The state-of-art of Dynamic Global Illumination in video-games

Benoit TOTH
benoit.tothdenimal@gmail.com
CNAM-ENJMIN
Angoulême, France

ABSTRACT

Realistic Global Illumination is extremely costly. Video-games often preferred to pre-compute it, sacrificing dynamic lighting or moving geometry. However, more and more fully dynamic solutions are emerging. In this article, I will discuss the state-of-art of Dynamic Global Illumination methods that are used in video-games, giving you a brief explanation of how they work and listing their pros and cons.

CCS CONCEPTS

• **Computing methodologies → Computer graphics**.

KEYWORDS

global illumination, real-time rendering, indirect lighting, GPU

ACM Reference Format:
Benoit TOTH . 2021. The state-of-art of Dynamic Global Illumination in video-games. In *Angoumême.* ACM, New York, NY, USA, 5 pages.

1 INTRODUCTION

Global illumination (GI) always has been a key part of rendering to achieve realistic scene and it is no surprise that video-games are always trying to improve it. GI is the combination of direct lighting, light directly emitted from a surface or received by a light source, and indirect lighting, light that is bouncing from surface to surface. And indirect lighting is the main problem that we struggle to solve in real-time to achieve good-looking GI.

1.1 The rendering equation

$$L_o(\mathbf{p},\mathbf{v}) = L_e(\mathbf{p},\mathbf{v}) + \int_{\mathbf{l}\in\Omega} f(\mathbf{l},\mathbf{v}) L_i(\mathbf{p},\mathbf{l})(\mathbf{n}\cdot\mathbf{l})^+ d\mathbf{l} \tag{1}$$

The rendering equation (1)[6] has been proposed in 1986 by Kajiya. The term $L_o(\mathbf{p},\mathbf{v})$ represents the outgoing radiance from the point $\mathbf{p}$ in the direction $\mathbf{v}$. In radiometry, the basic unit is the radiant flux, which measures the energy over time, the power. Radiance measures how much radiant flux there is on a surface, per area and per steradian. Therefore, the radiance coming from the point $\mathbf{p}$ following the direction $\mathbf{v}$ to a certain pixel of the screen will represent how bright the pixel is. To compute correct GI, we need to resolve this equation for every pixels. Most of the methods I will discuss here propose their own way to solve it.

In the equation we add the emitted radiance $L_e(\mathbf{p},\mathbf{v})$ to the integral of direction $\mathbf{l}$ over the hemisphere Ω centered around the normal $\mathbf{n}$ where :

- $f(\mathbf{l},\mathbf{v})$ represents the BRDF (Bidirectional reflectance distribution function) that tells us how light is reflected
- $L_i(\mathbf{p},\mathbf{l})$ is the incoming radiance from the direction $\mathbf{l}$
- $(\mathbf{n}\cdot\mathbf{l})^+$ is the dot product clamp to positive values

The number of directions over the hemisphere is infinite. This is what makes indirect lighting hard to compute. We have no choice but to make approximations here.

1.2 Different ways to compute Global Illumination

There are full solutions algorithms that compute real-life looking lighting like path tracing [6] or photon mapping [5] . They can compute even the most complex lighting effects like caustics and easily handling both specular and diffuse materials. However those are offline methods that are not usable for real-time application and I won't discuss them here.

I won't talk about pre-computed solutions neither like light maps [1] or Precomputed Radiance Transfert (PRT) [15] because they restrict us to static lighting or geometry or both and cost a huge production time by their needs to be pre-computed. Still, they are often preferred by video-games because they doesn't sacrifice performance like dynamic GI could.

What I will discuss here are entirely fully dynamic GI methods that allow us a total expression in our scene and immediate result. However, they come at a cost. Trade-offs are often made, they are more expensive and common problems like light leaking appear. It won't be an exhaustive list of every methods because certain are not used anymore and also some of them are variation of those I will present you.

2 SCREEN SPACE GLOBAL ILLUMINATION

Generally, the cost of GI increases with the complexity of the scene. This is why Screen Space Global Illumination (SSGI) always has been very popular among video-games. Non Screen Space methods usually use massive amount of intersection tests, which requires a lot of querying when dealing with large and detailed scenes. Whereas Screen Space methods are bounded to the resolution of the image, and therefore remain very affordable.

