Multisampled Antialiasing of Per-pixel Geometry

Michael Schwarz and Marc Stamminger

University of Erlangen-Nuremberg

Abstract

Many algorithms exist which generate per-pixel geometry by selectively discarding fragments generated for a simple bounding geometry. On the other hand, multisampling support has become ubiquitous and is almost free in current graphics hardware. In this paper we leverage the ability of setting a pixel's coverage mask in the pixel shader to make seemingly inherently pixel-based per-pixel geometry approaches compatible with multisampled antialiasing. We consider both the rendering of curve regions as well as displacement mapping with antialiased outer and inner silhouettes.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Antialiasing

1. Introduction

Many approaches exist for improving visual quality by antialiasing. A popular technique which has direct hardware support is *multisampling* [AMHH08]. Here, each pixel is composed of several multisamples distributed across the pixel (cf. Fig. 2), with color and depth being stored for each multisample. During rendering, a fragment's color is evaluated only at the pixel center (or a different location within the rendered primitive's footprint in case of centroid sampling) while a coverage mask is determined by the rasterizer, identifying those multisamples that are covered by the rendered primitive's footprint and to which the computed color applies. Multisampling is supported by all recent GPUs and usually incurs at best a minor performance impact.

At the same time, many algorithms have been developed which generate *per-pixel geometry*. They render a simple bounding geometry to trigger the pixel shader's execution and then selectively discard fragments depending on whether they belong to the actual geometry or not. One class of algorithms employs an implicit equation to determine a pixel's membership to the desired geometry. For instance, regions bounded by Bézier curves [LB05] may be drawn this way. Another class of algorithms casts a ray from the camera through the pixel center into some geometry, for example height fields in per-pixel displacement mapping [SKU08]. A major advantage of such techniques is that one can zoom in without requiring a finer retessellation of the geometry to preserve the visual smoothness. On the downside, these approaches are not directly compatible with multisampling, often resulting in a jagged appearance.

With the recently introduced Direct3D-10.1-class graphics hardware, it has become possible to read multisample positions within the pixel shader and output a coverage mask. In principle, per-pixel geometry algorithms can hence be adapted to support multisampled antialiasing. However, the most direct way of just executing the pixel shader for each multisample separately runs counter to one core idea of multisampling, the different execution frequency of potentially expensive shading, performed only once per fragment, and coverage determination, done for each multisample.

In this paper, we present techniques for augmenting existing per-pixel geometry approaches with support for multisampled antialiasing that preserve the execution frequency of the shader computations, i.e. their time complexity remains constant instead of linear in the number of multisamples. First, we discuss the rendering of curve regions (Sec. 2), extending an approach that evaluates an implicit equation. Second, as an example for a ray-casting technique, we deal with displacement mapping (Sec. 3), accounting for both outer and inner silhouettes.

2. Curve rendering

An elegant algorithm for rendering regions bounded by quadratic and cubic Bézier curves was introduced by Loop and Blinn [LB05]. In the quadratic case, where a curve is

Figure 1: *Example of curve rendering with 8 multisamples. While alpha blending and alpha-to-coverage fail at some interface pixels, both per-multisample evaluation and our approximate coverage technique yield good visual results.*

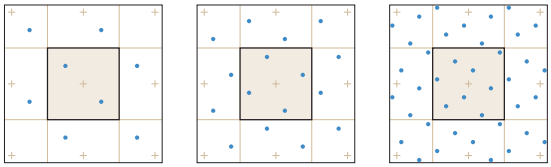


Figure 2: *Multisample patterns for the standard quality level defined by Direct3D 10.1.*

specified by control points $\mathbf{b}_0$, $\mathbf{b}_1$, $\mathbf{b}_2$, they render a triangle with vertex positions $\mathbf{b}_i$ and (u,v) texture coordinates $(0,0)$, $(\frac{1}{2},0)$, $(1,1)$, respectively. In the pixel shader, the implicit function $f(u,v) = u^2 - v$ is evaluated. Its sign determines whether the pixel is inside or outside the region bounded by the curve. For the convex region of the curve, a value $f(u,v) > 0$ identifies pixels outside; in such cases the fragment gets discarded in the pixel shader.

