UberBake light baking system

•••

Trifonov Andrey

What is UberBake?

UberBake is a global illumination system developed by Activision, which supports limited lighting changes in response to certain player interactions.

Goal:

Dynamically update the precomputed lighting at run-time with minimal performance and memory overhead.

The main idea

We can choose a limited subset of user interactions that affect lighting and receive many of the benefits of a fully dynamic global illumination solution.

This made it possible to:

1) efficiently pre-compute lighting changes associated with each interaction

2) implement a run-time system that, on average, is no slower than fully static implementation and uses a minimal amount of extra memory, that scales linearly with the number of allowed interactions

Existing and Alternative Solutions

- ➢ Real-time Light Transport (dynamic lighting and geometry)
  - Interpolation artifacts
  - Requires dedicated ray tracing hardware
- ➢ Precomputed Light Transport
  - Incompatible with arbitrary geometry changes
- ➢ Precomputed Lighting
  - Do not allow any dynamic lighting changes

Representing lighting

- Which lighting do we store?
- How do we store lighting values?

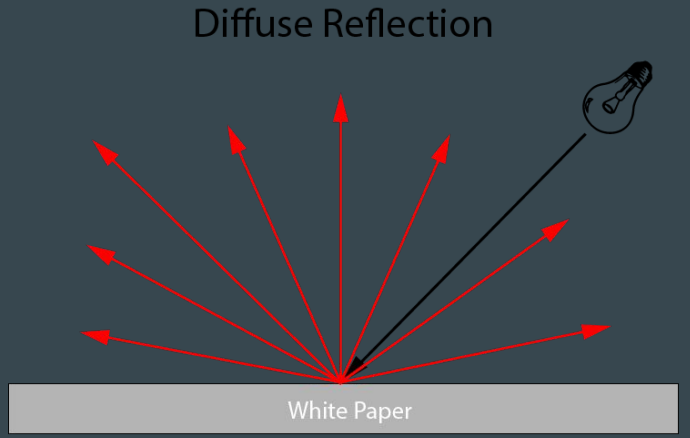
Which lighting do we store?

Heckbert's path notation:

L - light sources 

D - diffuse reflections 

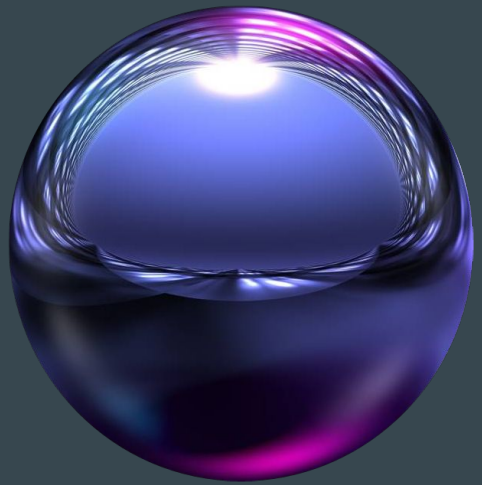
R - receivers 



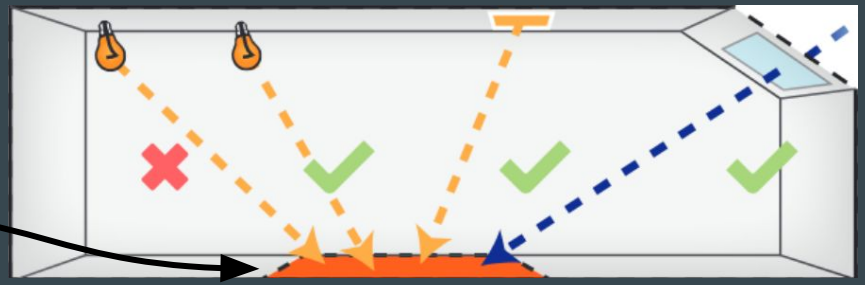
Only precomputing diffuse indirect lighting (LD+R)

Why don’t we include specular or direct lighting?

