The SAGE Graphics Architecture

Michael Deering[††]   David Naegle[†]

Sun Microsystems

ABSTRACT

The Scalable, Advanced Graphics Environment (SAGE) is a new high-end, multi-chip rendering architecture. Each single SAGE board can render in excess of 80 million fully lit, textured, antialiased triangles per second. SAGE brings high quality antialiasing filters to video rate hardware for the first time. To achieve this, the concept of a frame buffer is replaced by a fully double-buffered sample buffer of between 1 and 16 non-uniformly placed samples per final output pixel. The video output raster of samples is subject to convolution by a 5×5 programmable reconstruction and bandpass filter that replaces the traditional RAMDAC. The reconstruction filter processes up to 400 samples per output pixel, and supports any radially symmetric filter, including those with negative lobes (full Mitchell-Netravali filter). Each SAGE board comprises four parallel rendering sub-units, and supports up to two video output channels. Multiple SAGE systems can be tiled together to support even higher fill rates, resolutions, and performance.

CR Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture; I.3.3 [Computer Graphics]: Picture/ Image Generation *Display Algorithms*; I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism.

Additional Keywords and Phrases: rendering hardware, antialiasing, graphics hardware, frame buffer algorithms, graphics systems, hardware systems, video.

1 INTRODUCTION

The history of computer graphics hardware has been blazed by high end architectures, ever advancing in features and performance. But such systems have also been ever increasing in cost to develop, and in more recent times many new graphics features have instead made their debut in lower cost implementations aimed at home entertainment markets. But the formidable cost constraints on products for the consumer market precludes many features, interfaces, and levels of performance essential to the higher end needs of the scientific, medical, manufacturing, visual simulation, and other industrial markets. For the forseeable future, the only way to meet these ever growing needs is to architect graphics systems where multiple ren-

[†]901 San Antonio Road, UMPK27-101
Palo Alto, CA 94303-4900
david.naegle@Eng.Sun.Com (650) 786-3939
[††]michael.deering@acm.org

dering chips can be applied in parallel, but appear to the application as a single, high performance rendering pipe. While there have been some attempts to build such systems out of arrays of inexpensive game chips [Stoll et al. 2001], such chips were never designed to be clustered, and so far at least, the resulting systems have not yet been proven effective for traditional high end applications.

In this paper we describe the architecture of *SAGE* (Scalable Advanced Graphics Environment), a major new multi-chip high end rendering system designed to meet the needs of industrial strength 3D graphics applications. A single SAGE board can render over 80 million fully lit, textured, antialiased triangles per second. SAGE brings high quality antialiasing filters to video rate hardware for the first time. To achieve this, the concept of a frame buffer is replaced by a fully double-buffered sample buffer of between 1 and 16 non-uniformly placed samples per final output pixel. The raster of samples is subject to convolution by a 5×5 programmable reconstruction and bandpass filter that replaces the traditional RAMDAC. This convolution is performed on-the-fly during video output, adding less than one additional scan line time of latency. The reconstruction filter processes up to 400 samples per output pixel, and supports any radially symmetric filter, including those with negative lobes, e.g., full Mitchell-Netravali filters. Each SAGE board contains four parallel rendering sub-units, and supports up to two video output channels. Multiple SAGE systems can be tiled together to support even higher fill rates, resolutions, and performance.

The flow of the paper is as follows: after some discussion of market requirements, an overview of the SAGE architecture will be presented, followed by more detailed discussion of the various pipeline stages. The focus here is on the mechanisms that allow us to seamlessly aggregate single chip rasterizers into a more powerful overall system. In the second half of the paper, our approach to video rate antialiasing is described in depth, as it is the most novel feature of the new architecture.

2 HIGH END MARKET NEEDS

Customers have ever increasing needs for higher display resolution, but unfortunately display technology doesn't come close to following Moore's Law: there has been less than a factor of two improvement in available display resolution in more than a decade; less than 10% a year [Akeley 2001]. But IC process capability *has* followed Moore's Law; it has advanced sufficiently to allow us to put photo-realistic software renderers' antialiasing algorithms into real-time hardware. This allows us to increase the effective resolution of current display devices.

The preceding observation was one of the prime motivators of SAGE's very high quality real-time antialiasing pipeline: massive supersampling with large area resampling and bandpass filters in effect provides a back end architecture very similar to software rendering. The pin-count, storage, and computational requirements of these proven algorithms will be beyond the reach of single chip rendering pipelines for several more years to come, even with the an-

nual boost from Moore. This is because pin-count × pin-data-rate = bandwidth-per-chip also lags well below Moore's rate.

Another way to increase display resolutions would be to tile the displays, but we found that most customers who use tiling *also* want each display antialiased. Part of the reason for this is that, so far, projectors adequate for tiling cost more each than a SAGE system.

The historical trend of ever increasing demand for higher triangle rates is coupled with a nearly corresponding decrease in the median size of triangles, so the implied demand for the product of the two, fill rate, is more slowly increasing. This makes sense; applications are displaying scenes of slowly increasing depth complexity but with finer and finer surface tesselations to produce more and more detailed and realistic imagery. But as the median triangle size approaches a pixel, they effectively become micro-triangles, and further reductions in triangle size will have diminishing returns in visible detail or realism. For hardware architectures, this means that more attention should be paid to other measures of increasing the detail and realism of images, such as antialiasing and more complex shader support. This trend was already becoming apparent during the design of SAGE. This is why we put more emphasis on increasing the sampling density than on the triangle rate.

Window system and other legacy API and application support are important market requirements, and in the design of SAGE we made sure that these were not forgotten. Even non-antialiased applications work with SAGE's video resizing.

3 SAGE ARCHITECTURE

3.1 Overview

SAGE's block diagram is seen in figure 1. In this diagram we have expanded out the external buses, internal FIFOs, and internal multiplexers / load-balancing switches so that the overall data flow and required sorting may be more easily seen at the system level. SAGE's inter-chip connections are typically unidirectional, point-to-point, source-synchronous digital interconnects. The top half of SAGE's diagram is fairly similar to other sort-last architectures: command load balancing is performed across parallel transform and rasterize blocks, followed by the "sort-last" tree (the *Sched* chips) interfacing to the frame buffer. In SAGE, however, the frame buffer is replaced with a sample buffer containing 20 million samples. On the output side of the sample buffer, SAGE introduces an entirely new graphics hardware pipeline stage that replaces the RAMDAC: a sample sort tree followed by parallel *Convolve* chips that apply a 5×5 programmable reconstruction and bandpass filter to rasters of samples. Each Convolve chip is responsible for antialiasing a separate vertical column of the screen; the finished pixels are emitted from the Convolves in video raster order.