As shown in Fig. 1, this simple per-pixel procedure is prone to aliasing, leading to a jagged appearance. To alleviate this, Loop and Blinn [LB05] suggested an antialiasing scheme where for each pixel an alpha value approximating the pixel area covered by the curve region is determined. They first compute the gradient $\nabla g(X,Y)$ for $g(X,Y) = f(\Psi(X,Y))$ where Ψ is the mapping from pixel space (X,Y) to texture space (u,v), and calculate an approximate signed distance $d(X,Y) = g(X,Y)/\|\nabla g(X,Y)\|$. This distance is then used to derive an alpha value; for the convex curve region case

$$\alpha(X,Y) = \max\big(0, \min\big(1, \tfrac{1}{2} - d(X,Y)\big)\big).$$

While this approach works well in practice for isolated curves, multiple overlapping curves cause problems because it is unclear how to combine their alpha values which represent coverage factors. Treating them as transparency values and performing alpha blending [PD84] can lead to visual artifacts, only some of which may be prevented by depth-sorted rendering of the curve regions.

Consider the situation in Fig. 1 again. Since the blue curve region is rendered atop the other ones, we get a correct result at the interface of the blue region with the interior of the other two curve regions. However, at pixels intersected by two or more curves, simple alpha blending fails irrespective of the rendering order. This is especially obvious at the interface of the red and the green regions, which both

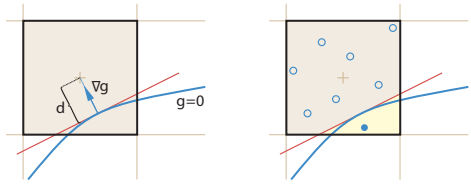


Figure 3: *Utilizing the (approximate) gradient ∇g and signed distance d of the curve $g = 0$ (left), we determine a half space and derive the contained multisamples (right) via a look-up texture.*

use the same curve as boundary; here alpha blending causes the occluded black background to actually have a non-zero contribution. (Note that in this special case additive blending would work except in the pixel intersected by all three curves.)

In a multisampled setup, alpha-to-coverage [AMHH08] can be used instead of alpha blending. The alpha value is mapped to a coverage mask where the percentage of set multisamples corresponds to the alpha value. While it circumvents the necessity for sorting, it basically fails in the same cases as alpha blending. For instance, if a pixel has an alpha value of 50% for both the green and the red curve, alpha-to-coverage yields the same coverage mask for both curves, i.e. the overall coverage would be only 50% instead of 100%.

In contrast, correct multisampled antialiasing can be achieved by evaluating $f(u,v)$ at each multisample of a pixel and setting the corresponding coverage mask. While correct, this approach somehow defies the idea of multisampling; in particular we want the pixel shader's time complexity to be constant and not linear in the number of multisamples.

To this end, we compute $\nabla g(X,Y)$ and $d(X,Y)$ as before. However, instead of deriving an alpha value, we use these quantities to derive an approximation of the exact coverage mask. The line $\langle \mathbf{x}, \mathbf{n} \rangle + d(X,Y) = 0$ with $\mathbf{n} = \nabla g(X,Y)/\|\nabla g(X,Y)\|$ separates the pixel area into two half spaces (cf. Fig. 3). We precompute the coverage masks for a number of half spaces and provide them in a 2D look-up texture. For texture parameterization, we employ the Hough transform [DH72] in a way similar to Eisemann and Décoret [ED07]. That is, all we have to do is to load the corresponding coverage mask from a texture based on $\nabla g(X,Y)$ and $d(X,Y)$ and output this mask in addition to the color in the pixel shader.

Due to the half space approximation, the coverage mask is not always exact. Recall however that the alpha value is also only approximate in nature. Moreover, while the alpha value is a function of distance to the pixel center only, the half space approximation also accounts for orientation. As exemplarily shown in Fig. 1, the visual result with our approximate coverage approach is basically indistinguishable from the multisample-accurate solution. Concerning performance, our approach has a negligible impact. On an AMD Radeon HD 4850 at 1024×768 resolution and with 8 multisamples, the scene from Fig. 1 renders at 450 Hz with our technique compared to 456 Hz with alpha-based antialiasing. In contrast per-sample evaluation runs only at 229 Hz.

The cubic curve case can be dealt with analogously, only the texture coordinates associated with the control points, g and ∇g must be adapted accordingly. The same technique can also be applied to other shapes defined by an implicit equation.

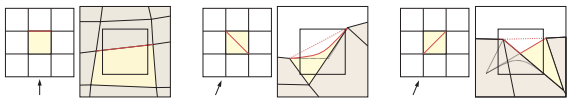
3. Displacement mapping

In the real-time rendering context, displacement mapping [Coo84] is usually implemented by drawing a simple bounding geometry and performing ray casting against the height field defining the displacement in the pixel shader [SKU08]. If the height field is missed by the spawned ray, the fragment gets discarded.

