Resolution Matched Virtual Shadow Maps

Matěj Sakmary*
Supervised by: Jiří Bittner†

Department of Computer Graphics and Interaction
Czech Technical University in Prague
Prague / Czech Republic

Abstract

In this article, we present Resolution Matched Virtual Shadow Maps (RMVSMs), a method for rendering shadow maps with greatly improved quality and memory efficiency compared to traditional methods. We separate logical address space from its physical backing by splitting each shadow map cascade into a set of virtual pages. This achieves the appearance of a large contiguous memory without the need to reserve backing physical memory. For each frame, we find the set of visible pages by analyzing the depth buffer. Each visible page is assigned physical memory allocated from a designated memory pool. To efficiently fill visible pages with shadow map information, we utilize granular culling of the scene geometry. We show that our implementation is well suited for multiple light sources of various types, including directional lights, point lights, and spotlights. Further, we show that this technique scales to an arbitrary number of cascades as only a fraction of virtual pages are visible and need to be backed each frame. We can thus achieve any desired texel-to-pixel density at any distance with few wasted shadow texels.

Keywords: Shadows, Shadow map, Shadow Mapping, Virtual, Virtual textures, Memory Paging, Meshlets, Mesh Shading, Sparse, Culling, Real Time Shadows, Point Light Shadows

1 Introduction

Shadows are one of the most important aspects of a three-dimensional scene. By providing visual queues on shapes and the relative positions of objects, they define how the scene is perceived and interpreted. Calculating and storing visibility, which is at the core of each shadowing solution, remains a challenging problem. Even though recent advancements in hardware technology make ray tracing a promising direction, shadow mapping remains a standard and widely used technique for solving visibility queries.

Despite that, current shadow mapping techniques still suffer from artifacts caused by insufficient shadow map resolution. Increasing resolution to alleviate these issues quickly becomes impractical due to memory consumption and hardware limitations. These issues are further accentuated if multiple light sources are considered, each requiring one or more separate shadow maps. Even when we ignore the memory consumption issues, naively rendering all shadow maps becomes prohibitive due to the sheer cost of rasterizing the scene for each shadow map.

Figure 1: On the left undesirable effects caused by insufficient shadow map resolution which results in blocky looking shadows. On the right shadows produced by RMVSM.

Although many current techniques reduce issues caused by insufficient resolution, the fundamental compromise between shadow quality and memory cost remains. It is common for rendering engines to provide several options for shadows, each with their own set of trade-offs. For example, screen-space ray tracing [10] can be used in addition to shadow mapping implementations. This is done to achieve sharper and more detailed contact shadows. Specialized techniques are often used for different light sources, further complicating the implementation.

In this work, we provide a unified technique capable of efficiently storing and rendering multiple shadow maps for various light types. Our main contribution lies in extending previously described implementation of Virtual Shadow Maps [13] to allow for spot and point light sources. Additionally, we describe the specifics of geometry representation and culling, which were previously not explained in a lot of detail.

In Section 2, we discuss current solutions to the problems outlined above and their shortcomings. In Section 3, we provide an overview of the ideas our technique leverages. We also describe how these ideas generalize to multiple lights and light types in a scene. Following this, in

*matej.sakmary@gmail.com
†bittner@fel.cvut.cz

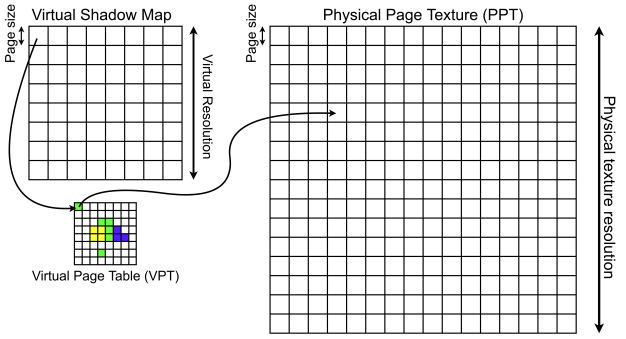

Figure 2: The virtualization layout adapted by our technique. Each entry stores metadata about the page, denoted by different colors in the image. In this example, white pages are not allocated, green pages are allocated and visible this frame, yellow pages are allocated but not visible this frame and finally blue pages are newly seen, and thus dirty and requiring a backing page from the PPT. The PPT itself is also denoted in the image. When a page is backed, the VPT entry stores coordinates to the corresponding page in the PPT.

