

Adding Atmospheric Scattering and Transparency to a Deferred Rendering Pipeline for Camera Based ADAS Tests

Sabrina Heppner¹, Marius Dransfeld² and Gitta Domik³

Abstract: This paper describes the implementation of a real-time global illumination model (atmospheric scattering and transparency) based on deferred rendering for the realization of virtual test drives used for the evaluation of camera based ADAS. Atmospheric scattering and transparent objects (e.g. windows) contribute to a realistic visualization of outdoor street scenes. However, both are not considered in the basic deferred rendering pipeline. To create an adequate adaption, the influence of the atmospheric scattering on transparent objects has to be considered. This work compares several techniques for the integration of transparency in the deferred rendering global illumination model. The realization of an A-Buffer implemented as Linked List provides the best results concerning the image quality. Weighted Blended Order-Independent Transparency also achieves adequate results. Inferred Lighting is inapplicable for the given use case.

Keywords: ADAS, advanced driver assistance system, global illumination, atmospheric scattering, transparency, deferred rendering, real time

1 Introduction

Camera based Advanced Driver Assistance Systems (ADAS) have gained increasing importance. Testing a new camera based ADAS during its development performing test drives with real cars is impractical, as they are cost- and time-intensive. Also the existence of a car prototype, which is produced late in the development process, is a precondition. An alternative for real-world test drives are tests in a three-dimensional virtual environment embedded in a HIL (Hardware-in-the-Loop) simulation. To ensure that the virtual tests create sufficient results as closely as possible to the results of real test drives, it is important that the rendering is as realistic as possible and real-time capable. A global illumination model for atmospheric scattering is one possibility to create realistic direct and indirect illumination. Global illumination describes a class of algorithms in computer graphics to create more realistic images by taking indirect illumination into account. By default, only local illumination is used and only direct illumination is computed. The interaction of light with particles in the air is not considered. Global illumination additionally takes this interaction into account and thus integrates indirect illumination. If light interacts with particles in the air, it is called atmospheric scattering. Because of the high recursion depth, the calculation of indirect illumination the direct way is very time-consuming.

¹ Paderborn University, CVB, Fürstenallee 11, 33102 Paderborn, sabrina.heppner@uni-paderborn.de

² Paderborn University, CVB, Fürstenallee 11, 33102 Paderborn, mariusdr@mail.uni-paderborn.de

³ Paderborn University, CVB, Fürstenallee 11, 33102 Paderborn, domik@uni-paderborn.de

To establish rendering at real-time frame rates, a deferred rendering pipeline⁴ can be used. Deferred rendering is an image based technique used in games and real-time applications gaining in popularity [La10] [Le11]. In contrast to forward rendering⁵, which is used by default, deferred rendering is a two-phase technique which decouples the rendering of geometry and lights into distinct passes: Geometry phase and illumination phase. During the geometry phase, data (e.g. surface normal, diffuse color, depth) needed for the illumination calculation is stored in the G-Buffer (geometry buffer). During the illumination phase, the data stored in the G-Buffer is read and combined with the information of the light to compute the final surface color. As this computation is done once per pixel, there is no overdraw possible, which is one major performance problem of forward rendering.

As with many other applications with three-dimensional graphics output, the test scenes used in software for the visualization of ADAS tests use transparent objects. Examples are windows in cars, buildings, noise barriers, bus stops or any other objects including parts made of glass. Another type of object uses textures with a dedicated alpha channel. It can be used to display objects like trees or fences. To keep the state of realism high, it is important that transparent objects like these are rendered correctly. Unfortunately, this is not feasible with a default deferred rendering pipeline described before. Therefore, an adequate adaption has to be found. This paper will first take a look at the physical background of atmospheric scattering and transparency. Afterwards, possibilities for the realization inside a rendering pipeline are considered and compared. In the end there will be a consistent concept to render outdoor scenes including atmospheric scattering and transparency based on deferred rendering. For readers who are interested in the physical background, section 2 gives a short overview. Those who like to get to know the concepts in depth are invited to read subsections 3.1 to 3.3 (atmospheric scattering) and subsection 4.1 (transparency). If the interest mainly applies to the results, skip to subsection 3.4 and 4.2.

2 Physical Background

We first consider the physical principles of atmospheric scattering and transparency.

