Utilizing Jump Flooding in Image-Based Soft Shadows *

Guodong Rong and Tiow-Seng Tan
School of Computing
National University of Singapore
{rongguod | tants}@comp.nus.edu.sg

ABSTRACT

This paper studies the usage of the GPU as a collection of groups of related processing units, where each group *communicates* in some way to complete a computation efficiently and effectively. In particular, we use the GPU to perform jump flooding to pass information among groups of processing units in the design of two simple real-time soft shadow algorithms. These two algorithms are purely image-based in generating plausible soft shadows. Their computational costs depend mainly on the resolution of the shadow map or the screen. They run on an order of magnitude faster than existing comparable methods.

Categories and Subject Descriptors

I.3.1 [**Computer Graphics**]: Graphics processors; I.3.3 [**Computer Graphics**]: Bitmap and framebuffer operations; I.3.7 [**Computer Graphics**]: Color, shading, shadowing, and texture

General Terms

Algorithms, Performance, Experimentation

Keywords

Game programming, interactive application, hard shadow, programmable graphics hardware, penumbra map, parallel prefix

1. INTRODUCTION

The modern programmable GPU is generally viewed as a powerful processor because of its many processing units working simultaneously. Most published works on GPU computations capitalize on using each processing unit sophisticatedly but more or less individually to perform some tasks [15]. On the other hand, in the history of parallel computation (with the GPU as one of its present form), we note that there are many sophisticated communication patterns that assist in computations. For example, Ladner and Fischer [13] present the parallel prefix to solve the prefix problem with associative operation. Motivated by these works, we attempt to seek a good understanding of the computational possibilities with sophisticated communication patterns among processing units in the GPU.

In this aspect, there are a few different flavors of recent works such as the bitonic sorting [7], the N-buffer [10], and the jump flooding algorithm [17]. The last paper is most relevant to our work here. It successfully uses the GPU to efficiently compute very accurate 2D Voronoi diagrams and distance transforms. This is an exciting attempt in understanding and using communication similar to that in the parallel prefix to design non-trivial algorithms. That work relies heavily on the empty circle property of Voronoi diagram [5] to be able to quantify the quality of the outputs. However, it was not known whether the jump flooding algorithm could still be useful to other computation problems that might not have a special property to guarantee the quality of the results. Our work here ascertains that the jump flooding algorithm is indeed useful to real-time shadow generation.

In interactive applications, shadows can enhance the realism of the rendered images by providing information about positions and relationships among objects, and contribute to over all good immersing experience. Real-time shadow (especially soft shadow) generation costs a significant amount of CPU and GPU cycles. Real-time shadow algorithms can be categorized into object-based and image-based [12]. Object-based algorithms must deal with all the geometries in the scene. So they do not scale well for complex scenes. In contrast, the speed of image-based algorithms is less dependent on the complexity of the scene. To this end, this paper has the following two major contributions:

- It demonstrates ways to use jump flooding on a computational problem that has no known property to guarantee the quality of the outputs. This advances the understanding of using the communication pattern similar to that in the parallel prefix to handle yet another type of non-associative operation.

- It presents two simple image-based soft shadow algorithms JFA-L (jump flooding in light space) and JFA-E (jump flooding in eye-space) that are of an order of magnitude faster than existing comparable algorithms. They achieve good frame rates with the current GPU

*The project website is *http://www.comp.nus.edu.sg/~tants/softShadow.html*

VRST'06, November 1–3, 2006, Limassol, Cyprus.

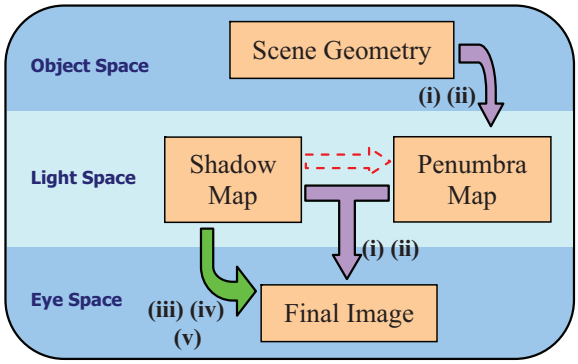


Figure 1: The computational mechanisms of the recent soft shadow algorithms based on shadow mapping. The dashed red arrow is our JFA-L's way of generating penumbra map directly from shadow map to assist the computation of soft shadows. Our JFA-E also follows the approach indicated by the green arrow (same as iii, iv and v).

for very complex scenes of over hundreds of thousands of triangles.

This paper is organized as following. Section 2 briefly surveys the state-of-the-art soft shadow generation algorithms. Section 3 outlines the jump flooding algorithm. Section 4 and Section 5 describe JFA-L and JFA-E respectively. Section 6 discusses our soft shadow results. Finally, Section 7 concludes the paper with possible future work.

