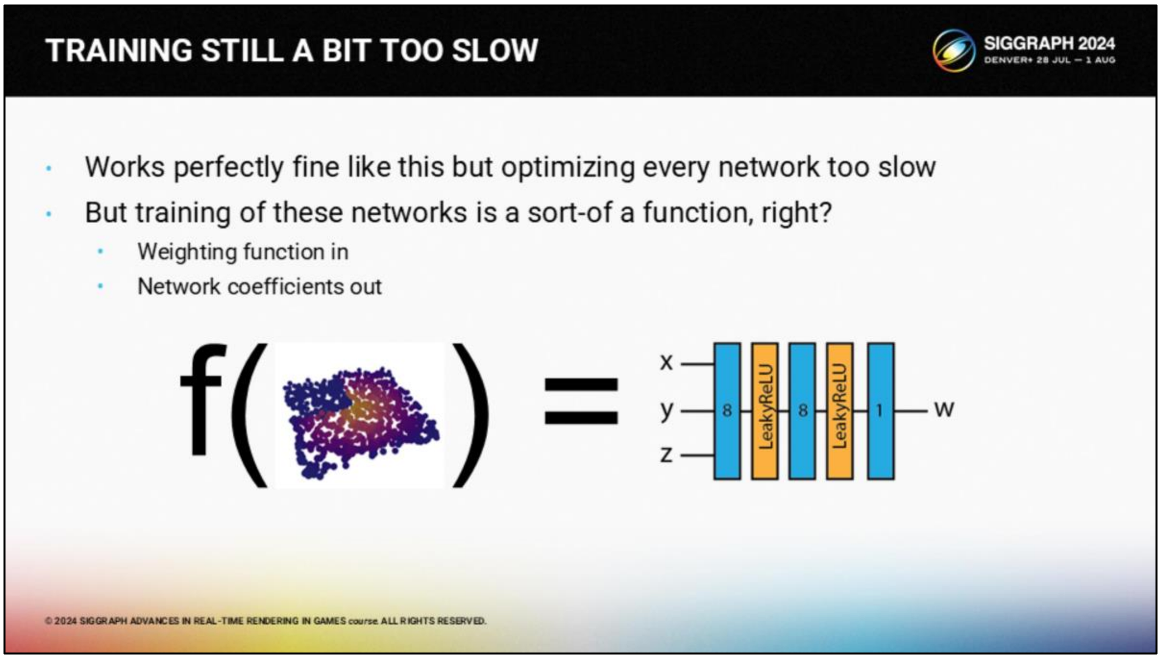
ADAPTIVE HIERARCHY

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Probes organized in hierarchy
- 4 subdiv tree
- 3 possible payload levels
- Traverse down, gather 8 probes, possibly different sizes

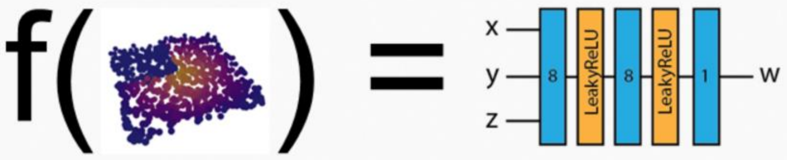


The probes are organized in a fairly standard hierarchy – it's a tree with a subdivision factor of 4 on every axis, so each node can have up to 64 children. The tree is 3 levels deep maximum. Each node in the tree stores a payload that's a brick of probes - that can be 2x2x2 – with probes just in the corners, 3x3x3 with extra probe in the center or 5x5x5. Neighboring nodes share the edge and corner probes, and corner and edge probes are shared between neighbors.



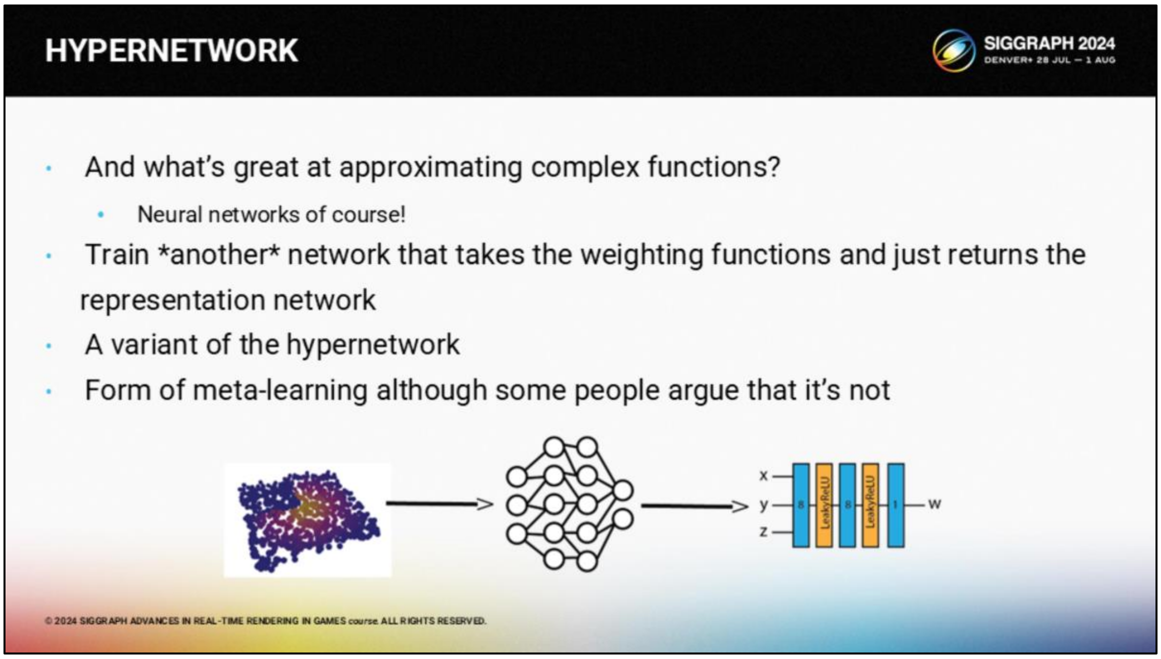
TRAINING STILL A BIT TOO SLOW

- Works perfectly fine like this but optimizing every network too slow
- But training of these networks is a sort-of a function, right?
  - Weighting function in
  - Network coefficients out



But we're not done yet! Despite having a really fast optimizer, it's still too slow to optimize the network for every weighting function. Just to give you some idea, my Threadripper can do around 60-80 weighting functions a second – that means around an hour for a decently sized map – a bit too slow for our taste.

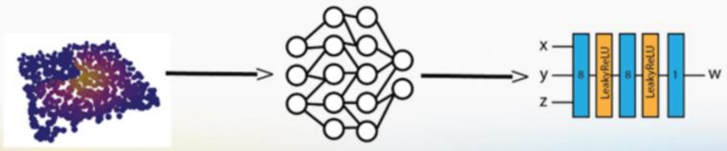
But if you think of this whole training process it's sort of a function, right? It takes the weighting function as an input and produces some network coefficents as the output. Of course it's a complicated multidimensional function, that we have no way of decribing.



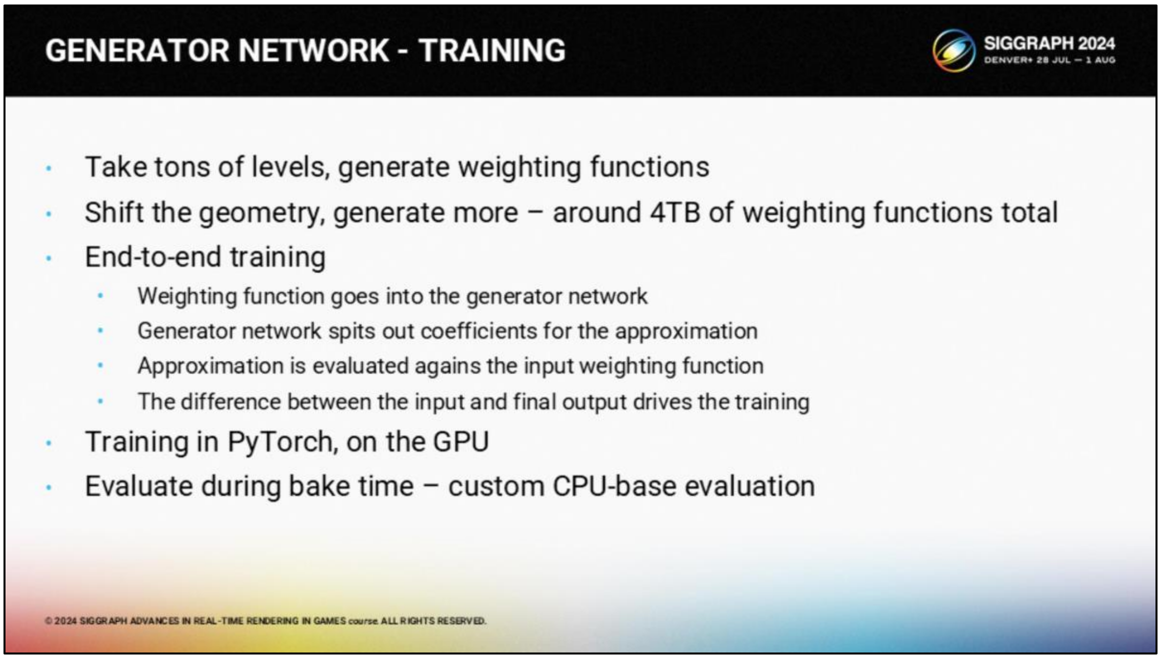
HYPERNETWORK

SIGGRAPH 2024 DENVER+ 28 JUL — 1 AUG

- And what's great at approximating complex functions?
  - Neural networks of course!
- Train *another* network that takes the weighting functions and just returns the representation network
- A variant of the hypernetwork
- Form of meta-learning although some people argue that it's not



But guess what's great at approximating such functions! Neural networks
So we train *another* network that approximates the training process. It takes the generated function as an input and directly, without any iterative process, produces coefficients of the network that approximate that weighting function on the output.
We call this network a "generator network" and it's a form of so-called hypernetwork, which is a form of meta learning (as least we think so, and some people very firmly try to tell us that it is not)

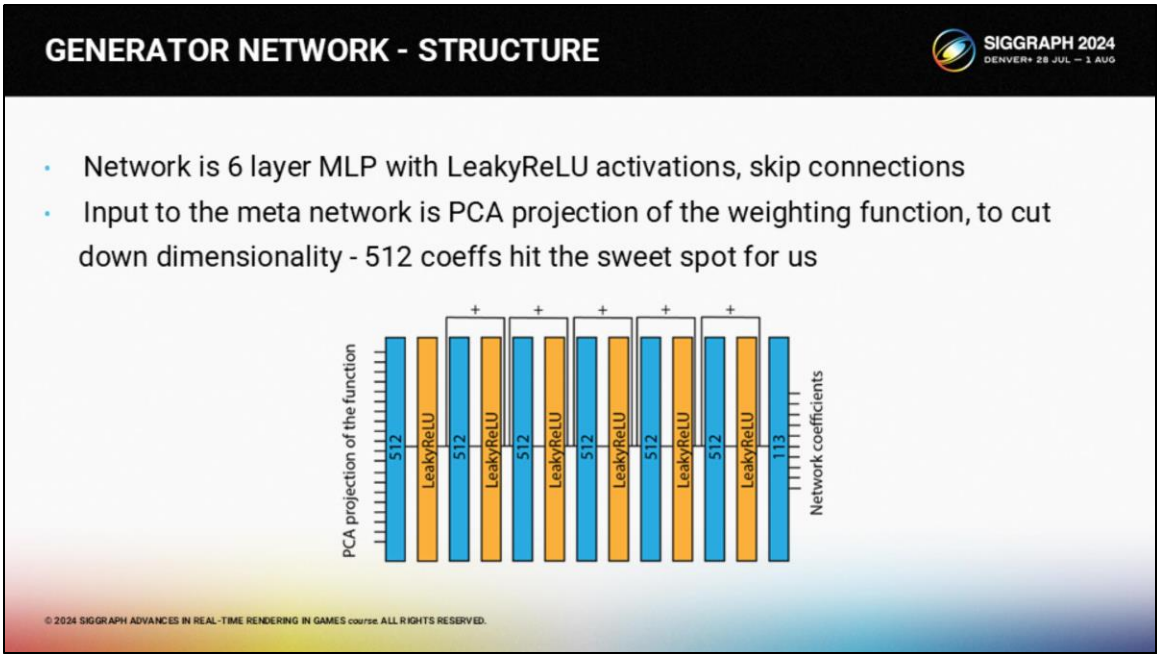


GENERATOR NETWORK - TRAINING

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Take tons of levels, generate weighting functions
- Shift the geometry, generate more – around 4TB of weighting functions total
- End-to-end training
  - Weighting function goes into the generator network
  - Generator network spits out coefficients for the approximation
  - Approximation is evaluated agains the input weighting function
  - The difference between the input and final output drives the training
- Training in PyTorch, on the GPU
- Evaluate during bake time – custom CPU-base evaluation

