Scalable Real-time Global Illumination for Large Scenes

Anton Yudintsev
Gaijin Entertainment

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Lighting in Enlisted

- Highly dynamic environments
  - Full time of day
  - Huge change in climates and seasons
  - Weather conditions
- Dynamic scenes
  - Millions of entities
  - Large area (16..64 sq. km)
  - Destructible environments
  - Buildable fortifications
  - Big scale

Time of Day and weather condition

Detail scale for 64 sq km locations

Challenges

- Time of day, weather
  - Need for correct lighting that looks right

- Immersion of the player
  - Light needs to be ‘right’ everywhere
  - Large range in conditions across a single map, with interiors and exteriors

Challenges

- Huge scale + time of day + weather conditions
  - Unable to pre-bake everything
  - 64sq km * (at least) 4 time of days * 4 weather conditions = at very least 8 Gb of compressed light data,
  - with inability to change scene
- Correct lighting affects gameplay
  - Especially in indoors
  - Making screen-space approach unacceptable

History and overview

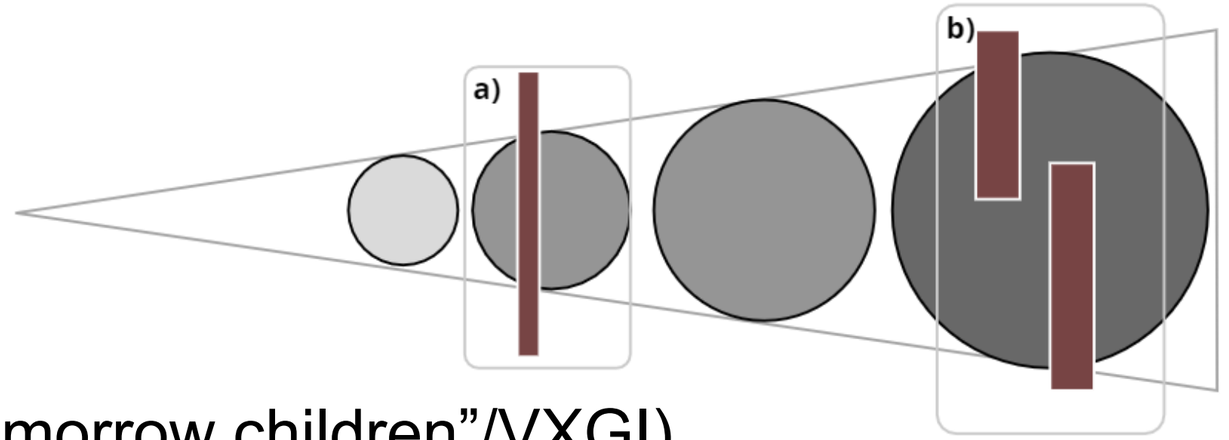
- Lightmap/Static probes (or Volume maps) - pre-baked for single time/weather
- G-buffer probes, different approaches (usually pre-baked) - sparse/low-detail, static scenes, significant time of iteration
- Voxel Cone Tracing Global Illumination - thin walls pass light, usually - one bounce (otherwise expensive), limited number of light sources (otherwise too expensive).
- Virtual Point Lights/RSM/Imperfect shadows – performance cost increases with light count, one bounce
- Light Propagation Volumes Global Illumination - one bounce, limited number of light sources and light types
- Shared g-buffer “surfels” per probe – static scenes, heavy pre-compute

Initial approach

- We have tried a few existing techniques first:
- Voxel Cone Tracing Global Illumination,
- Light propagation volumes,
- Dynamically placed light probes,
- in combination with screen space techniques

Initial approach

- Voxel Cone Tracing was the closest thing, but suffered from few drawbacks:
  - Single bounce
  - Limited light count(performance cost)
  - Thin walls bleed the light (or darkness)
  - Tricks to voxelize scene correctly (see “Tomorrow children”/VXGI)
  - Performance is acceptable in ¼ or 1/16 resolution, temporal SSAA (requires denoising)