2.1 Atmospheric Scattering

When light passes through the earth's atmosphere, it interacts with particles, namely molecules and aerosols. Three different types of interaction happen: Absorption, outscattering and inscattering. Absorption and outscattering cause a reduction of radiation. Absorption converts light into a different form of energy. During an outscattering event, some light rays are diverting into different directions than the initial traveling direction. In contrast to that, during an inscattering event additional light is redirected into the traveling direction which

⁴ A rendering pipeline is a logical model of computations needed for the process from 3D scene data to a 2D image in a raster-display system [LHL14].

⁵ Each object is rendered separately using material color, lights, etc. to determine the final color of the pixel associated with an object [Va14].

causes an increase of the radiation. Incoming light can either come directly from a light source or it is itself the result of scattering off other particles. The effect that light changes its direction several times due to scattering is called multiple scattering. In particular the scattering effect causes the indirect illumination in outdoor scenes.

The results of light interaction are visible effects in outdoor scenes. The most important effect is the indirect illumination. Other effects are the blue color of the sky, the coloring of the sky depending on the weather and location, the color gradient on the horizon, the coloring of dawn and twilight, the aerial perspective, the aureole and the “blue hour” [Ha96] [Ho07] [Ka91] [KRZ06] [LL01] [Ne14] [Mi12]. For the human perception these are familiar effects. Already Leonardo Da Vinci made use of them to give paintings a more realistic look [Ko12]. Not implementing these effects, e.g. by coloring the sky in an uniform blue, has the implication that a rendered scene receives an unrealistic appearance (best seen in our results shown in Figure 5). Consequently, by implementing these effects, a virtual scene becomes more realistic.

2.2 Transparency

An object can be described as transparent, translucent (partly transparent and partly opaque) or opaque (non-transparent) [Ho13]. These characteristics depend on how much light is absorbed and scattered while passing through the matter. When light emitted by a light source hits an object, some rays will be reflected and the rest will be refracted. While the refracted light is traveling through the object, it might be absorbed and scattered. If there is no scattering, the scene behind the object will be visible through the object. This effect is called transparency. In the case of water and glass very little light of the visible spectrum is absorbed. Therefore, they appear not just transparent, but very clear.

3 State of the Art: Rendering Global Illumination in Outdoor Scenes

In outdoor scenes atmospheric scattering has an important influence on illumination. Because of multiple scattering, the possible depth of recursion is much higher as if just surface reflection is considered. Hence, the standard rendering pipeline and traditional rendering methods, like ray tracing and radiosity, are not suitable to create a physical based rendering of outdoor illumination [Kl87] [Wa07]. In the following, different modern approaches are considered.

3.1 Previous Approaches

Much research has been done to create mathematical models for integrating atmospheric scattering into a virtual scene. The developed approaches can be divided into three groups: Simulation based methods, analytical models, and approximations. Simulation based methods make use of numerical solutions. Analytical models and approximations avoid numerical calculations with the help of analytical or rather approximated calculations. Figure 1

shows an overview of approaches including the chronological order and relations to each others.

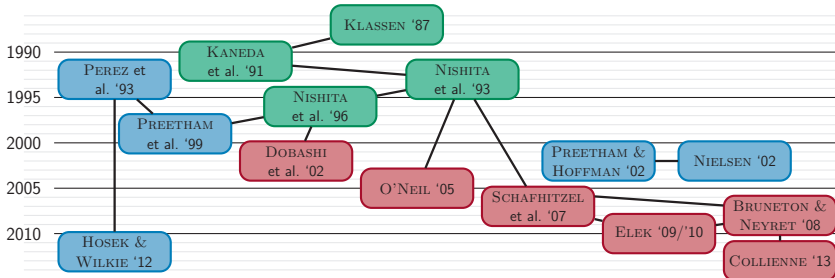


Fig. 1: Rendering of atmospheric scattering (green: Simulation based approaches, blue: Analytical approaches, red: Approximations)

Figure 1 shows that the most modern approaches make use of approximations. This way the calculation in real time can be realized. A comparison of the given approaches shows that the approach of BRUNETON & NEYRET [BN08] is the most innovative one fitting the given use case. It is an enhancement of the approach of SCHAFHITZEL et al. [SFE07] and the only approach implementing multiple scattering and shadow [BN08] [Co13] [Ko12]. Several other researchers use the approach of BRUNETON & NEYRET for their work.

3.2 Precalculated Lookup Tables

In the approach of BRUNETON & NEYRET, multiple scattering is implemented by an incremental calculation of the scattering values performed as precalculation on the GPU [EK10]. Scattering values are written into lookup tables stored as textures. For performing the precalculation, nine shaders, three 2D textures (one for intermediate results and two for final results), and four 3D textures (three for intermediate results and one for final results) are required. Figure 2 shows the complexity of the precalculation process and the relationship between different shaders and textures. In each step, exactly one scattering order is calculated using data of the previous scattering order [EK10].