2. PREVIOUS WORK ON SOFT SHADOW

We briefly review several algorithms closely related to the work in this paper. They are summarized in Figure 1. All of them are based on shadow mapping [21], and use shadow maps obtained from a single sample on the light (normally the center of the light). For other soft shadow algorithms, please see the survey by Akenine-Möller and Haines [1] and Hasenfratz et al. [12].

Wyman and Hansen's penumbra maps [22] ((i) in Figure 1) and Chan and Durand's smoothies [8] ((ii) in Figure 1) both build some additional geometries in object space and then render them to generate a penumbra map. This penumbra map, together with the shadow map, can be used to generate the final soft shadows. Both algorithms are object-based, and require the objects in the scene to be polygonal and do not scale well for complex scenes.

Arvo et al. [2] use shadow map to generate an image with hard shadows, and then use flood filling method to spread out the occluding information from the boundaries of hard shadows to obtain soft shadows ((iii) in Figure 1). Besides some technical problems of the method (discussed in Section 5), their approach of flooding is slow for real-time applications.

Uralsky [18] adapts percentage closer filtering method [16] to generate soft shadow directly from shadow map ((iv) in Figure 1). The algorithm uses Monte Carlo sampling and thus generates noisy effect in the results. Uralsky uses back faces (with respect to the light) to build the shadow map. This constrains the objects to be closed manifolds. It can also generate erroneous self shadow for thin objects and brighter regions for concave objects (see also [19]). Part III of the accompanying video (downloadable at the project website) demonstrates these issues. Furthermore, the optimization in Uralsky's algorithm can result in dark spots in small fully lit areas.

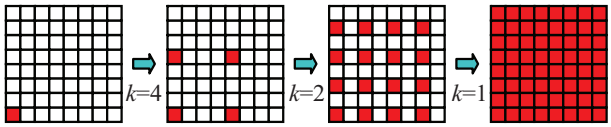


Figure 2: The process of JFA for a 8×8 grid with a single seed at the bottom left corner of the grid.

Brabec and Seidel [6] also generate soft shadow directly from shadow maps ((v) in Figure 1). But the searching for the nearest silhouette pixels is too slow for real-time applications, especially for wide penumbra. And their use of object identities prohibits self shadows.

We learned recently of the work by Guennebaud et al. [11]. Like us, they also developed another purely image-based method to generate real-time soft shadows.

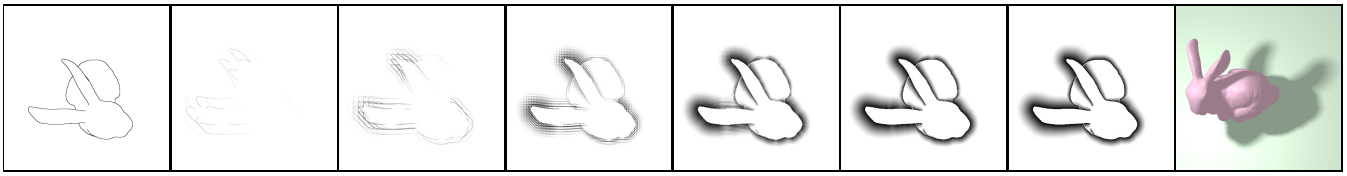
3. JUMP FLOODING ALGORITHM

Suppose we have a pixel designated as a seed in an $n \times n$ grid, and we want to propagate its information to all the other pixels. One simple way is to use the standard flood filling algorithm. In every round of the flooding, a pixel at (x, y) obtains information from its (maximum) 8 neighboring pixels at $(x+a, y+b)$ where $a, b \in \{-1, 0, 1\}$. We call this a round with *step length* of 1. The number of rounds of such a method is linear to the resolution of the grid.

Jump flooding algorithm (JFA) can reduce the number of rounds to logarithmic, by varying the step lengths. The step lengths in these rounds are in the order: $2^{\lceil \lg n \rceil - 1}$, $2^{\lceil \lg n \rceil - 2}$, ..., 1. In a round with step length of k, a pixel at (x, y) obtains information from other (maximum) 8 pixels at $(x + a, y + b)$ where $a, b \in \{-k, 0, k\}$. Figure 2 demonstrates the process of JFA for a 8×8 grid with a single seed.

When executing JFA with more than one seed in a round, a pixel p may receive (information of) more than one seed from other pixels. Among all the seeds arriving at p, p has to decide a seed s to record to pass on to other pixels in subsequent rounds. The other seeds s' that arriving at p but not recorded by p are said to be *killed* by s at p. In the case of using JFA to compute Voronoi diagram, p records the seed s closest to p among all the seeds known so far by p.

