mallumo: Framework for Real-Time Global Illumination

Matúš Talčík*
Áron Samuel Kovács†
Supervised by: Jan Čejka‡

Faculty of Informatics
Masaryk University
Czech Republic

Abstract

Fast computation of global illumination is one of the most researched topics in computer graphics. This paper presents a real-time computer graphics library called mallumo, which achieves global illumination with voxel-based cone tracing, and its components, namely support for high dynamic range data stored in voxels, anisotropic filtering of voxels, support for different cone distributions, rendering of atmospheric effects, and a method for real-time rendering of volumetric clouds. These components are compared and benchmarked in various configurations.

Keywords: voxel-based, global illumination, atmosphere, rendering, real-time

1 Introduction

Achieving photorealism is crucial in many industries such as film effects, video games or architecture visualization. The major component in making images photo-realistic is the calculation of a set of effects known as global illumination (GI), such as specular and diffuse reflections, refractions, and soft shadows. The computation of GI is a widely known and studied problem and the most widely used method to achieve the effects mentioned above is by simulating light transport.

Correctly simulating light transport is a computationally expensive operation, due to the recursive nature of rendering equation, so many alternative methods were developed to approximate the desired result in various ways. Approximations can be divided into two groups: offline, which tries to stay close to physical correctness at the expense of computational time, and real-time, which foregoes correctness to provide visually convincing results usable for interactive use cases. The former group has seen steady progress in the last decades, mostly using Monte Carlo methods, and is widely used in the movie and architectural visualization industries.

Most real-time solutions that emerged in the last decade have seen very few applications, mainly because they still require a relatively large portion of the graphics processing unit's (GPU) processing power while existing applications already utilize most of GPU's power, not leaving enough for GI computation. Each real-time GI algorithm has a different trade-off regarding the quality or required precomputation.

In this paper, we present our implementation of a cone tracing method operating on voxel grid based on work of Green and Crassin [6], which requires features found only in recent GPU generations, for example, atomics, arbitrary read and writes to textures, geometry shaders, and compute shaders. These features allow the implementation to run in real-time on current hardware, although they usually eliminate these methods as candidates for inclusion in game engines, where support for web browsers, mobile systems, or older computers is required.

In order to create realistic images, mallumo also contains algorithms for rendering atmosphere and clouds. To provide their realistic simulation in real-time interactive applications, a great effort is spent to make them look convincing, but in many cases, this effort falls short, and the results may not be of desired quality. Fortunately, the latest advances in hardware made it possible to render volumetric clouds in real-time. In mallumo, we procedurally generate volumetric clouds, and implement atmospheric effects such as the colour of the sky dependent on the position of the sun, and transmittance of light in objects inside the atmosphere and beyond the atmospheric boundary.

2 Related Work

There are many alternative approaches to computing global illumination in real-time. Virtual points lights algorithm introduced by Keller [12] computes many intersections of light rays which then all light the scene. This algorithm requires a costly precomputation phase and is therefore suited only for static geometry. Reflective shadow maps introduced by Dachsbacher [7] reuse shadow maps as indirect illumination sources. This method needs to importance sample the shadow maps to obtain real-time