3.2 Command Distribution

At the top of the pipeline, the *Master* chip performs DMA from the host to fetch OpenGL command and graphics data streams. The DMA engine is bidirectional, and contains an MMU so that application data can reside anywhere in virtual memory: no locking of application data regions is necessary. For geometric primitives, streams of vertex data are distributed in a load-balanced way to the four parallel render pipelines below.

3.3 Rendering: Transform, Lighting, Setup, Rasterization

Each render pipeline consists of two custom chips plus several memory chips. The first custom chip is a *MAJC* multi-processor [Tremblay et al. 2000], the second is the *Rasterize* chip, which performs set-up, rasterization, and textured drawing.

Many previous architectures are sort-middle (following the taxonomy of [Molnar et al. 1994]): they have parallel transform, lighting, and set-up pipelines, but recombine the streams into a one-primitive-at-a-time distributed drawing stage. In architecting SAGE, certain bandwidth advantages motivated our choice of a sort-last architecture: as the average pixel size of application triangles shrinks, the size of the set-up data becomes larger than the actual sample data of the triangle. This also improves efficiency by allowing each rasterization chip to generate all the samples in a triangle, rather than just a fraction of them, as occurs with interleaved rasterization.

The MAJC chip contains specialized vertex data handling circuits that support two fully programmable VLIW CPUs. These CPUs are programmed to implement the classic graphics pipeline stages of transform, clip check, clipping, face determination, lighting, and some geometric primitive set-up operations. The special vertex data circuitry handles vertex strip and mesh connectivity data, so that the CPUs only see streams of non-redundant vertex data most of the time. Thus redundant lighting computations are avoided, and vertex re-use can asymptotically reduce the required vertex processing operations to 1/2 of a vertex per triangle processed when vertex mesh formats are used.

To support a sample buffer with programmable non-uniform sample positions, the hardware fill algorithms must be extended beyond simple scan-line interpolation. Generalizations of plane equation evaluation are needed to ensure correctly sampled renderings of geometric primitives. Furthermore, the sample fill rate has to run 8 to 16 times faster than the rasterize fill of a non-supersampled machine just to keep up. The aggregate equivalent commodity DRAM bandwidth of SAGE's eight 3DRAM memory interleaves is in excess of 80 gigabytes per second.

The Rasterize chip rasterizes textured triangles, lines, and dots into the sample buffer at the current sample density. It also performs some imaging functions and more traditional raster-op and window operations. Each MAJC + Rasterization pipeline can render more than 20 million lit textured supersampled triangles per second. Each Rasterizer chip has it own dedicated 256 megabytes of texture memory. This supports 256 megabytes of user texture memory, at four times the bandwidth of a single pipe, *or* up to one gigabyte of texture memory, when applications use the OpenGL targeted texture extension (this is a common case for volume visualization applications).

For textured triangles, the Rasterize chip first determines which pixels are touched (even fractionally) by the triangle, applies layers of (MIP-mapped, optionally anisotropic filtered) texturing to each pixel, determines which (irregular) sample positions are within the triangle, and interpolates the color, Z, and alpha channel data to each sample point. The output is a stream of sample rgbaZ packets with a screen pixel xy address and sample index implying the sample's sub-pixel location within that pixel. Each Rasterize chip has two external output buses so that the first stage routing of sample data to sample memory is performed before the samples leave the Rasterize chip.

3.4 Sort-Last

Below the Rasterize chips lies a network comprising two *Sched* chips to route samples produced by any of the Rasterize outputs to any pixel interleave of the sample buffer below. As samples arrive in the Sched chip input FIFO from Rasterize chips, they are routed into the appropriate second stage FIFO based on their destination memory interleave. The output of each second stage FIFO is controlled by the load balancing switch for its memory interleave. Each switch acts like a traffic light at a busy intersection; traffic from one source is allowed to flow unimpeded for a time while the other sources are blocked. This (programmable) hysteresis in the flow of
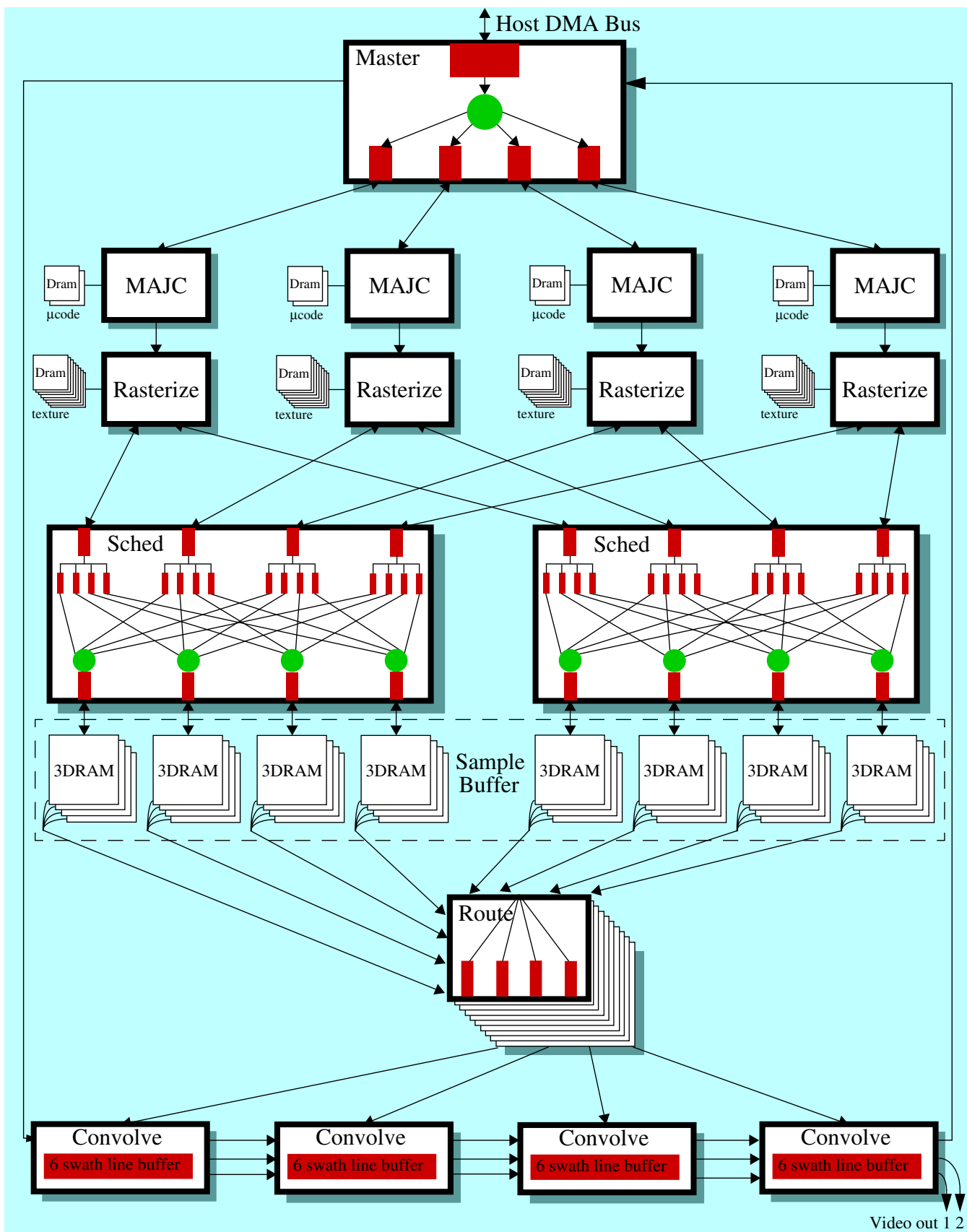
Figure 1. SAGE block diagram. Thick boxes are CUSTOM CHIPS. Red boxes are FIFOS. Green circles are load balancing SWITCHES.