Our soft shadow algorithms JFA-L (Section 4) and JFA-E (Section 5) also rely heavily on propagating occluders' (seeds') information stored in pixels on silhouettes or boundaries to other pixels to calculate their intensities. Here, we use the calculated intensity to determine whether an occluder is to be recorded at a pixel p. We assume the light source is a circle for our following discussion. For p receiving information of an occluder o, it calculates the intensity by shooting a ray from p to o (as in [2]). If the ray does not intersect the light source, p is considered to be either in an umbra region or in a fully lit region based on whether it is in hard shadows, and does not propagate o to other pixels in subsequent rounds. On the other hand, if the ray intersects the light source at a point q, it first estimates the intensity



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 3: JFA-L algorithm. (a) boundaries (silhouette pixels) in shadow map; (b)-(g) steps of jump flooding with step lengths of 32, 16, 8, 4, 2 and 1; and (h) final result generated using the penumbra map (g) and the shadow map. The 6 rounds of jump flooding in (b)-(g) generate a penumbra region with maximum width of 63 pixels. This number of rounds (instead of a complete 9 rounds of JFA for this example of 512×512 image) is generally enough for most applications.

based on the proportion of the light source visible to p by partitioning the light source with the line orthogonal to cq and passing through q, where c is the center of the light. For an outer (inner, respectively) penumbra point p, if the intensity is smaller (larger, respectively) than that already stored in p, p then keeps the newly calculated intensity, and goes on to propagate o to other pixels in subsequent rounds.

Such propagating of occluders' information to calculate intensities, when implemented as jump flooding (with occluders replacing seeds) to gain good speed, is unclear to work at all. Unlike the case of Voronoi diagram computation with the empty circle property, we do not know of any property here to provide a guarantee of good soft shadows with such a non-trivial communication pattern. We address in the next two sections the situations presented by JFA-L and JFA-E to generate plausible soft shadows.

4. JUMP FLOODING IN LIGHT SPACE

The novelty of JFA-L is to efficiently generate a penumbra map directly from shadow map by using JFA in light space (as shown by the red dashed arrow in Figure 1). It uses hard shadows to approximate umbra regions. So all the penumbra regions can be seen by the center of the light, and thus can be stored into a texture called *penumbra map*. It only generates *outer penumbrae*, i.e. penumbrae out of hard shadows.

JFA-L algorithm includes four major phases. The second phase and each round of jump flooding in the third phase are achieved with the standard technique of drawing a quad of the same size as the shadow map to trigger a fragment program running on every pixel.

4.1 JFA-L Algorithm

Building Shadow Map. The first phase of our algorithm is almost identical to that of the standard shadow mapping, except that we also store the light space coordinate of every pixel in addition to the depth value. To increase the resolution of the useful part of the shadow map, one can use, for example, trapezoidal shadow map approach [14].

Locating Occluders. For every pixel in the shadow map, we test its eight neighboring pixels. If the depth of one or more neighboring pixel is larger than its depth (greater than a certain threshold), it is marked as a *silhouette pixel*. These silhouette pixels having information of their light space coordinates behave as occluders for other pixels in penumbra regions.

Generating Penumbra Map. To compute a penumbra map, we spread out the coordinate information of occluders from the silhouette pixels to other pixels using jump flooding (see Figure 3 and Part I of the accompanying video for our implementation of jump flooding with 6 rounds). When a pixel receives the coordinate of an occluder from flooding, it shoots a ray from itself to the occluder to calculate the intensity as mentioned in Section 3. Among all occluders received, a pixel keeps the one computed with minimum intensity value and then passes it on to other pixels in subsequent rounds. Here, the calculation of the intensity can be pre-computed and stored in a texture indexed by the intersection point of the ray with the light.

Generating Final Image. In the last phase, we render the scene from the eye position. For every pixel, we transform it to the light space. First, we check whether the pixel is in hard shadows. If so, the intensity of this pixel is set to 0. Otherwise, we check whether there is a coordinate value stored in the corresponding pixel in the penumbra map. If so, this pixel is in a penumbra region, and its intensity is then calculated by shooting a ray from this pixel to the respective coordinate value in the penumbra map. Note that we do not directly use the intensity stored in the penumbra map as that can result in blocky soft shadows when the shadow map does not have a good resolution. If both of the above fail, the pixel is fully lit and thus has a normal intensity of 1.

4.2 Analysis

An immediate limitation of JFA-L is that it does not generate inner penumbrae, i.e. penumbrae inside of hard shadows. We have investigated many attempts (such as the second layer shadow maps [20, 3]), but found no suitable supplement to the JFA-L algorithm to generate inner penumbrae without introducing new problems. Nevertheless, from our experiment and our analysis below, JFA-L remains very useful in generating convincing soft shadows for real-time purposes.