Lighting in Enlisted: observation

- Light and weather changes slowly
  - Even ‘cinematographically’ scaled is 2-4 real hours for full day-night cycle
- "Short-time” GI effects are usually still minutes long
  - Except for highly dynamic lights (moving or blinking)
- Camera movements speed is limited
  - Character movement is limited to 20km/h
  - Ground vehicle movements is <70km/h
- Dynamic scene changes are infrequent
  - Destruction happen may be once per 10 seconds at least

Lighting in Enlisted: observation

- A lot of data is available in g-buffer
  - The most important part of the world is where player look now (or recently)
  - And in gbuffer for environment light probes
- While ‘general lighting information’ is also important, it doesn’t have to be very detailed.
  - While we still keep “fair play”
  - screen space only solutions improves details, but can’t capture that

Our solution – in one page

- Voxel representation of a scene (like Voxel Cone Tracing)
- Initially voxelized lit scene around the camera as good as reasonably can, be as fast as possible
  - Collision geometry
  - lower LoDs of entities
  - Heightmap data
- Feedback voxelized lit scene from screen gbuffer!
- Visible irradiance volmap is (partially) recomputed each frame, with a honest brute-force raycasting (no light leaking)

Our solution – first results

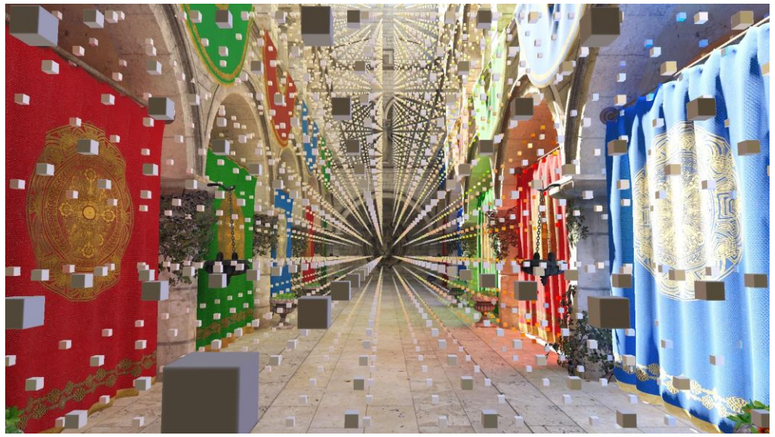
- Multiple bounce with indirect shadowing!

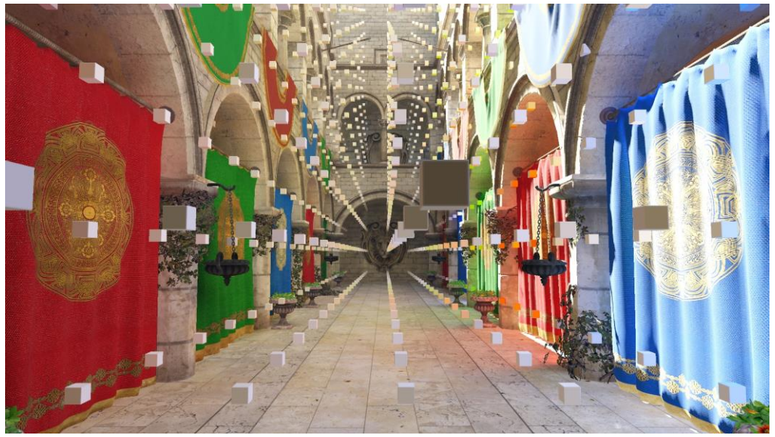
- Average performance cost is only <1.6msec on Xbox One in medium quality, 0.7 msec on low quality or asynced!

