VC-1: A Scalable Graphics Computer with Virtual Local Frame Buffers

Satoshi Nishimura Tosiyasu L. Kunii

The University of Aizu[†]

Abstract

The VC-1 is a parallel graphics machine for polygon rendering based on image composition. This paper describes the architecture of the VC-1 along with a parallel polygon rendering algorithm for it. The structure of the VC-1 is a loosely-coupled array of 16 general-purpose processors, each of which is equipped with a local frame buffer. The contents of the local frame buffers are merged in real time for generating the final image. The local frame buffers are virtualized with a demand-paging technique, by which the image memory capacity for each local frame buffer is reduced to one eighth of full-screen capacity. Polygons are rendered in either pixel parallel or polygon parallel depending on the on-screen area of each polygon. The real performance of the VC-1 as well as estimated performance for systems with up to 256 processors is shown.

CR Categories and Subject Descriptors: C.1.2 [Processor Architectures]: Multiprocessors - MIMD, Parallel processors; B.3.2 [Memory Structures]: Design Styles - Virtual memory; I.3.1 [Computer Graphics]: Hardware Architecture - Parallel processing, Raster display devices; I.3.3 [Computer Graphics]: Picture/Image Generation - Display algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Visible line/surface algorithms.

Additional Key Words and Phrases: demand paging, frame buffers, parallel polygon rendering, scalable.

1 Introduction

The increasing demand for real-time generation of high-quality images puts more and more emphasis on the importance of scalability in high-end graphics machines. In mechanical CAD or virtual reality applications, solid models composed of more than one million polygons are often used for generating high-quality images. To display such models at the rate of 30 frames per second, more than 30 million polygons need to be processed in one second, which requires hundreds of processors with current technology. Thus, scalability is the most important issue in order to satisfy such a demand.

The image composition architecture proposed by Molnar [8] is the most promising candidate for future polygon-rendering machines because of its linear scalability. According to Molnar et al. [9], graphics machine architectures can be categorized into three types. The first category, called *sort-first*, performs both geometry calculation and rasterization in pixel parallel, while the second category, named *sort-middle*, executes geometry calculation in object parallel and rasterization in pixel parallel. The sort-middle category, to which most of today's commercial machines including Silicon Graphics RealityEngine [1] belong, has limits on the number of processors since its communication network becomes bottlenecked on a large-scale system when the results of geometry calculation are sent to every rasterization processor. The sort-first category has a similar problem. The last category, called *sort-last*, performs both geometry calculation and rasterization in object parallel, and is possibly scalable since the required bandwidth of its communication network is almost constant against the number of polygons. The sort-last category can be further divided into two classes based on which set of data is transmitted via the communication network. In the first class, each rasterization processor sends only the pixels generated [7]. This class is hard to scale because implementing a scalable communication network for this class is difficult. The second class of the sort-last category is the image composition architecture, in which case each processor outputs all the pixels on the screen. With image composition architecture, a linearly-structured communication network can be used, and therefore, there is no difficulty with increasing the number of processors.

The history of image composition architecture descends from Cohen and Demetrescu's proposal [2]. Various modifications or additions are later applied by many researchers; for example, the integration of a geometry calculation unit in each processor [13], anti-aliasing by generating a depth-sorting list of pixel values [14, 16], anti-aliasing by alpha-blending [15], and the implementation of Phong shading on this architecture [4]. Molnar's architecture differs from these architectures in that each processor can handle plural polygons while the architectures listed above can handle just one polygon per processor.

Figure 1 depicts Molnar's image composition architecture. Each processor has its own full-screen frame buffer, called a local frame buffer, which holds a subimage (including Z-values) generated by the processor. This subimage possibly overlaps with the subimages generated by other processors. The contents of all the local frame buffers are merged periodically by the image merger at the speed of the CRT scan. During merging, depth values are taken into account in order to accomplish hidden surface removal. Palovuori [12] has implemented a system based on this architecture.

One of the problems with this architecture lies in the memory cost for local frame buffers. Since frame buffer bandwidth is one of the critical factors of graphics machines, frame buffer memory should be fast as possible. However, enabling fast memory of full-screen capacity for each processor is impractical if the number of processors is large. Therefore, the use of low-speed memories is imposed, which will degrade the system's performance.

One possible method for overcoming this problem is the region-based approach utilized in the PixelFlow system [10]. In this approach, the screen is divided into several regions, and each local frame buffer holds the pixel information only for one region. To hold pixel information for the entire screen, a global frame buffer is placed between the image merger and the CRT. Prior to rasterization, each processor performs geometry calculations and classifies polygons according to regions using xy-buckets. Then, the polygons are rasterized in multiple passes: for each pass, images for one region are generated on the local frame buffers, and then, the images are merged and stored into the appropriate section on the

[†] Tsuruga, Ikki-machi, Aizu-Wakamatsu City, Fukushima, 965-80, Japan.
Email: nisim@u-aizu.ac.jp, kunii@u-aizu.ac.jp

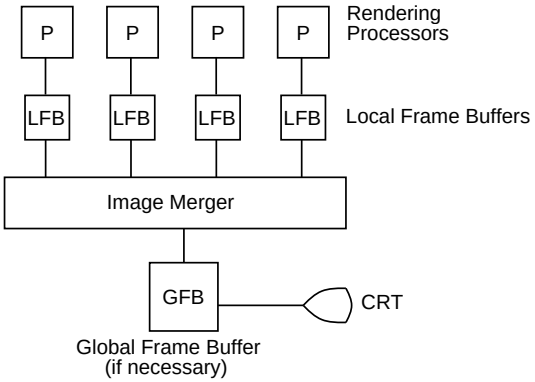


Figure 1: Image composition architecture

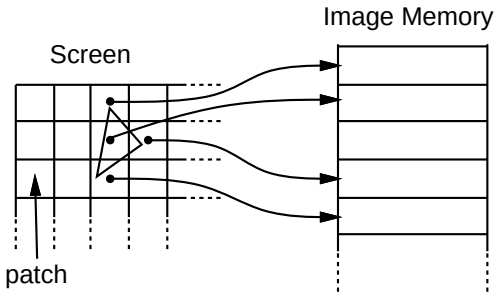


Figure 2: Virtual local frame buffer

global frame buffer. With this method, the memory requirement for the local frame buffer is reduced in inverse proportion to the number of regions. However, it suffers from the following disadvantages:

1. The load balancing problem becomes more difficult.
   In the full-screen approach, it is enough to execute global synchronization once per frame. In the region-based approach, global synchronization is necessary for each pass. Therefore, processor idle time increases since a processor can not start the processing in the next region until all the other processors finish processing the current region.

