Real-Time Visualization of a Sparse Parametric Mixture Model for BTF Rendering

Nuno Silva[1,⋆], Luís Paulo Santos[2], and Donald Fussell[3]

[1] Centro de Computação Gráfica, Campus de Azurém, Guimarães, Portugal
[2] Dep. Informática, Universidade do Minho, Braga, Portugal
[3] The University of Texas at Austin, Texas, USA

Abstract. Bidirectional Texture Functions (BTF) allow high quality visualization of real world materials exhibiting complex appearance and details that can not be faithfully represented using simpler analytical or parametric representations. Accurate representations of such materials require huge amounts of data, hindering real time rendering. BTFs compress the raw original data, constituting a compromise between visual quality and rendering time. This paper presents an implementation of a state of the art BTF representation on the GPU, allowing interactive high fidelity visualization of complex geometric models textured with multiple BTFs. Scalability with respect to the geometric complexity, amount of lights and number of BTFs is also studied.

1 Introduction

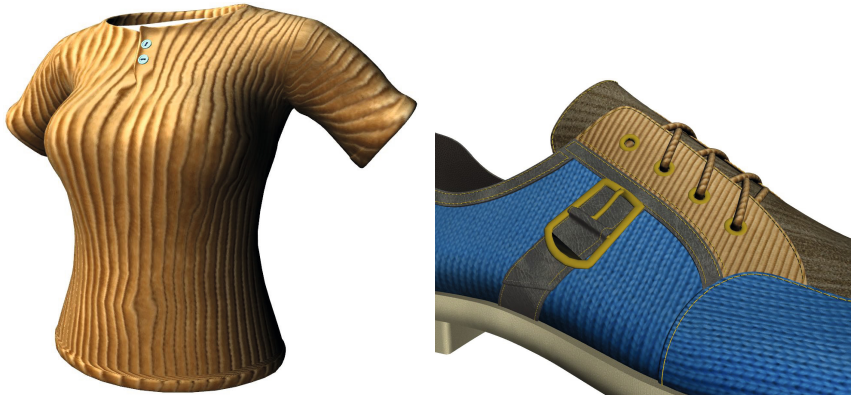
Digital representations of complex materials can have a major role in footwear and textile industries, assisting designers and artists in virtual prototyping new products and providing end-users realistic visualizations of such products. For the designer, the usefulness of these tools is greatly dependent on both the representation quality and high fidelity interactive visualization rates; achieving both these requirements is a challenging task.

A material's appearance depends on the way radiant flux is scattered when it hits a surface, and varies, among others, according to incoming light and observation directions [1]. Parametric Bidirectional Reflectance Distribution Functions (BRDF) are often used to model a material's appearance, but they cannot simulate many complex lighting phenomena such as self-shadowing, self-occlusion, sub-surface scattering and inter-reflections. Instead, image based approaches, such as the Bidirectional Texture Function (BTF), are becoming ever more popular due to the realism they can provide, the improvement in acquisition systems quality and the increasing computational power of GPUs [1–3].

The BTF is a 6D function [4] that models a material's appearance at a given point on the surface by recording several images captured under different lighting

⋆ Work partially funded by QREN project nbr. 13114 TOPIC Shoe and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project PEst-OE/EEI/UI0752/2011.

G. Bebis et al. (Eds.): ISVC 2012, Part I, LNCS 7431, pp. 719–728, 2012.



(a) (b)

Fig. 1. High fidelity interactive rendering: a corduroy shirt (a), and a shoe with multiple BTFs (b)

and viewing directions. The images are stored in large tables, and rendering involves simple look-ups within these. A single BTF can take up several gigabytes of storage, too much to be of practical use in real-time rendering. To tackle this problem, the BTF data must be transformed into a compact and efficiently renderable representation, without compromising image fidelity [2, 3, 5–7].

Wu et al. [8] presented a novel general representation for BTFs, the Sparse Parametric Mixture Model (SPMM). They demonstrate their approach using a parallel ray tracer, which, although achieving high quality visualizations, is far from interactive. This paper presents a rasterization oriented visualizer for SPMMs that achieves interactive frame rates for complex models involving multiple BTFs, without compromising on visual quality. Rendering times are dependent on the number of geometric primitives, number of fragments mapped with an SPMM and number of light sources; scalability is studied with respect to these parameters. The proposed interactive visualizer is currently being used by project partners in the Portuguese footwear industry.

