A PARALLEL RENDERING APPROACH TO THE ADAPTIVE SUPERSAMPLING METHOD [a]

SAM LIN, RYNSON W. H. LAU †, XIAOLA LIN, P. Y. S. CHEUNG
Department of Electrical and Electronic Engineering,
The University of Hong Kong, Hong Kong
E-mail: wslin@eee.hku.hk

†*Department of Computing,*
The Hong Kong Polytechnic University, Hong Kong

Original z-buffer method is a very efficient method for image generation. The limitation is that it introduces aliases into the output image. Although many methods have been proposed to address this problem. Most of them suffer from requiring a large memory space, demanding for high computational power, or having some other limitations. Recently, we presented a simple anti-aliasing method based on the supersampling method. Instead of supersampling every pixel, we supersample edge pixels only. In this paper, we discuss various approaches for parallelizing the method and their effects on memory usage and performance.

1 Introduction

Real-time image rendering become more common due to the popularity of visual applications such as computer games and virtual reality. Various parallel rendering methods have been proposed. However, for a high quality image rendering applications, anti-aliasing may be required which seriously increases the amounts of data transferred between processors. Hence, the rendering performance may be degraded due to the network contention. Recently, we presented a scan-conversion method called the adaptive supersampling method [6]. The method requires minimal extra computational and memory costs to do anti-aliasing. We have simulated the algorithm and measured the memory usage among processes. In this paper, we consider different approaches to parallelize the adaptive supersampling method. We look at how these approaches affect the overall memory usage and performance of the method.

2 The Adaptive Supersampling Method

Image rendering based on the original z-buffer algorithm [2] requires to store only the color value and the depth value of the closest object at each pixel. However, having these two values alone produces aliased images. This aliasing is the result of the point sampling nature of the z-buffer method as shown in

[a] This work was supported in part by the Hong Kong CRGC grant F163-C.

(0-7803-4229-1/97/$10.00 1997 IEEE)

figure 1a. There are many methods proposed to solve the aliasing problem. The supersampling z-buffer method [4] supersamples the scene and then filters the image down into the output resolution shown in figure 1b. The problems of this method are that high memory and computational costs are required to generate the image. Another anti-aliasing method is called the A-buffer method [1]. This method basically breaks polygons into pixel fragments. The visible fragments are accumulating in a temporary buffer for hidden surface removal and anti-aliasing. Memory usage of this method depends on the complexity of the scene and hence there's no theoretical upper limit. As such, this method normally requires run-time memory allocation, which makes real-time image generation difficult. As the traditional supersampling method allocates

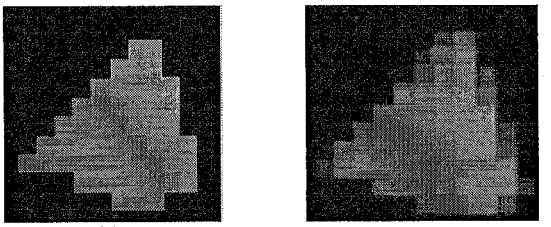

Figure 1: (a) Aliased Image. (b) Anti-aliased image.

memory to store the depth and color value of each sub-pixel, a lot of memory is needed and the actual memory usage is directly proportional to the sub-pixel resolution. However, we have notices that the aliasing problem occurs around polygon edges and lines where surfaces intersect with each other. Since these edge pixels only contribute to a small portion of the total number of pixels in the image (not exceed 20% for most of our tested images), our idea here is to do supersampling only when we need to [6]. Figure 2 shows the difference between the two methods. This may result in a considerable amount of saving in both memory and processing time compared to the traditional supersampling method. To achieve this, when scan-converting a polygon, if the polygon covers the whole pixel, we sample the pixel only once. If the polygon partially covers the pixel, we perform a supersampling of that pixel. This requires a buffer that can handle the information generated from either of the two sampling resolutions. Here, we apply a technique similar to the one used in the A-buffer method. A standard 2D buffer is used for normal z-buffer scan-conversion (one sample per pixel). When a polygon edge is encountered, a large memory block is allocated to the pixel for storing the high-resolution samples. By using this method, there is at most one memory block allocated to a pixel no matter

how many polygon edges found in the pixel. So, there is no need to traverse through a possible long list of fragments as in the A-buffer method.

