performed on the SRAM which acts as a first level cache, the page buffers then act as a second level cache. Both levels of cache in FBRAM are organised into rectangular blocks of pixels, rather than simple horizontal segments; this allows primitives with small spans to be cached efficiently. Simulated performance figures for the caches are given in **[DeSL94]**.

and vector operations. The i860 is used in a number of current graphics systems including the Silicon Graphics Reality Engine **[Akel93]** and Pixel-Planes 5 purely for its abilities in the geometry section.

6.3.2 FBRAM

FBRAM is an enhanced memory device optimised for 3D graphics rendering **[DeSL94]**. It removes the costly read-arithmetic-write operation needed for Z buffering and alpha blending, and also provides caching to increase frame buffer bandwidth.

FBRAM eliminates the costly read-arithmetic-write operation by incorporating a *pixel ALU*, on the FBRAM memory ICs, which performs Z comparison, alpha blending and any logical operations. Therefore the external view of a Z buffered write consists of a single transaction with the FBRAM. Implementing an internal *pixel ALU* requires that the entire Z value (which in this implementation is 32 bits wide) is stored in a single IC; this is unusual since most memory chips provide only an eight or sixteen bit data bus. The FBRAM has an output, which states whether the Z comparison succeeded, which is connected to an input to the FBRAM IC holding the colour information, so that the colour information is only written if the Z comparison succeeds. FBRAM contains four 8-bit multiplier-adders to enable the alpha blending calculations (on the red, green, blue and alpha channels) to be performed in one operation.

The use of an on-chip ALU enables FBRAM to accept pixels at the rate of 100 million pixels per second; there now exists the problem of accessing the actual storage fast enough. Deering et. al. point out that the limiting factor in frame buffer bandwidth is not the storage technology but is actually related to providing access to it from outside the IC via a narrow 8, 16 or 32 bit bus. Internally to a DRAM chip there is a potential bandwidth of 10 GByte/second at the page buffer which is used to hold the contents of one row of the array before being accessed. FBRAM contains 2Kbit of fast SRAM and four banks of DRAM, each containing a page buffer. All operations by the pixel ALU are

6.2.3 Summary: Existing software

Using conventional operating systems, such as Unix, for virtual reality produces highly unpredictable behaviour. Modified versions of Unix, such as IRIX can help this situation, however, specialised real time operating systems may be more predictable.

Graphics libraries play an important role in the performance of VR systems. Simple libraries such as GL, cause a heavy overhead due to library calls, while leaving the application to perform unusual optimisations to get the best performance out of the hardware. Higher level libraries, such as Performer, provide a much more useful library, with less overheads. The task of optimisation for the hardware now moves into the library and away from the application, allowing natural data structures and algorithms to be used within the application.

The combined use of IRIX REACT™ and Performer provides an efficient solution to the use of coarse grain parallelism in a VR application. Both the rendering process and the application can be distributed over the available processors in a predictable manner. The predictability of IRIX running on multiple processors combined with the automated stress management system in Performer can help produce a near constant frame rate.

6.3 Enabling technology

The Intel i860 and FBRAM are examples of devices which although based on conventional technology are optimised for use in rendering systems.

6.3.1 The Intel i860

The Intel i860 is a general purpose processor **[GrKB89]** which includes hardware to accelerate rendering. It includes hardware for performing Z comparison and easing the job of shading, as well as integer and floating point arithmetic units optimised for matrix can be combined into single processes and thus remove interprocess communication overhead. Effectively the subdivision onto multiple processors extends the graphics rendering pipeline back into the host and dedicates two of the hosts processors to performing database traversal and culling. Since these are general purpose processors, they exhibit extreme flexibility and, in some applications, may be able to provide extra processing power for the application. As in any other pipeline, this extension causes increased latency, however Performer can be configured so that the cull and draw processes are linked via a FIFO allowing their execution to overlap and thus reduce the latency. One latency problem in this (and other retained mode libraries) is that rendering does not start until editing has finished. To minimise this effect, Performer allows the reading of input sensors (in particular the head position sensor) immediately before signalling the start of a new frame, and hence immediately before rendering.

6.2.2.6 RenderWare

RenderWare, by Criterion Software Ltd., is a fundamentally different type of library from those described above since it performs *all* rendering using the host processor. The RenderWare library is available on a number of different platforms including IBM PC compatible systems, Silicon Graphics and Sun workstations. Criterion claims to be able to render over 100 000 Gouraud shaded triangles per second with hidden surface removal on a top of the range personal computer, and over 50 000 Perspective correct textured triangles per second on the same computer **[Crit94]**. Very little information has been made available on the rendering method used by RenderWare. Although the performance of this type of rendering software is much lower than the performance of the rendering hardware found in workstations, it is comparable to the speeds provided by low cost rendering accelerators, such as GLiNT™.

- Intersection testing

  Performer provides a separate model traversal stage for intersection testing. The application can use this to test whether (and where) a number of line segments intersect the model. The line segments are placed around important objects to provide collision detection. To increase the efficiency of the intersection testing process, the model may have a simpler representation of an object which is used during intersection testing.

As stated above, one of the most unusual features of Performer is its support for parallel processing. This feature is important in light of the heavy load placed on the host system by the rendering, culling phases and the actual application. Silicon Graphics offer a number of configurations of graphics workstation with multiple processors, most containing a small number of processors (the largest has 24). Performer is designed to take advantage of this small number of processors by using coarse grain parallelism.

The rendering pipeline is split into application, cull, and draw stages; another stage, intersection, is provided which is not part of the rendering pipeline. The application updates the model as needed and then signals to Performer that a new frame should be drawn. At this time, Performer traverses the hierarchy and performs some maintenance on it and then starts the cull phase. The cull phase traverses the hierarchy, performs culling, and gathers primitives for rendering. These primitives are sorted to ensure primitives with the same graphics modes are rendered at the same time reducing the number of mode changes required; Rohlf and Helmann state in **[RoHe94]** that this sorting can increase rendering throughput by over 50%. The output of the cull stage is a display list which is sent to the draw stage. The draw stage traverses this list, calling GL to perform the rendering.

In a multiprocessor environment with three processors, the application, cull and draw stages can each execute on a separate processor; in a system with less processors, stages number of processors in multiprocessor systems. Performer provides many optimisations aimed directly at virtual reality applications:

- Optimisations for Silicon Graphics hardware and GL libraries

  Performer sorts primitives into uniform groups, having the same characteristics (e.g. number of sides, attribute bindings etc.). Routines specifically optimised for that particular set of characteristics are then used to send the primitives to GL. Each one of these routines is simple and optimised for the Silicon Graphics hardware and GL.