This paper is organized as follows: first, background on material appearance and their representations is presented, along with the SPMM and other related work. Then, our approach to achieve real-time rates is described, followed by a discussion of results. The paper terminates with conclusions and future work.

2 Appearance Modeling and Visualization

Most real world materials exhibit complex appearance that can be described at three levels [2, 9]. The macroscale is the large scale geometry of the object, traditionally modeled with explicit representations, such as polygon meshes. The microscale level relates to interactions of light with a point on the surface of the material, and can be represented using BRDFs. The mesoscale level is in

between these two and comprises various subtle lighting effects such as self-shadowing, self-occlusion, sub-surface scattering and inter-reflections, which cannot be faithfully represented with the BRDF; instead, image based approaches are used, the most common method being texture mapping. Spatially-Varying BRDFs (SVBRDF) [10], which can be seen as a combination of texture mapping and BRDFs, account for materials that have different BRDFs throughout their surfaces; while addressing some of the issues raised at the mesoscale level, they cannot capture self-shadowing and self-occlusion. For that, BTFs, an image driven approach, are often used instead. A Bidirectional Subsurface Scattering Reflectance Distribution Function (BSSRDF) can model all these light phenomena, but is much too complex to be usable in interactive rendering pipelines, and current capturing systems only allow to measure subsets of this function [1, 2].

It must be noted that appearance can be reproduced by following a procedural approach, i.e., hand-tuning an algorithm and mathematical functions until the desired effect is achieved [11]. This is, however, a time consuming task that demands high level of expertise and might still fail to accurately simulate real world materials. Image based approaches are ever more popular because appearance is measured using cameras, demanding no particular expertise, thus being more adequate for project partners in the Portuguese footwear industry.

2.1 Bidirectional Texture Function

The BTF emerged as an alternative that represents both mesoscale and microscale levels. For each wavelength, it models the material appearance based on a point on the surface (x), and the incident and reflection directions (ω_i, ω_o).

BTFs model the appearance of a material from several images captured under different observation and lighting directions; they can be seen as a special class of the SVBRDF since surfaces are assumed to be planar [12]. A good quality BTF, such as the ones in the Bonn database [7], encodes 81×81 images for light and viewing directions, each consisting of 256^2 texels with three spectral values (RGB). This corresponds to roughly 1.2GB of raw data for a single BTF, not including High Dynamic Range (HDR) effects.

Achieving interactive visualization rates of objects with multiple BTFs requires compressing the raw data, while preserving as many of the relevant features of the BTF as possible. Compression must exploit the redundancy in the data in an efficient way, and allow fast decompression for real-time rendering. Refer to [2] and [3] for further details on BTF modeling.

2.2 The Sparse Parametric Mixture Model

In the SPMM representation proposed by Wu et al. [8], the captured data is analyzed and fit into a number of different parametric functions, each defined as a cosine-weighted rotated BRDF. Equation 1 describes such functions, where $f_j(k_j, \cdot)$ is one analytical BRDF model, with parameters k_j. R is a rotation that transforms a vector into the local coordinate system defined by local normal n_j.

$$\rho_j(\omega_i, \omega_o) = f_j(k_j, R(\omega_i), R(\omega_o))(n_j \cdot \omega_i). \quad (1)$$

The original data at a point x can be approximated by using a weighted linear combination of m such functions (see equation 2 and figure 2), each with weight α_j. Subtle appearance details that cannot be fit into the parametric functions are stored as a residual function, ϵ_x, which is obtained by subtracting the linear combination of parametric functions from the original BTF.

$$\mathrm{BTF}_x(\omega_i, \omega_o)(n_x \cdot \omega_i) = \sum_{j=1}^{m} \alpha_j \rho_j(\omega_i, \omega_o) + \epsilon_x(\omega_i, \omega_o). \quad (2)$$

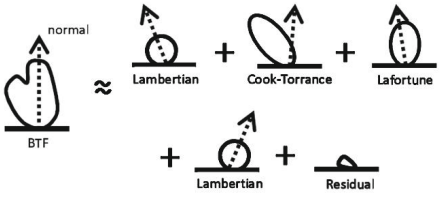


Fig. 2. Illustration of the SPMM representation. The BTF is approximated by a sum of analytical models and a residual part. Each analytical model has its own local frame. Adapted from Wu et al. [8].