*matus.talcik@mail.muni.cz
†aron.kovacs@mail.muni.cz
‡xcejka2@fi.muni.cz

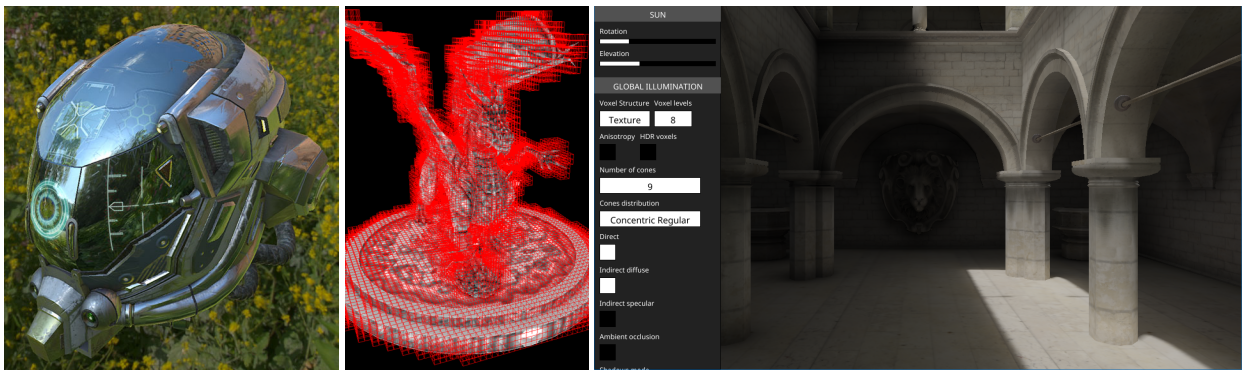

Figure 1: Examples of mallumo supported features: (Left-to-right) object with physically based materials and image based lighting, sparse octagonal tree, graphical user interface.

speed which results in severe artefacts and not enough samples for specular reflections. Forward light cuts introduced by Laurent et al. [14] improves upon virtual point lights, but supports only diffuse indirect illumination. Sparse radiance probes by Silvennoinen [22] is one of the most recent approaches with results very close to ground truth. However, it needs a very long precomputation time and therefore supports only static geometry. There is also a closed-source implementation of voxel-based global illumination by NVIDIA [17].

Various models were created to simulate effects in the atmosphere. Nishita et al. [15] precomputes values for single scattering, but the second scattering is computed dynamically for each sampling point along the view ray, and thus determining single pixel value takes linear time in respect to the number of sampling points. Preetham et al. [20] precomputes the previous model for many view directions, sun directions, and turbidity values, so it takes constant time for each view ray but is limited to ground view and the sun being above the horizon. Other alternatives include Nishita et al. [16] or Haber et al. [10]

For clouds, prerendered or photographed skybox and skydome[9] approaches are often chosen for their simplicity, however, they only provide static clouds. If dynamic clouds are desired, alternatives to voxel-based clouds are e.g. clouds modelled from implicit volumes[8] which are distorted by a perturbation function or are composed of either particles or polygons[18].

Our research is based on bachelor theses [13, 23] which contain detailed description of algorithms presented in this paper.

3 mallumo

mallumo is a real-time rendering library written in the Rust programming language. It provides a safe abstraction over OpenGL to prevent most errors caused by incorrect usage of the API that are caught at compile time, which is achieved by repurposing the Rust type system and ownership model for OpenGL constructs. The errors that can not be caught in compile time, e.g. running out of GPU memory, are detected during runtime and reported to the user of this rendering library, in a way that makes it possible to react to them. The version of OpenGL used is 4.5 with support for only latest functions to reduce the driver overhead.

The library uses physically-based materials to achieve a photorealistic look. They are based on the microfacet model with input values and terms aligned with those in Unreal Engine 4 [11] and enhanced diffuse model by Burley [4]. Furthermore, mallumo supports high dynamic range image based lighting to produce realistic specular term for materials. A combination of physically based materials and image based lighting can be seen in the left part of Figure 1.

The support for industry standard physically based materials allows loading any scene with all the materials and textures using glTF 3D format. Deferred rendering is used to reduce the number of calculations and serves as an input to the calculation of voxel-based global illumination. Shadows maps are implemented and are used for calculation of first bounce of light before the second bounce is applied. They can optionally generate texture of world positions, not only depth information which is useful for injecting the first bounce into the voxel structure. In order to save memory consumption on voxel structure needed for cone tracing a sparse octagonal tree is implemented, see an example of such tree in the middle part of Figure 1.

Furthermore, mallumo supports postprocessing effects including blurring, gamma correction, and several tone mapping functions to produce the final image. For producing easily controllable examples a set of functions producing graphical user interface is available, see an interface for controlling voxel-based global illumination in the right part of Figure 1.