Section 4 we give a high-level overview of how the geometry in our scene is represented and how this is leveraged allowing for highly efficient culling. Lastly, in Section 5, we provide and discuss the results we were able to achieve using our technique, and in Section 6 we conclude the article and offer promising directions for further research.

2 Previous work

The fundamental issue with shadow mapping are the artifacts introduced by insufficient shadow-map resolution. These occur when a pixel projected into the shadow map covers less than a texel in the shadow map. The same shadow map texel is used to determine visibility for multiple pixels, resulting in blocky looking shadows. To avoid this, at least one-to-one mapping between shadow map texels and pixels needs to be ensured. An example of the discussed issues can be seen in Figure 1.

Cascaded Shadow Maps (CSM) is a popular algorithm used to ensure a better distribution of shadow map texels for directional lights [3]. In CSM, the viewer's frustum is partitioned into smaller frusta, each fit with its own light space matrix and assigned a shadow map. The scene is then rendered to each shadow map using the cascade's frustum. This results in a denser shadow map being placed near the viewer where detail is most needed, while distant regions receive coarser shadows. *Sample Distribution Shadow Maps* (SDSM) improve CSM by using a per-frame analysis of the scene to calculate tight fit frusta, maximizing the efficiency of the shadow sample distribution regardless of scene content [7].

Adaptive Shadow Maps (ASM) take a different approach [4]. A hierarchical structure is introduced that is used to match the resolution of the shadow map with the resolution required by individual pixels. This structure is iteratively refined using an edge-detecting algorithm that ensures an efficient distribution of the texel shadow edges. A slightly different approach was proposed by *Resolution Matched Shadow Maps* (RMSM) [8]. RMSMs forgo the expensive iterative refinement step in favor of detecting the desired resolution of each pixel. Similarly to ASMs, a quadtree is used to partition the resulting shadow map.

Another approach improving ASM is called *Queried Shadow Maps* (QSM) [5]. By using occlusion queries present in GPU hardware and improving the heuristics of the refinement algorithm, QSMs offer much better performance in highly dynamic scenes. This is mainly because the improved refinement algorithm can be executed completely in each frame, which is not true for ASMs, which rely on caching to deliver the best performance.

Recently, *Efficient Virtual Shadow Maps for Many Lights* introduced the concept of virtualizing shadow maps by separating the shadow map itself from its physical storage [11]. This, in combination with efficient culling and clustered shading, allows for very high numbers of shadow-casting point lights.

Finally, *Virtual Shadow Maps* describes the complexities of implementing virtual shadow maps for directional lights [13]. Our article builds upon and extends this publication.

3 Shadow map virtualization

As already stated above, the main issue that shadowmapping faces is insufficient resolution. This can be solved by increasing the resolution of the shadow map; however, this quickly becomes impractical. A situation where a single pixel maps to too many shadow map texels should also be avoided. A one-to-one mapping between pixels and shadow map texels is the most desirable. Due to the non-uniformity of the area of the shadowmap covered by each pixel, a different resolution is required for different parts of the shadow map in order to achieve an ideal mapping.

These requirements are very similar to the ones demanded from ordinary textures. Textures usually have the highest resolution possible to allow for good-looking closeups, while relying on mipmaps for more distant views. To combat the memory demands of high-resolution textures, a technique called virtual texturing is often used [2, 16]. By treating a shadow map as an ordinary texture, we can apply all of these concepts to increase the efficiency with which we render and store shadow maps.