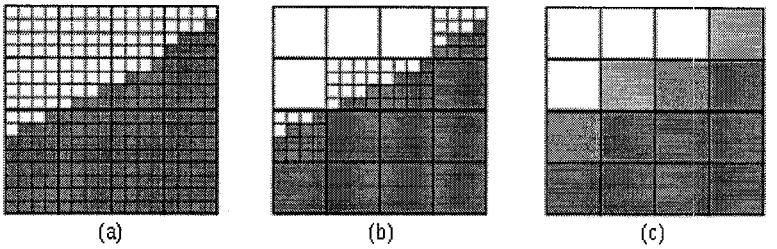

Figure 2: (a) Traditional Supersampling Method. (b) Adaptive Supersampling Method. (c) Resultant image after filtered down the supersampled pixels from (a) and (b).

3 Parallel Rendering Algorithms

In parallel rendering, we concern the common issues in parallel computing, such as the method for data decomposition, task granularity, scalability, and load balancing. Data decomposition in parallel rendering attempts to split data into multiple streams and computing the individual rendering units simultaneously [5,7,8].

3.1 Screen Subdivision Scheme

Screen subdivision scheme is one of the data-decomposition methods by which data are divided according their geometric description of the scene. In most cases, processors are assigned to handle a group of subdivided screen regions, geometry primitives are transferred to the corresponding processor to do rasterization. However, apart from the issue of data-decomposition, the strategies of partitioning the image-space would affect the load balance between processes. We can classify the strategies for dealing load imbalance as either static or dynamic.

3.1.1 Static Load Balance Scheme

The static partitioning approach subdivides the image space regardless of the location of the polygon data set in the image. Experiments show large area of partition regions usually result in poor load balancing while the smaller the partition regions, the better the load balancing between processes [3,5,7]. However, fine-grained computations generally incur more overheads for task

scheduling, communication, redundant calculations and more data replication among processors. Therefore using the coarsest granularity for tasks, which allows reasonable load balance, may be the best way for achieving good parallel timings. It is shown that the granularity ratio fixed at eight or twelve would give an optimal performance, but different scenes may give different results. The image size we use in testing is 512x512 pixels with 24 bits per pixel. We have constructed several data files to test our algorithm and load distribution among processors. For the data file that contains 4,000 polygons randomly distributed on the screen. The size of each polygon is 50 pixels. Figure 7a shows the output of the data file and figure 3 presents the completion time and the average idle time of processors when we divide the screen into 64 tiles with 64x64 pixels in each region and assign them to 8 processors. We set the granularity ratio to 8 and found that the load distribution was not as good as using a higher granularity ratio. The average completion time using the scheme was 0.53s and the average idle time was 0.19s. Hence, the percentage of average idle time to the completion time is about 35%.

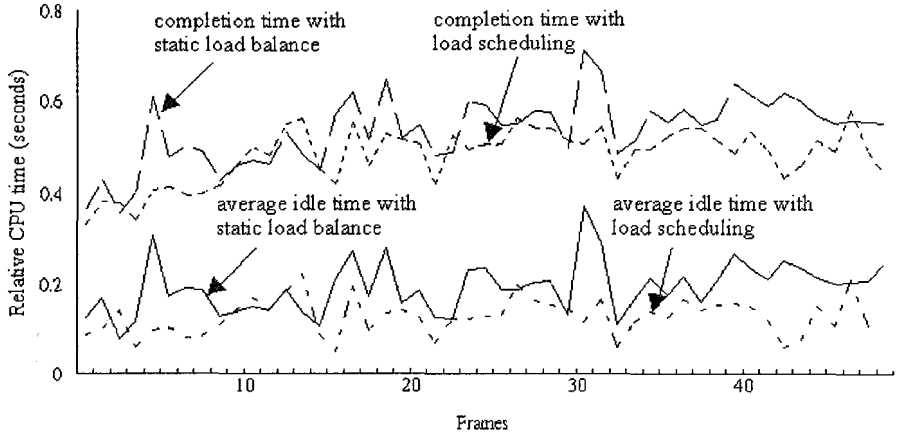

Figure 3: The completion time and average idle time during 50 frames of animation. The output of test case is shown in figure 7(a).