2. The clipping time increases.
   Polygon clipping occurs not only with the screen boundary but also with region boundaries.

In order to solve these problems, we propose an alternative approach called a *virtual local frame buffer* for reducing the frame buffer memory requirement. Section 2 describes the principle of the virtual local frame buffer. Section 3 presents a prototype machine called the VC-1 developed for justifying the virtual local frame buffer. Section 4 discusses a parallel polygon rendering algorithm suitable for the VC-1 architecture. Section 5 shows the results of performance measurement and compares the virtual frame buffer approach with the region-based approach. Section 6 discusses the scalability of the VC-1 architecture. Section 7 concludes this paper with some suggestions for future research.

2 Virtual Local Frame Buffer

Access to the local frame buffer (LFB) tends to have space locality. In other words, a processor seldom generates an image filling the entire screen. In most cases, only a part of the screen is accessed.

The virtual local frame buffer (VLFB) utilizes this characteristic to reduce the memory requirement. We use a demand-paging technique so that the LFB can virtually hold the pixel information of the entire screen. The screen is divided into small equal-sized rectangular regions called *patches*, and memory units are allocated only to such patches that the processor has accessed (Figure 2). To implement this, two types of memories are used: *image memory* and a *patch table*. Image memory holds pixel information and its capacity is less than full-screen capacity. The patch table maintains the access existence and image memory address of every patch.

In most cases, generated images can be completely stored in image memory. However, when access locality is weak, it is possible that image memory is exhausted. This situation is called a *local frame buffer overflow* (LFB overflow). To handle this, the global frame buffer (GFB) in Figure 1, which is a full-screen frame buffer holding both color and depth information, is necessary. The GFB is cumulative: it stores pixel values only when a new pixel value is closer to the viewer than the one previously stored in the GFB. When an LFB overflow occurs in a processor, the processor suspends rendering until a certain number of patches in the LFB are transferred to the GFB, and then it continues rendering. Since the GFB is cumulative, the patches can be reused to store a different part of the screen once its contents are transferred to the GFB.

There are two concerns with the VLFB. One is that LFB access time may increase because two memories, the patch table and image memory, are accessed during each access operation. However, by organizing a two-stage pipeline consisting of patch table access and image memory access, the access time increase will be mitigated. The other concern is processor idle time due to LFB overflows. In our implementation, true real-time image generation no longer becomes possible if an LFB overflow occurs in some processor. However, LFB overflows are nearly avoidable with the careful consideration of the image memory capacity as well as a software technique called adaptive parallel rasterization described in Section 4.2.

3 The VC-1

The VC-1 (Figures 3, 10 and 11) is a loosely-coupled multiprocessor with a frame buffer subsystem containing the VLFBs. The primary purposes of the VC-1 are to evaluate our architecture and also to provide an environment for developing parallel software for the architecture. Rendering speed is not a primary design goal; instead, flexibility is maximized. For this reason, the VC-1 has no special hardware accelerators for rasterization.

3.1 Processing Elements

The VC-1 has 16 processing elements, each of which contains the Intel i860 processor operating at 40MHz together with an 8MB local memory. The peak floating-point operation performance of the VC-1 is 1.3 GFLOPS. This high performance makes it possible to apply the VC-1 not only to polygon rendering but also to ray tracing or other numerical applications.

The processing elements are connected by both point-to-point communication links and a broadcast bus. The point-to-point communication links are organized in a modified 2D-torus topology and able to exchange data between any pair of processors or between a processor and the host computer at the speed of 20Mbits/sec. The broadcast bus can transmit data from either the host computer or one of the processors to all the processors. The peak bandwidth of the broadcast bus is 27Mbytes/sec. With polygon rendering, only the broadcast bus is utilized; communication links are reserved for other algorithms.

The sync line is a 1-bit line realizing global synchronization across all the processors. The host computer can optionally participate in the global synchronization.

3.2 The Host Computer

The host computer is an IBM-compatible personal computer equipped with communication interfaces for the VC-1. All the data

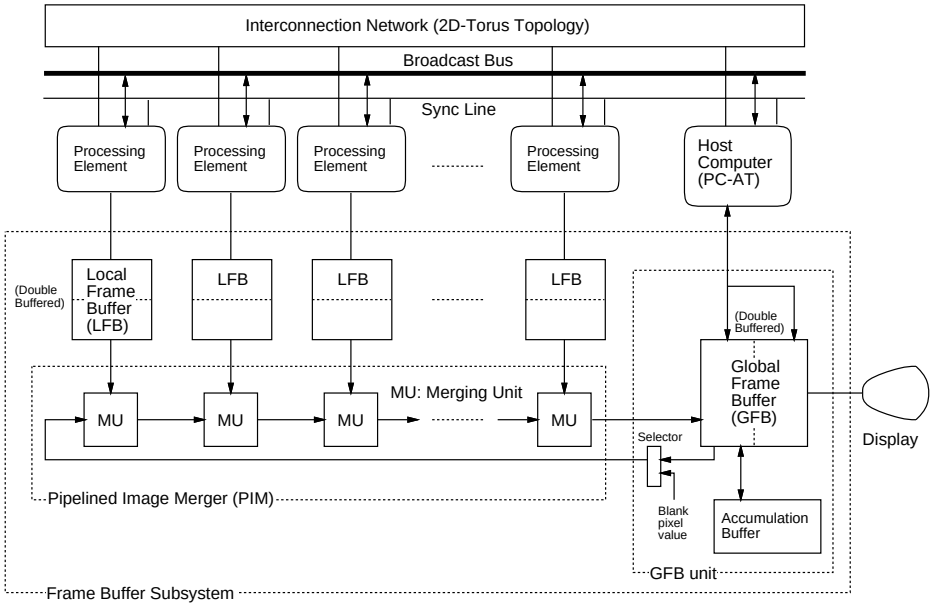


Figure 3: The structure of the VC-1

necessary for calculation, including i860 program codes or polygon data, initially reside in the hard disk of the host computer and are loaded to the local memory of the processors via the broadcast bus.

