A Survey on Methods for Omnidirectional Shadow Rendering

Jan Navrátil
inavrati@fit.vutbr.cz
Faculty of Information Technology
Brno University of Technology,
Brno, Czech Republic

Jozef Kobrtek
xkobrt00@stud.fit.vutbr.cz
Faculty of Information Technology
Brno University of Technology,
Brno, Czech Republic

Pavel Zemčík
zemcik@fit.vutbr.cz
Faculty of Information Technology
Brno University of Technology,
Brno, Czech Republic

ABSTRACT

This paper focuses on methods of rendering shadows cast by point light sources. The goal is to summarize advantages and disadvantages of methods based on shadow mapping. We compare the traditional approach that exploits cube maps with the Dual–Paraboloid mapping. All of the methods are implemented on the latest hardware and they exploit capabilities of current GPUs. We also implemented optimization techniques which decrease the computational time. We examine the time the methods spent in particular rendering passes and we evaluate their overall performance. Finally, we conclude the comparison with some recommendations for typical applications in which the methods of interest can be exploited. We also suggest some direction of future investigation.

Keywords: shadow mapping, rendering, GPU, performance, cube maps, Dual–Paraboloid mapping

1 INTRODUCTION

Shadows play very important role in modern graphics applications as they increase overall visual cue from a rendered image. The shadow mapping algorithm [Wil78] and the technique based on shadow volumes [Cro77] are the most popular techniques for adding shadows to 3D scenes.

A well known disadvantage of the shadow mapping algorithm is the limited resolution of textures which store the depth information. Furthermore, it is also difficult to render shadows cast from point light sources. The basic shadow mapping algorithm cannot cover the whole environment with a single texture and thus additional computations are required. Such additional computations decrease the performance especially in scenes with a complex geometry.

The technique based on shadow volumes can easily render shadows from point light sources with per pixel accuracy. However, a high fill rate rapidly reduces the computational performance even for moderate sized scenes. Even though some optimization approaches exist [LWGM04], interactive applications mostly use the shadow mapping algorithm.

In this paper, we investigate several approaches for rendering shadows cast from point light sources based on the shadow mapping algorithm. Our contribution is the evaluation of the advantages and disadvantages of the approaches. We compare the existing methods especially with respect to their performance. We present some figures related to the time spent on generation of shadow maps on GPUs [MGR+05, Gru07] and also some frame times related to a camera view. We will also discuss the efficiency of all of the presented methods and potential implementation problems related to GPUs. Since the paper is restricted to the specific case of the shadow mapping algoritm we do not consider the shadow volumes approaches [LWGM04, VBGP09] as well as techniques that increase visual quality of shadows [WSP04]. Because they add some additional processing steps that might influence the results.

In Section 2, we refer to some techniques related to shadow rendering. We also mention some existing surveys. Section 3 introduces some issues that may arise when implementing the presented methods. We demonstrate all of the methods and their optimization in Section 4 and in Section 5, we present our experiments and discuss their results. We conclude our work in Section 6 where we also suggest some areas of future investigation.

2 RELATED WORK

For high quality shadow rendering, techniques such as ray tracing can be used. However, the shadow volumes algorithm or the shadow mapping approach are the most frequently used in interactive applications. The shadow volume technique [Cro77] provides per-pixel accuracy, its main disadvantage is a huge required fill rate. This fact does not allow for its common use in interactive applications. We can, however, find some methods that reduce the fill rate [LWGM04]. Nevertheless, the shadow mapping is the most popular algorithm for shadow ren-

dering. Basically, two approaches exist to render shadows cast from omnidirectional light sources using the shadow mapping algorithm. Firstly, shadow maps can be represented by faces of a cube map [Ger04]. In this case, six render passes are needed to fill the data into the cube map faces. Secondly, the Dual–Paraboloid mapping technique [BAS02, OBM06] can be used. It is capable of capturing the whole environment in two render passes. However, the mapping is not linear and thus not fully supported by contemporary graphics hardware. Recently, different techniques have been introduced [CVM11, HWL+11] that discuss other types of parametrizatons.

All of the above mentioned methods are capable of rendering shadows cast from omnidirectional (point) light sources. However, they all have some limitations and their usage may depend on the application and on scene complexity. Some surveys of shadow rendering have already been published, but they generally compare visual quality of the shadows with respect to the aliasing error [SWP10] or they address problem of soft shadow rendering [HLHS03]. In these cases, mostly directional light sources have been taken into account. The omnidirectional light sources need an extra treatment for creating shadow maps but also for reducing the aliasing error. Vanek et al. [VNHZ11] did some experiments with Dual–Paraboloid mapping technique but they did not work with the cube maps approach at all. They considered the cube map approach ineffective for omnidirectional light sources.