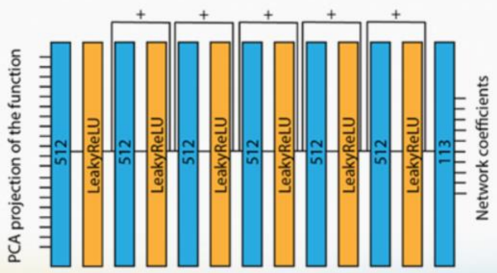
We basically took a bunch of maps and generated weighting functions for them. Then we randomly shifted things and generated more weighting function, altogether around 4TB of training data. We train this network completely offline, this time with PyTorch, on the GPU,. It takes around 18h to train it to convergence, but it’s not an issue as we do it only once. Then we export it from PyTorch and use it during the bake. Whenever lighters light a map, this gets evaluated for every probe. Again, we have a custom SSE code for that, just to take advantage of the known structure of the network and shave off some seconds from the inference times



GENERATOR NETWORK - STRUCTURE

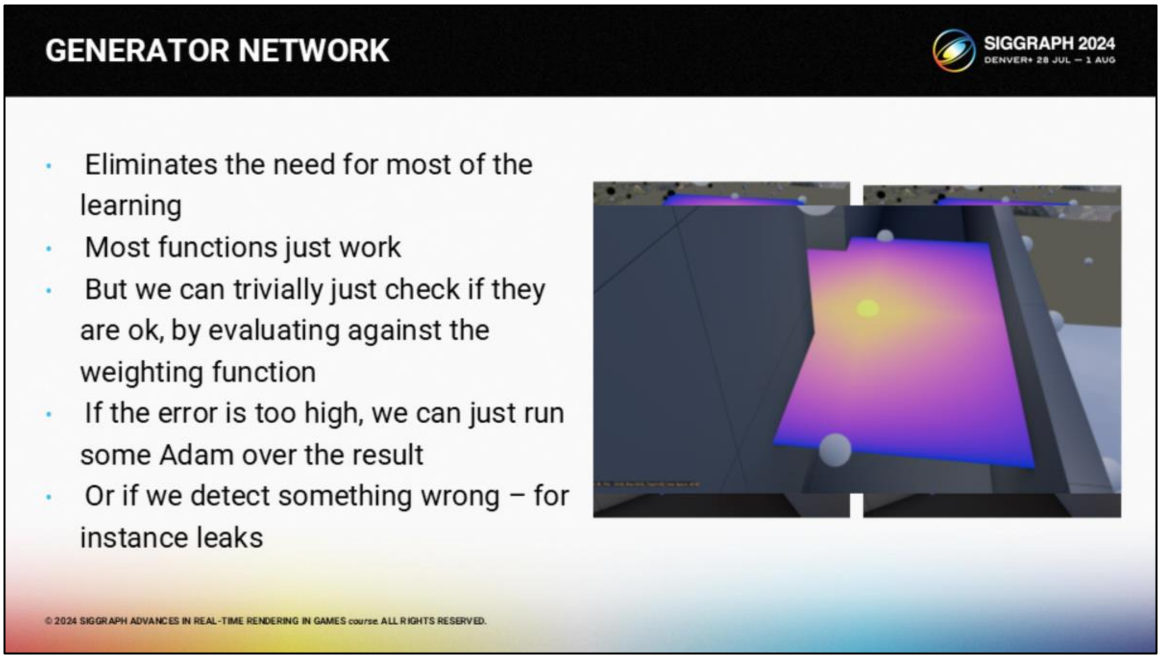
SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Network is 6 layer MLP with LeakyReLU activations, skip connections
- Input to the meta network is PCA projection of the weighting function, to cut down dimensionality - 512 coeffs hit the sweet spot for us



That network is a 6 layer percepron with LeakyReLU activations and skip connections. But the actual architecture is not really that crucial here, the network is pretty resiliant to changes. this particular setup hit some sweet spot for us, any further increase of the size came with minimal improvements, so we just sticked to that.

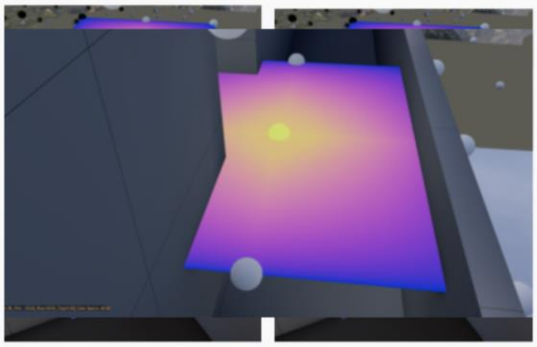
As an input it takes a PCA projection of the weighting functions, its first 512 coefficients. The PCA basis is derived from the whole training set and limits the size of the input, - we only pass the first 512 coefficients – again, this was a good balance between the evaluation time and quality.



GENERATOR NETWORK

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

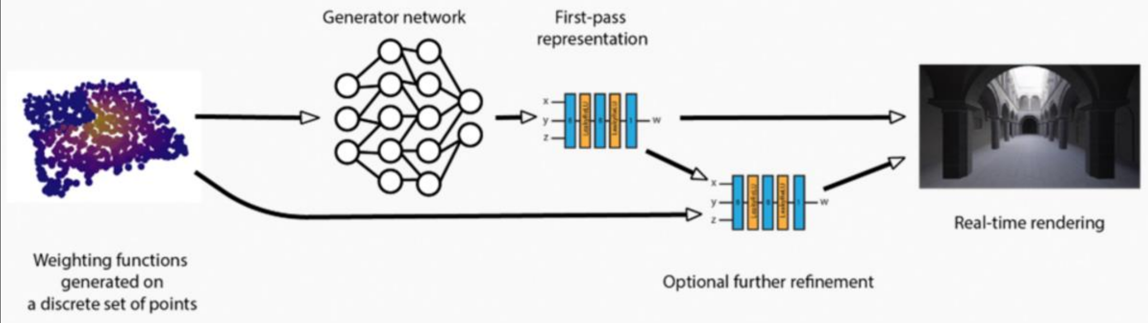
- Eliminates the need for most of the learning
- Most functions just work
- But we can trivially just check if they are ok, by evaluating against the weighting function
- If the error is too high, we can just run some Adam over the result
- Or if we detect something wrong – for instance leaks



The generator network pretty much eliminates the need to do any training for majority of the functions –we just put in the weighting function in, get the coefficients on the other end and we're done. But there are situations, where the result is not ideal, maybe there was too little of certain types of functions in the training data – like in the example here, where the initial output from the generator network leaks through the wall a bit. But this is of course not a problem, because we can just take it and run regular optimization further, starting from what the generator network gave us, and after a bit of training it's all good.

COMPLETE SYSTEM

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

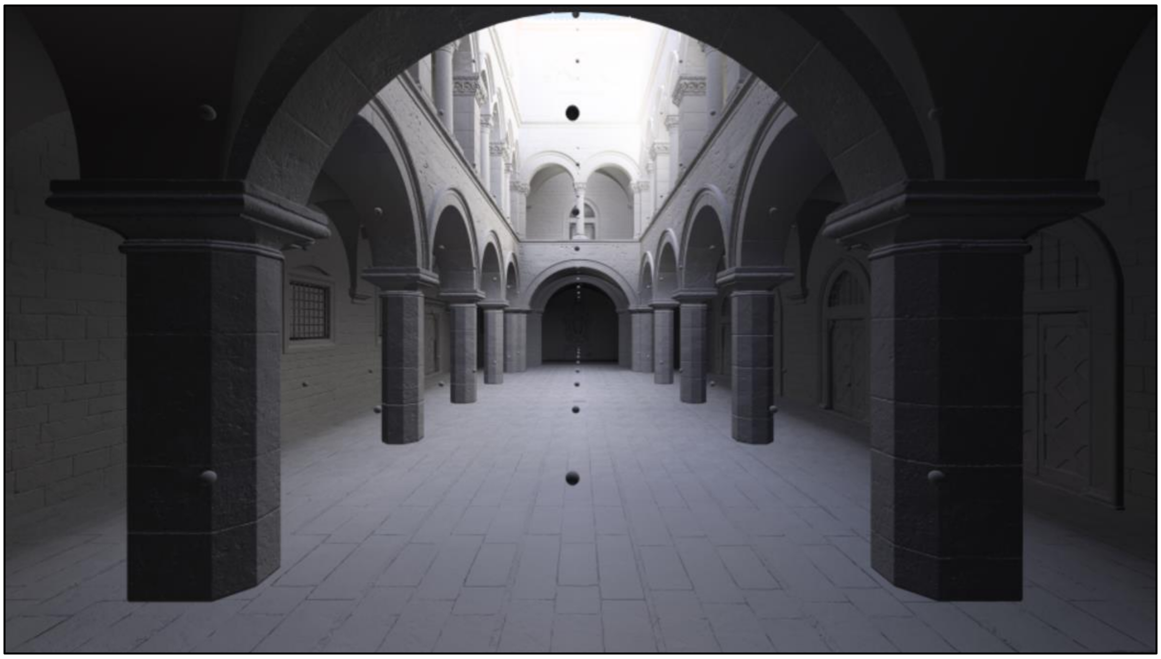

Generator network
First-pass representation
Weighting functions generated on a discrete set of points
Optional further refinement
Real-time rendering
x
y
z
w

This is the schematics of the complete system: we start with the weighting functions, we push them through the generator network to get the first-pass representation, which we can further refine if needed

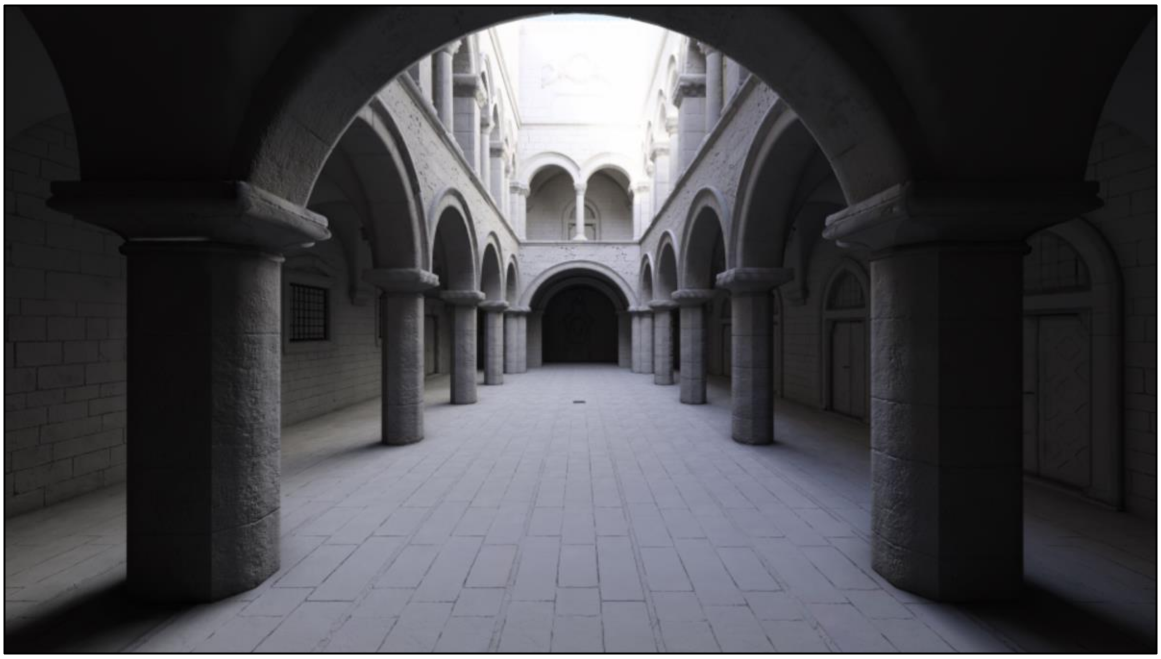
SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