The host computer has a direct access path to the global frame buffer. This is useful for dumping created images or displaying a cursor on the screen, for example.

3.3 Frame Buffer Subsystem

Local Frame Buffer

The LFB is double-buffered so that the processor can generate the image of the next frame in parallel with the image composition of the previous frame. One buffer is called the front buffer, which is accessed by the processor, and the other is called the back buffer, of which the contents are transmitted to the image merger. These buffers are swapped at the end of each frame generation as well as at the LFB overflow.

Figure 4 shows a block diagram of the LFB unit. The color and depth values are stored in *image memory* consisting of high-speed static RAMs. The image memory capacity for each buffer is 1/4.7 of full-screen capacity. The address space of the image memory is divided into *pages*, each of which has the capacity for storing a patch. Each page is assigned a unique sequential number called a page identifier.

The *patch table* is a high-speed static RAM that maintains the status of every patch. The status consists of a 1-bit field, called a page existence flag, and a 12-bit field, called a page ID field. The page existence flag indicates whether an image memory page is allocated or not. The page ID field contains the identifier of the associated image memory page. Patch size options include 4×4, 8×8, and 16×16 pixels. Usually, the 4×4-pixel configuration yields the best performance. Other patch sizes are designed for evaluation purposes.

The *scan counter* is a free-running counter that controls the transmission from the LFB to the image merger. The counter value advances according to the raster scan order; it starts from the upper-left corner of the screen and ends at the lower-right corner. When it reaches the end, it restarts immediately at the beginning. The clock and reset signals of this counter come from the GFB unit.

The *page ID generator* (PIG) is a counter used for the automatic allocation of a new image memory page. Its value indicates the identifier of the next free page.

The LFB access operations are performed through a two-stage

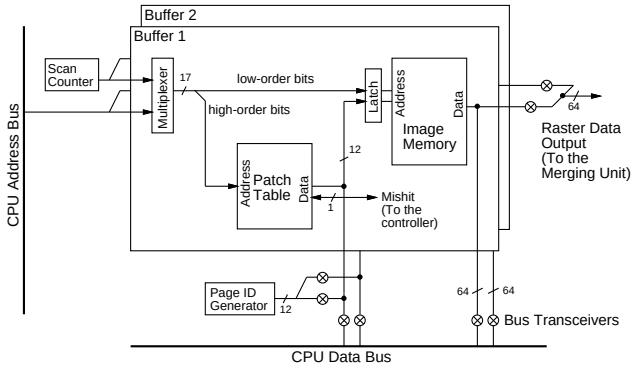


Figure 4: Local frame buffer

pipeline. The first stage reads the patch table and examines the page existence flag. If the flag is false, the first stage also updates the page table entry with the PIG value, sets the page existence flag to true, and increments the PIG. The second stage accesses the image memory. In the first stage, it takes 4 CPU clocks if the page existence flag is false; otherwise, it takes 3 clocks. The second stage always takes 3 clocks. Therefore, LFB access operations can be executed at intervals of 3 or 4 clocks. For more detailed descriptions about LFBs, see [11].

Pipelined Image Merger

The pipelined image merger (PIM) periodically merges the images stored in the LFBs according to the raster scan order. The PIM consists of a linearly-connected array of per-processor elements called merging units (MUs). Each MU receives two pixel values and outputs whichever one has the smallest screen depth. When a corresponding image memory page is not resident on an LFB, the LFB sends a possible maximum depth value to its MU, which makes the MU forward the pixel value from the neighboring unit.

Each MU performs a depth comparison and data selection through a 4-stage pipeline. Therefore, the PIM, as a whole, constructs a $4n$-stage pipeline, where n represents the number of processors. To realize pipeline processing, the scan counter in each

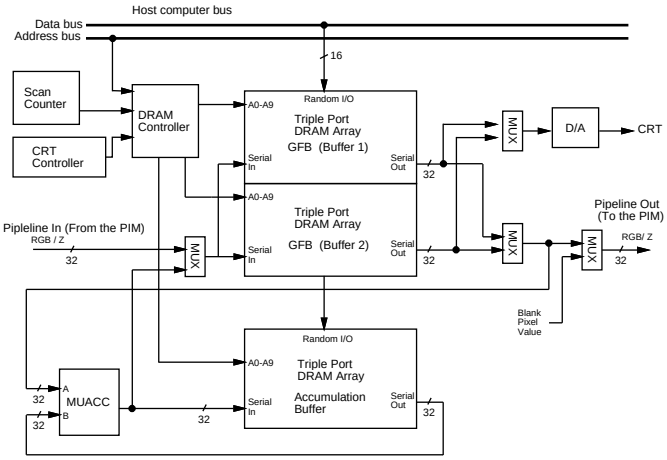


Figure 5: Global frame buffer unit

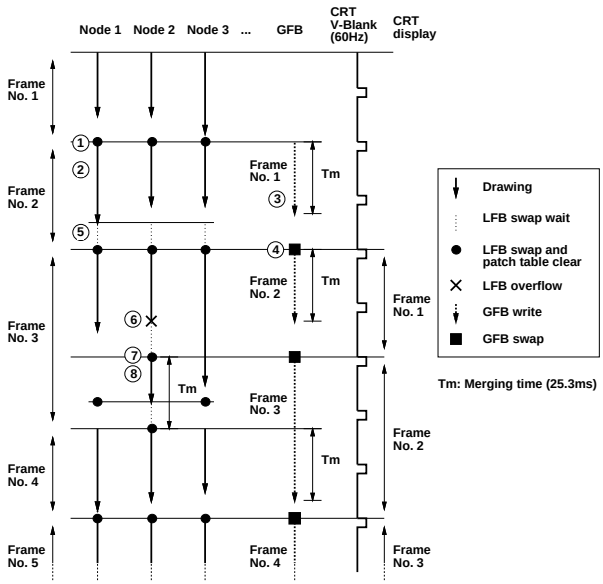


Figure 6: A timing diagram for buffer swapping

LFB is advanced by 4 pixels in comparison with the succeeding LFB.