3 ISSUES OF THE SHADOW MAPPING ALGORITHM

3.1 Overview of the Algorithm

The first step of the shadow mapping algorithm is creation of the shadow map. A virtual camera is placed in the position of a light source. Then the scene is rendered as viewed from the virtual camera and the depth information is captured in the shadow map. In the second step, the scene is rendered from a camera point of view and the rendered pixels are compared with values stored in the shadow map.

During the process of the creation of the shadow map, the geometry has to be transformed to the light space coordinate system. For this purpose, the transformation matrix has to provide an appropriate transformation based on the type of the light source.

3.2 Linear Transformation and Interpolation

In case of directional light sources, orthogonal projection is used since all of the light rays have the same direction. For spotlights, perspective projection is used since the spotlights cover only certain part of the scene.

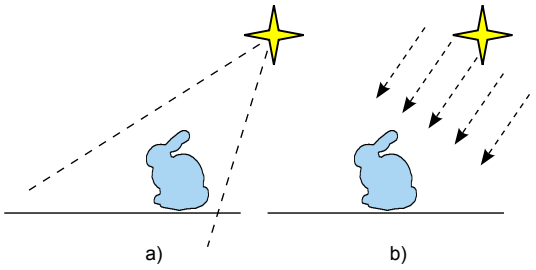

Figure 1: (left)A virtual camera for spotlights creates the view frustum. The frustum covers only a part of the scene based on a direction of the spotlight. (right) Directional lights use orthographic projection, because direction the light rays are parallel.

Then, the field-of-view in perspective projection is similar to the concept of falloff angle in spotlights. (see Figure 1). The perspective projection has a limited field-of-view range and thus it can not cover the whole environment. However, both projections are linear and thus they do not allow for covering the 180 degree field-of-view appropriately. To cover the whole environment, multiple linear projections are required. This means that if we want to use the basic shadow mapping algorithm for omnidirectional light sources, multiple render passes are necessary to create the shadow map (see Figure 2). Otherwise, a non-linear transformation has to be used.

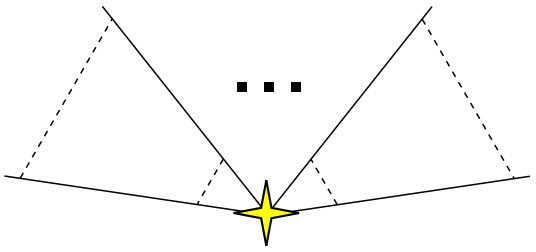

Figure 2: Multiple frusta have to be placed next to each other to cover the whole environment.

When we apply a projection, represented by a matrix, on vertices in the vertex shader, the fragments in the fragment shader are linearly interpolated. Instead of multiple linear projections, we can apply a non-linear transformation. The non-linear transformation, however, does not work well with the interpolation scheme used in graphics hardware. Straight lines are curved after the transformation (see Figure 3). It causes unwanted artifact for large polygons. The solution for these artifacts is to refine tessellation of the scene. For small polygons, the artifacts are not noticeable.

3.3 Limited Resolution

Stamminger et al. [SD02] described two types of aliasing: *perspective* and *projection*. Perspective aliasing is caused by limited resolution of shadow texture when

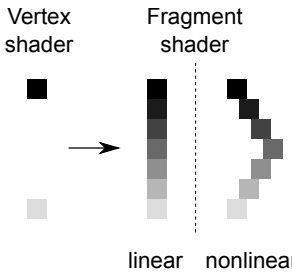

Figure 3: Fragments, that have to be rasterized between two vertices, are linearly interpolated in fragment shaders. Nonlinear parameterization can cause that fragments do not lie on a line.

the shadow map is undersampled while projection aliasing appears when the direction of light rays is parallel to the surface. Some methods exist that try to reduce the perspective aliasing artifacts on shadow boundaries. The shadow map can be filtered to make the shadow smooth [Fer05].

4 OVERVIEW OF METHODS

In this section, we present an overview of various methods for rendering shadows cast from omnidirectional light sources. We describe principles of each of the methods and we discuss their advantages and disadvantages. Furthermore, we present some optimization techniques that eliminate some of the disadvantages in order to achieve the best results for each of the methods. For our purpose, we only deal with omnidirectional light sources. It means that the light is emitted from a single point in space: therefore, we neglect an extent of the light source.