Including specular or direct lighting would require storing data at a much higher resolution to allow satisfactory reconstruction quality.



We store direct lighting for ‘expensive’ light sources though



Primary light	Run-time (per pixel)
Static light	Baked (in areas with many lights)
Emissives	Baked (again, too expensive for real-time)
Sky light	Baked (prohibitively expensive)

How do we store baked light?

Multiple storage formats:

- Directional lighting encoding
- Lightmaps
- Local Light Grids (LLGs)
- The global light grid (GLG)

Lighting is consistent despite the storage format

- Local Light Grids (LLGs)
- Lightmaps
- Global Light Grid for dynamic objects

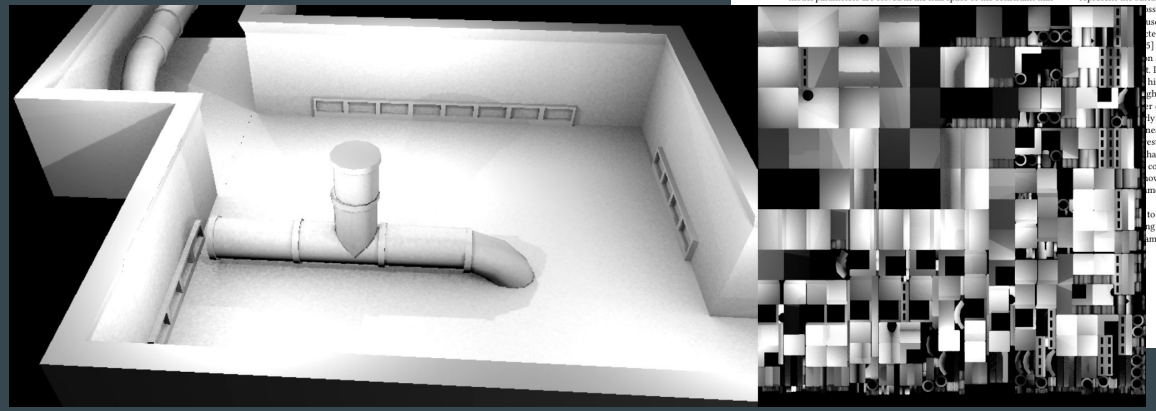
rendering pipeline is unaware of the difference

Lightmaps

Used for wall segments, or entire buildings

Resolution in scaled for each surface relatively

Ambient Highlight Direction (AHD) for encoding



Directional Lightmap Encoding Insights

Peter-Pike Sloan
Activision Publishing
ppsloan@activision.com

Ari Silvennoinen
Activision Publishing
Ari.Silvennoinen@
activision.com

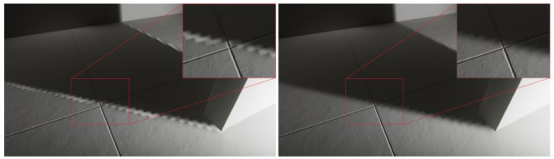


Figure 1: Left is AHD interpolation, right is new encoding. There is a static light near the ground shadowed by a pillar, and a light bouncing off the back wall that has higher albedo. Interpolating AHD, you can see artifacts in the transition from lit to shadowed. Using the new encoding, the lighting interpolates properly. The images are looking at lighting only, no albedo. ©Activision Publishing, Inc.

ABSTRACT

Lightmaps that respond to normal mapping are commonly used in video games. This short paper describes a novel parameterization of a standard lightmap encoding, Ambient Highlight Direction (AHD) — a model for directional irradiance consisting of ambient and directional light — that eliminates common interpolation artifacts. It also describes a technique for fitting the AHD model to lighting represented as spherical harmonics, where the unknown model parameters are solved in the null space of the constraint that

