JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 31, 1071-1084 (2015)

Clip Space Sample Culling for Motion Blur and Defocus Blur

YI-JENG WU[1], DER-LOR WAY[2], YU-TING TSAI[3] AND ZEN-CHUNG SHIH[1]
[1]*Institute of Multimedia Engineering*
National Chiao Tung University
Hsinchu, 300 Taiwan
[2]*Department of New Media Art*
Taipei National University of Arts
Taipei, 112 Taiwan
[3]*Department of Computer Science and Engineering*
Yuan Ze University
Chungli, Taoyuan, 320 Taiwan

Motion blur and defocus blur are two common visual effects for rendering realistic camera images. This paper presents a novel clip space culling for stochastic rasterization to render motion and defocus blur effects. Our proposed algorithm reduces the sample coverage using the clip space information in camera lens domain (UV) and time domain (T). First, samples outside the camera lens were culled in stage I using the linear relationship between camera lens and vertex position. Second, samples outside the time bounds were culled in stage II using the triangle similarity in clip space to find the intersection time. Each pixel was computed within two linear bounds only once. Our method achieves good sample test efficiency with low computation cost for real-time stochastic rasterization. Finally, the proposed method is demonstrated by means of various experiments, and a comparison is made with previous works. Our algorithm was able to handle these two blur effects simultaneously and performed better than others did.

Keywords: motion blur, defocus blur, stochastic rasterization, sample test efficiency (STE), focal depth

1. INTRODUCTION

Real photographs and videos often contain motion blur and defocus blur. Motion blur is caused when objects move during finite shutter time. Defocus blur is an effect determined by the depth of field. Depth of field refers to the range of distance that appears acceptably sharp. It varies depending on camera type, aperture and focusing distance, although print size and viewing distance can also influence our perception of depth of field. However, traditional computer graphic rasterization is based on the pinhole camera and an extremely fast shutter, which means there is no motion blur or defocus blur in the rendered image.

These two blur effects are highly desirable features for real-time games. Several real-time rendering algorithms use point sampling on the screen, and average the resulting colors to obtain the pixel colors. However, these two blur effects are used cautiously in real-time systems due to the high cost of integrating polygon-screen coverage in space, time and lens dimensions. Some previous approaches have also estimated this integral by stochastic point sampling polygons [1, 4-6, 20] in 5-dimensional space (screen *XY*, lens position *UV*, time *T*). Möller Jacob *et al.* demonstrated motion blur caused by both

Received October 17, 2013; revised June 26 & August 9, 2014; accepted September 4, 2014.
Communicated by Yung-Yu Chuang.

translation and rotation [1]. They compared the uniform sampling with their stochastic sample results in their paper Fig. 7 using four samples per pixel. Munkberg *et al.* presented a method for rasterization-based point sampling using a time-continuous triangle depiction [1, 20], as shown Fig. 1. Their approach rendered motion blurred images with sufficient quality for real-time graphics, with only four samples per pixel. Since current GPUs already support spatial supersampling with that amount of samples, their algorithm can be integrated into an existing GPU. There are two major challenges for high performance implementation. First, high sample test efficiency is challenging, because it is hard to localize a moving or defocused polygon in 5D for generating a tight set of candidate sample points. Second, performing point-in-polygon tests in high dimensions is expensive.

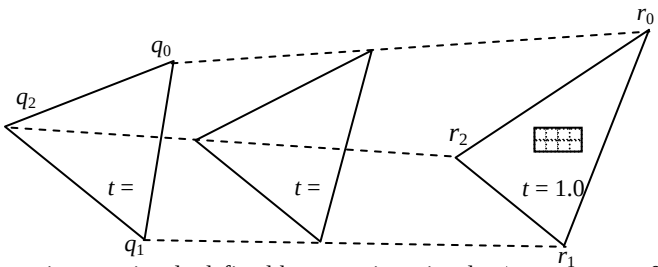


Fig. 1. A time-continuous triangle defined by a starting triangle, $\Delta q_0 q_1 q_2$, at $t = 0$, and an ending triangle, $\Delta r_0 r_1 r_2$, at $t = 1$ [1, 20].

In a traditional rasterizer, a 2D bounding box is used to bind a triangle on the screen space. Although, all of the samples are inside this bounding box, most of them are invisible. For example, a moving triangle only passes a pixel at a time dimension close to 30ms. Therefore, all of the stochastic samples far from this time will be invisible. In such cases, most of the samples failed. Good sample test efficiency (STE) is the most important tasks. STE is the percentage of point-in-polygon tests that results in hits [8]. The sample test of a stochastic rasterizer is more expensive than the use of traditional rasterization. However, stochastic rasterization does not significantly increase the cost of shading [6, 21].