Since fitting all the texels of the BTF is computationally expensive, spatial coherence of the original data is exploited through multilevel k-means clustering. The full fitting algorithm is applied to some selected representative texels, and the resulting parametric functions are used as a dictionary to accelerate the fitting of the other texels in the cluster. The residual function ϵ_x is also computed on a per-cluster basis. It is suggested that ϵ_x can be improved by storing a few additional basis error functions and respective coefficients obtained from Local Principal Component Analysis (LPCA) [5]; this allows marginal improvements of image quality at the cost of increased memory requirements. We support both options, allowing adaptation of image quality, and thus rendering times, to the available computing power on the graphics board (see sections 3.1 and 4).

By fitting BTF data into a sum of parametric functions, the SPMM provides a general representation that allows the volume of data to be significantly reduced, enables efficient rendering and intuitive editing of parameters. Wu et al. tested the SPMM with the Bonn database [7] of BTFs; our GPU implementation is based on those representations.

3 Implementation

Our interactive visualizer uses the OpenGL API to communicate with the GPU and GLSL to program the shading process. Efficient use of the GPU requires

SPMM data to be stored in the device texture memory, as 1D, 2D or 3D arrays [13, 14], allowing fast random accesses and high bandwidth. Thus, texture memory stores the cluster identifier for each texel, the parametric functions ρ_j, their corresponding parameters k_j and weights α_j, and the residual function ϵ_x.

The SPMM is not geared towards optimal GPU performance because it consists of a linear combination of different analytical BRDFs, and directly translating the CPU shader into a GPU shader results in a lot of branching and loop instructions (and a very large shader code base). Transforming the data structures in order to fit the GPU streaming programming model is also challenging.

First, OpenGL only allows to store basic data types into texture memory: 1, 2, or 4 byte words, fixed or floating point. The data format used is a crucial aspect in GPUs because it determines the amount of storage and bandwidth required; we minimize the number of bytes required for each texture. SPMMs generated from BTFs in the Bonn database [7] group texels into 32 clusters, thus the cluster identifier can be stored with a single byte. Other textures that use fixed point values store indexes or identifiers and so they are represented with 2 bytes. Textures that store the parametric function weights, their parameters and the residual function are in the floating point format, encoded in half-precision with 2 bytes. In our experiments this does not produce visible artifacts whilst greatly reducing memory requirements and improving render time.

The second issue is that the number of parametric functions m is not the same across all texels, and dynamically sending this information to the GPU would make real-time rendering hard to achieve. Calculating the maximum number of functions for all texels and letting all fragment shaders do roughly the same amount of work resulted in poor performance. Our implementation precomputes m for each texel and stores it in an additional texture. This resulted in higher frame rates, especially noticeable when the number of parametric functions varies greatly from one texel to another, i.e., when the BTF exhibits locally complex reflectance variations.

The third problem arises from the sparse discretization of the view/lighting directions during the BTF measurement. In order to render the BTF under novel viewing or lighting directions, interpolation of the closest view/lighting slots must be performed. The residual function must be interpolated in order to avoid the appearance of artifacts when these slots change. The parametric functions are not affected, since they are defined over the entire upper hemisphere. Those direction slots can be interpreted as 3D points and projected onto a set of points on the XY plane by ignoring the Z component. We then apply a Delaunay triangulation and store the resulting triangles in a texture. Interpolating now consists of a ray-triangle test using barycentric coordinates [15], resulting in the appropriate interpolation weights in case of a hit. This must be done separately for the view and lighting directions, for a total of nine interpolation weights.

Finally, 2D arrays are transformed into slices of 3D arrays to avoid exceeding the capabilities of the hardware. Figure 3 depicts the used data structures (some of them presented in the following section) and the data flow to render the full SPMM.