JFA-L does not calculate exact intensities at penumbra regions but only some form of approximations. This is not an issue for real-time applications, such as games, as long as the algorithm can achieve the following two goals of *parity* and *smoothness* in order to generate convincing soft shadows. On the parity, an algorithm calculates soft shadow intensity for a pixel if, and only if, the pixel is in a penumbra region. On the smoothness, calculated intensities of adjacent penumbra pixels must vary smoothly.

Parity. We have the following simple argument that when a pixel p is calculated to be in a penumbra region by JFA-L, then this is indeed so for p. Let o be the occluder received at p through jump flooding for the calculation of the intensity of p. By the fact that the ray from p to o intersects the light source, p cannot be fully lit. Also, p cannot be in an umbra region too, as our algorithm does not assign intensity to pixels in umbra regions. So, p can only be in a penumbra region, as claimed.

On the other hand, if p does not receive intensity value through jump flooding, then it is not certain that p is not in any penumbra region. When such p is indeed in the middle of a penumbra region, the undesirable visual effect is the existence of holes in the calculated penumbra region. In all our experiments, we do not observe such effects. One explanation is that penumbra regions tend to be narrow in width, thus are highly unlikely to accommodate visible holes. Another possible explanation is to view the jump flooding computation of intensity here as in some way related to the jump flooding computation of distance in [17]. This view is reasonable, but not absolute, as the intensity of a pixel is in some way proportional to the distance from the silhouette pixels in the light space. By the result of [17] that JFA computes excellent approximation to Voronoi diagram, we can expect JFA to communicate occluders to almost all, if not all, pixels (especially those nearby ones) in penumbra regions.

Smoothness. We have the following reasoning to believe that JFA-L computes smooth transition in intensity from pixel to pixel. By the way the intensity is defined, the calculated intensity with a single occluder for a continuous surface is smooth across the pixels (in the penumbra map) representing the surface.

So, let us suppose that all occluders can reach all pixels (which is clearly not true because occluders are killed along the way during jump flooding). Then, for any two neighboring penumbra pixels (technically, it should be two infinitesimally close by points), regardless of the fact that they record the same or different occluders, the calculated intensities are smooth across the pixels as we always keep in each pixel the occluder that generates the minimum intensity.

By the nature of jump flooding with many ways to reach from one pixel to another pixel throughout the rounds, many occluders have good chances to communicate to many pixels before being killed. Thus, there are enough occluders to reach enough relevant pixels, i.e. a partial fulfillment of the assumption in the previous paragraph, and these are enough to generate plausible smooth intensity across pixels. Additionally, as mentioned before, penumbra regions are narrow and near to the hard shadow boundaries, the occluders recorded for two adjacent pixels are either the same or nearby occluders, and thus generate smooth intensity across the adjacent penumbra pixels.

Flooding Experiment. To provide some confidence to the above reasoning, we use the fantasy scene (Figure 10) of over a hundred thousands of triangles to generate two series of 462 frames (as shown in Part V of the accompanying video), one using jump flooding and the other using standard flooding to generate the corresponding penumbra maps. The percentages of pixels (with respect to the total number of 430k to 940k penumbra pixels in each image) with differences in intensities (ΔI) for each frame are plotted as four curves in Figure 4(a). The four curves represent differences in intensities of below 0.05, 0.05 to below 0.1, 0.1 to below 0.2, and at least 0.2. We note that the intensities of the majority of pixels as generated by jump flooding and standard flooding are of small differences, if not the same. Figure 4(b) and 4(c) show the penumbra maps (at the pergola area) of the frame with the largest difference. We also show the frames with large differences in Part VI of the accompanying video. We do not see significant visible differences for these frames. In the experiment, we also note that the number of pixels with differences in parity between jump flooding and standard flooding is less than 0.1%, and this thus is not of significance at all.

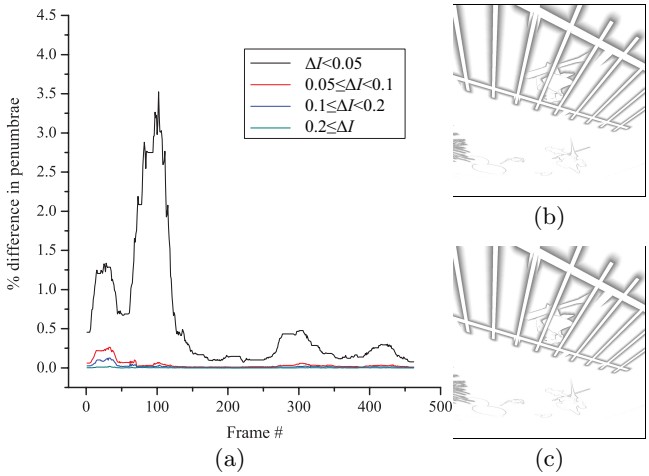


Figure 4: (a) Statistics of pixels with differences in intensities; (b) penumbra map generated by jump flooding; and (c) penumbra map generated by standard flooding.