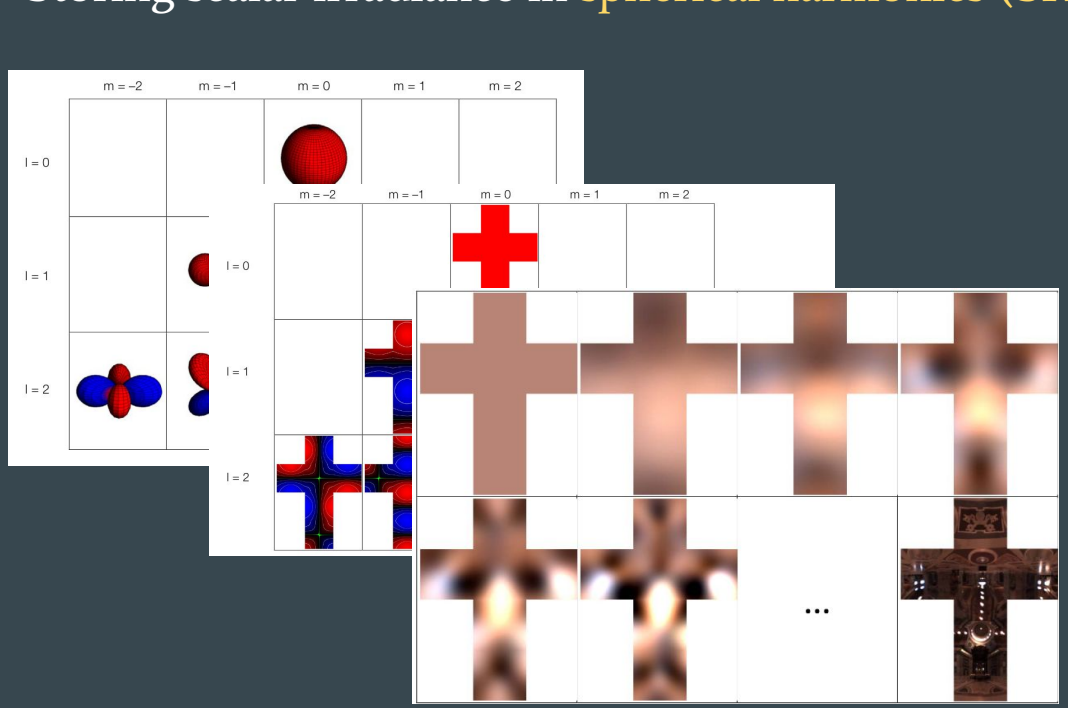
1 INTRODUCTION

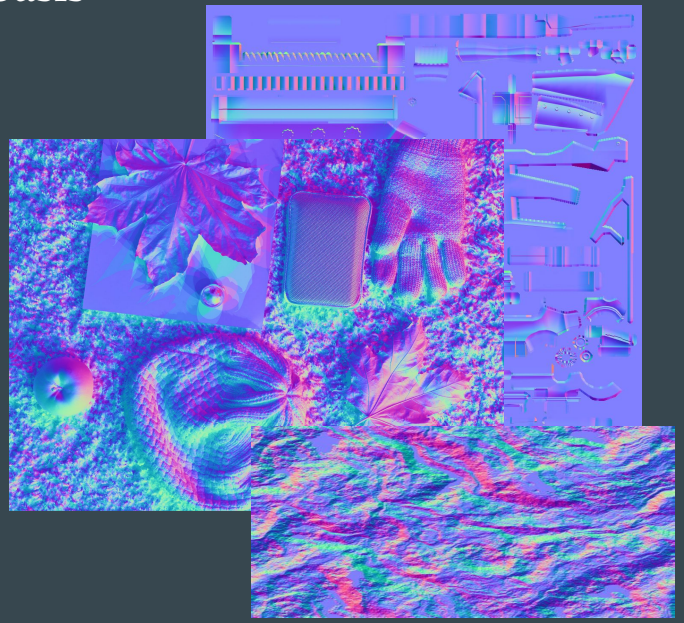
Since global illumination is often too expensive to compute in real-time, it is common for video games to precompute some light transport paths and store them on surfaces for static objects and in volumes for dynamic objects and effects. Lightmaps represent lighting response stored in textures and are often directional [Chen 2008; McTaggart 2004]: the data stored at each surface location can be evaluated using a higher frequency normal map, to more accurately represent the surface response. This is commonly used for diffuse
ssy reflectance can also be handled. Different rep-
ased: spherical harmonics, or subsets of spherical
cted the hemisphere, spherical Gaussians [Neubelt
5] and simple models of incident radiance. In this
n a popular model, Ambient Highlight Direction,
. It encodes a single High Dynamic Range (HDR)
highlight direction and HDR highlight color for
ght. While various forms of spherical harmonics
er quality, they require significantly more coeffi-
ly in HDR lighting environments. Unfortunately,
near representation, so simply interpolating the
esult in visual artifacts. We present two alternate
hat explicitly encode irradiance and interpolate it
correct, pushing the non-linearities outside of this
ow why the artifacts occur when naively interpo-
meters and how our parameterization addresses