- Level-of-detail switching

  Performer can choose from a number of models of an object to use during rendering based on the distance to the eye, the field of view and the graphics stress level; which measures the amount of time needed to render the scene. Performer uses morphing to smoothly blend between different representations.

- View frustum culling

  Performer uses a mixture of bounding spheres and bounding boxes to cull the hierarchy against the view frustum.

- Texturing for billboards to replace geometry (section **5.4.1**)

  Billboards are used to represent simple objects like trees in a more efficient manner.

- Transformation elimination

  Traversing the model hierarchy and constantly updating the current transformation matrix can be expensive; where sections of models are static, Performer precalculates some of the transformations (at the cost of a larger data structure).

encloses a volume completely and thus back-face culling can be used for rendering.

- Optimised for the hardware

  The Inventor routines which call GL to perform the rendering, are optimised using Silicon Graphics' knowledge of their graphics libraries and hardware. In this way many of the efficiency losses associated with the use of libraries are reduced. Since Inventor is built on top of GL, the host still has to execute many GL library calls to perform the rendering of each frame and is thus still bound by that loss of efficiency.

- More functionality

  The Inventor hierarchy can be used for picking as well as rendering thus removing the need to have a copy of the hierarchy for picking. The application can also store extra information in the hierarchy and manipulate it for its own purposes, removing the need for the application to have a private copy. The higher level of functionality (and more useful primitives) allow applications to be written in Inventor with a lot less effort than GL.

Inventor still generates a high computational load for the host processor, both for tessellation and communication with GL.

6.2.2.5 Silicon Graphics Performer

Performer is a graphics library specifically designed for "*visual simulation, virtual reality, and other real-time 3D graphics applications*", which like Inventor, is built on top of Silicon Graphics GL and maintains a hierarchical scene description **[RoHe94]**. One of its primary aims is to provide a high and constant redraw rate, which is also one of the primary requirements of a VR rendering system as outlined in **Chapter 4** . One of the unusual features of Performer is that it attempts to distribute the host load across a implements a retained mode style of execution. Inventor also provides support for common operations on the model such as *picking* (which allows the user to select an object), manipulation of objects by the user, and the storage and retrieval of the model to disc. The application edits the model using Inventor library calls. Inventor has a number of advantages over libraries similar to GL or PHIGS:

- More useful primitives

  While the primitive objects in GL and PHIGS are simple polygons (and in GL, NURBS) Inventor provides cones, cubes, cylinders, text, collections of polygons and NURBS as its primitives; each of these primitives is represented by one C++ class. Creating a cone under inventor consists of just one line of code (with one library call) **[SGI92c]**:

  ```
  root->addChild(new SoCone);
  ```

  In GL, an application would have had to perform the tessellation of the cone itself[1], Inventor provides tessellation facilities for the application and allows the application to set the required level of tessellation; or Inventor can select the level of tessellation based on the size of the object in either screen space or object space.

- Extensibility

  Inventor is extensible, allowing the programmer to create new classes of object specially optimised for his application.

- Hinting

  The Inventor classes which represent collections of polygons allow the application to provide hints to the rendering software about the collection; one example of this is that the application can state that a collection of polygons

1. GL does provide tessellation of NURBS.

feature of a graphics language since the application is forced to perform a large amount of preprocessing before being able to render the scene.

All model traversal and model editing must also be performed by the host; this removes all knowledge of the model structure from the rendering system, meaning that it cannot use knowledge about the hierarchy to perform culling, and that all such operations must be performed by the host.

6.2.2.2 OpenGL

OpenGL is the successor to Silicon Graphics GL and is now becoming a standard graphics library available on many platforms. It has the same basic structure as GL except that while GL provided support for querying input devices, OpenGL expects that to be done by a separate piece of software.

6.2.2.3 PHIGS

PHIGS is a graphics library which uses the retained mode model of the graphics system. The application program describes the model hierarchy to the graphics library which, in conjunction with the rendering hardware, performs all model traversal and renders the scene; it is only necessary to describe the model to the graphics library once. The disadvantage of the retained mode scheme is that editing the model is more complex than in immediate mode systems. In GL updating the position of an object requires the application to change one of its own data structures; however in PHIGS, updating a position or altering a model requires a call to PHIGS to cause the editing of the copy of the model stored internally to PHIGS **[HuHe91]**. This editing step can be expensive, and may be impractical on systems which change a large amount of the model in each frame.

6.2.2.4 Silicon Graphics Inventor

The Inventor library is built on top of the low level rendering library (typically GL). Inventor maintains a hierarchical model description for the application, and therefore tines to setup the lighting and material type, and then for each triangle would perform the following[1]:

```
bgnpolygon();
n3f(trianglenormal1);
v3f(trianglevertex1);
n3f(trianglenormal2);
v3f(trianglevertex2);
n3f(trianglenormal3);
v3f(trianglevertex3);
endpolygon();
```

This process has required at least eight library calls (although optimisations are available through the use of triangle meshes). As was shown above the number of library calls made in typical programs can be so high that the time taken to process the calls can consume much of the host processors available time per frame. The time of 7.25 ms given above did not take into account the processing in each subroutine, the time taken to pass subroutine parameters or the processing by the application to traverse its data structures; once all these have been taken into account it is possible to see that the host processor can become the limiting factor in the rendering speed of systems using GL

One feature of GL that causes a large number of procedure calls (and is time consuming on Silicon Graphics hardware) is changing global state, such as the colour. For this reason, many GL applications perform non-intuitive reordering of primitives. One example of this is sorting polygons on their colour so that the number of times the colour is changed is reduced **[Prat93]**. The author regards this type of problem as an undesirable

1. Where trianglevertex1..3 are arrays containing three elements corresponding to the x, y, and z coordinates of the vertices.

	CADCentre test model	CADCentre plant model	Division house	Division bedroom	Jack	Radiosity scene
Triangles at rasterisation stage	4579	22952	2455	506	9439	4612
Number of vertices	18948	92195	16340	4585	18944	17024
Total number of GL calls	41214	192094	54165	14094	47262	50125

Table 6.1: Typical applications use of the GL graphics library in one frame

that the compiler could not remove the subroutine calls during optimisation. With 200 calls in each loop iteration, the overhead of the loop itself should be small. The result of this timing shows that one subroutine call takes approximately 145ns. Many of the applications shown in table 6.1 use around 50 000 subroutine calls just for graphics library calls; this is equivalent to 7.25ms of host processor time taken on the entry and return from the library calls alone. The severity of this problem depends on the design of the graphics library, which influences the number of calls which must be made; this issue is discussed in detail below.