RESULTS

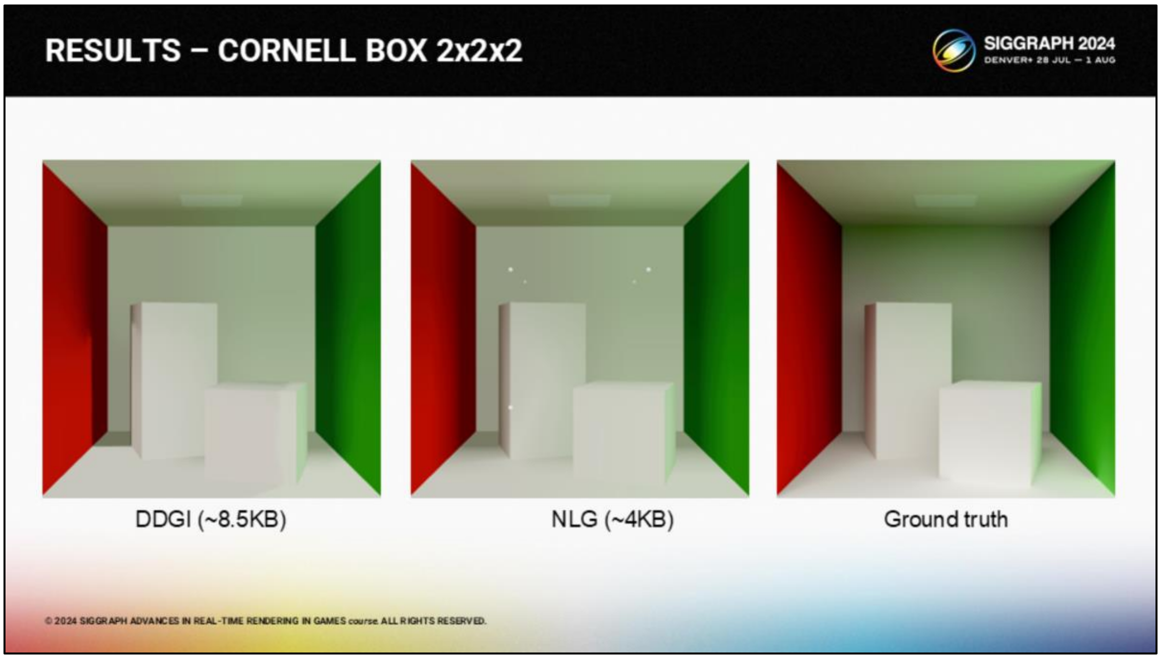
Now for some results



First the obligatory Sponza shot. This is evaluated per-pixel. Even though the density of the probes is really low, the results are still acceptable, there are no leaking artifacts

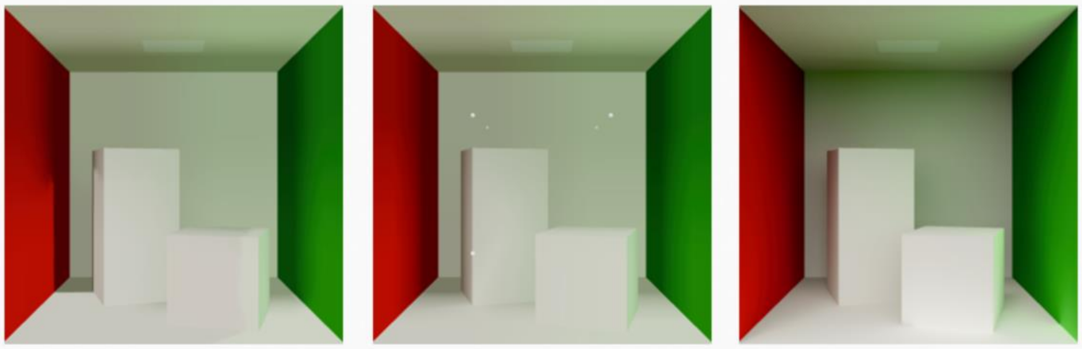


This is the ground truth for that, of course it’s much better quality, but given how little memory it takes, it’s not half bad

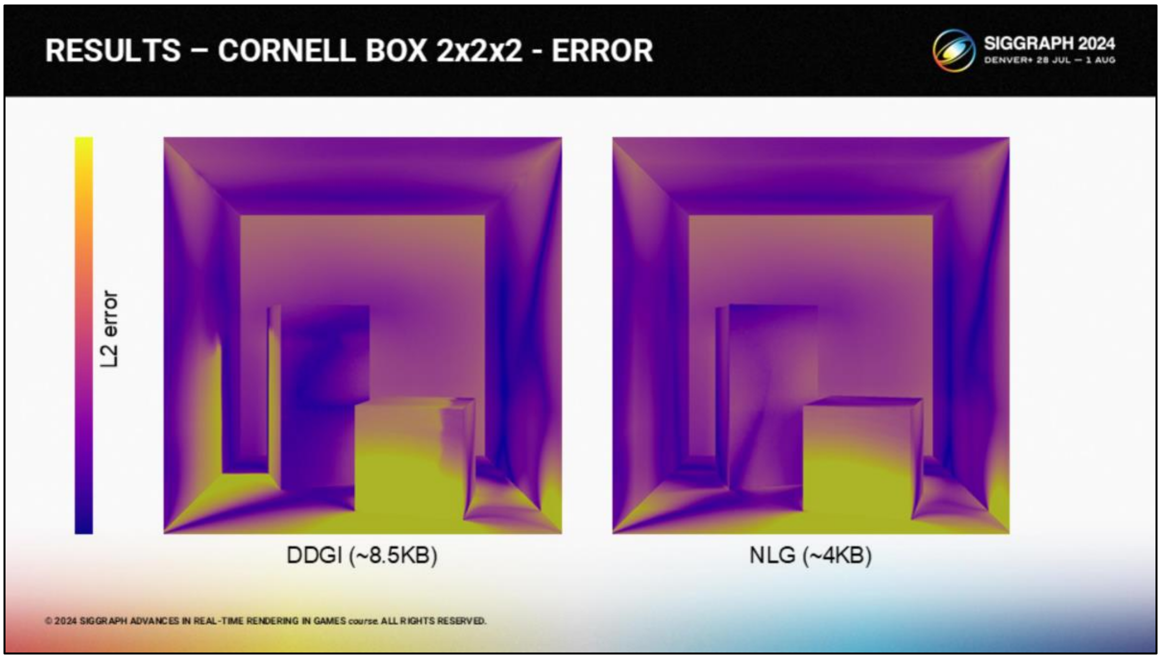


RESULTS – CORNELL BOX 2x2x2

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

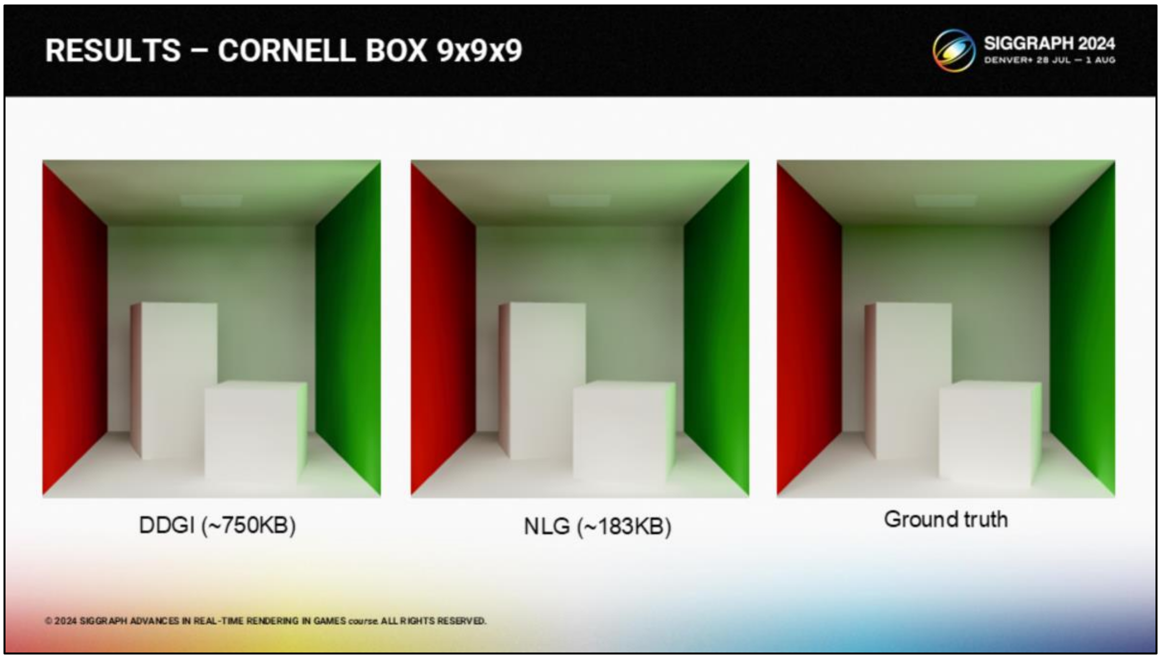

DDGI (~8.5KB) NLG (~4KB) Ground truth

Another compulsory test, Cornell Box, with 8 probes.
Here we compare it to the diffuse component of the DDGI, that relies on the low resolution, octahedral depth maps to discard occluded probes – the weighting functions are implicit, based on these depth buffers. At such a low probe density, it suffers from some discontinuity artifacts, despite using more memory



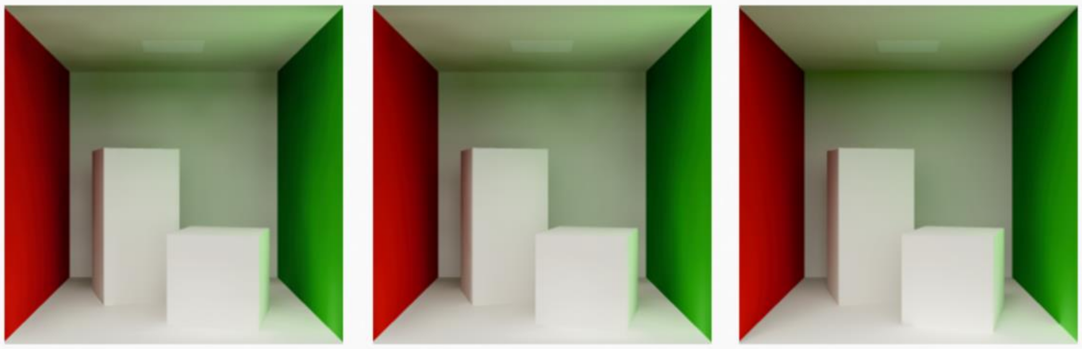
RESULTS – CORNELL BOX 2x2x2 - ERROR

Here are the error plots – in general they are pretty similar in terms of magnitude, but the NLG one changes smoothly which is an important characteristic

