Min-Max Mipmaps for Efficient 2D Occlusion Culling

Simon Scheckel[1] Andreas Kolb[2]
Computer Graphics Group, University of Siegen, Hölderlinstr. 3, 57074 Siegen, Germany
[1]simon.scheckel@student.uni-siegen.de; [2]andreas.kolb@uni-siegen.de

ABSTRACT

3D culling techniques are well established to improve rendering performance, but cannot be applied to 2D games in which the scene is composed of partially transparent textures in a known layer arrangement. Commonly, 2D rendering is achieved in a simple back-to-front blending scheme. This paper discusses options to realize 2D occlusion culling techniques using standard OpenGL functionalities, and introduces a novel 2D culling technique based on min-max mipmaps. We evaluate the performance of the different techniques for different scenarios.

Keywords

2D Picture/Image Generation; Textures and Framebuffer Operations; Mipmaps

1 INTRODUCTION & PRIOR WORK

In 2D graphics applications, such as games, the scene consists of a set of *layers* (2D textures with transparency) in a defined "depth" order. The resulting 2D rendering is commonly achieved in applying a simple back-to-front blending scheme.

While the primary target for both, 2D and 3D games is to optimize for high-quality graphics, subject to performance considerations and/or given hardware restriction, the major structural difference lies in the nearly vanishing geometry processing requirements for, and the known "spatial" structure of, 2D applications. Furthermore, 2D games rather target mobile platforms which impose greater hardware restrictions.

3D Culling Techniques: In 3D graphics applications, culling methods are commonly used in order to discard geometry on object level [3]. Culling techniques are designed to deal with the *a-priori unknown depth structure* of the 3D scene to be rendered which consists mainly of *fully opaque objects*.

Early culling techniques usually attempted to exploit specific scene or object structures, such as portal culling for buildings [13], or the aspect graph related to the projetive structure of polygonal models [12]; see Cohen et al. [3] for a in-depth discussion of early culling techniques. Alternative, image based approaches such as a hierarchical z-buffer approach by Green et al. [7] and the hierarchical occlusion maps by Zhang et al. [15] are applicable to generic scenes. Both approaches are related to our work, as they use hierarchical data structures in order to detect objects which are (potentially) hidden by occluders; see Sec. 4.

Hardware occlusion queries offer an efficient utility to test the visibility of bounding geometries against the depth buffer before rendering the contained object. Bittner et al. [1] realize an efficient coherent culling method, which uses an octree and temporal coherence in order to allow for an asynchronous handling of occlusion queries and rendering tasks.

Overdraw Reduction: The performance of 2D applications is steered by the GPU's fill-rate capability and the amount of overdraw present in the current scene. Thus, 2D culling is equivalent to *reduce overdraw operations*. In 3D rendering engines, deferred shading and lighting is used to reduce unnecessary shading operations by applying a fast z-prepass [10]. This technique can be applied to 2D image synthesis as well, however, in the prepass only rendering of opaque areas can be handled, while transparent regions need to be handled differently; see Sec. 2.

An alternative 2D approach is to identify opaque and not fully transparent texture areas and convert them into meshes, thus preventing overdraw for fully transparent regions. Church follows this idea by cutting off transparent regions using "corner cutting" and triangulating the remaining areas [2]. However, this technique cannot handle multiple opaque regions and holes, which frequently occur in 2D games.

Contribution: As the direct application of 3D occlusion culling methods to 2D rendering is of limited use, this paper discusses automated approaches to improve performance for 2D rendering by overdraw reduction. Beside alpha-testing (Sec. 2) and contour tracing with triangulation (Sec. 3), which do not lead to siginifcant performance gains, we introduce a novel 2D occlusion culling technique based on min-max mipmaps. The main advantages of our approach are

- generation of isosceles triangles covering fully opaque or fully transparent regions, resulting in superior geometry and rasterization performance, and
- adaptivity of the triangle refinement level, leading to a performance-optimal ratio between triangle size and triangle count.