The following section describes a number of different graphics libraries; most of the graphics libraries described are produced by Silicon Graphics since these were available on the Silicon Graphics Crimson in the Advanced Interfaces Group.

6.2.2.1 The Silicon Graphics 'GL' graphics library

The Silicon Graphics GL library is the standard method of communicating with all Silicon Graphics rendering systems **[SGI90]**. Although it provides some support for the creation of display lists it is primarily designed for immediate mode systems and thus requires the host to describe the scene to the rendering system in every frame. To plot a lighted triangle with separate vertex normals the application must call a number of rousible. A program written using a graphics library will not necessarily execute efficiently on all platforms.

- Lack of application optimisation

  The point brought out above is mainly concerned with the application's loss of knowledge about the hardware. Another similar problem is that the rendering system loses knowledge about the application. Where the programmer has complete control, it may be possible for him to provide 'hints' to the rendering system; for example he could turn off depth comparisons while drawing a background.

- Hardware limitation

  Many graphics libraries are written for a particular type of rendering system; for example many libraries can only handle polygons rendered using Z buffering. It may be difficult to implement a graphics library of this type on a system using one of the other rendering algorithms (such as the scanning span-line method).

- The performance cost of the graphics library

  When using a graphics library, the programmer has to perform subroutine calls to the graphics library even for very simple operations, which could have been coded explicitly in time-critical pieces of code. Table 6.1 shows a table of the number of graphics calls[1] made by the applications analysed in **Chapter 5** .

  The author has timed subroutine entry on a Silicon Graphics Indy by timing the time taken to execute 500 000 iterations of a loop containing 200 calls to an empty subroutine. The subroutine was placed in a separate source file to ensure

1. The total number of GL calls made was counted using the Silicon Graphics *GLprof* tool. This converts the GL calls into OpenGL calls and stores these, with extra profiling information, in a trace file. This figure is a count of the number of calls in the trace file with the extra profiling information removed; this figure may be slightly different from the actual number of calls made, due to the GL to OpenGL conversion process.

of multiple tasks is tuned to real time applications. Compared with working with a bare system without an operating system, VxWorks offers a full set of development tools, and also provides support for various processor architectures and devices, thus offering portability between systems. The main advantage of an operating system like VxWorks to a Virtual Reality system is that the context switches, due to time sliced multitasking and also due to interrupts, are more predictable, more controllable and faster than a conventional Unix system **[Ande94]**.

6.2.2 The graphics library

It is now rare for application programs to access graphics rendering hardware directly. Application programs usually communicate with the rendering hardware via a standard *graphics library* instead. This has a number of advantages:

- Hardware independence

  A program can be written to work with many different types of rendering hardware; only the library has to be replaced when the program is executed with a different rendering system.

- Reduced application development time

  Programming graphics hardware is a complex task requiring a large amount of work, involving detailed information on the design of the hardware. Graphics libraries hide the detail of programming the graphics hardware from the application programmer, reducing development times.

Graphics libraries do however have a number of disadvantages:

- Lack of hardware optimisation

  When a programmer writes an application which accesses hardware directly, he will optimise the access routines for his application and will write the application in such a way that it uses the hardware in the most efficient manner posshows that the operating system needs more controllable memory management facilities for real time applications.

6.2.1.3 Interrupt response time

Interrupt handling in a multitasking operating system, such as Unix, is a complex task. One important situation in a VR application, is where a process is waiting for input from a position sensor; when data arrives an interrupt should occur and the operating system should start executing the process again. Most versions of Unix do not specify any limits on the time from the interrupt occurring until the process starts executing. In the REACT™ system in IRIX this time is specified as being a maximum of 200 µs in a correctly configured system, this contrasts with a few milliseconds on some non-real time versions of Unix. Associated with interrupt response is the load on the CPU caused by interrupts. In a typical VR system it would not be unusual to have a number of devices providing interrupts (for example: position sensors, speech recognition, graphics systems) and this can place a heavy load on the CPU. REACT™ allows interrupts to be directed to particular processors in a multiprocessor system, so that the time critical processes are only affected by hardware which they are servicing.

6.2.1.4 Other solutions to the problem

In systems which are not based on workstations it is often possible to remove the operating system altogether and thus give the time critical routines complete freedom in their resource usage. The penalty of removing the operating system altogether is a much higher development cost, due to the need to write everything from scratch, and a lack of portability.

Although versions of Unix are now emerging with some support for real time applications, a number of other operating systems have been written specifically for the purpose of use in real time systems; VxWorks produced by Wind River Systems is one such operating system **[WRS94]**. It is Unix-like in its programming interface, but its handling

The REACT™ extensions to IRIX (described in **[SGI94]**) solve this problem in a number of ways. The first method allows a process to be assigned a *fixed priority* which is independent of the amount of CPU time previously consumed. Thus time critical processes can be given a higher priority than any of the other processes on the system.

The second method, only usable on a multiprocessor system, executes the time critical processes on a separate CPU. REACT™ also allows the scheduler to be disabled on CPUs running time critical software, giving time critical processes control of their CPU allocation.

6.2.1.2 Memory management

Unix systems support paged virtual memory. Determining which pages get paged out to disc (and when this paging is likely to happen) is difficult to predict. In most real time applications the time taken to swap to disc is unacceptable. One solution is to ensure that enough physical memory is available for the application; even with this restriction, it is still possible that background processes will occasionally need to be called which will potentially cause pages to be paged out. The solution given in REACT™ is to allow processes to lock all their code and data into physical memory, and thus effectively remove themselves from the virtual memory system.

The NPSnet system described in **[Prat93]** uses very large terrain databases (in the region of 64MByte in size) where only a small portion is visible at a time; thus locking the entire process in memory is too restrictive. Most versions of Unix only attempt to fetch pages from disc when the page is accessed. This meant that when rendering a scene, a section of the database would be reached which had not recently been accessed and the rendering would have to stop while the terrain was paged in. In NPSnet they implemented a form of *predictive paging* which solved this problem by predicting when a new section of terrain would be required and using another processor to request that page, thus forcing the operating system to have the page ready in advance. This example

Infrared tracker **[WABG92]**) become common, many of the benefits are lost, since this system is only capable of removing latency due to rotational changes.

6.2 Software considerations

As in all applications, the software environment plays a large part in the performance and usability of VR systems. This section considers a number of the more important areas of the software which must be carefully considered in a VR system.