This paper presents a sample visibility process that is not dependent on the amount of motion and defocus blur. Our proposed algorithm culls non-visible samples and reaches a high STE. Our method renders the camera-like images by the following steps: (1) Generating a 2D bounding box for each triangle in the geometry shader; (2) In stage I, all stochastic samples are tested whether or not they are inside the bounding boxes; (3) In stage II, only samples inside the bounding box are subjected to the actual triangle interception for coverage examination. First, a conventional camera lens is used to determine the moving triangle's maximum and minimum camera lens ranges. All samples are culled outside of this range $[u_{max}, u_{min}]$, $[v_{max}, v_{min}]$. Second, a triangular equation-like relationship in clipping space is used to cull samples with both time dimension (T) and camera lens dimension (U, V). Each pixel only needs to compute this equation once. Then, every different sample dimension can use the same test for culling. Finally, the inside samples are checked and the final image is rendered. Users can render images with motion blur only, or defocus blur only, or both effects at the same time. The major contributions of this paper are as follows:

(1) A two-stage sample culling algorithm for both motion and defocus blur stochastic rasterization with low-cost computing.
(2) A novel method providing high sample test efficiency.
(3) The algorithm can handle not only motion blur or defocus blur, but also both effects simultaneously.

2. RELATED WORKS

Sungetal and Demmers *et al.* presented an outstanding overview of previous works in motion blur and defocus blur [7, 22]. Many blurry effect renderings are usually created by post processing. These approaches work well in some areas in a frame, but frequently produce artifacts. However, there is a procedure for fast approximations which improves the performance of direct 5D integration. In the following, only related works of particular interest to our research have been reviewed briefly.

Haeberli and Akeley [10] proposed a hardware-supported method called accumulation buffer. They rendered multiple images in each buffer, with average pixel values as the final color. While this brute-force method gets a good result when a quantity of images is rendered, the computation cost is very high. The "ghosting" artifact can appear when the number of rendered images is reduced. Stochastic rasterization is another approach for rendering an image once with multiple samplings. Each sample has a unique 5-dimensional space coordinate. Fatahalian *et al.* [8] presented the Interleave UVT algorithm using as many samples as possible with the accumulation buffer method. These samples stem from different time steps and lens locations. Brunhaver *et al.* [3] proposed three rasterization designs: one is optimized for no blurred triangle micro polygons; the other is the stochastic rasterization of micro polygons with motion and defocus blur; and the third is a hybrid combination of the previous two. Their method reaches wide area and high efficiency by using low-precision operations and rasterizing pairs of adjacent triangles in parallel. Gribel *et al.* [9] provided a rasterizer that computes analytical visibility to render accurate motion blur, based on time-dependent edge equations. To keep the frame buffer requirements low, they also presented a new oracle-based compression algorithm for the time intervals.

Akenine-Möller *et al.* [1] created a bounding box for each moving triangle on the screen space. They checked all of the stochastic samples inside this bounding box. They also adapted this oriented bounding box into a 2D convex hull, which made the testing area smaller. However, most of the samples inside the bounding box did not hit the triangle. In fact, as a pixel is inside the triangle at time = 0, it seems impossible that the samples with high t values could be covered when the motion distance is large. To make the sample test more efficient, solving the sample visibility problem is a major challenge, such as how to reject samples that cannot be hit to increase the sample test efficiency (STE). They also presented a novel test for depth of field rasterization [2, 16]. Their test is based on removing half-space regions of the UV-space on the lens, from where the triangle cannot be seen through a tile of pixels. McGuire *et al.* offered [15] a hybrid algorithm for rendering approximate motion and defocus blur with a precise stochastic visibility evaluation. The algorithm manipulates dynamic triangle meshes for which per-vertex velocity or corresponding vertices from the previous frame are available.

Lee [14] presented a physically-based rendering; it outperforms previous approaches by involving a novel GPU-based tracing method. Their solution achieves greater precision than competing real-time solutions, and the experimental results are mostly indistinguishable from offline rendering. Ragan-Kelley *et al.* [22] proposed a generalized approach to decoupling shading from visibility sampling in graphics pipelines, called as decoupled sampling. Decoupled sampling enables stochastic supersampling of motion and defocus blur at reduced shading cost, as well as controllable or adaptive shading rates which trade off shading quality for performance. They provided extensions of two modern graphics pipelines to support decoupled sampling: GPU-style sort-last fragment architecture, and a Larrabee style sort-middle pipeline. Laine *et al.* [11-13] used a dual-space culling algorithm to solve the visibility test. For the defocus blur, they used the linear relationship between the lens location and the space location. By interpolating two axes-aligned bounding boxes of the triangle on lens locations (0, 0) and (1, 1), they obtained the maximum and minimum lens locations on the pixel. With motion blur, the world space affine motion is not affinal in screen space.