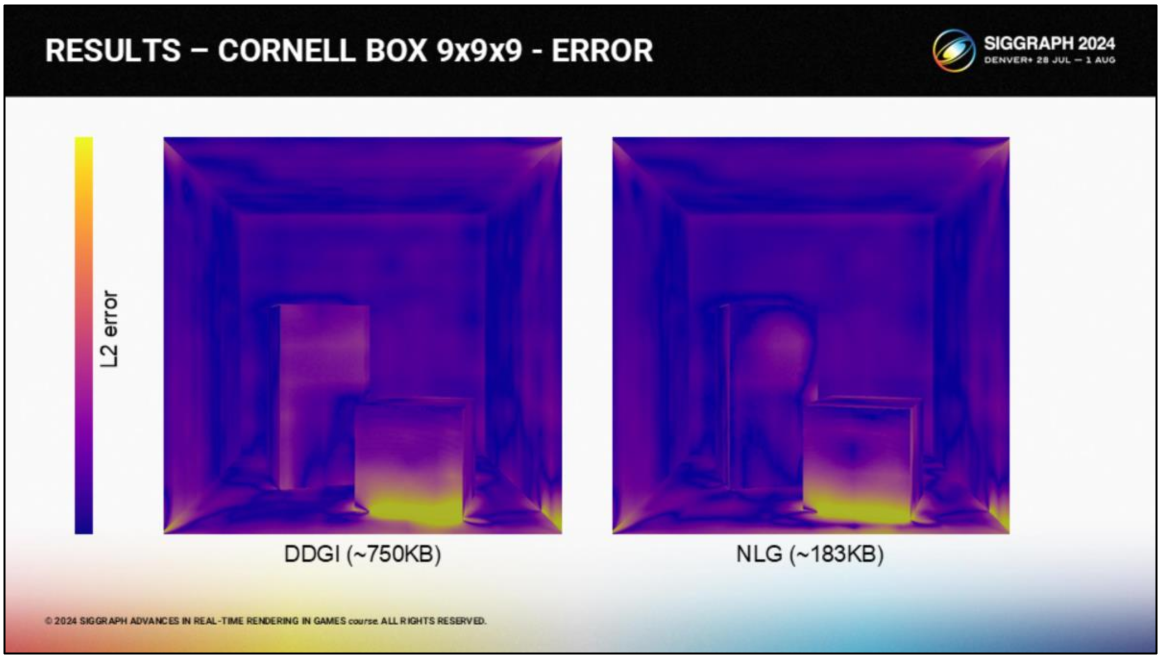


RESULTS – CORNELL BOX 9x9x9

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG



This is cornell box again, with much higher density of probes, this is 9x9x9 probes. NLG in such setup looks pretty much the same as DDGI, but uses much less memory



RESULTS – CORNELL BOX 9x9x9 - ERROR

Here are the error plots

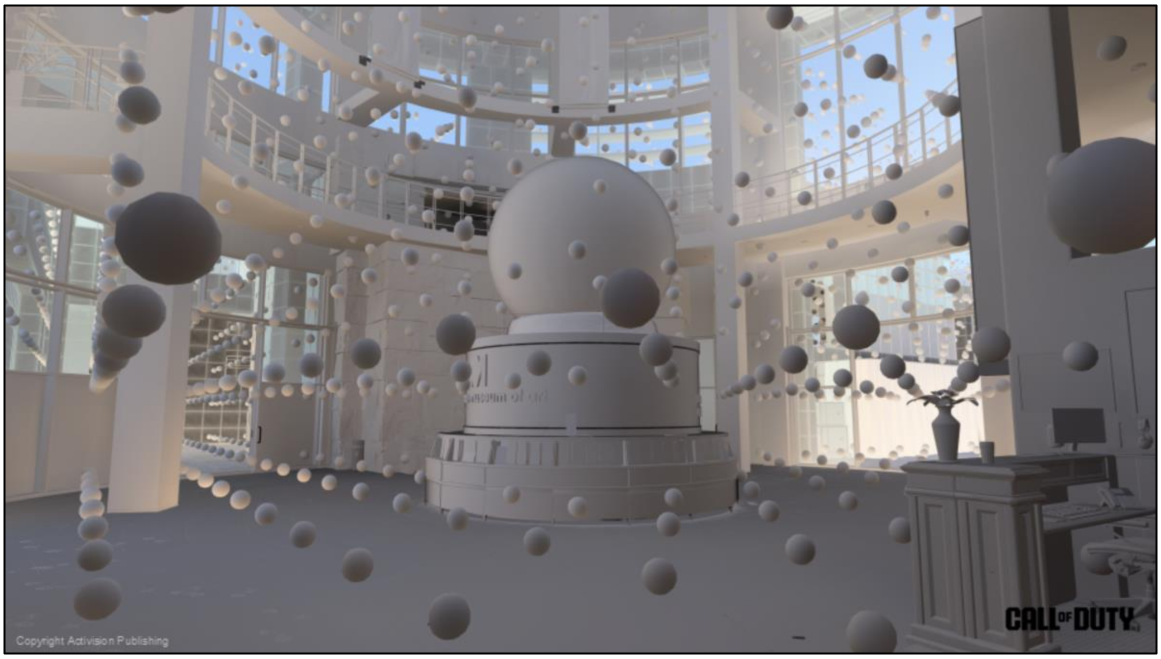
Copyright Activision Publishing
CALL OF DUTY

Here are some screenshots from shipped maps, for demonstration again rendered with per-pixel evaluation, which we don’t do.

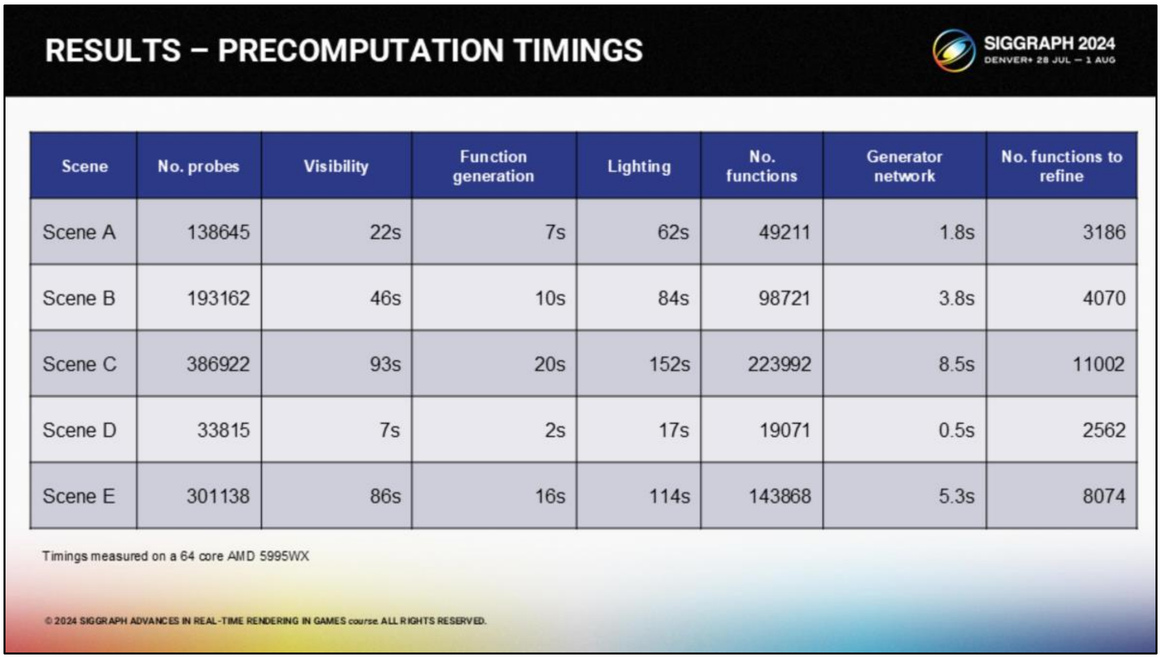
Copyright Activision Publishing
CALL OF DUTY

Copyright Activision Publishing

Copyright Activision Publishing
CALL OF DUTY



Here’s some other map, this time with the probes so you can see the density.



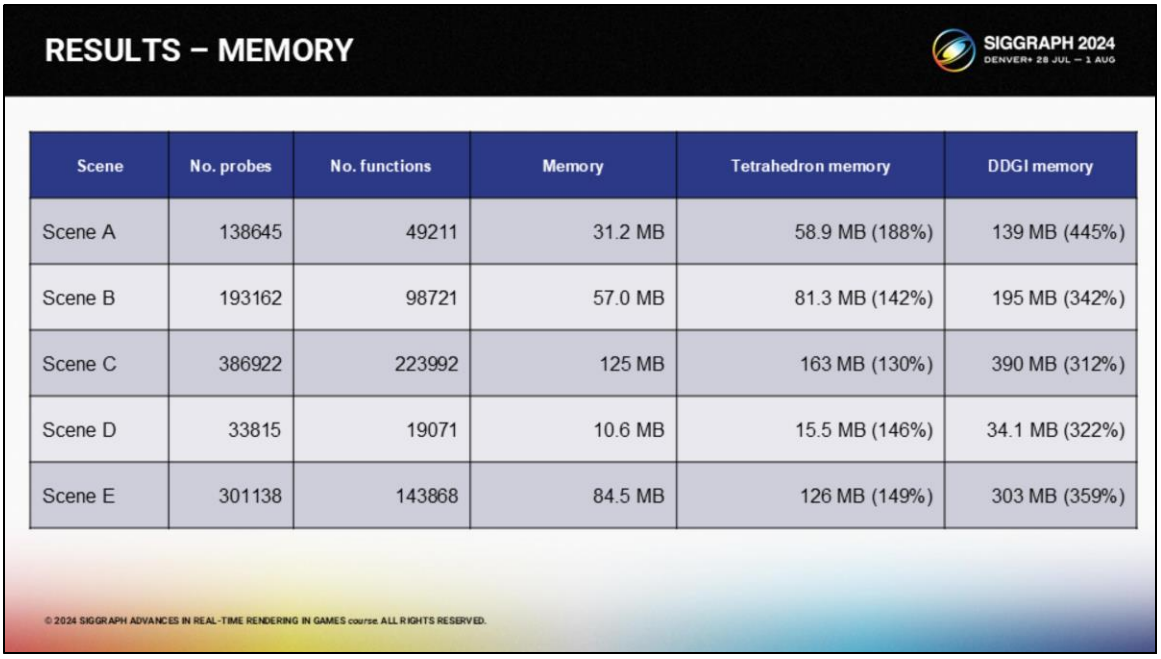
RESULTS – PRECOMPUTATION TIMINGS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

Scene	No. probes	Visibility	Function generation	Lighting	No. functions	Generator network	No. functions to refine
Scene A	138645	22s	7s	62s	49211	1.8s	3186
Scene B	193162	46s	10s	84s	98721	3.8s	4070
Scene C	386922	93s	20s	152s	223992	8.5s	11002
Scene D	33815	7s	2s	17s	19071	0.5s	2562
Scene E	301138	86s	16s	114s	143868	5.3s	8074

Timings measured on a 64 core AMD 5995WX

Here are some example precomputation times for a bunch of maps. The largest cost is lighting calculations, next visibility. We report the number of functions separately from the number of probes, since we run a de-dupping pass and combine identical weighting functions.
As you can see the generator network is extremely efficient way of getting the final representation, even though it's not always perfect.
On my pretty strong machine, it can process around 60 thousands probes per second, while direct optimization of the networks runs at around 50-60 probes per second on the same machine. The generator network is crucial part of the whole system, without it, it wouldn't be practical to do it in production.
And even though we have an enormous compute farm all this baking happens so often, and there's so much demand for that, that it's impossible for us to do this any other way than locally, on the lighters machine (just like the rest of the baking)