sample data from different Rasterize chips ensures good cache locality within the 3DRAM memories below.

The third layer of FIFOs in the Sched chip is a final sample pre-write queue in front of a single memory *interleave*. (An interleave is a group of 4 3DRAM chips connected to the same pre-write queue.). The Sched chips snoop this queue to perform 3DRAM cache prefetches, before scheduling the sample writes into the sample buffer.

Because the parallel rasterized sample streams merge together here, the Sched chip is also the place where special control tokens enforce various render-order constraints. For example, most algorithms that make use of the stencil buffer require at least two passes—one to "prepare" the stencil buffer with a special pattern, and then another pass with sample writes conditionally enabled by the presence of that special stencil pattern. Clearly all stencil writes of the first pass must complete before any of the second pass sample writes can be allowed to go forward. When a given interleave on a given Sched chip encounters a special synchronization token marking a hard ordering constraint (e.g, the boundary between the two passes), then no more samples from that rasterizer will be processed until the other three rasterizer inputs have also encountered and stopped at the synchronization token. When this occurs, all samples generated by primitives that entered the SAGE system before the synchronization token have been processed (the first pass in our example), and now it is okay to allow the pending samples that entered the system after the synchronizing token to proceed. The OpenGL driver knows to generate this ordering token when it is in unordered rendering mode, and then sees a command to transition to ordered rendering mode immediately followed by a command to change back to unordered rendering mode. Other more complex situations are supported by more complex special token generation by the driver.

As controllers of the 3DRAM chips, the Sched chips also respond to requests from the Convolve chips for streams of samples to be sent out over the 3DRAM video output pins to the parallel Convolve chips to generate the video output.

3.5 The Sample Buffer

The sample buffer consists of 32 3DRAM chips, organized into eight independent interleaves of four chips each. On the input side, four 3DRAMs share a single set of control, address, and data lines to one of four sets of memory interleave pins on a Sched chip. On the output side, each 3DRAM outputs 40-bit samples by double pumping 20 video output pins. Each of these pins has an individual wire to a *Route* chip, for a total of 640 wires entering the lower route network.

Logically, the sample buffer is organized as a two dimensional raster of lists of samples. All lists are the same length, because all pixels on the screen have the same number of samples. The list-order of a sample implies its sub-pixel location; the Rasterize and Convolve chips contain identical sample-location tables accessed by the sample index, so no space is allocated within the sample buffer for the sub-pixel location of the sample. The memories are interleaved per-sample: adjacent samples in a list are in different 3DRAM packages.

Unlike first-generation 3DRAM components, the new 3DRAMs used on SAGE contain an internal 2:1 multiplexer driven by each sample's window ID, so sample-by-sample double-buffering occurs inside the 3DRAM chip. Thus only the final rgb alpha/window control bits emerge in the 40-bit-per-sample output packet.

The size of the sample buffer is enough to support 1280×1024 double-buffered samples with Z at a sample density of 8, or 1920×1200 at a sample density of 4. The high sample densities require correspondingly high render bandwidth into the sample buffer. This was achieved by a new generation of 3DRAM [Deering et al. 1994]. Because 3DRAM per-

forms z-buffer compare and alpha buffer blending internally, the traditional z-buffer read-modify-write operation is simplified into just a write operation. The important operation of clearing the sample buffer for a new frame of rendering is also potentially adversely affected by the high sample density, but this too is greatly accelerated by the 3DRAM chips; initializing all samples in a 1280x1024x8 sample raster takes less than 200 usec. (less than 2% of a 76-hz frame time).

3.6 Sample Raster Delivery

The 640 outputs of the sample buffer feed into an array of 10 Route chips. Each Route chip is a 2-bit slice of a router function. Each Route chip connects to 2 output data pins from each of the 32 3DRAMs, and can redirect this data to any of the four Convolve chips attached to it below. Because of the need for the Convolve chips to be fed a contiguous vertical swath of the pixel interleaved sample buffer, samples are read from the sample buffers in quarter scan line wide, one pixel high bursts directed at one of the four Convolve chips. (More details will be discussed in the Convolve section.) It is the job of the Route chip to absorb these bursts into internal FIFOs, and then dribble them back out to their destination Convolve chip.

3.7 Convolution, CLUT, Video Timing

Finally, the four Convolve chips perform the reconstruction and band limiting filtering of the raster stream of samples, producing pixels that are fed into the next Convolve chip before final video output. The Convolve chips replace the digital portion of the RAM-DAC; they contain color look-up and gamma tables, as well as video timing generator, cursor logic, and genlock interface. The Convolve chips do not contain D/A converters. Instead, the Convolve video outputs are digital, to support various existing and emerging digital video interfaces. Two high quality external D/A converters and an S-video interface are on the SAGE video daughter board to support analog video devices.

The traditional graphics hardware taxonomy refers to this section as *display*, however the RenderMan term *imaging pipeline* may be a more accurate description of this new functionality.

The next several sections describe the convolution processing in more detail, starting off with a discussion of previous attempts to implement video rate antialiasing.

4 CONVOLUTION INTRODUCTION

For over a decade now, users of most (batch) photorealistic rendering software have been able to obtain high quality antialiased imagery, usually by means of various supersampling algorithms. However, for real-time hardware systems, cost constraints have precluded the deployment of all but the most simplistic approximations to these algorithms. Fill rate limitations make real-time generation of enough samples challenging. Restrictions in hardware polygon fill algorithms can preclude sub-pixel spatially variant sampling. Memory costs and bandwidth limits have prohibited use of double-buffered supersampled frame buffers. Finally, the computational cost of real-time antialiasing reconstruction filters has limited hardware implementations to box or tent filters, which are inferior to most software reconstruction filters.