4.1 Cube Shadow Maps Technique

In Section 3, we mentioned how problematic it is to cover the whole environment with traditional projection transformations. In order to create shadow maps for an omnidirectional light source, it is necessary to point the virtual camera into six directions. The view direction of the virtual camera should point toward directions defined by the axes of the local coordinate system of the cube: positive X, negative X, positive Y, negative Y, positive Z and negative Z. This is almost identical to the way how a cube map for environment mapping is generated except that in this case depth values are stored instead of color.

Basics of the Cube Shadow Maps

The faces of the cube represent shadow maps and directions of the faces shows the particular direction for the virtual camera (see Figure 4). In order to cover the whole environment, the traditional shadow mapping algorithm exploits cube maps to visualize shadows cast from point lights. To fill the data in the cube shadow map, six render passes have to be performed. The GPUs generally support the cube shadow maps which are thus easy to implement.

The biggest disadvantage of the cube shadow maps is that six render passes are often too expensive. This fact can cause rapid decrease of performance for complex scenes with high number of polygons. Even if per-object frustum culling is applied, rendering of shadows is still very expensive compared to rendering of the rest of the scene.

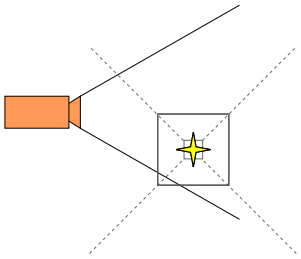

Figure 4: Illustration of the cube shadow maps technique. Each face of the cube stores depth values for a certain part of the scene.

Efficient Cube Face Frustum Culling

The methods of reduction of the number of passes have been investigated [KN05]. If the light source is outside the view frustum, then we can skip rendering of at least one face of the shadow map. This leads to the most noticeable effect on the performance

For our experiments, we use the following technique for efficient culling of cube map faces. A camera view frustum and each cube map frustum are tested for their mutual intersection. Those frusta that do not intersect can be discarded for further rendering because they do not affect the final image. The efficient culling of arbitrary frustum F against the camera view frustum V works as follows. The frusta are defined by 8 boundary points and 12 boundary edges. To determine whether the two frusta intersect, two symmetric tests have to be performed. Firstly, it should be tested whether a boundary point of one frustum lies inside other frustum (see Figure 5a). Secondly, it should be tested whether a boundary edge of one frustum intersects one or more clip planes of other frustum (see Figure 5b) [KN05].

For each face of the cube shadow map, we investigate whether the camera view frustum intersects the shadow face frustum and vice versa. If it is not the case, the shadow face frustum does not affect the scene and we can skip the additional processing (see Figure 6).

It is also necessary to take into account shadow casters outside the view frustum. If we cull the shadow caster against the view frustum, the projected shadow may still be visible in the view frustum. On the other

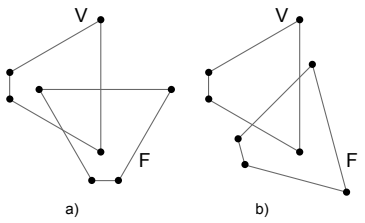

Figure 5: A frustum consists of boundary points and boundary edges. Two frusta intersect when (a) at least one boundary point of the frustum F lies inside other the frustum V or (b) at least one boundary edge of the frustum F intersects a face of the frustum V.

hand, culling the shadow caster against the cube map frustum draws invisible shadows as well. King et al. [KN05] suggest to use frustum-frustum intersection test described above for the shadow casters as well. Since we use point light sources, rays are emitted from a single point towards all shadow casters. This is analogous to the perspective projections. If the shadow casters are enclosed by bounding objects, frusta representing the projected shadows can be created [KN05] and then the frustum-frustum test can be applied in this case as well. These tests are performed once per frame.

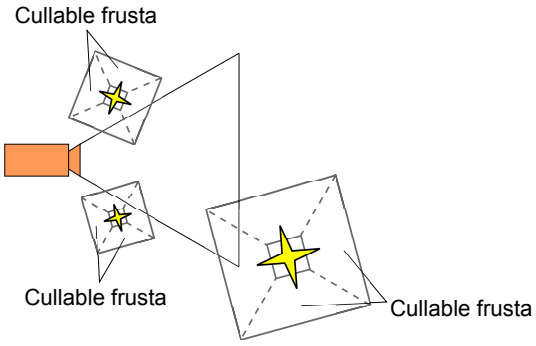

Figure 6: If the light source lies outside the camera view frustum, at least one face is cullable.