The time needed for merging one full-screen image (denoted by T_m) is equal to $(M + \beta) \cdot N \cdot t + 4 \cdot n \cdot t$, where M and N are respectively horizontal/vertical screen sizes, t is the PIM clock period, and β is the number of PIM clocks required for per-scanline overhead processing in the GFB unit. The first term represents the time for scanning the entire screen, and the second term represents the pipeline delay time. Obviously, T_m should be less than the target frame interval (33.3ms in our case). In the VC-1, those parameters are $M = 640,\ N = 480,\ n = 16,\ \beta = 20,$ and $t = 80$ nsec., and therefore, $T_m = 25.3$ msec.

Global Frame Buffer

There are two purposes for inserting the GFB between the PIM and the CRT display:

- to deal with the LFB overflow, and
- to make the PIM raster scan asynchronous with the CRT raster scan.
  This is preferable because the scanning frequency can be set independently to a value convenient for each unit, and also because the image composition can be continued even during the blanking periods of the CRT scan.

Figure 5 shows the structure of the GFB unit. The GFB is double-buffered: one buffer is for CRT refresh, and the other buffer receives images from the PIM. Double-buffering prevents incomplete images from appearing on the CRT. Both of the buffers have full-screen capacity and hold both color and depth information. The roles of these buffers are alternated at the end of image generation.

In addition to the GFBs, there is another full-screen buffer called an *accumulation buffer*, which implements anti-aliasing with multi-pass accumulation [6]. The merging unit for the accumulation buffer (MUACC) is used to accumulate the image stored in one of the GFBs into the accumulation buffer.

The GFBs and accumulation buffer are implemented with triple-port DRAMs. The two serial access ports of the triple-port DRAMs are used for data transmissions with the PIM or to refresh the CRT, while the random access port is utilized for direct access from the host computer.

Fast Screen Clearing

In the VC-1, the LFBs are cleared as follows. The image memory is automatically cleared by hardware while it functions as the back buffer. When the back buffer is read out according to the scan counter, it is cleared immediately after each read operation. As for the patch table, the processor must clear all the patch table entries with software after each LFB swap.

Explicit clearance of the GFB is avoided in the following way. During the first round of the image merging scan after the GFB is swapped, blank pixel values are sent to the PIM input (i.e., to the most left MU in Figure 3) so that the old contents of the GFB are replaced. After the first round is completed, the GFB becomes cumulative by sending the old contents of the GFB to the PIM input.

Buffer Swap Control

A timing diagram for buffer swapping is illustrated in Figure 6. When each processor completes image generation, it waits for all of the other processors to complete their tasks using global synchronization, and then it swaps its LFB (①). After this swapping, the processor immediately clears the patch table and starts generating the next frame (②). Parallel with this, the contents of back buffers are merged into the GFB (③). When the merge is completed (i.e., T_m has passed since the last LFB swap), the host computer swaps the GFB after synchronization with the CRT vertical blanking period (④).

When image generation is completed earlier than the GFB swap for the previous frame, the LFB swap is postponed until the completion of the GFB swap (⑤). Without this control, the current frame's image will be mixed with the previous frame's image. An LFB swap on a processor is also blocked if T_m has not passed since the last LFB swap on that processor; otherwise, the unsent contents in the back buffer will be lost.

When an LFB overflow occurs in a particular processor (⑥), an extra LFB swap is executed on that processor (⑦). Like a normal LFB swap, this LFB swap is also postponed until both of the conditions mentioned above are met. Then, the processor clears its patch table and resumes drawing (⑧). A processor's worst-case idle time upon an LFB overflow is 33.3ms. We could reduce this time by utilizing the back buffer's image memory pages immediately after they are sent to the GFB rather than waiting until all the contents in the back buffer are transferred; however, this greatly complicates LFB hardware.

4 Parallel Polygon Rendering on the VC-1

An outline of the rendering routine on the VC-1 is as follows. First, the scene database is transmitted from the host computer to each processor's local memory via the broadcast bus. Polygon data are partitioned into subsets, each of which is loaded to a different processor's local memory. Other database components, such as the eye position or light source information, are duplicated to all of the processors' local memory. Next, the host computer broadcasts a command packet to all the processors, by which each processor starts geometric calculations and rasterization for assigned polygons. When the image generation is completed, frame buffers are swapped as described previously.

In generating animated images, the scene database on each processor's local memory is incrementally modified, i.e., only the inter-frame difference of the database is transmitted between frames rather than retransmitting the whole database. For example, if the eye position has changed between frames, only the viewing matrix is transmitted.

4.1 Polygon Assignment

The mapping of polygons to processors is one of the key factors influencing system performance. Three points have to be considered in relation to this problem: (1) load balancing, (2) the reduction of duplicated polygon vertices among processors, and (3) the minimization of LFB overflows. As for (1), it is not enough for reasonable load balancing only to keep the number of polygons even, since the processing time of a polygon depends on its size and clipping conditions. Problems (1) and (2) have been discussed by some researchers [3, 5]; however, (3) is a new problem specific to our architecture.

The simplest way to achieve load balancing is to allocate independent polygons one by one to the most lightly loaded processor at each moment. If there is more than one choice, the target processor is selected randomly among them. This method is advantageous in (1); however, it is unacceptable with respect to (2) because common vertices shared by neighboring polygons are likely scattered to different processors. Moreover, this method is disadvantageous in (3) because each processor generates a highly discrete image consuming a large number of patches.