6.2.1 The Operating system

Many of the graphics workstations used for virtual reality use a derivative of the Unix operating system; this operating system was never intended to control real time applications and can impose a large and highly unpredictable overhead in terms of both CPU and memory availability.

6.2.1.1 Multitasking

Unix incorporates conventional time sliced multitasking. After each time slice the Unix scheduler must choose a process to execute in the next time slot. This choice is based on a *priority value* that is partially chosen by the user but which also reflects how much processor time the task has used previously. In this way, a low priority process may use the CPU occasionally even while a very high priority process still needs it.

In the IRIX version of Unix, used on the Silicon Graphics range of workstations, the default time slice is 30 ms **[SGI94]**. Even on a lightly loaded workstation it is not unusual for one of the system processes to need a time slice occasionally - during this time the processes of the VR application will be unable to update the display. **[AzBi94]** state that "*pauses of 60-200ms are common occurrences on our Sun4 host.*" under one version of Unix.

of which can operate at very high performance levels. PixelFlow in particular is highly configurable allowing a different number of shader and renderer cards to be used for different applications. Examining the geometry stages of the rendering system shows a slightly different spread of flexibility among systems. The VGX has a highly inflexible system based on pipeline stages each performing one particular task. The Pixel Planes 5 and PixelFlow system use Intel i860 processors to perform all geometry operations, which are general purpose processors optimised for graphics use. Being fully programmable, they offer a high level of flexibility, while offering reasonable efficiency for graphics operations. The Leo system however offers the most interesting geometry system. Using custom LeoDraw chips, it provides a fully programmable system specifically optimised for graphics; this flexibility is especially evident in Leo's ability to perform dynamic NURBS tessellation.

- Image quality and realism

  The provision of dynamically tessellated NURBS by Leo promises to offer high quality curved surfaces with a lower overhead (and thus higher image complexity). Other systems, such as the VGXT, provide fully perspective corrected, MIPmapped texturing, which provides highly realistic images. PixelFlow offers antialiasing and high quality shading as well as high scene complexities.

The Regan and Pose address recalculation pipeline appears to do well on many of the criteria given above; it does however suffer from two disadvantages. The first is that it is complex and expensive, due to the large amount of frame buffer memory (which must be built out of static RAM), and the complexity of logic required; this problem will diminish as the price of RAM falls and is not much worse than some of the rendering systems with massively interleaved frame buffers. The second problem is that if head tracking systems which allow significant translational movement (such as the UNC

- Dynamic Scene Limitations

  None of the systems use any type of pre-computation to increase the rendering speed and thus do not enforce any limitations on how often the scene description can be modified. Pixel-Planes 5 was operated as a retained mode system using a Sun SparcStation as a host processor; early versions suffered from a limited host to rendering system bandwidth which reduced the amount of editing that could be performed **[TRCB93][Ells94]**.

- Freedom of viewing parameters and model geometry

  None of the systems provides any unusual limitations on the viewing parameters or the model geometry.

- Flexibility

  The flexibility of the rendering systems presented here varies greatly. The rendering and geometry stages will be considered separately; first the rasterisation stage. The Vialls renderer can only perform simple Gouraud shading (albeit with transparency) but would require hardware modification to perform any other types of shading. Similarly the LeoDraw stage of Leo can only perform Gouraud shading. Both the Image chip, and the Pöpsel and Tikwinski Hardware shader, provide extreme flexibility in the shading stage by allowing a general purpose processor to implement complex shading procedures; although flexible, in general this approach can not generate shaded pixels fast enough. The VGX provides flexible Image engines capable of performing many types of texturing and shading; the provision of the stencil planes enhances the flexibility greatly, allowing Constructive Solid Geometry (**Appendix A**) to be implemented, which is not possible on many of the other systems. The most flexible systems of all at the rasterisation stage are definitely the Pixel Planes and PixelFlow systems; the programmability of each pixel processor allows a wide variety of different algorithms to be performed on the same hardware; all systems with bad latency characteristics, caused by non-overlapping pipeline stages, can reduce their latency by allocating less time for each stage; this leads to low efficiency and poor rendering capabilities - this is the approach taken by the low latency configurations of Pixel-Planes and PixelFlow. PROOF must transform all primitives before rasterisation can begin; however, since the latency in the object pipeline is small, architectures similar to PROOF can provide an acceptably low latency. Systems similar to PROOF may be able to provide a guaranteed frame rate allowing the latency due to double buffering to be removed.

The Regan and Pose address recalculation pipeline offers perhaps the best solution to the latency problem in the HMD based systems.

- Consistency with the physical world

  The only system presented here to account for real world consistency issues is the Just-In-Time pixel system described in section **6.1.7.5**.

- Multiple image generation

  Neither of the workstation graphics accelerators, VGX and Leo, provide direct support for separate image generation; however both provide support for frame sequential stereo using LCD shutter glasses. As stated above multiple image outputs can be added to the VGX system; although this is both expensive and adds an appreciable latency.

- Correction for distortion in the display device

  The only system to address this problem is the Regan and Pose address recalculation pipeline; not only can distortion correction be implemented in the Regan and Pose system, it fits in cleanly and with no performance penalty.

information, may find their performance, even in the low end applications they are aimed at, is limited by available host performance.

Those systems which can perform complex shading without significant performance degradation also have a good chance of producing a high frame rate, due to the ability to reduce the required scene complexity; the use of deferred shading in systems like PixelFlow helps perform complex shading with a low penalty.

While the description above has concentrated on general VR systems, a system designed for a particular application, such as the one by Vialls, does not necessarily require a large number of primitives to be rendered. The system by Vialls produces a high redraw rate for the given application, but does not render a particularly large number of polygons.

The priority rendering system by Regan and Pose offers a different perspective to this problem; instead of offering a faster way of rendering, it reduces the frequency with which objects need to be redrawn.

- A constant redraw rate

  The redraw rate of Pixel-Planes 4 is independent of the size, orientation and position of the objects in the scene. Some systems, such as the Vialls renderer and PROOF have redraw rates which are largely independent of primitive size, but both suffer from worst case behaviours which can increase the rendering time; in the case of the Vialls system this could occur when a large number of primitives intersect the same scan-line and cause the hidden surface and shading tasks to be particularly expensive.

- Low latency

  Most of the conventional systems such as the VGX and Leo offer reasonably low latencies, those that require primitive sorting (Vialls, Pixel-Planes 5 in its standard configuration, and PixelFlow) have increased latencies. Any of the

The low cost and performance of GLiNT™ and other similar accelerators such as the Austek Microsystems A1060 **[Kinn94]** will enable high quality virtual reality applications to be implemented for a much lower cost and thus be made available to a much larger range of users for a wider range of application areas.