4.2 Dual–Paraboloid Mapping Algorithm

In the following text, we will discuss the Dual–Paraboloid Mapping algorithm (DPSM) [BAS02]. The mapping is based on two paraboloids attached back-to-back, each capturing one hemisphere:

$$f(x,y) = \frac{1}{2} - \frac{1}{2}(x^2+y^2), \quad x^2+y^2 \leq 1 \qquad (1)$$

In principle, a single hemisphere mapping can be imagined as an exploitation of a totally reflective mirror which reflects incident rays from the hemisphere into the direction of the paraboloid (see Figure 7). The rays may carry some information about the environment (mostly distance to the light) and the information can be stored into a texture. The texture coordinates are computed according to coordinates of the point where the ray is reflected. The Dual–Paraboloid mapping basically maps 3D space to 2D which is represented by the shadow map.

The algorithm needs only two render passes to capture the whole environment. Thus, it is more efficient than the cube shadow maps technique. Other parameterization can be certainly found (spherical, cube mapping etc.) but the proposed parabolic parameterization maintains its simplicity and performance, e.g. in GPU implementation [OBM06]. It minimizes the amount of used memory and the number of render passes that are necessary to cover the whole environment.

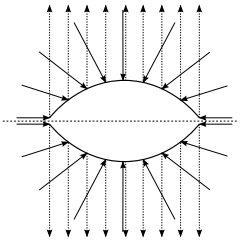

Figure 7: Two paraboloids attached back-to-back can capture the environment from all directions.

Nevertheless, the DPSM algorithm has also some disadvantages. While in the cube shadow map approach, all the transformations needed to create the shadow map are linear, they do not need any extra treatment on GPUs. This mainly concerns interpolation process between vertex and fragment shader (see Sec 3). When using the DPSM algorithm, the rendered scene needs to be finely tessellated because the mapping is not linear and thus it does not work well for large polygons. It may, however, introduce new bottlenecks.

4.3 Limitations of Geometry Shader

It is also possible to exploit both shadow mapping methods utilizing a geometry shader in order to reduce the number of render passes from six (two in Dual–Paraboloid mapping algorithm) to a single one [Eng08]. In this case, we exploited capablities of the frequently used graphics card, i.e., NVIDIA GeForce GTX560Ti, which supports geometry shaders.

The core of this method is usage of multiple render targets for and rendering all of the six cube map faces at once. The geometry shader transforms each of the incoming triangles with view-projection matrix of the corresponding cube. A naive approach sends all the incoming geometry data to all render targets, producing three to five times more geometry data than necessary. Such data is, however, anyhow discarded in the following rendering phases by the rasterizer. This leads to a massive performance penalty, as seen in Table 1. The results were measured on the same scene with the shadow map resolution set to 1024^2.

	avg. FPS
Cube6	6.19
Cube6Optim	20.3
DP	18.81
DPOptim	30.90

Table 1: All the methods exploit geometry shader and render the shadow maps in one pass.

This method was further optimized by testing each object bounding sphere against view frusta of the cube map faces, or, in case of Dual–Paraboloid mapping algorithm, against plane dividing scene in place of both paraboloids. Cube shadow mapping method was sped up by 227%, but still resulting in a very poor performance. Dual–Paraboloid mapping approach did not benefit that much from optimization, resulting in only 64% increase of performance, but also scoring far less than multi-pass methods.

Despite the optimizations, these methods did not overcome above mentioned optimized 6-pass techniques (described in Section 4.1). The core problem of the geometry shader is its execution model. It outputs data in a serial fashion with no parallelism used. Utilizing vertex shader and multiple passes overcomes the above mentioned geometry shader solutions despite switching of the render targets and updating resources between render calls.

5 EXPERIMENTAL RESULTS

We implemented the experimental framework in DirectX11 on an Intel Core i5 CPU 661 running at 3.33GHz using NVIDIA GeForce GTX560Ti GPU. The rendered images have resolution of 1024×768. We used the 32bit render target for the shadow maps. The resulting graphs were generated from an experimental walkthrough of a demo scene. The benchmarking scene had approximately 3 millions of vertices.