During run time, the illumination of each pixel has to be calculated. It is composed by the directly incoming light, the reflected light, and the inscattered light. Equation 1 is used to perform this calculation:

$$L(x, \vec{v}, \vec{s}) = L_{direct}(x, \vec{v}, \vec{s}) + L_{reflected}(x, \vec{v}, \vec{s}) + L_{inscattered}(x, \vec{v}, \vec{s}) \quad (1)$$

x is the illuminated point, \vec{s} the direction to the sun and \vec{v} the view direction. How the single factors are composed is visualized in Figure 3. As marked in this figure, some components include integrals. Because of the huge calculation effort arising from these integrals, they are (completely or partially) calculated offline as a preprocess, stored in lookup tables called transmittance, inscatter, and radiance texture and just have to be read during run time. Some additional components are calculated online. For creating an illumination of high quality, the values have to be very precise. Offering precise values while performing

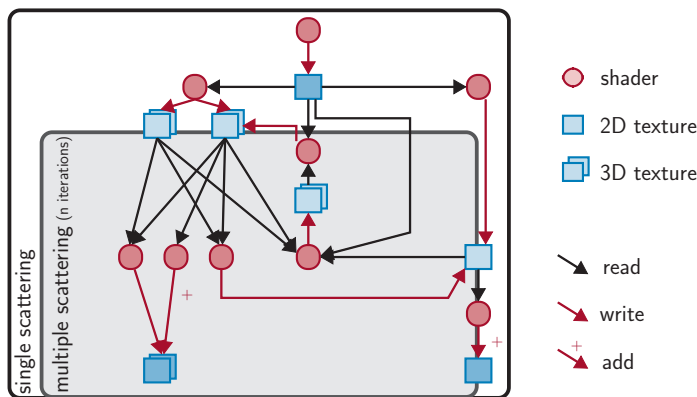


Fig. 2: Precalculation process

the calculation offline and storing the results in textures, would cause a high memory requirement. Therefore, these values are calculated at run time. There is no noticeable slowdown with this implementation. The real-time capability is still maintained.

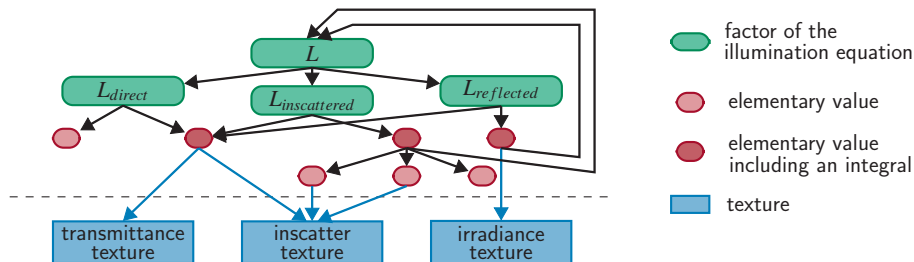


Fig. 3: Factors for the illumination calculation

Because of the consideration of the Mie theory and the Rayleigh theory in the precalculated values, all effects triggered by atmospheric scattering can be reproduced. Every time scattering parameters are changed, a rerun of the precalculation has to be performed [Ko12].

3.3 Deferred Rendering Pipeline

The approach of BRUNETON & NEYRET is adequate to be performed in a deferred rendering pipeline. Figure 4 shows this concept. In an offline phase, the precalculation is performed. As input for the precalculation parameters for the illumination are needed. The results of the calculation are stored in lookup tables to be used in the illumination phase. The execution of the online phase is divided into geometry and illumination phase. During the geometry phase, all data of the 3D scene needed for illumination calculations is stored in the G-Buffer. In the illumination phase the data from the G-Buffer is used in combination with the lookup textures to construct the illumination of the final scene. The illumination is

adapted immediately when the 3D scene is changed. Changes of the scattering parameter cause a rerun of the precalculation.

