Real-Time Shadows for Large-Scale Geospatial Visualization

Michaela Niedermayer [*]
Supervised by: Jürgen Waser, Daniel Cornel

VRVis Zentrum für Virtual Reality und Visualisierung Forschungs- GmbH
Vienna / Austria

Abstract

Floods are often catastrophic and can cause enormous damage. Simulations are used to predict risks and respond to them early, like in the decision-support system Visdom. A true-to-life representation of the entire scene is important, as the visualization of the results must be understandable also to non-experts, such as decision-makers or the general public. What is missing so far are shadows, although they are particularly well suited to recognize relations of objects to each other. This paper covers the implementation of shadows in Visdom, to increase the realism of the scene. This is very complex for scenes on city or even country scale and requires a lot of own strategies, as there are no ready-made solutions. We present an adaptation of cascaded shadow maps for our purposes, as well as a variety of improvements to increase the shadow quality in our application. The result of this work is a flexible visualization of soft shadows for a variety of different-sized scenes in real time, which increase the realism and spatial perception.

Keywords: Real-Time, Shadow Maps, Cascaded Shadow Maps, Soft Shadows

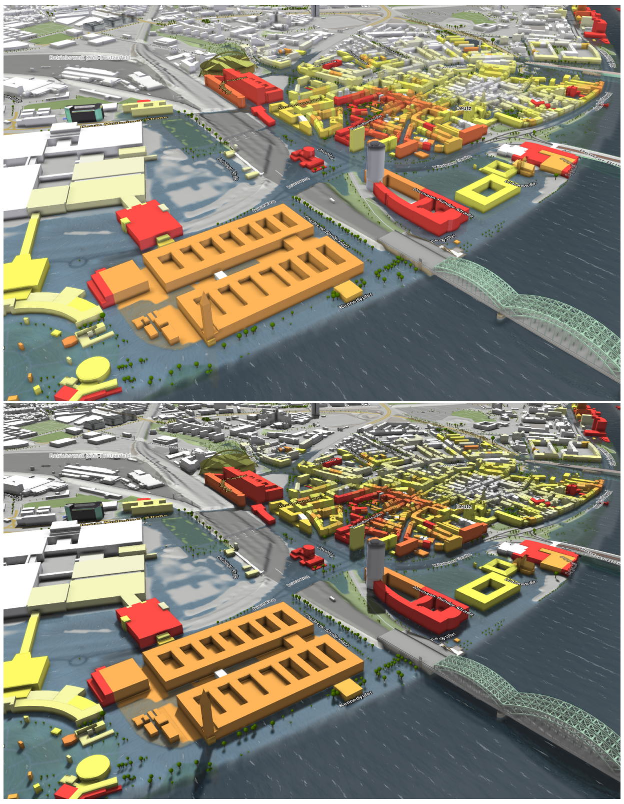


Figure 1: A scene view from the city of Cologne without (top) and with (bottom) shadows.

1 Introduction

Floods are dangerous and often unpredictable events that can occur due to different causes, including heavy rainfall, snow melting, river overtoppings, dike breaches and many more. The financial damages of these natural events amount to several million Euros. Visdom is a software framework used for decision support in flood management [1]. It helps to simulate and visualize floods over a specific period in different cities or countries (e.g., for Austria or for the city of Cologne, Germany) and allows flood managers, for example, to investigate which buildings would be in danger in various scenarios. These simulation results are mapped to the geospatial domain in a 3D visualization. The visualization should not be too complex that important results can be easily distinguished, but it should be true-to-life that also non-experts are able to understand it quickly. The visualization has already a high degree of realism in many areas of the application, but not in illumination. Shadows are missing in Visdom, although they would be well suited to get knowledge about the relations of the objects to each other. Shadows are the best solution to determine the relative location of objects. They anchor objects in the scene, indicate the position of light sources and give a greater sense of depth of the scene. This is demonstrated in Figure 1 with our own results. The aim of this work is the implementation of high-quality shadows without artifacts in the 3D visualization to increase the realism and spatial perception for large-scale scenes, without reducing the performance significantly. It is a complex task to implement shadows for such large scenes. There exist no ready-made solutions that could be implemented without adaptions to our use case. We implement shadows with an adapted cascaded shadow maps (CSM) algorithm and further improvements to increase the quality of the shadows. The first step is the implementation of standard

[*]michaela.niedermayer@aon.at

shadow mapping that has some known limitations, which lead to artifacts in diverse situations. An issue of shadow mapping is aliasing (perspective and projection aliasing), which is an artifact caused by under-sampling due to the low resolution of the depth map near the observer's eye. Another problem is incorrect self-shadowing of objects. It is caused by the quantization of z-buffer surfaces when the transformed point is not lying right on the surface of which it is a part, but above or below [12]. The suggested solutions for artifacts in synthetic cases do not always work in real-world scenarios, so robust solutions have to be adapted or developed. In addition, some artifacts stem from faults in the geospatial data gathered from different sources over which we have no control. Further, the succeeding change to cascaded shadow maps is difficult, as it is normally not used for orthographic projection. These are the challenging problems, since existing approaches do not tackle these issues and, therefore, own solutions need to be developed.