2.1 Screen Space Ambient Occlusion

One of the very first GI effect computed in Screen Space is ambient occlusion. It is a non-realistic but plausible effect based on the observation that a point will receive more of less lighting if accessible.

$$k_A(\mathbf{p}) = \frac{1}{\pi}\int_{l\in\Omega} v(\mathbf{p},\mathbf{l})(\mathbf{n}\cdot\mathbf{l})^+ d\mathbf{l} \tag{2}$$

k_A is the occlusion factor that will tells us how much the point $\mathbf{p}$ is occluded. To compute it, we have to ingrate over a set of direction $\mathbf{l}$ where :

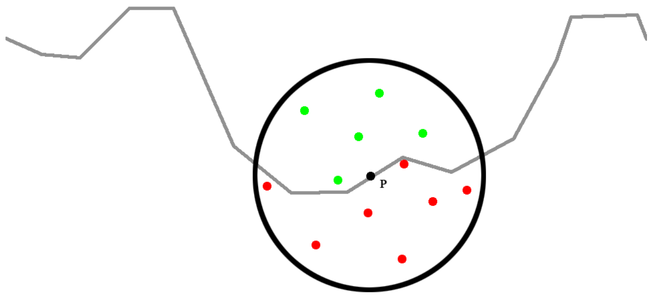

Figure 1: SSAO Crytek's method

- $v(\mathbf{p}, \mathbf{l})$ is the visibility function
- $(\mathbf{n} \cdot \mathbf{l})^+$ is the dot product clamp to positive values

Various ways exist to compute the visibility function. Crytek came up with simple solution in screen space that produces pleasing result and is computationally cheap [12]. They sample a point in a sphere around the point **p**, and then compare its depth with the depth buffer to determine whether or not the point is occluded. Over the years, other methods have been proposed like Horizon Based Ambient Occlusion that produces more accurate and pleasing result. However they all shared the same limitations, they are limited to the screen information, thus producing only approximations. They are also not real GI, but only an effect of it.

2.2 Screen Space Directional Occlusion

Screen Space Ambient Occlusion (SSAO) doesn't take into account the incoming light and is only adding plausible details to shadow. We can make a slight improvement to the algorithm so it can produce color bleeding, directed shadow and even one bounce of global illumination. Screen Space Directional Occlusion (SSDO) [14] does two things more than Crytek's method.

First for every sample points considered as non-occluded, it accumulates radiance coming from the direction of the sample point to **p**. It gives us oriented and colored shadows. Then for every sample points considered as occluded, they find the point above them that occlude them and compute its reflected radiance from direct lighting to the point **p** which give us color bleeding but also one bounce of global illumination.

SSDO is a simple and cheap way to compute SSGI. There are more implementation that can give better result at a higher cost using ray marching against the depth buffer for example. Screen Space Reflections (SSR) are also very popular because its provides nice glossy reflection without ray-tracing.

However screen space methods have their downsides too. Because they are limited to our screen information, it may produce inaccurate results if the occluder appears to be off-screen. Especially for Crytek's method, if an occluder appears to be between the point we shade and our sample point, it won't be take the occlusion into account. It could be solved by using ray-marching or depth peeling but then the computational cost would tremendously increases. Temporal artefacts may also appear, color bleeding and glossy reflection can disappear when moving the camera. Most of the time SSGI is added on top of others GI solutions to add details at lower cost.

3 LIGHT PROPAGATION VOLUMES

Light Propagation Volumes (LPVs) are a solution initially proposed for CryEngine 3 by Kaplanyan[7] in 2009. It can be considered as the evolution of older methods. Virtual Point Light (VPL) which were first introduced with Instant Radiosity [9] and Reflective Shadow Map (RSM) [2].

The idea here is to simulate the light propagating through a grid that discretizes the scene which is stored into a low-resolution volume texture. The grid moves with the camera, it is a local solution. Each cell stores the incoming radiance, to do so they use spherical harmonics functions that allows them to store it from different directions.

After constructing the grid, they inject light into cells by using RSM. RSM are similar to shadow map. We render the scene from the light, but instead of only storing the depth like we would with shadow map, we also store position, albedo, and normal. Afterwards, those information would normally be used to directly light up the scene with one bounce of indirect lighting. Inspired by Splatting Indirect Illumination technique [3] here they use a subset of RSM pixels as VPLs. VPLs are basically point light created on the fly. However, shading points with VPLs quickly become expensive as their number increases. With LPVs, each VPL sees their radiance information stored into its appropriate cells of the grid.

