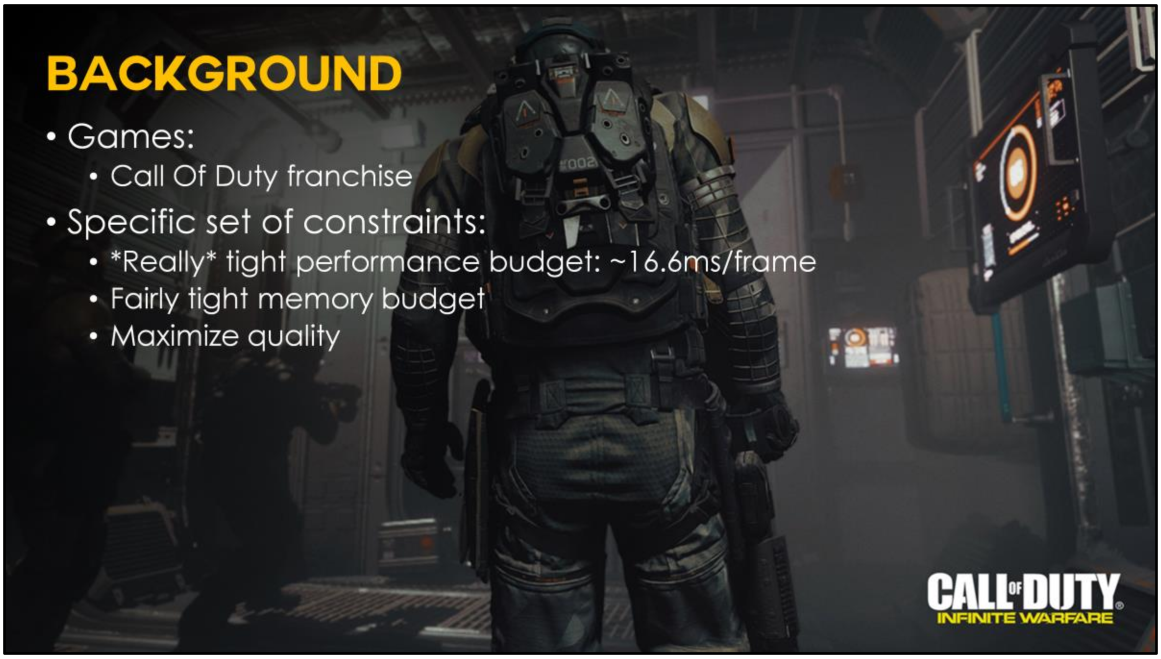
AMBIENT DICE

Michał Iwanicki & Peter-Pike Sloan

Activision Central Tech

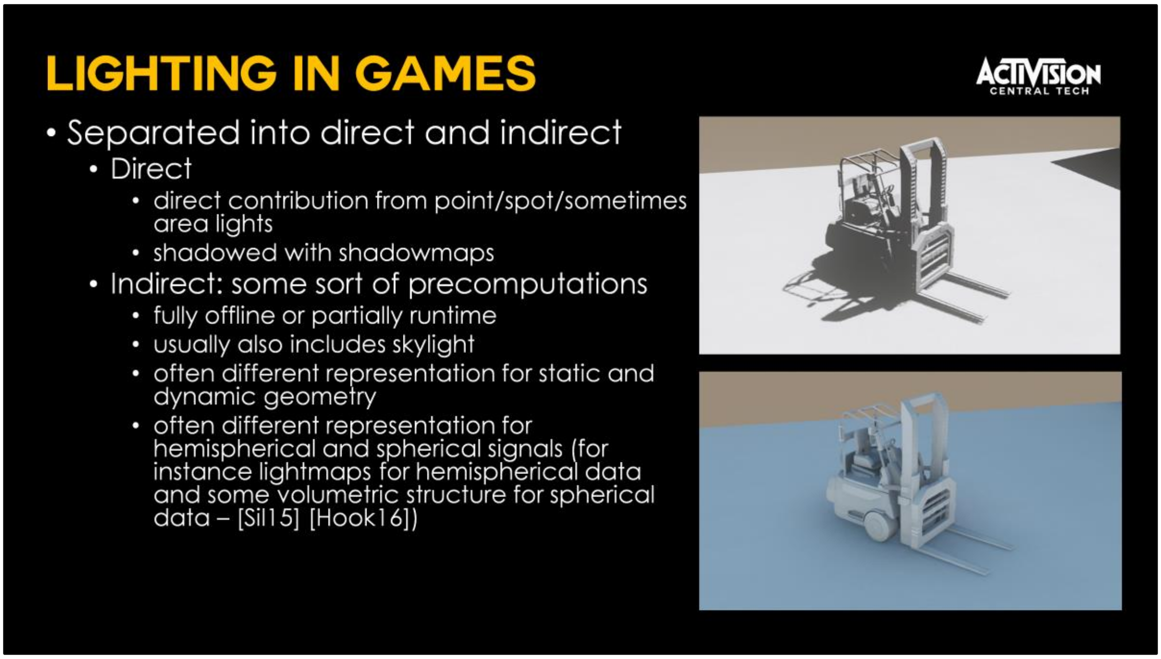
Hi everyone, my name is Michal Iwanicki.
I’m here today with with Peter-Pike Sloan and I’m going to talk about a family of new spherical basis function that we call Ambient Dice (AD).



BACKGROUND

- Games:
  - Call Of Duty franchise
- Specific set of constraints:
  - *Really* tight performance budget: ~16.6ms/frame
  - Fairly tight memory budget
  - Maximize quality

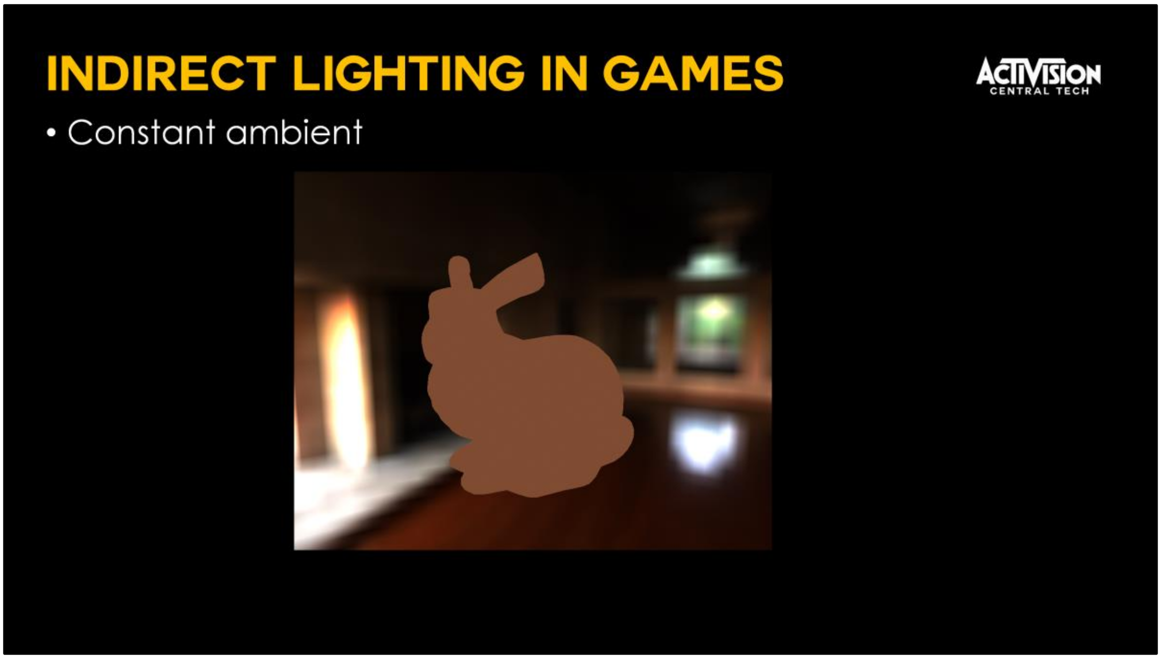
Something that I really want to cover today is not only the details of the research itself, but also what were the motivations for certain choices that we made, what were the constraints on our solution.
Because we work in games, we work for Activision Central Tech, doing R&D for various studios, most notably for the Call Of Duty franchise.
And even though it is “R&D” it’s mostly driven by the “D“ part – which means that we’re doing the research for a particular title, it needs to be ready on time, we are actually responsible for putting it in the final game, and it needs to run within a very strict performance and memory constraints



LIGHTING IN GAMES

- Separated into direct and indirect
  - Direct
    - direct contribution from point/spot/sometimes area lights
    - shadowed with shadowmaps
  - Indirect: some sort of precomputations
    - fully offline or partially runtime
    - usually also includes skylight
    - often different representation for static and dynamic geometry
    - often different representation for hemispherical and spherical signals (for instance lightmaps for hemispherical data and some volumetric structure for spherical data – [Sil15] [Hook16])

Most recently we've been working on lighting, so I will give you a brief introduction to lighting in games.
Traditionally, game lighting is separated into two parts that are handled separately.
First, there is direct lighting, with contributions from traditional point, spot, and recently area lights, which are often shadowed with some form of shadowmaps.
Then there is the indirect lighting, which is different in that it usually involves some sort of precomputations, storing the results in some data structures, and then resampling the results during the actual rendering.
It's generally computed at a spatial frequency that's unrelated to the actual final rendered image.
The indirect lighting data usually comes in two flavors: there are hemispherical signals that describe lighting on surfaces, and fully spherical signals for describing lighting in space.
We often use different methods for representing the two.



INDIRECT LIGHTING IN GAMES

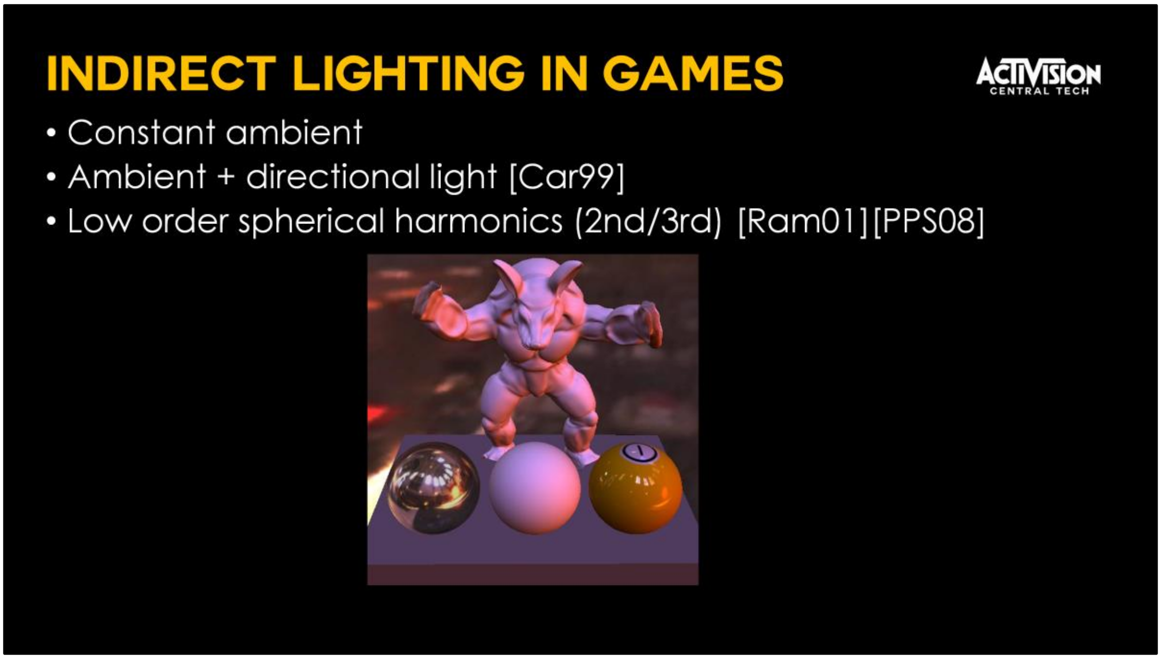
- Constant ambient

We’ll focus on indirect lighting here. To be even more precise, we will focus on the diffuse contribution and how it is stored and represented, and we want it to represent a full spherical signal.
There are a number of solutions used in games.
I don’t really think we can say that this is actually a solved problem.
The solutions range from constant ambient which looks bad, to a combination of ambient and diffuse light.
The latter does a reasonable job when used to represent hemispherical signal, but fails miserably when representing spherical signal, generating those pee-stain like artifacts when the direction of the light changes rapidly.
Then there are low order spherical harmonics (SH): second and third order.
The problem with the second is that the quality is not really enough, and the third order is slightly too expensive.
Recently, people have also been using sets of spherical Gaussian (SG) lobes, which are fairly expensive, but have the advantage in that they can do a reasonable job representing radiance for low gloss specular materials.
And then there’s Ambient Cube (AC), which is a set of 6 clamped cos^2 lobes oriented along cardinal axes.
The quality they offer is fairly low, but it is really cheap.

INDIRECT LIGHTING IN GAMES

ACTIVISION CENTRAL TECH

- Constant ambient
- Ambient + directional light [Car99]



INDIRECT LIGHTING IN GAMES

- Constant ambient
- Ambient + directional light [Car99]
- Low order spherical harmonics (2nd/3rd) [Ram01][PPS08]

INDIRECT LIGHTING IN GAMES

ACTIVISION
CENTRAL TECH

- Constant ambient
- Ambient + directional light [Car99]
- Low order spherical harmonics (2nd/3rd) [Ram01][PPS08]
- Spherical Gaussians [Tsai06][Wang07][RAD15]

All-Frequency Precomputed Radiance Transfer using Spherical Radial Basis Functions and Clustered Tensor Approximation

Yu-Ting Tsai* Zen-Chung Shih†

Department of Computer Science, National Chiao Tung University, Taiwan

Abstract

This paper introduces a new data representation and compression technique for *precomputed radiance transfer* (PRT). The light transfer functions and light sources are modeled with *spherical radial basis functions* (SRBFs). A SRBF is a rotation-invariant func-

equency Rendering of Dynamic, Spatially-Va

Jiaping Wang* Peiran Ren† * Minmin Gong* John Snyde

*Microsoft Research Asia †Tsinghua University ‡Microso

Figure 1: Real-time rendering results of our system. (a) Plastic teapot with spatially-varying Cook-Torrance model. (b) Iron dragon with spatially-varying Ward model. (c) Brushed metal dish and balls with spatially-varying Ashikhmin-Shirley model. (d) Red satin cloth and greeting card with measured SVBRDFs.