A more practical method is to allocate clusters of polygons to the most lightly loaded processor rather than allocating independent ones [5]. The number of polygons in each cluster is called the cluster size (denoted by w). Each cluster contains up to w polygons which are geometrically close to each other. The sequence of polygons specified implicitly by the user through a graphics library tends to have space coherence, i.e., two polygons consecutively indicated by the user have a high probability of being geometrically close. Therefore, it is sufficient when constructing clusters to let the first w polygons in the sequence be the first cluster, let the next w polygons be the second cluster, and so on. With this method, the best balance of the aforementioned three points can be achieved by tuning the value of w.

4.2 Adaptive Parallel Rasterization

The clustering method is not sufficient for reasonable load balancing if some polygons occupy a large area on the screen. For example, the scene shown in Figure 13(c) has one large-sized polygon constructing the table. Such a polygon not only causes load concentration but also increases the LFB overflow. To avoid these problems, we developed a technique called adaptive parallel rasterization (APR).

The APR method rasterizes a polygon in pixel parallel when its estimated size is larger than a predetermined value called the APR threshold; otherwise, the polygon is rasterized in polygon parallel as usual. After geometry calculation, the processors calculate the on-screen area of each polygon. If the area is larger than the APR threshold, the screen coordinates and colors of the vertices of the polygon are broadcast to all the processors using the broadcast bus. When a processor receives the broadcast data, it rasterizes the polygon only for the pixels that are assigned to the processor. Figure 12 illustrates a processor assignment based on APR.

One of the concerns of APR is the overhead due to the area calculation of the polygons. However, on the VC-1 which calculates the area as the vector product of the two edges of a triangle, it is known from experiments that the overhead is less than 3% of the total computation time.

5 Evaluation Results

In this section, the VC-1 architecture and our parallel polygon rendering methods are evaluated using the sample scenes shown in Figure 13. For systems with 16 or less processors, real performance is shown. For systems with 17 to 256 processors, estimated performance is displayed.

Using the VC-1, the rendering time for over 16 processors is estimated. When APR is not performed, the rendering time is estimated using one of the processors and the host computer as follows. Initially, the rendering time for each logical processor of the estimation target system is measured on the physical processor, and then, the system rendering time is calculated as the maximum of the measured rendering times. If APR is active, the rendering time is split into polygon-parallel and pixel-parallel parts, and the estimation is performed in two passes; the first pass determines the polygon-parallel part of the rendering time, and the second pass figures out the complete rendering time. These methods yield exactly the same performance as real performance (See [11] for justification). For a 16-processor system or smaller, it is confirmed from experiments that the performance estimated by this method coincides with real performance.

5.1 Rendering Time

Figure 7 indicates the rendering time for each sample scene varying the number of processors. The rendering time decreases in inverse proportion to the number of processors as long as they do not reach 33.3ms, which is the lower limit determined by the speed of the image merger and the rate of CRT refresh. For scenes composed of many polygons, such as the *ter250* scene (containing 203522 polygons), the excellent scalability of the VC-1 architecture is observed. In fact, a speed 190 times faster is achieved with 256 processors for the *ter250* scene. The maximum rendering performance of the VC-1 calculated from Figure 7 is 514K polygons per second for a 16-processor system and 6.1M polygons per second for a 256-processor system.

Figure 4.2 shows the rendering time as a function of image memory capacity. As the image memory capacity decreases, the rendering time increases because of LFB overflows. In a 16-processor system, it is observed that 1/8 of full-screen capacity is enough to render all the sample scenes without significant performance degradation. Similarly, in a 256-processor system, image memory capacity can be reduced to 1/32 of full-screen capacity.

5.2 The Effect of Adaptive Parallel Rasterization

Figure 9 displays the relationship between the APR threshold and rendering time. As the APR threshold becomes larger (i.e., as the ratio of polygon-parallel rasterization increases), load balancing becomes worse and also LFB overflows increase. Conversely, as the APR threshold becomes smaller, total rasterization time as well as broadcast communication time increase. Therefore, we have a minimum rendering time at some APR threshold. When the *table* scene is rendered with 16 processors, the most appropriate APR threshold is 500, where the rendering time is reduced to 75% of the rendering time without APR (i.e., the rendering time when the APR threshold is infinity).

It becomes clear from further experiments that the main advantageous effect of APR is the reduction of LFB overflows rather than the improvement of load balancing. Without APR, a polygon whose

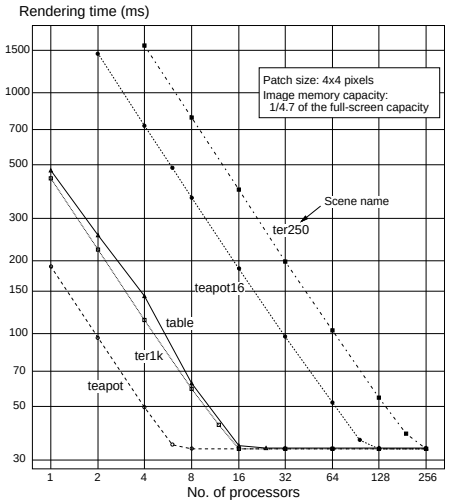
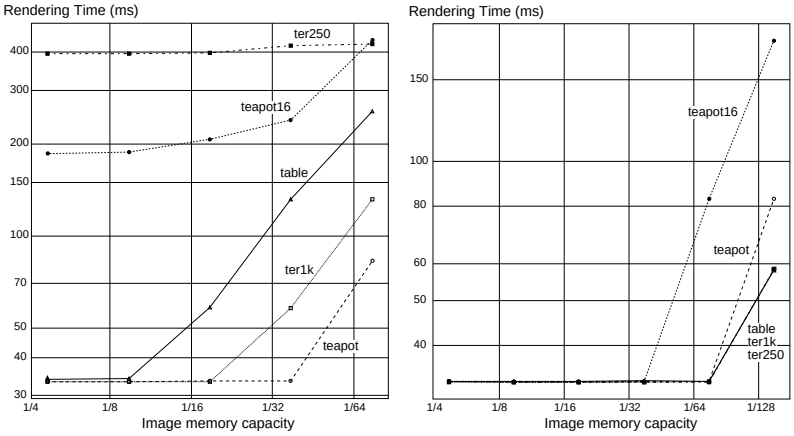