The source code and further information about the library can be found at `https://gitlab.com/mallumo/mallumo`.

4 Extensions To Voxel-Based Global Illumination

For computing global illumination, a scene geometry is turned into voxel representation and then sampled using cones to obtain indirect illumination. To obtain a voxel representation of a scene in an arbitrary data structure, mallumo uses a method of voxelization described in the work of Green and Crassin [6].

Depending on what data structure is used, a different compute path is taken at the end of voxelization. Fragments of volume textures are written directly, but in case of sparse voxel octrees, fragments are added at the end of a buffer, called the voxel fragment list, to be used for building the node pool, and then written to a volume texture, called the brick pool, which stores sparse data densely. The node pool contains pointers to the brick pool.

Voxelised data of a scene are sampled by cone tracing using a method proposed by Amanatides [1]. In this work, rays are represented as cones defined by their origin, direction, and angle of spread. The spread angle is the angle between the centre line of the cone and the cone boundary as measured at the apex of the cone. Deferred shading is used to determine for which surface points a computation of the indirect illumination is needed. At each such point, a series of cones is sent through the voxel structure to calculate the inirect illumination effects.

Several cones are sent to calculate the diffuse component. One tight cone is sent in the reflected direction with respect to the viewpoint to capture the specular component. The aperture of this specular cone is based on roughness value in the PBR model used. Ambient occlusion can be obtained in the same way as the diffuse component, but sampling only the opacity value. When calculated for indoor environments, a weight function decaying with the distance is used for obtaining more practical values. Additionally, a crude approximation of an area light is obtained by storing and sampling emissive value separately in a voxel data structure. Cones are sampling values from a voxel data structure by stepping along the ray and calculating the desired the mip level based on the diameter. Example of voxel-based cone tracing can be seen in Figure 2.

Figure 2: Top: a scene lit only using direct light. Bottom: a scene lit using voxel-based global illumination.

4.1 High Dynamic Range Storage

A high dynamic range of colours may be desirable to voxelise and store in a voxel data structure. However, voxelization of data must be done atomically and most GPUs have their atomic operations limited to only 32 bits. This is sufficient if each RGB colour channel is represented with 8 bits and the fourth channel is dedicated to counting number of fragments belonging to a voxel. However, the amount of data that can be stored in 24 bits is not enough to capture both colour and intensity of very bright light sources. This translates to wrong calculations with the intensity of the sun being too high or emission of an area light contributing to the final image. An example of such calculation can be seen in Figure 3.

A solution we implemented is to create a voxel data structure, called the mutex structure, of the same size as the voxel data structure containing data so that each voxel has an associated mutex in the lock structure. The mutex structure contains boolean values, set to false at the beginning of voxelization process, expressing the state of whether the associated voxel is currently being modified or not. Similarly, a count voxel structure is also introduced to separate the count and allow the maximum of 128 bits to be used for HDR values.

Each thread writing a voxel value then uses atomic exchange function to obtain the lock on a voxel after which it can write its value. There is additionally a limit on the number of unsuccessful tries of taking the lock. This may result in loss of some data, but prevents stalling the GPU.

An alternative to our approach may be the use of Ward's RGBE 32 bit format [24], which can be seen as a compromise between the amount of memory used and the precision of colour and intensity stored.

4.2 Anisotropic Filtering

Mipmapping of a voxel data storage is required for effective cone tracing. There are two options on how to mipmap voxels, both of which we implemented in mallumo. First one is isotropic, where eight voxels are equally accounted for in the calculation of one final voxel in the

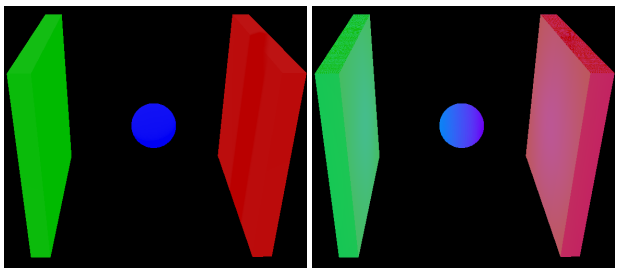

Figure 3: Example of deficiency of low dynamic range values in case of area lights. Walls acts as diffuse objects with RGB values of (0, 1, 0) and (1, 0, 0) respectively. The sphere acts as an area light with RGB value of (0, 0, 10). On the left, a low dynamic range cone tracing. On the right, a high dynamic range cone tracing.