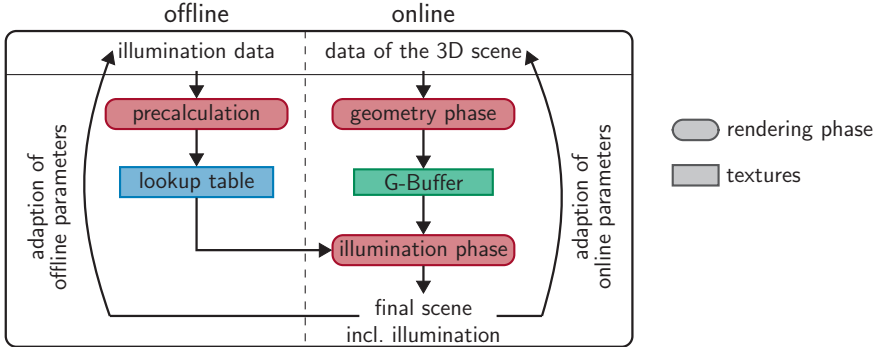


Fig. 4: Rendering of atmospheric scattering in a deferred rendering pipeline

3.4 Our Results of Deferred Rendering by BRUNETON & NEYRET

We implemented the previously described technique by BRUNETON & NEYRET for several traffic scenes and discussed our results (examples in Figure 5) with our domain experts.

3.4.1 Performance

Typically a camera has a frame rate of approximately 30 fps, but faster cameras exist. Therefore, developer of HIL simulators offer a frame rate of 60 fps. Thus, we define 30 fps as strong and 60 fps as weak real-time requirement. To evaluate the performance of the global illumination model for atmospheric scattering, two different types of hardware are used:

- Computer 1: Intel Core i7-3770, 3.40 GHz, nVidia Quadro 600, 8 GB RAM
- Computer 2: Intel Core i7-3770, 3.40 GHz, nVidia GeForce GTX 660 Ti, 8 GB RAM

The rendering time for 10 263 to 219 826 vertices on a standard PC (Computer 1) guarantees frame rates of more than 60 fps. Using a more powerful GPU (Computer 2) decreases rendering time to one fifth of the original rendering time. The rendering time excludes the one-time cost of the precalculation. The time of the precalculation strongly depends on the scattering order (recursion depth of multiple scattering) and on the hardware. For scattering order 1, less than one second⁶ is needed, for multiple scattering time increases⁷. In the case of the powerful GPU, the performance increases slightly by less than a half second. The weaker GPU records a stronger degradation of more than 10 seconds. A higher scattering

⁶ scattering order 1; Computer 1: $\sim 0.95s$, Computer 2: $\sim 0.19s$

⁷ scattering order 4; Computer 1: $\sim 13.30s$, Computer 2: $\sim 1.10s$

order than 4 is redundant because the adjustments are very small and not recognizable by the human eye. As expressed by our domain experts, all measured performance values for the precalculations are acceptable. Even the long measurements can be tolerated because of the infrequent execution of the precalculation. Most important is the run time itself.

3.4.2 Visual Quality

The visual quality is optimized while using a global illumination model for atmospheric scattering. The improvement can be comprehended best by the effects of the atmospheric scattering. Directly implemented by [BN08] are the following effects: Indirect illumination, blue color of the sky, color gradient on the horizon, coloring of dawn and twilight, aureole, aerial perspective. Furthermore, the effect “blue hour” is integrated by adding a blue coloring when the sun’s position is under a certain threshold value. The more realistic look of the scene is assessed as useful and pleasing and will be implemented in the product of our domain experts. Figure 5 contrasts a scene rendered without and including atmospheric scattering. It is significant that the integration of atmospheric scattering creates a much more realistic look.

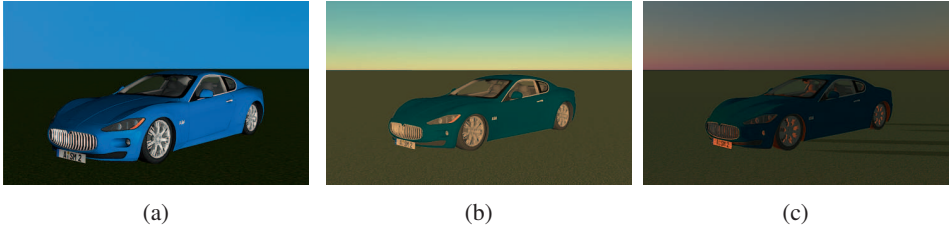


Fig. 5: Comparison: Rendering without (a) and including ((b) morning sun, (c) “blue hour”) atmospheric scattering

4 State of the Art: Rendering Transparency

In the typical situation of rendering transparency an opaque object is partially obscured by a transparent object. The light travels through the transparent object in a straight line (special cases like scattering inside the model are ignored). While rendering such a scene, the shading for both, opaque and transparent objects, are computed separately. The two colors are combined by a process called blending to produce a single color which, if done correctly, gives the visual impression that the transparent object is in front of the opaque object. The blending process can be described formally with the help of Equation 2 where A and B represent the color of a pixel associated with object A and object B and the alpha value of A (A_α) describes the opacity of A .

$$A \text{ OVER } B = A + B \cdot (1 - A_\alpha) \quad (2)$$

This underlying equation defining this process is commutative, meaning the order of the rendered objects is important. As an additional simplification, transparent objects which are

not captured by the camera but placed in front of the light source and thus influencing the color of the light are not considered.