Each shadow map is divided into a set of pages and is fully virtualized. It is thus represented by a single texture called Virtual Page Texture (VPT). VPT is of much lower resolution than the full virtual resolution of the shadow map. Every entry in the VPT contains the metadata for one page. That is, each entry in the VPT contains the allocation status and coordinates of the allocated page in the Physical

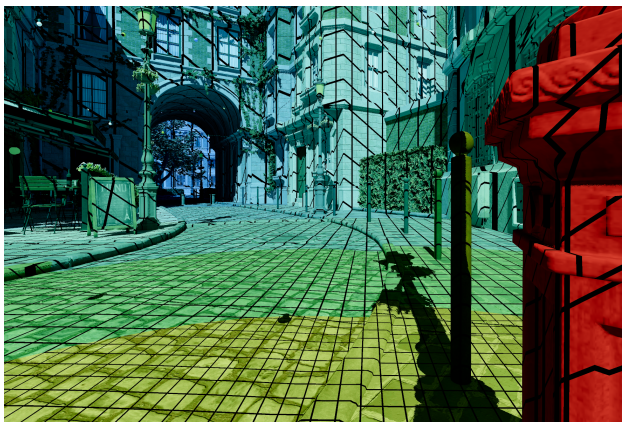

Figure 3: Visualization of the directional shadow map virtualization and paging scheme. Black lines denote individual shadow map tiles. Different colors denote different clip levels. We can clearly see how each clip level increases the page size. The objects close to the viewing position have low clipmap levels, because they require the most resolution. The further the objects get from the camera the less resolution we need and so higher clip levels are requested.

Page Texture (PPT). The PPT is a second texture that we introduce. It is shared by all shadow maps and is used to provide the physical backing for the allocated pages in the VPT. Because all lights share the PPT, the page size is uniform across all shadow maps and light types. The virtualization scheme can be seen in Figure 2.

An important part of our algorithm is determining the shadow map pages that will be required for this frame. For these purposes, the scene is rendered from the view of the main camera into a depth buffer. Following this, a pass over the acquired depth is performed. During this pass, we project each pixel into VPTs of each light. Once we have the footprint in the shadowmap, we can mark the pages that this pixel will require.

This is followed by an allocation step. By analyzing the PPT we can find the set of pages that can be reused. This set is used to allocate VPT pages, marked in the previous step, that do not have physical backing. We work with the assumption that the PPT always has enough space to back all visible pages.

The last part of our algorithm we would like to highlight is shadowmap caching. Typically, the frame-to-frame change in the visible, and thus marked, pages is not that high. This can be leveraged by preserving the visibility information across frames. When a VPT page from the last frame remains visible and its data in the PPT remain valid, we reuse it as opposed to re-rendering the page. Now we will describe how these concepts apply to individual light types present in the scene.

3.1 Directional lights

A typical scene contains only one or two directional lights. They always affect the entire scene and thus are particularly difficult to provide consistent shadows for. Due to their large area of influence, we represent them as a virtual texture with extremely high resolution. To allow for this approach, we need to use a concept called clipmap [15, 1]. The total virtual resolution is determined by the number of clipmaps and the resolution of the zeroth clipmap.

All levels of a clipmap maintain the same resolution. However, each level is separate and is represented by a single VPT. Lastly, every clipmap level covers twice the area of the previous one. The varying texel density of the clipmap is thus ensured by the area covered by a single clip map, not by varying resolution levels themselves. Figure 3 shows a visualization of the virtualized clip map scheme.

Directional lights use an orthogonal camera, which greatly simplifies the process of marking the required pages. This is because we do not need to project each pixel into the actual shadow map and instead can perform all our calculations in world space. A scheme depicting this process can be seen in Figure 4.

3.2 Spot lights

A scene usually contains a larger number of spotlights. However, unlike for directional lights, we do not need such a large virtual resolution. This is mainly due to the fact that their effects are much more localized. As such, resolutions of four to two thousand texels are often sufficient.

Clipmaps are thus no longer needed, and we can instead use the classic mipchain. This is a key difference between spotlights and directional lights. Directional lights have clip levels with constant resolution that cover an increasing area. Spot lights have mipmap levels with decreasing resolution that cover the same area.

While slightly complicating rendering of the shadow map, as will be described in the following chapters, it al-

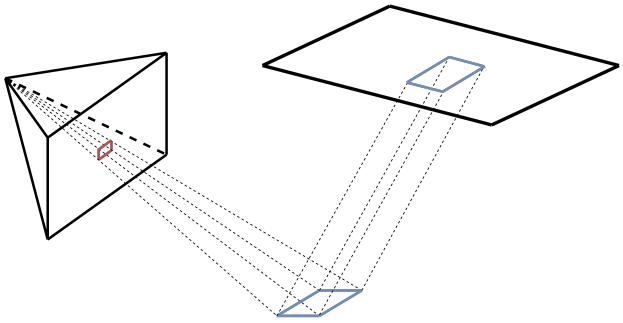

Figure 4: Illustrative figure showing how pixels are projected to determine the affected shadow map texels. The red square denotes a single pixel. It is first projected back into world space. All remaining calculations can be performed in world space. This is possible due to the orthogonal projection used by directional lights.