Rendering The Alternate History of The Order: 1886

Matt Pettineo – Ready At Dawn Studios

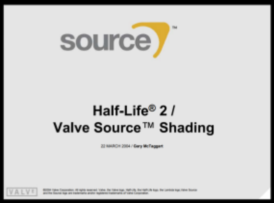
Advances in Realtime Rendering In Games 2015

Rendering The Alternate History of The Order: 1886 | SIGGRAPH 2015

INDIRECT LIGHTING IN GAMES

ACTIVISION
CENTRAL TECH

- Constant ambient
- Ambient + directional light [Car99]
- Low order spherical harmonics (2nd/3rd) [Ram01][PPS08]
- Spherical Gaussians [Tsai06][Wang07][RAD15]
- Ambient Cube [McTag04] [Hook16]



source™

Half-Life® 2 /
Valve Source™ Shading

22 MARCH 2004 / Gary McTaggart

VALVE

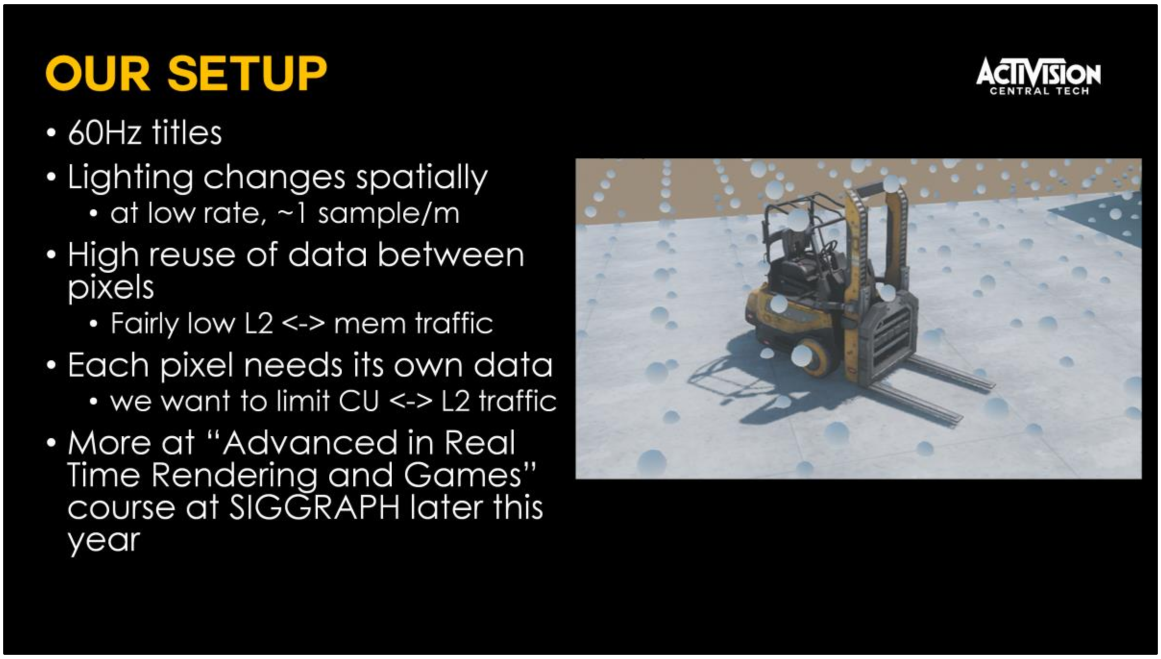
ACTIVISION.
treyarch

Render the Possibilities
SIGGRAPH2016

Volumetric
Global Illumination
At Treyarch

JT Hooker
Treyarch Senior Graphics Engineer

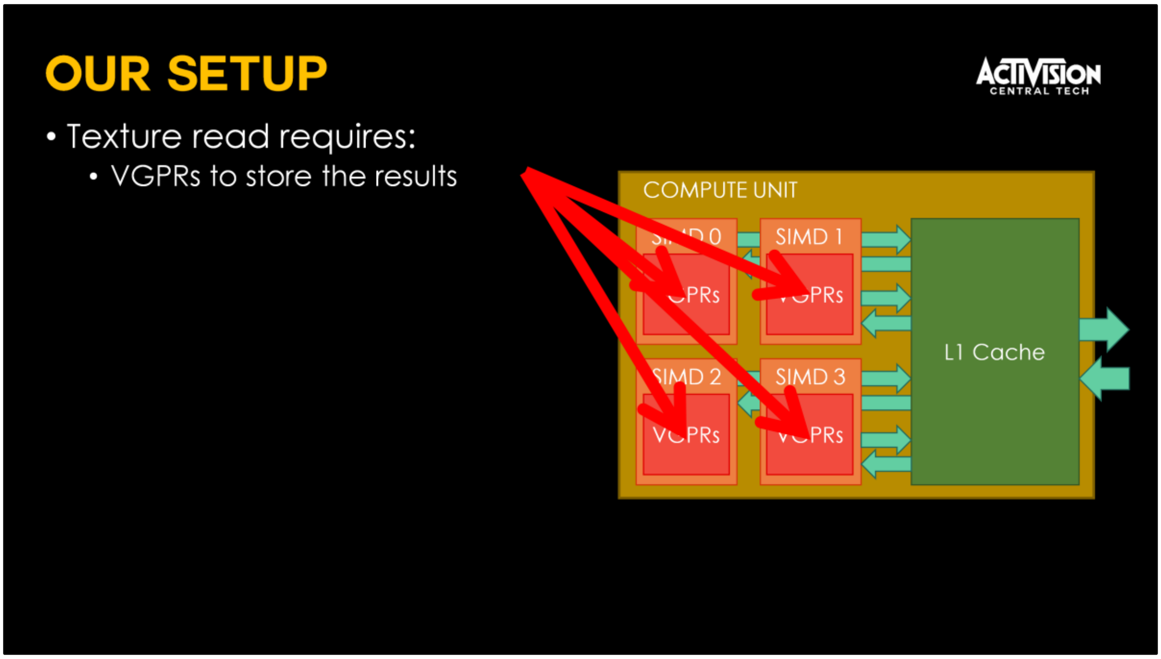
Advances in Real-Time Rendering course, SIGGRAPH 2016



OUR SETUP

- 60Hz titles
- Lighting changes spatially
  - at low rate, ~1 sample/m
- High reuse of data between pixels
  - Fairly low L2 <-> mem traffic
- Each pixel needs its own data
  - we want to limit CU <-> L2 traffic
- More at “Advanced in Real Time Rendering and Games” course at SIGGRAPH later this year

We need something that is really, really cheap.
Most of our games need to run at 60Hz, which means we have ~16.6 ms to render a whole frame.
This gives us around 7-8ms to do the actual scene rendering.
The rest is spent on shadows, special effects, etc.
At the same time, we dont want to compromise quality.
For instance, we want the lighting on objects to change spatially.
But since it’s only indirect lighting, we dont really need it to change at very high rate.
A single sample every meter is roughly the density that we are aiming for.
This means that when rendering objects we will need to use some sort of volumetric structure to store the lighting.
Neighboring pixels will use mostly the same pieces of that structure.
They will be accessing lighting data from roughly the same areas of memory.
The traffic between memory and L2 cache will be relatively small.
But on the other hand, it *will* be different for every pixel.
Every pixel *will* need to use slightly different data, and one of the main goals for us was to limit that amount of data.



OUR SETUP

- Texture read requires:
  - VGPRs to store the results

COMPUTE UNIT

SIMD 0

VGPRs

SIMD 1

VGPRs

SIMD 2

VGPRs

SIMD 3

VGPRs

L1 Cache

Why? Because accessing memory from a shader is really costly in terms of the resources it needs.

The diagram here shows a simplified view of a compute unit on both of the current generation consoles.

To access memory we need to allocate some registers to store the results.

It needs to send the actual fetch request to the texture unit, where it takes some amount of space in the internal queues.

Then there is the bandwidth utilization between the compute unit and L2.

Our main problem is that even without doing anything, just by using constant ambient, all the other features that we have in the shader cause us to be limited by the register counts that we use.

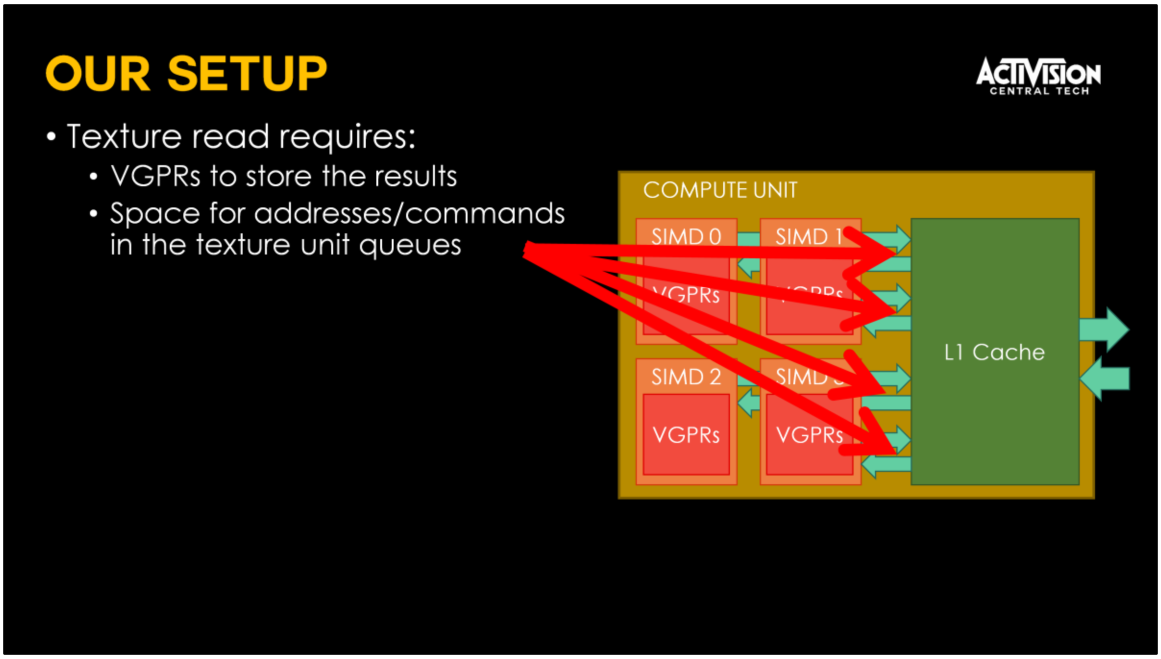
We're almost limited by the queues in the texture units, and while we have some slack in the available bandwidth, we really want to avoid adding too many memory access requests.

All these factors are in a delicate balance: you can use more registers, but then you limit the number of shader instances you can have in flight.

This, in turn, limits their ability to hide the memory lookup latency, etc.

16KB of L1 / ((avg) 4 wavefronts/SIMD * 4 SIMDs/CU * (avg) 4 lookups in flight * 64 bytes/cache line = 4096bytes) = 4, so there's room to hold 4 cache-lines per

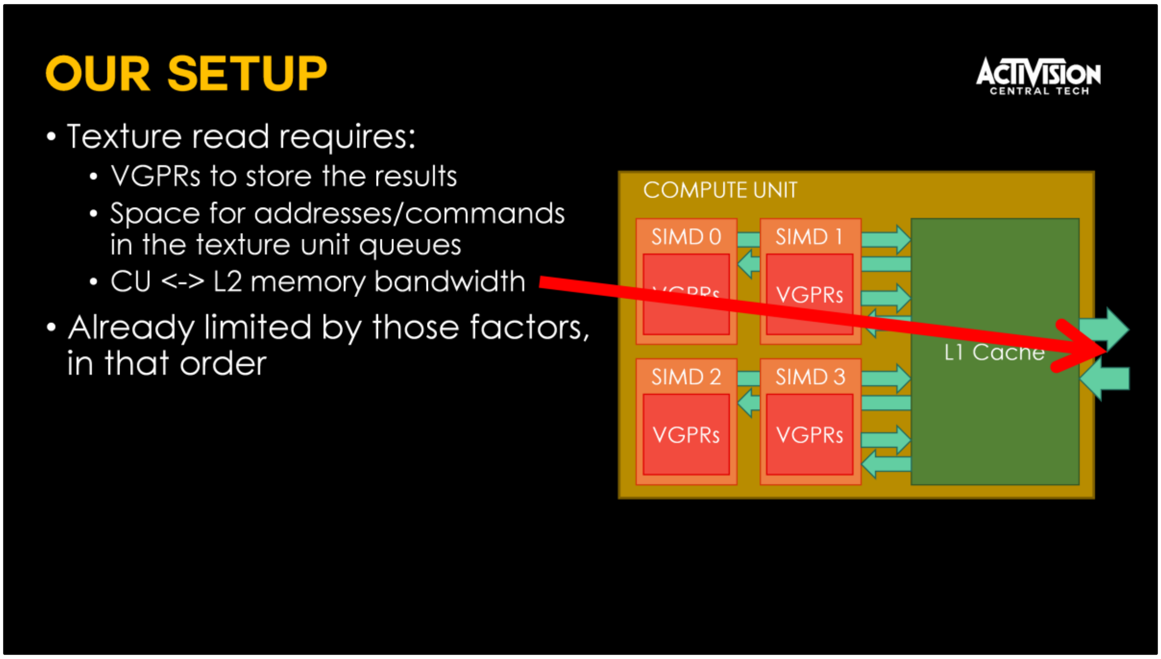
wavefront per lookup.
There are 64 threads in each wavefront, each thread performing lookups from independent positions in textures.
Lookup positions are usually close within a wavefront, so it might not require all four cache lines, but there’s also not really a lot of temporal reuse.
Mostly there is spatial reuse, with the L1<->L2 path constantly busy.