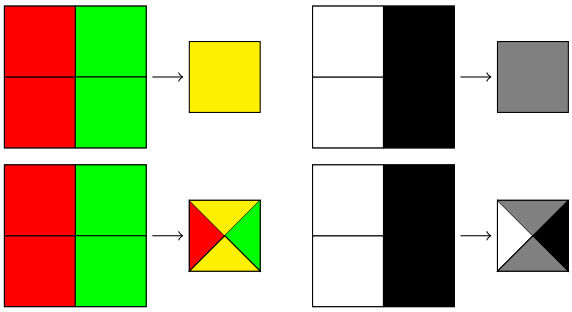

Figure 4: Difference between isotropic and anisotropic mipmapping. Top: isotropic filtering, notice how different coloured voxels are combined into one completely different colour and fully opaque and fully transparent voxels become partially transparent. Bottom: anisotropic filtering: notice how colours and transparency are preserved, averaging is performed only from some sides.

higher mip level. However, such method creates artefacts, for example combining two different colours of a double-sided wall into one that is incorrect for both sides, or combining transparency making transparent voxels visible and opaque voxels transparent, see top part of Figure 4.

To solve this problem, anisotropic mipmapping can be used. For every voxel, each side is mipmapped separately, see bottom part of Figure 4. To use cone tracing on the anisotropic voxel data structure, it has to sample a weighted average of three visible faces.

A shortcoming of this method is its higher memory usage. This increase is roughly 1.85 times more when compared to the isotropic case, as can be derived by summing the size of volume texture and all its mipmaps.

4.3 Cone Distributions

The original work of Crassin et al. [5] did not address a distribution of cones over the hemisphere, only rough estimate of their number was given, to be between five and nine. Due to this problem we implemented cone distributions based on the work of Appelbaum and Weiss [2].

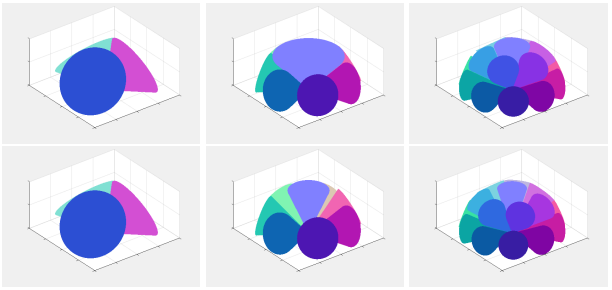

Figure 5: From left to right, examples of concentric non-overlapping cone distributions for n of 3, 8, and 18. The top part contains non-uniform distributions and the bottom part contains uniform distributions. In case of three cones, the distribution is exactly the same, but the difference increases with more cones, and decreases again when using more than a dozen of cones.

The problem of distributing n number of non-overlapping cones over a hemisphere is a problem of packing non-overlapping circles on hemisphere surface. The solution consists of packing the circles in concentric circles as shown in Figure 5. The spread angle of cones can either be uniform or non-uniform. Distributions of cones are precalculated and then rotated to match the normal space of each rendered point in space.

5 Atmosphere and Clouds

mallumo implements an atmospheric model of Bruneton et al. [3], which is a clear sky atmospheric model that assumes atmosphere made out of only molecules and small particles such as aerosols. The atmosphere itself is a spherical layer of these particles around a globe. The distribution of particles in the atmosphere is not uniform, however, the density at each point is parametrised only by height, not by altitude or longitude. While the precomputation phase for this model is rather costly, rendering is done in constant time.

mallumo also implements a method for cloud rendering based on the work presented by Schneider [21]. An example of two renderings that differ in the sun position can be seen in Figure 6. The method has a small precomputation phase in which are generated noise textures. The rendering part consists of raymarching along the view ray and accumulating light provided by the atmosphere.