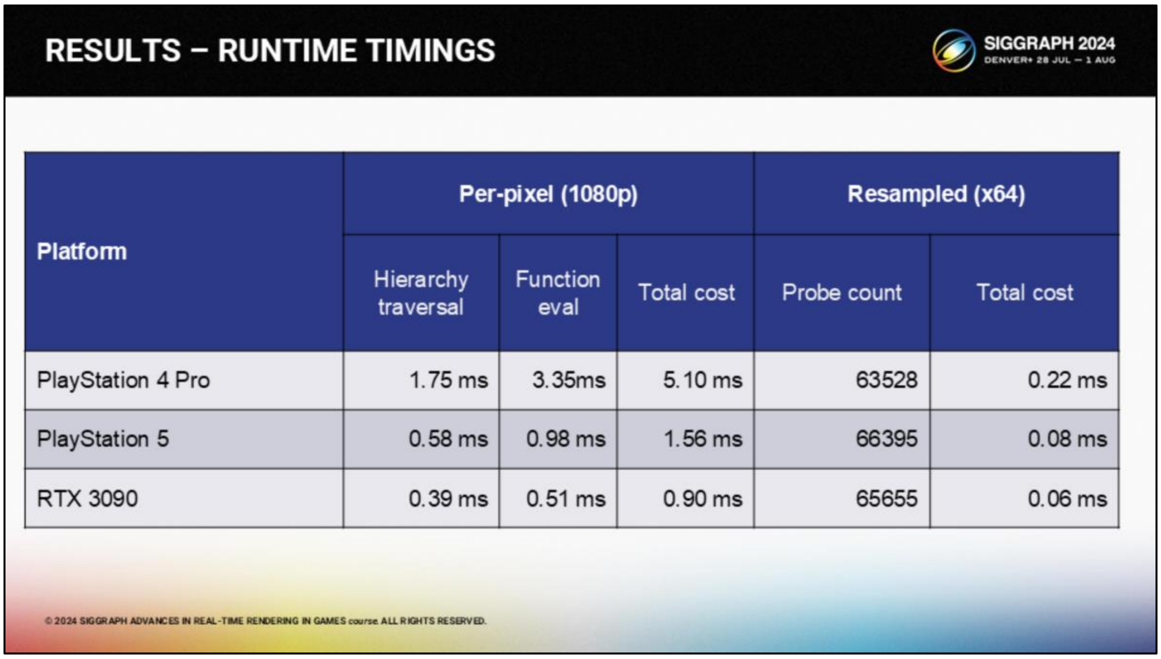


RESULTS – MEMORY

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

Scene	No. probes	No. functions	Memory	Tetrahedron memory	DDGI memory
Scene A	138645	49211	31.2 MB	58.9 MB (188%)	139 MB (445%)
Scene B	193162	98721	57.0 MB	81.3 MB (142%)	195 MB (342%)
Scene C	386922	223992	125 MB	163 MB (130%)	390 MB (312%)
Scene D	33815	19071	10.6 MB	15.5 MB (146%)	34.1 MB (322%)
Scene E	301138	143868	84.5 MB	126 MB (149%)	303 MB (359%)

Here are memory numbers for the same scenes. We usually save around 33% of the memory compared to out old system. DDGI is quite a bit more expensive memory wise, because every probe needs to store a unique 2 channel depth buffer, that totals around 1KB
The percentages are compared to the neural lightgrid memory usage.



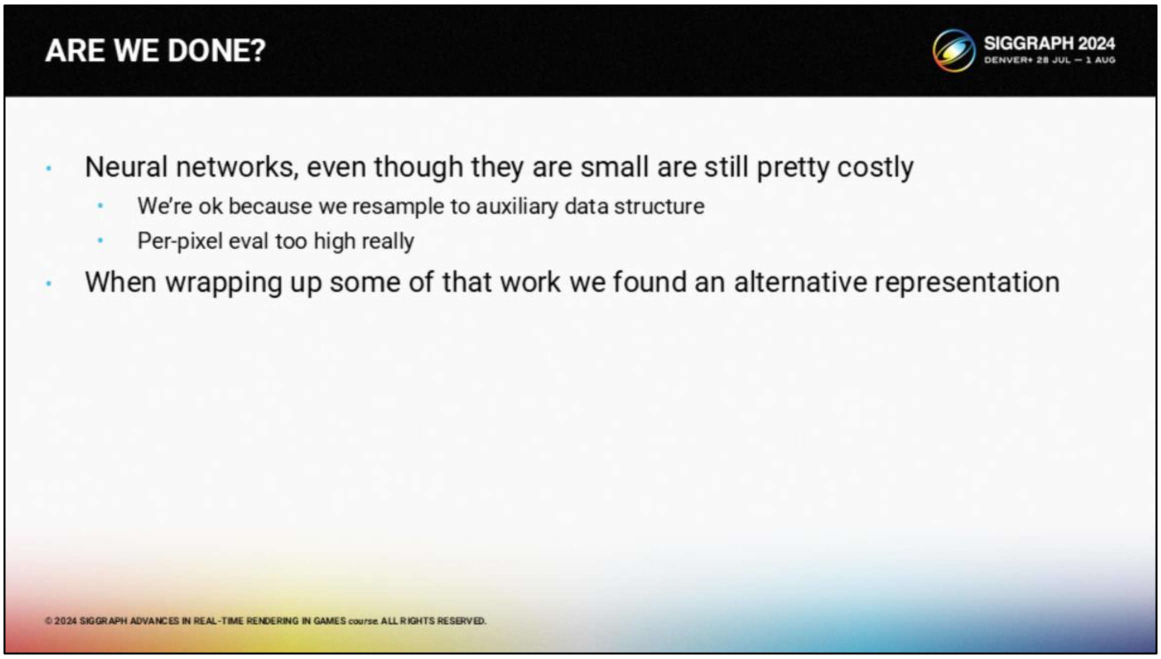
RESULTS – RUNTIME TIMINGS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

Platform	Per-pixel (1080p)			Resampled (x64)	
	Hierarchy traversal	Function eval	Total cost	Probe count	Total cost
PlayStation 4 Pro	1.75 ms	3.35ms	5.10 ms	63528	0.22 ms
PlayStation 5	0.58 ms	0.98 ms	1.56 ms	66395	0.08 ms
RTX 3090	0.39 ms	0.51 ms	0.90 ms	65655	0.06 ms

Here we have the performance number. First part of the table is for per-pixel evaluation. This is pretty costly, especially on the last gen Playstation 4, on newer platforms it becomes manageable. And a significant part of that cost if the hierarchy traversal, which is tied to some of our needs and could be significantly simplified to reduce it, so there is lots of headroom for improvement here, which we never looked into.

But like I said before, we don’t really evaluate this per pixel, we resample to a texture, so the number of lookups is much lower. Actually, to measure it reliably we had to crank it up by a factor of 64, otherwise the measurements were just lost in the noise.

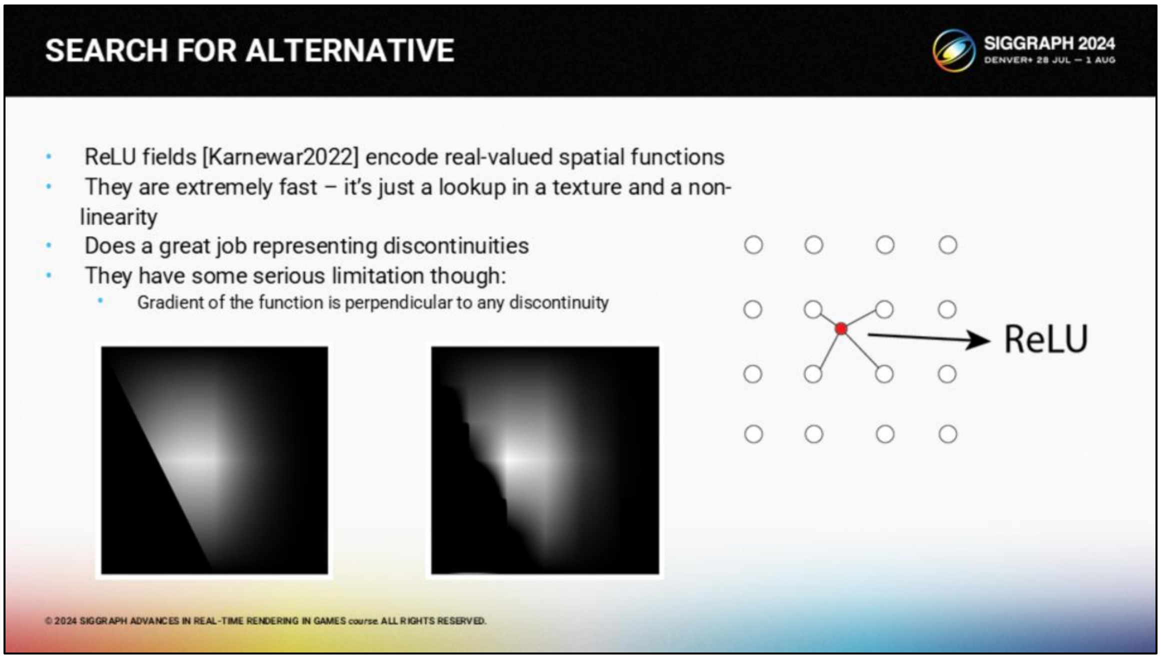


ARE WE DONE?

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Neural networks, even though they are small are still pretty costly
  - We're ok because we resample to auxiliary data structure
  - Per-pixel eval too high really
- When wrapping up some of that work we found an alternative representation

Altogether that forms a nice solution that we were pretty happy with.
The only thing that's kind-of not perfect is the lookup cost – it's a bit too high in general case. Since we resample to auxiliary data structure, we don't really mind, but it would be great if this could be use directly per-pixel too.
And when wrapping some things up, we actually found such representation



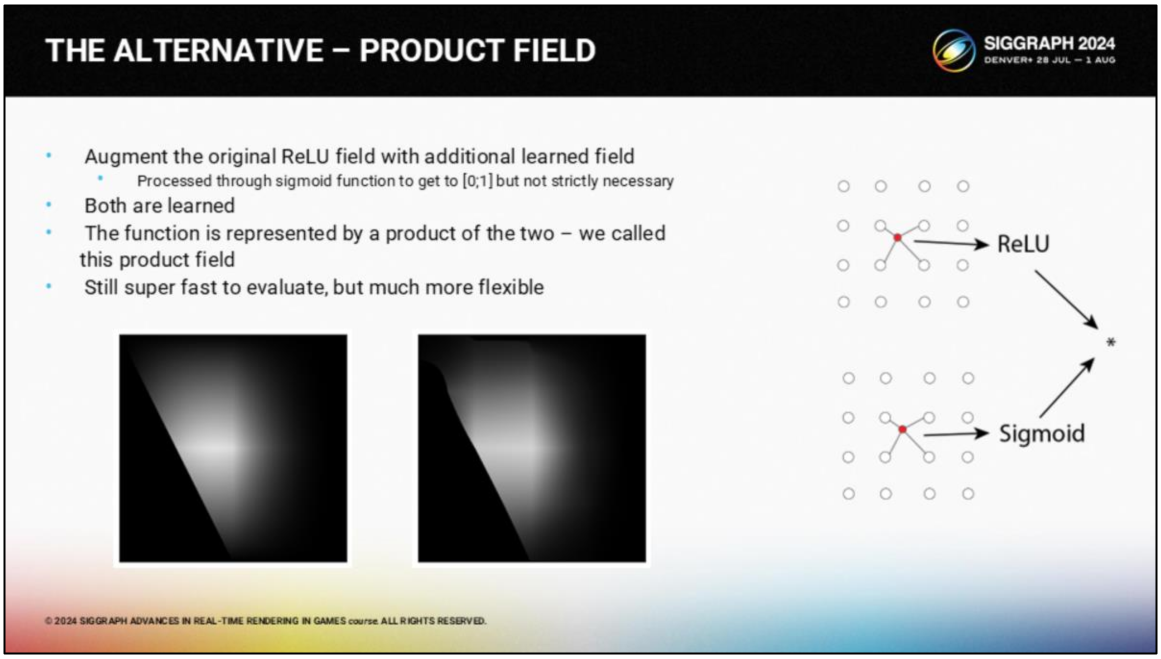
SEARCH FOR ALTERNATIVE

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- ReLU fields [Karnewar2022] encode real-valued spatial functions
- They are extremely fast – it's just a lookup in a texture and a non-linearity
- Does a great job representing discontinuities
- They have some serious limitation though:
  - Gradient of the function is perpendicular to any discontinuity

ReLU

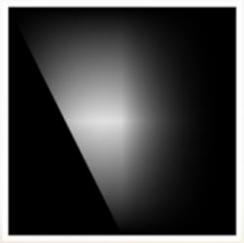
Two years ago, there was a paper on SIGGRAPH that introduced ReLU fields. It's a nice short paper to read if you have a bit of free time, but if you don't it's pretty much just a texture with floating point values that you saturate after the lookup, pretty much like an SDF. The values in that texture, the field, are learned to represent 1d functions. They are of course extremely fast to evaluate, and they are pretty good at representing discontinuities – so they seem like a nice fit for representing weighting functions.
Unfortunately, one of their intrinsic characteristic is that the gradient of the reconstructed function is always perpendicular to the discontinuity, so they do a poor job with things like the function on the top right – which is a 2d version of a typical weighting function.