Our implementation does not introduce any hardware specific features. We can assume that the difference between the approaches would not be principally different.

5.1 Frame Time in Walkthrough

The first measurement shows dependence of the frame time for the walkthrough of the scene for all of the implemented methods. The unoptimized variants of the cube shadow maps and the Dual–Paraboloid shadow mapping (DPSM) show the worst results. In this approach, for every pass, all the geometry is rendered. Naturally, six render passes of the cube shadow maps lead into the highest frame time.

The basic optimization technique provided the bounding object frustum culling against the view frustum, the cube shadow maps frustum and the clipping plane for paraboloids. In this case, the same amount of geometry is rendered in both approaches. The overhead for increased number of the render passes for the cube shadow maps had no effect on an overall time for a single frame and thus the resulting frame times are similar.

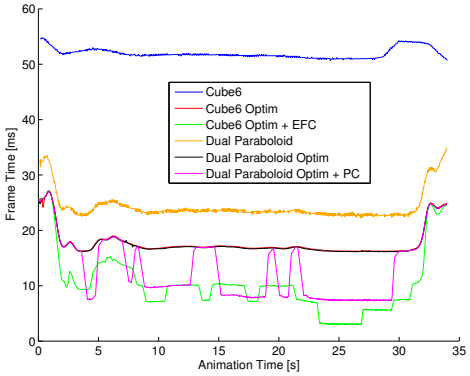

Figure 8: Frame times for the walkthrough of the scene for all implemented methods.

The cube shadow maps approach exhibits the best result with the effective cube face frustum culling - *EFC* (see Section 4.1). The plot shown in Figure 8 shows that the DPSM increased the performance only by skipping one paraboloid wherever appropriate (using plane clipping - *PC*). Otherwise, all of the geometry had to be rendered in two passes. The cube shadow maps approach can skip up to five render passes and thus it achieved the best results (e.g. in 25th second of the walkthrough). The frame time in the DPSM depends mainly on the amount of rendered geometry and also the amount of geometry in the given hemisphere. As can be seen in the plot, the DPSM saved only 50% of the computation time when rendering the scene only for one side. However, the cube shadow maps saved up to 83% of the performance. Furthermore, Figure 9 shows that the DPSM uses only one paraboloid most of the time and also that the cube shadow map rarely performed all six passes. This is mainly because the light source lied outside the camera view frustum.

5.2 Timings of Render Passes

Since the shadow mapping algorithm renders shadows in two passes, we investigated frame times for the passes for all implemented methods. The time for final shadow rendering showed to be equivalent for all methods, because it mainly depends on number of rendered geometry. Here, the view frustum culling was employed. The most noticeable differences were in times for generation of the shadow map.

As shown in Figure 10, the methods without any optimization had to render all the geometry six times in case of the cube shadow maps (blue) or two times in case of

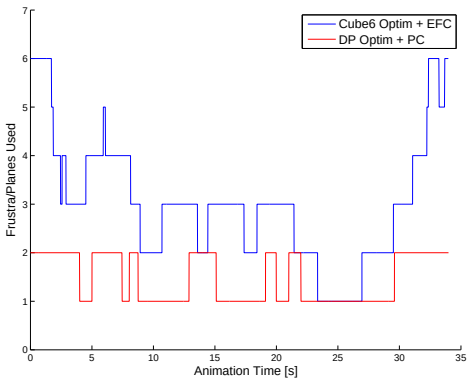

Figure 9: The plot shows the number of processed cube faces (blue) and the number of rendered paraboloid sides (red).

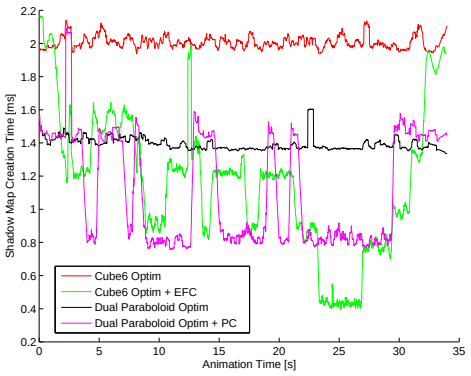

Figure 10: Evaluation of the times which all methods spent on the shadow map generation. For better illustration, unoptimized methods are not visible, because they had very poor results as compared to optimized techniques.

the DPSM algorithm (yellow). There are also some differences between methods where a frustum and plane culling is applied. The DPSM algorithm was faster compared to the cube shadow maps. An overall amount of rendered geometry was equivalent in both cases so there seems to be some additional overhead in the cube shadow maps technique.