2 State of the Art

In the following subsections the state of the art of shadow mapping, cascaded shadow maps and soft shadow techniques are explained.

2.1 Shadow Mapping

The two best-known algorithms for shadows are shadow volumes [4] and shadow mapping. As shadow mapping is significantly faster than shadow volumes in many cases, it is state of the art for real-time applications. Another obvious choice for shadow computation is ray casting, but with a single ray per fragment it results in hard shadows and needs more implementation effort than shadow mapping. For soft shadows more ray tracing would be needed, which is, in spite of modern hardware support, in world space much slower than an image-space method. Shadow Mapping [12] is a technique done in image space with two render passes. In the first render pass (z-buffer pass), a depth map from the point of view of the light source, i.e., in light space, is created. In the second render pass, the scene is rendered from the observer's eye, i.e., the camera of the scene. During rendering, the fragments of shadow receivers are mapped to the light space, the depth values of the same point of interest of both passes are then compared to each other. If the fragment depth from the second render pass is larger than from the first, the fragment is in shadow and gets shaded accordingly. As shadow mapping is done in image space, some limitations must be observed [8], e.g., that all objects that cast a shadow in the viewed scene are part of the depth map. In addition to the aliasing problems, shadow mapping is view-dependent and it is problematic for omnidirectional lights. Besides the major advantages of shadow mapping, which are efficiency and speed, it enables self-shadowing, is independent of scene complexity and the depth map can sometimes be reused.

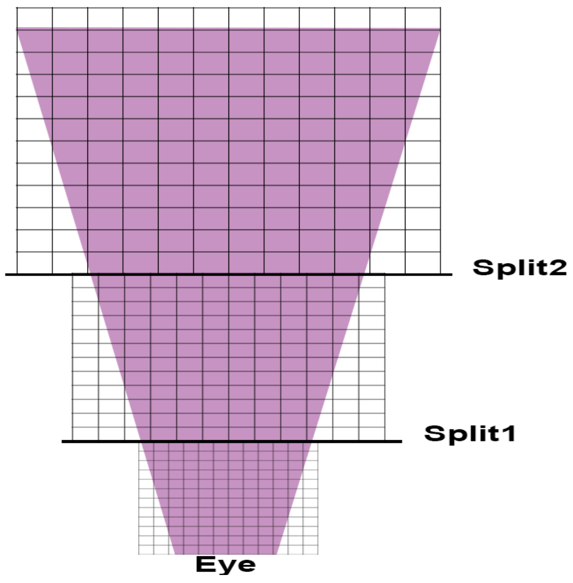


Figure 2: Cascaded Shadow Mapping splits.

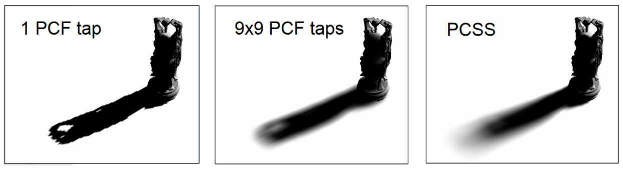


Figure 3: A shadow of a statue with no use of PCF (left), PCF with a 9×9 kernel size (middle) and PCSS (right). Images are taken from [3].

2.2 Cascaded Shadow Maps

Cascaded shadow maps (CSM) can be used to address the problem of producing high-quality shadows in large environments. It is a partitioning method where the view frustum is split into different parts and a depth map is created for each of them (example in Figure 2). Through the separate depth maps, the resolution in each subfrustum is higher and aliasing artifacts are reduced, which improves the shadow quality. In which shadow map the lookup is taken depends on the fragment's depth. CSM is often seen as an discretization of *Perspective Shadow Maps* [11], which has the idea to wrap the light frustum to exactly coincide with the view frustum [5].

2.3 Soft Shadows

Traditional shadow mapping produces hard shadows, which causes sharp shadow edges and a lot of aliasing. *Percentage closer filtering (PCF)* [9] provides an anti-aliasing solution where the results of the depth map lookup are sampled around the projected point. The binary results are filtered in a given kernel and the data average is calculated, so the new shadow value is in the range between zero and one instead of exactly zero or one. The bigger the size of the PCF kernel, the softer are the resulting shadows

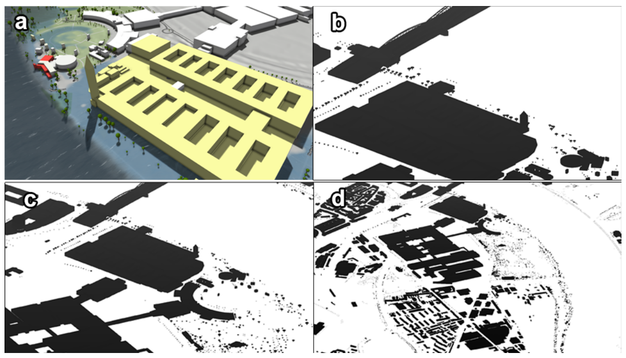


Figure 4: (a) Camera view: rendered scene. (b-d) Light view: depth maps of three different cascades of image (a).