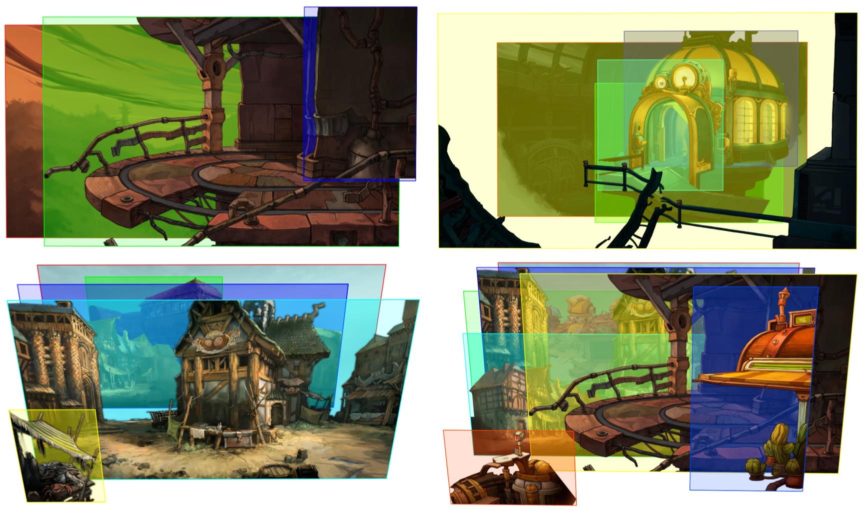

Figure 1: Sample scenes used for evaluation. The scene parameters are depicted in Tab. 1.

2 ALPHA-TEST-BASED CULLING

Alpha testing can naturally be used to identify the foremost, opaque layer at pixel level. Utilizing the alpha test requires a two-pass process. Given a layer sequence `layer[i]` sorted in depth order, in the first pass opaque pixels are drawn in the depth buffer in a front-to-back manner. Due to the front-to-back processing, areas behind near opaque regions are already discarded in the early depth test. In the second back-to-front pass, layer pixels hidden behind opaque pixels are again not drawn. However, in the worst case all layer pixels are drawn twice. Furthermore, transparent regions are not taken care of. An extension of this algorithm involves drawing already in the first pass (later referred as alpha2), but it still has to draw all transparent parts.

The main disadvantage in the alpha test algorithm is that all, i.e. opaque and transparent, regions are rasterized in each pass, even though in the first pass we are dealing with opaque regions only and, in the second pass, only foremost opaque pixel and transparent regions are of interest.

3 CONTOUR TRACING AND TRIANGULATION

A natural approach is to represent fully opaque *inner regions*, as well as the *outer regions* (fully transparent regions) in a layer texture as (triangular) meshes. Hereby, both regions need to fulfill a *consistency criteria*: the inner region may not contain pixels that are not opaque and the outer region must cover all transparent pixels.

We generate these kinds of triangulations on the CPU in a prepocessing step as follows:

Noise Reduction: By applying opening and closing to remove small features without violation of our consistency criteria.

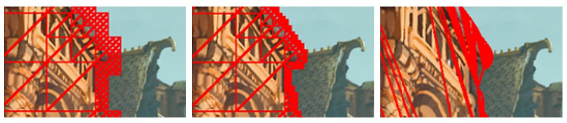

Figure 2: Sample triangulations generated with the Mip8, Mip2 and the contour-based mathods. Images from [4].

Contour Tracing: Tracing the boundary of the opening inner regions using a scanline-based, Moore-Neighbor tracing.

Reduction of Contour Points: In order to avoid a large set of triangles that need to be transfered to the GPU, we further reduce the set of contour points. However, we need to ensure, that this reduction does not violate the consistency criteria. Thus, we only can remove inner, collinear points.

Triangulation: The final triangulation is generated using a sweep-line algorithm and requires a list of closed contours in the image plane as input [6].

4 MAXIMUM MIPMAPS

As contour tracing & triangulation often leads to a large amount of long and thin triangles (see Fig. 2), we propose a novel minimum and maximum mipmaps (*min-max mipmaps*) technique in order to generate nearly ideal triangulations. Min-max mipmaps have already been used in computer graphics, e.g., for soft shadows [9], global illumination [11], collision detection using geometry images [8], and ray casting of terrain data [14].

Our approach computes min-max mipmaps as lower and upper bound for the alpha values in the area represented by each pixel in the mipmap. Thus, a mipmap pixel resembles a fully opaque or fully transparent region if its min and max mipmap values are equal to 1 or

0, respectively. We generate the min-max mipmap using a simple fragment shader, combining 4 pixels into one in each hierarchy generation step.

The triangulation algorithm classifies layer areas as opaque (inner triangulation) or transparent (outer triangulation) directly on the min-max mipmap hierarchy in a top-down manner. Starting from the highest (1x1 px) mipmap level, we recursively check for maximum and minimum pixel values and if the pixel cannot be classified as fully opaque or transparent we traverse to the next finer hierarchy level. If a (refined) mip-map pixel represents a fully opaque or fully transparent rectangular, the related layer region is tessellated using two isosceles triangles. The classification process for both maximum mipmaps and contour tracing is done **only once** for each layer with the outer and inner regions and results in a list of triangles that are used to draw.