5.1 Modelling Clouds

Modelling approach of clouds is based on raymarching, as described by Schneider [21]. View direction is determined for each pixel and samples are accumulated along the view ray. Density is computed at each sample point, and the sample is illuminated by the atmosphere, in this case using the Bruneton model. The density is given by

Figure 6: Renderings of the clouds with the sun at the zenith angle of 50° (top) and 66° (bottom).

a set of parameters whose product gives the final density. The first parameter is coverage, which determines whether there is a cloud. While technically the coverage parameter is either 0 or 1, it is better to define it as a gradient, to provide smoother scale and to prevent hard edges. The second parameter is given by altitude: the higher the sample, the denser the cloud. The third parameter is given by the cloud's type. We implemented three types of clouds: stratus, cumulus, and cumulonimbus. Again, these are implemented as gradients over altitude, and the cloud type parameter is a gradient over the types of clouds. With this method it is possible to define numerous types, however, this is not sufficient for the cloud types, for which is characteristic that individual clouds overlay each other.

5.2 Noise Textures for Clouds

This method of rendering clouds uses three noise textures and one control texture. At each point, the three noise textures are queried and combined, and the control texture provides information about coverage, cloud type, and turbidity. The textures are generated from the following noises: Worley noise, Simplex noise, and Curl noise.

Worley noise was introduced by Steven Worley [25] and is regularly used for caustics and water effects. It has round features, which make it suitable for modelling billow like shapes similar to clouds. Simplex noise was introduced and later improved by Ken Perlin [19]. Layering 2D Simplex noise at increasing frequencies creates visually interesting and detailed hills and valleys. This extended to 3D provides clouds with a more detailed look. Curl noise is used for fluid, fire, and smoke effects, and when used as a lookup table is a relatively inexpensive method to distort cloud shapes and add turbulence.

Two of the three noise textures are created by modulating and layering inverted Worley and Simplex noise. Curl noise is used for the third one. An example of layers of different noise textures can be seen in Figure 7.

5.3 Rendering Clouds

Rendering clouds is done by raymarching, i.e., for a point and a vector, a set of points is sampled along the ray. In the case of clouds, the data that are collected from samples are densities at each sample point. The densities that are collected are used in several ways. When raymarching a view ray, density is used to occlude what is behind the sample, and at each view ray's sample, it is also required to determine its colour.

Raymarching is costly which leads to poor framerate. This can be partially alleviated with the following techniques, at the loss of visual quality. Luckily, pixel level details are not necessary for clouds, so the loss is not that important. The most basic optimisation technique is to simply render at a lower resolution and then stretch the image. In addition to it, reprojection is based on exploiting the assumption, that the camera does not move rapidly between frames. Thus the majority of pixels would have mostly correct values if they were not redrawn, albeit they may be slightly moved, which is corrected by taking into account the previous view matrix.

6 Results

We compare the timings of several voxel-based cone tracing methods in the Table 1. Selected operations are: voxelization of a scene, injection of light into a voxel structure, and cone tracing with 3, 9, and 18 cones. The operations were performed on the Sponza testing scene. Results were obtained by running each operation 100 times and averaging the timings obtained. Tests were performed on Windows 10, NVIDIA Geforce GTX 1070 with 8GB of memory with driver version 391.24, Intel Core i7 920 CPU. Acronyms used in the table are:

- VT/SVO - type of storage, either Volume Texture (VT) or sparse voxel octree (SVO)
- ISO/ANI - type of voxels, either isotropic (ISO) or anisotropic (ANI)
- LDR/HDR - range of colours, either Low Dynamic Range (LDR) or High Dynamic Range (HDR)
- V - Voxelization operation
- RI - Radiance injection operation