to use spherical harmonics internally when pre-
g data, even if the final format does not use them.
meters from SH is relatively straightforward, but

Directional lighting encoding

Storing scalar irradiance in spherical harmonics (SH) basis

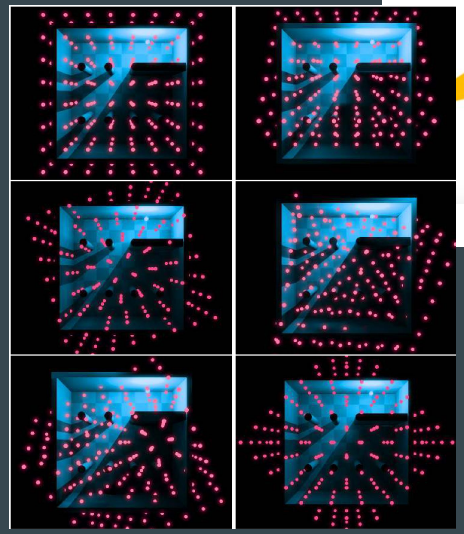




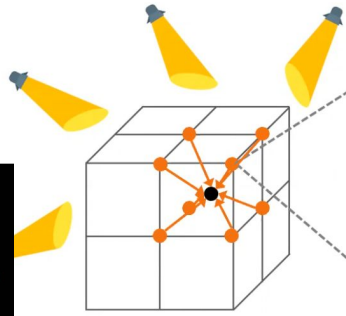
Local Light Grids (LLGs)

Used for small or intricate props

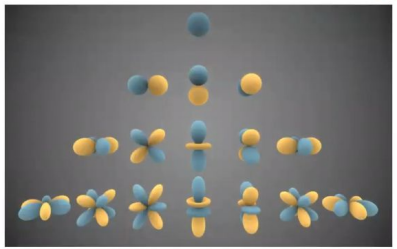
We use Euclidean grid



- Computing SH coefficients and gradients on a sparse 3D grid
- Interpolating at any intermediate shading point



Spherical Harmonics



Coefficients $L_{lm}(\boldsymbol{x}) = \int_{Q(\boldsymbol{x})} Y_{lm}(\boldsymbol{\omega})\,\mathrm{d}\boldsymbol{\omega}$

$\partial_x L_{lm}(\boldsymbol{x}),\ \partial_y L_{lm}(\boldsymbol{x}),\ \partial_z L_{lm}(\boldsymbol{x})$

Baking light for moving objects (GLG)

Dynamic objects do not affect global illumination

But precomputed global illumination influence them

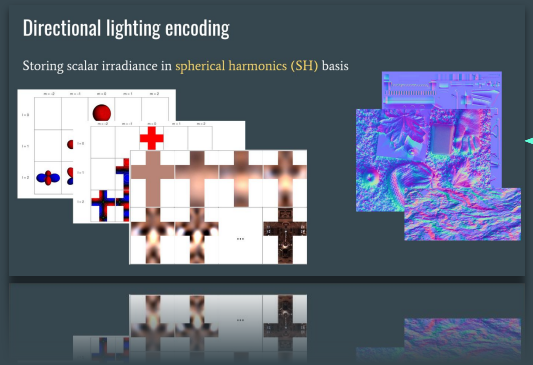
We need to sample arbitrary points in space

