Advanced Rendering Techniques

A review of real-time global illumination techniques for modern hardware

Julius Narvilas

School of Computing Science

Newcastle University

Newcastle

1 Abstract

As the demand for realistic graphics in video games continues to rise, there is interest in efficiently achieving advanced visual effects like global illumination for real-time applications. Rasterization based graphics processing hardware is considered not well suited to simulate light behaviour but is now powerful enough to approximate lighting in different ways. This paper will look into currently available techniques that address the problem using vastly different approaches. The aim is to identify strengths and limitations of each approach and asses the availability on consumer grade hardware.

2 Introduction

Hardware and software improvements related to graphics processing have helped video games to pursue more complex visual effects and raise the bar higher for realistic real-time rendering. Life-like graphics play a big role in enhancing our sense of immersion but simulation of light behaviour that we observe in the real world is not that simple. Some light effects are complex to implement and often do not appear in the environment. On the other hand, indirect lighting is a common phenomenon and has recently gain more traction in video game graphics.

In computer generated graphics, simulation of light that reflects off of objects that are not light emitters is referred to as global illumination (GI). This feature can greatly improve the visual quality of the rendered images but is often not incorporated due to the technical difficulties of implementation. A common solution for GI and many other light effects is ray tracing. It is an expensive technique that is used for rendering at non-interactive frame rates, but can produce high quality results and can be used in pre-process steps to record accurate visual data of static geometry.

An example of combining expensive offline procedures to approximate GI would be the video game “Far Cry 3” (Gilabert & Stefanov, 2012). Ray tracing has been used to prepare video game assets that describe the lighting in various chunks of the environment. Complex but more frame rate friendly techniques are then used to approximate GI. Non-static objects represent only a small amount of the environment,

so the visual effects are fairly accurate and can be achieved at interactive frame rates on console (Xbox 360, PlayStation 3) grade hardware.

A more ideal solution of illumination for real-time applications like video games would be a fully dynamic one, allowing for large amount of non-static geometry and without an expensive offline asset generation step. A technique that would fill this criteria is called voxel cone tracing (Crassin, Neyret, Sainz, Green, & Eisemann, NVIDIA Research, 2011). It utilises the power of a modern graphics processing unit (GPU) to maintain a hierarchical sparse voxel octree version of the scene geometry. This structure is used to sample irradiance and perform cone tracing while maintaining interactive frame rate.

A simpler approach for a dynamic illumination solution is screen-space directional occlusion (SSDO). It is an extension to the traditional ambient occlusion (AO), which is often used in feature films and video games. SSDO uses the information present at AO render passes and direction information of the light to approximate one-bounce indirect light transport. This solution performance does not depend on scene complexity and requires only minor additional computations to the widely used AO.

In this literature review I will aim to evaluate these approaches, taking into consideration the limitations of less powerful devices like mobile phones. Recent hardware improvements have already allowed such devices to pursue more intricate visuals and even virtual reality (VR). I will discuss the benefits and drawbacks of these implementation methods and attempt to assess the availability under computational power constraints.

3 Methods

GI is a computationally intensive effect because it requires visibility information between arbitrary positions in 3D space, which is a complex problem for rasterization based rendering. Realistic light simulation also requires to combine many light calculations for each rendered pixel to simulate light bouncing off to a large number of directions. The challenge of implementing GI in video games comes from approximating light behaviour well enough to get plausible results while also achieving high rendering frame rates for interactive gameplay.

3.1 Ray Tracing

Ray casting and ray tracing are rendering techniques that try to simulate reverse light photon paths from the cameras point of view to determine the final colour of pixels. The first algorithm for rendering using this idea was presented by Arthur Appel in 1968 (Appel, 1968). That approach has since been termed “ray casting” and was named by Scott Roth in 1982 (Roth, 1982). Ray casting refers to a rendering process of testing ray-surface intersections to compute pixel colours. A ray is cast from every pixel into the environment, fallowing the viewing direction. This creates the path most photons would have taken to reach that pixel and the surface intersection point indicates the transported colour. This was later expanded to generate new rays from intersections (Whitted, 1979) and named ray tracing.