6.1.10 Summary: Existing Hardware

It is difficult to provide a meaningful comparison of all the systems described above since they each address different sections of the pipeline. Only the Silicon Graphics VGX, Leo, and the Pixel-Planes and PixelFlow systems constitute an entire rendering system.

To be able to compare the systems it is necessary to compare their performance with the requirements given in **Chapter 4** :

- A high redraw rate

  Pixel-Planes 5 and PixelFlow offer the highest rendering rate (in primitives per second) of the systems described in this chapter and can (other considerations aside) produce the highest redraw rate for complex scenes. Both of these systems produce high rendering rates through massive parallelism, both through the use of the logic enhanced memory chips and through the use of image space subdivision providing parallelism. PixelFlow and PROOF both offers the possibility of being almost infinitely scalable due to image composition.

  The two workstation products, the Silicon Graphics VGX and the Sun ZX, have different approaches to host interaction. Where the VGX requires that the host feeds it individual vertices and commands, the ZX will fetch data from memory with little host intervention. In real applications where the host is highly utilised, the approach used in the ZX is a clear benefit. GLiNT, and other similar systems which expect the host to provide vast amounts of setup

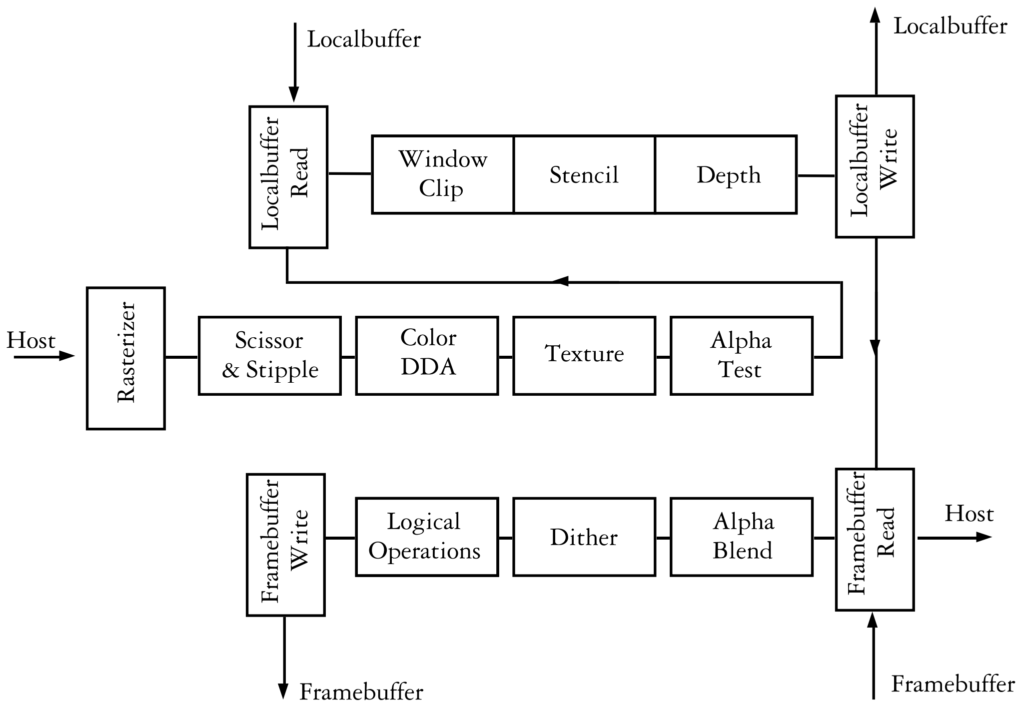

Figure 6.4: A block diagram of the GLiNT™ 3D Graphics Processor

tle detail about the rendering algorithm used by GLiNT™ was made available by 3DLabs, other than the block diagram and information in **[DuPo93]**. From this information it is possible to conclude that GLiNT™ provides a conventional Z buffer (connected to the 'local buffer' port) with Gouraud shading, alpha blending and texturing. Texture mapping is performed with perspective correction and the MIP map algorithm is supported.

GLiNT™ is unusual in that the rasterisation, shading, texturing and hidden surface removal is performed in one integrated circuit[1]; this high level of integration allows it to be produced at low cost (approximately £100 in large quantities **[Harr94]**). Although the performance of GLiNT™ is not as high as the rendering engines in workstations such as the Silicon Graphics VGXT, its quoted performance of 300 000 Gouraud shaded triangles/second is more than adequate for many applications.

1. With the addition of memory for the frame buffer, depth buffer and texture maps.

problem if the viewer is moving fast, but in most systems this is most likely due to the user travelling in a simulated vehicle whose position can be predicted with high degrees of accuracy.

- Correction for display device distortion is included, which also helps maintain consistency with the real world.
- Multiple images are generated for stereo viewing by duplication of the address recalculation pipeline. This would imply that only one set of rendering hardware is required for both eyes.
- No restriction is placed on head movement or on the scene being rendered; although it is preferable to perform as little translation as possible.

The disadvantages of the Regan and Pose system are in complexity and cost. The address recalculation pipeline must operate at pixel rates and consists of many large look up tables, multiplication stages and other logic intensive stages. These stages are complex and expensive; in addition the non-sequential access of display memory rules out the use of conventional VRAM and instead much more expensive static RAM must be used. The frame buffer must also be larger, due both to the need to store an entire cube around the user's head and for the need to store multiple versions of this frame buffer to accommodate objects being updated at different rates.

6.1.9 3DLabs 'GLiNT'

A number of semiconductor manufacturers are designing rendering accelerator components to be used in low cost expansions for personal computers. An example of this type of component is the GLiNT™ 3D Graphics Processor from 3D Labs. A block diagram of the GLiNT™ is shown in Figure 6.4.