Various alternatives to stochastic supersampling have been tried over the years in attempts to avoid high hardware costs, but to date all such attempts have limited the generality of the rendering and have not seen much use in real-time general-purpose graphics hardware systems. Their use has been confined to applications whose structure could be adequately constrained: flight simulation and some video games.

Once the non-uniform supersampling approach is taken, a number of other rendering effects can be performed by applications through

the use of multi-pass algorithms and user programmable sample mask patterns. These include motion blur, depth of field, anisotropic texture filtering, subject to supported sample densities. In this paper we do not directly address these features, rather, we focuses on the basic back-end architecture required to support filtered supersampled buffers.

5 PREVIOUS ANTIALIASING WORK

5.1 PREVIOUS WORK, SOFTWARE

Antialiasing has a rich and detailed history. The mainstream approach in recent years has been to evaluate the image function at multiple irregularly spaced sample points per pixel, followed by applying a reconstruction filter and then resampling with a low-pass filter. Originally referred to as stochastic supersampling, the basic idea is to trade off visually annoying aliasing artifacts (jaggies) for less visually perceptible noise. [Glassner 1995] contains an excellent survey and discussion of the many variants of this approach that have been implemented over the years. The pioneering commercial software implementation of this approach is Pixar's Photo-Realistic RenderMan [Cook et al. 1987][Upstill 1990].

PREVIOUS WORK, HARDWARE

5.2 Flight simulators, back-to-front sorting-based algorithms

Real-time antialiasing has been a requirement of flight simulation hardware for several decades. However, most of the early work in the field took advantage of known scene structure, usually the ability to constrain the rendering of primitives to back-to-front. But these algorithms do not scale well as the average scene complexity grows from a few hundred to millions of polygons per frame.

5.3 Percentage Coverage Algorithms

Some systems, for example [Akeley 1993][Winner et al. 1997], have employed polygon antialiasing algorithms based on storing extra information per pixel about what polygon fragments cover what fractions of the pixel. In principle, algorithms of this class can produce higher quality results than even supersampling techniques, because the exact area contribution of each polygon fragment to the final visible pixel can be known. In practice, hardware systems can only afford to maintain a limited amount of shape information about a limited number of polygon fragments within each pixel. For scenes consisting of small numbers of large polygons, most polygons are very much greater in area than a pixel, and the vast majority of pixels are either completely covered by just one or two polygons. Occlusion edges and silhouettes would then have their jaggies removed. With care, even corner cases when more than two polygons of one continuous surface land within one pixel can often be merged back into the single polygon case.

However, with today's typical polygon shrinking towards a micro-polygon, such algorithms rapidly become confused, causing unacceptable visible artifacts.

5.4 Multi-pass Stochastic Accumulation Buffers

The first attempted support for general full scene antialiasing independent of render order in near-real-time hardware was the multi-pass stochastic accumulation buffer [Deering et al. 1988][Haeberli and Akeley 1990]. The approach here was to render the scene multiple times with different sub-pixel initial screen offsets, and then combine these samples with an incremental filter into an accumulation buffer before final display. However, the multiple passes and

the overhead of filtering and image copying reduced the performance of the systems by an order of magnitude or more, while still adding substantial cost for the (deeper pixel) accumulation buffer. As a result, while the technique has been supported by multiple vendors, it has never found much use in interactive applications. Also, because the sub-pixel sample positions correlate between pixels, the final quality does not match that of software systems.

5.5 Supersampling

Some architectures have implemented subsets of the general supersampling antialiasing algorithm. [Akeley 1993] and [Montrym et al. 1997] implement a one through eight sample-per-pixel rendering into a single-buffered sample buffer. When sample rendering is complete, the samples within each pixel are all averaged together and transferred to an output pixel buffer for video display. The combined reconstruction and low-pass filter is thus a 1×1 box filter, and does not require any multiplies. The 1x1 region of support also eliminates the need for neighboring pixels to communicate during filtering. While the quality does not match that of batch software renderers, the results are appreciably better than no supersampling, and have proven good enough to be used in flight simulation and virtual set applications, among others. [Eyles at. al. 1997] implemented supersampled rendering with a 1×1 weighted filter.

At the lower end, some simple processing for antialiasing is beginning to show up in game chips. [Tarolli et al. 1999] appears to be an implementation of a 2×2 single buffered supersampled buffer, but it is not clear if other than box filtering is supported. The nVidia Geforce3 supports sample densities of either 2 or 4, with either a 1×1 box filter, or a 3×3 tent filter [Dominé 2001]. The resultant quality is better than no antialiasing at all, but still far from the quality of batch photorealism software. The frame rates, however, do suffer almost linearly in proportion to sample density.

The OpenGL 1.3 specification does contain support for supersampling, but only in the context of applying the filtering *before* the render buffer to display buffer swap.

6 SAGE SUPERSAMPLING ISSUES

6.1 Programmable Nonuniform Sample Pattern

An important component of high-quality supersampling based antialiasing algorithms is the use of carefully controlled sample patterns that are not locally periodic. Today's best patterns are constrained random perturbations of uniform grids. Software algorithms can afford the luxury of caching tens of thousands of pixel area worth of pre-computed sample patterns. On-chip hardware is much more severely space constrained; SAGE only supports a pattern RAM of 64 (2x2 pixels x 16 samples) of 6-bit x and 6-bit y sub-pixel offset entries. However, the effective non-repeating size of this pattern is extended to 128x128 pixels by the use of a 2D hardware hash function that permutes access to the pattern entries. The effectiveness of this hash function can be seen in Figure 5, where each large colored dot corresponds to a sample. Because the table is so small, it is easily changeable in real-time on a frame-by-frame basis, supporting temporal perturbation of the sample pattern.

Note that in the SAGE system the sample tables for the frame currently being displayed are stored in the Convolve chips, while the sample tables for the frame currently being rendered are stored in the Rasterizer chips. If the tables are not static, system software must ensure that they are updated at the appropriate time boundaries.

7 CONVOLVE CHIP ARCHITECTURE DETAILS

One of the primary ways in which our architecture differs from previous systems is that there is no attempt to compute the antialiased pixels on the render side of the frame buffer. As far as the sample buffer is concerned, the output display device is capable of displaying supersamples; it is up to the back end reconstruction filter pipeline to convert streams of supersamples into antialiased pixels on the fly at full high-resolution video rates.

