Journal of Circuits, Systems, and Computers, Vol. 11, No. 4 (2002) 377–391

REDUCE THE MEMORY BANDWIDTH OF 3D GRAPHICS HARDWARE WITH A NOVEL RASTERIZER*

CHENG-HSIEN CHEN† and CHEN-YI LEE‡

Department of Electronics Engineering, National Chiao Tung University,
1001 TaHsueh Road, Hsinchu, Taiwan, 300, Republic of China
†*chchen@royals.ee.nctu.edu.tw*
‡*cylee@royals.ee.nctu.edu.tw*

Received 15 April 2002
Accepted 27 June 2002

Currently, memory bandwidth has become the main bottleneck in graphics system. Reducing the memory access can reduce the power consumption and boost overall system performance. Low power technique is more important for graphics applications on handheld or mobile device. In this paper, we propose a novel visibility driven rasterizer to reduce the memory access and operations on invisible pixels. It integrates with two-level hierarchical Z-buffer to do visibility driven rasterization. The rasterization scheme is tile-order scan-line based, and the rasterizer can smartly change the tile-size depending on the triangle size. This technique can balance the rasterization loading under different triangles. Moreover, we propose a fast visibility test algorithm to quickly reject a group of pixels within the tile. Simulation results show that the overall bandwidth reduction can be up to 60% under our test images.

Keywords: Graphics processor; rasterizer; hierarchical Z-buffer.

1. Introduction

Three-dimensional computer graphics is widely used in various applications. Now, the graphics processor has become the essential component in desktop PC. It is capable of rendering million of triangles in one second. However, there are several bottlenecks of graphics processor to generate life-like images at real-time. The most serious one is memory bandwidth. Currently, the graphics processors[1,2] have wide memory bus (128-bit) and double data rate memory (up to DDR533) to boost the performance. The power consumption is huge. In the graphics system, a large portion of power is spent on memory bus transition. Off-chip memory bus transition consumes many times power than on-chip circuit transition. It challenges the designs of PCB board and high-speed memory interface. Besides, it makes it difficult to run graphics application on handheld or mobile devices under limited hardware

*Work supported by the National Science Council of Taiwan, R.O.C., under Grant *NSC* 90-2218-*E*-009-035.

resource. When the applications become more and more complex, it needs lots of effort and cost to increase the bandwidth for future system. Thus, it is urgent to reduce the power consumption of memory access in graphics system.

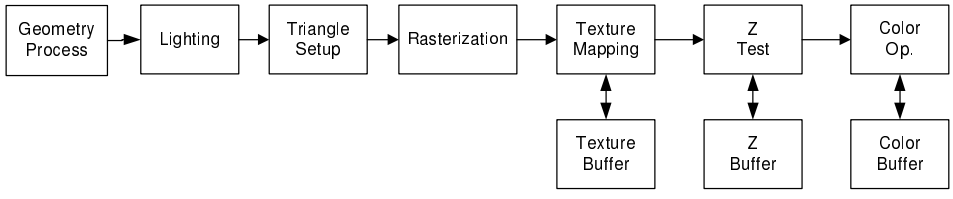

Fig. 1. Traditional triangle-based rendering flow.

Figure 1 shows the rendering flow of traditional triangle-based graphics hardware. The input triangles pass through the geometry transform, lighting and setup stages, and then scan-convert to pixels in rasterization stage. There are several memory accesses in the pipeline: texture buffer, Z buffer and color buffer accesses. Many gigabyte data will be transferred between the processor and memory to achieve the rendering speed of million triangles per second. Texture buffer and Z-buffer accesses dominate the bandwidth consumption. Several techniques have been proposed to overcome the bottleneck. Textures, which are two-dimensional bitmap images, are used to add realism to three-dimensional objects. For texture access, texture compression[3] and cache[4] techniques can reduce the texture buffer bandwidth effectively. Z-buffer, which was first proposed by Catmull,[5] is used to determine the visibility of every pixel. It is simple and easy to implement in hardware. But its efficiency is low. Every pixel will query Z-buffer to resolve the visibility, but most of them are hidden. Several visibility test and occlusion culling algorithms had proposed to accelerate this process.[6] However, most of them need lots of preprocessing and are not easy to be integrated into current hardware architecture. Thus we propose a two-level hierarchical Z-buffer[7] (HZ-buffer) to reduce the memory access. It is suitable for hardware implementation. This technique is application invisible, and can be integrated into current pipeline smoothly. Although two-level HZ-buffer performs well to reduce Z-buffer access, it still has space to improve the efficiency. In this paper, we propose a novel visibility driven rasterizer. It can improve the efficiency of HZ-buffer, and reduce the operations of scan-conversion as well as memory access. The novel rasterizer incorporates with two-level HZ-buffer to determine the visibility of a group of pixels at one time. Its scan-conversion scheme is tile-order scan-line based. Besides, it can smartly choose the tile size for different triangle size. This can balance the loading of rasterizer between large and small triangles. Simulation results show that the reduction of overall memory access can be up to $50 \sim 60\%$ under our test images.