5. JUMP FLOODING IN EYE SPACE

The novelty of JFA-E is to efficiently generate a final image with soft shadow from an image with hard shadows. This algorithm seems to follow immediately from Arvo et al.'s algorithm [2] by replacing the standard flooding with JFA. However, we need to deal with two problems. Firstly, Arvo et al.'s algorithm sometimes falsely recognizes pixels on hard shadow boundaries when using the conditions that such a pixel (1) must lie between lit and shadow regions, and (2) must be occluded by a silhouette pixel in shadow map. This is because condition (2) is not foolproof: Some pixels occluded by a silhouette pixel in shadow map may also be occluded by other objects in between them, and thus not on hard shadow boundaries. These incorrectly identified pixels lead to holes in umbra regions. Part IV of the accompanying video demonstrates this phenomenon. Secondly, we need to address an undesirable "jump too far" effect of the jump flooding process that also results in holes in umbra regions (in Section 5.2. and Figure 6). The next subsection describes the five phases of the JFA-E algorithm.

5.1 JFA-E Algorithm

Building Shadow Map. This is the same as that of JFA-L.

Generating Hard Shadows. In generating hard shad-

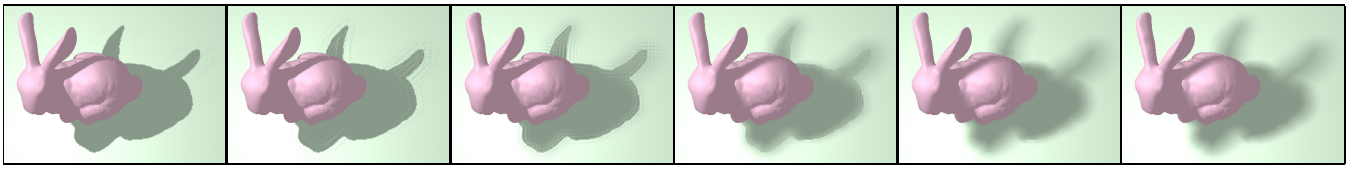


Figure 5: JFA-E algorithm. (a)-(f) are steps of jump flooding with step lengths of 32, 16, 8, 4, 2 and 1, and (f) is the final result.

ows with the standard technique, we also store the 3D coordinates of points corresponding to all pixels. Each coordinate is stored in RGB channels, and a flag indicating whether the pixel is in hard shadows in the alpha channel.

Locating Occluders. To extract the boundaries of hard shadows, we check for every pixel in hard shadows its eight neighboring pixels with a fragment program. If at least one of the neighboring pixels is lit, and if the distance (in 3D space) between the corresponding points of this lit pixel and the pixel in hard shadows is small enough, we consider this pixel as a *boundary pixel*. Once a boundary pixel is ascertained, the coordinate of the corresponding occluding silhouette pixel is also recorded for the pixel into a texture, which is the input to jump flooding.

Generating Penumbra Regions. Then, we run a fragment program with the required number of rounds of jump flooding to calculate intensities (see Figure 5 and Part II of the accompanying video for our implementation of jump flooding with 6 rounds). As in the JFA-L approach, we use a look-up texture indexed by the intersection point to obtain pre-calculated intensity values. For each pixel in a penumbra region, we keep the highest or lowest intensity value, respectively, during the flooding according to whether it is inside or outside, respectively, hard shadows.

Generating Final Image. For every pixel, the result of jump flooding in the previous phase can determine whether it is in a penumbra region. If so, its intensity in the final image is read from the output in the previous phase. Otherwise, we use the hard shadow result to determine whether it is in hard shadows. If so, its intensity in the final image is set to 0. If both of the above fail, the pixel is fully lit and thus has a normal intensity of 1.

5.2 Analysis

Jump Too Far. The advantage of JFA-E as compared to JFA-L is that it attempts to generate both the inner and outer penumbrae. However, this comes with a side effect of some parts of umbra regions to be mistaken as inner penumbrae that appeared visually as holes inside umbra regions. See Figure 6(a) for an illustration where we have the situation as seen from the eye. Pixel p' is on hard shadow boundaries, and it may jump to q' in a round with a large step length. The ray passing through point q and the occluder point e intersects with the light. So q' is treated as in a penumbra region, while it should be in an umbra region. q' may further flood its occluder to other nearby pixels. This results in a hole around point q' in the final image. Figure 6(b) gives an example in our fantasy scene with such an artifact.