OUR SETUP

ACTIVISION CENTRAL TECH

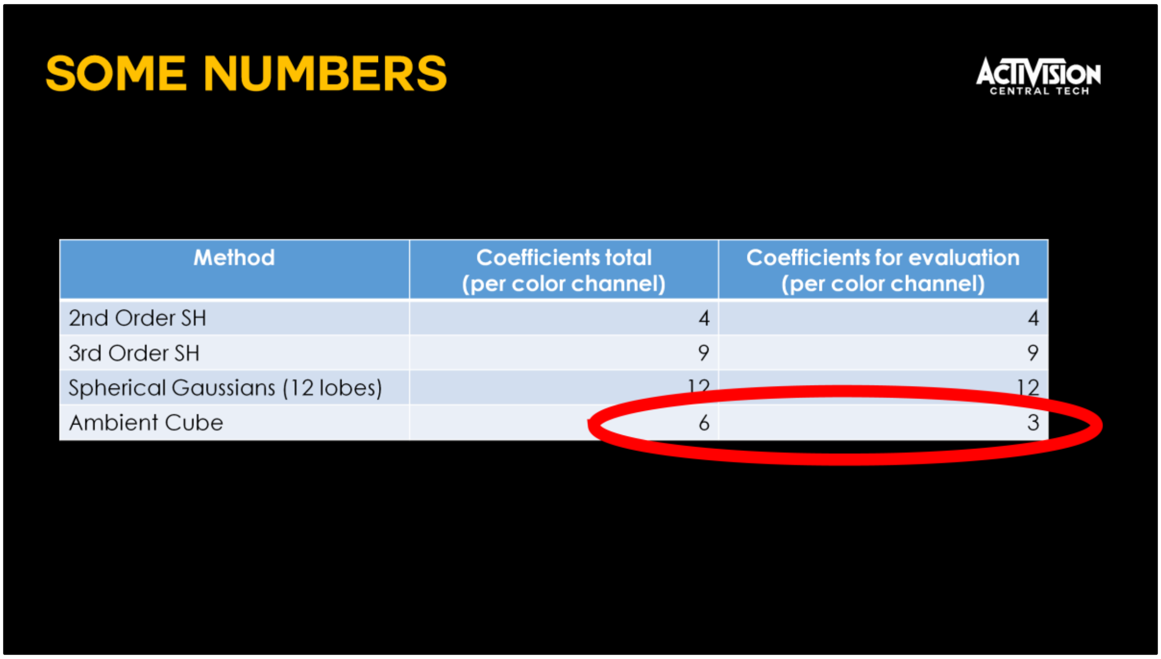
- Texture read requires:
  - VGPRs to store the results
  - Space for addresses/commands in the texture unit queues



OUR SETUP

ACTIVISION CENTRAL TECH

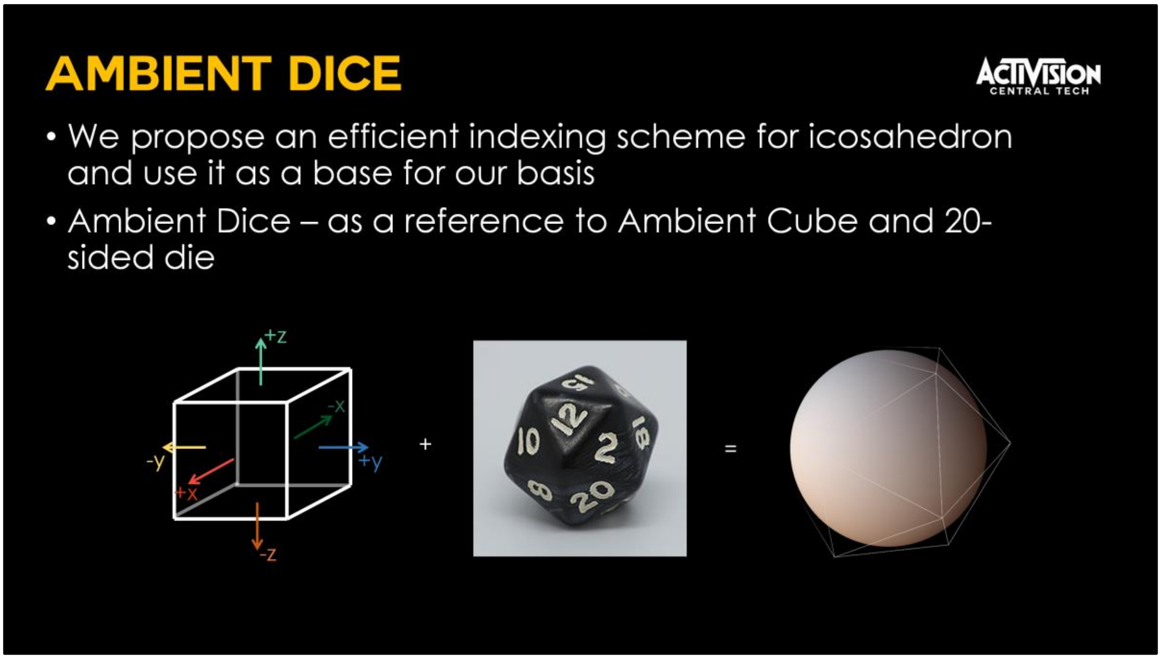
- Texture read requires:
  - VGPRs to store the results
  - Space for addresses/commands in the texture unit queues
  - CU <-> L2 memory bandwidth
- Already limited by those factors, in that order



SOME NUMBERS

Method	Coefficients total (per color channel)	Coefficients for evaluation (per color channel)
2nd Order SH	4	4
3rd Order SH	9	9
Spherical Gaussians (12 lobes)	12	12
Ambient Cube	6	3

The table here shows the number of coefficients that have to be fetched from the memory to compute the diffuse lighting in some of the cases discussed before.
The interesting thing here is the AC basis, which doesn’t need all the coefficients to perform the evaluation.
This is due to the fact that it uses the clamped cos^2 lobes, which have a local support.
This is a great property, and our main motivation was to find a modern alernative for that, something that would also be locally supported but provide better quality, comparable to at least 3rd-order SH (but at a lower memory access cost).
We were fine with using more memory total as those are things we can reason about and solve separately, as long as we wouldn’t need to access it all.

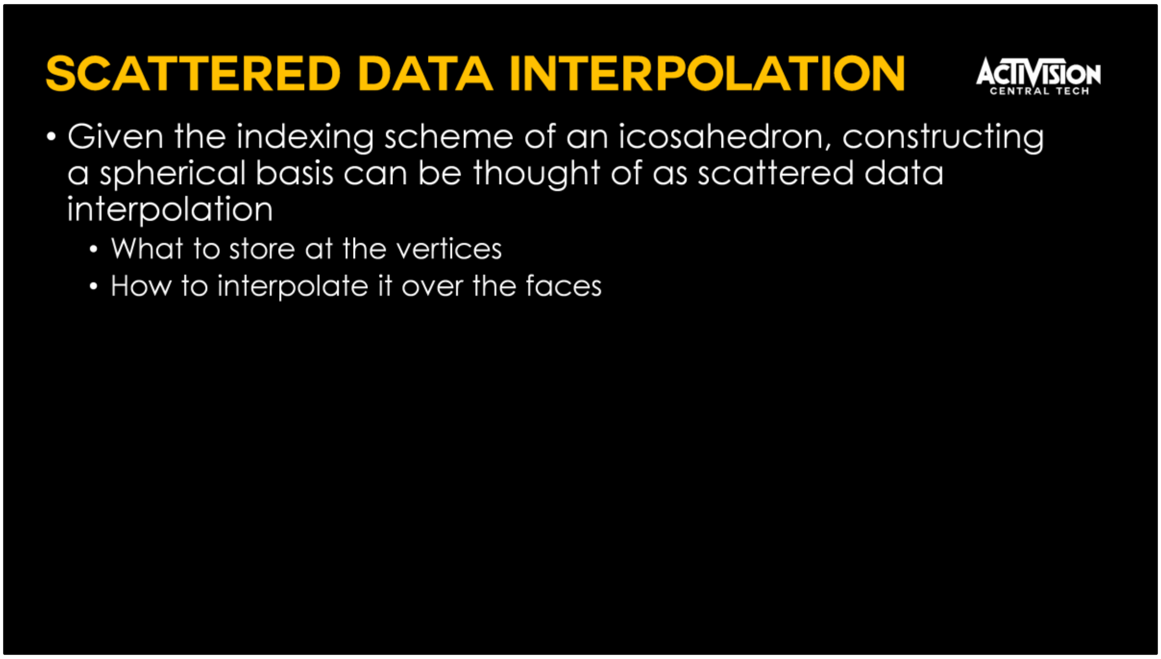


AMBIENT DICE

ACTIVISION CENTRAL TECH

- We propose an efficient indexing scheme for icosahedron and use it as a base for our basis
- Ambient Dice – as a reference to Ambient Cube and 20-sided die

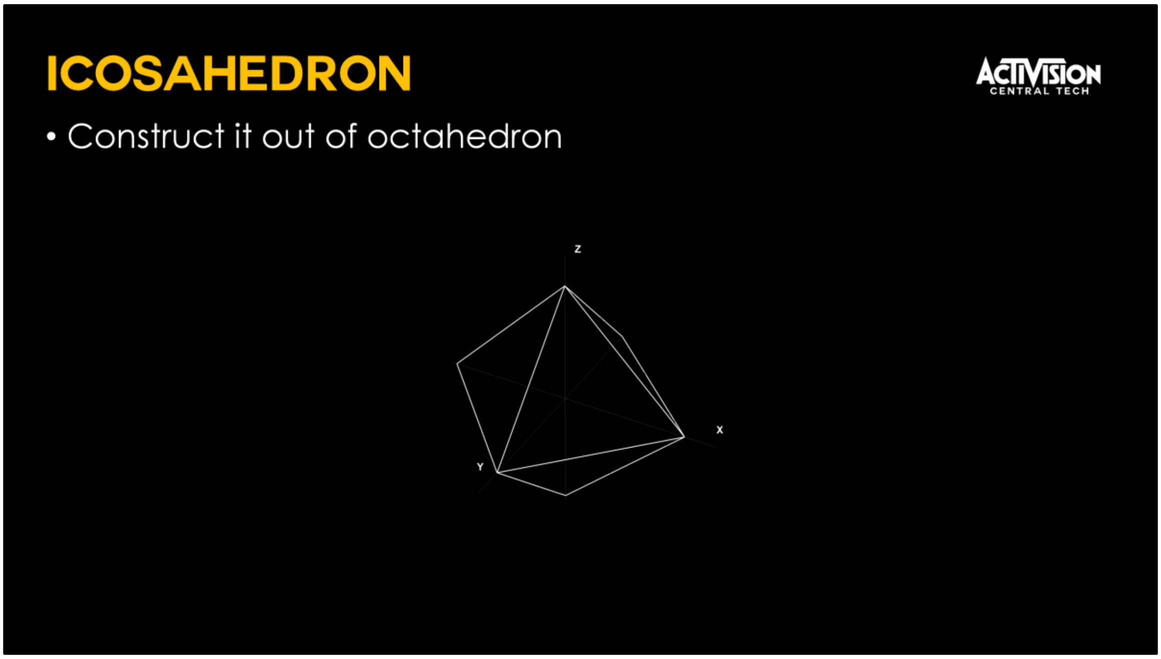
AC is based, unsurprisingly, on a cube: each lobe points to one of its faces.
A cube has the convenient property that it's really easy to figure out which 3 faces are aligned with a given direction, and thus which 3 lobes we are going to need.
We just take the signs of the direction vector coordinates.
Our idea was to find a similar indexing scheme for some other regular polyhedron.
After looking into it a bit, we found one for icosahedron, and we based our basis on that.
The basis comes in different variants, and we called the whole family Ambient Dice, as a reference to Ambient Cube and 20-sided die from pen and paper role-playing games.



SCATTERED DATA INTERPOLATION

- Given the indexing scheme of an icosahedron, constructing a spherical basis can be thought of as scattered data interpolation
  - What to store at the vertices
  - How to interpolate it over the faces

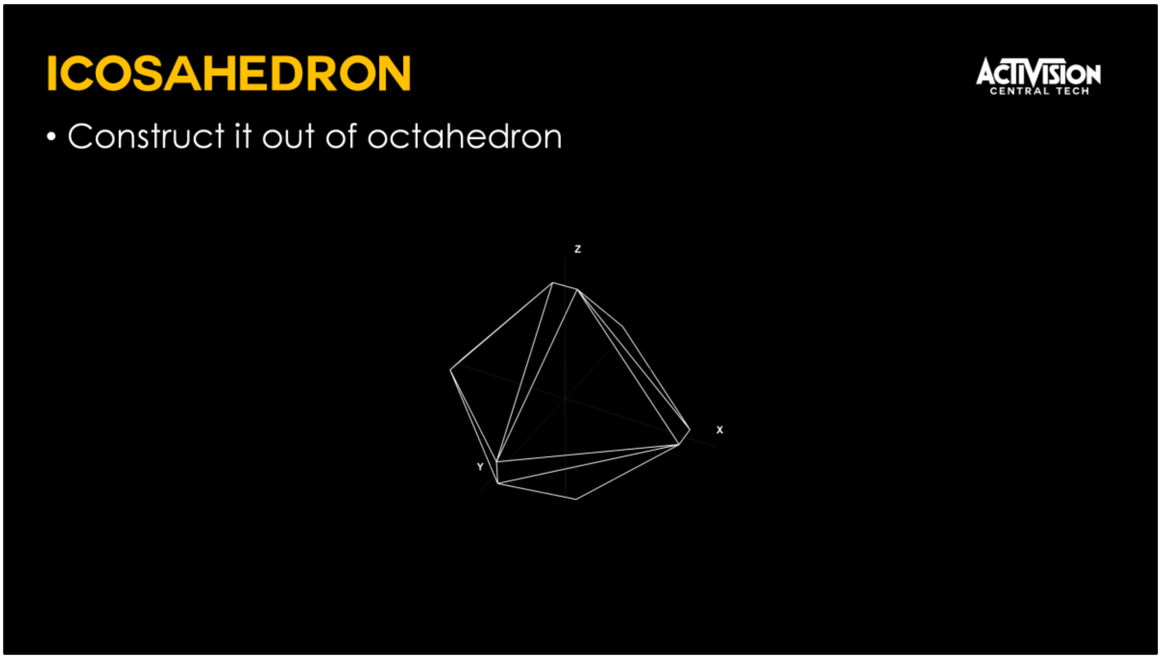
The general idea is to take the icosahedron, find an efficient indexing scheme and create a basis through scattered data interpolation.
We would store something at the vertices and/or faces of the polyhedron, and define some way of interpolating those values across the whole sphere, and this will give us the spherical basis.