2. Previous Work

In most of the 3D applications, the graphics processor spends lots of time processing invisible triangles and pixels. These operations decrease the fill-rate, and consume lots of bandwidth and power. We can increase the system performance and reduce power by discarding invisible part of the geometry as early as possible. Several algorithms have been proposed to accelerate visibility determination.[6] For example: object space pre-processing with binary space partition (BSP),[8] object space octrees combining with image space Z pyramid,[9] portals culling[10] and image space hierarchical occlusion map.[11] The goal of these visibility and occlusion culling algorithms is to reject invisible objects at early stage. However, the above algorithms need lots of pre-processing and cannot be integrated into current hardware architecture without modifying the application. Thus for real-time interactive applications, a hardware support efficient visibility test algorithm is very important.

Hierarchical Z-buffer, derived from the Z pyramid of N. Greene,[9] is suitable for hardware implementation. The simplified 8×8 HZ-buffer is used in current commercial product.[1] We also proposed a two-level HZ-buffer to test the visibility at both triangle-level and pixel-level.[7] It can effectively reduce the bandwidth of Z-buffer and eliminate unnecessary operations of invisible pixels. We will briefly introduce this in Sec. 2.

Beside two-level HZ-buffer visibility test, the rasterization stage will also influence the memory system performance. Rasterization means scan-converting the primitives (triangle) into fragments (pixels). There are two popular scan-conversion techniques as shown in Figs. 2(a) and 2(b): scan-line[12−14] based and stamp-based.[15,16] Scan-line based means transverse the triangle scan-line by scan-line. It starts from the vertex and walks along the edge and then horizontal line. Digital differential analyzer (DDA) is used to interpolate correspond color and depth of each pixel. Scan-line based is simple to implement, but it will decrease the memory performance of texture access because the lines may cross multiple pages of cache and cause cache miss. Stamp-based render pixels by $n \times m$ block size has good cache temporal locality.[4] It moves $n \times m$ pixels across the triangle and evaluates three edge equations[14] for each pixel of the stamp to determine whether the pixel is inside the triangle. Although stamp-based scan-conversion results in good memory

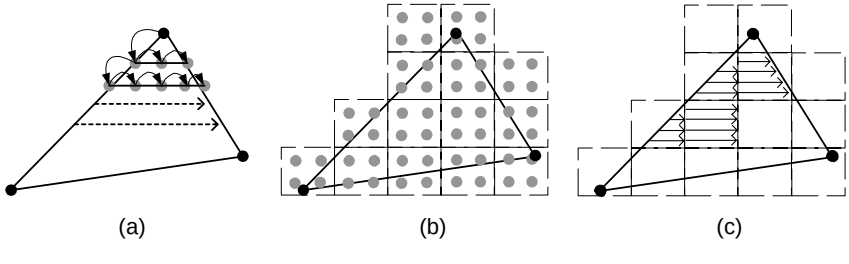

Fig. 2. (a) Scan-line based, (b) stamp-based, and (c) tile-order scan-line based rasterization.

performance, it requires efficient polygon traversal algorithm[17,18] and the hardware cost is higher than scan-line based. Besides, its efficiency is low for small triangles. Thus we combine both advantages and choose tile-order scan-line based algorithm (Fig. 2(c)). It has good cache temporal locality and the location of pixel is explicit.

The visibility test can combine rasterization to form visibility driven rasterization.[19] Visibility driven rasterization can save the operations of hidden pixels during rasterization. Meißner[19] also proposed a visibility driven rasterization scheme. It maintains a visibility mask in rasterizer and updates it for several frames. It requires the generation of a scene hierarchy and the bounding box for each entity before rendering. In this paper we propose another visibility driven rasterizer. It smartly chooses the tile-size for different triangle and incorporates with two-level HZ-buffer to accelerate the visibility test process. The HZ-buffer efficiency can be increased, too.

3. Proposed Visibility Driven Rasterizer

3.1. *Architecture*

Figure 3 shows the modified pipeline of our visibility driven rasterizer and two-level hierarchical Z-buffer. The triangle visibility test is placed before lighting stage and a pixel visibility test is done after rasterizer. The HZ-Buffer management unit maintains the correct depth information of HZ-Buffer. Moreover a bit-mask cache is proposed to store the temporal pixel coverage information and feedback to management unit to update HZ-buffer. The rasterizer performs visibility driven scan-conversion by fetching the visibility information from HZ-buffer.