The peak data rates required to support this are impressive: the frame buffer has to output 1.6 billion samples per second, or approximately 8 gigabytes per second of data. Real-time high-quality filtering of this much data is beyond the capabilities of today's silicon in a single chip. Thus, we had to find a way to spread the convolution processing of this fire-hose of data across multiple chips. As seen in Figure 1, our convolution pipeline is split up into four chips. Each chip is assigned a different vertical swath of the screen's samples. Because reconstruction filters of up to 5×5 are supported, each of these vertical swaths must overlap their horizontal neighbors by up to 2 pixels (half the filter width). The final video stream is assembled as video is passed from chip to chip; each chip inserts its portion of each scan line into the aggregate stream. The last chip delivers the complete video stream. (An optional second video stream also emerges from this last chip.)

The 5×5 filter size also implies that each sample will potentially be used in up to 25 different pixel computations. To avoid re-fetching samples from off-chip, 6 swath-lines worth of sample data is cached on each Convolve chip. (This RAM consumes half the active area of the chip.)

The internal architecture of the chip is shown in Figure 2. The video generation process for each chip starts with the generation of a raster of convolution center (output pixel center) locations across and down each swath. As the convolution center location moves, sample data is transferred from the swath-line buffers into a 5×5 filter processor array.

A schematic of a filter processor is shown in Figure 3. Each filter processor is responsible for all the samples from one pixel from the sample buffer; the 5×5 array has access to all the samples that may contribute to a single output pixel. The filter processor computes the contribution of its samples to the total convolution; the partial results from all 25 filter processors are then summed to form the unnormalized convolution result. Because of the nonuniform, non-locally repeating properties of good sample patterns, it is not feasible to cache pre-computed convolution filter coefficients. Instead, each filter processor contains circuitry for dynamically computing cus-
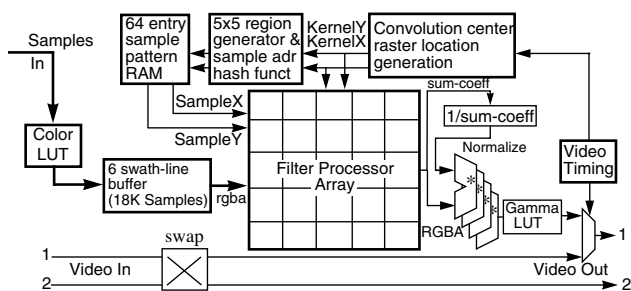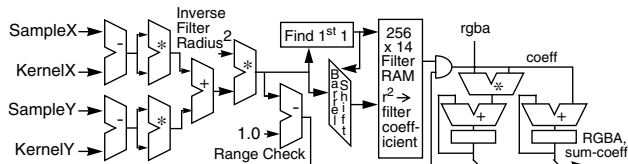
tom filter coefficients for arbitrary sample locations. It also contains the multiplier-accumulator that actually weights its samples.

This filter coefficient computation proceeds as follows. First, the sample location relative to the convolution center is computed by subtracting the sample xy location (generated by the sample pattern RAM) from the convolution center xy location. Squaring and summing the these delta xy components results in the squared radial distance of the sample location from the convolution center. This squared distance is scaled by the square of the inverse filter radius; results greater than unity force a zero filter coefficient. Next the squared distance is encoded into a 3-bit exponent, 5-bit mantissa (+1 hidden bit) floating-point representation. This 8-bit floating-point number is then used as an index into a (RAM) table of squared distance vs. filter coefficients. From a numeric linearity point of view, the squaring and floating-point encoding nearly cancel out, resulting in accurate, relatively equally-spaced filter coefficients. This can be seen in a plot of the synthesized filter values vs. distance in Figure 4.

The filter coefficient output of this table is a signed 14-bit floating-point number, which is used to weight the rgba sample values. The multiplied result is converted back into a 27-bit fixed-point number, and directed into a set of summing trees. A separate running sum of applied filter coefficients is similarly calculated.

Thus our hardware places only two restrictions on the reconstruction filter: it must be radially symmetric in the convolution space, and the filter radial cross section has to be quantized to 256 values. Note that through non-uniform video and/or screen space scalings, elliptical filters in physical display space can be supported. Separable filters have theoretical advantages over radial filters, but radial filtering was less complex to implement in hardware.

Technically our filter is a *weighted average filter*, because of how we handle filter normalization. We perform a floating-point reciprocal operation on the sum of the filter coefficients, and a normalizing multiply on r, g, b, and a. There was an unexpected advantage in using dynamic normalization: in simulations, the error compared to the exact solution came out well below expectations. This is because slight errors in coefficient generation produce a similar bias in the normalization value and are mostly canceled out. Remaining numeric errors in coefficient generation have less perceptual effect because they are equivalent to a *correct* coefficient at an incorrect estimation of the sample distance from the center of the filter (error in sample position). But samples should be quite representative of the true underlying image in their vicinity. Most visible errors in antialiased output are due to a sample just missing a significant change in the underlying image (e.g. from black to white). The contribution to image output errors due to errors in computing filter coefficient values is quite small by comparison.

Hyper-accurate filtering can preserve the quantization present in the original samples (sometimes called contouring). To mitigate contouring, we *dither* 12-bit rgba samples computed during rendering to the 10-bit rgba values actually stored in the sample buffer. Con-
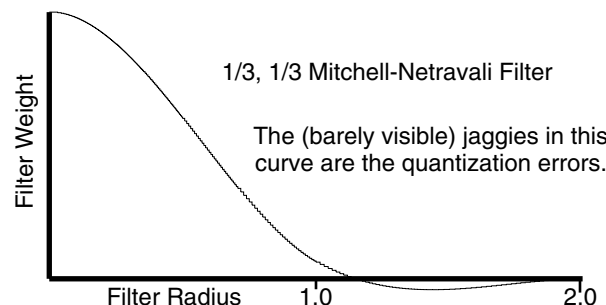
Figure 2: Convolution chip architecture.

Figure 3: Filter Processor Detail.

1/3, 1/3 Mitchell-Netravali Filter

The (barely visible) jaggies in this curve are the quantization errors.

Figure 4: Numerical accuracy of filter representation.

volution reconstruction of dithered sample values effectively reverses this dithering, achieving 12 bits of accuracy per rgba component.

7.1 Video Outputs

Up to two simultaneous, potentially asynchronous, video rasters can be generated in parallel by partitioning the four Convolve chips into two subsets; both video streams will emerge from the digital video out ports of the last Convolve chip. The swap circuit shown in Figure 2 allows each Convolve chip to add its results to either of the incoming streams, and pass the other through unmodified. (There is also a post-processing swap not shown.)

One use for this second channel is to be able to read the antialiased image back into the computer, through the outer ring bus shown in Figure 1. Without this option, the host computer would have no way to get a copy of the antialiased image! This is useful when performing antialiased rendering intended for later reuse as reflection maps, etc.