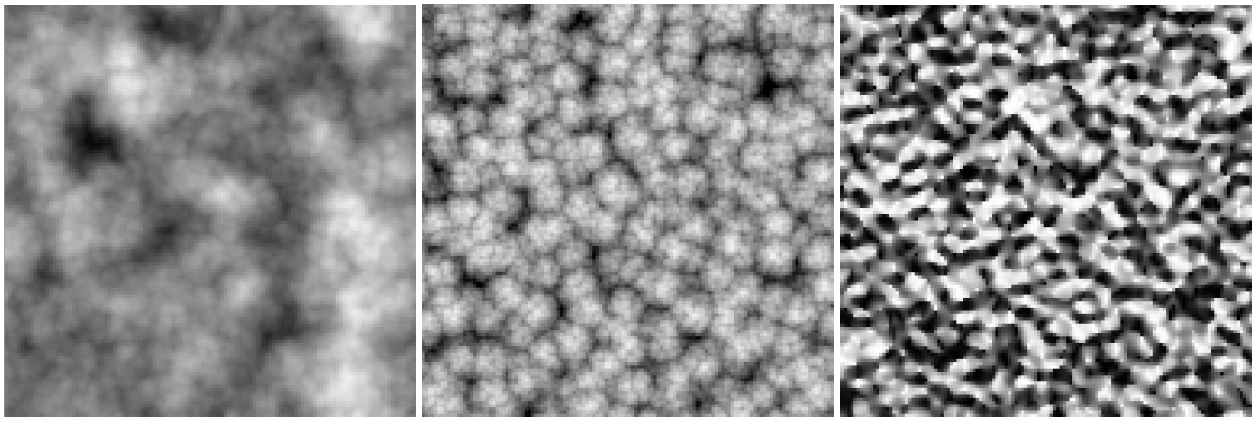

Figure 7: From left to right, examples of different layers of the noise textures for clouds generation: Simplex+Worley noise, layered Worley noise and Curl noise.

	Dimension	V (ms)	RI (ms)	CT_3 (ms)	CT_9 (ms)	CT_{18} (ms)
VT + ISO + LDR	256	5.03	1.17	1.42	4.17	10.00
VT + ISO + LDR	512	24.53	7.80	1.53	4.71	11.59
VT + ISO + HDR	256	14.33	3.36	4.83	22.87	60.03
VT + ANI + LDR	256	6.10	1.55	1.61	4.50	10.10
VT + ANI + LDR	512	34.11	10.72	1.70	5.96	13.17
VT + ANI + HDR	256	19.80	4.50	6.74	28.00	68.52
SVO + ISO + LDR	256	787.51	76.55	4.04	22.00	60.52
SVO + ISO + LDR	512	2201.27	124.79	4.81	26.61	74.01
SVO + ISO + HDR	256	725.39	74.44	5.80	31.16	84.90
SVO + ANI + LDR	256	954.42	130.45	3.14	13.27	34.07
SVO + ANI + LDR	512	2759.37	367.47	3.60	16.55	41.98
SVO + ANI + HDR	256	948.32	144.43	7.47	30.94	76.71

Table 1: Voxel-based cone tracing performance of most relevant configurations.

- CT_x - Cone Tracing with x number of cones

Some of the combinations of a storage miss resolution of size of 512 which is due to the fact that some of these configurations have higher memory usage than the GPU on which tests were performed. Even when a GPUs driver handles memory larger than available on GPU, it is simply moved to CPUs memory and retrieved on demand, leading to undesirable performance loss. For proper interpretation, it must be noted that voxelization and radiance injection steps include mipmapping of a voxel storage.

As can be seen, anisotropy has minimal performance penalty during cone tracing, but due to writing to 6 times more textures during mipmapping, it incurs cost during a voxel structure creation steps. Adding high dynamic range roughly doubles the time of cone tracing in case of volume texture, but can add far more significant overhead during texture creation, up to 10 times more. This is caused by both, using the mutex structure, and having to write 4 times more data. Sparse voxel octree incurs the highest cost in all stages. The voxelization step is slowed down by the need to go through voxel fragment list, nodepool, and brickpool before successfully voxelizing a scene. In the case of cone tracing, voxels must be first found by traversing the octree (nodepool) before retrieving them and manually interpolating them, both of which are costly.