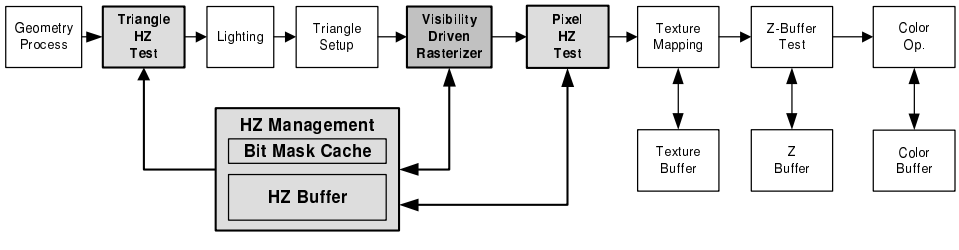

Fig. 3. The block diagram of two-level hierarchical Z-buffer and visibility driven rasterizer.

3.2. *Two-level hierarchical Z-buffer*

3.2.1. *Triangle and pixel visibility test*

Hierarchical Z-buffer is a reduced resolution of original Z-buffer. Figure 4 shows the concept of HZ-buffer. The pixel in higher level hierarchy represents the farthest value in covered lower level block. In previous literatures,[7] we shows different configurations of two-level hierarchical Z-buffer. It performs well to reduce Z-buffer access and efficiently discard hidden triangles and pixels.

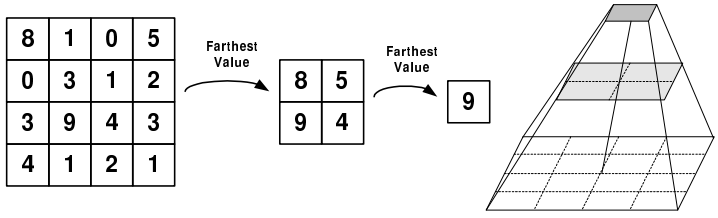

Fig. 4. The concept of hierarchical Z-buffer.

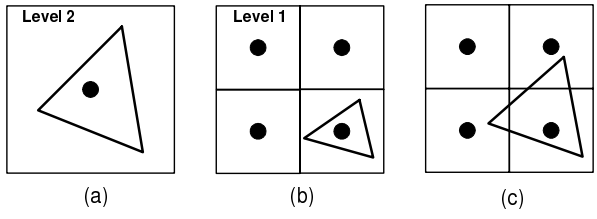

Fig. 5. Triangle hierarchical Z-buffer visibility test scheme.

There are two visibility test stages in the pipeline. The first one is done at triangle-level. For those triangles, which fall into high-level or low-level block (Figs. 5(a) and 5(b)), we test the visibility before it enters the lighting stage. The test is done by comparing the farthest vertex with the depth of corresponding block in hierarchical Z-buffer. For those triangles that cross multiple blocks (Fig. 5(c)), we leave it to visibility driven rasterizer to determine the visibility. Another visibility test is done pixel by pixel after rasterizer. By combining triangle and pixel hierarchical Z-buffer visibility test, we can quickly reject hidden primitives at early stage and save memory bandwidth as well as computing power.

3.2.2. *Hierarchical Z-buffer management*

Although HZ-buffer can efficiently discard invisible pixels, the challenge of hardware implementation is the HZ-Buffer update issue. From the definition of hierarchical Z-buffer, the pixel in higher level represents the farthest value in lower level block. If the low-level block size is $n \times n$, we have to fetch and compare $n \times n$ pixels to find the farthest one. This operation will be done every time, when the Z-buffer updates. This will slow down the performance and increase the memory access. Thus we propose a bit-mask cache to store the temporal pixel coverage information for several blocks (Fig. 6). The temporal farthest value and coverage mask of each block is stored in cache. By evaluating the coverage mask, we can find whether the block is fully covered and then update HZ-buffer by this temporal depth value. Simulation results show that 16 blocks bit-mask cache size is enough for good HZ-buffer performance.[7]

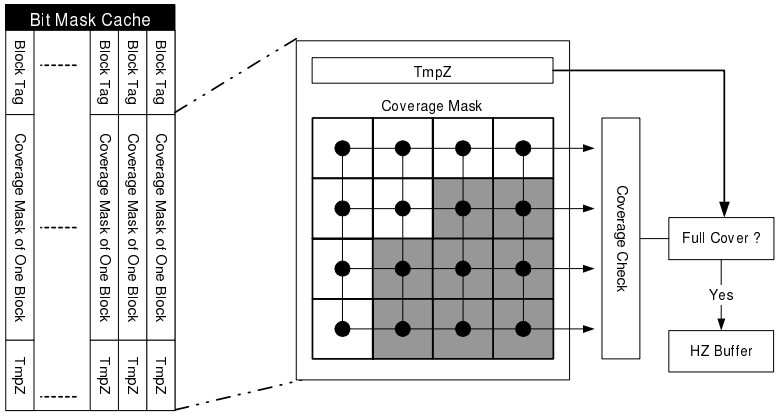

Fig. 6. The block diagram of bit-mask cache.

Table 1. The hierarchical Z-buffer size under 8-bit $16 \times 16 \sim 8 \times 8$ configuration.