The Convolve video timing circuit can run as either a sync master or as a sync slave genlocked to an external sync source. The two video streams can be sub-regions of a what the window system and rendering system think of as a single display (useful for tiling two lower-resolution projectors/displays). Alternatively, the two video sources can be two separate, potentially asynchronous, image regions with potentially different sample densities.

7.2 Video Resizing

A key benefit of the SAGE architecture is that video resolution is determined by the convolution hardware, not by rendering hardware. Thus the same hardware used for antialiasing also provides an extremely high quality video rescaler, with better filtering quality than is possible with an external scaler because it operates on the original samples, not pixels, *and* SAGE correctly performs the filtering in linear light space. One use of this is to generate NTSC video of arbitrary zoomed and panned sub-regions of a higher resolution display, as might be used to document a software program. A more important use is in systems with real-time guarantees: to conserve fill rate, the actual size of the image rendered can be dynamically reduced, and then interpolated back up to the fixed video output size. So a flight simulator using a 1280×1024 video format might actually be rendering at 960×768 when the load gets heavy, saving nearly half the fill time. The system described in [Montrym et al. 1997] also supports dynamic video resizing, but uses a simple tent filter, and performs the filtering in a non-linear (post-gamma) light space.

7.3 Fully Antialiased Alpha Channel

SAGE's sample filtering algorithm operates not only on the rgb channels, but also on stored double-buffered alpha if enabled. For example, for virtual set applications this means that SAGE automatically generates a very high quality "soft key" signal for blending antialiased edges of virtual objects in front of physical objects, as well as blending variably transparent rendered objects in front of physical objects.

8 CHOICE OF RECONSTRUCTION FILTER

Because it is programmable, the choice of reconstruction filters can be left to the end user, but in general we have found that the same Mitchell-Netravali family of cubic filters [Mitchell and Netravali 1988] used in high quality software renderers work well for hardware. The choice of reconstruction filter has a subjective component: some users prefer smoother filters that banish all jaggies at the expense of a slight blurring of the image; other users desire a filter that preserves sharpness at the risk of a few artifacts. There is also a display-device-specific aspect to the choice of reconstruction fil-

ter: to get close to the same end-user "look" on a CRT vs. a flat panel LCD display, slightly different filters are needed. While not of general use, more exotic filters can be used to help simulate the appearance of special imaging devices.

8.1 Effects of Negative Lobes

One of the prices that must be paid for the use of high quality reconstruction filters is occasional artifacts ("ringing" or "fringing") due to the presence of negative lobes. Our filter hardware clamps negative color components to zero, and it also keeps a histogram of the frequency and extent of such occurrences. This histogram data can be used to dynamically reduce the negative lobes of the reconstruction filter if artifacts are too severe.

9 Legacy and Compatibility Issues

There are several legacy and compatibility issues that SAGE must address. Many of these are handled by properties associated with window ID tags that are part of *each* sample.

One example is support for applications that were programmed assuming a non-linear light space and/or a pseudo color space. The non-linear light space is typically a particular gamma space. SAGE supports these applications by providing pseudo color, direct color, and non-linear true color LUTs as specified by window ID properties of samples. In SAGE, these LUTs are applied to samples before the convolution. Of course, most 3D rendering is performed in linear light space, and so can by-pass these pre-convolve LUTs. This pre-processing ensures that all sample inputs to the antialiasing convolution process are in the same linear light space. After the convolution process generates (linear light space) pixels, the pixels are converted to the proper non-linear light space (e.g. gamma correction) for the particular display device attached to the system.

Not all pixels should be antialiased. Proper emulation of 2D window system rendering and legacy applications require accurate emulation of all those jaggies. Our solution is to disable any filtering of such pixels via a special property of the window ID tag of the pixels of such windows. Instead, when so tagged, a sample (typically the one closest to the convolution center) is chosen to be output in place of the convolution result. Thus it is possible for the screen to simultaneously support antialiased and non-antialiased windows. Because our filter has a 5×5 extent, care must be taken to ensure that such unfiltered pixels do not contribute any samples to nearby filtered pixels. This is the case, for example, when a non-antialiased window occludes an antialiased window. Once again the dynamic filter normalization circuit comes to the rescue; we simply don't apply any filter coefficients from aliased pixels within the 5×5 window of an antialiased pixel, and still get unit volume under the kernel. The same approach is also used to eliminate artifacts at the visible video border, in place of the traditional approach of adding an extra non-visible strip of 2 pixels all around the full screen.

Other legacy issues include proper support of traditional antialiased lines when also subject to supersampling and filtering. Our goal is to allow as much as possible for existing applications to move to full scene antialiased operation with minimal source code changes.

10 RESULTS

10.1 Images

Figure 5 is an image from the SAGE debugging simulator, and shows the details of our sampling pattern for sample density 16 rendering. The intensity of each dot corresponds to the computed sample value for rgb; the green lines are triangle tesselation boundaries; the faint red grid lines are the pixel boundaries. 11 triangles are shown: a 12-segment radius-3 pie wedge with one slice missing.

Because SAGE's native output environment is a display, the next set of images are digital photos of functional SAGE hardware driving a CRT screen. Figures 6 through 10 are shots of a portion of a 1280×1024 CRT display. Each shows the same portion of the same object, a honeybee. The differences are in the sample count and reconstruction filter. Figure 6 shows one (uniformly spaced) sample per pixel, with no reconstruction filtering. Figures 7 through 10 are rendered using a sample density of 8. Figures 7 and 10 use the 4×4 Mitchell-Netravali 1/3 1/3 filter of Figure 4. Figure 8 uses a diameter 4 cylinder filter, and shows considerable blur. Figure 9 uses a Laplacian filter, and shows enhanced edges. Figure 10 is a wider (approximately 800×800 pixel) shot of the bee.

10.2 Comparison to RenderMan

During SAGE's development, Pixar's Photorealistic RenderMan (PRMAN) was used to verify the quality of the antialiasing algorithms. Custom RenderMan shaders were written to mimic the different lighting algorithms employed. The same scene descriptions, camera parameters, sampling rates, and reconstruction filters were used to generate images from both renderers. The resulting images cannot be expected to be numerically identical at every pixel, primarily because of the different sample patterns used, as well as the different numeric accuracies employed. (PRMAN uses full 32-bit IEEE floating-point arithmetic internally.) So as a control, we also ran PRMAN at a sample density of 256. Numerically, comparing our hardware 16 sample rendering with that of PRMAN, fewer than 1% of the pixels differed in value by more than 6% (the contribution of a single sample). However, about the same variance was seen between the 16-sample and 256 sample PRMAN images. This explains the visual results: in general, expert observers could not determine which image was rendered by which system.