Figure 7: Rendering time versus the number of processors



(a) No. of processors = 16 (b) No. of processors = 256

Figure 8: Rendering time versus image memory capacity

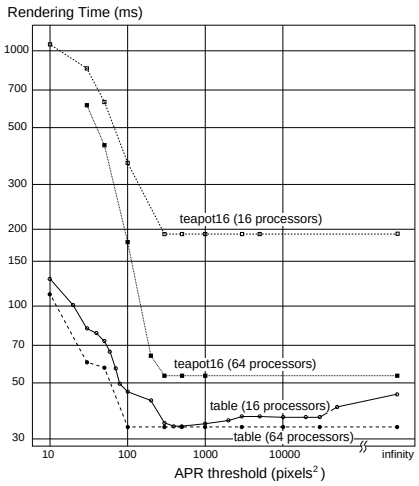


Figure 9: Rendering time versus the APR threshold

Table 1: Comparison with the region-based approach

Memory-saving method	VLFB	RB 2×2†	RB 4×4†	RB 4×4†
Number of buffers	2	2	2	4
Image memory capacity for each buffer	1/8	1/4	1/16	1/16
Total image memory capacity	1/4	1/2	1/8	1/4
Rendering time	33.4ms	51.0ms	73.5ms	66.9ms
Geometry calculation time	12.3ms	14.1ms	14.1ms	14.1ms
Rasterization time	14.7ms	29.1ms	41.5ms	41.5ms
Global synchronization time	1.4ms	7.3ms	17.3ms	10.7ms

† 'RB $m \times n$' means the region-based approach in which the screen is divided into $m \times n$ regions.

image is larger than the image memory capacity would inevitably cause an LFB overflow, which often leads to significant performance degradation. APR avoids LFB overflows by dividing such a large polygon into pieces and effectively making image memory consumption even among processors.

5.3 Comparison with the Region-Based Approach

As described in Section 1, the PixelFlow system reduces image memory capacity by dividing the screen into regions and rasterizing polygons in multiple passes. Each renderer in the PixelFlow system actually has several buffers to improve load balancing. We simulated this memory-saving method on the VC-1 for comparison.

Table 1 contrasts the performances for the two different memory-saving methods when the *ter1k* scene is rendered with 16 processors. We averaged these values over 48 different view angles. The number of buffers indicates the number of independently accessible LFB banks in a processor. For the VLFB, this number is 2 because we use double-buffering. For the region-based approach, this number corresponds to the maximum number of regions which each renderer can hold. The geometry calculation time refers to the total time for transformation, lighting, and region classification. The rasterization time represents the total time for clipping and rasterization. The global synchronization time indicates processor idle time due to load imbalance.

With the VLFB, the *ter1k* scene can be rendered in 33.4ms using an image memory whose total capacity is 1/4 of full-screen capacity. On the other hand, if the region-based approach is taken, the rendering time increases considerably even if the image memory capacity is 1/2. There are two main reasons for this: the increase of rasterization time due to additional clipping tasks for region boundaries and the increase of global synchronization time resulting from the aggravation of load balancing.

It is possible to introduce special hardware for clipping in order to improve the performance for the region-based approach. Nevertheless, this will only lighten the increase in rasterization time. The increase in global synchronization time is essential except for that due to the load imbalance caused by the clipping.

The superiority of the VLFB becomes more remarkable in large-scale systems. In the VLFB, the image memory capacity decreases as the number of processors increases. However, the region-based approach has no such characteristic.

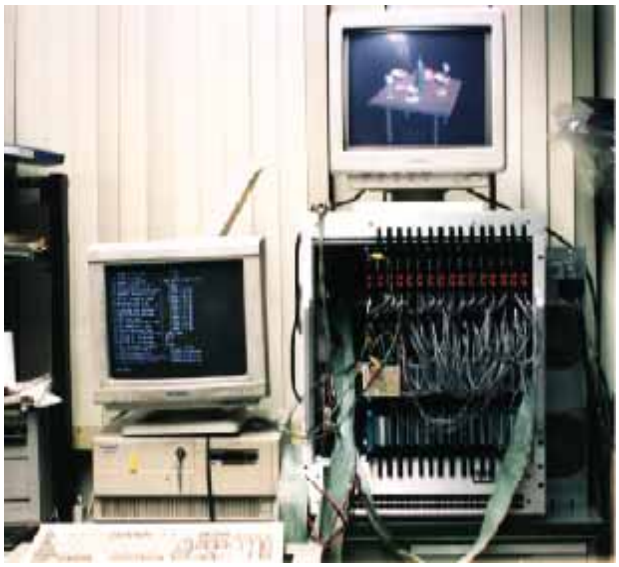


Figure 10: The VC-1

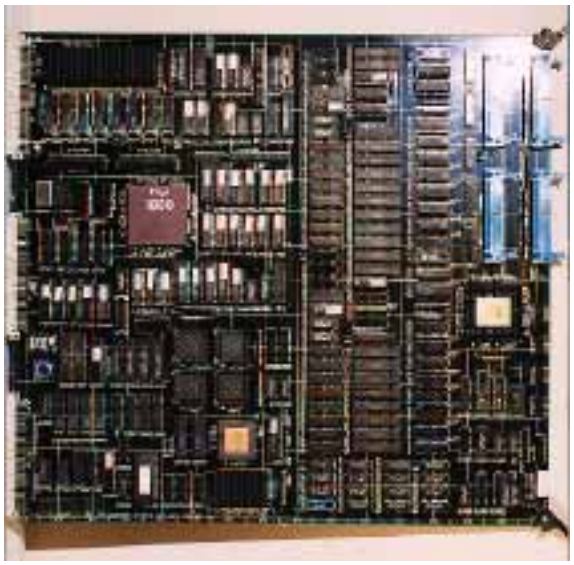


Figure 11: The VC-1 main board
This board contains a CPU, a local memory, communication interfaces, an LFB unit and a merging unit. It is a 6-layer printed circuit board whose size is 356 x 400 mm.