Nowadays ray tracing algorithms closely model the physics of light. Visual effects like GI, shadows, reflection and refraction are natural computation results, whereas traditional rasterization techniques have to add special handling for such features. Ray tracing was never widely adopted as a rendering technique for video games because the hardware could not render high quality images at interactive frame rates. This type of rendering is often used to generate assets like light maps because of the accuracy of the image, but Imagination Technologies have been pursuing a fully ray traced rendering solution. In 2014 the company announced PowerVR Wizard GPU family, that focuses on ray tracing and promises to provide 300 million rays per second (Imagination Technologies, 2014). In 2016 at Consumer Electronics Show (CES) PowerVR GR6500 GPU was used to showcase dynamic ray traced scenes in real-time (Imagination Technologies, 2016).

Figure 1 Real-time ray tracing example (Imagination Technologies, 2016)

Traditional GPUs are extremely good at data parallel operations that obey a predictable structure, but are inefficient at handling the complexity of keeping track of coherency between rays. PowerVR Wizard family GPUs can use ray tracing for real-time applications because they have been designed with this problem in mind. This is proof that real-time ray tracing can be used for video games and in the future could become the dominant rendering approach because of many visual effects that it provides automatically. However, the requirement for dedicated hardware that does not have a large market penetration makes this approach undesirable at this point in time.

3.2 Deferred Radiance Transfer Volumes

In recent years the common implementation of GI in video games has been a mixture of offline and online techniques. The example discussed in this review is the method Ubisoft developers used for GI in “Far Cry 3”, which they called deferred radiance transfer volumes (DRTV) (Gilabert & Stefanov, 2012). Their implementation is focused around precomputed probes that are used while lighting the scene to approximate GI in real-time.

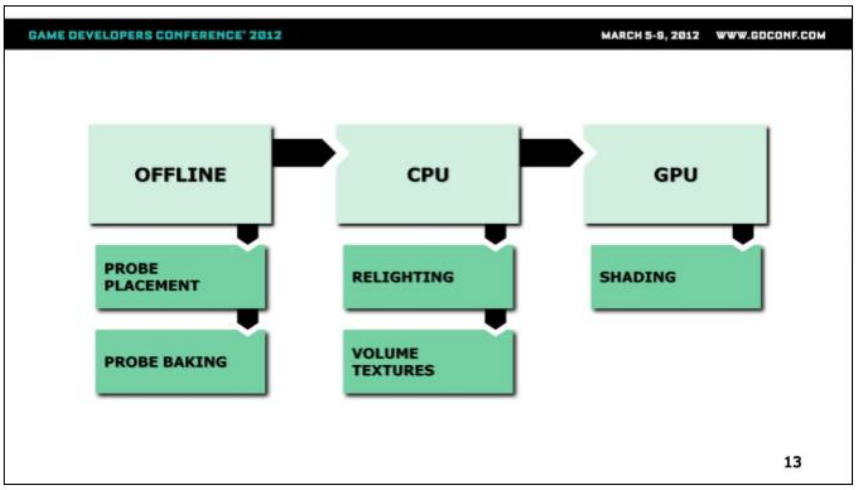

Figure 2 Deferred radiance transfer volumes system (Gilabert & Stefanov, 2012)

The entire system is illustrated in the figure above. In the offline step artists place probes in the world to create sampling points for approximating GI information. Radiance transfer is then computed for every probe. These probes are also recalculated in real-time during the game, whenever the lighting environment changes, and can support dynamic time of day cycles. New irradiance values are generated and inserted into a number of volume textures. During rendering the GPUs use these textures to shade everything in screen space.

The offline processes can be automated fairly well. Solution that was used to create "Far Cry 3" automatically placed probes in the environment, forming more density around areas that needed more sampling. The vegetation was combined into large bounding boxes to push the probes outside of forests with lots of light blocking objects. However, final adjustments still have to be made by hand and some problem areas like indoors need special treatment to prevent outdoor probes to contribute to illumination.