Transformation, lighting and setup calculations are performed on the host processor which then passes the results to GLiNT™ for the rest of the rendering process. Very lit-

The other feature of the Regan and Pose design is *Priority Rendering*; this feature is really only applicable to a system with the address recalculation pipeline. The image of objects on the cube in the address recalculation pipeline, only change due to the object being modified or translation of the object relative to the viewer. Furthermore, except for objects which are close to the viewer, movement relative to the viewer is likely to produce a slow movement of the image on the cube; thus most objects do not need rendering in every frame. Regan and Pose take advantage of this effect by holding a number of separate frame buffers and using image composition to combine the contents of the frame buffers at the end of the recalculation pipeline. Each frame buffer holds an image representing all six sides of the display cube, however objects only appear in one of the frame buffers. The *Priority Rendering* method that Regan and Pose describe in **[RePo94]** involves updating these frame buffers at different rates. Objects which move fast relative to the user are placed in a frame buffer which is updated at sixty times a second (for example). Slower moving objects are placed into frame buffers which are updated at thirty, fifteen, seven and a half, and three and three-quarter times per second. The frame buffer which an object is placed in is determined by an approximation of its speed of movement relative to the user. An example given by Regan and Pose in **[RePo94]**, shows that nearly 40% of objects may be placed in a frame buffer being refreshed at only seven and a half times a second while less than 10% of the objects require refresh at the full sixty times a second.

This system, by Regan and Pose satisfies (or helps to satisfy) a number of the requirements of VR systems set out in **Chapter 4** :

- The priority rendering system helps to maintain a high redraw rate for those objects which need it.
- Latency relative to head orientation is very small due to the address recalculation pipeline. Other forms of latency are minimised due to the high redraw rate on objects which are especially important. Latency due to translation can still be a

determined only by the viewer's head position; head orientation is taken into account during the display scan-out stage. In this scheme a new head orientation needs to propagate through the address recalculation pipeline, which operates at pixel rates; this propagation takes less than a microsecond. Regan and Pose note that the most noticeable type of latency in VR systems is from head rotation and not from translation, since translation is often performed 'artificially' by the user pressing a button or performing a gesture.

The most unusual section of the Regan and Pose system is the *Address Recalculation Pipeline*. Operating at pixel rates, the Address Recalculation Pipeline selects for each pixel on the display a pixel from the display memory. The pixel selection process uses the user's head orientation data, and information on the optics in the head mounted display, to calculate the position on the face of the rendered cube which corresponds to the current pixel on the display device. The pipeline takes as input a stream of pixel locations on the display device and outputs the RGB value of the pixel.

The first stage of the pipeline is the conversion of the pixel's screen location into a vector pointing in the direction of the pixel relative to the user's head. This process is carried out by a large (3 MByte) look up table. The values in this table can be distorted to account for the effects of wide angle viewing lenses at no extra cost.

The vectors from this stage are multiplied by the user's head orientation matrix to obtain a vector pointing to the position of the pixel in world space. These vectors must now be converted into an address in display memory. The method used to perform this is described in **[RePo93]**. Four adjacent pixels are then accessed from display memory and then filtered to provide the eventual RGB value.

The latency of the system to head rotations is thus one frame time plus the time taken for the head position measurement[1].

1. In this case latency is being measured from head movement to the *end* of scan-out time.

to one whole frame) to render 4000 triangles and the bandwidth of the composition network over the available time is high enough to allow 2048 x 1024 antialiased pixels to be rendered.

- Three frame times, 48ms

  In this configuration, the system operates in a similar way to a conventional pipeline. In the first frame transformations take place, then in the second frame rasterisation, composition, shading and transfer to the frame buffer. The third frame is taken up with scan-out.

Since PixelFlow relies on bin-sorting, the transformation and rendering stages do not overlap; low latency is only achieved by reducing the time available to each stage. In **[EnLL94]** Anselmo Lastra suggests that a certain amount of overlap could be achieved by splitting the transformation and geometry stage into separate transform and setup stages. The processor would perform transformations first and sort primitives into bins. The setup calculations would then be performed for the first area that was to be rasterised. As that area is being rasterised, the setup calculations for the rest of the areas are performed in parallel with the rasterisation. This increases the utilisation of the hardware and increases the available performance.

6.1.8 The Regan and Pose address recalculation pipeline

In **[RePo93]** and **[RePo94]**, Regan and Pose describe one of the few graphics architectures designed specifically for virtual reality applications. Its main aim is the reduction of latency in HMD based systems, but it also includes a means of reducing the amount of rendering required and a method of compensating for the distortion caused by wide angle optics used in HMDs.

In their architecture, a scene is rendered onto the six faces of a cube notionally surrounding the viewer's head. The position and orientation of this cube in the virtual world is

- Scalable

  In **[MoEP92]** simulation results are shown for systems with up to 128 renderer cards. These simulations show near linear improvements in performance (in terms of primitives/second) for suitably complex data sets.

- Powerful

  The simulation results show performances nearing 100 million triangles/second on complex data sets when configured with large numbers of renderers.

6.1.7.8 Latency in PixelFlow[1]

Three different configurations of PixelFlow are considered by England in **[EnLL94]**, each capable of rendering with a given latency[2]:

- One frame time, 16ms

  In this configuration the entire rendering process is performed during the retrace time of the display (1.6ms). This time is split into 0.8ms for transformation and display list traversal and 0.8ms for rasterisation, shading, composition etc. The specifications of the rasterisation system and transformation engine state that each renderer can render 400 triangles in this time. The compositor network bandwidth limits the display size to 512 x 256 pixels (with 5 sample anti-aliasing). With enough rendering engines, it would in theory be possible to display any complexity of image in this time.

- Two frame times, 32ms

  All the processing is now performed in one frame, rather than the refresh time for the frame. Each renderer now has enough time (16ms which is equivalent

1. The figures in this section are based on rough estimates provided by members of UNC in a personal communication **[EnLL94]**.

2. In this context latency is measured from the beginning of the transform stage to the END of the frame scan-out time.

The compositor port is 256 bits wide, and operates bit serially on the same bit of 256 separate pixels at once. Composition operations can occur in parallel with rasterisation and thus one area can be rasterised while another is composited.

In immediate mode, the host broadcasts primitives to the various renderers. The geometry processor in each renderer transforms all the primitives assigned to that renderer and places them in bins based on the position of the primitive on the screen (in much the same way as Pixel-Planes 5). Each renderer then starts rendering the primitives out of one of the bins. When all renderers have finished rendering a particular area of the screen, the renderers simultaneously transmit the same area along the pipeline through the compositor ports. At the end of the pipeline of renderers, a composited version of that 128x128 pixel area of the display is presented to the shader. Each pixel processor has enough memory to store a number of areas of the display; thus if one renderer finishes before another, it starts rendering the next area. The system supports supersampled antialiasing (see section **A.8.1.2**) and thus each area is rendered a number of times. When all the samples for a particular area have arrived, the shader performs appropriate texture mapping, shading and supersampling and then passes the result to the frame buffer.

The PixelFlow system has many advantages in common with the Pixel-Planes 5 system; in particular:

- Highly flexible

  The system is configurable with a varying number of renderers and shaders to meet the required performance. Each renderer and shader is, in itself, highly programmable to perform many different algorithms. Rendering of spheres, and volume data is also supported.