Irradiance map

- We store irradiance in nested volume maps around the camera (3d-clipmap)
- Each cascade is ~64x32x64
- Cell size is 0.45m*3^i (or 0.9m*3^i on low-end devices, i – cascade no)
- We have chosen HL2 Ambient Cube basis
  - Non-orthogonal basis
  - But very GPU friendly to sample from
  - Can be easily changed to other basis’

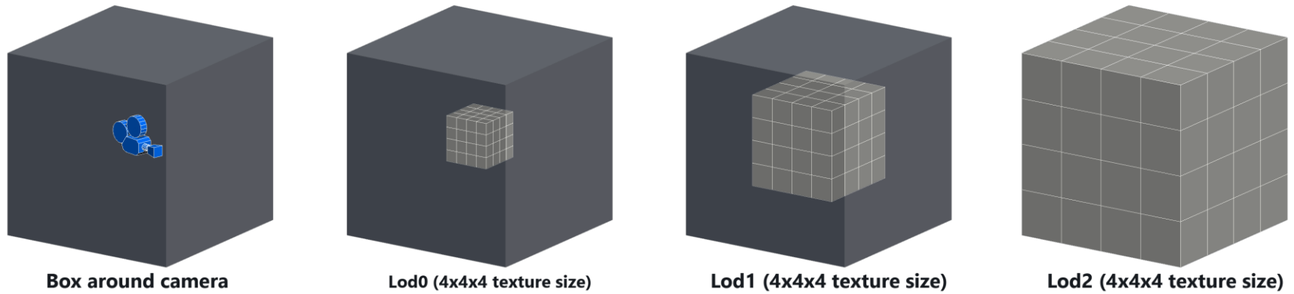
Irradiance map: Sponza





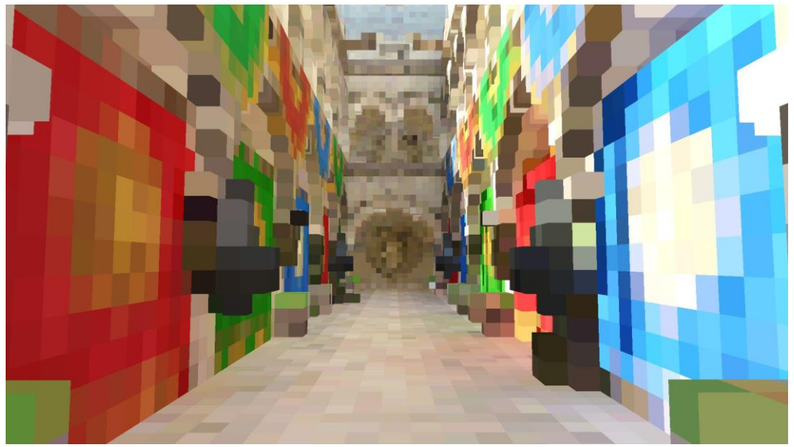
Basic scene parametrization

- We store scene in nested volume map around the camera (3d-clipmap)



  - Each cascade is 128x64x128
  - Cell size is 0.25m*3^i (or 0.5m*3^i on low-end devices, i-cascade no)
  - We store either completely lit results, or lighting+albedo in two volume textures

Scene parametrization: Sponza



Initial scene fill

- When camera moves, we ‘fill-in’ new voxels ‘toroidal way’ (similar to texture WRAP)
- We fill this new voxels with Heightmap data, and either collision geometry (vertex colored) or low-level LODs of entities.
- We immediately light this new voxels with sun, indirect light irradiance and most important lights in the area

Scene feedback loop

- On a medium settings the scene is constantly updated with stochastically chosen 32k g-buffer pixels
- For each stochastically selected g-buffer pixel, we lit it, using it’s albedo, normal and position with both direct light and indirect irradiance map
- We update voxelized scene representation with this new lit color using moving average

Scene feedback loop

- This provides feedback loop, as we update scene voxels with relit g-buffer pixels them using current irradiance volume map;

and irradiance volume map is updated with current scene

voxels

- It is not only gives multiple bounce, but also solves voxelization issues (walls thinner then 2 voxels and accuracy)
- Additionally, environmental probes (when they are rendering) provide more data which wasn’t captured by ‘main’ camera

Irradiance map initialization

- When camera moves, we fill-in new texels (probes)
- For finer cascades we copy data from coarser ones
- For “scene-intersecting” and coarsest cascade map, we trace 64 rays to get better initial approximation
- We mark their temporal convergence weight with magic (“not-really-computed”) value, so they will be relit as soon as they get visible

Irradiance map – computation loop

- We stochastically choose several hundred visible ‘probes’ (positions) in our irradiance volume map
- The probability of selection depends on visibility of the ‘probe’ and convergence factor of probe (how much it changed during last time)
- For selected ‘probes’ we raycast 1024-2048 rays (depending on settings) within our scene, accumulating results with moving average

Sponza: result

Irradiance map - computation queues

- In order to converge fast, it is important to have different queues for different “probes” in irradiance map
- First-seen, never-computed to be computed asap, even with less quality. We use 256 rays, but queue of 4096 probes
- Non-intersecting scene probes not participating in light transfer, but we still need to compute them for dynamic objects, volumetrics, and particles. We use 1024 rays, and queue size of just 64-128 probes.