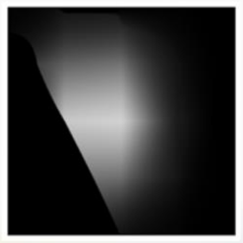


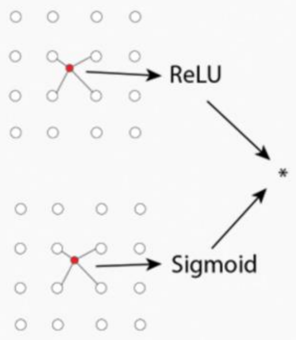
THE ALTERNATIVE – PRODUCT FIELD

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

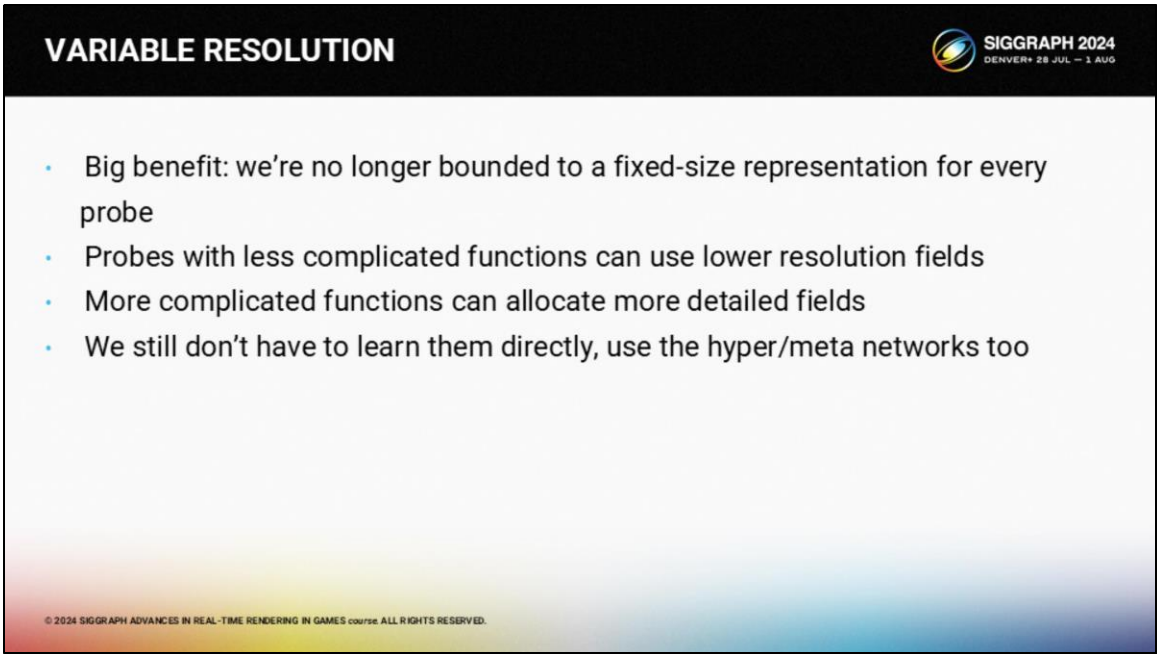
- Augment the original ReLU field with additional learned field
  - Processed through sigmoid function to get to [0;1] but not strictly necessary
- Both are learned
- The function is represented by a product of the two – we called this product field
- Still super fast to evaluate, but much more flexible







To fix this, we augment the ReLU field, with another, additional, learned field. We allow for arbitrary values in the other field and we process it with sigmoid function to get it into 0-1 range. It is not strictly necessary, but it stabilizes the learning process by unifying the learning rates between the two fields.
The output of this thing we called a “product field” is a product of these two components. It can represent both smooth functions as well as arbitrary discontinuities. The ReLU part mostly models hard edges, while the sigmoid deals with smooth area. It’s still super fast to evaluate but now also perfect as representation of our weighting functions.

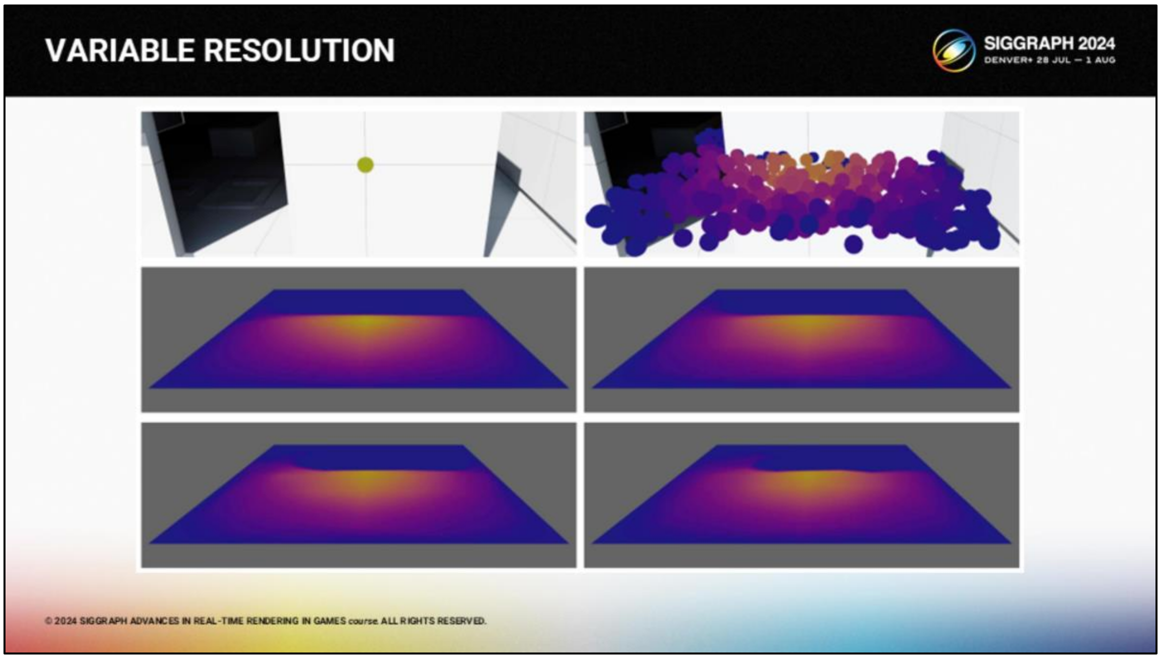


VARIABLE RESOLUTION

SIGGRAPH 2024 DENVER+ 28 JUL — 1 AUG

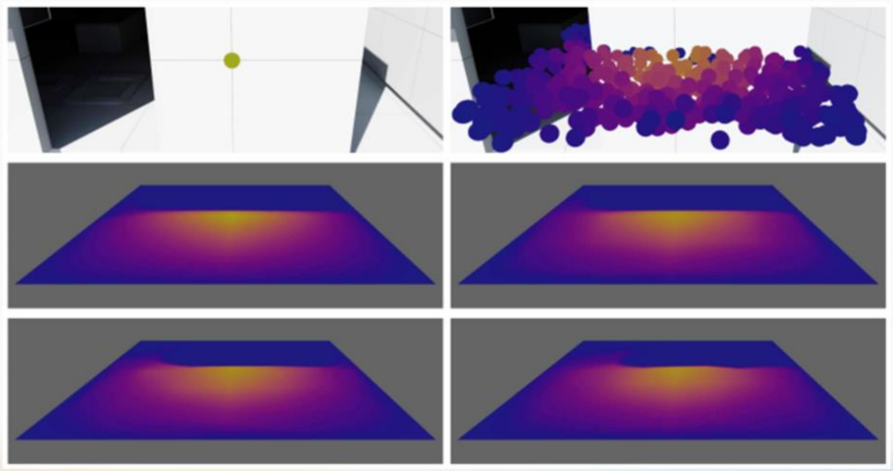
- Big benefit: we're no longer bounded to a fixed-size representation for every probe
- Probes with less complicated functions can use lower resolution fields
- More complicated functions can allocate more detailed fields
- We still don't have to learn them directly, use the hyper/meta networks too

One big advantage that the product field has over the network is that every function can use a product field with different resolution –complex functions can use higher resolution ones, simple functions can allocate smaller fields. It all still goes through the exact same code, it's just a matter of doing some scaling of the uvs before performing the lookup.

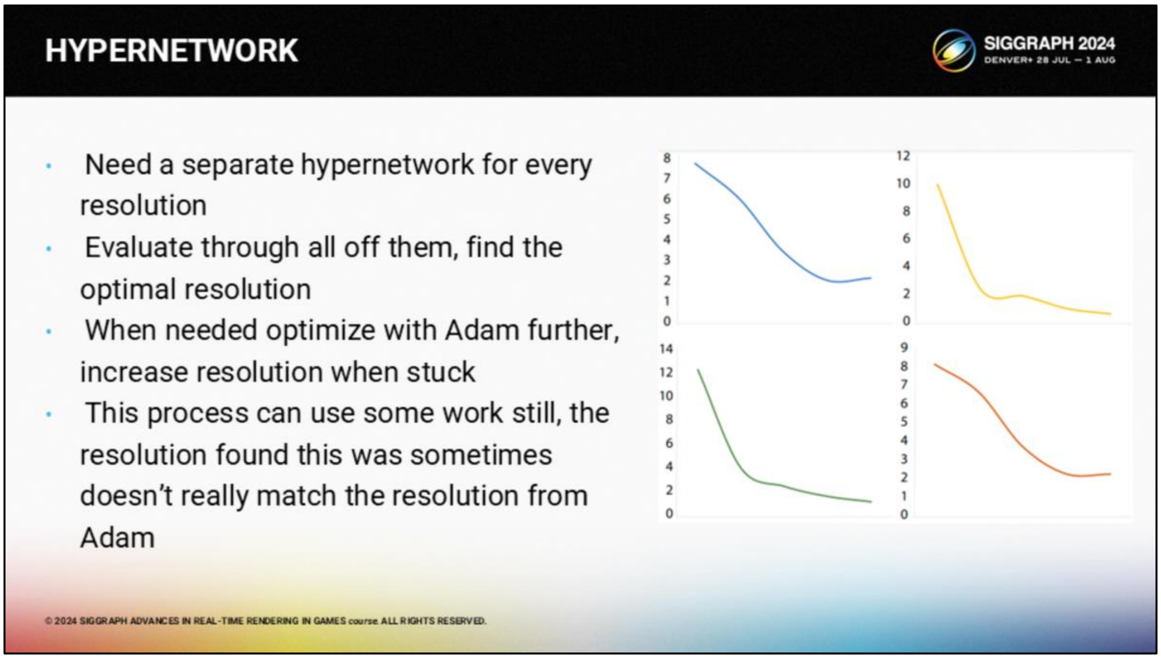


VARIABLE RESOLUTION

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG



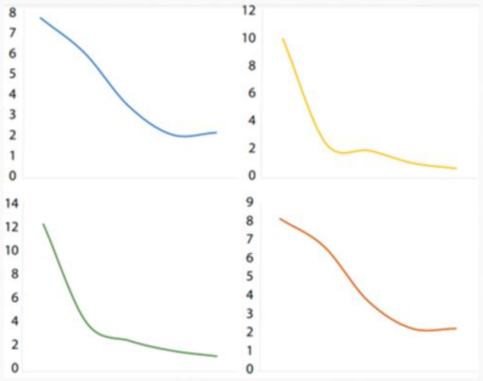
This is an example of some random function approximated with a product field of increasing resolution 3x3x3, 5x5x5, 7x7x7, 9x9x9