The comparison of memory usage of voxel storage solutions is shown in Table 2. Comparing volume texture and sparse voxel octree is non-trivial. Therefore, the memory cost of all substructures of sparse voxel octree are in the table.

Voxel Fragment List and Node Pool are both overheads. Each node in the node pool always has a fixed size of 16 bytes per node, 4 bytes for next pointer and 12 bytes for pointers to their neighbours in X, Y, and Z axes. Each fragment contains its position and its albedo, resulting in a total of 16 bytes. Brick Pool is expressed as a number of voxels which is, for bricks of 3^3 voxels per node, 27 times number of nodes.

In the last column, a percentage of voxels used in sparse voxel octree compared to a volume texture is calculated. As can be seen, any resolution lower than 128 produces worse results in case of sparse voxel octree, which is caused by the fact that at such low resolution, every voxel is intersected by at least one polygon. The aforedescribed comparison is neutral to the amount of data stored in a voxel storage. For example, in case of a high dynamic

Dimension	Voxel Fragment List (MB)	Node Pool (MB)	Brick Pool (Voxels)	Ratio (%)
32	6.12	0.03	55 539	169.49
64	7.10	0.18	323 811	123.55
128	10.48	0.87	1 542 483	73.55
256	20.80	3.94	6 974 235	41.57
512	54.22	17.90	31 671 459	23.60
1024	173.46	84.21	149 002 011	13.87

Table 2: Memory Comparison of Volume Texure and Sparse Voxel Octree.

Downsampling / Reprojection	G1(ms)	G2(ms)	G3(ms)
1 / 2	1258.392	132.948	88.641
1 / 4	484.961	56.288	35.162
2 / 2	598.094	55.686	34.406
2 / 4	216.141	23.176	17.586
4 / 2	235.653	23.323	17.229
4 / 4	80.553	9.456	13.209

Table 3: Cloud rendering speed at FullHD resolution.

range, relative savings for sparse voxel octree are still the same, the actual storage is simply four times higher.

If compression of memory and amount of detail are the main concern and voxelization can be done as precomputation step, sparse voxel octrees may serve as a viable alternative to naive volume textures.

6.1 Clouds

Benchmarks were compiled with Rust 1.26, and were done on 3 PCs, all running Windows 10, with the following GPUs: NVIDIA GeForce GTX 860M, NVIDIA GeForce GTX 1070 and AMD Radeon RX Vega 64. These GPUs are listed as G1, G2, G3, respectively. The results are obtained by averaging 500 runs.

Table 3 shows that doubling downsampling ratio reduces the rendering time to half, in some configurations to one third. This is slightly unexpected as doubling downsampling ratio reduces the final image to a fourth of its size. One possible explanation is the overhead of the cloud rendering method, possibly waiting too long for the texture lookups. Doubling the number of frames needed to compose the final image using reprojection reduces the rendering time by two thirds on some configurations and by third on others. This can be explained by the overhead of reprojection, which is not worth on smaller resolutions, which downsampling imitates.

7 Conclusion

We presented our rendering library mallumo with its entire pipeline and all the associated techniques used to support our photorealistic techniques. We have provided a detailed description of our implementation, including voxel-based global illumination with focus on the support for high-dynamic range storage in voxels, anisotropic filtering, and cone distribution, as well as a procedural generation of atmosphere and clouds. In the future, we plan to optimize these techniques further and integrate them with existing real-time rendering software and game engines.

References

[1] John Amanatides. Ray tracing with cones. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 129–135, New York, NY, USA, 1984. ACM.

[2] J Appelbaum and Y Weiss. The packing of circles on a hemisphere. *Measurement Science and Technology*, 10(11):1015, 1999.