Generally, the DPSM algorithm was faster when only one paraboloid was processed. The cube shadow map technique reached the similar times when only 2 faces were processed. The plot in Figure 10 also shows that in time 25 s, the cube shadow maps technique achieved the best results. In this case, only one face was processed which is mainly based on the position of a light sources relative to a camera (see Figure 11).

5.3 Effect of Shadow Map Resolution

We also investigated how the shadow map resolution affects the frame rate. In Table 2 and Table 3 you can see the results for various shadow map sizes. As you can see, the optimization techniques brought an increase in frame rate.

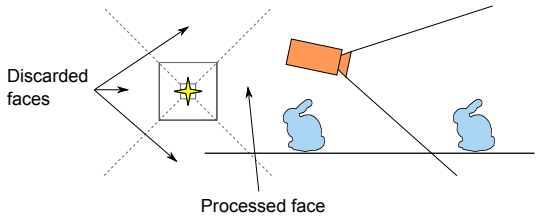

Figure 11: An illustration of the situation when only one face is processed during shadow map generation pass. Figure shows that only one cube face frustum intersects with the camera view frustum.

Considering shadow map as a texture storing single 32-bit value per texel, memory consumption of the cube shadow maps was from 24MB (1024×1024) to 384MB (4096×4096). Whereas the Dual–Paraboloid mapping approach uses one third of memory compared to the cube shadow maps (8MB to 128MB), it is more computationally intensive. Utilizing efficient frustum culling methods, we can save computation time by reducing number of the render passes and size of the geometry data, which also reduces memory utilization (less number of values stored due to frustum culling).

When taking 1024^2 resolution of shadow map as 100% performance for each method, switching to 2048^2 causes performance drop off only by 6.54% in average, but greatly increases shadow quality. Choosing 4096^2 resolution for shadow map takes 25.76% performance penalty in average.

Image quality of the result of Dual–Paraboloid mapping technique depends on the geometry of the occluding object. As described in [BAS02, OBM06], the Dual–Paraboloid mapping causes low-polygonal casters to produce incorrect shadows. Increasing shadow map resolution does improve shadow quality, but still can not match the quality of details achieved by the cube shadow maps approach (see Figure 12).

	1024^2	2048^2	4096^2
Cube6	75.71	70.04	47.9
Cube6Optim	150.43	116.76	64.04
Cube6Opt+EFC	188.71	151.67	89.68
DP	167.95	146.62	97.52
DPOptim	207.24	178.67	109.4
DPOptim+PC	208.15	180.24	110.95

Table 2: FPS of low-poly scene (600K vertices)

5.4 Position of a Light Source Relative to Geometry

We also performed an experiment where we focused on position of a light source relative to the geometry. This experiment was inspired by techniques for computation of interactive global illumination [RGK+08].

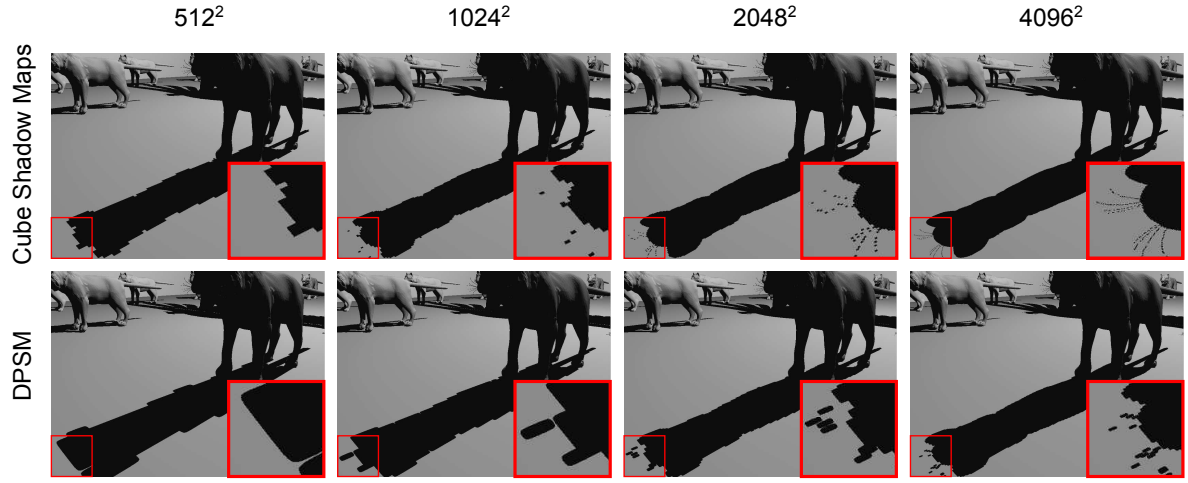

Figure 12: Figure shows how the shadow map resolution influences the shadow quality. Since a single paraboloid covers one hemisphere, one shadow map texel is projected on the large area in the scene (as compared to the cube shadow maps). This leads to worse quality of shadows.