	$16 \times 16 \sim 8 \times 8$	
Bi-level compression	No	Yes
1600×1200	31.88 KB	18.75 KB
1280×1024	21.76 KB	12.8 KB
1024×768	13.06 KB	7.68 KB
800×600	7.97 KB	4.69 KB

3.2.3. *Dynamic bi-level compression*

The hardware cost of HZ-buffer depends on the configuration, depth numerical accuracy and screen resolution. To reduce the hardware cost, we propose a dynamic bi-level compression technique[7] to reduce the buffer size. The buffer size can decrease 40%. The concept is to explore the image space coherence of HZ-buffer and carefully assign the depth value for high and low level blocks. The performance degradation is very small and the decompression flow is very simple. Table 1 shows the buffer size reduction under 8-bit accuracy and $16 \times 16 \sim 8 \times 8$ configurations. Simulation results will be shown in Sec. 4.

3.3. *Visibility driven rasterizer*

3.3.1. *Tile-order scan-line based polygon traverse*

Our visibility driven rasterizer scan-conversion scheme is tile-order scan-line based. The triangle is rendered tile by tile and scan-line by scan-line in each tile (Fig. 2(c)). Tile-order has better memory performance, and scan-line based makes it simple to traverse triangle. Comparing with the whole scan-line order polygon traverse (Fig. 2(a)), a little overhead will be paid for tile-order traverse. The state of the tile

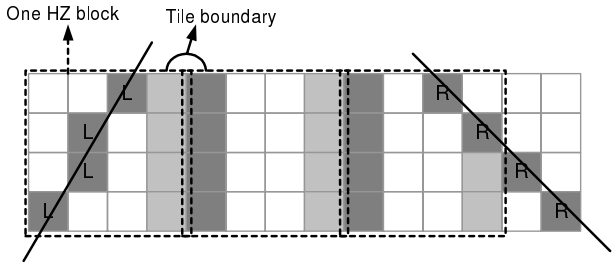

Fig. 7. 4×4 tile rendering.

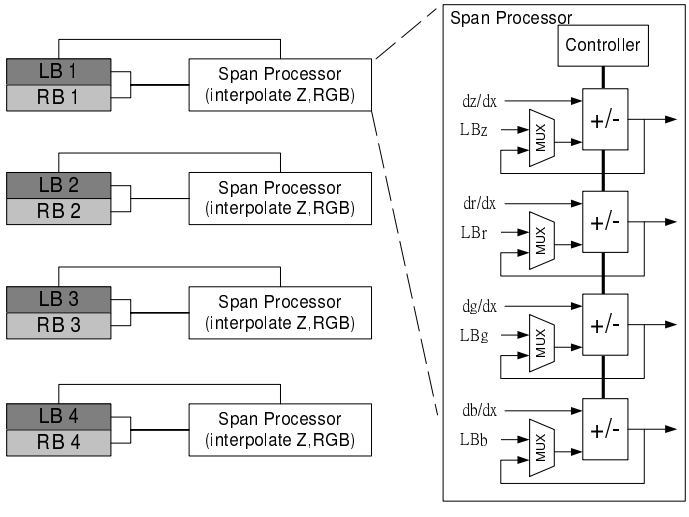

Fig. 8. Tile-order scan-line based triangle scan-conversion architecture.

boundary should be saved as the initial value for next tiles. Figure 7 shows a 4×4 tile-order polygon traverse scheme. The left and right boundaries of the edge are setup first, and then the rasterizer processes each tile at one time. The 4×4 tile rasterizer architecture is shown in Fig. 8. The parallel span processors interpolate the depth and the color of the pixel on each scan-line by digital differential analyzer (DDA). The DDA is an adder and a span processor has four DDA to interpolate both the depth and the RGB color. The new left and right boundary is stored in register LB and RB for next iteration. We can increase the number of boundary registers or adder unit to extend the architecture to process large tile. The number of span processors can also be extended to increase the throughput.

3.3.2. *Group of pixel visibility test*

In Sec. 3, we have introduced the two-level HZ-buffer visibility test. However, the triangles that cross multiple blocks are ignored at triangle-level test. Now, the

visibility driven rasterizer processes these triangles and quickly discard the invisible part of the triangles. It determines the visibility of a group of pixels. The group can be a tile or a scan-line in tile. When the test is fail, these pixels can be discarded immediately.

To determine the visibility of one tile or one scan-line, the maximum (or farthest) "Z" (or depth) must be found first, and then we use this value to compare with the depth in HZ-Buffer. If the test fails, it means that the whole tile or scan-line is invisible. For each tile in rasterizer, we can find the trend of depth variation from the slope dz/dx and dz/dy. The slopes are calculated in triangle setup stage, and they are the increment of depth in horizontal and vertical direction. The definitions are shown below.