Initial lighting – chicken-and-egg problem

- When camera teleports, all cascades around it is invalid. So we can't really light initial scene with irradiance (for second bounce) and we can't calculate initial irradiance without initial scene.
- We do that in two passes.
- Voxelize scene, lit it only with direct light
- Calculate irradiance for sky light and second bounce, than re-voxelize scene
- Rarely happen (cut-scenes)

Rendering with irradiance map

- Choose the best possible cascade
- Sample three out of six irradiance volume map textures based on sign of normal (see HL2 Ambient Cube).
- On border blend it with next cascade
- Same in deferred and forward passes (and volumetric lighting)

Rendering diffuse with irradiance map

Listing 1. Ambient Cube Sampling

```
float3 sampleAmbientCube(float3 normal,//normal
  float3 tc,//(0..1) address in one cascade
  int cascade_index,// cascade index
  const int cascades_count// cascades count
  )
{
  float3 nSquared = normal*normal, isNegative = normal<0 ? 0.5 : 0;
  tc.z *= 0.5;
  float3 tcz = tc.zzz + isNegative;
  tcz = tcz*(1./cascades_count) + float(cascade_index)/cascades_count;//

  return nSquared.x * color_x.SampleLevel(LinearFilterSamplerState, float3(tc.xy, tcz.x), 0).rgb +
         nSquared.y * color_y.SampleLevel(LinearFilterSamplerState, float3(tc.xy, tcz.y), 0).rgb +
         nSquared.z * color_z.SampleLevel(LinearFilterSamplerState, float3(tc.xy, tcz.z), 0).rgb;
}
```

Environment Specular/Reflections:

- GI scene can be darker than nearest environment probe. Without adjustment, this will lead to incorrect reflections/specular.
- The simple fix is proposed in “Volumetric Global Illumination at Treyarch”

```
float maxSpecular = diffuseGILum * maximumSpecValue;
float3 reflection = cubeMapSample *
   adjustedMaxSpec / (maxSpecular + luminance(cubeMapSample) );
```

- good on rough surfaces

Environment Specular/Reflections:

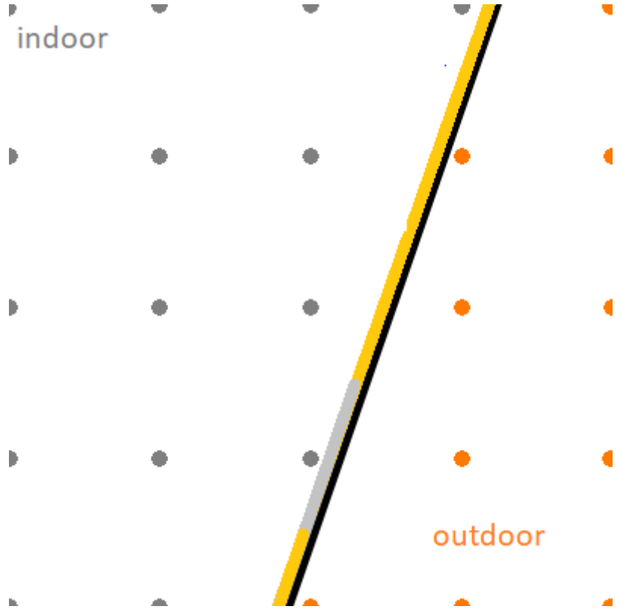
- We can also make Voxel Cone Tracing for smoother surfaces.
- Although that is significantly slower than the simple approach above, we only need that at “smooth enough” pixels and where SSR failed.
- Voxel Cone Tracing for Specular is way faster than for Diffuse because just one cone is probably sufficient.
- “Highest” settings only.