10.3 Data Rates and Computational Requirements

A double-buffered sample buffer supporting 8× supersampled 1280×1024 imagery requires storage of over 20 million samples (approximately an eighth of a gigabyte, including single-buffered Z). For 76 Hz video display, because of overheads and fragmentation effects, we designed in a peak video output bandwidth of 1.6 billion samples per second, or 8 gigabytes per second. (Note that the render fill data rate has to be several times *larger than this* to support interesting depth complexity scenes at full frame rates.)

A 5×5 filter at a sample density of 4 requires $25*4*4$ floating-point multiply-adds per output pixel, or 800 operations per pixel. A similar number of operations are needed to generate all the filter coefficients per pixel. At peak video output rates of 250 MHz, the total operation count per second exceeds 0.4 teraflops. While these are numbers generated by specialized hardware, it is important to note that a (much) greater number of flops *are* consumed by general purpose computers running equivalent software antialiasing algorithms for equivalent work.

10.4 Scalable

The SAGE chip set was designed to scale the performance of a two chip rendering sub-system into a parallel pipeline rendering system. Not all the combinatorial variations of chip configurations allowed by the SAGE architecture have been described in this paper on the first implementation of a SAGE chip-set based system. In addition, each current SAGE board has all the necessary hooks to be scaled at the computer system level, to support even higher fill rates, resolutions, and performance. These include the ability to function as a sync slave, and synchronization signals for both stereo frame parity and render buffer flip, as well as some special features enabled by the architecture of the SAGE Convolve chip. SAGE is not unique in this respect; one can also tile multiple commodity PC solutions together. But with SAGE, one starts with a much more powerful building block with high geometry and fill rates, large texture stores, and that already performs high quality supersampled antialiasing.

11 OTHER FEATURES

SAGE is a complex multi-chip machine. Details of textured rendering, lighting, picking, texture read-back, context switching, etc. were re-implemented for SAGE, often somewhat differently than has been done before. In this short paper, we chose to focus on the major effects that the antialiasing algorithm had on the architecture; this is not to detract from other areas of 3D graphics hardware where the implementers pushed the envelope as well.

12 IMPLEMENTATION STATUS

Complete SAGE prototype hardware is up and running, with OpenGL rendering and full scene antialiasing with arbitrary filters as described in this paper. The board is shown in Figure 11.

13 CONCLUSIONS

A new high end architecture and implementation for 3D graphics rendering, SAGE, has been described. The performance goal of over 80 million lit, textured, antialiased triangles per second has been met. We have also achieved our goal of producing a hardware antialiasing system whose images are numerically and perceptually indistinguishable from images generated by the antialiasing portion of leading software renderers. This is achieved through the use of a hardware double-buffered sample buffer with on-the-fly video-out spatial filtering, capable of implementing non-uniform supersampling with cubic reconstruction filters.

ACKNOWLEDGEMENTS

REFERENCES

AKELEY, K. 1993. RealityEngine Graphics. In *Proceedings of SIGGRAPH 1993*, ACM Press / ACM SIGGRAPH, New York. Kajiya, J., Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 109-116.

AKELEY, K. 2001. Course notes of CS448A, taught fall semester at Stanford University. URL: http://graphics.stanford.edu/courses/cs448a-01-fall/lectures/lecture5/

COOK, R, CARPENTER, L, and CATMULL, E. 1987. The Reyes Image Rendering Architecture. In *Computer Graphics (Proceedings of SIGGRAPH 87), 21 (4)* ACM, 95-102.

DEERING, M., WINNER, S., SCHEDIWY, B., DUFFY, C and HUNT, N. 1988. The Triangle Processor and Normal Vector Shader: A VLSI system for High Performance Graphics. In *Computer Graphics (Proceedings of SIGGRAPH 88), 22 (4)* ACM, 21-30.

DEERING, M., SCHLAPP, S., and LAVELLE, M. 1994. FBRAM: A new Form of Memory Optimized for 3D Graphics. In *Proceedings of SIGGRAPH 1994*, ACM Press / ACM SIGGRAPH, New York. Glassner, A., Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 167-174.

DOMINÉ, S. 2001. OpenGL Pixel Formats and Multisample Antialiasing. URL: http://developer.nvidia.com/docs/IO/1594/ATT/PixelformatsAndMultisample.pdf

EYLES, J., MOLNAR, S., POULTON, J., GREER, T., LASTRA, A., EN-GLAND, N., and WESTOVER, L. 1997. PixelFlow: The Realization. '97 Eurographics/SIGGRAPH Workshop on Graphics Hardware (Los Angeles, CA, Aug 3-4, 1997).

GLASSNER, A. 1995. *Principles of Digital Image Synthesis*. Morgan Kaufmann.

HAEBERLI, P., and AKELEY, K. 1990. The Accumulation Buffer: Hardware Support for High-Quality Rendering. In *Computer Graphics (Proceedings of SIGGRAPH 90), 24 (4)* ACM, 309-318.

MITCHELL, D., and NETRAVALI, A. 1988. Reconstruction Filters in Computer Graphics. In *Computer Graphics (Proceedings of SIGGRAPH 88), 22 (4)* ACM, 221, 228.

MOLNAR, S., COX, M., ELLSWORTH, D., and FUCHS, H. 1994. A Sorting Classification of Parallel Rendering, *IEEE Computer Graphics and Applications*, July 1994, 23-32.

MONTRYM, J., BAUM, D., DIGNAM, D., and MIGDAL, C. 1997. InfiniteReality: A Real-Time Graphics System. In *Proceedings of SIGGRAPH 1997*, ACM Press / ACM SIGGRAPH, New York. Whitted, T., Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 293-302.

STOLL, G., ELDRIDGE, M., PATTERSON, D., WEBB, A., BERMAN, S., LEVY, R., CAYWOOD, C., TAVEIRA, M., HUNT., S., and HANRAHAN, P. 2001. Lightning-2: A High Performance Display Subsystem for PC Clusters. In *Proceedings of SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, New York. E. Fume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 141-148.

TAROLLI, G.1999. Real-Time Cinematic Effects on the PC: The 3Dfx T-Buffer. Hot 3D presentation in Eurographics/SIGGRAPH Workshop on Graphics Hardware 1999, IEEE Press.

TREMBLAY, M., CHAN, J., CHAUDHRY, S., CONIGLIARO, A., TSE, S.S., 2000. The MAJC Architecture; A Synthesis of Parallelism and Scalability. *IEEE Micro Mag*. Nov/Dec 2000, Vol 20, 12-25.

UPSTILL, S. 1990. *The RenderMan Companion*. Addison-Wesley.