Assume triangle vertices

$$(x1, y1, z1),\ (x2, y2, z2),\ (x3, y3, z3) \text{ and } y1 < y2 < y3$$

$$\text{area} = (x3 - x1) \times (y2 - y1) - (x2 - x1) \times (y3 - y1)$$

$$dzdx = \frac{(y2 - y1) \times (z3 - z1) - (y3 - y1) \times (z2 - z1)}{\text{area}}$$

$$dzdy = \frac{(x3 - x1) \times (z2 - z1) - (z3 - z1) \times (x2 - x1)}{\text{area}}$$

There are four different depth variations in one tile (Fig. 9). We can obtain the maximum (or farthest) "Z" according to the sign of dz/dx and dz/dy. The maximum will be one of the four corners in the tile. Then we use this MaxZ to do visibility test with HZ-buffer. The maximum in one scan-line can also be easily found according to the sign of dz/dx.

Although we can easily find the farthest pixel in one tile, this technique cannot apply on partial covered tile. For partial covered tile, it needs more effort to find the maximum "Z" in the covered area. Sometimes, the cost of searching process will be equal to rasterize this tile. Thus it needs fast algorithm to do visibility test with this tile. Figure 10 shows a partial covered tile. The depth decreases in X direction and increases in Y direction. Thus the farthest value may be L2, L3, or LH. The exact one will be obtained by comparing the absolutely value of dz/dx and dz/dy. Instead of exactly evaluating the maximum in covered area, we conservatively estimate the

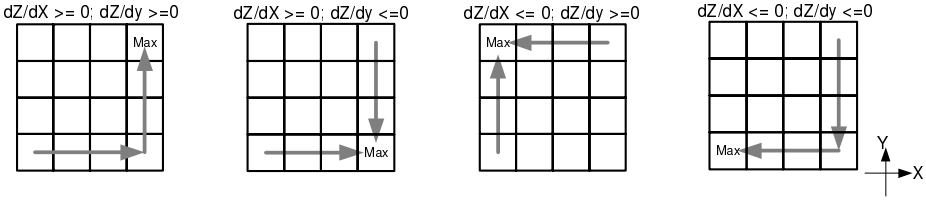

Fig. 9. Four different depth variation in one tile.

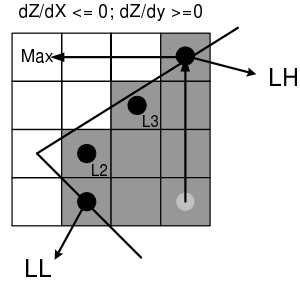

Fig. 10. A partial covered tile (LH: left-highest pixel, LL: left-lowest pixel).

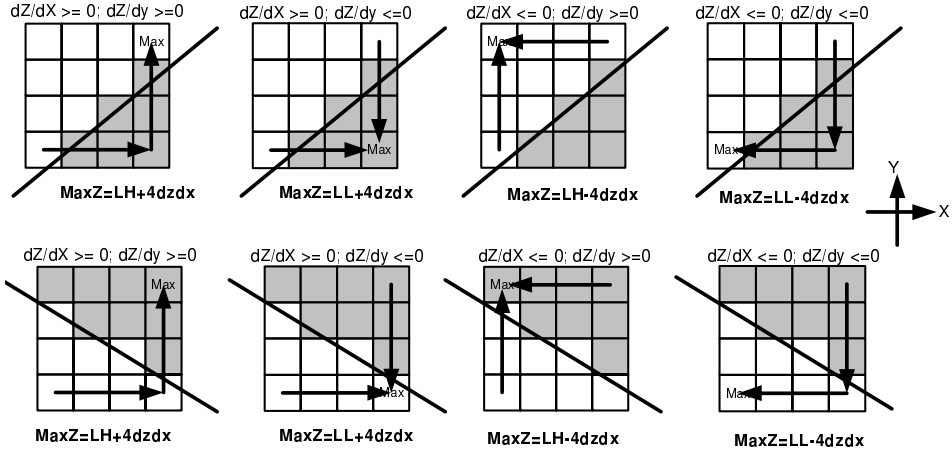

Fig. 11. Four conservative maximum "Z" estimations.

farthest value in this tile. Figure 11 shows the maximum "Z" estimation. The proof of conservative maximum depth estimation is shown as follows.

Conservative estimation:

$$\begin{aligned}
\text{maxL} &= \left(\frac{dz}{dy} \geq 0\right) ? \,(\text{LH} : \text{LL})\,; \\
\text{maxL1} &= \text{maxL} + \text{tile_size} \times \frac{dz}{dx} \\
\text{maxL2} &= \text{maxL} - \text{tile_size} \times \frac{dz}{dx} \\
\text{estimate_maxZ} &= \left(\frac{dz}{dy} \geq 0\right) ? \,(\text{maxL1} : \text{maxL2})\,;
\end{aligned}$$

Proof.

$$\text{if} \quad \frac{dz}{dy} \geq 0 \quad \text{and} \quad \frac{dz}{dx} \geq 0$$

The real maximum can be represented as:

$$\text{maxZ} = \text{LL} + a \times \frac{dz}{dx} + b \times \frac{dz}{dy}\,; (0 \le a < \text{tile\_size};\ \ 0 \le b < \text{tile\_size})$$

LH can be represented as follows:

$$\begin{aligned} \text{LH} &= \text{LL} + c \times \frac{dz}{dx} + d \times \frac{dz}{dy}\,; \\ \because \text{ LH is left top pixel} &\Rightarrow c \le a,\ b \le d \\ \therefore \text{ maxZ} &\le \text{LL} + a \times \frac{dz}{dx} + d \times \frac{dz}{dy}\,; \\ &\le \text{LL} + d \times \frac{dz}{dy} + (c + \text{tile\_size}) \times \frac{dz}{dx}\,; \\ &= \text{LH} + \text{tile\_size} \times \frac{dz}{dx} = \text{estimate\_maxZ}. \end{aligned}$$

□

The proofs of other cases are the same.