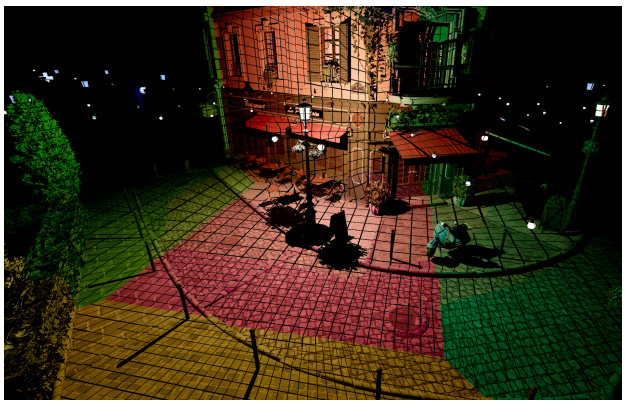

Figure 5: Visualization of paging and virtualization of a single point light. Black lines denote individual shadow map pages. Different colors determine separate point light faces. The sizes of the pages imply the mip map from which the tile was selected. The tiles are now distorted by the perspective projection of the point light and no longer appear strictly rectangular. Additionally, the more complicated mip level selection can be observed. As the distance towards the light decreases, the resolution of the shadow map increases, which leads to lower mip map levels required (visible on the bottom, red, face of the point light).

lows one to reduce the number of VPTs required to represent the light. Where directional lights require an entire VPT per clipmap, the mipchain of the VPT can be leveraged. Thus, a spotlight can be represented by a single VPT. Another difference is the usage of perspective projection for the spotlights. This makes marking the requested pages much more expensive. We can no longer perform the marking in view space and need to project fully into shadowmap space.

3.3 Point lights

The Last light type that will be described are point lights. Point lights are often represented as cubemaps. A cubemap can be represented by six rotated spotlights that share an origin. This makes point lights in many ways similar to spot lights. As such, point lights do not pose a significant difference for our technique. All our shadow map textures are fully virtualized; a point light is thus simply represented as six VPTs, one for each face. A visualization of the point light paging scheme can be seen in Figure 5. Of all the light types, point lights have the most expensive page marking process. In addition to requiring the perspective projection described for spot lights, the cubemap face needs to be manually determined. Figure 6 shows a visualization of the spotlight marking process.

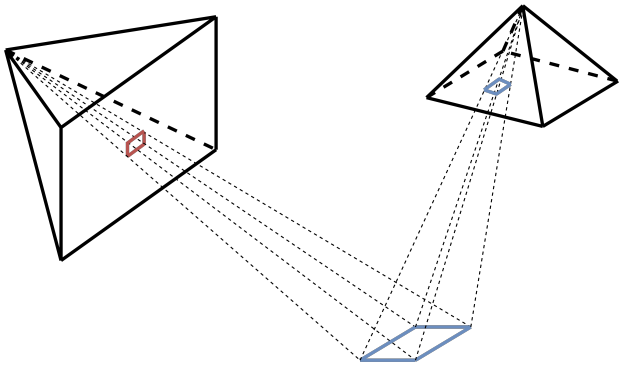

Figure 6: Illustrative figure showing how pixels are projected into a single spot light during marking phase. In contrast to directional lights, spot lights use perspective projection. This requires us to project the texel into world space before fully projecting into shadow map space, in order to find the footprint.

4 Shadowmap rendering

In this section, we will describe the process of rendering the shadows into pages backed by physical memory. We will be rendering the scene tens to hundreds of times. A highly efficient culling scheme is required to maintain real-time framerates. For this purpose, we adopt mesh shaders coupled with meshlets to represent our objects.