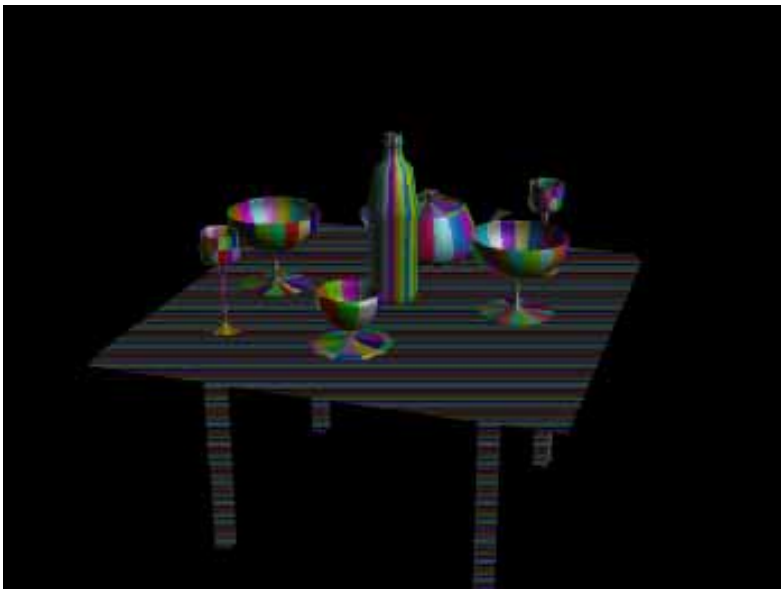


Figure 12: Processor assignment with APR
The color of each polygon identifies the processor to which the polygon is assigned. The large polygons constructing the table are rasterized in pixel parallel with line interleaving.

High-resolution TIFF versions of these images can be found on the CD-ROM in
`S96PR/papers/nishimur`

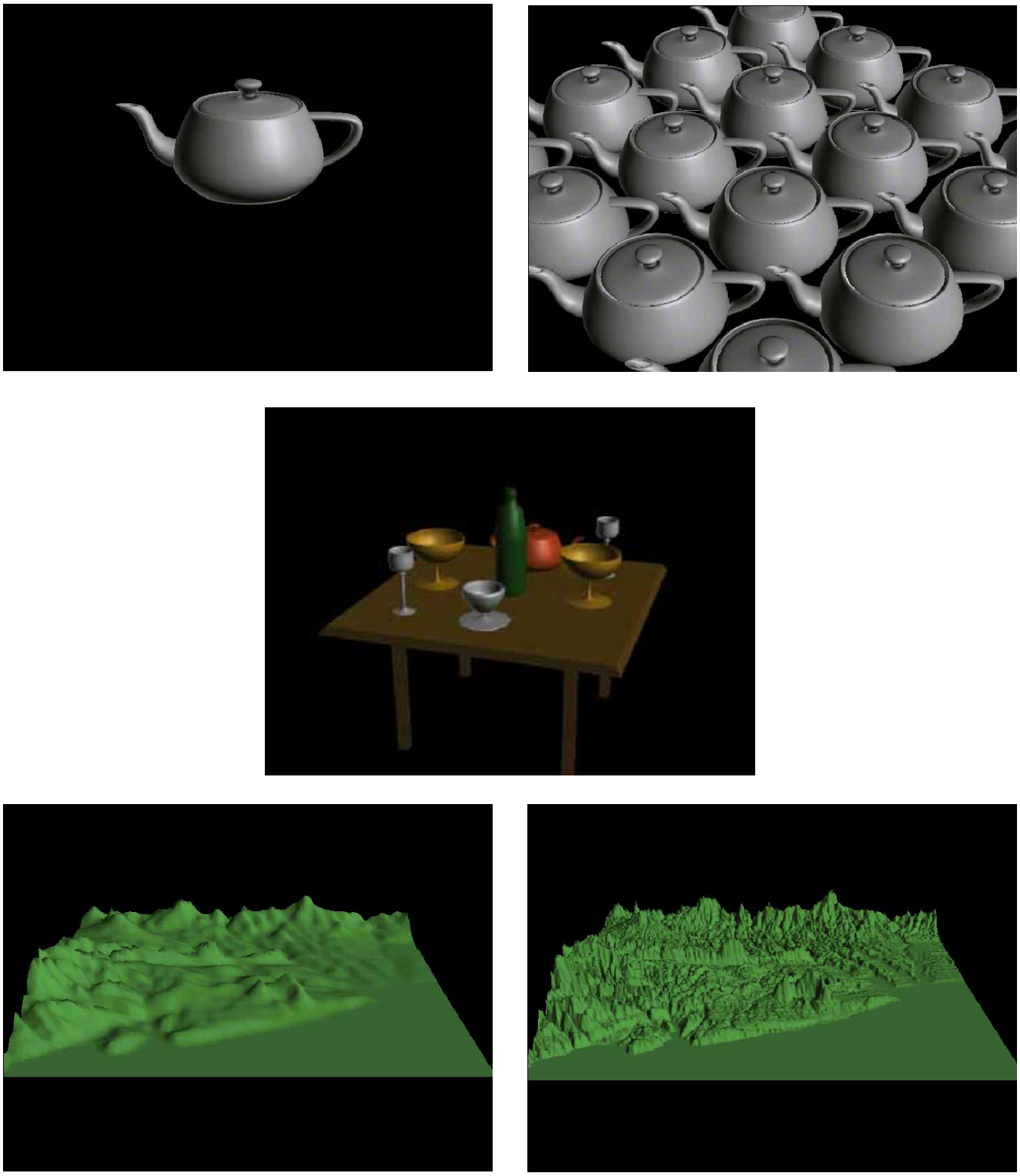


Figure 13: Sample scenes.

(a) teapot (No. of polygons = 4096) (b) teapot16 (No. of polygons = 65536) (c) table (No. of polygons = 10224)
(d) ter1k (No. of polygons = 12482) (e) ter250 (No. of polygons = 203522)

High-resolution TIFF versions of these images can be found on the CD-ROM in
`S96PR/papers/nishimur`

6 Discussions

As the number of processors increases, the APR threshold should be lowered, since the image memory capacity for each processor decreases. One might think that this limits the system scalability because the number of polygons transmitted via the broadcast bus increases. However, the average size of these polygons tends to become smaller as the number of processors increases, and therefore, we believe that this is not a serious limitation.