Munkberg and Akenine-Möller presented a hierarchical traversal algorithm for stochastic rasterization of motion blur, which efficiently reduces the number of inside tests needed to resolve spatiotemporal visibility [17]. Munkberg and Akenine-Möller [18] also proposed a hyperplane-culling algorithm. They provided an equation involving object location, lens coordinate and time coordinate to find linear bounds in UT space and VT space. They also created a hyperplane in XYUVT space for each triangle edge to cull samples. This hyperplane culling algorithm was selectively enabled when the blurry area was large enough. Our method modified the original method for creating linear bounds in clip space using the triangle similarity relationship. Our performance proved to be better than the original approach using these bounds to reduce the number of sample tests.

3. TWO-STAGE SAMPLE CULLING ALGORITHM

Most previous methods have rendered these two blur effects using stochastic rasterization. To render a motion blur image, the triangle's positions should be set up when the shutter opens ($t = 0$) and closes ($t = 1$). For each pixel covered the moving triangle tests different stochastic samples at different time coordinates. Similar to the ray intersection, the intersection is checked between the triangle and a ray shot from a pixel at time t. The color value of the intersection point is added if the intersection happens. Otherwise this sample is discarded. Finally, the average valid sample colors are computed as the pixel color.

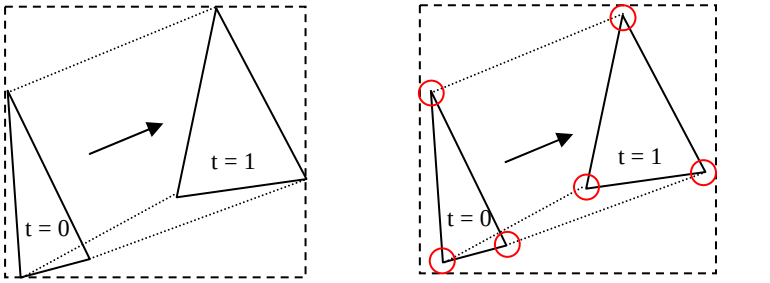
To render a defocus blur image, the lens radius is used to compute the circle of confusion (CoC). It is linearly dependent on depth w in the clip space, *i.e.* $\mathrm{CoC}(w) = a+wb$. Parameters a and b are constantly derived from the camera's aperture size and focal distance. The out-of-focus vertex position is transferred to different camera lens coordinates. A vertex position (x, y, z) in the clip space is rendered using the camera lens (u, v). The vertex position is transferred into $(x+uc, y+vc, z)$, where $c = a+zb$. For example, a pixel is displayed only by its position in lens coordinates. The intersection test is similar to the previous one. The ray checks the intersection with the triangle in the lens coordinates.

Both above mentioned blur effects are rendered with stochastic rasterization, and

the intersection test determines the triangle position in the camera lens (u, v) and time coordinate (t). However, a lot of samples are not visible for a given pixel. The traditional method requires considerable computer time and results in low sample test efficiency. This paper introduces a novel 2-stage sample culling test for stochastic rasterization to increase sample test efficiency. For each triangle, a corresponding bound geometry 2D convex hull or bounding box in the screen space is created. The motion of triangles is linear in world space in shutter time. Each pixel inside the box is sampled at different 5D sample locations (x, y, t, u, v). In the first culling stage, samples are culled if they are outside the rough camera lens bounds. In the second stage, the samples inside the camera lens bounds are tested using the triangle equation in the clip space with the point's position. Thus, the time bounds of each sample's (u, v) location is determined. Finally, a complete triangle intersection is checked and the defocus range blurry image is rendered.

3.1 Bounding Box and Defocus Range

Each triangle has a bounding box that is large enough to cover the entire triangle's motion. The most conservative bounding is the whole viewing frustum. The tightest bounding geometry is the front-facing surfaces of the triangle's swept volume, but this bound is covered on the sides. The 2D bounding box of the moving triangle at the near plane bounds the motion triangle. The bounding box is changed into a convex hull if it is in in-focus images.



(a) Without defocus (b) With defocus (Red: circles of confusion)

Fig. 2. A 2D bounding box for a moving triangle.

Fig. 2 illustrates the 2D bounding box. The vertex positions of the triangles should be calculated at the beginning and end of the shutter time. The triangle transfers into the screen space bounding box in the geometry shader. These 6 vertices are projected into screen space and get the bounding box's maximum and minimum boundary. These boundaries are changed by passing the camera lens's radius and the focus depth. The maximum circle of confusion is computed with the minimum and the maximum depth from the triangle vertices. If the triangle passes the $z = 0$ plane, the intersection point is at the near plane from the 6 triangle edges and 3 moving point paths, as shown in Fig. 3.

Fig. 4 displays the motion blur effect being added during the in-focus image rendering. The bound is tighter when using the 2D convex hull. This technique is not used for the defocus blur because the computing time of all 6 circles of confusion with different depths would be too high.