To integrate transparency into a deferred rendering pipeline, the use of different approaches is conceivable. The techniques we will focus on are Forward Transparency, Weighted Blended Ordered-Independent Transparency, Inferred Lighting, and an A-Buffer implemented as Linked List on the GPU.

While integrating transparency into the atmospheric scattering concept, the precomputation remains unchanged. It is only important to define how transparent objects are rendered with the effect of atmospheric scattering. The illumination equation (see Equation 1) and the equation of standard alpha (see Equation 2) blending can be combined trivially. The colors of the pixels associated with object *A* and *B* are calculated by the illumination equation and combined with the help of alpha blending. Thus, the atmospheric scattering model can be extended intuitively to include transparent objects. There are no modifications needed to add neither the atmospheric scattering nor the blending operator.

4.1 Techniques

The straight forward technique is called Forward Transparency. Here the transparent objects are rendered in a separate forward rendering pass called transparent pass after the deferred rendering. During this transparent pass, the transparent objects are rendered as in a forward rendering pipeline. Each object is rendered separately using material color, lights, etc. to determine the final color of the pixel associated with an object [Va14]. Thus, the transparent pass has the same disadvantages as normal forward rendering for opaque objects, e.g. performance decrease with an increasing number of lights. Furthermore, Forward Transparency does not solve the sorting issues arising from ensuring correct blending of multiple transparent objects placed successive.

Forward rendering can be improved by a technique called Weighted Blended Order-Independent Transparency. This technique removes the need to sort transparent objects prior to rendering. The idea of blended Order-Independent Transparency in general is to redefine the blending order to be commutative which allows performing the blending process in arbitrary order. This technique was shaped by the following approaches: [Me07] [BM08] [MB13] [MB14]. In total five rendering passes are needed: Geometry pass, light pass, clear pass, transparent pass and composition pass.

Another approach, Inferred Lighting, was invented by KIRCHER & LAWRENCE [KL09]. It is a variation of deferred rendering to render transparent objects directly into the G-Buffer and separate them from opaque objects in a later pass. Thus, the main advantage of Inferred Lighting over deferred rendering is that transparent and opaque pixels can be lit in the same way. The rendering is separated into three passes: Geometry pass, light pass, and material pass. The lighting calculation can be performed at a lower resolution than the final output. This can improve the rendering performance significantly because the illumination equation has to be computed for fewer pixels and the smaller buffer requires less memory bandwidth.

The A-Buffer (anti-aliased, area-average, accumulation buffer) was first introduced by CARPENTER [Ca84] as a general hidden surface mechanism and an improvement over the Z-Buffer. Among other advantages, it allows rendering of order-independent transparency. During rendering, for each pixel either a single fragment or a pointer to a list of fragments is saved. Hardware accelerated implementation of an A-Buffer was proposed by YANG et al. [Ya10] and LEFEBVRE et al. [LHL14]. They create and maintain linked lists of fragments for each pixel entirely on the GPU. Previous A-Buffer implementations are designed without considering a deferred rendering pipeline. SCHOLLMAYER et al. [SBF15] describe a full deferred rendering pipeline which integrates an A-Buffer implementation using linked lists. A-Buffer implemented as Linked List approach consists of three rendering passes: Geometry pass, light-culling pass and shade-composing pass.

There exist various further techniques which are either too slow to be used in real-time rendering or exhibit other properties which makes them unsuitable for the given domain [Mal1].

4.2 Our Results of Three Transparency Techniques

To compare results of transparency in a deferred rendering pipeline including atmospheric scattering, we implemented three techniques⁸: Weighted Blended Order-Independent Transparency (WBOIT), Inferred Lights (IL), and A-Buffer/Linked Lists (LL). As basis for evaluation we chose performance and visual quality.