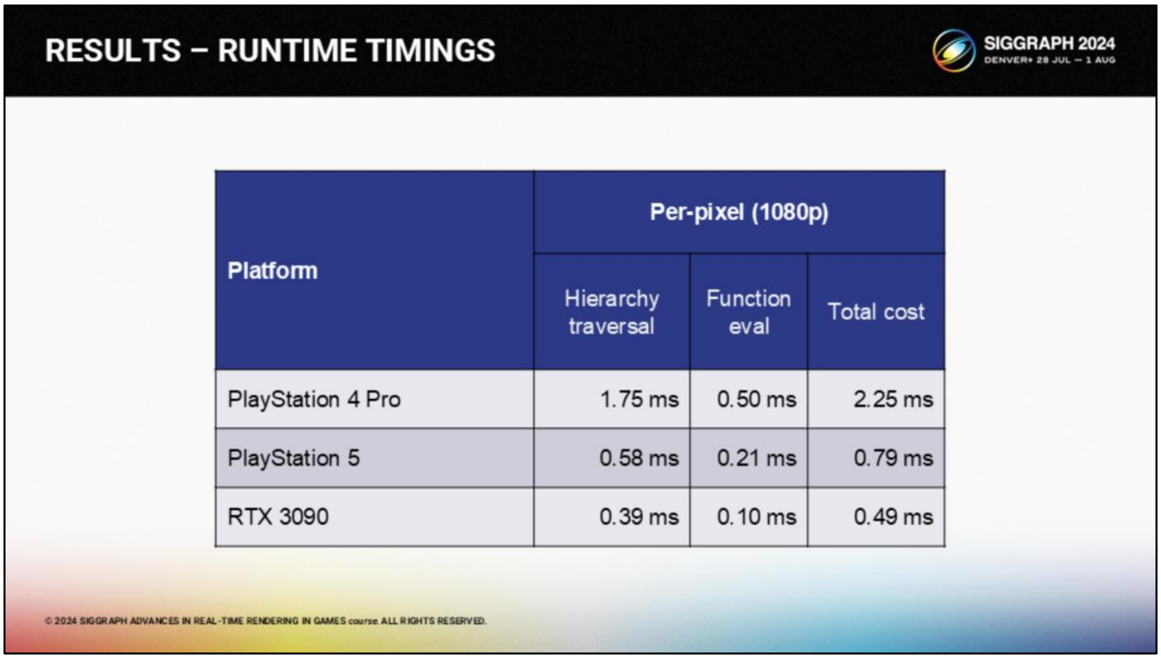
HYPERNETWORK

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Need a separate hypernetwork for every resolution
- Evaluate through all off them, find the optimal resolution
- When needed optimize with Adam further, increase resolution when stuck
- This process can use some work still, the resolution found this was sometimes doesn't really match the resolution from Adam



Everything else stays exactly the same, we only replace just the weighting function representation. We still have the same structure, same generation process.
Instead of a single generator network we now have 4 of them, and we evaluate them all to find the optimal resolutions. The plots on the right are the error plots of the approximation as a function of product field size. They generally flat out at some point, and this is the resolution that we pick.
This process still can use some work, as the resolution found through analyzing the outputs from the generator network sometimes doesn't match the resolution from direct optimization.

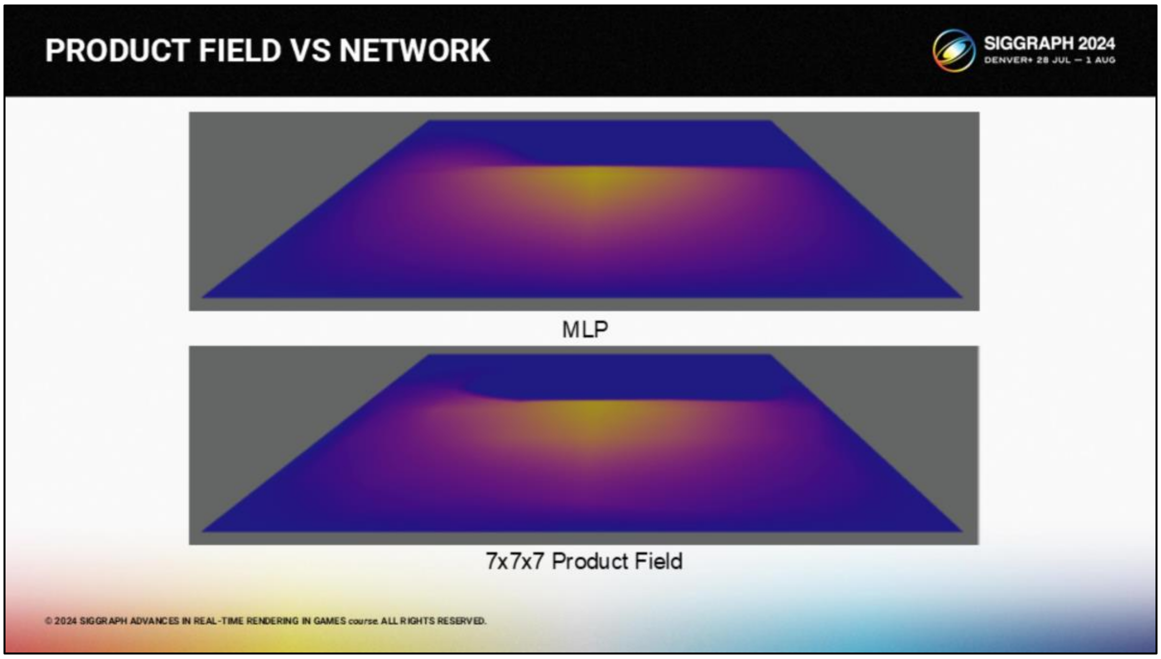


RESULTS – RUNTIME TIMINGS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

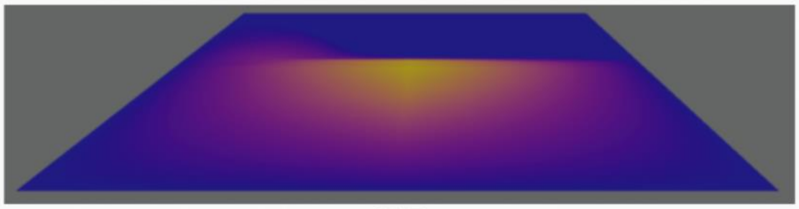
Platform	Per-pixel (1080p)		
	Hierarchy traversal	Function eval	Total cost
PlayStation 4 Pro	1.75 ms	0.50 ms	2.25 ms
PlayStation 5	0.58 ms	0.21 ms	0.79 ms
RTX 3090	0.39 ms	0.10 ms	0.49 ms

These are the timings for a full screen lookup of the product field version. As you might expect, hierarchy traversal is the dominant cost now, the actual function evaluation is only a small part of the overall cost, so if we were to use per-pixel we would definitely look for some improvements for the hierarchy.

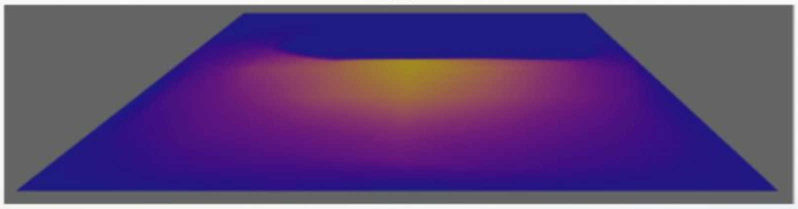


PRODUCT FIELD VS NETWORK

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

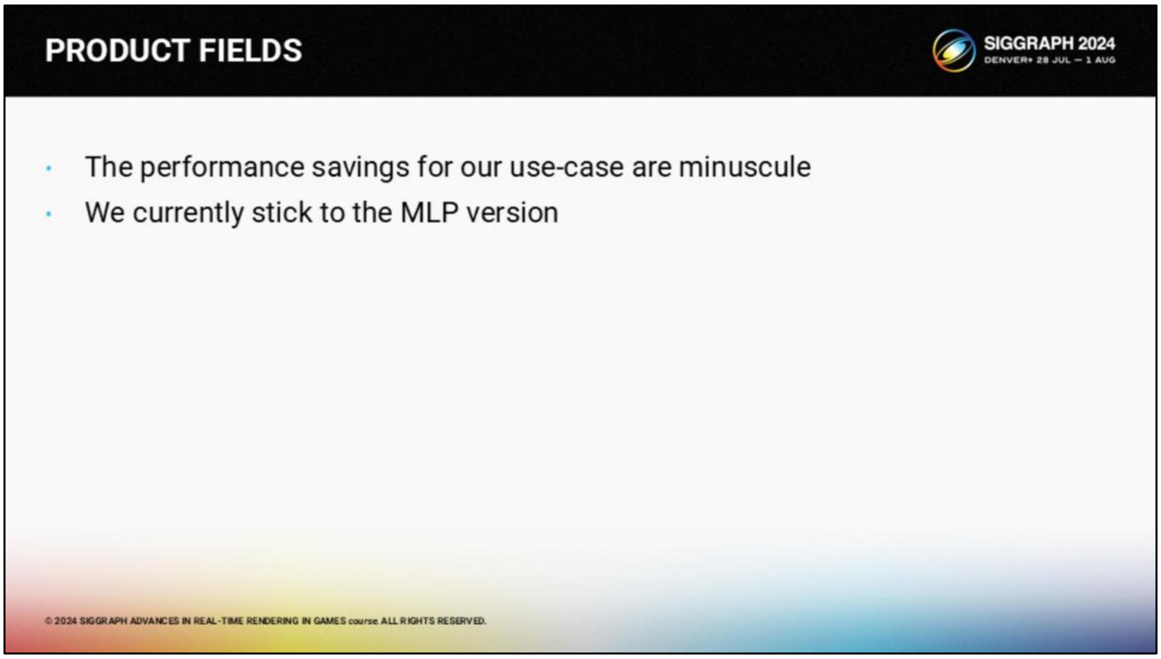


MLP



7x7x7 Product Field

This is a comparison of the product field and MLP representations.

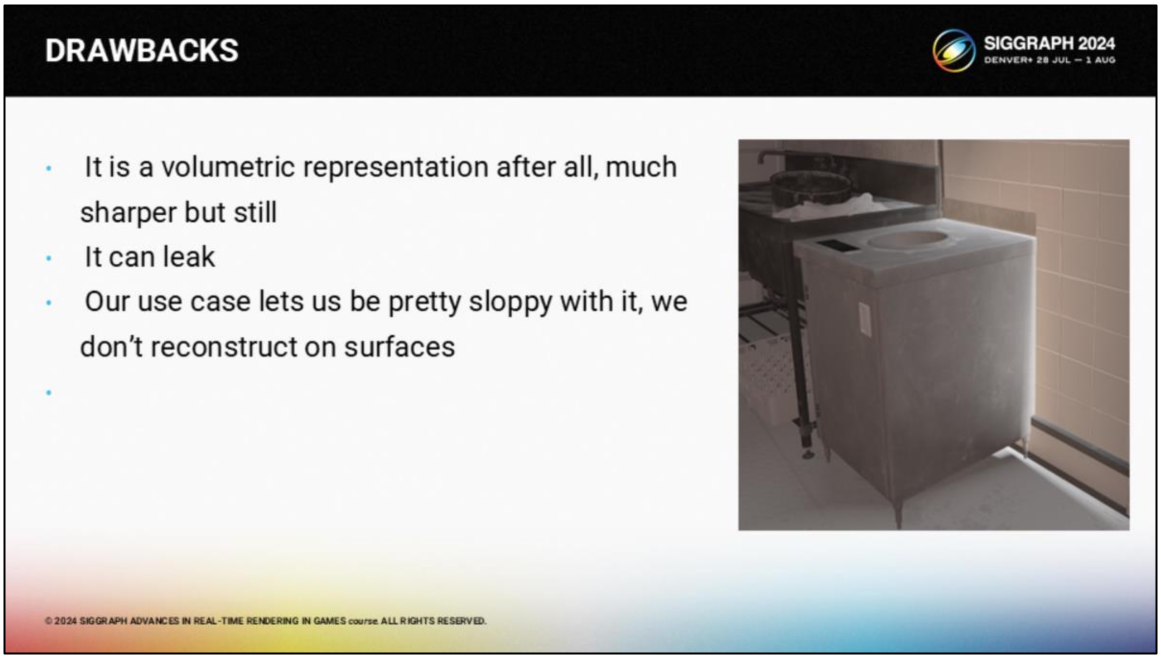


PRODUCT FIELDS

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- The performance savings for our use-case are minuscule
- We currently stick to the MLP version

As a final comment for that: we currently stick with the MLP version, as in our case the perf saving are not noticeable, and since moving to the product field is a non-zero production cost, we just never had any hard motivation.