ICOSAHEDRON

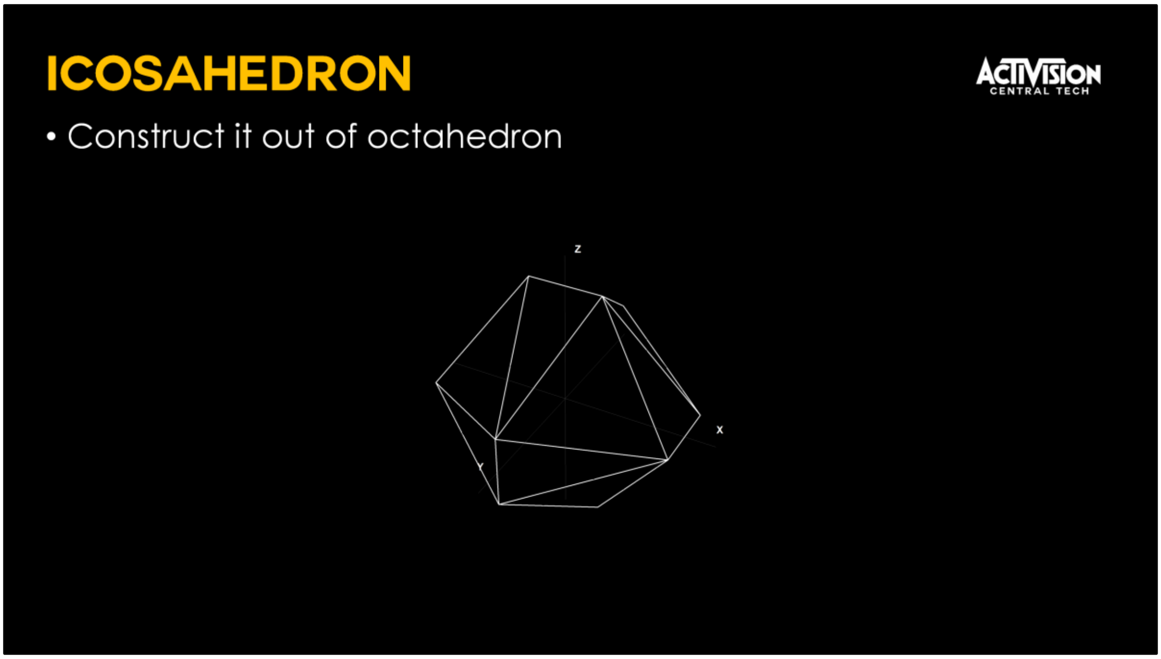
- Construct it out of octahedron

First, the indexing scheme.
If you look at an icosahedron it’s not really obvious how to figure out which triangle you’re pointing to.
We base the indexing scheme on the construction of the icosahrdon from an octahedron.
If we take the octahedron oriented along cardinal axes, and split its vertices into two, insert extra edges there, and expand them when all the edges are the same length, we get an icosahedron.



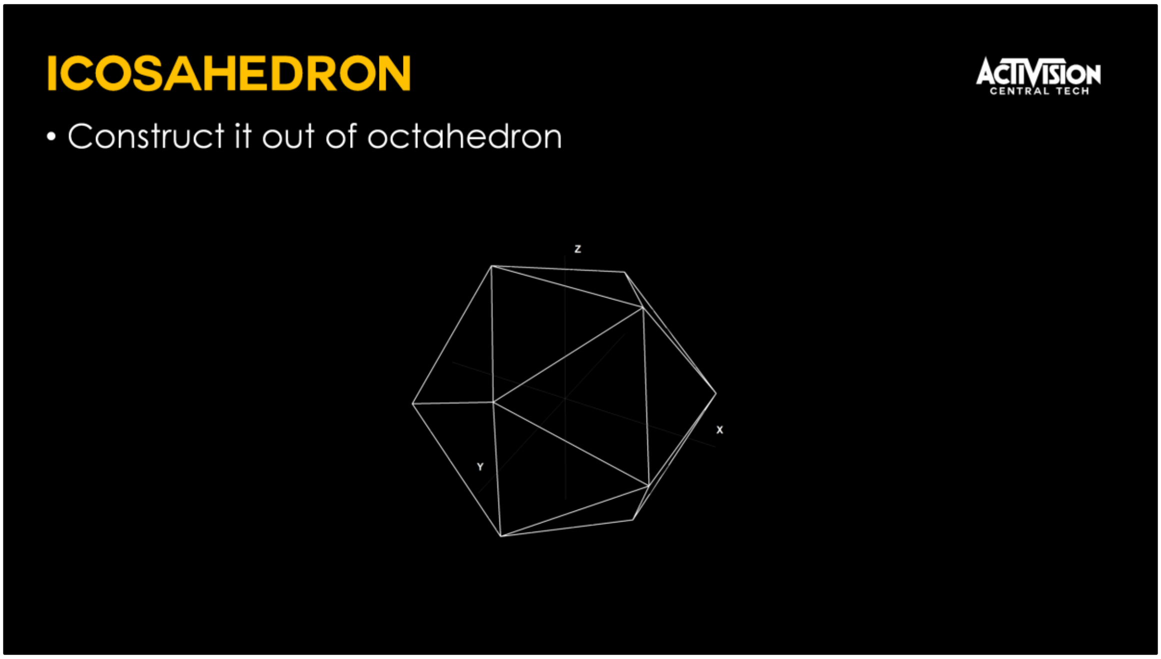
ICOSAHEDRON

- Construct it out of octahedron



ICOSAHEDRON

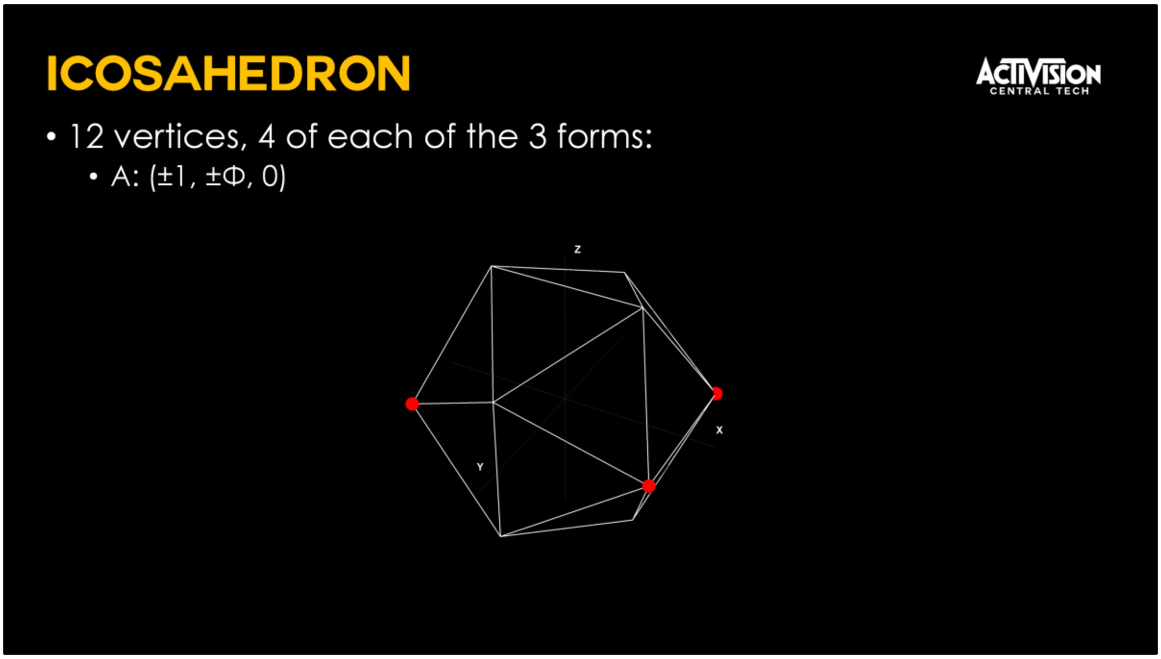
- Construct it out of octahedron



ICOSAHEDRON

ACTIVISION CENTRAL TECH

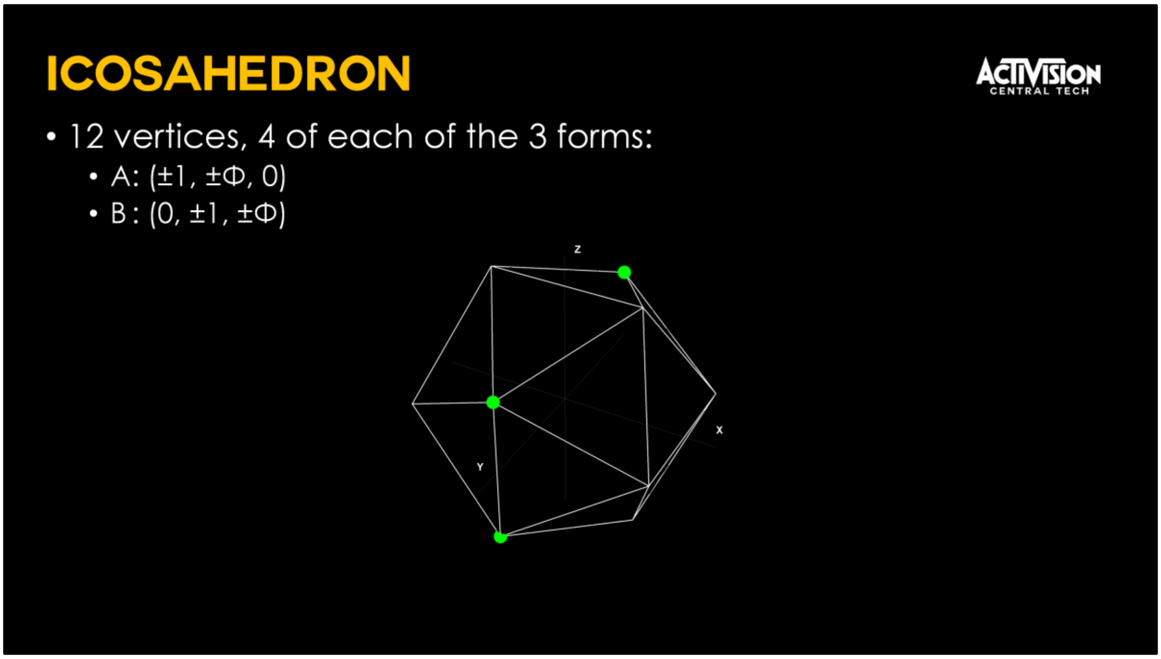
- Construct it out of octahedron



ICOSAHEDRON

- 12 vertices, 4 of each of the 3 forms:
  - A: (±1, ±Φ, 0)

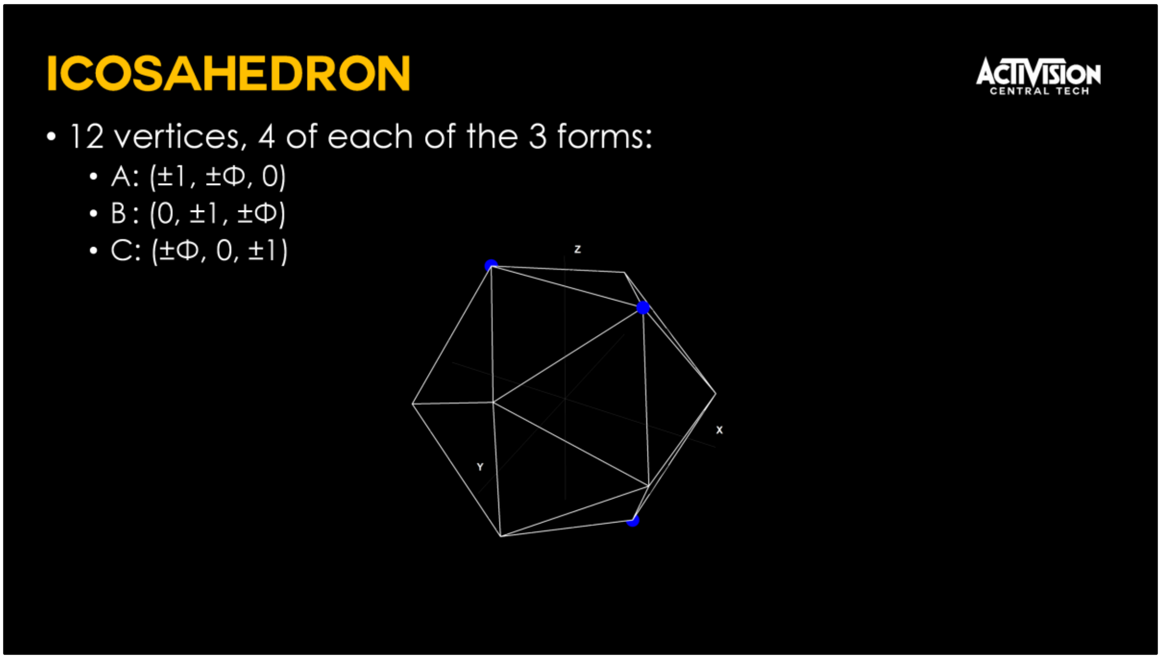
It has some convenient properties.
First, all its vertices are of one of three forms, where capital phi is the reciprocal of the golden ratio.
As you can see they are just cyclic permutations of each other, there are 4 vertices in each of such groups, two antipodal pairs, and we can assign an index to them by extraxting the sign bits: group A will have indices 0-3, group B will have 4-7, etc.