4.1 Scene representation

A scene is at the root of the hierarchical structure used to represent geometry. It contains unique meshes which are further divided into a set of meshlets. Each meshlet contains information about a bounded number of triangles. The bound is typically set very low; around 64 to 128 is used in practice. This greatly aids in work distribution on the GPU by closely fitting the mesh shading pipeline. Figure 7 shows a diagram that represents a simple scene.

Meshes only represent unique geometries present in the scene. They need to be accompanied by a tree-like structure of entities describing the scene layout and instancing of individual meshes. Every entity stores a transform and, optionally, indices to one or more meshes. The entity graph is used purely on the CPU. At the start of every frame, we walk through the entity graph and flatten it into a set of draw lists. A draw list is a list of mesh indices that logically group meshes with shared properties. For example, transparent and opaque meshes would be present in two separate draw lists.

4.2 Shadowmap rasterization

Once the draw lists of shadow casting meshes are produced, the shadow rendering can begin. As mentioned above, each meshlet contains a bounded number of triangles. However, the number of meshlets in each mesh is

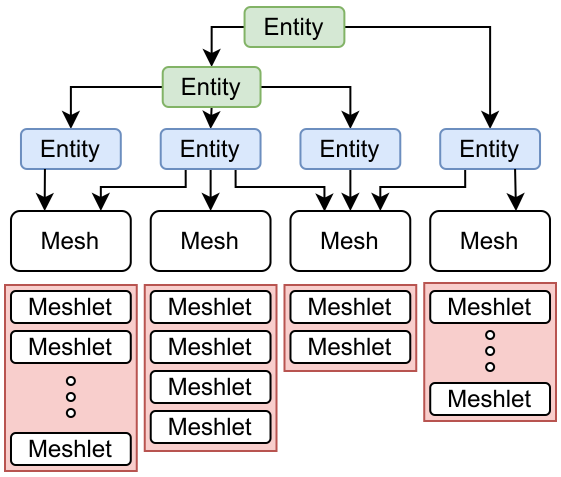

Figure 7: Scene representation Hierarchy. The bottom half shows the unique meshes and meshlets present in the scene. The upper part shows an example of the entity graph. Entities in blue reference at least one mesh, while green entities only contain a transform. Note that a single mesh can be referenced by more than one entity in the entity graph.

highly variable. This poses a significant challenge when scheduling work for the GPU. Because of this, we include an additional work expansion step.

Work expansion performs further flattening of the draw list, unwrapping meshes into a list of meshlet instances. Due to a relatively high number of meshlets present in a scene, this step is performed on the GPU. Once we have the list of meshlet instances, we utilize the indirect capabilities to dispatch a draw call. A task shader is assigned to each meshlet in the meshlet list. After processing the meshlet, it dispatches a group of mesh shaders. Finally, each mesh shader processes a subset of the meshlets' vertices and writes the resulting triangles.

There are slight differences in how individual light types are handled. As already described, every clip map of a directional light has the same resolution. We can pack meshlets for all clip maps into a single meshlet list, resulting in a single indirect draw to render all clip maps at once.

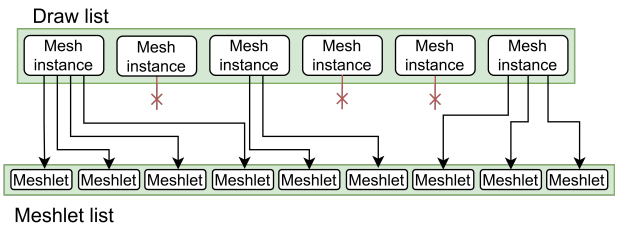

Figure 8: The process of draw list flattening during work expansion. The input is a draw list containing indices to individual meshes. The work expansion algorithm unwraps each mesh into individual meshlets that are placed into a second buffer. During this step some meshes can be culled, resulting in no meshlets placed into the expansion buffer.

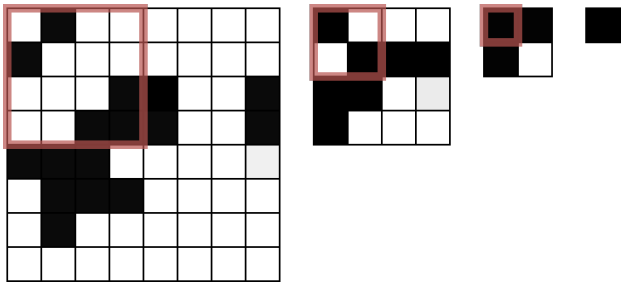

Figure 9: Four levels of Hierarchical Page Buffer (HPB). Black tiles denote dirty pages, that is, pages that will be drawn into this frame. Each level is a 2×2 logical "OR" reduction of the previous level. The red square denotes how areas across HPB levels map upon each other.