[3] Eric Bruneton and Fabrice Neyret. Precomputed atmospheric scattering. In *Proceedings of the Nineteenth Eurographics Conference on Rendering*, EGSR '08, pages 1079–1086, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[4] Brent Burley. Physically-based shading at Disney. In *ACM SIGGRAPH 2012 Courses*, New York, NY, USA, August 2012. ACM.

[5] Cyril Crassin. *GigaVoxels: A Voxel-Based Rendering Pipeline For Efficient Exploration Of Large And Detailed Scenes*. PhD thesis, Universite de Grenoble, 7 2011.

[6] Cyril Crassin and Simon Green. *Octree-Based Sparse Voxelization Using The GPU Hardware Rasterizer*, chapter 22. CRC Press, Patrick Cozzi and Christophe Riccio, 2012.

[7] Carsten Dachsbacher and Marc Stamminger. Reflective sha-dow maps. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 203–231, New York, NY, USA, 2005. ACM.

[8] David Ebert. Volumetric modeling with implicit functions: A cloud is born. *Visual Proceedings of SIGGRAPH*, 1997, 05 1997.

[9] Ned Greene. Environment mapping and other applications of world projections. *Computer Graphics and Applications, IEEE*, 6:21–29, 12 1986.

[10] Jörg Haber, Marcus Magnor, and Hans-Peter Seidel. Physically-based simulation of twilight phenomena. *ACM Trans. Graph.*, 24(4):1353–1373, October 2005.

[11] Brian Karis. Real shading in Unreal Engine 4. In *ACM SIGGRAPH 2013 Courses*, New York, NY, USA, August 2013. ACM.

[12] Alexander Keller. Instant radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[13] Áron Samuel KOVÁCS. Atmospheric effects for real-time global illumination, 2018 [cit. 2019-03-16].

[14] Gilles LAURENT, Cyril DELALANDRE, Gregoire De LA RIVIERE, and Tamy BOUBEKEUR. Forward light cuts: A scalable approach to real-time global illumination. *Compurer Graphics Forum (Proc. EGSR 2016)*, 35(4):79–88, 2016.

[15] Tomoyuki Nishita, Yoshinori Dobashi, and Eihachiro Nakamae. Display of clouds taking into account multiple anisotropic scattering and sky light. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 379–386, New York, NY, USA, 1996. ACM.

[16] Tomoyuki Nishita, Takao Sirai, Katsumi Tadamura, and Eihachiro Nakamae. Display of the earth taking into account atmospheric scattering. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '93, pages 175–182, New York, NY, USA, 1993. ACM.

[17] NVIDIA. NVIDIA VXGI. `https://developer.nvidia.com/nvidia-vxgi`, 2019. Accessed 11-February-2019.

[18] Eduardo Pavez, Philip Chou, Ricardo De Queiroz, and Antonio Ortega. Dynamic polygon clouds: Representation and compression for vr/ar. *APSIPA Transactions on Signal and Information Processing*, 7, 01 2018.

[19] Ken Perlin. Improving noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 681–682, New York, NY, USA, 2002. ACM.

[20] A. J. Preetham, Peter Shirley, and Brian Smits. A practical analytic model for daylight. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 91–100, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[21] Andrew Schneider. The Real-time Volumetric Cloudscapes of Horizon: Zero Dawn.

[22] Ari Silvennoinen and Jaakko Lehtinen. Real-time global illumination by precomputed local reconstruction from sparse radiance probes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 36(6):230:1–230:13, November 2017.

[23] Matúš TALČÍK. Comparison of cone tracing methods for real-time global illumination, 2018 [cit. 2019-03-16].

[24] Greg Ward. Ii.5 - real pixels. In JAMES ARVO, editor, *Graphics Gems II*, pages 80 – 83. Morgan Kaufmann, San Diego, 1991.

[25] Steven Worley. A cellular texture basis function. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 291–294, New York, NY, USA, 1996. ACM.