We have two simple ways to reduce the occurrences of such problems. The first way is to record object identities of occluders, and to allow jump flooding of occluder only to pixels with the same object identity. (Note that this is not the same use of object identities as in [6] where self-shadowing is lost.) This effectively eliminates the jump too far problem of wrong inner penumbrae. However, this method still fails for some concave objects. The second way is a simple form of continuity check: a midpoint between the boundary of the occluder and the pixel under consideration must have penumbra intensity before the pixel under consideration can have penumbra intensity. Again, one can create an arrangement to show that this does not fully remove the jump too far problem. Nevertheless, our experiments with both of them are very positive – each manages to remove all the hole artifacts.

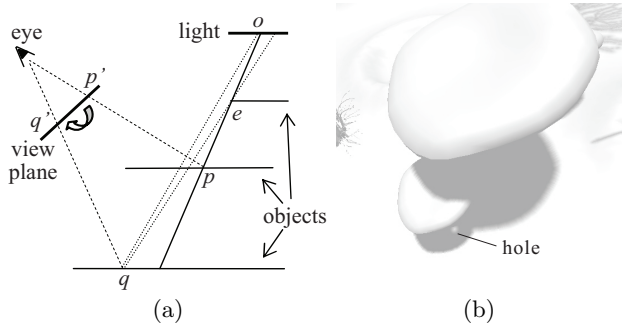


Figure 6: (a) 2D illustration of the jump too far problem; and (b) a corresponding example of an undesirable hole in the hard shadow under the mushrooms in our fantasy scene.

Parity and Smoothness. We note that the relationship of distance and intensity used in our argument for JFA-L is generally not true here for flooding in eye space. This is because the image seen by the eye is distorted by the projection. Two points may be very near to each other when seen from the light, but are actually very far apart when seen from the eye. So we cannot ascertain that the intensity is in any way related to the distance in the eye space. Nevertheless, we do not detect significant problems, as testified in the experiment discussed in the next paragraph, in using jump flooding in place of standard flooding.

Flooding Experiment. We also generate from the fantasy scene (Figure 10) the two series of 462 frames, one using jump flooding and the other using standard flooding. Figure 7(a) shows the corresponding curves in the case of JFA-E (with respect to the total number of 10k to 99k penumbra pixels in each image) as that of JFA-L in Figure 4(a). The results of the frame with the maximum difference are shown

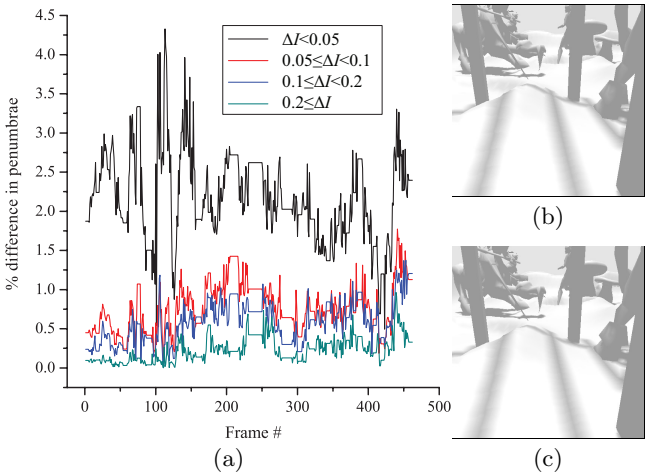


Figure 7: (a) Statistics of pixels with differences in intensities; (b) result generated by jump flooding; and (c) result generated by standard flooding.

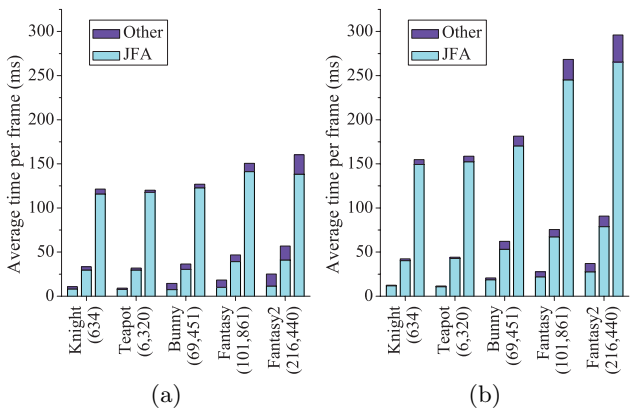


Figure 8: Comparison of the time of JFA and other parts in (a) JFA-L and (b)JFA-E. The three time bars for every model (from left to right) represent resolutions of 256×256, 512×512 and 1024×1024 of the shadow map (for JFA-L) or the screen (for JFA-E). The numbers in the parenthesis are the numbers of triangles of the models.