4.2.1 Two Different ADAS Scenes

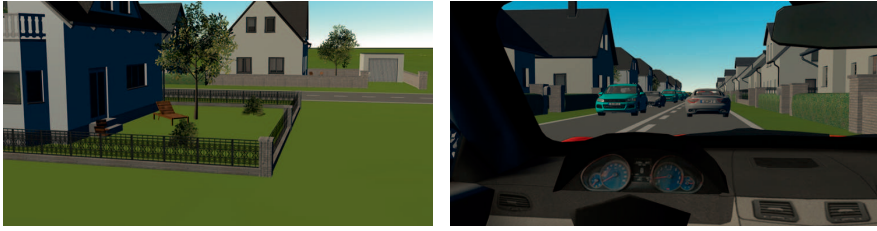
For evaluation, two suitable ADAS scenes of different complexity are created. The setting of the first scene is a small village with several houses, cars, streets, and a forest with 400 randomly placed and oriented trees. There are several surfaces using the alpha channel, e.g. windows, foliage, and fences (see Figure 6a). The second scene is more complex. It is used to evaluate the implementations for a traffic scene. It contains a road with numerous cars surrounded by houses on both sides (see Figure 6b). All tests are conducted with a resolution of 1920 x 1080 pixels and all statements in the following refer to this resolution.

4.2.2 Performance

For the performance test, two different types of hardware are used:

- Computer 1: Intel Core i7 860, nVidia Quadro K2000, 4GM RAM (slow, less memory)
- Computer 2: Intel Core i5 4750, nVidia GTX 660, 16GB RAM (fast, more memory)

⁸ Forward rendering is not considered because the performance is much lower than the performance of the other techniques.



(a) Test scene 1: Small village with forest

(b) Test scene 2: Traffic scene

Fig. 6: Test scenes

The performance is recorded from 13 different views of the first scene and 5 different views of the second scene. These views are used to test a variation of different conditions. In the case of scene 1, depending on the view, 25 to 2130 objects, 4436 to 92 833 triangles, and 10 263 to 219 826 vertices are given. In the second scene there are significantly more triangles and vertices to be rendered. On average there are ~ 600 objects, $\sim 350\,000$ triangles, and $\sim 420\,000$ vertices visible. For example, the view in Figure 6b, a close-up view of the windshield from inside of a car, includes a large amount of transparent fragments (~ 1.5 million).

Because precomputation remains unchanged, its performance is unaffected. During run time, the light distribution function has to be evaluated at every transparent fragment. The most expensive operation is the adding of inscattered light as it involves many lookups in a 3D texture.

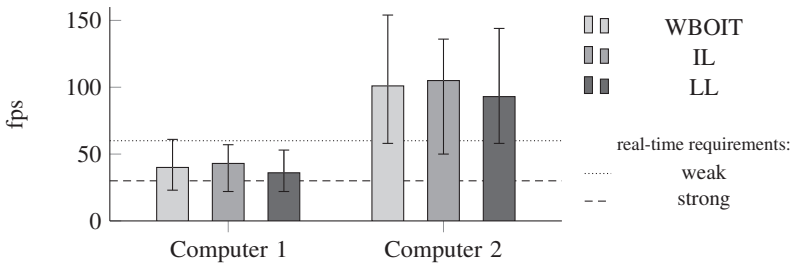


Fig. 7: Average and minimal/maximal performance

Figure 7 shows the results of the performance test. Each bar visualizes the average performance of the implementation of one technique on one computer. Furthermore, minimal and maximal performance occurring during the test are marked. The dashed line and the dotted line visualize the strong (30 fps) and the weak (60 fps) real-time requirements⁹. It is significant that all three techniques offer similar results. The weaker computer (Computer 1) reaches the weak real-time requirement. In the case of the more powerful computer (Computer 2) even the worst measured performance values exceed the number of 60 fps and thus the hard real-time requirement is fulfilled.

⁹ see section 3.4.1

4.2.3 Visual Quality

On visual quality, we again rely on our domain experts for a qualitative assessment. The domain experts observed scenes including transparent objects rendered with different methods. They identified features that differ for the given methods. Table 1 shows screenshots used for comparison. In addition to two ADAS scenes we include a teapot scene demonstrating rendering mistakes arising from a high number of completely transparent overlapping objects not apparent in the ADAS scenes.