In Fig. 11, there are four cases under different dz/dx and dz/dy. Both LL and LH are available in the beginning. According to the sign of dz/dx and dz/dy, the estimated maxZ will be LL (or LH) add/sub tile_size $\times dz/dx$. Because tile size is power of two, the multiplication can be replaced by a shift. The conservative maximum depth estimation forces the estimated maxZ larger or equal to the real maximum in this tile. Thus the visibility test with this estimated maxZ would not produce error in final images.

3.3.3. *Adaptive changing tile size*

In addition to visibility test, loading balance is also a problem in rasterizer. Applications, which include lots of small triangles, are triangle-rate limited due to heavy geometry operation loading. However other applications, which include lots of large triangles, are usually fill-rate limited. The rasterizer will generate more pixels. Figure 12 shows the benchmark result of GPU[2] under different triangle size. Due to geometry overloading, the triangle rate would not increase when the triangle size are under 10 pixels. Beside, the pixel fill rate will also saturate, when the triangle size increase. Because different triangle sizes will cause different throughput of rasterizer, other pipeline stages will stall to wait the data.

To balance the loading of different triangle, we try to accelerate the visibility test of large triangles by increasing the tile-size. Our visibility driven rasterizer can change the tile-size depending on the triangle size. For large triangle, the tile size is as large as high-level HZ-buffer block. This can reduce the latency of visibility test especially for large hidden triangle. For small triangle, we choose small tile rasterization. The decision depends on the number of scan-line inside the triangle. If the number of scan-line is larger than two times of low-level block, we change to

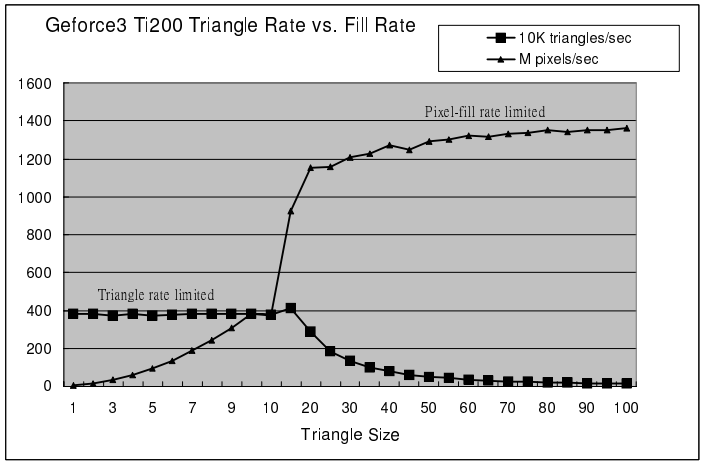

Fig. 12. The triangle rate and pixel fill rate benchmark of GPU.

large-tile rasterization. Otherwise, the small-tile is used. By dynamically changing the tile size, we can balance the loading of different triangles.

Combining all techniques, Fig. 13 shows the overall flow of our visibility driven rasterizer for each triangle. First, we will choose the tile-size according to the triangle size. Then we have to setup the initial status for tile-order scan-conversion. The left boundary and right boundary of the triangle have to be evaluated first. Following, the rasterizer processes tile by tile. It tests tile visibility and scan-line visibility during rasterization. The operation will finish until it reaches the end of triangle.

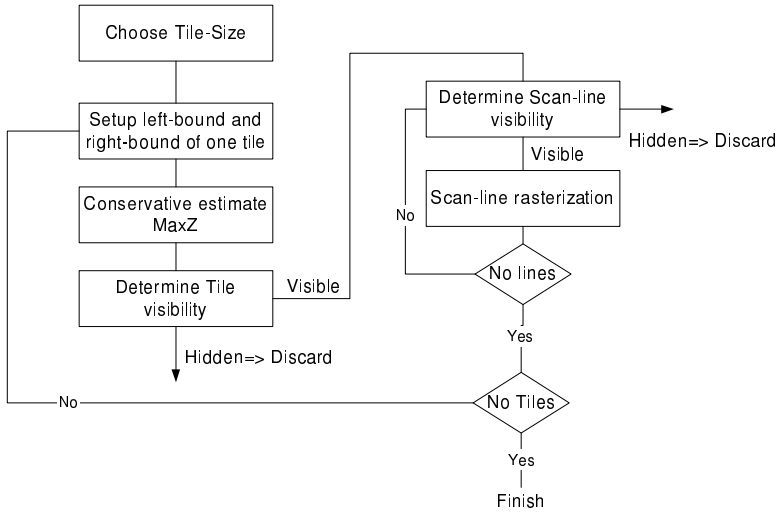

Fig. 13. The flow of visibility driven rasterization.