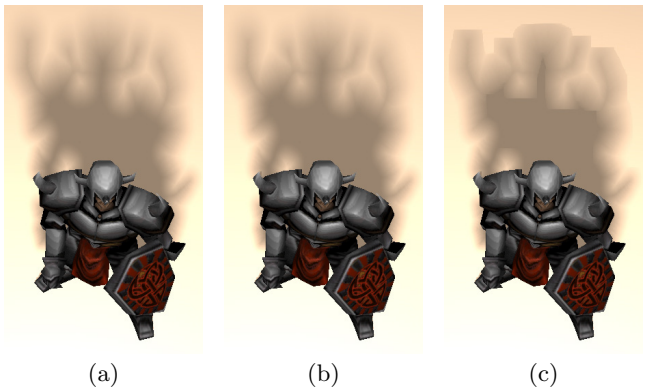


Figure 9: Soft shadows of Knight generated using (a) JFA-E with 6 passes, (b) Arvo et al's algorithm with 63 passes, and (c) Arvo et al's algorithm with 20 passes.

in Figure 7(b) and 7(c). We also show the frames with large differences in Part VI of the accompanying video. Once again there are no significant and noticeable visual differences between each pair of frames. In the experiment, we also note that the number of pixels with differences in parity between jump flooding and standard flooding is less than 6.8%.

6. EXPERIMENTAL RESULTS

We have implemented JFA-L and JFA-E with trapezoidal shadow maps [14] using Visual C++.net 2003 and Cg 1.4. The hardware platform is Pentium IV 3.0GHz, 1GB DDR2 RAM with a NVIDIA GeForce 6800 GT PCI-X, 256MB DDR3 video memory.

JFA-L Results. We have run JFA-L on five models with increasing magnitude of triangles. For every model, we use shadow maps of three different resolutions, while fixing the resolution of the screen at 512×512. See Part V of the accompanying video for the results. An output of the fantasy scene using JFA-L is shown in Figure 10(a). This scene includes a tree, a pergola, three mushrooms, a rock, a flower and four animated characters in a terrain. The Fantasy2 scene is similar to the Fantasy scene with duplicated objects to increase the number of triangles. Figure 8(a) shows the average time taken per frame. Each time bar highlights the part taken by JFA in a light color. It is clear that the time taken by JFA is mostly dependent on the resolution of the shadow map. For a same model, when we double the resolution, the number of the total pixels increases by four times and so is the time taken to perform JFA. Among different models, the time taken by JFA differs slightly as different number of pixels are active (with more complex silhouette having more) in executing the same fragment program.

JFA-E Results. Figure 8(b) shows the average time per frame of JFA-E for different models. We use screens of three different resolutions, while fixing the resolution of the shadow map at 1024×1024. See Part V of the accompanying video for results. Figure 10(b) shows an output of the fantasy scene using JFA-E. We note that JFA-E runs with 6 rounds of jump flooding on screen with resolution of 512×512 can achieve 23 frames per second for the knight model (Figure 9), as compared to Arvo et al's approach of about only 2 frames per second with 63 passes of standard flooding (to achieve similar quality as ours). When using only 20 passes as suggested in their paper, Arvo et al's algorithm can only achieve 6 to 7 frames per second, while generating results of lower quality.

JFA-L and JFA-E. By the way the intensity is calculated, both algorithms have the problem of shrinking umbra regions as that of Arvo et al.'s. This problem is observed at any cross shape shadows of two objects, when one is on top of the other; see, for example, the shadows cast by the crossing of the ear and the buttock of the bunny in Figure 3(h) and Figure 5(f).

Both algorithms rely on the quality of the shadow map. As such, when there is an artifact in the shadow map (for JFA-L) or hard shadows (for JFA-E), the corresponding soft shadows also have artifacts. In particular, when some fea-

(a) (b)

Figure 10: Results of the fantasy scene using (a) JFA-L, and (b) JFA-E.

tures of hard shadows appear and disappear, they are further exaggerated due to flooding and appear as undesirable blinking of soft shadows.

JFA-E has a problem with inner penumbrae when a small hole inside the hard shadows is changing in shape or disappearing due to the insufficient resolution of the shadow map. When this happens, we observe undesirable blinking of soft shadows – such effect does not appear in JFA-L as it has no inner penumbrae. JFA-E thus is not appropriate for scenes with small holes in hard shadows. Such blinking effects also appear when a part of the hard shadow boundaries is occluded by other objects in one frame but become visible in the next. Without this artifact, during walkthrough or flythrough, JFA-E produces more realistic soft shadows. On the other hand, we like JFA-L for it is robust in producing plausible soft shadows for scenes with complex hard shadow boundaries, and it runs faster than JFA-E.