6.1.7.7 PixelFlow

PixelFlow is another novel architecture from UNC. Its unusual features include the use of image composition, deferred shading, and the Pixel-Planes enhanced memory system.

The PixelFlow system consists of a pipeline containing renderer boards, shader boards and frame buffer boards. Each stage is connected to a high power host computer, which controls the system and provides primitives (in immediate mode) or sends editing commands (in retained mode).

All three board types consist of an array of 64 PixelFlow enhanced memory chips (forming a 128 x 128 array of pixels), a microprocessor, a section of VRAM (used as a FIFO buffer), and an *Image Generation Controller* ASIC which controls synchronisation of the logic enhanced memory chips (EMCs) and provides data conversion facilities between the EMCs and the microprocessor. In addition, the frame buffer board contains extra memory for the actual frame buffer storage, and DACs to provide video output. The shader also contains extra memory for storing textures.

The enhanced memory chips in PixelFlow have a number of enhancements over the Pixel-Planes 5 version[1]:

- 2048 bits per pixel and a pair of 256 bit communication buffers.
- A compositor port for image composition and transfer of data between pipeline stages.
- A local port for access to the frame buffer or texture memory.
- A number of speed enhancements including a higher clock speed, eight bit processors rather than bit serial, and fast multiply hardware for complex shading.

1. This list is derived from a list in **[MoEP92]**.

described by Mine and Bishop in **[MiBi93]**. They implemented Just-In-Time pixels on Pixel-Planes 5. In this configuration the graphics processors are again used in a Master/slave configuration, but the graphics processors perform the actual rendering without the use of renderer boards. The master graphics processor is responsible for distribution of tracking information to the slave processors and the synchronisation of the slave processors rendering. In a system with ‘n’ slave processors each slave processor calculates every nth scan-line. This system employs a technique called *beam-racing* where the slave processors perform the rendering of a scan-line as close as possible to the time when the scan-line will be displayed. This allows the latest tracking information to be used in the rendering of the scan-line. Unfortunately no details of the scene complexity which this implementation of Just-In-Time Pixels can cope with is given in **[MiBi93]**.

6.1.7.6 Summary of Pixel-Planes 5

Unlike most of the rendering systems described above, Pixel-Planes 5 offers a very high level of flexibility in its architecture. This can be seen by the very different configurations in which Pixel-Planes 5 has been used. The logic enhanced memory architecture allows Pixel-Planes 5 to be used for CSG based models and volume rendering as well as complex polygonal models **[FPEG89]**. Its ability to be configured with varying numbers of graphics processors and renderers, and the fact that the graphics processors are general purpose microprocessors, increases its flexibility even further. In general, Pixel-Planes 5 is also less affected by variation in the size and orientation of primitives (although not to clustering of large numbers of primitives) than conventional rendering systems. All these attributes make Pixel-Planes 5 highly suitable for use in virtual reality systems. Its one flaw, in its standard configuration, is the need to sort primitives based on screen position, thus increasing the system latency.

A modified version of Pixel-Planes 5, with added support for texturing, is available commercially from Division Ltd. **[Divi94]**.

SLATS continually renders frames, using the latest viewing position received from the host.[1] The TRANSFORM stage of SLATS is different from the standard configuration for a number of reasons:

- There is only one GP

  This removes the communication overhead between GPs and also removes contention for renderers.

- No bin sorting

  Since each renderer stores all its image regions locally it is no longer necessary to send all the primitives for one region before the next region can be processed by the renderer. Instead, primitives are sent to renderers as and when they have been transformed. In the prototype system all primitives are sent to all regions, making the rendering time and latency independent of primitive size, orientation and distribution.

This technique, while providing very low latency, allows only 140 triangles (per eye/per 60Hz field) to be rendered; this is because each renderer is passed every triangle and each renderer is responsible for four regions. In addition the necessity to store four regions per renderer reduces the amount of local memory available per pixel in each region; this reduction in memory stops the system performing Phong shading and other advanced shading techniques because there is no storage available at each pixel to hold intermediate values.

6.1.7.5 Just-In-Time Pixels

The problem of temporal distortion due to sequential scan-out of images to the display device, described in section **A.8.1**, is solved by the *Just-In-Time* pixels technique

1. If the host fails to provide viewing position information fast enough, SLATS uses the latest position received; it does not wait for the host.

tially distributing primitives (and the editing commands at each frame) to the SGPs. After the host has sent the editing commands (the SEND stage), the GPs update the model (the EDIT stage) and the start traversing and transforming the primitives (the TRANSFORM stage). The screen is split into 128 x 128 pixel regions, the TRANSFORM stage sorts the primitives into bins based on which screen regions the primitives overlap. After all SGPs complete the TRANSFORM stage, the MGP starts the RENDER stage which distributes the contents of these bins to renderers. After all bins for a particular area have been rendered, the contents of the renderer assigned to that area are copied to the frame buffer (the COPY stage), and the renderer is reassigned to a new area. Separate frame buffers are supported for stereoscopic views in which case all stages after SEND and EDIT are executed once for each view.

6.1.7.4 SLATS

SLATS is an experimental replacement for the standard Rendering Control system software described above, which reduces the latency of the system (measured from the start of transformation to the beginning of scan-out) to 16.7ms (one NTSC field time) **[CoOl94]**.

The SLATS system has been implemented on a Pixel-Planes 5 system configured with one GP, 5 renderers and 2 NTSC frame buffers. Unlike the standard configuration described above, SLATS processes the views for each eye simultaneously. In the configuration described in **[CoOl94]**, each renderer is preassigned two areas of the image for each eye - thus four areas in total per renderer. Each renderer stores all four areas in its local memory until the COPY stage.

- Power consumption and complexity

  The system consists of over 2000 logic enhanced memory chips (and numerous other control and processing chips). Besides being large and complex this requires nearly 4kW of power **[EAFG88]**.

6.1.7.3 Pixel-Planes 5

Pixel-Planes 5 attempts to reduce the inefficiency problems of Pixel-Planes 4 while increasing the rendering capacity. It consists of a high speed ring network onto which graphic processors, renderers, the host and the frame buffer are connected. The number of graphics processors can be varied be from two to over fifty and the number of renderers can be from one to more than twenty five depending on the needs of the particular application. **[HoLa93]**

The renderers are built from updated versions of the Pixel-Planes 4 rendering chips, which have enhanced expression evaluation trees capable of calculating the quadratic $Ax + By + C + Dx^2 + Exy + Fy^2$ **[FPEG89]**; This enables fast Phong shading to be implemented and simplifies the implementation of curved surfaces. Each renderer has 128 x 128 cells of logic enhanced memory.