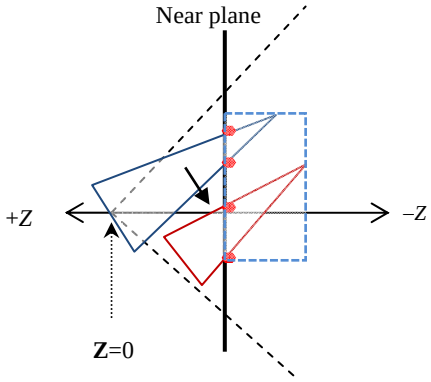


Fig. 3. A moving triangle crosses $Z = 0$ plane.

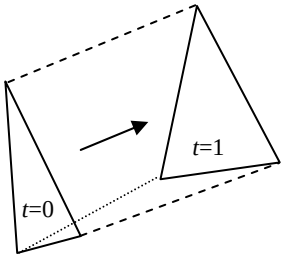


Fig. 4. A tight 2D convex hull.

3.2 Stage I

The motion of the vertex of a triangle in the clip space is a linear motion by: $p(t) = (1 - t)q + tr$, where q is the starting point and r is the ending point. For the defocus blur, the circle of confusion is also linear with time: $c(t) = a + w(t)b$. In stage I, samples outside the camera lens bounds are culled. The computation of these bounds is relatively simple because the clip-space vertex position is linear to the lens coordinates. The lens is normalized by: $(u, v) \in [-1, 1]^2$. Each pixel covered by the triangle is stochastically sampled with different parameters (t, u, v). Then, the visibility of the triangle is determined by camera lenses u and v at time t.

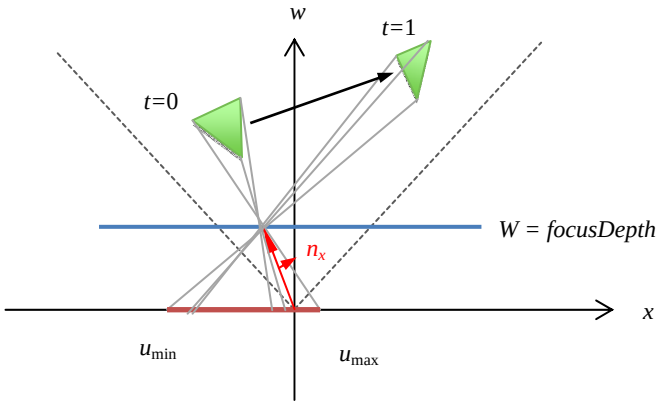


Fig. 5. A ray is shot from the camera to a screen pixel and intersects with the focal point. Six lines respectively connect a triangle vertex to the focal point and intersect with the camera lens at depth $w = 0$.

Fig. 5 illustrates how a ray is shot from a camera and passes through a screen pixel. This screen pixel is set as the focal point with the focus depth. In this xw space, all 6 triangle vertices are connected to the focal point. The 6 point positions are obtained at depth $w = 0$ using the maximum and minimum as the camera lens bounds: $u = (n_x, p(t))/(p(t).z - focusDepth)$, where $n_x = (-focusDepth, 0, focalPoint.x)$ is the normal vector of

the pixel ray. Then u is divided by the camera lens radius. Finally, the normalized camera lens bound is determined as the u dimension. The v dimension bound can be obtained in the same way with $n_x = (0, -focusDepth, focalPoint.y)$. The stochastic samples are culled with these rough camera lens bounds where (u, v) dimensions are outside the lens bounds. Therefore, the sample test efficiency is high.

3.3 Stage II

After the stage I culling, samples outside the camera lens bounds are culled. The remaining samples are examined with each (U, V) dimension to find the corresponding time bounds. All vertices are transformed with a different camera lens (U, V) relatively. For example, vertex $p = (x, y, w)$ is transformed into $p' = (x+uc, y+vc, w)$, where c is the circle of confusion of this vertex, $c = a+wb$. It is hard to compute the intersection time when a viewing ray hits the moving triangle in the screen space because the linear motion of triangle vertices is not linear to the viewing direction.

For each sample, a triangular relationship is formulated from 4 points in this clip space to obtain the intersection time. Consider a moving triangle as shown in Figure 6. First, the x coordinates of a triangle vertex at time $t = 0$ and $t = 1$ with camera lens u are computed, *i.e.*, $x_0 = x(0)+uc(0)$ and $x_1 = x(1)+uc(1)$. The x coordinates of the other 2 points (v_0 and v_1) are also found with the ray shot from the camera to the pixel which intersects with the triangle vertex's depth values. Note that the y coordinates of the 4 points can be computed in the same way.