WBOIT produces images with varying quality. For surfaces with low alpha values, including objects rendered with alpha mask textures like leaves (leaves (a)) or fences, the results are of desired quality indistinguishable from Linked Lists (leaves (c)). In the case of transparent surfaces with high alpha value or very saturated colors, rendering mistakes occur. Such an effect is recognizable in the very bright specular highlights on the windshield (windshield (a)). Furthermore, the screenshot of the teapots (teapots (a)) reveals that there is a problem concerning the perceived ordering. The colors of teapots in the background dominate too much.

Inferred Lighting shows the worst overall image quality of all three tested techniques. This low quality is caused by the low resolution rendering and subsequent upscaling of transparent surfaces. This is most notably seen at the border of transparent surfaces like the car window (windshield (b)). Lighting fails completely for cases where several surfaces with the same stipple pattern (see [KL09]) are rendered on top of each other (leaves (b)). This also applies for transparent objects overlapping themselves as seen in the case of the teapots (teapots (b)).





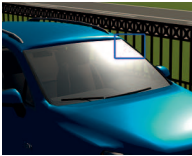



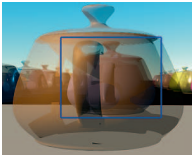



Linked Lists shows desirable image quality while rendering transparent surfaces due to the per-pixel sorting of transparent fragments (leaves, windshield, teapots (c)). The results should be identical to an offline rendering with an A-Buffer implementation. This only applies as long as there is enough video memory available to store all rendered fragments. If memory overflows, fragments have to be discarded, which leads to visual artifacts. Using older GPUs¹⁰, as the ones used for our evaluation, a flickering effect, not visible on still images, occurs. These older GPUs do not support atomic operations needed for the ordering of fragments with similar depth. Thus, the ordering can vary and this leads to a flickering effect.

Due to the multiple rendering mistakes, WBOIT and IL are rejected by our domain experts. In case of suitable graphics cards, LL is acceptable.

5 Summery & Conclusion

Our solution employs a deferred rendering pipeline which allows atmospheric scattering to be added without noticeable slowdown to the real-time requirements while at the same time

¹⁰ E.g. the `atomicMax` operation for 64-bit values is supported by nVidia GPUs starting with the Maxwell generation, e.g. nVidia GeForce GTX 970.

full view on scene	(a) WBOIT	(b) Inferred Lighting	(c) Linked List
leaves 	 ✓	 ✗	 ✓✓
windshield 	 ✗	 ✗	 ✓✓
teapots 	 ✗	 ✗	 ✓✓

Tab. 1: Comparison of visual quality (✓✓: good, ✓: acceptable, ✗: bad)

providing a pleasing natural effect. Regarding the implementation of transparent objects (e.g. windshield) and realistic rendering of fences and foliage (textures with dedicated alpha channel), we have found Weighted Blended Order-Independent, Inferred Lighting and Linked List to be techniques all satisfying the real-time requirements, while only Linked List reaches the visual realism requirements of our domain experts. However, best visual quality is attained at the cost of sufficient video memory and a modern GPU supporting atomic operations.

Acknowledgment

The models placed in the test scenes are provided and copyrighted by dSPACE GmbH.

References

- [BM08] Bavoil, Louis; Myers, Kevin: Order Independent Transparency with Dual Depth Peeling. NVIDIA OpenGL SDK, pp. 1–12, 2008.
- [BN08] Bruneton, Eric; Neyret, Fabrice: Precomputed Atmospheric Scattering. Computer Graphics Forum, 27(4):1079–1086, 2008.