Spotlights and point lights, however, use a mipmap chain, which has varying resolution. We cannot pack meshlets in the same way that we pack directional lights, as individual draw calls are required for each resolution. In contrast to directional lights, however, a typically scene contains multiple point lights and spot lights. This allows us to pack meshlets for non-directional lights by desired resolution. Assuming the same virtual resolution, only $log_2(resolution)$ draw calls are required to draw shadows for all non-directional lights.

4.3 Geometry culling

Meshlet packing reduces the number of draw calls but does not reduce the amount of geometry processed. In order to maintain real-time frame rates, we need to cull the geometry to decrease the load on the rasterization stages. The proposed culling scheme closely matches our scene representation and the logical steps of the drawing pipeline.

First, during the work expansion step, we cull the meshes in the draw list. In this step, we only cull the entire meshes. Once a part of the mesh is determined visible, all of its meshlets are written into the meshlet list. This process is visualized in Figure 8. Second, in the task shader, we perform meshlet culling. Each task shader invocation dispatches a group of mesh shaders if the respective meshlet is visible. Meshlets determined not to be visible are discarded. Lastly, during meshlet processing, we cull individual triangles. Due to the relatively expensive nature of fragment shader invocations, triangle culling is pivotal in achieving the fastest possible results for RMVSMs.

We use various standard methods to determine visibility, starting from the fastest and most conservative. For point lights and spotlights, we start with distance culling. Because we know the radius each light affects, we can discard objects that lie outside of this area. This method is not valid for directional light sources as their radius would be infinite. Following this we perform frustum culling which is applicable for all light types. The bounds of each object

Figure 10: A night scene rendered using our method. The scene contains 26 shadow casting point lights with varying radii of influence and a single directional light. The virtual resolution of each point light face is 2048 texels. The resolution of each clip level is 4096 texels.

are checked against all frustum planes. If an object lies outside the visible frustum, it is discarded. There are two additional culling methods that we can utilize only when culling triangles. The first is back face culling. Typically, this step is performed by the rasterization pipeline; however, with mesh shaders, this needs to be done manually. The second is called micro-triangle culling. This step will remove all triangles that would not result in any fragment shader invocation after rasterization.

Lastly, we utilize a method specialized for RMVSMs called Hierarchical Page Culling. This method is applicable to all light types as well as all three culling stages. As already discussed, potentially only a very limited set of pages is visible at one time. This makes the set of pages that need to be drawn in any given frame very small. Knowing this, we define a structure called the Hierarchical Page Buffer (HPB). This structure and the culling process related to it are very similar to how a hierarchical Z buffer [6] is used for occlusion culling.

The HPB is used to cull geometry that does not overlap any dirty pages. For directional light sources, we build the HPB for each VPT, which corresponds to one HPB for each clip level. Spot lights and point lights utilize mip maps and have a single VPT for all mip levels. We still need to cull individual mip levels separately, and thus need to construct one HPB for each level of the VPT. Each level in the HPB is constructed as a 2×2 reduction of the lower level. The reduction starts with the lowest level that reduces the VPT itself. For this, a logical "OR" reduction is performed on a status bit marking the page dirty. Figure 9 shows four levels of the resulting HPB.

With this we can perform a standard culling procedure. The axis-aligned bounding box of the object is projected into the texture space of the light. From the projection area, the appropriate level of the HPB is calculated. The level is determined so that only four texels need to be sampled in order to cover the whole projection area. If none of the texels are marked as dirty, the object is culled.

5 Results and discussion

Now we present the results that we were able to obtain using our method. The method was implemented in an open source research framework called Timberdoodle [12].

Next, we discuss the performance of the proposed method. The measurements were taken with an NVIDIA GeForce RTX 4070 Ti SUPER GPU and an AMD Ryzen 7 7800X3D CPU. Each test involved moving the camera along a predetermined path. This is important when measuring cached shadow maps, as for static cameras, all pages remain cached, resulting in a practically zero cost.