Compared to techniques like curve rendering discussed above, in displacement mapping aliasing occurs not only at boundaries of the mapped geometry but also at interior silhouettes. High-quality antialiasing hence necessitates expensive methods like casting multiple rays per pixel (for instance, one per multisample) or casting a beam. Our goal is to achieve reasonable antialiasing, both at outer and inner silhouettes, without having to cast more than one ray per pixel, keeping the performance impact acceptable. Note that the restriction to one ray implies that some features of the height map may get missed and hence determining accurate coverage masks is impossible.

We selected the state-of-the-art maximum mipmap approach of Tevs et al. [TIS08] as basis for our antialiasing technique. For each pixel a ray from the camera position to the pixel center is cast against the height field. Ultimately, the ray is intersected against a bilinear patch defined by 2×2 height map values using a rather expensive analytic solution. To avoid unnecessary intersection tests, a kind of maximum quadtree called maximum mipmap is employed as acceleration structure.

For each pixel, we start with an empty coverage mask and trace a ray against the height field until either the coverage mask is completely set or the ray left the bounding geometry. If the ray hits a bilinear patch, we perform shading and determine an approximate coverage mask. The pixel's coverage



(a) far edge (b) diagonal 1 (c) diagonal 2

Figure 4: *Based on the height field values, different patch horizons (red line) can occur. Left: top-down view of height map with ray. Right: pixel view.*

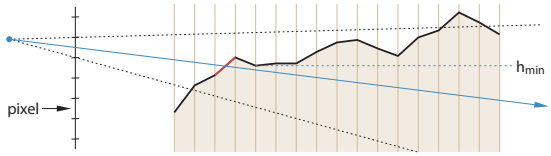
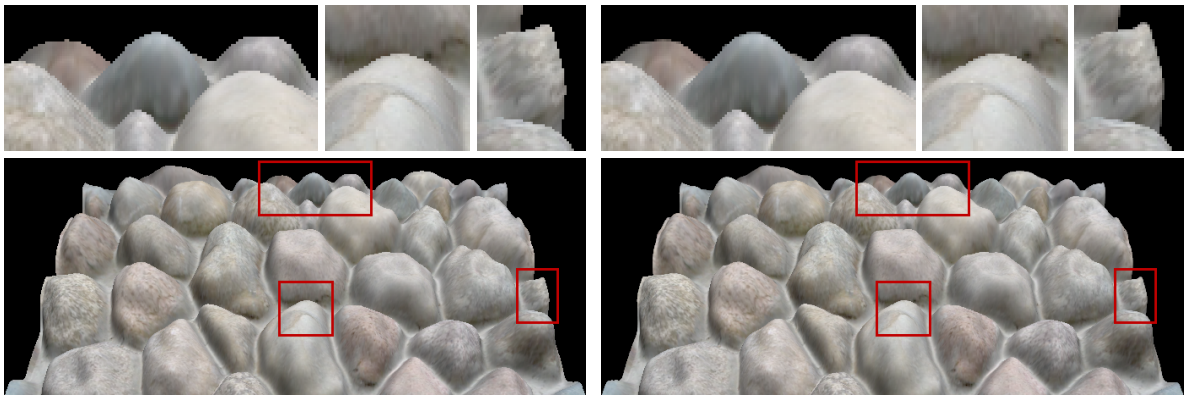


Figure 5: *A ray hitting a patch whose boundary constitutes an inner silhouette may never hit patches farther away, even if they cover the rest of the pixel.*

mask is updated by ORing in this new coverage mask. Moreover, we add the shading color to the pixel color, weighting it based on how many new coverage mask bits got set.

To determine the coverage mask for a hit patch, we resort to a heuristic. In particular, we also have to cheaply account for adjacent patches that cover the pixel area since otherwise visibility artifacts can occur. We first note that a patch's pixel coverage is mainly influenced by the patch's "horizon" (cf. Fig. 4). Based on this observation, we project all four patch vertices to pixel space and construct an initial horizon line defined by the two far-edge vertices. We then test the two remaining vertices against this line; if any of them is above the line, the horizon is updated accordingly. Consequently, we conservatively approximate the horizon in the diagonal cases (cf. Fig. 4 b, c) by a line. Finally, we consider the patch adjacent to the current patch in the ray's principal direction and test the pixel-space projections of its two far-edge vertices against the horizon line. If none of them is above the line, we assume the horizon to constitute an inner silhouette; the appropriate coverage mask is determined as in Sec. 2 for the half space below the horizon line by utilizing a look-up texture. On the other hand, if any of these two vertices is above the horizon, the adjacent patch is visible (like in Fig. 4 a) and we adopt a fully set coverage mask.