(example in Figure 3). In another soft shadow technique, *percentage closer soft shadows (PCSS)* [7], the filter size varies intelligently to achieve a plausible degree of softness as well as hard contact shadows. In the first step of PCSS (the blocker search), the depths that are closer to the light source than to the receivers are searched and averaged. Then the penumbra width is estimated with the following equation:

$$w_{\text{Penumbra}} = (d_{\text{Receiver}} - d_{\text{Blocker}}) \cdot w_{\text{Light}} / d_{\text{Blocker}}. \quad (1)$$

In the last step, a PCF filtering is performed using a kernel size proportional to the penumbra estimation. PCSS is generating perceptually accurate soft shadows without needing additional geometry (example in Figure 3). These are just two out of many possible approaches for anti-aliasing. Another well-known method is *Variance Shadow Maps (VSM)* where the mean of a distribution of depths is stored to compute the variance over a region [6]. It is not implemented, as the quality of PCSS with PCF filtering is preferred.

3 Methodology

In this section, the theoretical background of the implementation is explained. The goal is to implement a shadow mapping algorithm (subsection 3.1) to render good looking shadows in large environments. The implementation includes many methods to calculate the necessary values automatically, suitable for the actual scene and view. In the following subsections, the basic shadow mapping, cascaded shadow maps and further improvements are described.

3.1 Depth Map and Shadow Mapping

The first step is to create a single depth map for basic shadow mapping. The generation of the depth map is the first render pass of the shadow mapping algorithm. The created texture stores the scene from the light source's "view", which is called *focusing*. For shadow mapping, a directional light is used. This kind of light is comparable with a sun and has therefore no position in the scene, but a

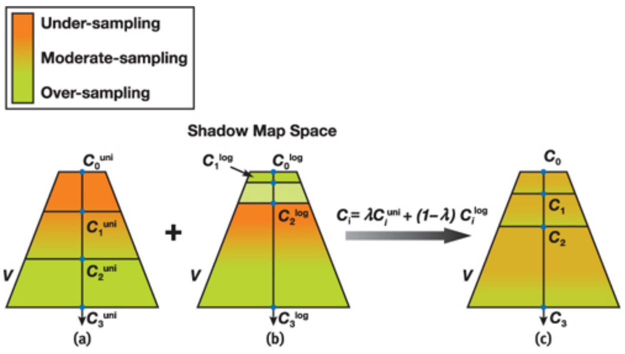


Figure 5: The practical split scheme (right) consists of the uniform split scheme (left) and the logarithmic split scheme (middle). Image from [10].

direction in which the light is cast. The depth map includes all kinds of meshes (buildings, bridges, etc.) and trees. The terrain is not included, which means the terrain does not cast shadows. A result of a depth map from the application can be seen in Figure 4, already divided in the cascades, which are explained in subsection 3.2.

After generating the depth map, the second render pass can be done. This is the normal rendering but with an additional shadow calculation method. The first step in the method is the transformation of each visible surface position to the light space and subsequent projection to the image space of the corresponding cascade's shadow map. The normalized distance to the light source is stored as the current depth. As the depth map is also in the range [0,1], the closest depth from the light's point of view can now be determined with a lookup to the shadow map. The last step is to compare the depths. If the current depth is larger than the closest depth the shadow is one.

3.2 Cascaded Shadow Mapping

Cascaded Shadow Mapping is a technique where multiple depth maps are used to overcome perspective aliasing. In our application, one to five cascades are possible. As described in subsection 2.2, the view frustum is split into different parts. The splits are located along the z-axis in view space. For the calculation of the split positions, a scheme, called *practical split scheme*, is applied, which provides a moderate sampling rate in the whole view frustum. The split scheme, i.e., in Figure 5, consists of two parts, which are the uniform part and the logarithmic part.

$$C_{\text{i}} = l(n(f/n)^{\text{i/m}}) + (1-l)(n + (f-n)(i/m)) \quad (2)$$

l is the amount in percent of how much the logarithmic part is taken. The equation of the split position for cascade i (from overall m cascades) is Equation 2, at which n is the near plane and f is the far plane.