- [Ca84] Carpenter, Loren: The A-buffer, an antialiased hidden surface method. *ACM Siggraph Computer Graphics*, 18(3):103–108, 1984.
- [Co13] Collienne, Peter; Wolff, Robin; Gerdt, Andreas; Kuhlen, Torsten: Physically Based Rendering of the Martian Atmosphere. In (Latoschik, Marc Erich, ed.): *Virtuelle und Erweiterte Realität – 10. Workshop der GI-Fachgruppe VR/AR*. Shaker, Aachen, pp. 97–108, 2013.
- [EK10] Elek, Oskar; Kmoch, Petr: Real-Time Spectral Scattering in Large-Scale Natural Participating Media. In (Hauser, Helwig; Klein, Reinhard, eds): *Proceedings of the 26th Spring Conference on Computer Graphics*. ACM, New York, pp. 77–84, 2010.
- [Ha96] Haltrin, Vladimir I.: Haltrin. A Real-Time Algorithm for Atmospheric Corrections of Airborne Remote Optical Measurements above the Ocean. In: *Proceedings of the Second International Airborne Remote Sensing Conference and Exhibition*. volume 3, pp. 63–72, 1996.
- [Ho07] Hoeppe, Götz: *Why the Sky is Blue – Discovering the Color of Life*. Princeton University Press, Princeton, 2007.
- [Ho13] Hoffman, Naty: *Background: Physics and Math of Shading*. SIGGRAPH, 2013.
- [Ka91] Kaneda, Kazufumi; Okamoto, Takashi; Nakamae, Eihachiro; Nishita, Tomoyuki: Photorealistic Image Synthesis for Outdoor Scenery under Various Atmospheric Conditions. *The Visual Computer*, 7(5-6):247–258, 1991.
- [Kl87] Klassen, R. Victor: Modeling the Effect of the Atmosphere on Light. *ACM Transactions on Graphics*, 6(3):215–237, 1987.
- [KL09] Kircher, Scott; Lawrance, Alan: Inferred Lighting: Fast Dynamic Lighting and Shadows for Opaque and Translucent Objects. In: *Proceedings of the 2009 ACM SIGGRAPH Symposium on Video Games*. Sandbox '09, ACM, New York, NY, USA, pp. 39–45, 2009.
- [Ko12] Kol, Timothy R.: *Analytical Sky Simulation – An Implementation and Analysis of Daytime Skylight Models*. Technical report, University, 2012.
- [KRZ06] Kment, Thomas; Rauter, Michael; Zotti, Georg: *Modelling of Daylight for Computer Graphics*. Technical report, Institut für Computergraphik und Algorithmen, Wien, 2006.
- [La10] Lake, A.: *Game Programming Gems 8*. IT Pro. Course Technology, 2010.
- [Le11] Lengyel, E.: *Game Engine Gems 2*. Taylor & Francis, 2011.
- [LHL14] Lefebvre, Sylvain; Hornus, Samuel; Lasram, Anass: Per-Pixel Lists for Single Pass A-Buffer. *GPU Pro 5: Advanced Rendering Techniques*, 2014.
- [LL01] Lynch, David K.; Livingston, William Charles: *Color and Light in Nature*. Cambridge University Press, Cambridge, 2001.
- [Ma11] Maule, Marilena; Comba, Joo L.D.; Torchelsen, Rafael P.; Bastos, Rui: A Survey of Raster-Based Transparency Techniques. *Computers & Graphics*, 35(6), 2011.
- [MB13] McGuire, Morgan; Bavoil, Louis: Weighted Blended Order-Independent Transparency. *Journal of Computer Graphics Techniques (JCGT)*, 2(2):122–141, 2013.
- [MB14] McGuire, Morgan; Bavoil, Louis: Weighted, Blended Order Independent Transparency. *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2014.
- [Me07] Meshkin, Houman: Sort-independent alpha blending. *GDC Talk*, 2007.

- [Mi12] Minnaert, Marcel: *Light and Color in the Outdoors*. Springer, New York, 2012.
- [Ne14] Nentwig, Mirko: *Untersuchungen zur Anwendung von computergenerierten Kamerabildern für die Entwicklung und den Test von Fahrerassistenzsystemen*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2014.
- [SBF15] Schollmeyer, Andre; Babanin, Andrey; Froehlich, Bernd: Order-Independent Transparency for Programmable Deferred Shading Pipelines. In: *Computer Graphics Forum*. volume 34. Wiley Online Library, pp. 67–76, 2015.
- [SFE07] Schaffhitzel, Tobias; Falk, Martin; Ertl, Thomas: Real-Time Rendering of Planets with Atmospheres. In (Rossignac, Jarek; Skala, Václav, eds): *Journal of WSCG*. University of West Bohemia, Plzen, pp. 91–98, 2007.
- [Va14] Varcholik, P.: *Real-Time 3D Rendering with DirectX and HLSL: A Practical Guide to Graphics Programming*. Game Design. Pearson Education, 2014.
- [Wa07] Wang, Changbo: Real-Time Rendering of Daylight Sky Scene for Virtual Environment. In (Ma, Lizhuang; Rauterberg, Matthias; Nakatsu, Ryohei, eds): *Entertainment Computing – ICEC 2007*, volume 4740, pp. 294–303. Springer, Berlin & Heidelberg, 2007.
- [Ya10] Yang, Jason C.; Hensley, Justin; Grün, Holger; Thibieroz, Nicolas: Real-time Concurrent Linked List Construction on the GPU. In: *Proceedings of the 21st Eurographics Conference on Rendering*. Eurographics Association, Aire-la-Ville, Switzerland, pp. 1297–1304, 2010.