Table 2. Simulation result.

		House		Chemical atoms	
Total triangle		316672		622560	
Original memory access		768902		2851709	
HZ-buffer compression		No	Yes	No	Yes
8 × 8-	Triangle discard rate	22.4%	21.96%	14.69%	14.29%
4 × 4	Pixel discard rate	5.9%	4.17%	24.1%	15.87%
	Large tile test*	6496	6496	156248	156246
	Large tile hidden *	546	1055	42899	60278
	Small tile test **	4798	4767	100857	100871
	Small tile hidden **	1184	1039	56025	47567
	Scan-line hidden ***	5996	3247	167536	103574
	Overall bandwidth reduction	49.73%	48.83%	58.54%	55.18%
16 × 16-	Triangle discard rate	25.66%	22.04%	9.98%	8.52%
8 × 8	Pixel discard rate	3.0%	2.61%	5.94%	4.79%
	Large tile test *	2572	2572	9261	9261
	Large tile hidden *	124	161	19	54
	Small tile test **	6316	6305	225100	225316
	Small tile hidden **	185	612	57563	50826
	Scan-line hidden ***	3474	2656	43364	34688
	Overall bandwidth reduction	48.02%	47.03%	45.25%	43.47%
		Car		Coffee Shop	
Total triangle		454746		288374	
Original memory access		781167		841034	
HZ-buffer compression		No	Yes	No	Yes
8 × 8-	Triangle discard rate	36.58%	35.7%	21.08%	17.75%
4 × 4	Pixel discard rate	21.62%	14.76%	7.31%	4.21%
	Large tile test *	7557	7557	11855	11855
	Large tile hidden *	1491	1832	2390	3411
	Small tile test **	22957	22337	6025	5908
	Small tile hidden **	15229	12251	3803	3149
	Scan-line hidden ***	7428	4968	12699	5770
	Overall bandwidth reduction	65.71%	62.76%	38.2%	36.24%
16 × 16-	Triangle discard rate	35.26%	30.82%	18.39%	13.77%
8 × 8	Pixel discard rate	12.84%	9.4%	5.55%	3.33%
	Large tile test *	2721	2721	5781	5781
	Large tile hidden *	170	275	647	1077
	Small tile test **	20462	20628	7064	7015
	Small tile hidden **	6227	5401	3037	2330
	Scan-line hidden ***	4415	3247	9059	4790
	Overall bandwidth reduction	60.72%	58.8%	37.82%	36.37%

* Large tile test: number of blocks that use large tile visibility test.
* Large tile hidden: number of blocks that fail large tile visibility test.
** Small tile test: number of blocks that use small tile visibility test.
** Small tile hidden: number of blocks that fail small tile visibility test.
*** Scan-line hidden: number of scan-lines that fail visibility test.

4. Simulation and Analysis

Table 2 shows the simulation results of our approach under 1600×1200 resolution, 64-entry bit-mask cache, 8-bit depth representation and two different HZ-buffer configurations with/without HZ-buffer compression. The test images includes hundred of thousands triangles. Table 2 also shows the number of large tile, small tile and line visibility test. The triangle discard rate represents the percentage of total triangles that fail the visibility test before lighting stage. The pixel discard rate represents the percentage of total pixel that fail HZ-Buffer visibility test after rasterization stage. The dynamic change of tile-size performs well to balance the loading. For example: "Coffee Shop", which includes many large triangles, has more large-tile rasterization. In contrast with "Coffee Shop", "Chemical Atoms" has more small-size triangles and more small-tile rasterization.

The performance of triangle discard rate depends on the property of application. If the application contains lots of small triangles (e.g., Chemical Atoms), the triangle discard rate will be decreased by increasing the block-size of HZ-buffer. Because large block-size results in coarse depth resolution and we cannot resolve the visibility of the triangle by HZ-buffer. The pixel discard rate is much smaller than previous approach,[7] because the rasterizer discards most of the invisible pixels. This shows the improvement of visibility driven rasterizer. Overall, we can see that the bandwidth reductions of the test images are between 30% to 60% under our approach. Moreover, Table 3 shows the bandwidth reduction improvement when comparing with previous scan-line based two-level HZ-buffer approach.[7] We can see that by combining two-level HZ-buffer and visibility driven rasterizer, the bandwidth can further be reduced about $15\% \sim 30\%$.

Table 3. Original: scan-line based two-level HZ-buffer versus Modified: tile-order scan-line based visibility driven rasterizer.