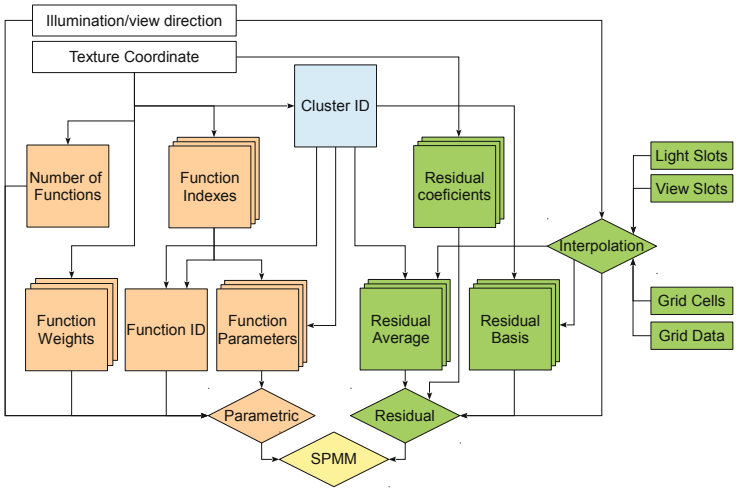


Fig. 3. The data flow in the fragment shader. The small rectangles, the squares and the stacked squares depict, respectively, 1D, 2D and 3D textures. The diamonds represent the combination of the inputs.

3.1 Optimizations

Profiling analysis indicated a bottleneck in the interpolation of the residual function. Since this is based on a ray-triangle test and the triangles are static and well distributed over the unit circle, a regular grid is used to quickly discard triangles, thus limiting intersections test to a (very small) subset of the triangles. A compact grid structure with minimal memory requirements [16] is built once in the CPU and uploaded to the GPU using 2 1D textures.

Equation 3 is used to perform the ray-triangle intersection test, where λ is the vector of barycentric coordinates, r is the ray vector, and T is a matrix formed by the Cartesian coordinates of the triangle vertices. This allows precomputation of the inverse matrix T^{-1} for each triangle, reducing the intersection test on the GPU to a vector subtraction and a matrix-vector multiplication. Additionally, to maximize data locality, the inverse matrix, the Cartesian coordinates of each vertex and its corresponding ID are packed together, for a total of 16 floats per triangle. With both these optimizations the computational cost of evaluating the residual function is roughly the same as evaluating the BRDFs, whereas before optimizations, the former was around 3 times longer than the later.

$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = T^{-1}(r - v_3); \\ \lambda_3 \quad = 1 - \lambda_1 - \lambda_2. \tag{3}$$

To take full advantage of the bandwidth of the GPU, we exploit the SIMD patterns shader instructions, and strive to perform texel fetches of the RGBA

channels of each texture. However, not all textures can have data packed in order to use all the available channels, and some demonstrated performance losses with this new arrangement of data. As we are aware this can change between GPU vendors and families of the same vendor, not much effort was put into fine tuning the data layout of textures.

Finally, at rendering time the number of LPCA components used to evaluate the residual function (see section 2.2) can be limited by a user defined parameter, allowing the program to adapt to the computing capabilities of the host machine. In our experiments, this can greatly increase the frame rate whilst having marginal impact on image quality.

4 Results

Experiments were conducted on a workstation equipped with an Intel 2.4GHz quad-core processor, 4GB RAM and with a Nvidia GeForce GTX 580 GPU, driver version 301.32. All tests were performed using the SPMM representation of the BTFs in the Bonn Database [7], which have a spatial resolution of 256×256 texels and an angular resolution of 81×81 directions. All the presented tests use the Wool SPMM (except when otherwise explicitly stated), using half precision floating point values (16 bits) which corresponds to 12.39MB of texture data; all the other SPMMs demonstrate similar results. The render target size is 512×512 pixels, all the LPCA coefficients in the residual function are evaluated for maximum visualization quality, and all the reported values are the mean of a 60 second run profiled using Nvidia Parallel Nsight 2.2.

4.1 Results Analysis

Figure 4(a) presents the frame rates achieved as a function of the percentage of pixels in the render target covered by a SPMM fragment. It is clear that our application is fill limited, since the most complex part of the code, the SPMM evaluation, is completely written in a GLSL fragment shader. Nevertheless, even with 80% of the pixels requiring an SPMM evaluation we achieve above 200 fps with our hardware configuration.

Figure 4(b) depicts performance variation with the number of visible SPMMs. The shoe model in figure 1(b) was initially mapped with the wool SPMM on five different materials; each new test replaced one of those materials with a different SPMM, until five were being used simultaneously, this way maintaining the same number of pixels covered by SPMMs in all tests. Results demonstrate that the number of SPMMs does not significantly affect frame rates, small differences being due to variations in the parameters that define each SPMM.