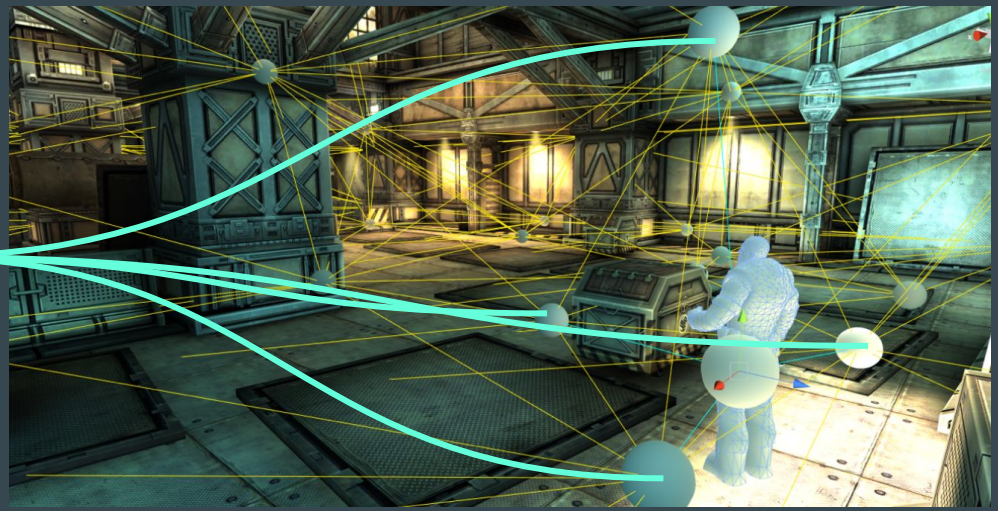
To evaluate static lighting at any position

Global Light Grids (GLGs)

Used for small or intricate props

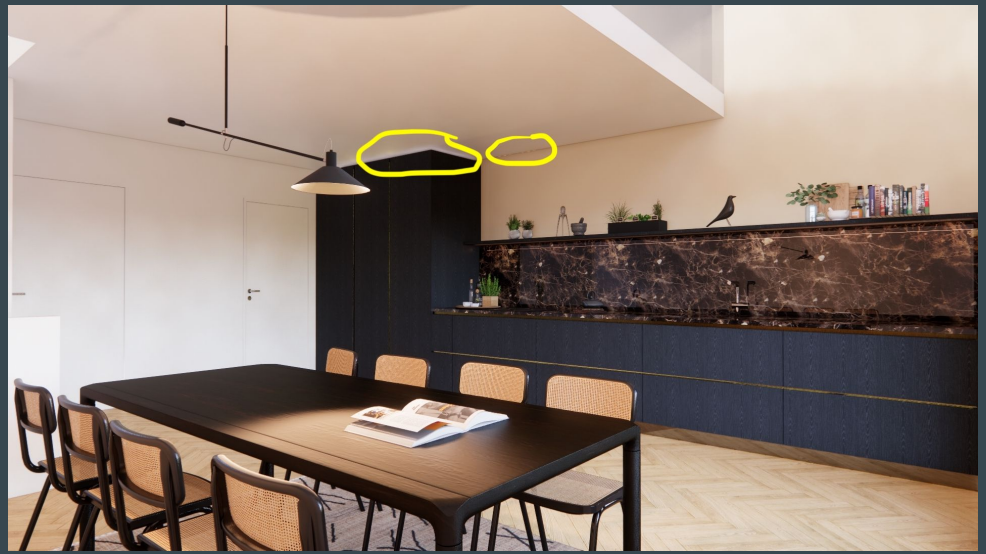
We use tetrahedral grid with variable distribution across the map