The baking process is done using a custom raytracing solution. The probes store low frequency precomputed radiance transfer (PRT) (Sloan, Kautz, & Snyder, 2002) and the sky visibility. A key difference with traditional PRT is that the algorithm does not compute geometry but empty space, because it is not known what the shading normal is going to be. The solution is to compute PRT for few directions (tree pointing upwards and one downwards). The probes are then used with light direction information to more

appropriately model light, coming from objects that are in shadows.

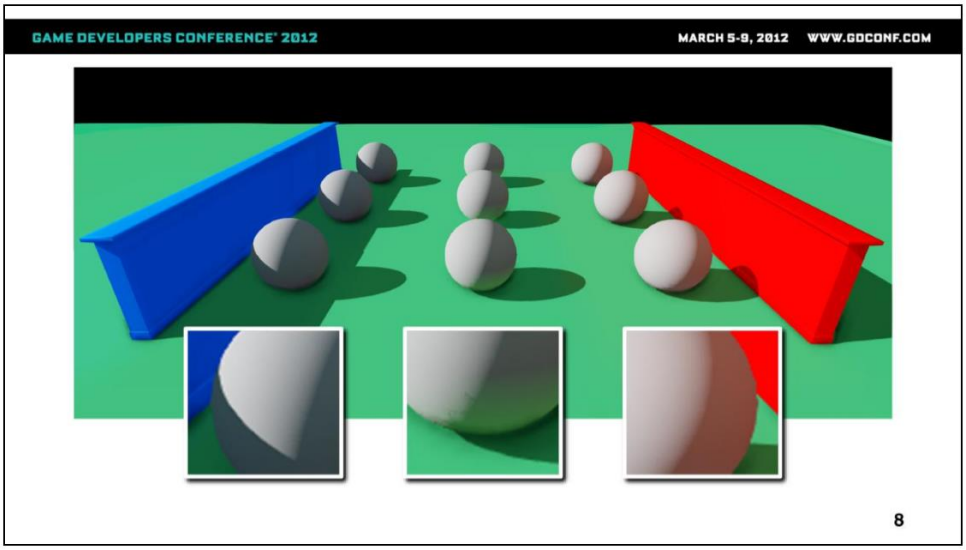

Figure 3 Deferred radiance transfer volumes GI example (Gilabert & Stefanov, 2012)

Once the probes are relight, the irradiance values are inserted into volume maps which fallow the camera, a similar approach to CryEngine's light propagation volumes (Kaplanyan, 2009). There are three volume textures (one for each colour channel), each texture being in RGBA8 format and 96x96x16 in size. Flushing the entire volume map is expensive, so only the necessary volume textures are updated when the camera position has changed.

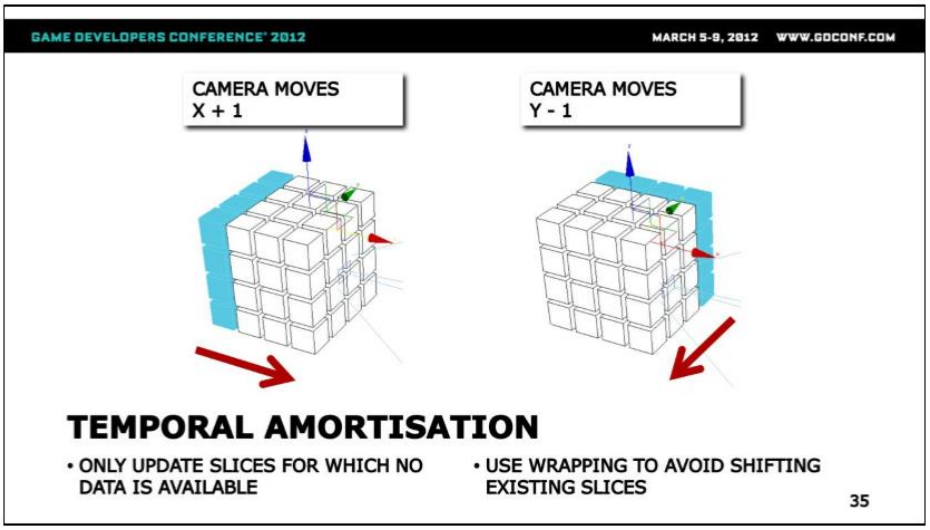

Figure 4 Deferred radiance transfer volumes data flushing on camera movement (Gilabert & Stefanov, 2012)