The next step in the cascaded shadow mapping algorithm is to compute an orthographic projection for each subfrustum. For simplicity and reusability of existing functionality,

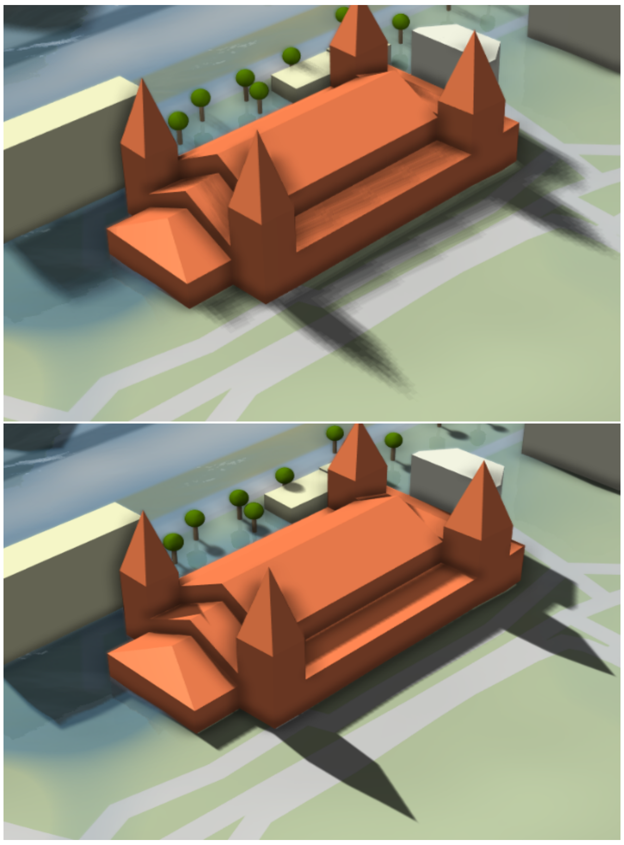


Figure 6: (Top) A screenshot of a small part of the scene without cascaded shadow mapping. (Bottom) The same scene when four cascades are used.

a *cascade camera* is created for each split part to store values like near/far plane, view matrix, projection matrix, etc. Furthermore, an additional camera is created for each cascade which faces the cascade from the direction of the directional light source, in the following called a *shadow camera*. All of these cameras are updated every time the original scene camera is changing. All the steps done for standard shadow mapping are now done for each cascade. From a list of cascade splits along the view direction, it can be decided for each visible surface position to which cascade and therefore shadow map it corresponds. In Figure 6, you can see the difference of the scene when using standard shadow mapping (top) and cascaded shadow mapping with four cascades (bottom).

3.3 Soft Shadows Improvements

One of two soft shadow techniques can be chosen in the application, the *Percentage Closer Filtering (PCF)* and the *Percentage Closer Soft Shadows (PCSS)*. For PCF, the size of the filter kernel can be selected in the application during run time. It is also possible to select a size of 1×1-filter kernel, which means no PCF filtering is done. The filtering itself is done as described in the previous subsection 2.3. For PCSS, two parameters can be set in the application at run time, namely the light size and the number of samples for the included PCF algorithm. First, the blocker search is done with a fixed number of the samples. After this, the penumbra size is calculated with the parameters set by the user. The last step is the PCF filtering with included stochastic sampling. Through the combination with stochastic sampling, the texels are less visible.

3.4 Further Improvements

The first improvement is a blending between the cascades, since it is sometimes visible where the next cascade begins. For all the fragments in the blend area of a user-defined size, a second shadow value is calculated. Both shadow values are linearly combined to get the final shadow result. Another improvement, which generates better shadow results, is a fade out of the shadow. Within a user-defined *fade out range*, the shadow intensity is gradually reduced and beyond that range, no shadows are drawn in order to reduce the cascade size. This is a big improvement for the shadow quality, since just the shadow range and fade out area is rendered in the depth maps instead of the whole scene.

4 Implementation

The whole shadow mapping algorithm with methods and improvements explained in section 3 is implemented in Visdom with the graphics API OpenGL. The code is written in the programming language C++.

4.1 Setup

There exist three important parameters (z-near, z-far, orthozoom factor) needed for the orthographic projection matrix. The values of these parameters vary for a good-looking shadow, depending on the actual view. Since the view and therefore the amount of rendered polygons changes a lot when zooming in and out, these three values are calculated automatically. Z-near and z-far of the shadow camera of each cascade are the first two values computed. With the splits of the cascades, we get the z-near and z-far values for the cascade cameras. These values are required to get the frustum corners of the actual cascade. After multiplying the frustum corners with the view matrix from the shadow camera of this cascade, we get the coordinates in view space and can store the minimum and maximum z-value.

The orthographic projection in the application is not a standard OpenGL projection matrix, but an reversed one, which means that it maps the far plane to zero instead of one [2]. The orthozoom factor is a value, which does not exist in a normal orthographic projection. This value is used in the application to scale the whole scene before the projection, to map the scene to a much smaller area. For the calculation of the factor, first the bounding box of the scene camera of one cascade is used, to get the intersection volume. In the