Fighting light leaking

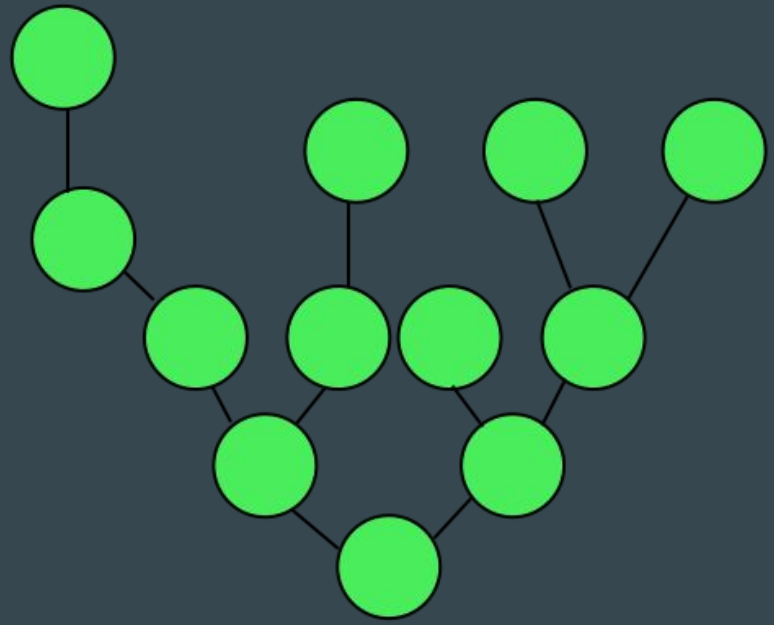
Store coarse visibility information per tetrahedral face and use it to cull non-visible probes



Baking via series expansion of rendering equation*

One bounce at a time, for a whole map

Reusage of information computed from the previous bounces



Player-driven dynamic lighting updates

Goals:

- Preserve the performance and memory characteristics of the static solution
- Make system extensible
- Make system capable of supporting complex lighting changes

Dynamic Light Sets (DLSs)

Make some sets of primary lights togglable

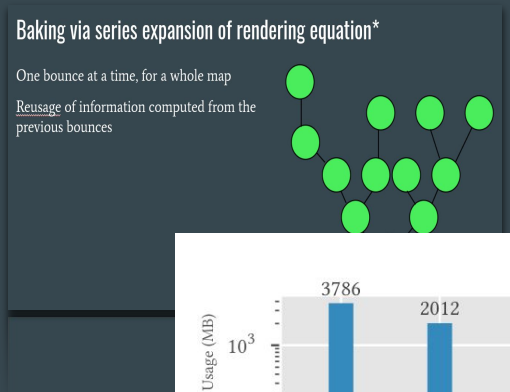
Solution:

1. Base baking (Lb): ignoring all dynamic sets
2. Run separate pass with just the relevant lights enabled
3. Associate blend weight (ωj) associated with each DLS
4. Calculate final lighting at run-time:

$$\mathcal{L} = \mathcal{L}_{\mathrm{B}} + \sum_{j} \omega_j \cdot \mathcal{S}_j .$$

Minimal overhead via sparse lighting storage

To find relevant receivers, each DLS computes D lighting and take 2 bounces via series-expansion baker. Compare intensity(indirect[texels lit by D]) with threshold