To compare soft shadows derived from shadow volume [9] and shadow mapping, we experiment with some codes available from the web on the shadow volume approach of [4]. This algorithm is object-based and handles only objects that are polygonal and manifolds. It does not seem to scale well for complex scenes as it performs with low frame rate for scenes containing just a few thousands of triangles.

7. CONCLUDING REMARKS

This paper presents two novel and simple real-time soft shadow algorithms based on shadow mapping. In JFA-L, we use jump flooding to generate a penumbra map directly from shadow map, and then use it to generate soft shadows. In JFA-E, we improve upon Arvo's algorithm in speed and in quality of soft shadows. Both algorithms are purely image-based. They can achieve interactive speed with plausible soft shadows even for very complex scenes. One possible future work is to record more information of occluders, such as silhouette edges rather than just silhouette points, to compute more accurate intensity.

In another view, these algorithms extend our understanding in utilizing GPU for non-trivial communication among processing units that is seldom found in the present uses of GPU. Possible future work is to continue searching for the use of jump flooding or other interesting communication patterns utilizing GPU.

8. ACKNOWLEDGMENTS

We would like to thank Tobias Martin and Edwin Zeng for fruitful discussions and helpful comments on this paper. This research is supported by National University of Singapore under the grant R-252-000-216-112 and Microsoft Research Asia under the grant R-252-000-259-720.

9. REFERENCES

[1] T. Akenine-Möller and E. Haines. *Real-Time Rendering*. AK Peters, Ltd., second edition, 2002.

[2] J. Arvo, M. Hirvikorpi, and J. Tyystjärvi. Approximate soft shadows using image-space flood-fill algorithm. *Computer Graphics Forum*, 23(3):271–280, 2004. (Proceedings of Eurographics 2004).

[3] J. Arvo and J. Westerholm. Hardware accelerated soft shadows using penumbra quads. *Journal of WSCG*, 12(1):11–18, 2004.

[4] U. Assarsson and T. Akenine-Möller. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics*, 22(3):511–520, 2003. (Proceedings of ACM SIGGRAPH 2003).

[5] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[6] S. Brabec and H.-P. Seidel. Single sample soft shadows using depth maps. In *Proceedings of Graphics Interface*, pages 219–228, 2002.

[7] I. Buck and T. Purcell. A toolkit for computation on GPUs. In R. Fernando, editor, *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, chapter 37, pages 621–636. Addison-Wesley, 2004.

[8] E. Chan and F. Durand. Rendering fake soft shadows with smoothies. In *Proceedings of Eurographics Symposium on Rendering*, pages 208–218, 2003.

[9] F. C. Crow. Shadow algorithms for computer graphics. *Computer Graphics*, 11(3):242–248, 1977. (Proceedings of ACM SIGGRAPH 77).

[10] X. Décoret. N-buffers for efficient depth map query. *Computer Graphics Forum*, 24(3):393–400, 2005. (Proceedings of Eurographics 2005).

[11] G. Guennebaud, L. Barthe, and M. Paulin. Real-time soft shadow mapping by backprojection. In *Proceedings of Eurographics Symposium on Rendering*, pages 227–234, 2006.

[12] J.-M. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion. A survey of real-time soft shadows algorithms. In *Eurographics*, pages 753–774, 2003. (State of the Art Reports).

[13] R. E. Ladner and M. J. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, 1980.

[14] T. Martin and T.-S. Tan. Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of Eurographics Symposium on Rendering*, pages 153–160, 2004.

[15] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. In *Eurographics*, pages 21–51, 2005. (State of the Art Reports).

[16] W. T. Reeves, D. H. Salesin, and R. L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics*, 21(4):283–291, 1987. (Proceedings of ACM SIGGRAPH 87).

[17] G. Rong and T.-S. Tan. Jump flooding in GPU with applications to Voronoi diagram and distance transform. In *Proceedings of the Symposium on Interactive 3D Graphics and Games*, pages 109–116. ACM Press, 2006.

[18] Y. Uralsky. Efficient soft-edged shadows using pixel shader branching. In M. Pharr and R. Fernando, editors, *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter 17, pages 269–282. Addison-Wesley, 2005.

[19] M. Valient and W. H. de Boer. Fractional-disk soft shadows. In W. Engel, editor, *ShaderX^3: Advanced Rendering with DirectX and OpenGL*, chapter 5.2, pages 411–424. Charles River Media, Inc., 2005.

[20] Y. Wang and S. Molnar. Second-depth shadow mapping. Technical Report TR94-019, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1994.

[21] L. Williams. Casting curved shadows on curved surfaces. *Computer Graphics*, 12(3):270–274, 1978. (Proceedings of ACM SIGGRAPH 78).

[22] C. Wyman and C. Hansen. Penumbra maps: approximate soft shadows in real-time. In *Proceedings of Eurographics Symposium on Rendering*, pages 202–207, 2003.