DRTV is a proven solution that offers real-time GI with reasonable resource requirements. The illumination approximation is plausible in most cases and can be manually adjusted to gain better results. There is flexibility to control the computation performance in many ways, but the vast scale of the implementation brings other disadvantages. This approach is hard to implement and has complex problems being addressed for both offline and online steps. There are fairly common special cases that need to be addressed manually and some performance enhancements rely on certain hardware accelerated features that might not be available on less powerful devices.

DRTV is overall a good solution for GI but requires a substantial development time to setup.

3.3 Voxel Cone Tracing

This approach is focused on pre-filtered hierarchical voxel representation of the scene geometry (Crassin, Neyret, Sainz, Green, & Eisemann, 2011). This data is stored on the GPU as a dynamic sparse voxel octree (Crassin, Neyret, Lefebvre, & Eisemann, 2009), generated from environment meshes. This provides a regular structure for light transfer, similar to cascaded light propagation volumes (Kaplanyan & Dachsbacher, 2010), using Gaussian-Lobes representation to encode a normal distribution function (NDF) and a distribution of light directions.

First the scene is rasterized from the point of view of each light source, injecting radiance (energy and direction) into the leaves of the sparse voxel octree during rendering. The radiance values are then filtered into the higher levels of the octree structure. After that, the scene can be rendered from the camera's point of view, where direct and indirect illumination is combined for each fragment. An approximate cone tracing is used to make the final gathering with around 5 large cones for diffuse and 1 cone for reflections.

The advantages of this method are:

- Almost geometry-independent execution costs.
- Calculation precision can be controlled based on distance from the camera.
- Plausible GI in complex environments at interactive frame rates.
- Smooth results with lower accuracy.
- Approximates two light bounces.

Both static and fully dynamic objects are stored in the same structure, but a time-stamp system is used to identify the difference and prevent unnecessary destruction of the scene data every frame update. The octree is built using the GPU rasterization pipeline, rasterizing the scene three times (along the X, Y and Z axis) at a resolution that represents the smallest octree subdivision. Once a leaf node is found, the necessary surface attributes are recorded.

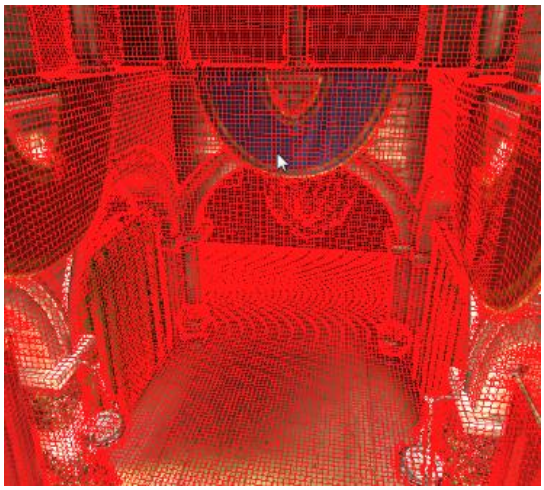

Figure 5 Sparse voxel octree (Crassin, Neyret, Sainz, Green, & Eisemann, 2011)

New nodes are created from a shared node buffer in pre-allocated video memory. A node subdivision is handled with per-node mutex that prevents multiple threads from subdividing a node at the same time. To avoid active waiting loops, a global thread list is used to record interrupted threads for later execution. The deferred list is rerun after the original rasterization pass and can generate more interrupted threads. Dynamic object updates function in the same way except they cannot overwrite existing static data and are placed at the end of the memory buffers for fast clearing.