The next step is to propagate light throughout the grid. It is an iterative step where at each step, cells propagate light to adjacent cells in the 6 axial-directions. Afterwards, we can shade the scene by using the radiance stored in the grid. However occlusion is not taken into account yet. A very similar process is done in parallel to calculate an occlusion factor into a geometry volume.

This method provide plausible dynamic GI achievable in real-time that can compute multi-bounce by modifying the propagation function and blurry glossy reflection by gathering the light through the grid in a certain direction. A cascaded version of this method has been proposed [8] for larger environment when the grid is too small to cover all the viewed scene, initially they used SSDO on top of it. Still, this method is limited by the resolution of the grid. Coarse representation of the scene limit us to low-frequency lighting and causes light leaking which were at the time acceptable issues to achieve dynamic GI. Nowadays this method tends to be left-a-side, others methods like Voxel Global Illumination are preferred.

4 VOXEL GLOBAL ILLUMINATION

Proposed by NVIDIA in 2011, Voxel Global Illumination (VXGI) has only been recently used in video-games. Initially too expensive, the method has been continuously improved by NVIDIA over the years. Their latest work is available as a plugin for Unreal Engine and

CryEngine also have their own implementation for their current dynamic GI solution.

The approach is to build a pre-filtered hierarchical voxel representation of the scene geometry in a way that allows an approximate voxel cone tracing method. NVIDIA initially used a sparse voxel octree but decided some years later to switch to a clip map [13] that allows smarter reconstruction when dealing with dynamic objects. To voxelize the scene, they rasterize meshes 3 times along the axis-coordinates in a resolution of the maximum size of the clip map. Each voxel contains textures that represent opacity and emittance values, their number may vary depending on the quality targetted. Those textures are computed rapidly after the construction or the update of the clip map. Afterwards, they inject light into voxels by using RSM and then perform approximate voxel cone tracing at every visible points by stepping along their hierarchical representation to accumulate illumination.

VXGI has the advantage to easily handle diffuse and specular surfaces by respectively tracing cone in every directions or just one. Area light are also easily implemented as emissive surfaces are handle during the process. This method can also compute a high quality ambient occlusion just by accumulating opacity during the tracing process.

Its main downsides are due to its coarse representation of the scene, flickering in the lighting can be observed on moving objects, additional filtering has to be done to mitigate the effect. Also light leaking and pixelated reflections can appear in some situation. It is due to a lack of precision during the cone tracing process. Reflection can be fixed with some filtering, however light leaking can only be prevent by thickening walls which can be annoying in some well-established scenes.

5 RTX GLOBAL ILLUMINATION

As NVIDIA's RTX graphics card came out in 2018, it became possible to achieve real-time performance with ray-traced global illumination. They have dedicated cores that greatly speed up the ray-tracing process. Along with these, Microsoft collaborated with NVDIA to propose DirectX Raytracing (DXR), an addition to the DirectX 12 API that was made to speed up ray-tracing too. Right now, it is still an early technology and a lot of work can still be done with as we will see in the next methods I will present you. Though even after speeding up by a lot the ray-tracing process, path tracing[6] is still way too expensive for real-time performance. We must use a much smaller set of rays although we are trying to obtain similar diffuse and specular GI. Here I will talk about how the Metro Exodus team used DXR and RTX to achieve their dynamic GI. [18]

To start, they cache Ray-traced information like color and hit distance into a light buffer. To do so, they first trace in screen space by using ray-marching and if a ray miss, they trace rays in a direction sample uniformly from a hemisphere above the point oriented with the normal. To choose those rays, they are using a procedural blue noise, that is then reuse later to re-construct rays and accumulate lighting into sphericals harmonics. They produce better result when filtered during the denoising step and gives us directional information that will help to compute specular lighting.

It is important to note that DXR use accelerated structures. We have Bottom-Level Accelerated Structure (BLAS) that will contains the meshes we will trace against, and Top-Level Accelerated Structure (TLAS) that contains the BLAS. The Metro Exodus team choose to rebuild only BLAS for animated or skinned meshes and they do it across multiple frames to mitigate performance cost. The geometry that they contains are about four times smaller than the original ones to reduce memory cost and ease up the intersection tests. TLAS, on the other side, are always rebuilt from scratch, to assure the best quality and compactness. Culling is apply to most of the BLAS contain in the TLAS by using logical visibility, reducing by a huge amount the geometry tested.