The main advantages of the mipmap-based approach is, that we can tune the amount of generated triangles by changing the *maximum level of refinement*. Apparently, a low maximum refinement level yields large "undecided" regions which need to be handled on a per-pixel level in the compositing stage. However, these "undecided" regions most likely contain many fully transparent and fully opaque regions which could be assigned to the inner or outer region, respectively. On the other hand, a high maximum refinement level leads to a large number of small triangles, resulting in a computational effort in the geometry processing and rasterization stages. We discuss the influence of the maximum refinement level in Sec. 5.

5 RESULTS

In this section we compare the 2D occlusion techniques to the standard back-to-front blending for 2D image synthesis. Here 'Alpha2' denotes the changed version of the alpha test with drawing in the first pass and 'Mip*N*' stands for our min-max mipmap-based occlusion culling, where the smallest region generated by the refinement is a NxN pixel region in the final image. For the comparison, we use four scenarios from the 2D games [4, 5]. Tab. 1 depicts the relevant key features of these scenes. The scenes consist of layers with different sizes, which we classified in FullHD, medium (1400x600) and small (600x400). To get genuine results, shader operations consist only of a matrix multiplication for the vertex shader and a texture lookup for the fragment shader. All scenes consist of static layers, that can be preprocessed as is the case for most 2D games. The synthesized images are FullHD throughout.

5.1 Preprocessing

Since we assume static layer geometries, the performance of the preprocessing stages of the individual algorithms is of little practical impact. Still, we present the preprocessing timings in Tab. 2 for FullHD, medium and small texture layers, as a reference for the estimated complexity.

	Layers	Px Cnt	Min Px Cnt
Scene 1	3	6,240k	2,238k (35.8%)
Scene 2	5	8,041k	2,148k (26.7%)
Scene 3	5	8,121k	2,088k (25.7%)
Scene 4	7	11,389k	2,132k (18.7%)

Table 1: Parameters for test scenes: Number of layers, the total count of pixels without culling ('Px Cnt') and the theoretical minimal count of drawn pixels ('Min Px Cnt').

5.2 Performance

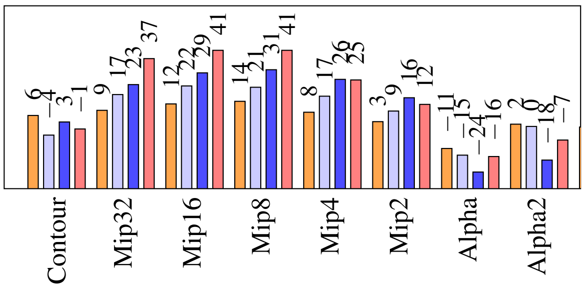

Figure 3: Relative performance gain compared to standard back-to-front blending on a GTX 770 with Nvidia Nsight [%].

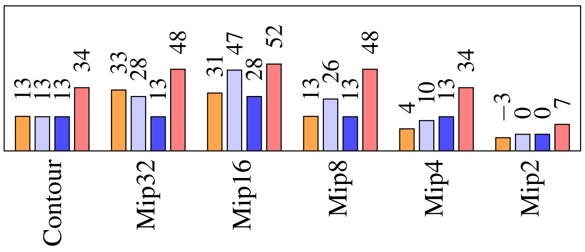

Figure 4: Relative performance gain compared to standard back-to-front blending on an iPad 3 [%]

Figs. 3 and 4 show the relative performance gain of each method compared to the standard back-to-front compositing scheme. Due to the fine granular culling on

Algorithm	1920x1080	1400x600	600x400
Loading layers	5	6	1
Contour tracing	127	126	33
Mip16	38	31	20
Mip8	37	33	22
Mip4	46	37	25
Mip2	49	33	24

Table 2: Preprocessing time [ms] for loading the layers and triangulation (including mipmap generation for Mip$N, N \in \{2,4,8,16\}$).

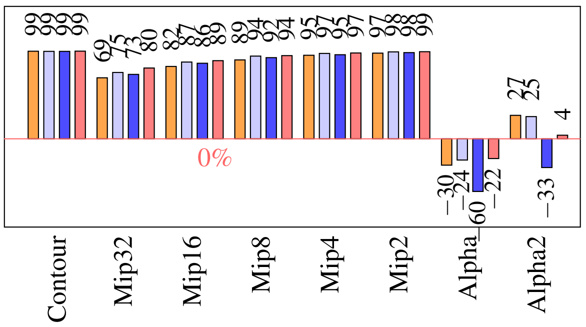

Figure 5: Culling efficiency: The relative amount of culled pixel, where 100% equals to the minimum amount of pixel draw, 0% to the standard back-to-front blending; see Tab. 1.