Now, 4 points are obtained in the xw clip space: two with camera lens (u, v) and their own circle of confusion $c(t)$ when the shutter is opened and closed. The other 2 points are the viewing ray's intersection points at the same depth values of the moving triangle vertex at time $t = 0$ and $t = 1$. Then, two similar triangles are found, where their corresponding edges are proportional in length:

$$\frac{|v_0 - x_0|}{|v_1 - x_1|} = \frac{|I - x_0|}{|I - x_1|}. \tag{1}$$

Then, intersection time t can be obtained as follows:

$$\frac{|v_0 - x_0|}{|v_1 - x_1|} = \frac{t}{1-t}. \tag{2}$$

By substituting x_0 and x_1, Eq. (2) can be rewritten as:

$$t = \frac{v_0 - x(0) - u \cdot c(0)}{(x(1) - v_1 - x(0) + v_0) + u \cdot (c(1) - c(0))}. \tag{3}$$

This is a rational function in u, and this function is also monotone; it can be bounded by two linear functions as shown in Fig. 7 (a). The two linear bounds are computed as follows:

$$\begin{cases} Green: t_{\max}(u) = t(-1) + \dfrac{u+1}{2} \cdot (t(1) - t(-1)) \\ Red: t_{\min}(u) = t(0) - \dfrac{\partial t}{\partial u}(0) + (u+1) \cdot \dfrac{\partial t}{\partial u}(0) \end{cases} \tag{4}$$

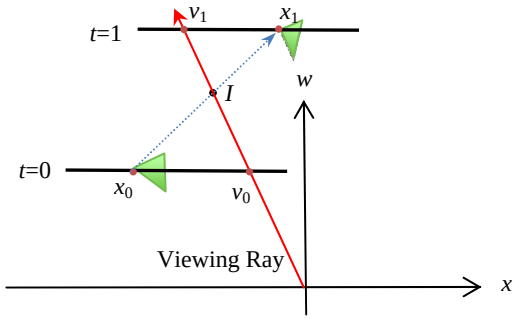


Fig. 6. Two similar triangles from x_0, x_1, v_0, v_1 and I in the xw clip space; x_0 and x_1 are the x coordinates of a moving triangle vertex at time $t = 0$ and $t = 1$; and v_0 and v_1 are the x coordinates of the viewing ray's intersection points at the same depth values of the moving vertex at time $t = 0$ and $t = 1$.

All 3 pairs of triangle vertices will respectively have 2 linear bounds. These linear bounds are then combined into only 2 bounds: a maximum bound $[t_{max}(-1), t_{max}(1)]$ and a minimum bound $[t_{min}(-1), t_{min}(1)]$. All stochastic samples are tested by the maximum and minimum bounds at the same pixel (same viewing direction), as shown in Fig. 7 (b). The sample is culled if the time dimension is larger than the maximum bound or smaller than the minimum bound. That is, only two bounds are computed for each pixel.

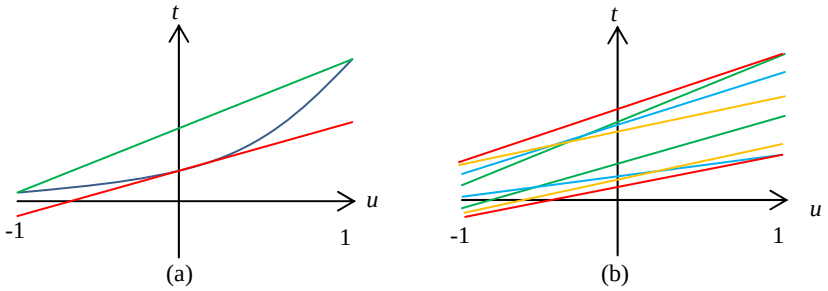


Fig. 7. (a) The rational function is bounded by two linear functions; (b) Combining all 3 pairs of linear bounds into 2 bounds (red lines).

3.4 Controllable Defocus Blur

People often want to control the depth of field parameters empirically, such as limiting foreground blur or extending the in-focus range, while preserving the foreground and background. Munkberg *et al.* [19] proposed a user controllable defocus blur method. This paper modified the circle of confusion by limiting its size near the focus depth:

$$c = \begin{cases} 0, & w - focusDeth < \varepsilon \\ a + wb, & \text{otherwise} \end{cases}. \tag{5}$$

It is assumed that the clip space circle of confusion radius is linear in the interior of

a triangle. It is easy to increase the focus range or decrease the foreground blur. If the triangle vertex is inside the extended focus range, all samples pass the stage 1 test. Set $u = v = 0$ is used to compute the stage 2 time bounds. Finally, the circle of confusion is modified when doing the triangle intersection test to render the extended focus range blurry image.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