The next step is probably the most important. The light information obtained is way too noisy due to the lack of rays count. To overcome this, they use a denoiser. We could use deep learning to do so but this solution is barely explored in real-time graphics application so they preferred to use a convolution method, which is a mathematical tool that can be used to reconstruct signals. If the image is still noisy, Temporal Anti-Aliasing can give us the extra filter that we are looking for.

We obtain a dynamic lighting with affordable performance that can provides at least one bounce of indirect illumination, color bleeding and specular. Additionally to GI, depth information obtained during the ray trace process can give us ambient occlusion information. Compare to SSAO this Ray-trace ambient occlusion (RTAO) is much more aware of enclosure spaces and gives us progressive darkening interior. The main issue of this RTX Global Illumination however, is that computing multi-bounce illumination is extremely expensive due to the exponential number of rays added. One bounce is enough to provide good-looking GI is most cases but interiors can suffer from it if no direct light source is present.

6 IRRADIANCE FIELDS

Initially irradiance probes are a pre-computed solution that allows us to compute lighting information into probes that is distributed all over the scene. Objects then dynamically interpolate lighting information with the closest probes. This allows dynamic geometry but restrict us to static lighting. Another issue can be observed when probes are close or inside geometry. It can cause bad interpolation on the geometry that will result as light or shadow leaking. In 2019, NVIDIA delivers a paper that aim to resolve those two issues with the use of RTX [11]. Inspired by this paper, a solution has also been proposed with the use of Signed Distance Field [4] (SDF) that I will discuss afterwards.

6.1 RTX method

They start by setting up a uniform three dimensional grid of probes. They allow artists to manually move them for better result, though they can avoid most of light leaking without moving them. A cascaded version of this grid is made in the same way as we saw with LPVs to save performance in larger scenes. Each probe stores lighting information into octahedron. They naturally unwraps to a cube so we can use textures. Those textures store irradiance and visibility information. To compute them they trace a set of rays around the probes that will gather information for them. Irradiance is averaged into 6 texels representing the 6 directions of the octahedron.

However, it is not that easy with the visibility. By storing the mean of ray length hit in a certain direction, we may miss a small aperture in a wall for example. What they do, is that they store the sum of the ray length but also the sum of the squared ray length. It allows calculation to detect variance and correctly approximate shading. Finally, during the shading process, we look for the eight probes around the point we shade, and for each probes, we compare its distance to the point and its visibility to decide how much of its lighting information we should accumulate. Still, updating every probes at each frame is extremely costly and not achievable in real-time. What they do is that they update probes across multiple frames at a variable frequency. They can sacrifice ray count and frequency update to reduce and stabilize performance cost on the fly.

This method can effectively produce dynamic diffuse GI with multi-bounce at scalable performance with almost no light leaking. I say almost because thin wall that are about 10 centimeters can still produce light leaking due to depth precision issues. Speculars are not supported although RTX can easily compute reflection on top of it. High-frequency contact shadow are not very well described too but using SSAO on top of it can do the trick. With those additions, this method can match path tracing results.

6.2 Signed Distance Field method

Alternatively, another method proposed a similar approach using SDF to get rid off hardware requirement and to totally eradicate light leaking. To do so, they use a SDF primitives to discretize the scene. Those primitives are packed into clusters to speed up SDF query. At first, cluster are treated as one SDF and if they are not rejected, its primitives are individually considered. Additionally, because of the light weight of this data structure, work is done to put them inside the L1 cache of the GPU and speed up access memory, therefore reducing furthermore SDF query cost. The same grid than before is set up, though if a probe falls inside an object, they relocate it. Probes only store Irradiance, and to sample it, they perform a set of sphere tracing from various direction to intersect with the scene and then use RSM to get the Irradiance value at those intersection points, therefore computing the first bounce of indirect lighting. Multi-bounce is done through multiple frame by interpolating probes between them. When shading a point, they interpolate probes close to it. But before that, they perform visibility tests against the SDF representation of the scene using ray marching. This test is done only one times for every 2x2 set of pixels.