	House		Chemical atoms		Cars		Coffee Shop	
TYPE	8×8-4×4	16×16-8×8	8×8-4×4	16×16-8×8	8×8-4×4	16×16-8×8	8×8-4×4	16×16-8×8
Original	21.5%	18.97%	42.55%	24.29%	42.65%	32.97%	18.4%	17.13%
Modified	49.73%	48.02%	58.54%	45.25%	65.71%	60.72%	38.2%	37.82%

5. Conclusion

Three-dimensional graphics applications are both computation and data intensive operations. It requires bandwidth and computation reduction techniques for future complex real-time applications. When bringing graphics applications into handheld or mobile devices, the power consumption becomes an important issue. Today, memory bandwidth bottleneck has become the main issue of graphics hardware design. Various techniques were proposed to save the bandwidth in different stages. In this

paper, we propose a novel visibility driven rasterizer with two-level HZ-buffer. Combining with two-level HZ-buffer, the rasterizer can quickly and efficiently discard invisible part of the triangle during rasterization. The power can also be reduced by saving the operations and memory access of hidden pixels in different stages: triangle HZ-buffer test stage, visibility driven rasterizer stage and pixel HZ-buffer test stage. Combining above techniques, the overall power consumption will decrease largely. Beside, two-level HZ-buffer visibility test is application invisible. The applications will get the benefit of HZ-buffer without modifying the rendering flow. Our approach is suitable for hardware implementation and can easily be integrated into current graphics pipeline. Simulation results show that the overall bandwidth reduction is quite large, leading to achieve a low-power solution.

Acknowledgment

The authors are grateful to the support from the National Science Council of Taiwan, R.O.C., under grant *NSC* 90-2218-*E*-009-035.

References

1. S. Morein, "ATI Radeon HyperZ Technology", *Eurographics Hardware Workshop 2000, Hot3D Panel*, 2000.
2. nVidia, "Technical brief: Geforce3: Lightspeed memory architecture", *http://www.nvidia.com*, 2001.
3. C.-H. Chen and C.-Y. Lee, "A JPEG-like texture compression with adaptive quantization for 3D graphics application", *The Visual Computer* **18**, 1 (2002) 29–40.
4. Z. S. Hakura and A. Gupta, "The design and analysis of a cache architecture for texture mapping", *Proceedings of the 24th International Symposium on Computer Architecture*, 1997, pp. 108–120.
5. C. Edwin, "Computer display of curved surfaces", *Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition and Data Structures*, 1975, pp. 11–17.
6. T. Moller and E. Haines, *Real-Time Rendering*. A. K. Peters, Natick MA USA, 1999.
7. C.-H. Chen and C.-Y. Lee, "Two-level hierarchical Z-buffer for 3D graphics hardware", *Proceedings of IEEE International Symposium on Circuits and Systems*, Vol. 2, 2002, pp. 253–256.
8. D. Gordon and S. Chen, "Front-to-back display of BSP trees", *IEEE Comput. Graphics and Appl.* **11**, 5 (1991) 79–85.
9. N. Greene, M. Kass, and G. Miller, "Hierarchical Z-buffer visibility", *Proceedings of SIGGRAPH*, July 1993, pp. 231–238.
10. D. Luebke and C. Georges, "Portals and mirrors: Simple, fast evaluation of potentially visible sets", *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, 1995, pp. 105–106.
11. H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff, "Visibility culling using hierarchical occlusion maps", *Proceedings of the 24th Conference on Computer Graphics and Interactive Techniques*, 1997, pp. 77–88.
12. R. W. Swanson and L. J. Thayer, "A fast shaded-polygon renderer", *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, Vol. 20, No. 4, 1986, pp. 95–102.
13. S. W. M. Kelly and K. Gould, "A scalable hardware render accelerator using a modified scanline algorithm", *Proceedings of SIGGRAPH*, 1992, pp. 241–248.

14. K. Akeley and T. Jermoluk, "High-performance polygon rendering", *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, Vol. 22, No. 4, 1988, pp. 239–246.
15. J. Pineda, "A parallel algorithm for polygon rasterization", *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, Vol. 22, No. 4, 1988, pp. 17–20.
16. J. McCormack, R. McNamara, C. Gianos, L. Seiler, N. P. Jouppi, K. Correll, T. Dutton, and J. Zurawski, "Neon: A fast single-chip 3D workstation graphics accelerator", *Technical Report 98.1, Compaq Western Research Laboratories*, 1998.
17. M. D. Waller, J. P. Ewins, M. White, and P. F. Lister, "Efficient primitive traversal using adaptive linear edge function algorithms", *Comput. Graphics* **23** (1993) 265–275.
18. J. McCormack and R. McNamara, "Tiled polygon traversal using half-plane edge function", *Proceedings of SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware 2000*, 2000, pp. 15–21.
19. M. Meißner, D. Bartz, R. Gunther, and W. Straßer, "Visibility driven rasterization", *Comput. Graphics Forum* **20**, 4 (2001) 283–294.