Method	Scenes 1	2	3	4
Con	7,457	9,895	5,750	16,154
Mip32	3,238	2,764	1,918	5,412
Mip16	7,182	6,724	4,656	12,448
Mip8	15,220	15,080	10,524	28,130
Mip4	27,668	28,814	19,722	53,822
Mip2	51,784	55,760	37,768	100,936

Figure 6: Number of triangle generated by the contour tracing (Con) and the mipmap methods Mip$N$ for $N \in \{2,4,8,16,32\}$ for the scenes (Sc) 1–4.

pixel level and the additional draws required to generate the depth map, the alpha test based approach performs very poorly. The iPad doesn't support Alpha Test, so only triangulation based methods are tested here. Most interestingly, the contour based triangulation approach performs poorly on the PC and somewhat better on the iPad, whereas our mipmap based approach constantly performs best for $N = 8$ and $N = 16$ on the PC and for $N = 16$ and $N = 32$ on the iPad.

On the other hand, we investigate the *culling efficiency* (see Fig. 5), i.e., the relative amount of culled pixels compared to the maximum possible amount of culling (see also Tab. 1). Here, the contour-based approach delivers the most accurate coverage of the regions to be culled. The alpha-blending approaches naturally require many additional draws in order to generate the opaque depth map, thus their efficiency in terms of overdraw reduction is very poor. The mipmap approach's efficiency increases as the level for generating triangles gets smaller, i.e., for decreasing N.

The mipmap approach generates isosceles triangles which can be rasterized more efficiently, whereas the contour based approach generates elongated triangles (see Fig 2). Compared to the PC, the iPad has a bigger performance hit for higher triangle counts, but it shows similar results as the desktop GPU solution 4.

6 CONCLUSION & FUTURE WORK

We present a novel technique to automatically generate efficient, isosceles triangulations for inner and outer regions of partially transparent textures which leads to a significant overdraw reduction in 2D rendering. Our technique can be adopted to the geometry performance capacities of the targeted platform, leading to a performance-optimal ratio between triangle size and triangle count. Future work may include dynamic textures, which requires efficient online triangulation methods.

7 REFERENCES

[1] J. Bittner, M. Wimmer, H. Piringer, and W. Purgathofer. Coherent hierarchical culling: Hardware occlusion queries made useful. *CGF*, 23(3):615–624, 2004.

[2] A. Church. *Depth-Cull Optimization of 2D Scenes for 3D Graphics Hardware*. 2014.

[3] D. Cohen-Or, Y. L. Chrysanthou, C. T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE TVCG*, 9(3):412–431, 2003.

[4] Daedalic Entertainment. *The Dark Eye: Chains of Satinav (Computer Game)*, 2012.

[5] Daedalic Entertainment. *Deponia (Computer Game)*, 2012.

[6] V. Domiter and B. Žalik. Sweep-line algorithm for constrained delaunay triangulation. *J. Geo. Inf. Sci.*, 22(4):449–462, 2008.

[7] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. In *Proc. SIGGRAPH*, pages 231–238, 1993.

[8] A. Greß, M. Guthe, and R. Klein. Gpu-based collision detection for deformable parameterized surfaces. *CGF*, 25(3):497–506, 2006.

[9] G. Guennebaud, L. Barthe, and M. Paulin. Real-time soft shadow mapping by backprojection. In *Rendering Techniques*, pages 227–234, 2006.

[10] T. Harada, J. McKee, and J. C. Yang. Forward+: A step toward film-style shading in real time. *GPU Pro 4: Adv. Rend. Techn.*, 4:115, 2013.

[11] G. Nichols, J. Shopf, and C. Wyman. Hierarchical image-space radiosity for interactive global illumination. *CGF*, 28(4):1141–1149, 2009.

[12] H. Plantinga and C. R. Dyer. Visibility, occlusion, and the aspect graph. *J. Computer Vision*, 5(2):137–160, 1990.

[13] S. J. Teller and C. H. Séquin. Visibility preprocessing for interactive walkthroughs. In *Proc. SIGGRAPH*, volume 25, pages 61–70, 1991.

[14] A. Tevs, I. Ihrke, and H.-P. Seidel. Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. In *I3D*, pages 183–190, 2008.

[15] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. Visibility culling using hierarchical occlusion maps. In *Proc. SIGGRAPH*, pages 77–88, 1997.