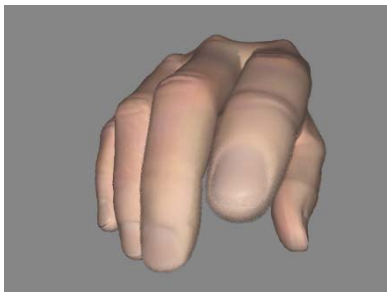
4.1 Implementation Detail

Our method was implemented using OpenGL with GLSL (OpenGL Shading Language). All vertex attributes were sent to the geometry shader. Each triangle was converted into a 2D bounding box or a 2D convex hull. Since this method destroys the triangle's structure, the positions of the original 6 vertices should be stored as output variables. Besides, each vertex CoC is packed with other vertex attributes from a previous shader to every emitted vertex, and each pixel inside the 2D bounding box is executed in pixel shader. The stochastic sample buffer was created in a CPU host program and sent to the pixel shader as a texture. This buffer had 3 channels at time t and camera lens u, v. The time values were uniform in [0, 1] as floating points. The camera lens values were uniform in [−1, 1]. The buffer size was 128*128. Each pixel took the stochastic samples from this texture. The use of an OpenGL Frame Buffer Object enabled buffer data in the GPU memory to be accessed immediately after the previous pass.

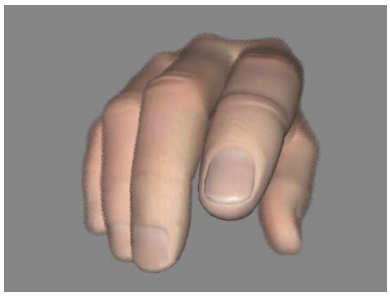
A multi-sample antialiasing (MSAA) was adopted to share geometric data with GLSL and OpenGL. For each pixel, a bit mask was used to store the hit or miss of the samples. gl_SampleMask was used as the bit mask in OpenGL. Coverage for the current fragment became the logical AND of the coverage mask and the output gl_SampleMask. On the other words, setting a bit in gl_SampleMask to zero while the corresponding sample is considered uncovered for multi-sample fragment operations. Finally, the pixel was set by average values whose corresponding mask bit was equal to one. Otherwise, this pixel was discarded when the bit mask was zero. If the stochastic sample passed the culling test and the triangle intersection test, the pixel color was rendered with the corresponding texture coordinate or the corresponding color.

4.2 Experimental Results

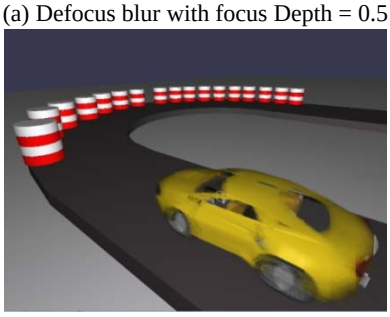
Experimental results were carried out on an Intel Core i7 950 CPU 3.07GHz and NVIDIA GeForce GTX 580 video card. Our algorithm was compared with two other culling methods: the traditional culling algorithm with bounding box and Laine's dual space culling algorithm [11]. Fig. 8 shows all experimental results rendered at 1024 × 768 pixels in real-time frame rates with 16 samples per pixel. Fig. 8 (a) shows a hand with focus depth 0.5, while the focus depth of Fig. 8 (b) is 0.3. Fig. 8 (c) displays a car with its motion blurred. Fig. 8 (d) illustrates a dragonfly in flight with the camera's motion blur. Fig. 8 (e) demonstrates the fairy forest with both motion blur and defocus blur. Fig. 8 (f) also illustrates the image with the enlarged focus range in detail. By increasing the focal range, the whole fairy face can be seen clearly.



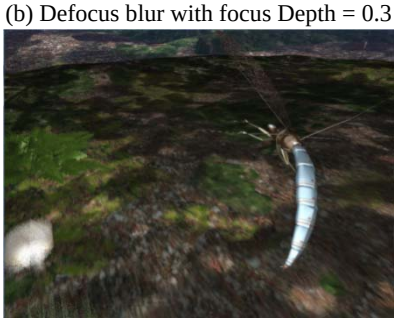
(a) Defocus blur with focus Depth = 0.5



(b) Defocus blur with focus Depth = 0.3



(c) **A** racing car with motion blur



(d) A flying dragonfly with camera motion blur

(e) Fairy forest with both motion and defocus blur

(f) Fairy forest with enlarged focus

Fig. 8. All results are 1024×768 pixels in real-time frame rates with 16 samples per pixel.

Table 1 presents the results of the STE and computation time comparison between our method and the other two culling tests. The number of triangles in the first 3D scene "Hand" is 15k; Car is 54k triangles; Dragonfly and Fairy forest are about 85k and 170k triangles, respectively. High STE means the quality is better. The STE of our method is double the Bbox in rendering the defocus blur. In rendering motion blur and both blurs, our method's STE is larger than the Bbox's. The results were rendered with motion or defocus blur only, and our algorithm's STE was almost the same as Laine's Dual space method because our stage I and II culling algorithms were done separately. However, our algorithm's efficiency was superior to Laine's method when both effects were rendered simultaneously. Table 1 also shows the computation time for each scene in milliseconds. Our algorithm required a little more computation time than Laine *et al.*'s when rendering large motion scenes. However, our method performed better when both effects were rendered.