ICOSAHEDRON

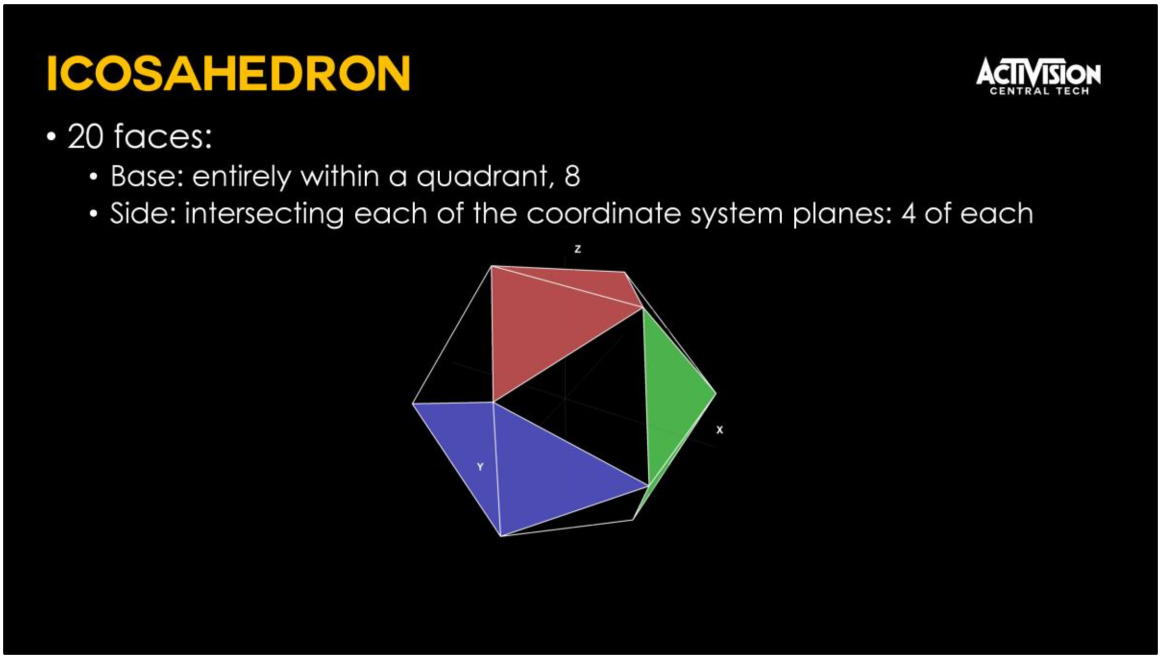
ACTIVISION CENTRAL TECH

- 12 vertices, 4 of each of the 3 forms:
  - A: (±1, ±Φ, 0)
  - B : (0, ±1, ±Φ)



ICOSAHEDRON

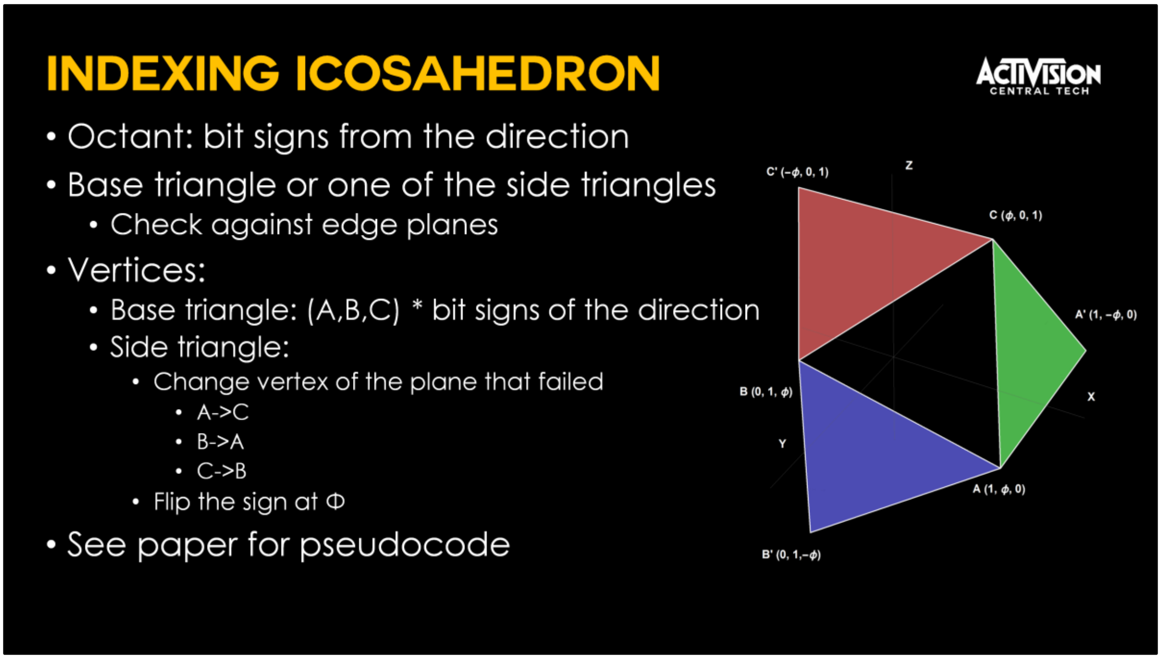
- 12 vertices, 4 of each of the 3 forms:
  - A: (±1, ±Φ, 0)
  - B : (0, ±1, ±Φ)
  - C: (±Φ, 0, ±1)



ICOSAHEDRON

- 20 faces:
  - Base: entirely within a quadrant, 8
  - Side: intersecting each of the coordinate system planes: 4 of each

An icosahedron has 20 faces, all triangles, and we divide them into “base” triangles, that are entirely contained within an octant of a coordinate system, and the 3 groups of “side” triangles, where triangles in each of group intersect one plane of the coordinate system.

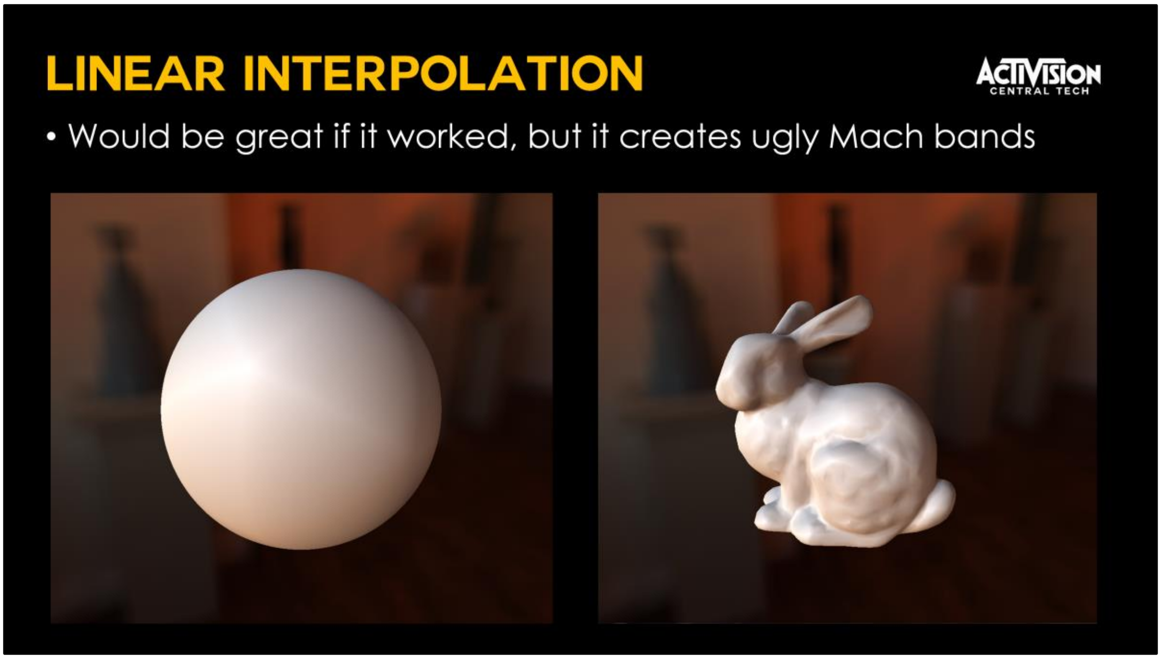


INDEXING ICOSAHEDRON

- Octant: bit signs from the direction
- Base triangle or one of the side triangles
  - Check against edge planes
- Vertices:
  - Base triangle: (A,B,C) * bit signs of the direction
  - Side triangle:
    - Change vertex of the plane that failed
      - A->C
      - B->A
      - C->B
    - Flip the sign at Φ
- See paper for pseudocode

When we want to figure out which triangle a given direction points to, we first figure out which octant it's pointing to, by simply extracting the sign bits from the direction vector.
Given the octant there are 4 triangles to choose from and you can make the choice with some simple math. In the end you have an index of a triangle, the coordinates of its three vertices, and their indices from 0 to 11.
The pseudo-code is in the paper, and it's actually super simple.
Given the octant we can either point to the base triangle or one of the side triangles, each belonging to a different group.
We can just check the direction against the planes passing through the edges of the base triangle.
Because of all the symetries, this can be done in the +++ octant, and the dot products needed for the tests are computed against vectors with one coordinate equal to 1, so they boil down to two multiply-adds.
Each vertex of the base triangle is from a different group, so if we point to the base triangle, we just use them, multiplied by the bit signs from the direction.
If we use any of the side triangles, one of the vertices gets replaced with a vertex from another group with one coordinate flipped.
And however complicated this sounds, the code for that is actually really simple, and

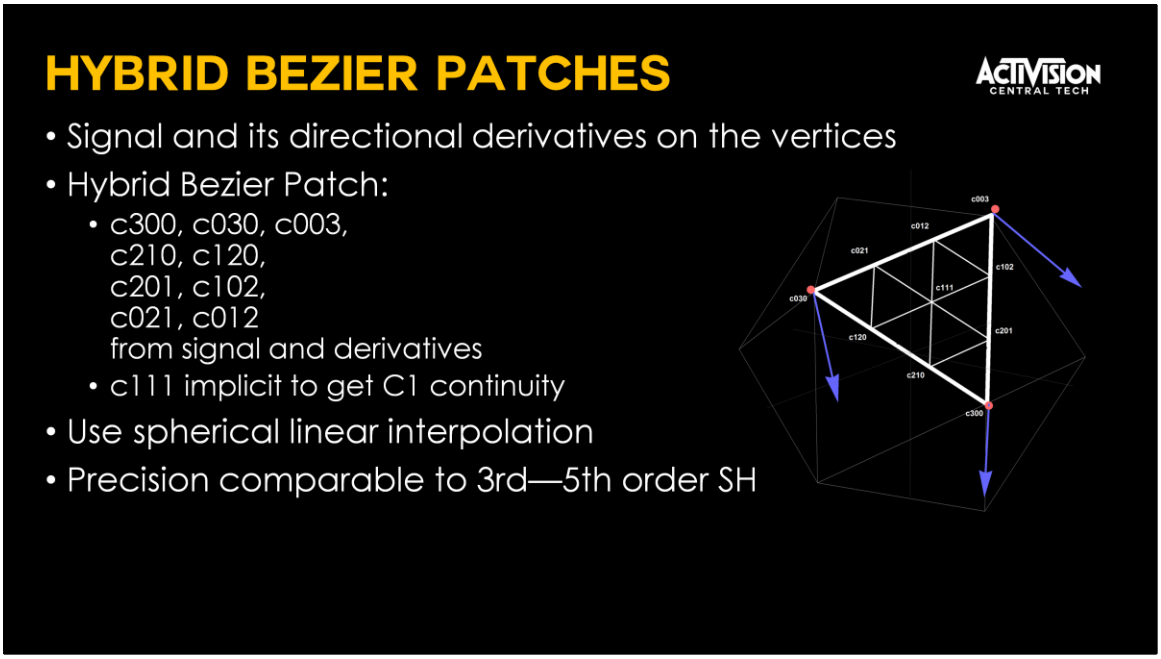
it’s in the paper.



LINEAR INTERPOLATION

- Would be great if it worked, but it creates ugly Mach bands

Now we need to figure out how to interpolate the data across the icosahedron.
Of course the most obvious solution would be to just use linear interpolation: that would only require storing a single RGB value for every vertex, and evaluation would only require fetching the 3 values of the vertices of the triangle we’re pointing to.
Unfortunately, this looks pretty bad.
The mach bands are really apparent, and even using more fancy spherical linear interpolation doesnt really help either.



HYBRID BEZIER PATCHES

- Signal and its directional derivatives on the vertices
- Hybrid Bezier Patch:
  - c300, c030, c003,
    c210, c120,
    c201, c102,
    c021, c012
    from signal and derivatives
  - c111 implicit to get C1 continuity
- Use spherical linear interpolation
- Precision comparable to 3rd—5th order SH

We tried a lot of different options, and the thing that worked really well were hybrid Bezier patches.

Hybrid Bezier Patch is just a convex combination of three quadratic bezier patches that have a different center control point.

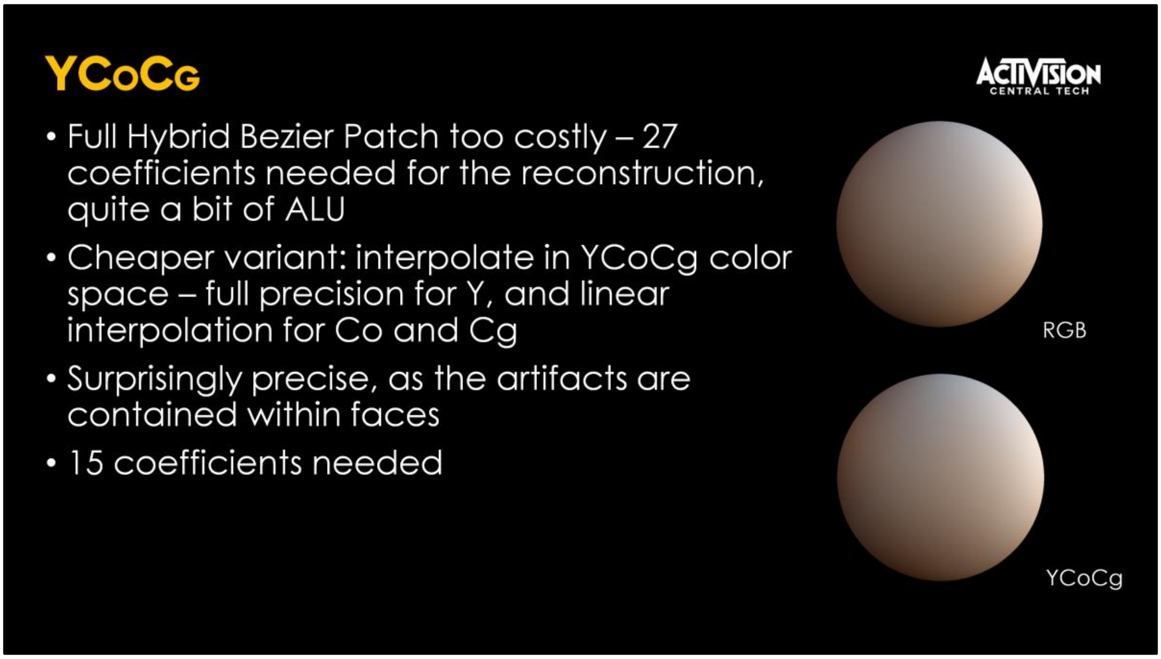
To construct the patch we need to store signal and its directional derivatives on the vertices of the icosahedron.

This uniquely identifies the 9 control points of the patch, and then the remaining one is chosen to create C1 continuity across neighboring patches.

This can be don’t without any extra data if we make the signal change linearly instead of quadratically along the shared edge.

The reconstruction consists of finding the triangle, grabbing the vertex data, computing the control points and then evaluating the hybrid Bezier patch.

And this looks really great, the quality of the reconstruction of the diffuse irradiance signal is comparable to often even 5th order SH.