WINNER, S., KELLY, M., PEASE, B., RIVARD, B., and YEN, Y. 1997. Hardware Accelerated Rendering Of Antialiasing Using A Modified A-buffer Algorithm. In *Proceedings of SIGGRAPH 1997*, ACM Press / ACM SIGGRAPH, New York. Whitted, T., Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, 307-316.
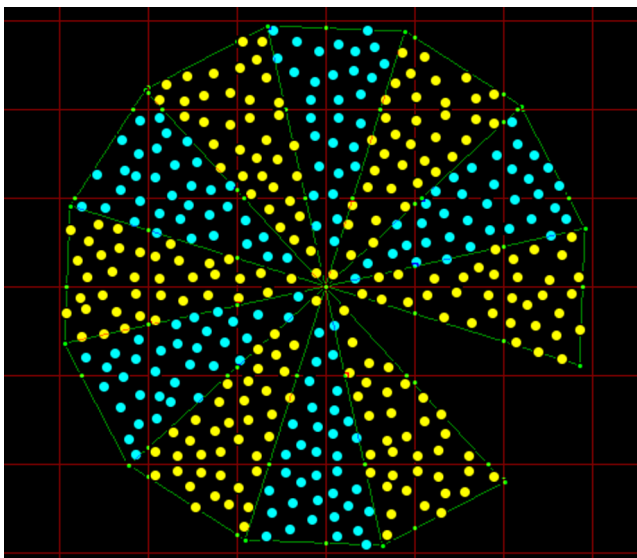
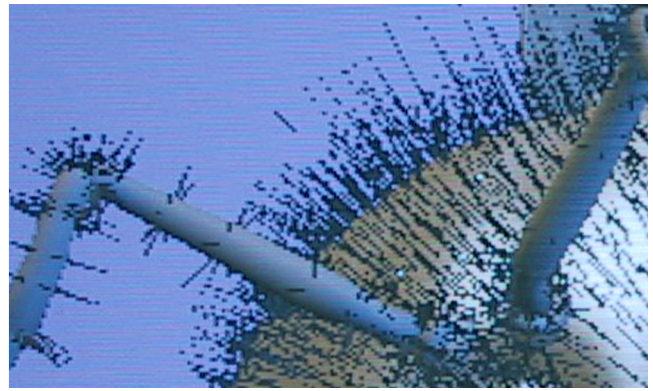**Figure 5: Sample pattern at density 16**
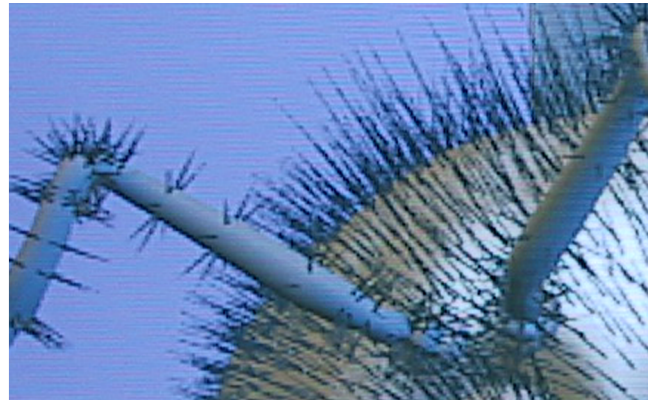

Figure 6: Bee, sample density 1, no filter.

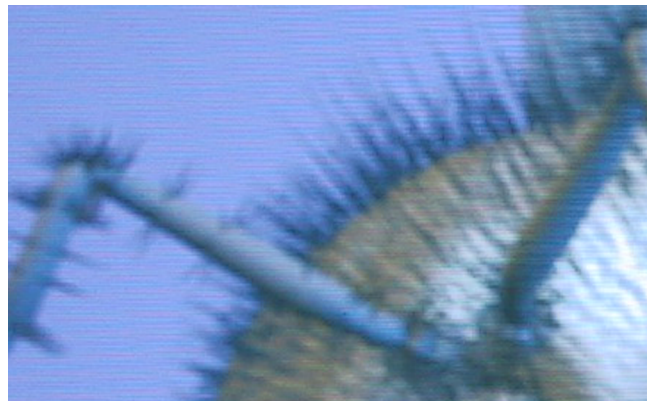**Figure 7: Bee, sample density 8, 1/3 1/3 Mitchell filter.**
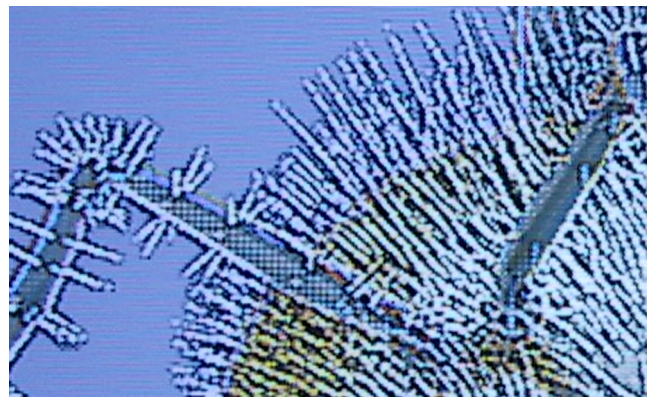

Figure 8: Bee, sample density 8, diameter 4 cylinder filter.

Figure 9: Bee, sample density 8, Laplacian filter.

691

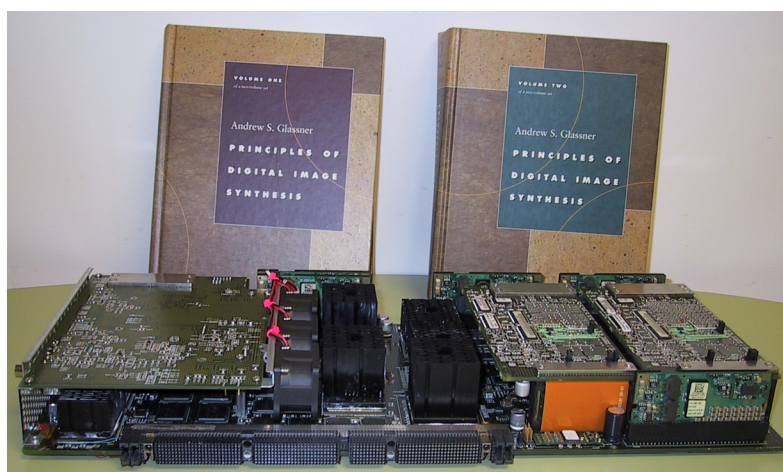**Figure 10: Bee, sample density 8, 1/3 1/3 Mitchell filter.**

Figure 11: Photo of prototype SAGE **board.**