Improved Dual-Space Bounds for Simultaneous Motion and Defocus Blur

Samuli Laine Tero Karras

NVIDIA Research*

Abstract

Our previous paper on stochastic rasterization [Laine et al. 2011] presented a method for constructing time and lens bounds to accelerate stochastic rasterization by skipping the costly 5D coverage test. Although the method works for the combined case of simultaneous motion and defocus blur, its efficiency drops when significant amounts of both effects are present. In this paper, we describe a bound computation method that treats time and lens domains in a unified fashion, and yields tight bounds also for the combined case.

1 Extended Dual Space

In previous work [Laine et al. 2011], time and lens bounds were considered separately, and in order to guarantee correctness, both of these domains had to be conservative with respect to the other. In other words, the lens bounds had to be valid for all instants of time, and the time bounds had to be valid for all lens positions. In situations where significant amounts of both effects occur simultaneously, this decreases the sample test efficiency (STE).

Treating the time and lens domains in a unified fashion lets us derive tight bounds even when both effects occur simultaneously. It also simplifies the overall algorithm.

We extend the dual-space idea of previous work into three dimensions. As before, the construction is based on parallel projection of a vertex onto camera plane, illustrated in Figure 1. The position of the projected point on camera plane is δ_x (resp. δ_y), and it is defined as a function of projection direction γ_x (resp. γ_y), lens position u (resp. v), and time t.

As before, the projection formulas are

$$\delta_x = x' - w' \cdot \gamma_x$$
$$\delta_y = y' - w' \cdot \gamma_y$$

where

$$x' = (1-t) \cdot (x_0 + u \cdot c_0) + t \cdot (x_1 + u \cdot c_1)$$
$$y' = (1-t) \cdot (y_0 + A \cdot v \cdot c_0) + t \cdot (y_1 + A \cdot v \cdot c_1)$$
$$w' = (1-t) \cdot w_0 + t \cdot w_1$$

Here, A is a constant scaling factor for making the circles of confusion circular regardless of viewport aspect ratio. As such, $\delta_x(\gamma_x, u, t)$ and $\delta_y(\gamma_y, v, y)$ are trilinear functions of their parameters. This is a crucial property, because it allows us to easily construct bounding functions for a group of δ functions corresponding to the vertices of a primitive.

*e-mail: {slaine,tkarras}@nvidia.com

NVIDIA Technical Report NVR-2011-004, November 2011.

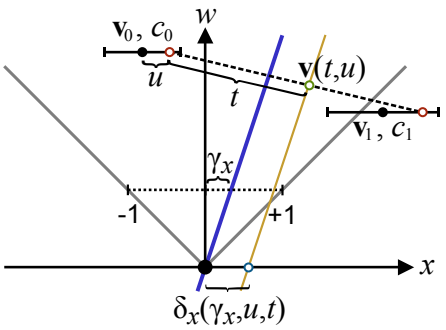

Figure 1: *Parallel projection of a point as the basis of the dual space, illustrated here for horizontal axis only. A vertex has position* $\mathbf{v}_0$ *and clip-space circle of confusion* c_0 *at* $t = 0$*, and similarly* $\mathbf{v}_1$ *and* c_1 *at* $t = 1$*. Value* $\delta_x(\gamma_x, u, t)$ *is obtained by applying lens coordinate* u *at* $t = 0$ *and* $t = 1$*, interpolating linearly according to* t*, and projecting onto camera plane in direction* γ_x*.*

2 Algorithm

In total, we form four bounding functions corresponding to min/max δ in horizontal direction, and similarly for vertical direction. Let us consider the maximum horizontal bound for a primitive. For this bound, the criterion that the primitive may overlap a screen-space point corresponding to projection direction γ_x with given (u, t) is that $\delta_x > 0$ for at least one vertex.[1] Therefore, to test all vertices of a primitive, we can take the maximum δ_x of all vertices before performing the comparison. Conversely, for minimum bound in horizontal direction we need the minimum δ_x.

Each vertex of the primitive defines individual $\delta_x(\gamma_x, u, t)$ and $\delta_y(\gamma_y, v, t)$ functions. For example, assume we need to find the maximum of a number of δ_x functions in some axis-aligned region of γ_x, u, t space. Since each of these functions is trilinear, we can represent them by their values at the eight corners of this region. By taking the maximum value of each of these δ_x functions at the corners, we obtain a trilinear function that bounds the given δ_x functions from above. In this fashion, we construct trilinear approximations for minimum and maximum δ functions for horizontal and vertical directions.