True cone tracing is an expensive operation so this solution proposes an approximation method, inspired by filtering for anti-aliasing (Crassin, Neyret, Lefebvre, & Eisemann, 2009). The octree structure is used to approximate the result of all cone rays in parallel. As the algorithm steps along the cone axis, a lookup is performed on the hierarchical structure at the level of that cone radius and quadrilinearly interpolated samples are used for smooth variation.

Voxel cone tracing is an advanced technique that fits well for video games. There is no offline asset generation step and illumination accounts for dynamic objects while maintaining an interactive frame rate. The presented implementation refers to certain features that are available at OpenGL 3.2 and GLSL 1.30 specifications, but the exact minimal feature requirements are not clear. At the moment there is a lot of traction in the industry towards this solution. Tim Sweeney, founder of Epic Games, announced that Unreal Engine is integrating voxel cone tracing approach for their lighting system and describes it as “the biggest breakthrough in lighting since Unreal Engine 1 introduced real-time Direct Illumination in 1995” (Burnes, 2012) and CryEngine is using a similar solution (CryEngine, 2016).

3.4 Directional Occlusion

Ambient occlusion is a technique to calculate how receptive to ambient light a point in the scene should be. AO displays the darkening of cavities but is only a crude approximation of actual illumination. There are several implementation variants of this technique (Bavoil, Sainz, & Dimitrov, 2008) (McGuire, Osman, Bukowski, & Hennessy, 2011) and SSDO is one of them that proposes an extension to the traditional AO model and introduces one bounce indirect illumination (Ritschel, Grosch, & Seidel, 2009).

AO of a point in a scene indicates the obstruction of visibility inside a hemisphere with respect to the surface normal vector. A screen space variant of this is usually computed using the Monte Carlo approach of sampling some points in the hemisphere and approximating results based on the sample set. This methodology shares some ideas with ray tracing by trying to simulate scattered ray bounces, gathering to a point of interest. What SSDO proposes is to use the sampled results and directional information of those rays to not only darken the cavities but also estimate a likely ambient colour being received.

The average visibility value is computed from multiple sampling directions, uniformly distributed over the hemisphere. Occluders are approximated in screen space by creating sampling points of a random distance (within the hemisphere) along each of the sampling directions. These points are then tested to separate into groups of those above or below a surface. Points below the surface are treated as occluders and are considered for influencing the ambient colour. If the sample surface normal is pointing away from

the occlusion calculation position, that sample is also discarded from ambient colour contribution to avoid colour bleeding.

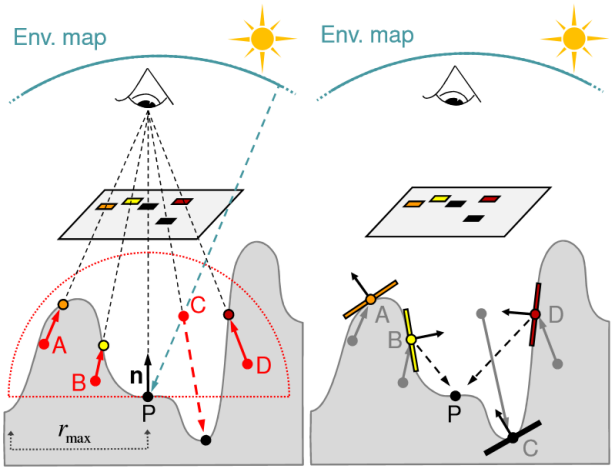

Figure 6 SSDO pixel sampling (Ritschel, Grosch, & Seidel, 2009)

SSDO is a simple solution for approximating GI for a small performance cost. There are downsides with screen space techniques related to missing geometry information but the results can be improved with depth peeling (Everitt, 2001) and multiple scene render calls. Although the GI effect is vastly simplified, the resulting calculations will appear plausible in most cases. This is a good solution for scenes that do not require highly accurate lighting effects or have strict limits for computer resources. The implementation simplicity, compared to other presented methods, and possible adaptation for less powerful devices makes SSDO also highly accessible.