3.1.2 Dynamic Load Balance Scheme

We have adopted a simple load scheduling method to forecast the loading among processors. We perform scheduling between the animation frames, but the screen regions are not reassigned to the processors. Instead, we adjust

the sizes of the regions based on the loading of the processors in the previous frame. Our scheme depends on the frame-to-frame coherence property in the distribution of polygons over the screen. Figure 3 shows the improvement of using the load-scheduling algorithm with the same granularity ratio as above. The average completion time is about 0.47s and the average idle time is about 0.12s. Hence, the percentage of average idle time to completion time is about 25%. Experiments show that our method of load scheduling will give good result when the granularity ratio is low. Figure 4 shows the effect on screen partitioning when load scheduling is activated. The size of a region with higher loading in the previous frame will be decreased. The area of a region is calculated according to the ratio of load between processors.

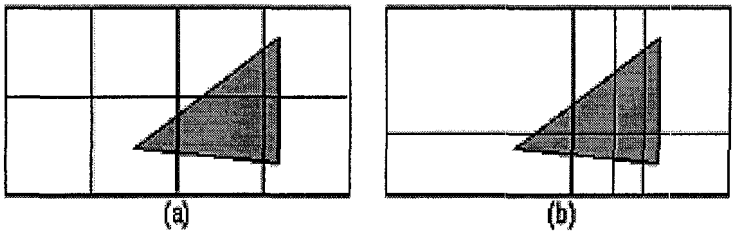

Figure 4: Screen partition: (a) without load scheduling, and (b)with load scheduling.

3.1.3 Memory Usage

Memory usage in generating an image is an important issue, especially in parallel computing. As the bandwidth of data communication affects the performance of the parallel algorithm greatly, and in order to produce anti-aliased images, we have to use more memory resulting in a higher demand for data transmission bandwidth in the image generation process, many parallel algorithms for rendering anti-aliased images suffer from lower performance. According to the results from our method, the total number of supersampled pixels is less than 20% of the total pixels in the image. One of the test cases containing 4,000 polygons consumes about 7.5MB of memory. Comparing with the 29.3MB memory usage in traditional supersampling methods, tremendous memory saving has been achieved in our method. Figure 5 shows the range of memory usage among 8 processors during the process. The range between minimum and maximum usage is narrow.The average memory usage by each processor is about 0.95MB. Figures 7c and d show the outputs of other test cases. The supersampled pixels used in these cases are below 10%, which consumed less than 4.6MB of memory. Figure 7b shows a test case that has a very

high complexity. The test file contains more than 32,000 polygons randomly distributed on the image. The supersampled pixels created in this case are still less than 50%, consuming 18MB memory. Note that this is an extreme example and is not likely to happen in most applications because in most applications, polygons are clustered to form objects, instead of randomly distributed around the screen space resulting in a lot of edge pixels being created. Figure 6 depicts the memory usage all our test cases.

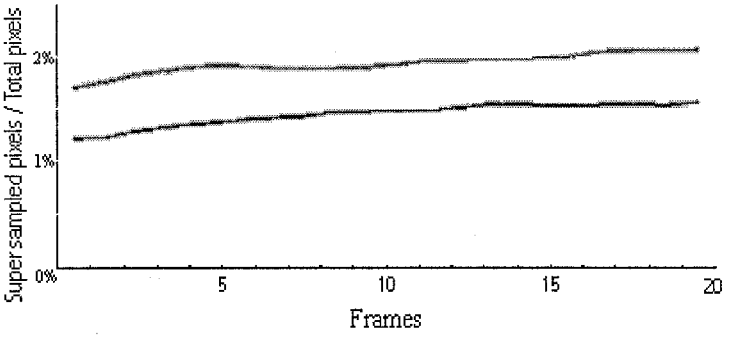

Figure 5: The maximum and minimum memory usage among 8 processors during 50 frames of animation.

4 Results and Discussion

From the results of our simulation, the memory usage of the supersampling method is very high and would greatly affect the parallel process. The adaptive supersampling method can maintain the memory usage in a lower level. In the tests shown in the previous section, the usage of supersampled pixels for most of the images do not exceed 20%, which require less than 10MB of memory to generate an image. This is much less than the amount of memory used in the supersampling method. The load-scheduling method also shows a significant improvement in load balance. The percentage of average idle time over the completion time is 25% and the original one is 35%. Although the method cannot produce a precise load adjustment between processors, it can minimize the overheads in the adaptive approach [8] in maintaining and retrieving non-local status, partitioning tasks, and data migration.