For u, v and t it is sufficient to assume full ranges when constructing the bounding δ function approximations, but for γ we determine a conservative range as described below in Section 2.2. The eight corners where the functions are evaluated in order to construct the approximating trilinear functions are therefore the combinations of $\gamma_x \in \{\gamma_{x\,\min}, \gamma_{x\,\max}\}$, $u \in \{-1, +1\}$, and $t \in \{0, 1\}$ for δ_x, and similarly for the vertical direction.

Still focusing on the maximum horizontal bound for the primitive, we compute the eight δ_x function values at the aforementioned corners, the value in each corner being the maximum of the δ_x functions of the individual vertices evaluated at this point. To test an individual sample against this bound, we could interpolate between

[1] It is also necessary that $\delta_x < 0$ for at least one vertex. This is handled by the minimum horizontal bound.

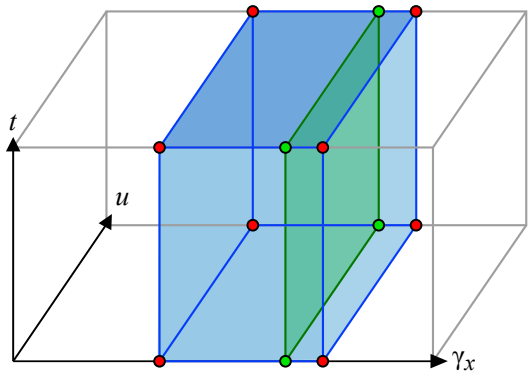

Figure 2: *Illustration of the u, t slicing. The large box is the entire γ_x, u, t space, and the shaded blue region is the volume bounded by γ_x range computed for the primitive. The values of the approximating trilinear δ_x functions are determined at the points denoted by red dots. To accelerate the testing of individual sampling points against the bounds, we extract a slice (green rectangle) according to the γ_x of a pixel tile by interpolating the corner values of the volume along γ_x axis.*

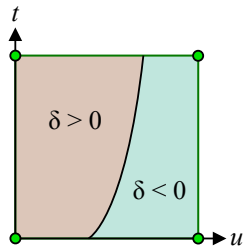

Figure 3: *A slice of the trilinear approximate δ_x is a bilinear function in u, t space. Testing the sign allows us to skip the 5D coverage tests for individual samples according to their u, t coordinates. In contrast to the previous method ([Laine et al. 2011]), this approach captures the combined effects of time and lens coordinates.*

function GetGammaRanges(vertex $v[N]$)

1: $(x_{\min}, x_{\max}, y_{\min}, y_{\max}) \leftarrow (+\infty, -\infty, +\infty, -\infty)$
2: $(\tilde{x}_{\min}, \tilde{x}_{\max}, \tilde{y}_{\min}, \tilde{y}_{\max}) \leftarrow (+\infty, -\infty, +\infty, -\infty)$
3: **for** each v **do**
4: **for** $i \in \{0, 1\}$ **do**
5: $(x, y, c_x) \leftarrow (v.x_i, v.y_i, v.c_i)/v.w_i$
6: $c_y \leftarrow A \cdot c_x$
7: **if** $v.w_i \geq 0$ **then**
8: $x_{\min} \leftarrow \min(x_{\min}, x - |c_x|)$
9: $x_{\max} \leftarrow \max(x_{\max}, x + |c_x|)$
10: $y_{\min} \leftarrow \min(y_{\min}, y - |c_y|)$
11: $y_{\max} \leftarrow \max(y_{\max}, y + |c_y|)$
12: **else**
13: $\tilde{x}_{\min} \leftarrow \min(\tilde{x}_{\min}, x - |c_x|)$
14: $\tilde{x}_{\max} \leftarrow \max(\tilde{x}_{\max}, x + |c_x|)$
15: $\tilde{y}_{\min} \leftarrow \min(\tilde{y}_{\min}, y - |c_y|)$
16: $\tilde{y}_{\max} \leftarrow \max(\tilde{y}_{\max}, y + |c_y|)$
17: **end if**
18: **end for**
19: **end for**
20: **if** $x_{\min} < \tilde{x}_{\max}$ **then** $x_{\min} \leftarrow -1$
21: **if** $x_{\max} > \tilde{x}_{\min}$ **then** $x_{\max} \leftarrow +1$
22: **if** $y_{\min} < \tilde{y}_{\max}$ **then** $y_{\min} \leftarrow -1$
23: **if** $y_{\max} > \tilde{y}_{\min}$ **then** $y_{\max} \leftarrow +1$
24: $x_{\min} \leftarrow \max(x_{\min}, -1)$
25: $x_{\max} \leftarrow \min(x_{\max}, +1)$
26: $y_{\min} \leftarrow \max(y_{\min}, -1)$
27: $y_{\max} \leftarrow \min(y_{\max}, +1)$
28: **return** $(x_{\min}, x_{\max}, y_{\min}, y_{\max})$