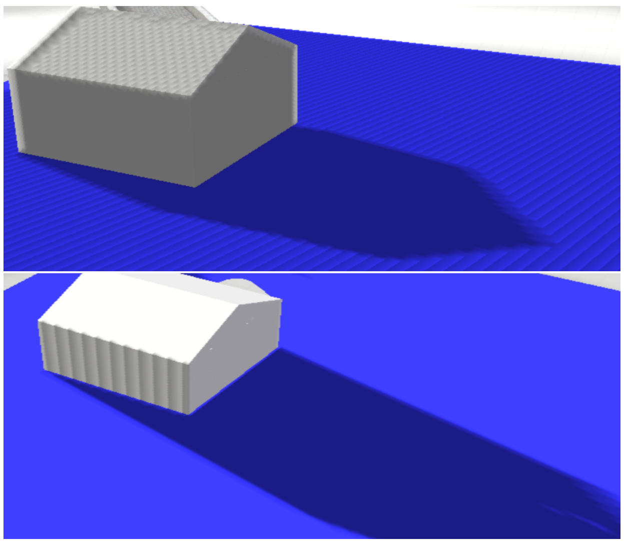


Figure 7: (Top) Example of incorrect self-shadowing. (Bottom) The incorrect self-shadowing is reduced, but shadow acne appears on the backface of the building.

next step, the intersection volume is projected onto the near plane of the scene camera. Further, a new bounding box around the points on the plane is built, to get the width and the height of this bounding box. The width and the height are each divided by two. The maximum of these two values is then set as the orthozoom factor.

4.2 Shadow Pass

The program required to generate the depth map consists of a vertex and a fragment shader. A framebuffer is used to store the rendered result in a texture. First, the 2D texture is created with a depth render target, as no color buffer is needed in this render pass. This texture is then attached as the framebuffer's depth buffer. The render logic and the vertex shader are the same as for rendering meshes. The fragment shader is simpler, because it only stores the depth values. In the vertex shader, the vertex coordinates are transformed from world space to the cascade's corresponding light space before projection. An important step before rendering the depth map is to change the viewport of the OpenGL API to the light viewport. Further, the OpenGL API offers a function to cull faces for the rendering, so they are not drawn in the application. For surface rendering, it is usual to cull the back faces, which means much less triangles need to be rendered. For the depth map creation, the value is set to front face culling. If the front face is culled, the shadow map stores the depth value of the back face instead of the front face. It does not change anything for solid objects if the depth is taken from the front or back face, as it does not matter if shadows are inside the objects.

Through the difference in the depth, there are no longer artifacts on the roofs of the buildings. Furthermore, the incorrect self-shadowing explained in subsection 2.1 is not visible any longer, because the z-value of the light is now always larger than the z-value of the eye, if the shaded point is not in shadow. An example of a small scene with and without incorrect self-shadowing can be seen in Figure 7. The only exception are objects which do not have any depth itself (e.g., a plane), as the front face culling would not change anything regarding the depth. After the use of front face culling, there are some artifacts on the back faces of the meshes, named shadow acne, which can be seen in Figure 7 in the image below. It occurs when the shadow receiver is orthogonal to the shadow map plane (i.e., projection aliasing). A solution for this problem is implemented by calculating the dot product of the directional light direction and the normal of the shaded point. If the result is larger than or equal to zero the fragment is in shadow.

4.3 Shading Pass

This render pass is the common shading pass with an additional shadow calculation. The computation is done in the fragment shader. Shadow camera parameters such as light-space matrices and near and far bounds are provided with a uniform buffer object. In the shader, the fragment's depth is used to find out in which cascade the shadow lookup is done. To compare the fragment with the corresponding fragment in the depth map, a transformation to the light screen space is required. The fragment's world-space position is transformed to the shadow map's image space. The calculation and shading is done as described in subsection 3.1. This algorithm returns hard shadows, which means the resulting shadow is a binary value and causes hard lines between the dark and lit regions. The soft shadow algorithms PCF and PCSS are used to smooth the transition and to get other shadow values between zero and one.

In the PCF algorithm, the user can select specified filter sizes between 1×1 and 9×9. In two for-loops, all shadow values of the neighbors of the actual fragment are summed up and averaged. The PCSS algorithm can be divided into three parts. In the first part, the blocker search, a fixed number of blocker samples are used. We set this value to five, by default. With the light size, the near plane of the cascade, in which the fragment is lying, and the current depth, a search width is defined. This search width combined with stochastic sampling gives an offset, when the closest depth is calculated. If the current depth is larger than the closest depth, a new blocker is found. The result is the sum of the blocker closest depth values divided by the number of found blockers. After the penumbra size calculation with the Equation 1, a PCF filtering is done. The user can set the number of the PCF samples for PCSS. The offset for the calculation of the current depth is the penumbra size combined with stochastic sampling. For every sample where the current depth is larger than the closest depth, an amount of one is added to the sum, which is at the end divided by the number of samples. The result is the amount of shadow which is then multiplied with the diffuse and specular colors.