YCoCg

- Full Hybrid Bezier Patch too costly – 27 coefficients needed for the reconstruction, quite a bit of ALU
- Cheaper variant: interpolate in YCoCg color space – full precision for Y, and linear interpolation for Co and Cg
- Surprisingly precise, as the artifacts are contained within faces
- 15 coefficients needed

RGB

YCoCg

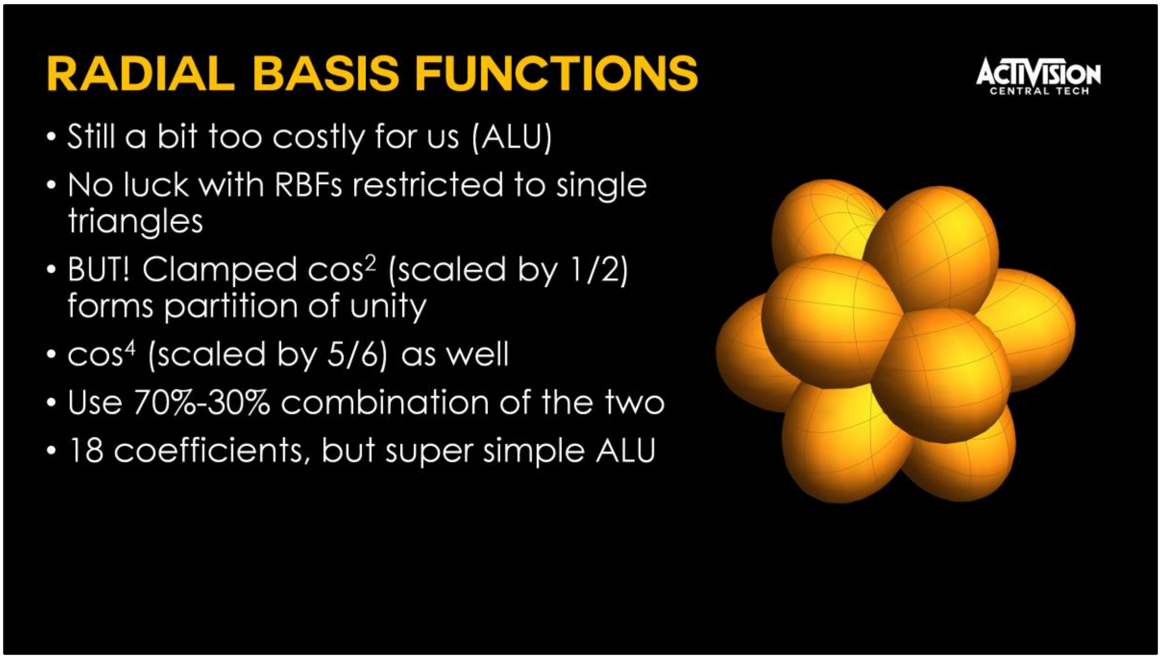
However, it was a bit too costly for our use-case, because it requires a fair share of ALU and we need 27 coefficients for the reconstruction.

That’s the same as for 3rd order SH, and even though the quality is higher, we would happily trade some of it for cheaper reconstruction.

Our first solution for that we was to switch to YCoCg color space and use the higher-order interpolation only for the luminosity component, as this is what our eyes are most sensitive to, and use simpler, linear interpolation for the chromaticity componnts.

And while it does generate some minor discolorations, they are very localized, contained within single faces of the icosahedron. In general, there’s almost no quality loss.

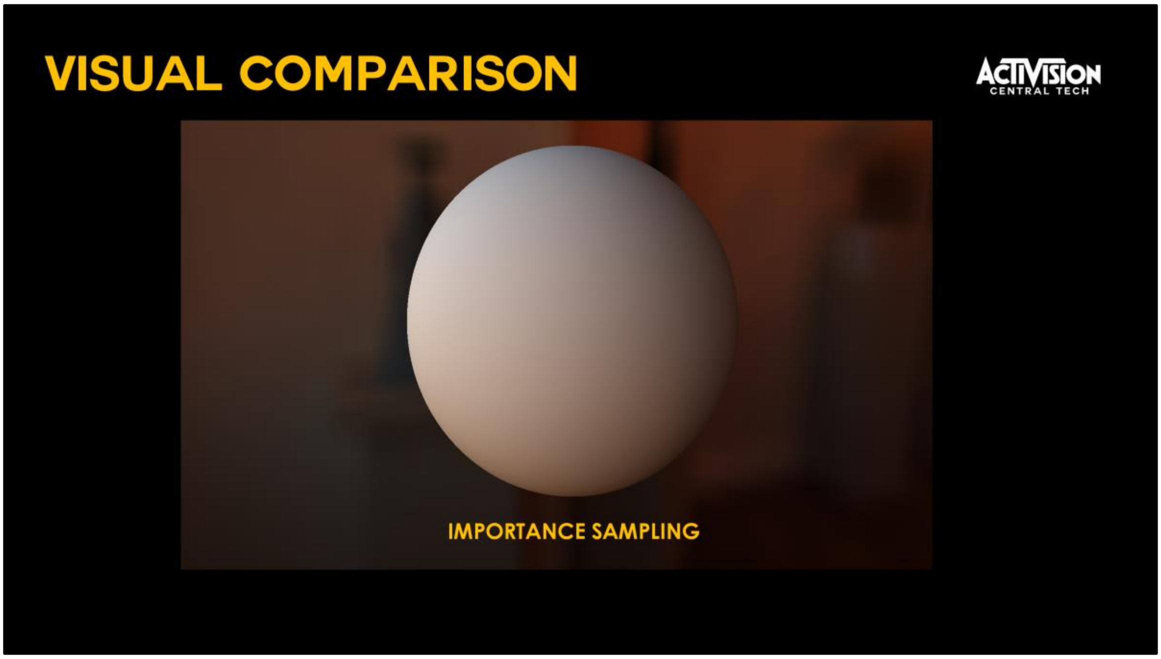
And it brings the cost down quite a bit – because we now need way less data for the reconstruction: only 15 coefficients.



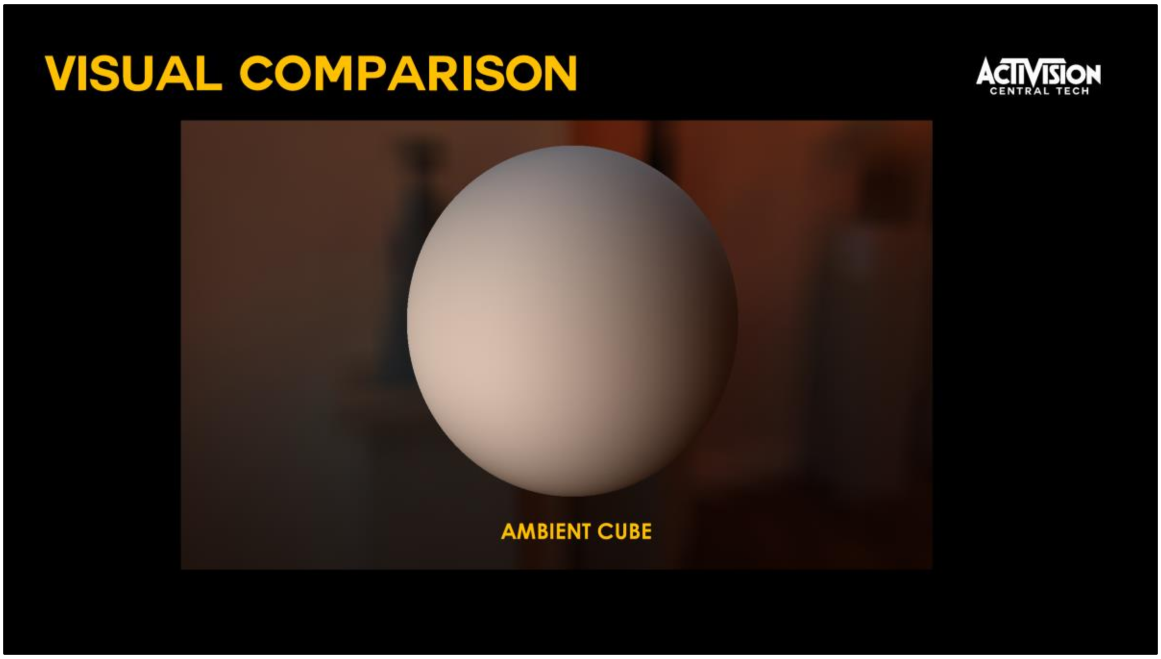
RADIAL BASIS FUNCTIONS

- Still a bit too costly for us (ALU)
- No luck with RBFs restricted to single triangles
- BUT! Clamped $\cos^2$ (scaled by 1/2) forms partition of unity
- $\cos^4$ (scaled by 5/6) as well
- Use 70%-30% combination of the two
- 18 coefficients, but super simple ALU

However, we still weren’t entirely happy, it was still a bit too costy in term of ALU.
While investigating different radial basis functions (RBF) we noticed that clamped cos^2 forms partition of unity over a sphere, and so does cos^4.
We tested different combinations of the two and settled on 70%-30% mix of those two lobes.
For evaluation we need 6 lobes, intersecting the hemisphere pointed to be the evaluation direction, so 18 coefficients for reconstruction.
In return, we get a much simpler reconstruction logic.