Figure 4: *Pseudocode for computing horizontal and vertical γ ranges (i.e., screen-space bounding rectangle) that are valid for the entire u, v, t range, and that extend to the edge of the screen where appropriate. The idea is to keep track of two bounding rectangles, one in front of (lines 8–11) and one behind (lines 13–16) the camera. The latter is used for extending the front bounding rectangle to the edge of the screen in the directions where the primitive may cross the camera plane (lines 20–23). The surprising-looking indexing is due to back rectangle being inverted. Lines 24–27 clamp the γ ranges to the screen extents.*

these δ_x values according to the γ_x, u, t coordinates of the sample, and compare the result against zero. However, this would be quite expensive, requiring four trilinear interpolations per sample in total (accounting for minimum/maximum horizontal/vertical bounds).

2.1 Per-tile slicing

A better strategy is to amortize part of the computation by extracting a slice of the γ_x, u, t volume that corresponds to a given pixel tile, as illustrated in Figure 2. By fixing γ_x, we obtain a bilinear slice in u, t space (Figure 3). An obvious problem is that the pixel tile does not correspond to a single γ_x value but a (small) range according to tile extents. This can be remedied by evaluating the corners of the bilinear slice at both minimum and maximum γ_x of the tile and taking the minimum or maximum result according to the direction of approximation.

This can be further optimized by noting that depending on the sign of the slope of δ_x along γ_x (i.e., $\partial\delta_x(\gamma_x, u, t)/\partial\gamma_x$) for any given u, t corner, we always pick either the value at either the smaller or greater γ_x, because the slope is constant when u and t are fixed. Therefore, assuming that the tile size is fixed, we can bias the δ_x values of each γ_x-oriented edge of the volume so that interpolating at, e.g., the center of tile always gives the desired minimum/maximum result for that corner.

With this per-primitive optimization, we can extract the four conservatively correct slices for a given pixel tile using 16 one-dimensional linear interpolations in total. Testing whether the 5D coverage test of an individual sample can be skipped requires four two-dimensional bilinear interpolations, one for each of the minimum/maximum horizontal/vertical bounds.

In addition to the bounding test, we calculate an approximate t span for detecting when the primitive may be inside the view frustum, and use this to cull samples with t outside this range. This is identical to the previous method, and improves efficiency in cases where geometry lies behind the camera for some duration of the frame time.

2.2 Determining γ range

The ranges for γ are obtained in a similar fashion as in the previous method, i.e., from the extents of the screen-space bounding rectangle for the primitive. These ranges have to cover the entire u, v, t range. As before, we need to detect where the primitive may cross the camera plane and extend the γ ranges to the edges of the screen in those directions. In the previous paper, we described a sweep-line algorithm for detecting these crossings, but

Scene	Bbox scan	[Laine et al. 2011]	**New method**
CONAN	23.6	23.6	23.6
motion	2.7	23.7	23.7
motion $\times 2$	1.3	24.0	24.0
defocus	1.7	23.1	23.6
defocus $\times 2$	0.7	21.9	23.5
both	0.7	5.6	23.7
both $\times 2$	0.4	2.9	23.7

Table 1: *Sample test efficiency results in the* CONAN *test scene (see the cited paper for images). The top row refers to static case without motion or defocus blur. There is marked improvement in cases with both motion and defocus blur occurring at the same time (the bottom two rows).*

Figure 4 shows an equivalent and conceptually simpler method. Whereas the previous method required separate bounding rectangles for minimum/maximum lens corners, we only need one conservative bounding rectangle.

3 Results

Table 1 gives the sample test efficiency (STE) percentages in one of the test scenes of the previous paper using the naive brute-force method, the previously presented method, and the new method described in this paper. It can be seen that the STE of the new method does not depend on the amount of motion or defocus blur, even when they occur at the same time.

References

LAINE, S., AILA, T., KARRAS, T., AND LEHTINEN, J. 2011. Clipless dual-space bounds for faster stochastic rasterization. *ACM Trans. Graph. 30*, 4, 106:1–106:6.