4 Conclusion

After researching different approaches for global illumination, the problems that have to be addressed are much clearer. The way current GPUs address rendering is not well suited for simulating lighting and a lot of techniques opt for complex approximations to achieve similar results. New hardware innovations for running algorithms that are more representative of light behaviour are indicative that ray tracing could solve a lot of lighting problems in the future. However, At this point in time the rasterization based alternatives are much more suitable.

Voxel cone tracing is quickly being adopted by the industry and I see it as a superior technique to a lot of implementations with offline and online steps. There are no complex offline processing and no need for manual adjustment to address error cases. However, the requirement for rendering the scene several times and managing a large octree structure can require too many resources for some modern GPUs.

At the moment there is a large install base for devices that do not have resources for running a lot of these advanced techniques. Approaches like SSDO can be ideal not only for such devices but also for a quick visual enhancement under time constraints and 2D sprite games that imitate 3D environments. I feel that VCT and SSDO are the best solutions that address different market needs and are achievable on consumer grade hardware at this point in time.

5 References

Appel, A. (1968). Some Techniques for Shading Machine Renderings of Solids. (pp. 37-45). AFIPS.

Bavoil, L., Sainz, M., & Dimitrov, R. (2008). Image-Space Horizon-Based Ambient Occlusion. *SIGGRAPH*.

Burnes, A. (2012, June 7). *GeForce Articles*. Retrieved from http://www.geforce.com/whats-new/articles/stunning-videos-show-unreal-engine-4s-next-gen-gtx-680-powered-real-time-graphics

Crassin, C., Neyret, F., Lefebvre, S., & Eisemann, E. (2009). GigaVoxels: Ray-Guided Streaming for Efficient and Detailed Voxel Rendering. *I3D*, (pp. 15-22).

Crassin, C., Neyret, F., Sainz, M., Green, S., & Eisemann, E. (2011). *Interactive Indirect Illumination Using Voxel Cone Tracing*. NVIDIA.

Crassin, C., Neyret, F., Sainz, M., Green, S., & Eisemann, E. (2011, September). *NVIDIA Research*. Retrieved from https://research.nvidia.com/publication/interactive-indirect-illumination-using-voxel-cone-tracing

CryEngine. (2016, March 31). *CryEngine API*. Retrieved from http://docs.cryengine.com/display/SDKDOC2/Voxel-Based+Global+Illumination

Everitt, C. (2001). *Introduction Interactive Order-Independent Transparency*. NVIDIA.

Gilabert, M., & Stefanov, N. (2012). Deferred Radiance Transfer Volumes. *Game Developers Conference*.

Imagination Technologies. (2014, March 18). *Imagination Technologies blog*. Retrieved from http://blog.imgtec.com/powervr-developers/powervr-gr6500-ray-tracing

Imagination Technologies. (2016, January 7). *Imagination Technologies blog*. Retrieved from http://blog.imgtec.com/powervr-developers/real-time-ray-tracing-on-powervr-gr6500-ces-2016

Kaplanyan, A. (2009). Light Propagation Volumes in CryEngine 3. *SIGGRAPH*.

Kaplanyan, A., & Dachsbacher, C. (2010). Cascaded Light Propagation Volumes for Real-Time Indirect Illumination. *I3D*.

McGuire, M., Osman, B., Bukowski, M., & Hennessy, P. (2011). *The Alchemy Screen-Space Ambient Obscurance Algorithm*.

Ritschel, T., Grosch, T., & Seidel, H.-P. (2009). Approximating Dynamic Global Illumination in Image Space. *I3D*, (pp. 75-82).

Roth, S. (1982). Ray Casting as Method of Solid Modelling. *Computer Graphics and Image Processing*, (pp. 109-144).

Sloan, P., Kautz, J., & Snyder, J. (2002). Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *SIGGRAPH*, (pp. 527-536).

Whitted, T. (1979). An Improved Illumination Model for Shaded Display. *Computer Graphics*. ACM SIGGRAPH.