To perform anti-aliasing with true real-time image generation or to increase screen resolution, the bandwidth of the image merger must be increased in proportion to the number of samples per pixel or the number of pixels on the screen. As a result, an extremely high bandwidth will be required. This is a major weakness of image composition architecture, although we think that future advancement in device technology will take care of this. As for scalability, this still remains regardless of these extensions.

If special hardware for rasterization is added to our architecture, the number of polygons per processor will increase. This does not necessarily mean an increase in image memory capacity. The image memory capacity is strongly related to the number of pixels accessed in each processor, which is represented as $a \cdot p/n$, where a is the average size of polygons, p is the total number of polygons, and n is the number of processors. Since $a \cdot p$ (i.e., the number of pixels totaled over all the polygons) usually remains constant against changes in the number of polygons, $a \cdot p/n$ is invariable no matter how p/n increases. As for scalability, adding special hardware does not influence this.

VC-1 is designed mainly for retained-mode graphics APIs. It is possible to support immediate-mode APIs if the host computer broadcasts the whole scene database for each frame; however, the system will no longer be scalable because the broadcast bus will be bottlenecked. It seems that there is no way to make a scalable machine with immediate-mode APIs since they enforce the serial traversal of the database in the host computer. Even with the retained-mode APIs, the broadcast bus may be bottlenecked when the coordinates of the polygon vertices are constantly changing. However, this problem can be solved by allowing the node processors to determine the coordinates by solving physical equations or dividing curved surfaces, for example.

7 Conclusions

In this paper, we proposed a novel frame buffer architecture called the *virtual local frame buffer* to reduce the memory requirement in parallel graphics machines based on image composition. To evaluate the architecture, we developed a prototype machine called the VC-1. From experiments on the VC-1, it was observed that the virtual local frame buffer technique reduces the image memory capacity to 1/8 of full-screen capacity in a 16-processor system and to 1/32 in a 256-processor system without sacrificing the system's performance. This reduction enables us to use fast static RAMs for the local frame buffer, by which rendering performance is improved.

The adaptive parallel rasterization method, which selects an appropriate parallelization approach based on a threshold value, is essential to the virtual local frame buffer. Without this technique, the system performance would be degraded considerably due to the exhaustion of the image memory. However, the problem of how to find the optimal threshold, which guarantees the avoidance of this exhaustion, is still open.

In the future, we would like to extend our architecture to support both anti-aliasing and texture mapping. We are also planning to add special hardware for rasterization to this architecture to improve polygon rendering performance.

Acknowledgments

We greatly appreciate the financial support of Kubota Computer, Inc. We are also grateful to the members of the past VC-1 project in the University of Tokyo, Ryo Mukai, Reiji Suda and Yukio Sakagawa for their help in hardware design, and Jun Naito for his assistance in software development. Special thanks go to Hiroyuki Nitta for his help in gathering equipment and materials and providing useful technical information. We would also like to thank Thomas Orr for his thoughtful comments.

References

[1] AKELEY, K. RealityEngine Graphics. *Proceedings of SIGGRAPH '93* (August 1993), 109–116.

[2] COHEN, D., AND DEMETRESCU, S. A VLSI approach to Computer Image Generation. Tech. rep., Information Sciences Institute, University of Southern California, 1979.

[3] COX, M. *Algorithms for Parallel Rendering*. PhD thesis, Dept. of Computer Science, Princeton University, 1995.

[4] DEERING, M., WINNER, S., SCHEDIWY, B., DUFFY, C., AND HUNT, N. The Triangle Processor and Normal Vector Shader: A VLSI System for High Performance Graphics. *ACM Computer Graphics 22*, 4 (August 1988), 21–30.

[5] ELLSWORTH, D., GOOD, H., AND TEBBS, B. Distributing Display Lists on a Multicomputer. *ACM Computer Graphics 24*, 2 (March 1990), 147–154.

[6] HAEBERLI, P., AND AKELEY, K. The Accumulation Buffer: Hardware Support for High-Quality Rendering. *ACM Computer Graphics 24*, 4 (August 1990), 309–318.

[7] KUBOTA COMPUTER INC. TITAN2 Technical Overview, 1993.

[8] MOLNAR, S. Combining Z-buffer Engines for Higher-Speed Rendering. In *Advances in Computer Graphics Hardware III* (1988), Springer Verlag, pp. 171–182.

[9] MOLNAR, S., COX, M., ELLSWORTH, D., AND FUCHS, H. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications 14*, 4 (July 1994), 23–32.

[10] MOLNAR, S., EYLES, J., AND POULTON, J. PixelFlow: High-Speed Rendering Using Image Composition. *ACM Computer Graphics 26*, 2 (July 1992), 231–240.

[11] NISHIMURA, S. *A Parallel Architecture for Computer Graphics Based on the Conflict-Free Multiport Frame Buffer*. Doctoral dissertation, Dept. of Information Science, Faculty of Science, The University of Tokyo, 1995.

[12] PALOVUORI, K. An Implementation of an Linearly Expandable Graphics Processing Architecture. Tech. Rep. 8-94, Dept. of Electrical Engineering, Tampere University of Technology, Finland, 1994.

[13] ROMAN, G. C., AND KIMURA, T. VLSI perspective of real-time hidden-surface elimination. *Computer-Aided Design 13*, 2 (March 1981), 99–107.

[14] SCHNEIDER, B.-O. A Processor for an Object-Oriented Rendering System. *Computer Graphics Forum 7*, 4 (1988), 301–310.

[15] SHAW, C. D., GREEN, M., AND SCHAEFFER, J. A VLSI Architecture for Image Composition. In *Advances in Computer Graphics Hardware III* (1988), Springer Verlag, pp. 183–199.

[16] WEINBERG, R. Parallel Processing Image Synthesis and Anti-Aliasing. *ACM Computer Graphics 15*, 3 (August 1981), 55–62.