We should combinate light sets faster

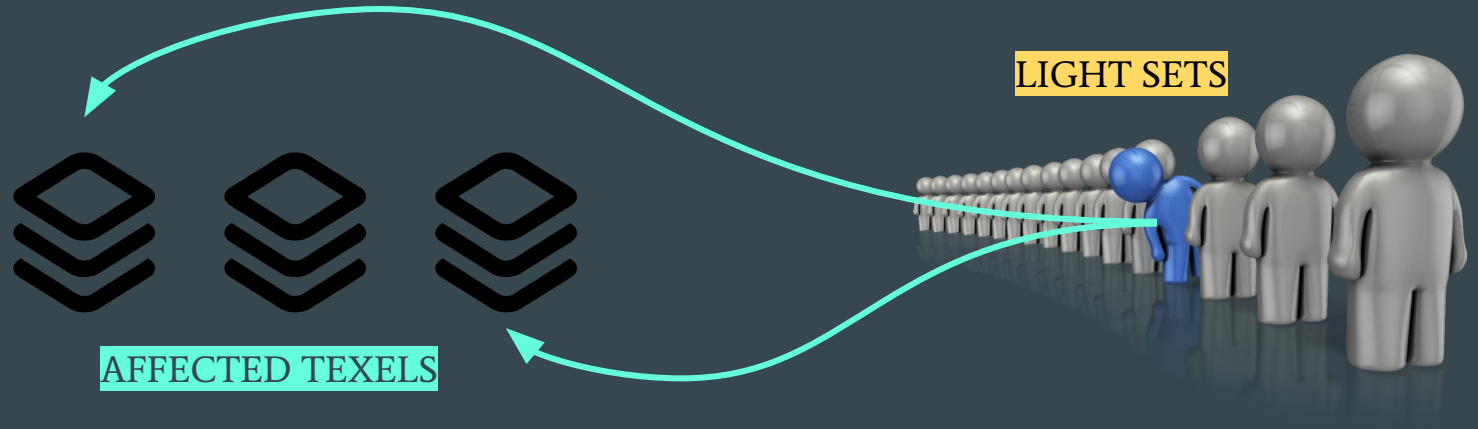
- Avoid problematic variable-length data structures
- Keep any memory overhead low
- Skip disabled DLS efficiently
- Skip updates for texels if corresponding LSs are unchanged
- Avoid read-modify-write overhead if possible

Preprocessing step

Cluster texels by the LS they are affected by

List of LS combinations with a set of affected texels each

We can then generate one compute dispatch per combination



Dispatch step

- ✓ List of LSs for texel was not modified? We are free to skip!
- ✓ Easy to skip turned off turned off LSs for a given dispatch
- ✓ Read-modify-write resources not required
- ✓ We avoided rewriting texels and variable-length data

Further modifications

The indoor lighting can be dominated by the light flowing through the door when opened, making it an important effect to compute

Only base bake

Lighting flowing through door

Doors as dynamic light sets (DLSs)

Challenge:

Express the lighting change as an additive component of the base lighting

$$\mathcal{L} = \mathcal{L}_{\mathrm{B}} + \sum_{j} \omega_j \cdot \mathcal{S}_j .$$

Simplifying assumptions

1. Only “closed” and “opened” state
2. Current “opening angle” controls weight of door’s DLS lineary
3. Disregard any light bouncing off the door in closed state
4. For each DLS and door’s DLS all other doors are closed

Light leaking through open door due to ignoring the door geometry in the ‘open’ state

Door ‘path guiding’ to reduce baking time

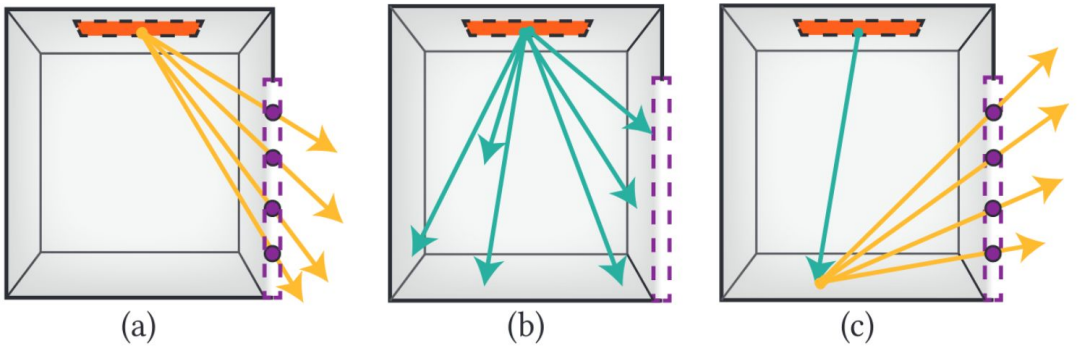


Fig. 7. For each receiver (red), we compute the lighting flowing through the door *directly* (a) by sampling points on its bounding box and casting rays towards it, and one bounce of indirect door lighting by steering final gather rays (b) towards regions of the room that receive strong direct lighting through the door (c).