	1024^2	2048^2	4096^2
Cube6	19.11	18.38	16.21
Cube6Optim	57.15	51.23	36.50
Cube6Opt+EFC	127.47	114.21	83.38
DP	41.50	39.74	33.17
DPOptim	57.47	54.32	42.85
DPOptim+PC	90.56	86.08	69.58

Table 3: FPS of high-poly scene (3M vertices)

In this case, *Virtual Point Lights* (VPLs) are generated on the surface to approximate indirect lighting. The reflected light is scattered into all directions. Therefore, some method is required to handle shadows from the reflected light. For this purpose, all the geometry data is positioned into one hemisphere relative to the light source. When the geometry is distributed around the light sources, it is useful to use the cube shadow maps technique, because it has better optimization strategy and it can easily manage the number of processed cube map faces. However, when we need to render only one hemisphere, the DPSM algorithm is more sufficient.

We measured times for generation of the shadow map in both of the presented techniques. Ritschel et al. [RGK+08] employed the Dual–Paraboloid mapping algorithm in their approach. They generated shadow maps for multiple VPLs (256 and more) from simplified geometry. We compared timings for the DPSM and the cube map technique.

In Figure 13, it can be seen that the DPSM algorithm is approximately two times faster than the cube shadow maps approach. The results are similar for various levels of the scene complexity. The Dual–Paraboloid mapping algorithm can be used despite its worse accuracy, because indirect lighting produces low-frequency shadows. In this case, the artifacts are blurred.

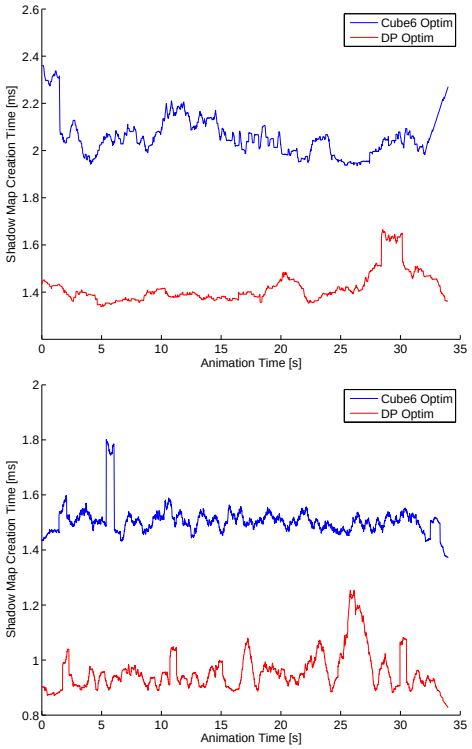

Figure 13: Figure illustrates times that the methods of interest spent on generation of the shadow map. In this case, the geometry is placed into one direction from the light source. The scene was represented by points only: 3 millions points (Top) and 100k points (Bottom).

6 CONCLUSION AND DISCUSSION

The goal of the work presented in this paper was to investigate the shadow mapping algorithm and techniques based on this algorithm as well as their capabilities to render shadows cast from point light sources. We ex-

amined two techniques that are based on the shadow mapping algorithm. The cube shadow maps approach exploits the traditional shadow mapping algorithm and renders the shadow map on cube faces. The Dual–Paraboloid shadow mapping uses nonlinear parameterization to render one hemisphere in one render pass.

The initial assumption was that multiple render passes performed by the cube shadow maps technique should be very time consuming process. The result of the measurement is that an unoptimized version of the cube shadow maps exhibits the worst performance of the examined algorithms. When a simple optimization technique was used significantly increased performance was reached, in fact, the best of the examined algorithms. The performance and the visual quality of the cube shadow maps is better compared to the Dual–Paraboloid algorithm. However, the Dual–Paraboloid algorithm produces better results if we consider the specific position of a light source related to a geometry, e.g., when computing illumination using VPLs.