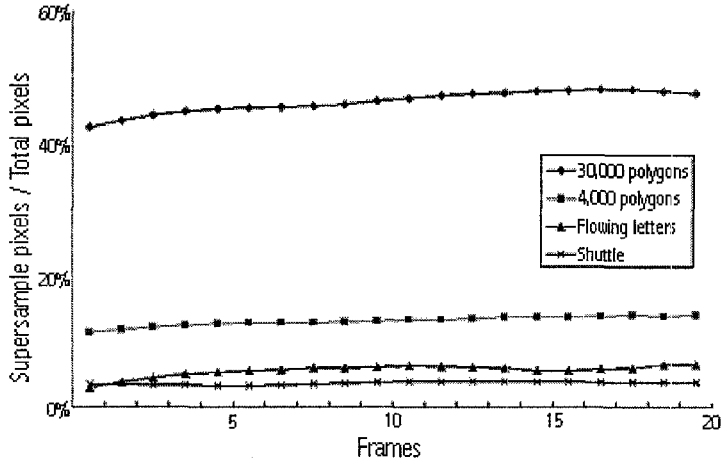

Figure 6: The usage of supersample pixels for all test cases shown in figure 7. In the case of rendering 30,000 small polygons, less than 50% of total number of pixels in the image need do supersampling. The other three cases have lower usage of supersampling pixels.

5 Conclusion

In this paper, we have described a scan-conversion method called the adaptive supersampling method. This method produces anti-aliased images but requires less memory than the supersampling z-buffer method. We have also presented the load distribution and scheduling method. The load scheduling provides a simple way to improve the load balance between processors so that the memory usage is kept roughly the same amount different processors.

References

1. L.Carpenter, "The A-buffer, an Antialiased Hidden Surface Method," *ACM Computer Graphics*, **18**(3), pp.103-108, July 1984.
2. E.Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces," Ph.D. Dissertation, Computer Science Department, University of Utah, 1974.
3. T.Crockett, "Parallel Rendering," Technical report 95-31, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, 1995.
4. F.Crow, "The Aliasing Problem in Computer-Generated Shaded Images," *Communications of the ACM*, **20**(11), pp.799-805, Nov.1977.

5. D.Ellsworth, "A New Algorithm for Interactive Graphics on Multicomputers," *IEEE Computer Graphics and Applications*, **14**(4), pp.33-40, July 1994.
6. R.W.H.Lau, "An Adaptive Supersampling Method," *Image Analysis Applications and Computer Graphics*, LNCS 1024, Springer-Verlag, pp.205-214, Dec.1995.
7. S.Whitman, *Multiprocessor Methods for Computer Graphics Rendering*, AK Peters, 1992.
8. S.Whitman, "A Task-Adaptive Parallel Graphics Renderer," *IEEE Computer Graphics and Applications*, **14**(4), pp.41-48, July 1994.

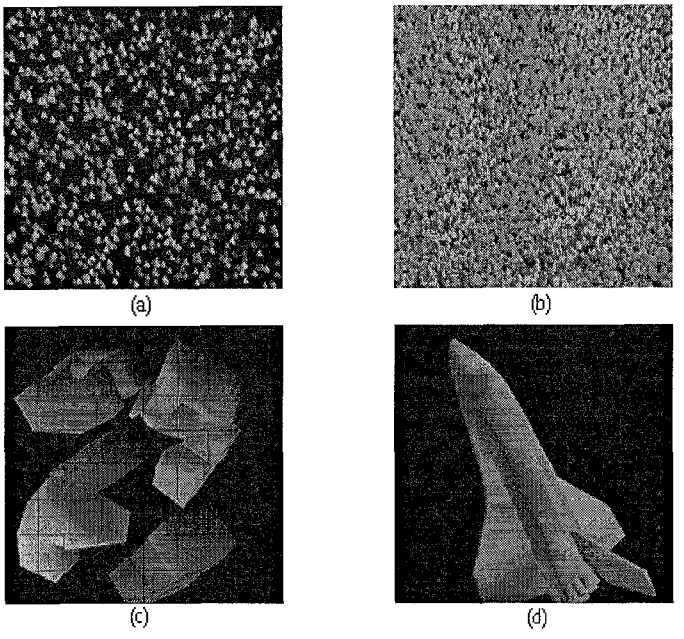

Figure 7: Sample test files:(a) contains more than 4,000 polygons, (b) contains more than 30,000 polygons. Each polygons has about 50 pixels in size distributed randomly on the screen.(c) contains very large polygons; screen is divided dynamically according to the load distribution of the image. (d) The output images of a 'shuttle'.