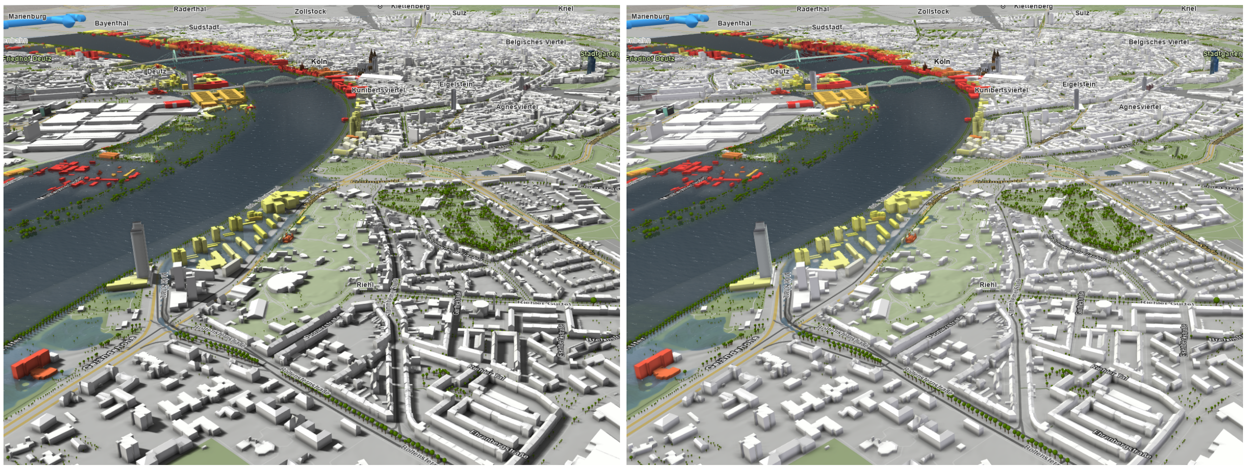


Figure 8: (Left) Shadow range of 5 km and fade out range of 500 m. (Right) No shadows.

5 Results and Discussion

In this section, the results of the implemented algorithm are shown and explained. First, the visual results are shown with short descriptions about what can be exactly seen in the resulting images. After this, the performance results are presented which are measured with the same camera settings as in Figure 8. In the last subsection, which is the discussion, a critical review of the results is done.

5.1 Visual Results

The results show the implemented shadow mapping algorithm in the city of Cologne, Germany. This city provides a good opportunity to show the results, as it is a big scene with high buildings, a river, detailed bridges, trees and some special buildings, e.g., the dome. The shadows make the scene much more realistic, especially in a direct comparison. Through cascaded shadow mapping and a limited shadow range, the shadow quality increases a lot. A result of a big scene with and without shadows can be seen in Figure 8, where the shadows are restricted to a range of 5 kilometers. Shadows are cast by trees and various kinds of meshes. That includes objects that are dynamically added at run time, e.g., walls and sandbag barriers, which are used to save endangered buildings from the floods. As soon as they are added to the scene, the shadow appears. The terrain receives shadows, but does not cast shadows itself. This could be an add-on in future work.

The algorithm shows objects with little details very accurately, e.g., one of the bridges which can be seen in Figure 9 on the image above. It is visible that the bridge is built out of many thin metal bars. Further, you can see in this figure the shadow of the bridge in the water twice. This happens as one shadow is on the ground of the water, and the other one is on the surface of the water. In Figure 9, in the image below, trees with their shadows are shown. In the figure a dynamic floodwall can be seen, which is inserted next to the river to protect the buildings around it. Beside this, the figure shows again the double shadow in the water area.

5.2 Performance Results

The computer used for testing the implementation has an integrated Intel Core i5-4690K processor with 3.50GHz and a memory of 32 GB RAM. As video card, the NVIDIA GeForce GTX 1070 is used. The operating system is Microsoft Windows 10 x64. The elapsed time on the GPU is measured with a pipeline statistics query from the OpenGL API. Performance tests are done for both render passes. The measured time of the first render pass includes the change of the viewport and culling, the binding of the program where all parameters are set and the rendering of the depth map itself. In all render classes (terrain, mesh and tree), the performance results of the shading pass include just the drawing of the elements, which is the actual rendering.

An interesting result shows the difference between the use of one and five cascades. This calculation includes the depth map generation as well as the shading pass for the meshes and for the terrain. As can be seen in Figure 10 at the top, there is not a big difference in the elapsed time between the amount of cascades for the second pass, but in the shadow map creation the value for five cascades is more than four times as high as for just one cascade. This is expected, as the scene geometry has to be rendered four more times. The diagram, shown in Figure 10 at the bottom, shows the elapsed time (y-axis) of the mesh, terrain and tree renderers (y-axis) when different soft shading techniques are used. The best looking results are gotten with the use of the PCF technique with a 9×9 filter size, but as can be seen in the figure the elapsed time to render this, is on average much higher than for the size 1×1 or for the other techniques. There is just a minimal difference between PCF 1×1 filter and PCSS with four samples, although a PCF size of 1×1 means there is no soft shading done. With increasing size of the samples in PCSS, also the elapsed time is increasing, but in proportion the raise is very low.