In same rare cases, it might happen that while the adjacent patch is not visible, patches further away are visible and cover the whole pixel but are never hit by the ray (cf. Fig. 5). As a consequence, the final coverage mask may incorrectly not be fully set, i.e. the background partially contributes to the final pixel color but shouldn't. To alleviate this, we first observe that if the ray exits the bounding geometry below the minimum height field value $h_{\min}$ of all patches traversed further along the ray, the height field is ultimately above the ray and hence covers at least part of the pixel. Each time, a patch it hit, we determine an approximation of $h_{\min}$ and



per pixel approximate coverage with adapted ray (8 multisamples)

Figure 6: *Example of displacement mapping with conventional per-pixel execution and our approximate coverage technique.*

check the ray against it; if the ray will exit below $h_{\min}$, we set a "below" bit. Finally, when the displacement mapping algorithm has completed and the coverage mask is not fully set, we check the below bit. If it is set, we upgrade the coverage mask to full coverage. The $h_{\min}$ values for each patch are stored in a pre-computed texture, where we quantize the ray directions to eight sectors.

Note that due to the simplified treatment of the visibility of neighboring patches, our method may miss some silhouettes and overestimate coverage. For the special cases where the patch is at the left or right boundary of the height field, we further determine the coverage mask for the left or right patch edge, respectively, and AND it with the horizon-based patch coverage mask to improve outer silhouette performance.

When casting the ray through the pixel center, silhouettes can only be detected if the pixel center lies below the corresponding patch horizon. As a consequence, coverage values of less than 50% can never occur, which negatively impacts the antialiasing capabilities. We hence adapt the cast ray towards the pixel corner in the downward direction and closest to the screen center. On the downside, since the ray is also used for shading, the displacement mapping result gets slightly distorted, which is however only noticeable when switching to the reference solution.

As shown in Fig. 6, our technique improves the visual quality compared to the ordinary per-pixel execution of the displacement mapping algorithm. In particular, when using an adapted ray, a reasonable antialiasing performance is achieved. However, there is usually still a gap to the quality obtained with the expensive per-multisample execution. Concerning performance, the visual improvements offered by our approach come at a moderate cost. On an AMD Radeon HD 4850 at 1024×768 resolution and with 8 multisamples, the rocks from Fig. 6 render at 75 Hz with conventional per-pixel execution and at 50 Hz with our technique while per-multisample execution runs at only 7.7 Hz.

4. Conclusion

Multisample support is ubiquitous and almost for free with current hardware. On the other hand many algorithms for generating per-pixel geometry exist, most of which ignore antialiasing issues. In this paper, we have presented two concrete techniques for augmenting such algorithms with multisampled antialiasing capabilities. As first example, we discussed the rendering of regions bounded by quadratic curves and also highlighted weaknesses with alternative antialiasing approaches. We then showed how to incorporate multisample coverage masks into a displacement mapping algorithm to improve visual quality of both outer and inner silhouettes.

References

[AMHH08] AKENINE-MÖLLER T., HAINES E., HOFFMAN N.: *Real-Time Rendering*, 3rd ed. A. K. Peters, 2008.

[Coo84] COOK R. L.: Shade trees. *Computer Graphics 18*, 3 (1984), 223–231.

[DH72] DUDA R. O., HART P. E.: Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM 15*, 1 (1972), 11–15.

[ED07] EISEMANN E., DÉCORET X.: Visibility sampling on GPU and applications. *Computer Graphics Forum 26*, 3 (2007), 535–544.

[LB05] LOOP C., BLINN J.: Resolution independent curve rendering using programmable graphics hardware. *ACM Transactions on Graphics 24*, 3 (2005), 1000–1009.

[PD84] PORTER T., DUFF T.: Compositing digital images. *Computer Graphics 18*, 3 (1984), 253–259.

[SKU08] SZIRMAY-KALOS L., UMENHOFFER T.: Displacement mapping on the GPU — State of the art. *Computer Graphics Forum 27*, 6 (2008), 1567–1592.

[TIS08] TEVS A., IHRKE I., SEIDEL H.-P.: Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. In *Proceedings of Symposium on Interactive 3D Graphics and Games (I3D 2008)* (2008), pp. 183–190.