DRAWBACKS

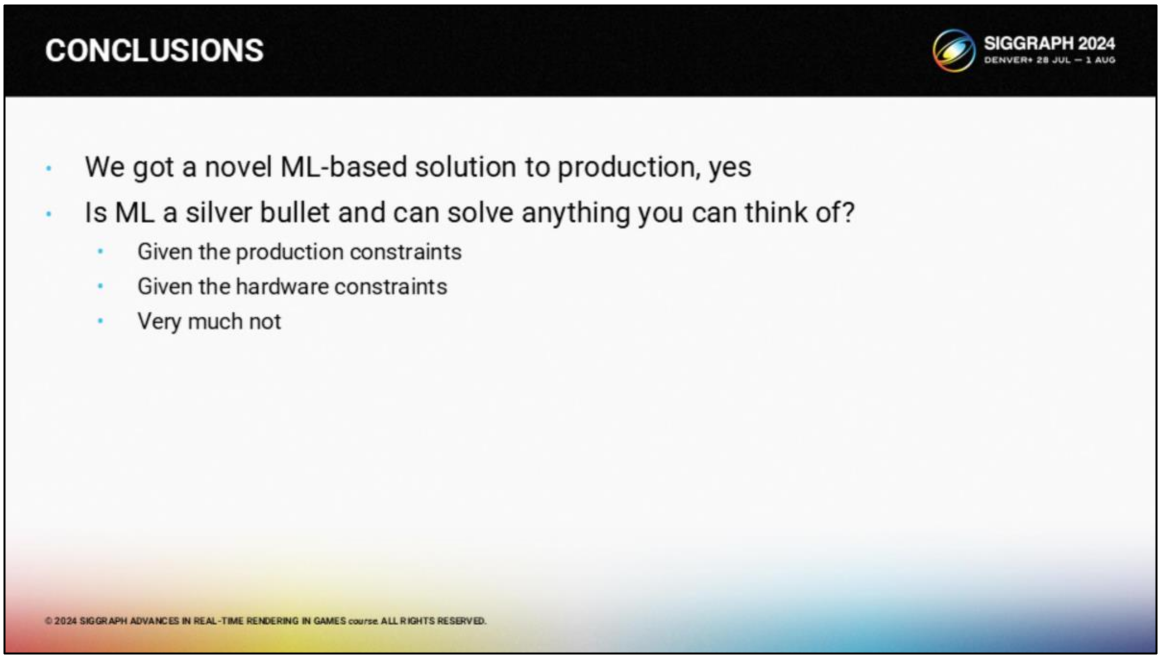
SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- It is a volumetric representation after all, much sharper but still
- It can leak
- Our use case lets us be pretty sloppy with it, we don't reconstruct on surfaces
-

And one slide with some drawbacks.
In the end it's still a volumetric representation, much sharper, but it can leak. For us it's not really a problem, since we resample and we don't really use values that are super close to wall, so we can be pretty sloppy. We can for instance afford to not to refinement of many probes, and just rely on the generator network to speed up the precomputation times – but I'm well aware that's not always the case. If you want to reconstruct per-pixel you need more precise representation, and you will need to spend extra time optimizing these representations to a higher precision.

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

CONCLUSIONS

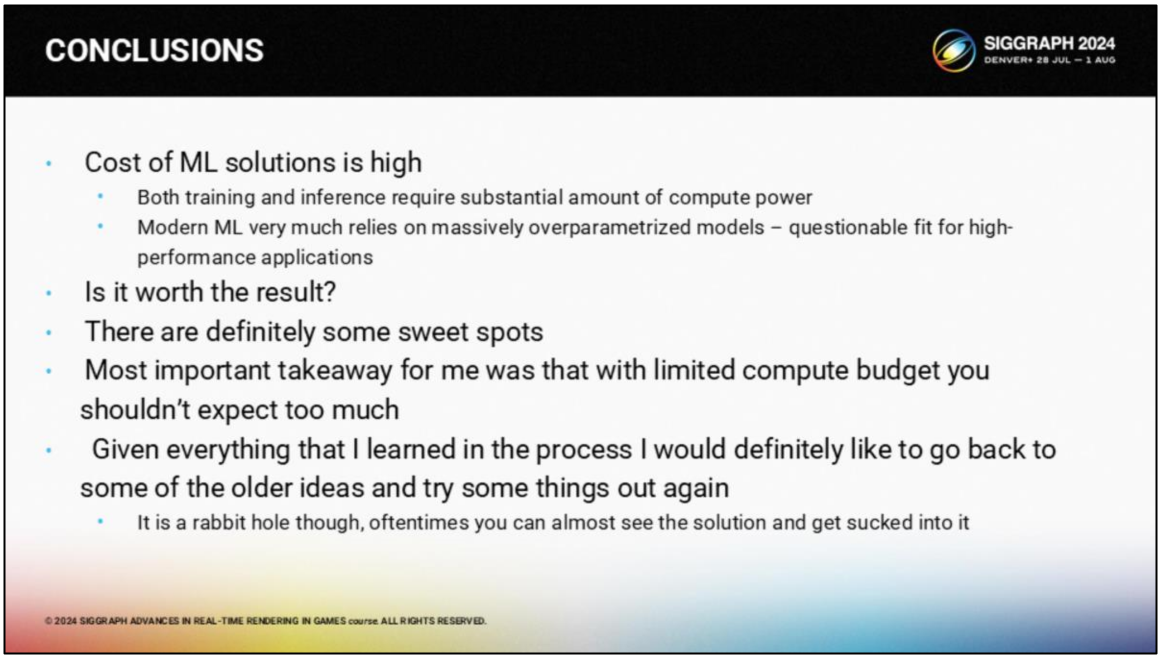


CONCLUSIONS

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- We got a novel ML-based solution to production, yes
- Is ML a silver bullet and can solve anything you can think of?
  - Given the production constraints
  - Given the hardware constraints
  - Very much not

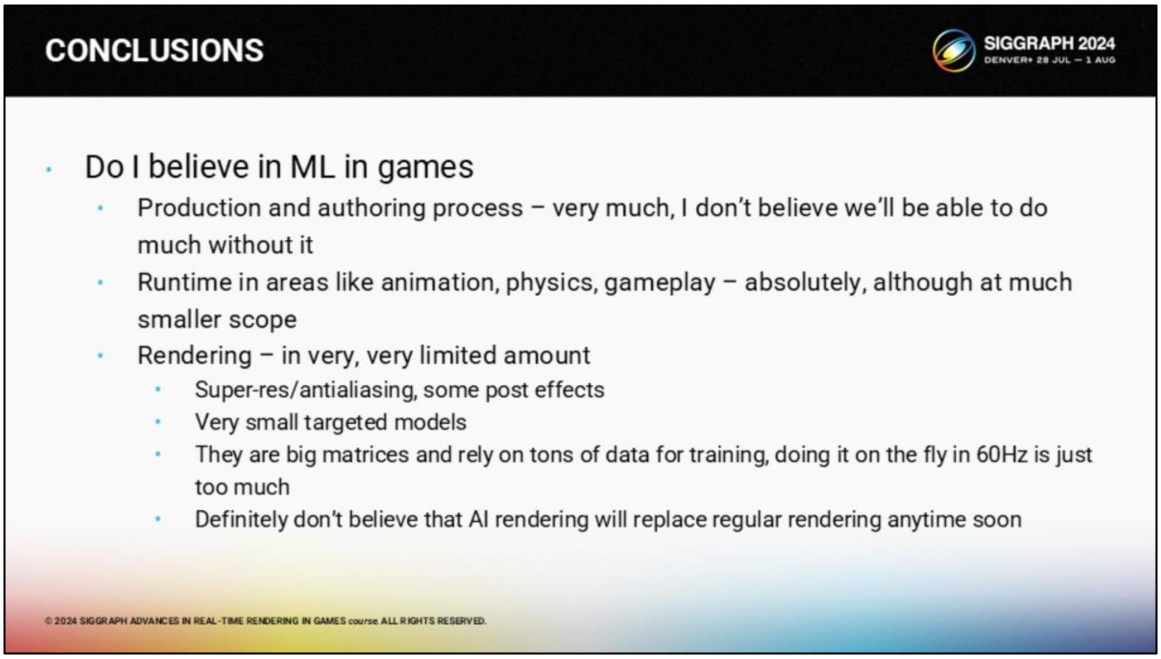
And that was our journey through getting ML solution in a shipped game. It did work, yes. Is it perfect, no. It did work for us, we’re happy with the result, but we have a very particular case, your milage may vary



CONCLUSIONS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

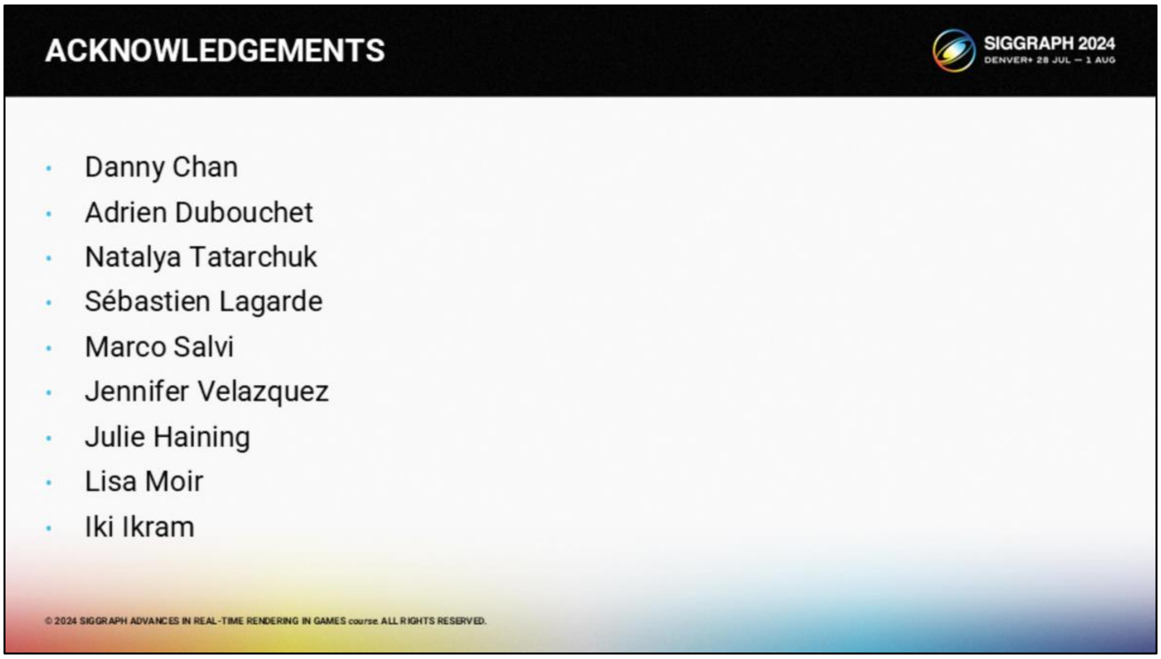
- Cost of ML solutions is high
  - Both training and inference require substantial amount of compute power
  - Modern ML very much relies on massively overparametrized models – questionable fit for high-performance applications
- Is it worth the result?
- There are definitely some sweet spots
- Most important takeaway for me was that with limited compute budget you shouldn’t expect too much
- Given everything that I learned in the process I would definitely like to go back to some of the older ideas and try some things out again
  - It is a rabbit hole though, oftentimes you can almost see the solution and get sucked into it



CONCLUSIONS

SIGGRAPH 2024
DENVER+ 28 JUL – 1 AUG

- Do I believe in ML in games
  - Production and authoring process – very much, I don't believe we'll be able to do much without it
  - Runtime in areas like animation, physics, gameplay – absolutely, although at much smaller scope
  - Rendering – in very, very limited amount
    - Super-res/antialiasing, some post effects
    - Very small targeted models
    - They are big matrices and rely on tons of data for training, doing it on the fly in 60Hz is just too much
    - Definitely don't believe that AI rendering will replace regular rendering anytime soon



ACKNOWLEDGEMENTS

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

- Danny Chan
- Adrien Dubouchet
- Natalya Tatarchuk
- Sébastien Lagarde
- Marco Salvi
- Jennifer Velazquez
- Julie Haining
- Lisa Moir
- Iki Ikram

SIGGRAPH 2024
DENVER+ 28 JUL — 1 AUG

Q&A