Since GPU hardware changes considerably with each new family, and associated compilers also differ accordingly, we used the GPU ShaderAnalyser tool from AMD to analyze and predict the fragment shader performance on various GPUs. We configured the analyzer to assume branch coherence of 90%, an

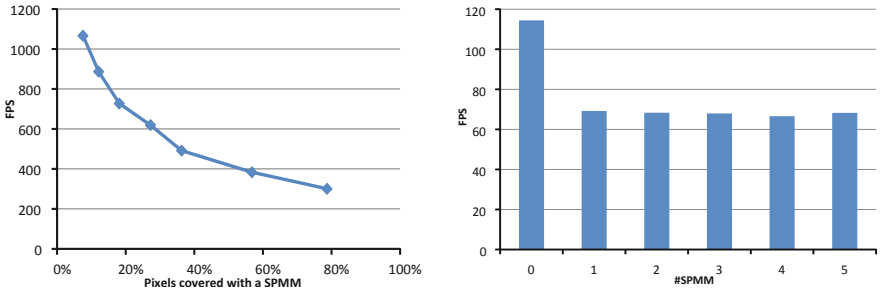


Fig. 4. (a) Performance variation with the amount of pixels covered by the Wool SPMM. (b) Performance variation with the number of SPMMs for a fixed number of pixels covered by SPMMs.

average loop count of 4 and maximum loop count of 8. The reported results for 4 diferent AMD Radeon GPU families are presented in table 1. These show that the shader is currently compute bound (see ALU:TEX and Bottleneck columns), and throughput increases sharply with the higher clock rates and core count available in newer GPU families. It is therefore to be expected that our visualizer performance will continue to scale across new generations of GPUs.

Table 1. GPU ShaderAnalyzer output for various AMD Radeon GPUs. Avg - Average number of cycles the shader is expected to take; ALU - The number of ALU instructions in the shader; TEX - The number of texture fetch instructions in the shader; CF - The number of control flow instructions in the shader; Throughput - Millions of pixels per second; CR - Clock rate in MHz; CC - Number of stream processors .

Name	Avg	ALU	TEX	CF	ALU:TEX	Bottleneck	Throughput	CR	CC
HD3870	263.06	826	105	213	**2.41**	ALU Ops	47	775	320
HD4890	105.22	840	109	213	**2.13**	ALU Ops	129	800	850
HD5870	55.94	845	109	212	**1.07**	ALU Ops	258	850	1600
HD6970	48.07	940	109	215	**1.17**	ALU Ops	293	880	1536

4.2 Scalability Analysis

Scalability of the visualizer with respect to the model and illumination complexity is of paramount importance on an industrial setting. Figure 5 depicts rendering times for various geometric complexity and directional light sources.

Rendering times increase linearly with the number of light sources. This is due to the entire evaluation of the fragment shader for each light, but as indicated in figure 3, not all data structures depend on the light direction; exploiting this fact in future implementations can improve scalability with the number of lights.

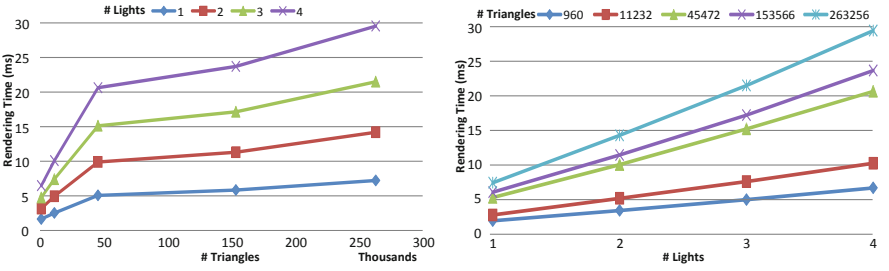


Fig. 5. Rendering times with increasing geometric complexity (a) and number of directional light sources (b)

The number of geometric primitives also has a significant impact on rendering time. Although the SPMM is entirely calculated in a fragment shader, thus independent from the vertex processing stage, our visualizer is not prepared to handle large amounts of vertex data. Since the final goal is to integrate it on a footwear CAD system developed by a project partner, this will feed the visualizer with only the visible geometric primitives. Utilization on other contexts is possible by applying appropriate culling techniques.