With this method, light leaking is totally gone and it has no hardware requirement. However it is not production ready at all. The SDF representation of the scene is set up manually for the moment, which is a huge amount of work for large scenes. Though they are looking for an automation of this process which can make this method very promising.

7 LUMENS

With the release of the early access of Unreal Engine 5, came Lumens [17]. It is a hybrid GI solution that provides support for non ray-tracing and ray-tracing graphic card. The first one uses SDF representation of the scene to trace against and surface caches to rapidly query light information. The second one focus on smart usage of ray tracing.

7.1 Software Ray-Tracing

For their first solution, they use three methods of ray-tracing. They start with screen traces, and when the ray goes off screen, they traces against pre-computed individual mesh SDF, however this come at a cost. For further away rays they use a more global SDF of the scene. Since Unreal Engine 4, the Mesh SDFs have been improved in term of resolution and memory cost. However those SDF only give us normal and hit point, but they don't have any color or lighting information attached to them. This is where Surface Cache comes into place. It is a huge low resolution atlas that captures meshes information from different direction which is updated in real-time. However it has one big limitation which that meshes can't have complex interior. It is easy to understand why, complex interior can't be capture by just looking at the mesh from different direction. It is important to know that this process was build along side with Nanite, another new feature of Unreal Engine 5 that virtualize geometry. It speeds up by an tremendously amount the surface cache process.

Lumens Software Ray-Tracing provides a dynamic GI solution for non ray-tracings graphic cards with affordable performance on high-end configuration. It supports diffuse and specular material. It has some limitation though. It only apply on static meshes, which are an Unreal Engine object that can't have their vertices animated in any ways, however they can move. Moreover large meshes will have poor representation and have to be divide into smaller modular pieces. Light leaking is also a problem for walls that are too thin.

7.2 Hardware Ray-Tracing

They were inspired by ray tracing methods . Limited by the amount of ray count, image obtained are very noisy and need a denoiser to be usable. Lumens tries to make better use of its low ray count. They do not trace for every pixels of the screen, instead they do what they call Screen Space Radiance Caching. They trace smaller set of pixels. They start with a uniform grid distribution at every 16 pixels then double the resolution of the grid at points where an interpolation test fails with adjacent points. This method has been introduced at Siggraph in 2007[10]. As a result, the image is down sample at $1/16^{th}$ of its initial resolution in the worst scenario. Because they make the assumption that incoming light is coherent, they consider this worse the trade. They also apply some jittering to the grid between frames that will be temporally filtered to cover more pixels. Lighting details are coming from the normal of the geometry, which will be used at full resolution and shaded with the low-resolution incoming lighting. As a result the lighting is already much more detailed than with classical ray-tracing. It is worse to note that Nanite provides extremely high detailed normal that Lumens can profit.

To continue to make better use of ray tracing, Lumens identifies where rays should be fired in priority, such as light source or lit surface. To do so, they re-project last frame lighting information and BRDF then use a probability density function with both of these to determine from where the incoming lighting should come. As a reminder, BRDF is the function that computes the probability of the light to be reflected. For more distant lighting that doesn't appear on screen, a World Space Radiance Caching is done. It places a few probes in a grid in world space, and then traces from them. They are in a much more smaller number that in Screen Space Radiance Caching though. Their result is then interpolate to the rest of the lighting information. To speed up Lumens tracing process, a Nanite proxy representation of the mesh are used which are significantly smaller than the full representation. Too cover the mismatch between the proxy and the full resolution mesh they trace against the screen because what is seen is the full mesh representation. Additionally to these, geometry is stored into a Bounding Volume Hierarchy [16] (BVH) that is built once for static geometry and rebuilt every frame for the rest. As a result we obtain good enough lighting information that can be then interpolated to the rest of the screen and temporally filtered to improve lighting stability.

Lumens Hardware Ray-Tracing provides a dynamic GI on extremely high detailed geometry thanks to Nanite with affordable performance on high-end configuratio. It supports diffuse and specular material. However Lumens is still in early access, therefore its implementation has still room for improvement. It is limited to 10 000 instances, suffers from huge performance cost with masked material, skinned mesh and overlapping geometry and it also doesn't have support for transparency yet.