Table 1. Sample test efficiency(1) and computation time(2).

Scene (Triangle Mesh)	Bbox		Laine *et al.* [11]		Our method	
	STE	Time	STE	Time	STE	Time
Fig. 9 (a,b): Hand (15k)						
Defocus blur	12%	135 ms	29%	131 ms	29%	123 ms
Defocus blur x2	10%	140 ms	28%	133 ms	27%	126 ms
Fig. 9 (c): Car (54k)						
Motion blur	9%	178 ms	28%	156 ms	27%	158 ms
Motion blur x2	4%	185 ms	25%	172 ms	26%	166 ms
Fig. 9 (d):Dragonfly (85k)						
Motion blur	3%	156 ms	23%	140 ms	22%	136 ms
Motion blur x2	1%	161 ms	24%	148 ms	24%	138 ms
Fig. 9 (e):Fairyforest (170k)						
Both blur	0.4%	212 ms	5%	183 ms	19%	175 ms
Both blur x2	0.2%	222 ms	2%	202 ms	13%	188 ms

Note: (1) STE results with 16 samples. (Higher is better)
(2) Computation time results with 16 samples. (Lower is better)
(3) x2 means the shutter time was doubled, and in case of defocus theaperture radius was doubled.

4.3 Discussion

It is also worth noting that parts of our method are related to the approach of Munkberg and Akenine-Möller [18]. Specifically, bounds in the stage II of our method are similar to the linear bounds in [18]; however, we chose to derive bounds on time, instead of on camera lens. The computational cost of Eq. (4) is thus slightly less than the bounds in [18], due to some trivial computation for the partial derivative of t at $u = 0$ (or $v = 0$). Specifically, this will save about six floating-point multiplications for a moving triangle when compared to bounds on camera lens. For complex dynamic scenes with a large number of moving triangles, this reduction will substantially improve the rendering performance.

Moreover, our method additionally integrates controllable defocus blur for more flexible depth-of-field effects, which is not supported in [18]. Note that tile-based culling is currently not allowed in our implementation, but it does not conflict with our method and can be seamlessly integrated if needed. We also do not consider the edge equation bound in [18], since it is computationally more expensive than our linear bounds. We believe that our method is more GPU-friendly and results in a better trade-off between rendering performance and image quality.

5. CONCLUSIONS AND FUTURE WORK

This paper proposed a 2-stage culling algorithm for stochastic rasterization. This proposed novel method computed the time bounds with a relative camera lens in clip space, and used these bounds to cull and get a high STE. Samples outside the camera lens were culled in stage I using the linear relationship between camera lens and vertex position. Samples outside the time bounds were culled in stage II using the triangle similarity in clip space to find the intersection time. Each pixel was computed within two

linear bounds only once. Our algorithm was able to handle motion blur and defocus blur simultaneously.

However, some improvements will be needed in the future, such as determining the bounding geometry more easily and accurately, reducing the execution time, increasing the test range from a per pixel test to a per tile test and accurately finding samples with a higher STE. The result of stage I was used to generate relative stochastic samples with different camera lenses. The result of stage II was used to set each previous sample's time dimension. These samples would hit the triangle with higher probability than fully stochastic samples.