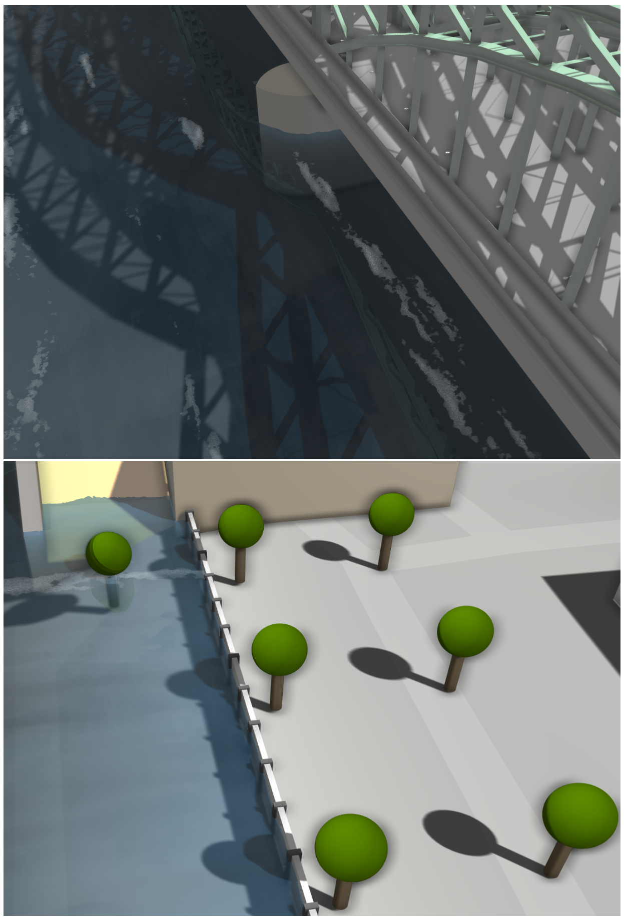


Figure 9: (Top) A bridge with many shadow details. (Bottom) Shadow of trees and double shadow in the water.

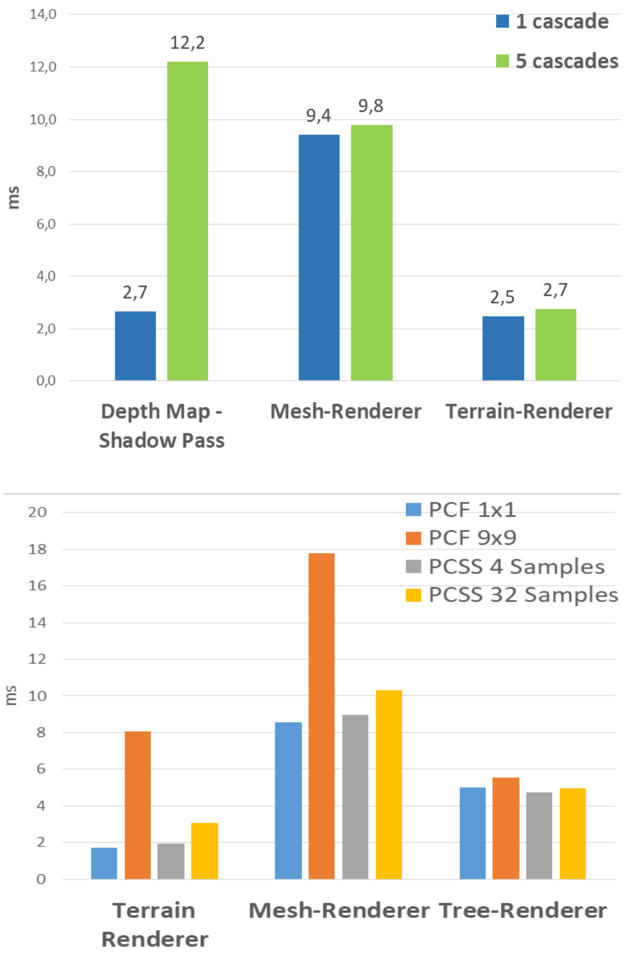


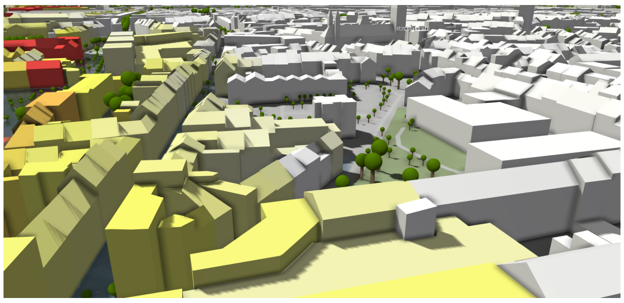
Figure 10: (Top) A diagram, which shows the different performance for using one or five cascades. The diagram includes both render passes. (Bottom) A diagram, which shows the performance for the use of different soft shading algorithms with different filter sizes and samples, measured in the shading pass for the terrain, the meshes and trees.