For the first scenario, we utilized the Lumberyard Bistro scene [9] containing $\approx 4,000,000$ triangles. The scene contained 26 point lights, each having virtual resolution of 2048 texels. Additionally, a single directional light was present. The directional light had 16 clip map levels, where each level had a resolution of 4096 texels. For this test, we used a PPT with a resolution of 16384 texels. This was mainly done to guarantee enough space for all pages and thus provide consistent test results. Figure 10 shows an image rendered during this test case. Although the test with the lower virtual resolution could fit into a smaller PPT, less space would be available for page caching, and the results would be skewed.

	Marking	Directional	Point
	64	4096	2048
Uncached	1543 µs	1132 µs	4193 µs
Cached	1086 µs	322 µs	145 µs
	128	4096	2048
Uncached	1471 µs	1098 µs	4225 µs
Cached	985 µs	297 µs	152 µs
	128	2048	1024
Uncached	1351 µs	781 µs	1862 µs
Cached	867 µs	182 µs	53 µs

Table 1: Mean GPU times for the three primary stages of the algorithm in various configurations. Cached: the scheme described in Section 3 for caching pages is used. Uncached: every visible page is redrawn every frame. The numbers in the header rows denote the resolution of each page, the virtual resolution for the directional shadow map, and the virtual resolution for each of the point lights.

We measured the performance in three test configurations. The results of the measurements can be seen in Table 1. The first uses the highest resolution textures and the smallest page size of the three. This allows for the best culling because the HPBs have the highest resolution. The downside of this is a slightly elevated cost of the marking pass, caused by less efficient scalarization utilized in the marking pass implementation.

Another notable thing is the decreased performance of the marking pass when caching is disabled. To mark the pages, atomic operations were utilized. Due to the relatively expensive nature of these operations, we attempt to avoid them when possible. When a page is already cached, we do not need to do any further work and thus can skip the atomic allocation. This can not be done when caching is disabled, because all pages are reallocated each frame.

With decreased resolution, performance increases almost linearly. Because there are four times fewer pixels to draw, the drawing takes almost a quarter of the time. The marking pass is also slightly cheaper. We attribute this to the increased spatial location of the smaller VPTs.

For a second test scenario, we used the Intel Sponza scene [14] with the Ivy, Curtains, and Trees add-ons. This results in $\approx 12,000,000$ triangles. This scene contained a total of 20 point light sources along with a single directional light source. An image of the scene rendered using our method can be seen in Figure 11.

There is one difference in the lights setup from the previous scene. The radius of influence for each point light was reduced to approximately half of the first test scene. This was done to match the smaller scale of the entire scene. Not only does a smaller light radius affect the visuals, it also improves the geometry culling. Indeed, while the number of triangles is approximately three times larger, the results when caching was utilized are comparable.

	Marking	Directional	Point
		16384	
Uncached	1543 µs	870 µs	7293 µs
Cached	1487 µs	312 µs	47 µs
		8192	
Uncached	1469 µs	915 µs	6925 µs
Cached	1512 µs	415 µs	322 µs

Table 2: Mean GPU times for the three primary stages of the algorithm. Same as in 1 cached utilizes the scheme described in Section 3 for caching pages, while uncached redraw all visible pages every frame. Two PPT resolutions were tested denoted by the numbers in the header rows. The virtual resolution for each clip map was 4096 texels and virtual resolution for each point light face was 1024 texels.

In this test case, we also measured the effect that the size of the PPT has on performance. We tried two PPT resolutions 16384 and 8192 texels. The resolution of each point light face was reduced to 1024 texels. This was done to allow for smaller PPT size without encountering issues of insufficient memory. The results can be seen in Table 2. The case with a resolution of 8192 texels can be directly compared with a typical CSM implementation, having four cascades with resolution of 4096 texels. With RMVSMs we can utilize the same amount of memory to back sixteen cascades with virtual resolution of 4096 texels as well as 20 point light sources.