REFERENCES

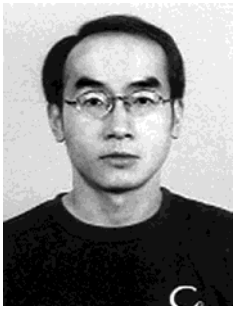
1. T. Akenine-Möller, J. Munkberg, and J. Hasselgren, "Stochastic rasterization using time-continuous triangles," in *Proceedings of International Conference on Graphics Hardware*, 2007, pp. 7-16.
2. T. Akenine-Möller, R. J. Toth, Munkberg, and J. Hasselgren, "Efficient depth of field rasterization using a tile test based on half-space culling," in *Proceedings of Computer Graphics Forum*, Vol. 31, 2012, pp. 3-18.
3. J. Brunhaver, K Fatahalian, and P. Hanrahan, "Hardware implementation of micropolygon rasterization with motion and defocus blur," in *Proceedings of International Conference on High-Performance Graphics*, 2010, pp. 1-9.
4. R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," *ACM Transactions on Graphics*, Vol. 18, 1984, pp. 137-145.
5. R. L. Cook, "Stochastic sampling in computer graphics," *ACM Transactions on Graphics*, Vol. 5, 1986, pp. 51-72.
6. R. L. Cook, L. Carpenter, and E. Catmull, The reyes image rendering architecture," *ACM Transactions on Graphics*, Vol. 21, 1987, pp. 95-102.
7. J. Demers, "Depth of field: A survey of techniques," *GPU Gems*, 2004, pp. 375-390.
8. K. Fatahalian, E. Luong, S. Boulos, K. Akeley, W. R. Mark, and P. Hanrahan, "Data-parallel rasterization of micropolygons with defocus and motion blur," in *Proceedings of High Performance Graphics*, 2009, pp. 59-68.
9. C. J. Gribel, M. Doggett, and T. Akenine-Möller, "Analytical motion blur rasterization with compression," in *Proceedings of International Conference on High-Performance Graphics*, 2010, pp. 163-172.
10. P. Haeberli and K. Akeley, "The accumulation buffer: hardware support for high-quality rendering," in *Proceedings of ACM SIGGRAPH*, 1990, pp. 309-318.
11. S. Laine, T. Aila, T. Karras, and J. Lehtinen, "Clipless dual-space bounds for faster stochastic rasterization," *ACM Transactions on Graphics*, Vol. 30, 2011, Article 106.
12. S. Laine and T. Karras, "Efficient triangle coverage tests for stochastic rasterization," Technical Report NVR-2011-003, NVIDIA, 2011.
13. S. Laine and T. Karras, "Improved dual-space bounds for simultaneous motion and defocus blur," Technical Report NVR-2011-004, NVIDIA, 2011.
14. S. Lee, E. Eisemann, and H.-P. Seidel, "Real-time lens blur effects and focus control," *ACM Transactions on Graphics*, Vol. 29, 2010, Article 65.

15. M. McGuirey, E. Enderton, P. Shirley, and D. Luebke, "Real-time stochastic rasterization on conventional GPU architectures," in *Proceedings of International Conference on High Performance Graphics*, 2010, pp. 173-182.
16. J. Munkberg and T. Akenine-Möller, "Backface culling for motion blur and depth of field," *Journal of Graphics, GPU, and Game Tools*, Vol. 15, 2011, pp. 123-139.
17. J. Munkberg, P. Clarberg, J. Hasselgren, R. Toth, M. Sugihara, and T. Akenine-Moller, "Hierarchical stochastic motion blur rasterization," in *Proceedings of International Conference on High-Performance Graphics*, 2011, pp. 107-118.
18. J. Munkberg and T. Akenine-Möller, "Hyperplane culling for stochastic rasterization," in *Proceedings of International Conference on Eurographics*, 2012, pp. 105-108.
19. J. Munkberg, R. Toth, and T. Akenine-Möller, "Per-vertex defocus blur for stochastic rasterization," in *Proceedings of Computer Graphics Forum*, Vol. 31, 2012, pp. 1385-1389.
20. J. Munkberg, "Toward a blurry rasterizer," *ACM SIGGRAPH Course, Beyond Programmable Shading*, 2011.
21. J. Ragan-Kelley, J. Lehtinen, J. Chen, M. Doggett, and F. Durand, "Decoupled sampling for graphics Pipelines," *ACM Transactions on Graphics*, Vol. 30, 2011, Article 17.
22. K. Sung, A. Pearce, and C. Wang, "Spatial-temporal antialiasing," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 8, 2002, pp. 144-153.

Yi-Jeng Wu (吳怡正) received the BS degree in Computer Science and Information Engineering from Taipei Municipal University of Education in 2011, MS degree in Computer Science from National Chiao Tung University in 2013. Now, He is an Engineer in Via Technologies. His current research interests are graphics driver.

Der-Lor Way (魏德樂) received the BS degree in Computer Science from Soochow University in 1986, MS degree in Computer Science from Chung-Yuan Christian University in 1988, Ph.D. degree in 2005 in Computer Science from National Chiao Tung University. He stayed in Industrial Technology Research Institute (ITRI) to research multimedia and virtual reality for 11 years. Currently, he is an Associate Professor in the Taipei National University of Arts, Taiwan. His research interests are in the area of non-photorealistic rendering, digital art and virtual reality.

Yu-Ting Tsai (蔡侑庭) received the B.S. and M.S. degrees in Electronics Engineering from National Chiao Tung University, Taiwan, in 2000 and 2002, respectively, and the Ph.D. degree in Computer Science from National Chiao Tung University, Taiwan , in2009. Currently, he is an assistant professor in the Department of Computer Science and Engineering at Yuan Ze University. His research interests include computer graphics, computer vision, machine learning, and signal processing.



Zen-Chung Shih (施仁忠) received the BS degree in Computer Science from Chung-Yuan Christian University in 1980, MS degree in 1982 and Ph.D. degree in 1985 in Computer Science from the National Tsing Hua University. Currently, he is a Professor in the Department of Computer and Information Science at National Chiao Tung University in Hsinchu. His current research interests include procedural texture synthesis, non-photorealistic rendering, global illumination, and virtual reality.