Honest raytracing = no light leaking?

- Voxel Cone tracing can (and will) lead to light leaking on thin walls
- Even for specular light it can be noticeable, so we still adjust maximum brightness.
- Diffuse GI doesn’t suffer from light leaking, because we honestly trace thousands of rays, not simplified “cones”
- Almost enough…

Interiors and outdoors: trilinear filtering

- Regular grids suffer from light/dark bleeding between indoor and out door due to trilinear filtering of irradiance map.
- Normal filtering* can hide it but, some issues still appear (especially on corners of thin walls)

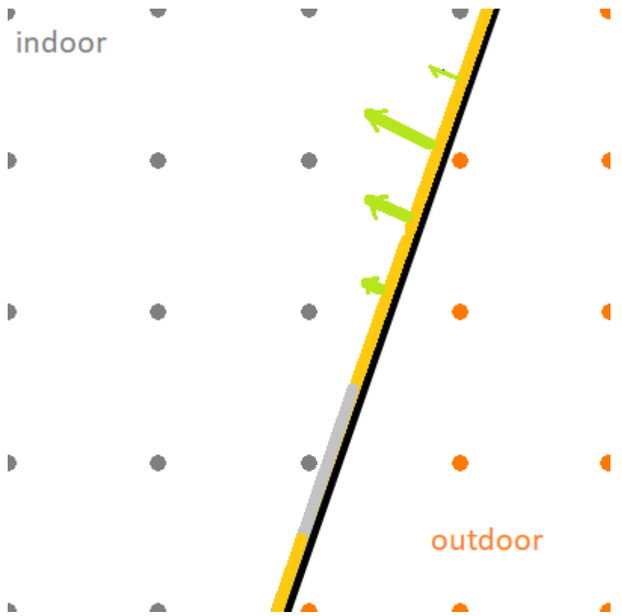


* see "*Multi-Scale Global Illumination in Quantum Break", Siggraph 2015*

Light leaking example

Interiors and outdoors

- We add convex for indoor volumes, and this allow two possible enhancements:
  - filtering where we sample in one set of irradiance maps
  - two sets of irradiance volume maps (indoor and outdoor)
- We have chosen the first method, due to it's simplicity, although filtering sample locations resulted in additional performance cost when rendering



Leaking:

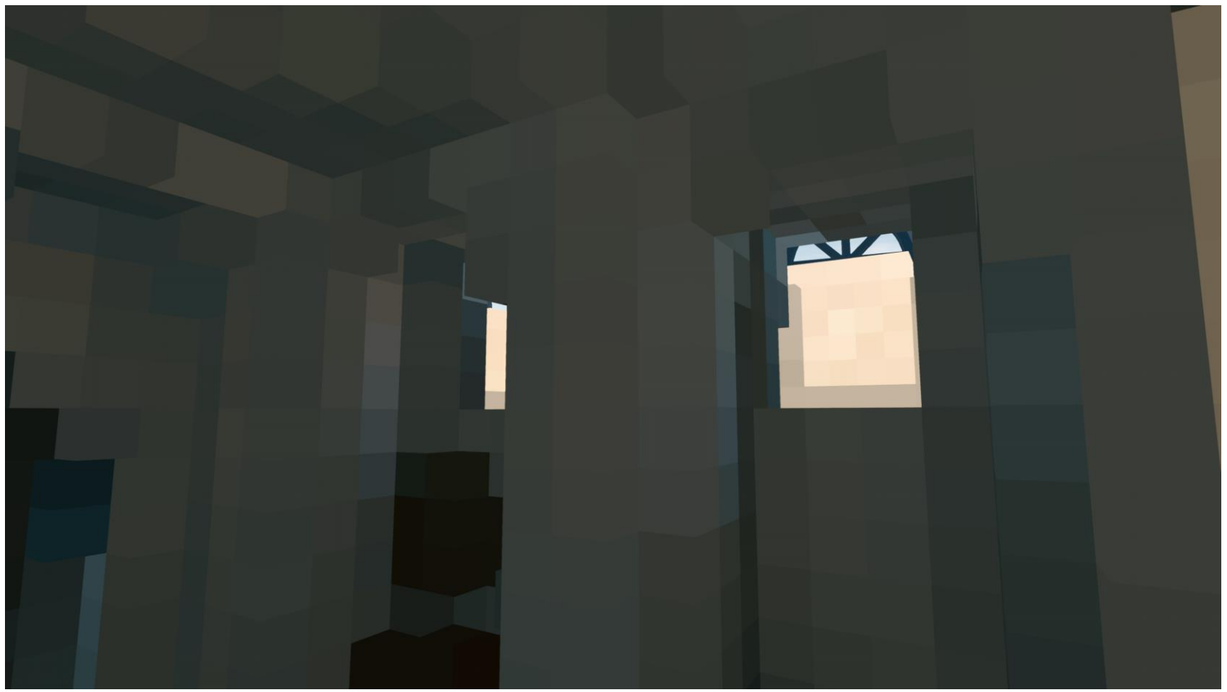
No leaking with convex offset filtering:

Interiors: over-darkening

- Inflation of opaque geometry with voxelized scene, can result in some places to be darker (small windows can collapse).

Interiors: over-darkening

- So we add “holes/windows” volumes (which won’t be filled with voxels ever)



Interiors: over-darkening

Interiors: with “windows” convexes

Other tricks and enhancements

- Cluster ray directions (to reduce branch divergence in different compute threads)
- Use current frame occlusion to not lit completely invisible probes/texels of irradiance maps
- Store ‘transparency’ for “a jour” geometry – fences, foliage.
- As we keep data premultiplied by transparency, this additionally allows us to create volumetric lights

Using HW RT caps to improve quality

- Raycasting voxel grid suffers from inaccuracies, in both precision of hit location, and quality of sampling data. Using modern HW RT can easily enhance the quality and still keep performance.
- Trace available (collision or LOD) data to get correct hit position.
- We calculate incoming ray light using actual hit normal and position, and so store only albedo in scene representation.


REALTIME GI BUDGET
~0.6MSEC ON XBOX ONE X, 4K
~1.7MSEC ON XBOX ONE, FULLHD

Global Illumination: conclusion

- Consistent indirect lighting with multiple light bounces
- Adjustable quality
  - From low-end PC to ultra-high-end HW support
  - Yet not affecting gameplay
  - Scalable detail size, as well ray-tracing quality.
- Dynamic (in a way)
  - Blow up a wall, destroy a building and light needs to flood in. Build a fortification – and it produce reflex and indirect shadows. Rapid iteration.

Pros:

- Maximum location size – Unlimited
- Number of light bounces – Unlimited
- Number of light sources – Almost unlimited
- Types of light sources — all that engine supports (plus some volumetric lights).
- No pre-bake, very small additional asset production overhead
- Day-Time changes, destruction and construction support

Limitations and requirement:

- Fast moving objects are not participated in light transfer (just receive light)
- Only (semi-)static lights – light can only change slowly.
- Smallest possible captured detail is limited with irradiance map resolution
- Deferred pipeline (at least lighting and ‘dynamic’ layers)
- Some Lower LOD/Collision data should be available on GPU

Memory footprint

Irradiance map (3 cascades)	13.5mb
Scene representation	16.5m (33mb with improved quality)
Temporal buffers	<1mb
Total	~31Mb

Performance breakdown

Movement amortized: 0.1-1msec, but each 8-16 frame	~0.1msec
Scene feedback:	~0.02msec
Irradiance map update • can be async computed • or even disabled	1.2msec
Lighting additional cost Full HD	0.3msec
Total async or 1-bounce: **Sync N-bounce:** **(measured on Xbox One)**	**0.7msec** **~1.6msec**

Further improvements

- It is possible to perform per-pixel ray-tracing (especially with HW RT) with some temporal denoising. In our experience 6spp is enough to achieve good quality. It is still rather slow (additional ~12msec on 1440p on 2080), but can be rendered at half res. No light-leaking, no filtering needed, way more detailed light.
- Or can be combined with screen space techniques.

Questions?