8 CONCLUSION

Dynamic Global Illumination in video-games has greatly been improved during the last decade. As we saw, past methods tend to continuously inspire other and evolve into more complex solution. And with always more powerful hardware, new approaches always become possible. At our best, we are currently capable to provide result that can match offline algorithms with only few trade off. Light leaking is still a problem but has been greatly reduced in most of cases. Dynamic Illumination is often compute over multiple frames, as a result lighting effect present a bit of latency, and complex GI effect such as caustics are still out of our hands. Latest solutions tend to focus on Ray-tracing now that specialized graphics cards came out, SDF also has been more and more used. For the future, NVIDIA is talking about RTX servers that would stream GI information directly to the player, it is the reason why they restricted their data structure to small texture for their Irradience Fields method.

REFERENCES

[1] Tomas Akenine-Mller, Eric Haines, and Naty Hoffman. 2018. *Real-Time Rendering, Fourth Edition* (4th ed.). A. K. Peters, Ltd., USA.

[2] Carsten Dachsbacher and Marc Stamminger. 2005. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games - SI3D '05*. ACM Press. https://doi.org/10.1145/1053427.1053460

[3] Carsten Dachsbacher and Marc Stamminger. 2006. Splatting indirect illumination. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games - SI3D '06*. ACM Press. https://doi.org/10.1145/1111411.1111428

[4] Jinkai Hu, Milo Yip, G. Elias Alonso, Shihao Gu, Xiangjun Tang, and Xiaogang Jin. 2020. Signed Distance Fields Dynamic Diffuse Global Illumination. arXiv:2007.14394 [cs.GR]

[5] Wojciech Jarosz, Henrik Wann Jensen, and Craig Donner. 2008. Advanced global illumination using photon mapping. In *ACM SIGGRAPH 2008 classes* (Los Angeles, California) *(SIGGRAPH '08)*. Association for Computing Machinery, New York, NY, USA, 1–112. https://doi.org/10.1145/1401132.1401136

[6] James T. Kajiya. 1986. The rendering equation. *ACM SIGGRAPH Computer Graphics* 20, 4 (aug 1986), 143–150. https://doi.org/10.1145/15886.15902

[7] Anton Kaplanyan. 2009. Light Propagation Volumes in CryEngine 3.

[8] Anton Kaplanyan and Carsten Dachsbacher. 2010. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games - I3D 10*. ACM Press. https://doi.org/10.1145/1730804.1730821

[9] Alexander Keller. 1997. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*. ACM Press. https://doi.org/10.1145/258734.258769

[10] Jaroslav Křivánek, Pascal Gautron, Greg Ward, Okan Arikan, and Henrik Wann Jensen. 2007. Practical global illumination with irradiance caching. In *ACM SIGGRAPH 2007 courses on - SIGGRAPH '07*. ACM Press. https://doi.org/10.1145/1281500.1281617

[11] Zander Majercik, Jean-Philippe Guertin, Derek Nowrouzezahrai, and Morgan McGuire. 2019. Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields. *Journal of Computer Graphics Techniques (JCGT)* 8, 2 (5 June 2019), 1–30. http://jcgt.org/published/0008/02/01/

[12] Martin Mittring. 2007. Finding next gen. In *ACM SIGGRAPH 2007 courses on - SIGGRAPH '07*. ACM Press. https://doi.org/10.1145/1281500.1281671

[13] Alexey Panteleev. 2014. PRACTICAL REAL-TIME VOXEL-BASED GLOBALILLUMINATION FOR CURRENT GPUS. In *NVIDIA GPU Technology Conference*.

[14] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. 2009. Approximating dynamic global illumination in image space. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games - I3D '09*. ACM Press. https://doi.org/10.1145/1507149.1507161

[15] Peter-Pike Sloan, Jan Kautz, and John Snyder. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics* 21, 3 (jul 2002), 527–536. https://doi.org/10.1145/566654.566612

[16] Ingo Wald, Solomon Boulos, and Peter Shirley. 2007. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics* 26, 1 (jan 2007), 6. https://doi.org/10.1145/1189762.1206075

[17] Daniel Wright. 2021. Radiance Caching for Real-time Global Illumination. In *Advances in Real-Time Rendering in Games, Part 2*.

[18] OLES SHYSHKOVTSOV SERGEI KARMALSKY B. A. D. Z. 2019. Exploring the Ray Traced Future in 'Metro Exodus'. *In Game Developers Conference (2019)* (2019).