Future work includes the complexity study that will improve the quality of measurements but since the timings depend mainly on the rendered geometry, however, as the complexity class is similar for all approaches, no significant differences are expected. It might be interesting to compare the implementation using the current hardware capabilities, e.g. CUDA. Evaluation of visual quality of the presented methods and their ability to deal with the aliasing problem in the shadow mapping algorithm is also subject of future work.

ACKNOWLEDGMENT

This work was supported by the European Regional Development Fund in the IT4Innovations Centre of Excellence project CZ.1.05/1.1.00/02.0070 and the Artemis JU project R3-COP, grant no. 100233.

REFERENCES

[AM00] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. *J. Graph. Tools*, 5(1):9–22, January 2000.

[BAS02] Stefan Brabec, Thomas Annen, and Hans-Peter Seidel. Shadow mapping for hemispherical and omnidirectional light sources. In *Proceedings of Computer Graphics International*, pages 397–408, 2002.

[Cro77] Franklin C. Crow. Shadow algorithms for computer graphics. *SIGGRAPH Comput. Graph.*, 11(2):242–248, 1977.

[CVM11] Marcel Stockli Contreras, Alberto José Ramírez Valadez, and Alejandro Jiménez Martínez. Dual sphere-unfolding method for single pass omni-directional shadow mapping. In *ACM SIGGRAPH 2011 Posters*, SIGGRAPH '11, pages 69:1–69:1, New York, NY, USA, 2011. ACM.

[Eng08] Wolfgang Engel, editor. *Programming Vertex, Geometry, and Pixel Shaders*. Charles River Media; 2 edition, 2008.

[Fer05] Randima Fernando. Percentage-closer soft shadows. In *ACM SIGGRAPH 2005 Sketches*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[Ger04] Philipp Gerasimov. Omnidirectional shadow mapping. In Randima Fernando, editor, *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*, pages 193–203. Addison Wesley, 2004.

[Gru07] Holger Gruen. Performance profiling with amd gpu tools: A case study. AMD Sponsored Session, GDC, March 2007.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, dec 2003.

[HWL+11] Tze-Yiu Ho, Liang Wan, Chi-Sing Leung, Ping-Man Lam, and Tien-Tsin Wong. Unicube for dynamic environment mapping. *IEEE Transactions on Visualization and Computer Graphics*, 17(1):51–63, January 2011.

[KN05] Gary King and William Newhall. Efficient omnidirectional shadow maps. In Wolfgang Engle, editor, *ShaderX3: Advanced Rendering with DirectX and OpenGL*, pages 435–448. Charles River Media, Hingham, MA, 2005.

[LWGM04] Brandon Lloyd, Jeremy Wendt, Naga Govindaraju, and Dinesh Manocha. Cc shadow volumes. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, pages 146–, New York, NY, USA, 2004. ACM.

[MGR+05] Victor Moya, Carlos Gonzalez, Jordi Roca, Agustin Fernandez, and Roger Espasa. Shader performance analysis on a modern gpu architecture. In *Proceedings of the 38th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 38, pages 355–364, Washington, DC, USA, 2005. IEEE Computer Society.

[OBM06] Brian Osman, Mike Bukowski, and Chris McEvoy. Practical implementation of dual paraboloid shadow maps. In *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 103–106. ACM, 2006.

[RGK+08] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–8. ACM, 2008.

[SD02] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 557–562. ACM, 2002.

[SWP10] Daniel Scherzer, Michael Wimmer, and Werner Purgathofer. A survey of real-time hard shadow mapping methods. In *EUROGRAPHICS 2010 State of the Art Reports*, 2010.

[VBGP09] Forest Vincent, Loïc Barthe, Gael Guennebaud, and Mathias Paulin. Soft Textured Shadow Volume. *Computer Graphics Forum*, 28(4):1111–1120, 2009.

[VNHZ11] Juraj Vanek, Jan Navrátil, Adam Herout, and Pavel Zemčík. High-quality shadows with improved paraboloid mapping. In *Proceedings of the 7th international conference on Advances in visual computing - Volume Part I*, ISVC'11, pages 421–430, Berlin, Heidelberg, 2011. Springer-Verlag.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274, 1978.

[WSP04] M. Wimmer, D. Scherzer, and W. Purgathofer. Light space perspective shadow maps. *In the Eurographics Symposium on Rendering*, 2004.