Nvidia Parallel Nsight reported, for all experiments, that the GPU is busy processing the workload 92% of the time of each frame, which reinforces evidence that the shader is compute bound and computing resources are being used near their peak performance.

5 Conclusion and Future Work

We presented a GPU visualizer that combines the compaction benefits of the original SPMM approach, a state of the art BTF representation format, with the performance benefits of more GPU friendly approaches, enabling high fidelity visualization at previously unreachable interactive rendering rates. It was shown that performance is fill rate limited, that the main bottleneck is the number of ALU operations in the shaders and the number of rendered SPMMs does not affect performance. By precomputing barycentric coordinate matrices and using acceleration structures we were able to further increase rendering rates, exploiting on average 92% of computational resources.

As future work we would like to improve scalability and expand the visualizer to allow real-time editing of SPMM parameters. We believe this can be of great use for digital designers and artists, assisting in rapid virtual prototyping of new products. Additionally, it would be interesting to support multiple GPUs and progressive rendering in order to adapt to the compute capabilities of the host machine. Also, mipmapping can boost performance and reduce aliasing artifacts.

References

1. Weyrich, T., Lawrence, J., Lensch, H., Rusinkiewicz, S., Zickler, T.: Principles of appearance acquisition and representation. In: ACM SIGGRAPH 2008 Classes, SIGGRAPH 2008, pp. 80:1–80:119. ACM, New York (2008)
2. Müller, G., Meseth, J., Sattler, M., Sarlette, R., Klein, R.: Acquisition, synthesis, and rendering of bidirectional texture functions. Computer Graphics Forum 24, 83–109 (2005)
3. Filip, J., Haindl, M.: Bidirectional texture function modeling: A state of the art survey. IEEE Transactions on Pattern Analysis and Machine Intelligence 31, 1921–1940 (2009)
4. Dana, K.J., van Ginneken, B., Nayar, S.K., Koenderink, J.J.: Reflectance and texture of real-world surfaces. ACM Trans. Graph. 18, 1–34 (1999)
5. Müller, G., Meseth, J., Klein, R.: Compression and real-time rendering of measured btfs using local pca. In: Ertl, T., Girod, B., Greiner, G., Niemann, H., Seidel, H.P., Steinbach, E., Westermann, R. (eds.) Vision, Modeling and Visualisation 2003, pp. 271–280. Akademische Verlagsgesellschaft Aka GmbH, Berlin (2003)
6. Ma, W.C., Chao, S.H., Chen, B.Y., Chang, C.F., Ouhyoung, M., Nishita, T.: An efficient representation of complex materials for real-time rendering. In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST 2004, pp. 150–153. ACM, New York (2004)
7. Sattler, M., Sarlette, R., Klein, R.: Efficient and realistic visualization of cloth. In: Eurographics Symposium on Rendering 2003 (2003)
8. Wu, H., Dorsey, J., Rushmeier, H.: A sparse parametric mixture model for btf compression, editing and rendering. Computer Graphics Forum 30, 465–473 (2011)
9. Suykens, F., Berge, K.V., Lagae, A., Dutr, P.: Interactive rendering with bidirectional texture functions. Computer Graphics Forum 22, 463–472 (2003)
10. McAllister, D.K., Lastra, A., Heidrich, W.: Efficient rendering of spatial bidirectional reflectance distribution functions. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, HWWS 2002, pp. 79–88. Eurographics Association, Aire-la-Ville (2002)
11. Ebert, D.S., Musgrave, F.K., Peachey, D., Perlin, K., Worley, S.: Texturing and Modeling: A Procedural Approach, 3rd edn. Morgan Kaufmann Publishers Inc., San Francisco (2002)
12. Lawrence, J.: Acquisition and representation of material appearance for editing and rendering. PhD thesis, Princeton, NJ, USA, AAI3214568 (2006)
13. Fernando, R.: GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics. Pearson Higher Education (2004)
14. Pharr, M., Fernando, R.: GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems). Addison-Wesley Professional (2005)
15. Pharr, M., Humphreys, G.: Physically Based Rendering: From Theory to Implementation, pp. 125–130. Morgan Kaufmann Publishers Inc., San Francisco (2004)
16. Lagae, A., Dutré, P.: Compact, fast and robust grids for ray tracing. In: Computer Graphics Forum (Proceedings of the 19th Eurographics Symposium on Rendering), vol. 27, pp. 1235–1244 (2008)