A reduction in performance can be observed when the PPT size is decreased. This affects both the directional and point lights as they share the PPT. That said, point lights are affected more because they benefit more from caching. For directional lights, pages still need to be invalidated and redrawn when the main camera moves. This is true even when caching is enabled and stems from the properties of clipmap. However, the above is not true for point lights. When the cache is large enough, all pages can be stored in the PPT and never redrawn. The performance of uncached shadow maps is independent of the PPT resolution as all pages are redrawn each frame.

6 Conclusion and future work

In this article we have presented an efficient method for rendering shadow maps. We showed that our method scales well for many light sources and generalizes to various light types, allowing for a unified approach. This is an improvement over existing methods that solve only a specific part of the problem. We have described how our method utilizes a specialized scene representation scheme to allow for highly efficient culling.

The marking pass is the most expensive part. Using a clustered shading approach proposed by [11] would reduce the number of lights for each pixel and reduce the

Figure 11: Intel Sponza rendered with our method. The scene contains 20 point lights with virtual resolutions of 1024 texels per point light face. The radii of influence for all point lights are identical and are comparatively smaller than in Figure 10. Additionally a single directional light source with virtual clip level resolution of 4096 texels is present.

required processing. Another improvement would be to utilize the hardware capabilities of modern GPUs to perform the marking step for us. Modern graphics APIs expose functionality called Sampler Feedback, which allows for capturing and recording texture sampling information and location. This could be used to project the pixel into the shadow map. As projection is the most costly operation of the marking pass, this could further increase the performance.

Lastly, sparse (or tiled) resources in graphic APIs are a promising choice for RMVSMs. Currently, the API for sparse resources does not allow for an indirect GPU-driven approach. Backing individual pages would require GPU read-back; however, we believe these issues can be negated by a clever approach.

Acknowledgments

This work was supported by the Grant Agency of the Czech Technical University in Prague, No SGS25/150/OHK3/3T/13.

References

[1] Arul Asirvatham and Hugues Hoppe. Terrain rendering using gpu-based geometry clipmaps. *GPU gems*, 2(2):27–46, 2005.

[2] Sean Barrett. Sparse virtual texture memory. `https://www.gdcvault.com/play/417/Sparse-Virtual-Texture`.

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

[4] Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P Greenberg. Adaptive shadow maps. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 387–390, 2001.

[5] Markus Giegl and Michael Wimmer. Queried virtual shadow maps. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 65–72, 2007.

[6] Ned Greene, Michael Kass, and Gavin Miller. Hierarchical z-buffer visibility. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 231–238, 1993.

[7] Andrew Lauritzen, Marco Salvi, and Aaron Lefohn. Sample distribution shadow maps. In *Symposium on Interactive 3D Graphics and Games*, pages 97–102, 2011.

[8] Aaron E Lefohn, Shubhabrata Sengupta, and John D Owens. Resolution-matched shadow maps. *ACM Transactions on Graphics (TOG)*, 26(4):20–es, 2007.

[9] Amazon Lumberyard. Amazon lumberyard bistro, open research content archive (orca), July 2017. http://developer.nvidia.com/orca/amazon-lumberyard-bistro.

[10] Morgan McGuire and Michael Mara. Efficient gpu screen-space ray tracing. *Journal of Computer Graphics Techniques (JCGT)*, 3(4):73–85, 2014.

[11] Ola Olsson, Erik Sintorn, Viktor Kämpe, Markus Billeter, and Ulf Assarsson. Efficient virtual shadow maps for many lights. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 87–96, 2014.

[12] Matěj Sakmary and Patrick Ahrens. Timberdoodle, 2025. https://github.com/Sunset-Flock/Timberdoodle.

[13] Matěj Sakmary, Jake Ryan, Justin Hall, and Alessio Lustri. Virtual shadow maps. In *GPU Zen 3*, pages 319–336. Black Cat Publising, 2024.

[14] Intel Sponza. Intel sponza scene, graphics research samples, May 2020. https://www.intel.com/content/www/us/en/developer/topic-technology/graphics-research/samples.html.

[15] Christopher C Tanner, Christopher J Migdal, and Michael T Jones. The clipmap: a virtual mipmap. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 151–158, 1998.

[16] JMP van Waveren and Evan Hart. Using virtual texturing to handle massive texture data. In *GPU Technology Conference*, volume 10, 2010.