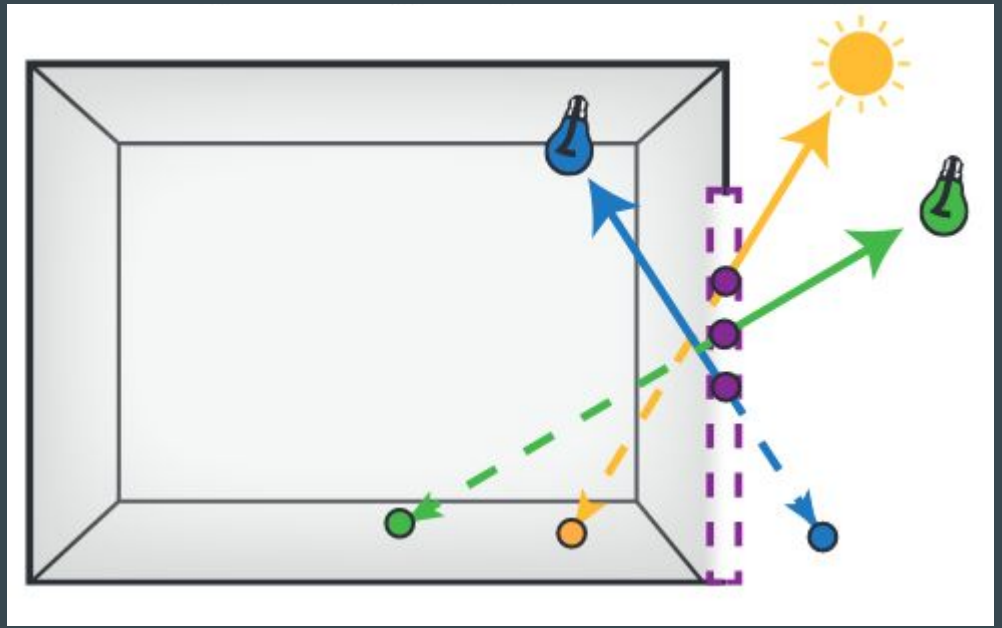
Shadow photons for path guiding


Door's bounding box samples


Shadow ray


Towards photon map texel

We send rays towards regions where the density of photons is high



Performance and memory statistics

	Level Statistics							Performance Statistics		
Level	# LM Texels	# LLG Probes	# GLG Probes	# DLS	# Doors	Memory (no DLS)	Memory (DLS)	Bake (no DLS)	Bake (DLS)	Run Time (sync)
Estate	2,637,824	50,805	229,725	132	0	28 MB	77 MB	10 / 20 min	33 / 133 min	1.47 ms
Consulate	3,276,800	32,783	229,250	61	0	32 MB	61 MB	9 / 18 min	19 / 46 min	0.61 ms
Townhoused	4,194,304	16,524	151,408	24	1	37 MB	50 MB	6 / 14 min	11 / 31 min	0.43 ms
Safehouse	2,363,392	77,110	232,617	14	3	27 MB	41 MB	9 / 18 min	12 / 34 min	0.42 ms
Yard	2,097,152	32,183	135,209	6	0	12 MB	27 MB	5 / 9 min	6 / 11 min	0.21 ms

Conclusion

Uberbake is extension of a traditional static light baking pipeline that allows to use it for modern AAA games etc.

This system works on the complete set of target hardware, ranging from high-end PCs to previous generation gaming consoles, allowing the use of lighting changes for gameplay purposes.