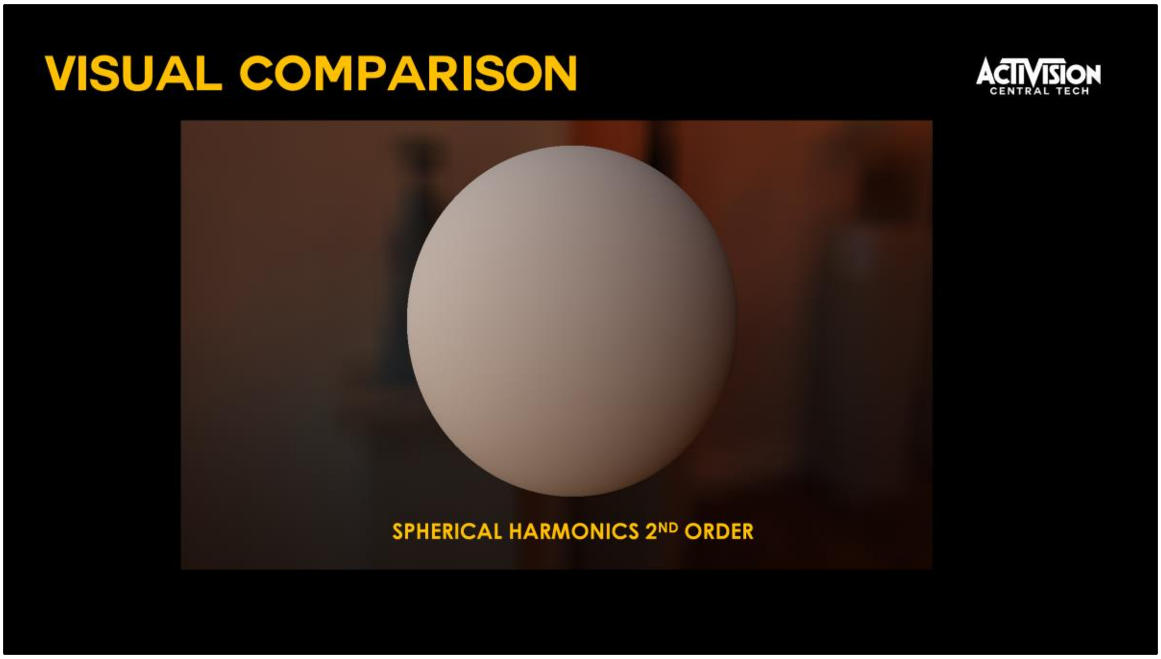
VISUAL COMPARISON



VISUAL COMPARISON

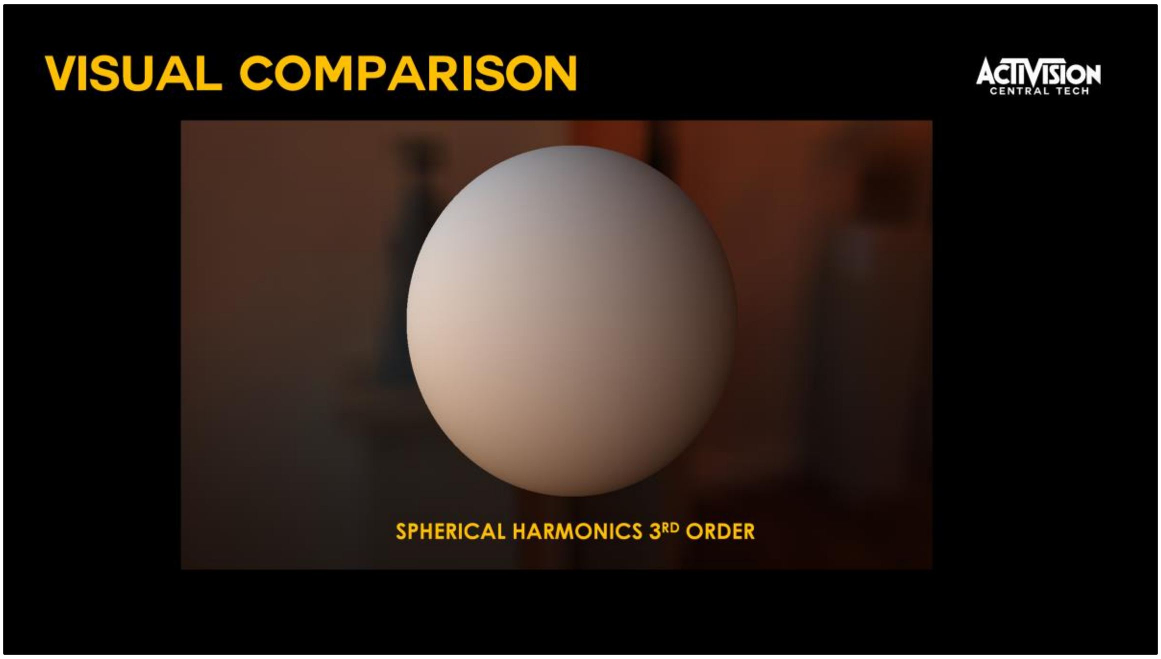
ACTIVISION CENTRAL TECH

AMBIENT CUBE



VISUAL COMPARISON

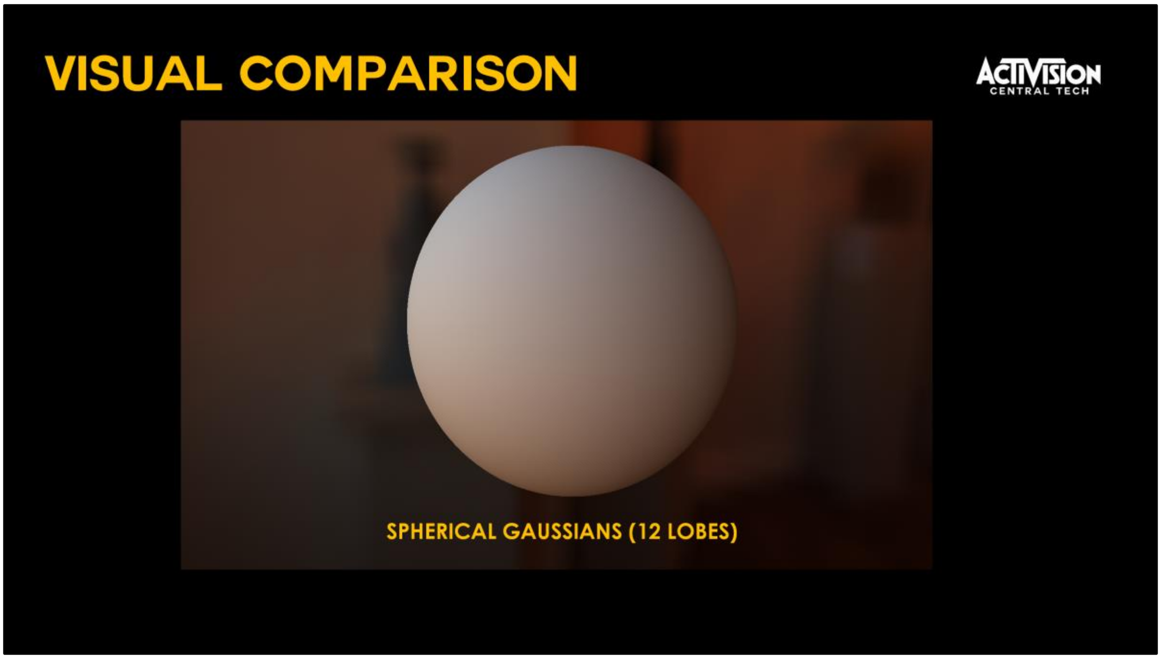
SPHERICAL HARMONICS 2^{ND} ORDER



VISUAL COMPARISON

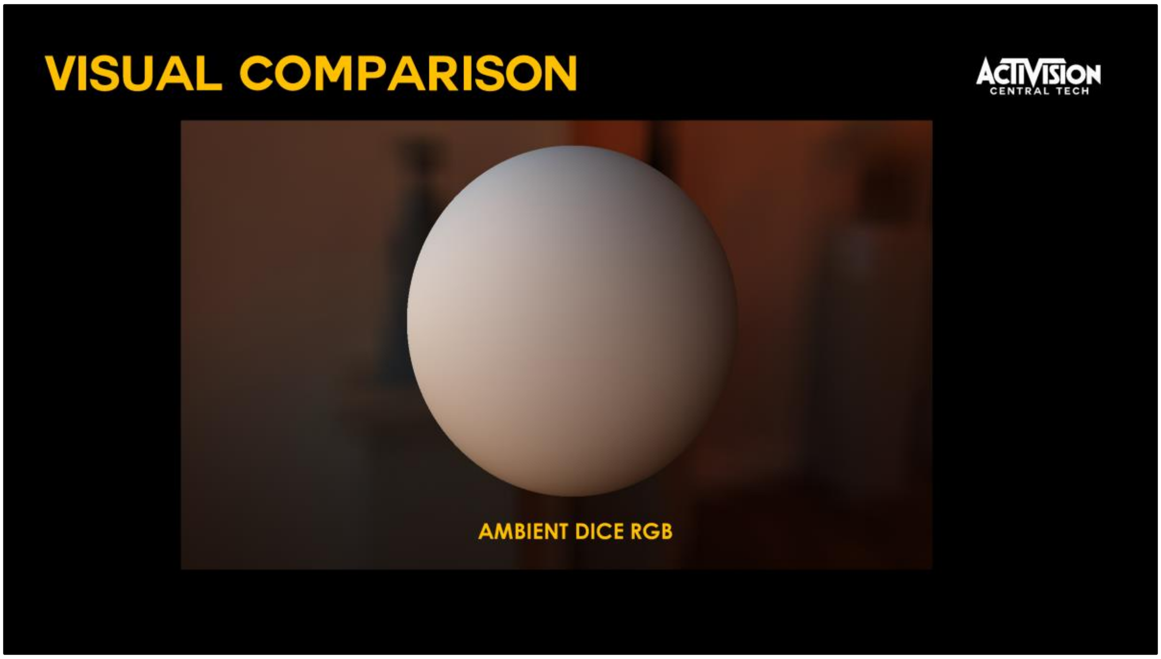
ACTIVISION CENTRAL TECH

SPHERICAL HARMONICS 3RD ORDER



VISUAL COMPARISON

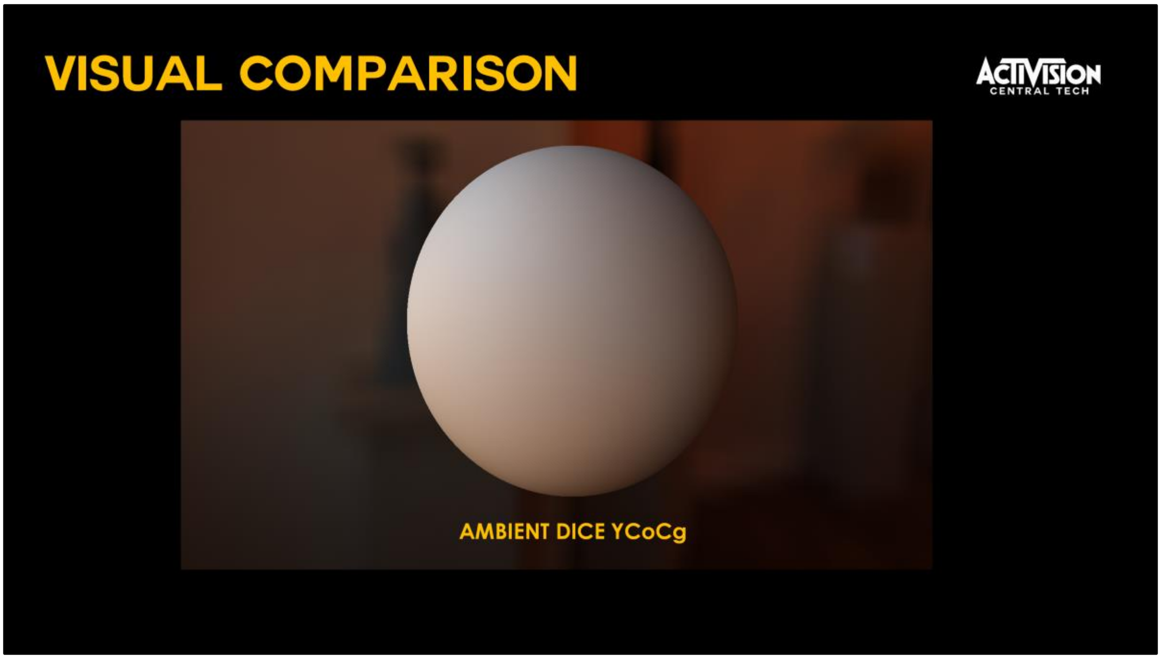
SPHERICAL GAUSSIANS (12 LOBES)



VISUAL COMPARISON

ACTIVISION CENTRAL TECH

AMBIENT DICE RGB

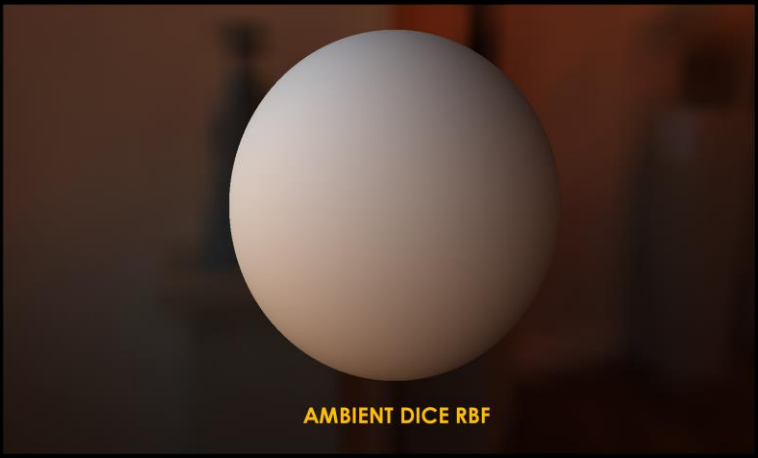


VISUAL COMPARISON

ACTIVISION CENTRAL TECH

AMBIENT DICE YCoCg

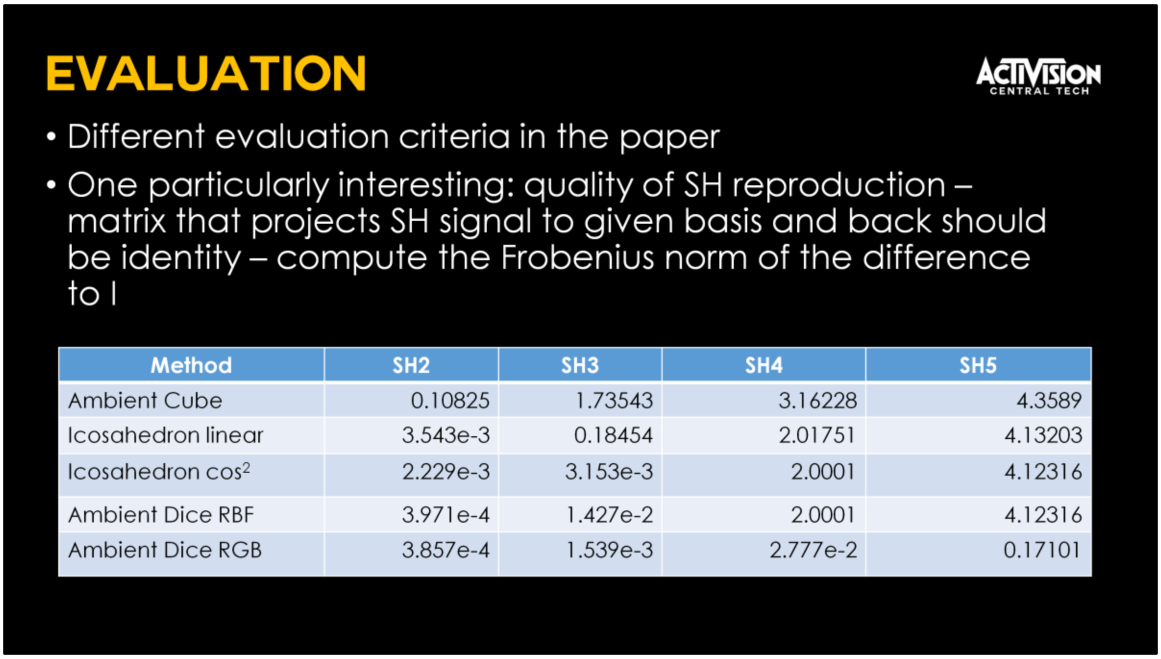
VISUAL COMPARISON



AMBIENT DICE RBF

VISUAL COMPARISON

IMPORTANCE SAMPLING



EVALUATION

- Different evaluation criteria in the paper
- One particularly interesting: quality of SH reproduction – matrix that projects SH signal to given basis and back should be identity – compute the Frobenius norm of the difference to I

Method	SH2	SH3	SH4	SH5
Ambient Cube	0.10825	1.73543	3.16228	4.3589
Icosahedron linear	3.543e-3	0.18454	2.01751	4.13203
Icosahedron cos^2	2.229e-3	3.153e-3	2.0001	4.12316
Ambient Dice RBF	3.971e-4	1.427e-2	2.0001	4.12316
Ambient Dice RGB	3.857e-4	1.539e-3	2.777e-2	0.17101

The paper covers some interesting details on the how to perform projection of the singals onto the basis, including cross-projection from different bases, but I wanted to mention here us one of the metrics that we use for the evaluation of the quality of the reconstruction.

We compute how well the given bases represents spherical harmonics of a given order.

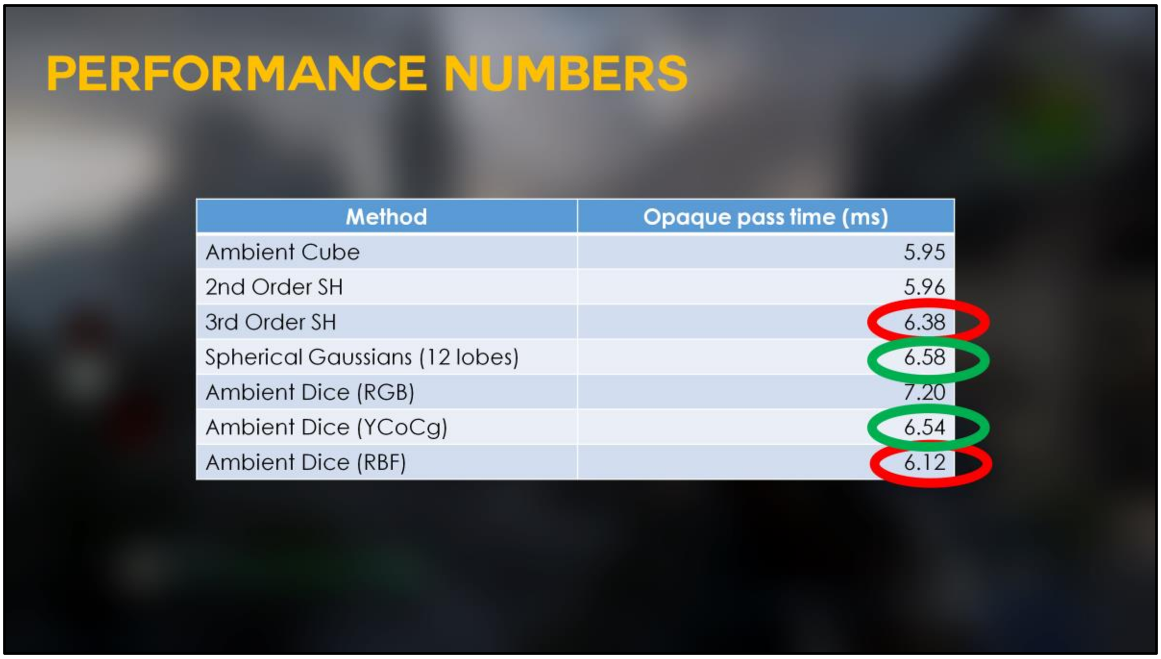
We can construct the matrix that takes the SH signal, projects it onto a given bases and then back to SH.