5.3 Discussion

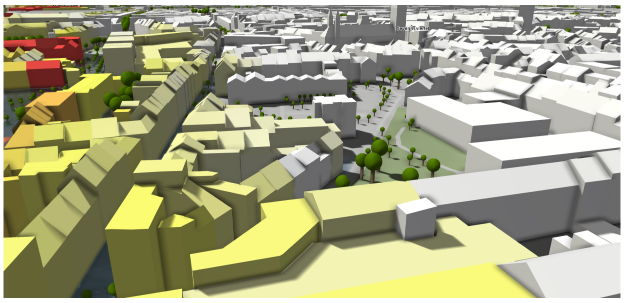
A difficult point in the shadow quality is that it depends much on the direction of the scene camera. When a scene is viewed top-down, the viewport includes just a few objects, and therefore just a few objects need to be rendered in the depth map. However, if the scene is viewed horizontally, the viewport of the camera can contain the whole scene in worst case. This leads to the problem that the entire scene needs to be rendered into the depth map and the positions near the camera are extremely under-sampled. Many improvements for this problem are added to the implementation, which are CSM, a practical split scheme and at least a maximum range for the shadow. All of them are very helpful to overcome the problem, but it is still visible that the shadow quality in a horizontal view is worse than in a top-down view. An example of such a horizontally viewed scene can be seen in Figure 11a. In this view, CSM is already used, but staircases in the shadow are visible, as no shadow range was used and the depth map has a resolution of 1024, which is too small for this scene view. With the usage of a maximum shadow range or a higher resolution of the depth maps, the result is much better, i.e., in Figure 11b where a shadow map size of 4096 is used.

All mentioned improvements are helpful, but it is still visible that the shadow quality in a horizontal view is worse than in a top-down view. Although it is visible, the improvements decreased this artifact to a minimum and therefore it is not annoying any longer. To implement the feature of a maximum shadow range is helpful in many ways. It improves the shadow quality and a little bit the performance when terrain and meshes are rendered. Nevertheless, the improvement has a disadvantage, one has to accept. In a top-down view, it can happen for tall buildings that shadows are still visible on the roof, but not on the ground.

One of the two soft shading techniques should always be used when rendering shadows, at which PCSS has a better proportion between performance and shadow quality. PCF with a filter size of 9×9 is the best choice for screenshots, since it gives the best looking results, but as described in subsection 5.2 it has a too low performance to use it as default filtering technique.



(a)



(b)

Figure 11: A horizontally viewed scene. (a) A shadow map size of 1024, which causes artifacts. (b) Much better results with shadow map size of 4096.

6 Conclusion

This paper covers the implementation of real-time shadows for large-scale geospatial visualization. An adapted version of the cascaded shadow maps technique is used to reach a good quality of the shadows and at interactive frame rates. Several improvements helped to reach the desired quality and a good performance. They further removed occurring artifacts. Examples of these artifacts are the incorrect self-shadowing, fixed with the use of front face culling, perspective aliasing, fixed with the use of a specific split scheme in CSM, and a visible cascade transition, which is solved by blending between the cascades. As shadow mapping produces hard shadows, we integrated two soft shadow techniques, which are PCF and PCSS. The result is a flexible visualization of real-time shadows, which improve the realism of the scene. An add-on that should be done in future work is casting shadows of the terrain.

7 Acknowledgements

This work was enabled by the Competence Centre VRVis. VRVis is funded by BMVIT, BMDW, Styria, SFG and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (854174) which is managed by FFG.

References

[1] Visdom. `http://www.visdom.at/`.

[2] Depth precision. `http://dev.theomader.com/depth-precision/`, May 2015.

[3] Louis Bavoil. Advanced soft shadow mapping techniques. `http://developer.download.nvidia.com/presentations/2008/GDC/GDC08_SoftShadowMapping.pdf`, February 2008.

[4] Franklin C. Crow. Shadow algorithms for computer graphics. In *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '77, pages 242–248, New York, NY, USA, 1977. ACM.

[5] Rouslan Dimitrov. Cascaded shadow maps. *Developer Documentation, NVIDIA Corp*, 2007.

[6] William Donnelly and Andrew Lauritzen. Variance shadow maps. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, I3D '06, pages 161–165, New York, NY, USA, 2006. ACM.

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

[8] J. M. Hasenfratz, M. Lapierre, N. Holzschuch, F. Sillion, and Artis GRAVIR/IMAG-INRIA. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, 2003.

[9] Lu Liu and Shuangjiu Xiao. Real-time soft shadows for large-scale virtual environments. In *2011 International Conference on Multimedia Technology*, pages 5464–5467, July 2011.

[10] Hubert Nguyen. *Gpu Gems 3*. Addison-Wesley Professional, first edition, 2007.

[11] Marc Stamminger and George Drettakis. Perspective shadow maps. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 557–562, New York, NY, USA, 2002. ACM.

[12] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '78, pages 270–274, New York, NY, USA, 1978. ACM.