In an ideal situation this would be an identity matrix, but it generally isn’t, and we can compute the Frobenius norm of the difference.

This gives us a single number describing how well the bases represents a given order of SH.

AD in the Bezier patch versions works really well up to 4th order SH, and reasonably at 5th order.

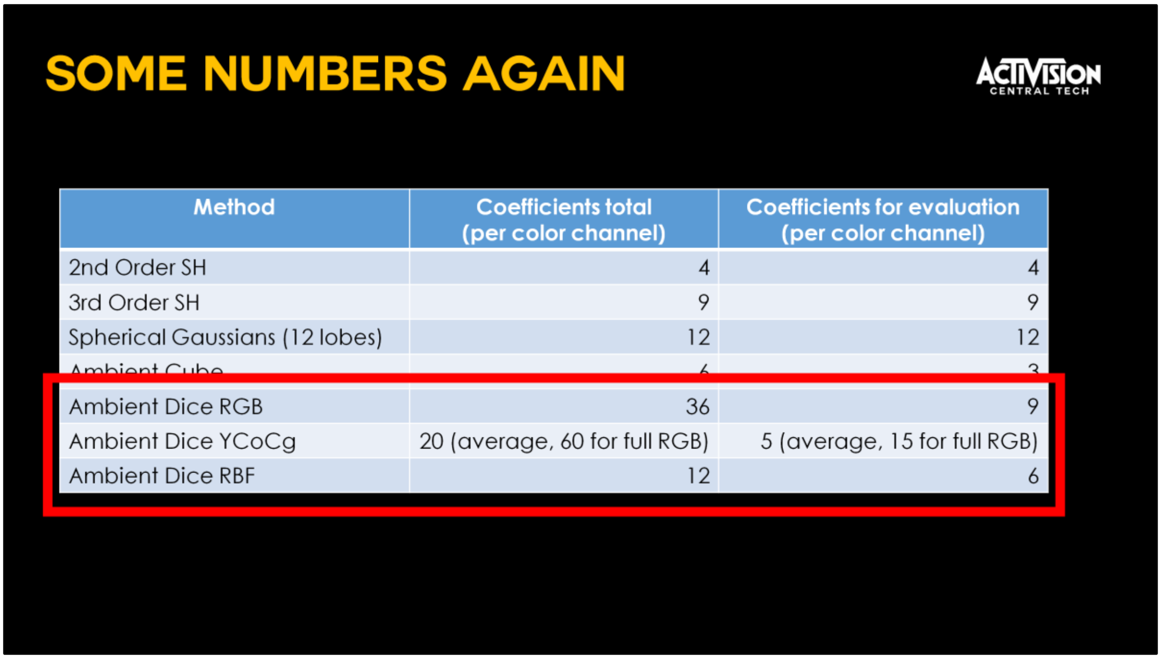
AD in the RBF setting works well up to 3rd order SH.



PERFORMANCE NUMBERS

Method	Opaque pass time (ms)
Ambient Cube	5.95
2nd Order SH	5.96
3rd Order SH	6.38
Spherical Gaussians (12 lobes)	6.58
Ambient Dice (RGB)	7.20
Ambient Dice (YCoCg)	6.54
Ambient Dice (RBF)	6.12

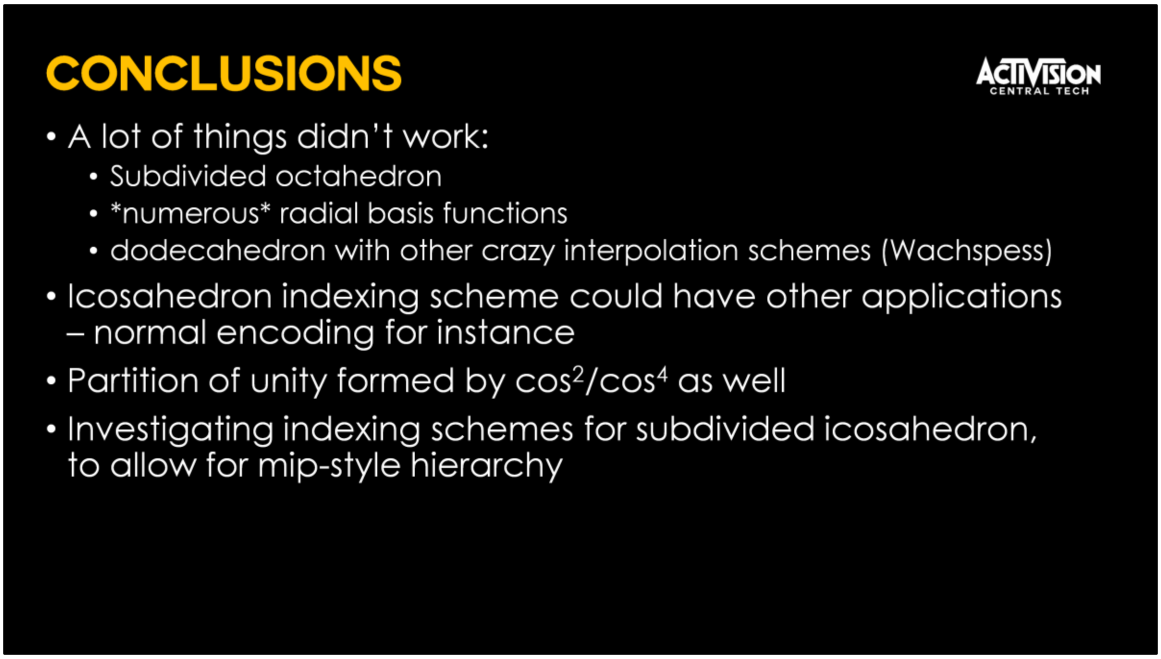
We also have data on the performance of the methods implemented in the Call Of Duty engine.
The timings are just for the rendering of the opaque pass, and we use two methods there for the indirect lighting – larger structural meshes use lightmaps and everything else uses the local, per-object volumes that store the indirect lighting in various representations.
I think the important bits from that table are the rows marked in red - SH3 and AD with RBF – the Ambient Dice is around 0.2 ms faster, placing AD between 2nd and 3rd order SH in terms of cost, while getting the quality of 3rd order SH.
The other bit is the YCoCg version.
That’s more expensive due to the more complicated ALU, but comparable to using 12 spherical Gaussian lobes.
The RBG version is of course very expensive both because of multiple texture fetches, and complicated ALU.



SOME NUMBERS AGAIN

Method	Coefficients total (per color channel)	Coefficients for evaluation (per color channel)
2nd Order SH	4	4
3rd Order SH	9	9
Spherical Gaussians (12 lobes)	12	12
Ambient Cube	6	3
Ambient Dice RGB	36	9
Ambient Dice YCoCg	20 (average, 60 for full RGB)	5 (average, 15 for full RGB)
Ambient Dice RBF	12	6

And here is the earlier table with the new rows.
While the total number of coefficients is higher with AD, they are well behaved coefficients, just colors and their derivatives, so it's easy to reason about them in terms of compression.
Especially the RBF variants, which need just the colors, and can be efficiently compressed with modern block compression.

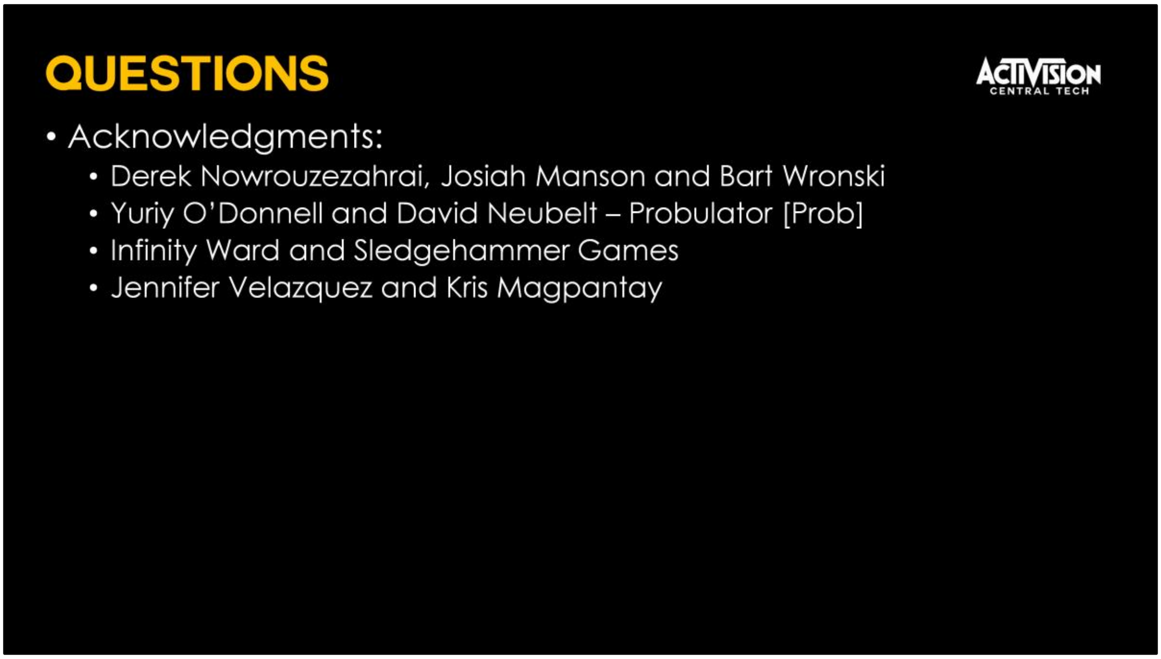


CONCLUSIONS

ACTIVISION CENTRAL TECH

- A lot of things didn't work:
  - Subdivided octahedron
  - *numerous* radial basis functions
  - dodecahedron with other crazy interpolation schemes (Wachspess)
- Icosahedron indexing scheme could have other applications – normal encoding for instance
- Partition of unity formed by $\cos^2/\cos^4$ as well
- Investigating indexing schemes for subdivided icosahedron, to allow for mip-style hierarchy

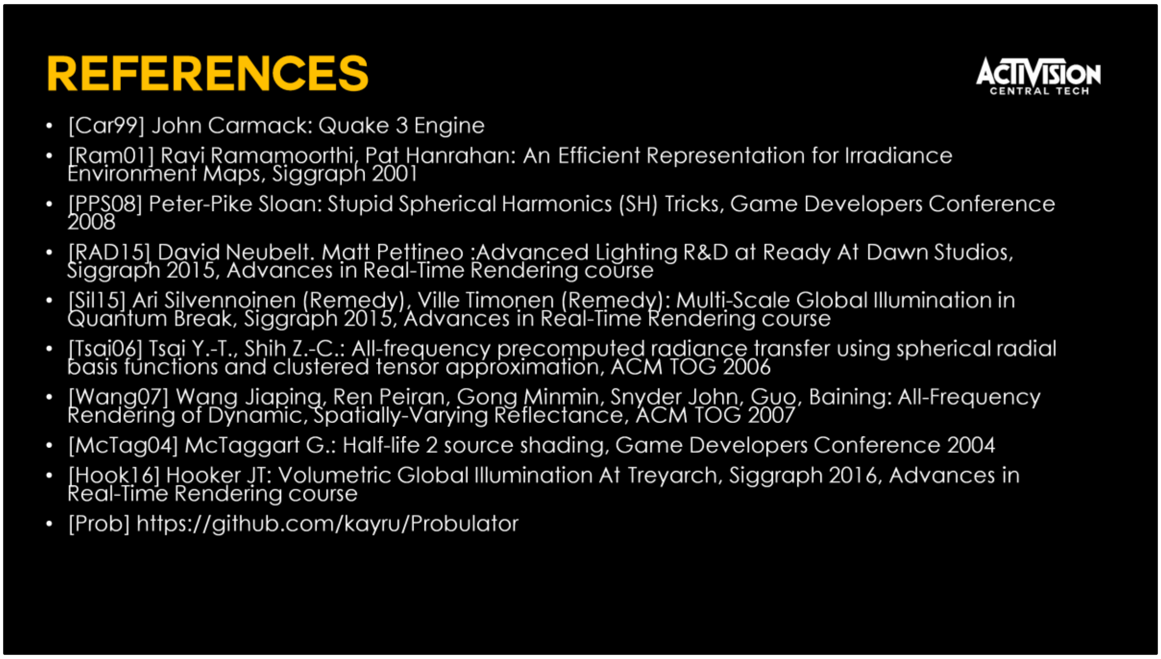
Subdivided octahedron, addressing is simple, see octahedral env mapping/normal encoding.
However it needs to duplicate certain values.
Any logic to remove the duplication blows up the ALU cost, way above the cost of icosahedron addressing.
Encoding normal vectors works, although the decoding cost is twice the cost of the octahedron normal encoding.
The resulting quality of the reconstruction is higher, but the extra quality might not be required by many applications.



QUESTIONS

ACTIVISION CENTRAL TECH

- Acknowledgments:
  - Derek Nowrouzezahrai, Josiah Manson and Bart Wronski
  - Yuriy O'Donnell and David Neubelt – Probulator [Prob]
  - Infinity Ward and Sledgehammer Games
  - Jennifer Velazquez and Kris Magpantay



REFERENCES

ACTIVISION CENTRAL TECH

- [Car99] John Carmack: Quake 3 Engine
- [Ram01] Ravi Ramamoorthi, Pat Hanrahan: An Efficient Representation for Irradiance Environment Maps, Siggraph 2001
- [PPS08] Peter-Pike Sloan: Stupid Spherical Harmonics (SH) Tricks, Game Developers Conference 2008
- [RAD15] David Neubelt. Matt Pettineo :Advanced Lighting R&D at Ready At Dawn Studios, Siggraph 2015, Advances in Real-Time Rendering course
- [Sil15] Ari Silvennoinen (Remedy), Ville Timonen (Remedy): Multi-Scale Global Illumination in Quantum Break, Siggraph 2015, Advances in Real-Time Rendering course
- [Tsai06] Tsai Y.-T., Shih Z.-C.: All-frequency precomputed radiance transfer using spherical radial basis functions and clustered tensor approximation, ACM TOG 2006
- [Wang07] Wang Jiaping, Ren Peiran, Gong Minmin, Snyder John, Guo, Baining: All-Frequency Rendering of Dynamic, Spatially-Varying Reflectance, ACM TOG 2007
- [McTag04] McTaggart G.: Half-life 2 source shading, Game Developers Conference 2004
- [Hook16] Hooker JT: Volumetric Global Illumination At Treyarch, Siggraph 2016, Advances in Real-Time Rendering course
- [Prob] https://github.com/kayru/Probulator