There exist a number of different ways of using the Pixel-Planes 5 rendering hardware. The following description, based on the description of the Rendering Control software given in **[CoOl94]**, suffers from poor latency characteristics due to the need to sort the primitives after transformation. Two other organisations, SLATS and Just-In-Time Pixels, described in sections **6.1.7.4** and **6.1.7.5** respectively, attempt to reduce latency and temporal distortion respectively.

The graphics processors are organised into a Master Graphics Processor (MGP) and a number of Slave Graphics Processors (SGPs). The MGP controls all other parts of the system. The system is configured as a retained mode rendering system, with the host inibehind the value from the expression tree. The value is stored into those cells still enabled. Finally, the coefficients for each colour component are sent together with a command to store the output of the expression tree in the enabled cells memory.

Video display is performed by scanning out the image from certain locations of each cell's local memory. Changing the section of local memory used for scan out allows double buffering.

6.1.7.2 Pixel-Planes 4

Pixel-Planes 4 consists of enough logic enhanced memory chips to provide one ALU per pixel, together with hardware to calculate the coefficients of the linear expressions and to control the memory array. Rasterisation uses the entire array of chips simultaneously to render one primitive with all commands and coefficients being transmitted to all chips simultaneously. The timing analysis given in **[FGHS85]** states that Pixel-Planes 4 is capable of rendering 30 000 four sided Gouraud shaded polygons per second. Unlike most other systems the time to render the polygon is completely independent of the size of the polygon.

The ability to perform operations on all pixels in the array simultaneously allows other facilities such as shadows and fast sphere rendering to be implemented in an efficient manner. Fuchs et. al. shows in **[FGHS85]** that shadows can be implemented using a technique based on shadow volumes.

Pixel-Planes 4 does suffer a number of problems:

- Inefficiency

  The whole array is used to render a small primitive even though it affects only a small proportion of the display.

Pineda's algorithm can be viewed as computing all the edge function values (and shading and depth information) simultaneously for one pixel (or a small block of pixels) at a time. The Pixel-Planes system evaluates each of the edge functions (and the shading and depth functions) one at a time - but for every pixel on the display simultaneously.

Edge functions and the functions defining linear shading and depth interpolation on a planar polygon can all be represented in the form:

$$i = Ax + By + C \qquad \text{EQ 6.1}$$

Each Pixel-Planes chip contains an array of logic enhanced memory; each cell of this array represents one pixel and consists of a serial ALU, an area of memory, an enable bit and a serial adder. A linear expression tree simultaneously calculates $Ax+C$ for every column of the memory array. Another expression tree calculates By for every row of the memory array. The output of the two linear expression trees is combined in the serial adder in each enhanced memory cell.[1]

All ALUs which have their enable bit set operate in lock step fashion on the same command. The operands for the ALUs can come from the memory on each cell, or from the linear expression evaluation tree (via the one bit adder in each cell). Results are stored in each cell's memory, or are used to clear conditionally the enable bit and thus disable further processing by that cell.

To render a triangle, the enable bit on all ALUs is set and the coefficients of each edge are sent to the expression tree together with a command to disable cells where the expression is less than zero. The coefficients to produce depth information are sent, together with a command to disable cells where the stored Z value represents a point

1. Some extra hardware is required to offset the results of the linear expression trees based on the position of the particular chip within the whole image.

It does however suffer from a number of major disadvantages:

- It is considerably more complex than a conventional Z buffered system.
- A hard limit is placed on the number of polygons per frame by the amount of storage in each section of the pipeline (and by the width of the buses which address them).
- The large amount of custom hardware leads to an inflexible system.
- The system has bad latency characteristics.

For VR applications it is the last of these problems which is likely to be the worst problem. The poor latency is due to the need to bucket sort edges before hidden surface removal, shading and span filling. The bucket sorting in the edge-list manager can be pipelined with the transformation in the preprocessor; however the output stage of the edge-list manager cannot be overlapped with the input stage.

In general the added complexity inherent in implementing this algorithm can not be justified except for the few applications which demand accurate transparency effects.

6.1.7 Pixel-Planes and Pixel flow

The University of North Carolina (UNC) at Chapel Hill have developed a number of novel rendering architectures based around a combination of logic enhanced memory and highly parallel expression evaluation. The description which follows describes the principle of operation of the Pixel-Planes rendering chips and then describes how this principle is used in Pixel-Planes 4, Pixel-Planes 5, and the forthcoming PixelFlow system.

6.1.7.1 The Pixel-Planes logic enhanced memory chip

The method of rasterisation employed in Pixel-Planes is similar to the edge based algorithm used in Pineda's algorithm (described in section **A.5**). A typical implementation of tion performed during the second stage. The second stage is the ‘output stage’, which maintains the active edge list, interpolates parameters from scan-line to scan-line and outputs edge information to the hidden-surface removal card.

The hidden-surface removal card takes the active edge list produced by the edge-list manager card and generates horizontal spans of constant depth ordering. It does this by maintaining an active polygon list in the manner described in section **A.2.2.2**. The shading card is sent information regarding the edges encountered, the size of each span and the polygons in the span.

The shading card calculates the colours at the ends of each span; this process includes blending to account for the effect of partially transparent polygons. This colour information is then used together with the length of the span to calculate values for the span interpolation process which takes place in the frame buffer card.

The frame buffer card takes the colour information for each span and fills the spans into a section of frame buffer memory local to each frame buffer card. Each frame buffer card provides a guaranteed fill rate of five million pixels per second, even where spans are only one pixel in length; the filling and interpolation is based on the Swanson and Thayer algorithm **[SwTh86]**. The frame buffer storage itself is double buffered to allow for the display refresh rate to be higher than the redraw rate. Finally, the output of the frame buffers are combined in the video card, which drives the display device.

The system is implemented using conventional processors (the AMD 29000) and custom hardware. **[Vial93]** suggests ways to extend the system to accommodate shadows, Phong shading, texture mapping and anti-aliased rendering. The advantages of the techniques used in this system over the conventional Z buffer approach are:

- Support for non-refractive transparency.
- Each pixel of the frame buffer is written once and once only per frame.
- If one arm of the architecture fails a recognisable picture is still presented.

values for the interpolation of lighting and depth. The output of the preprocessing stage consists of a list of polygons and a list of edges (neither of which is in any particular order).

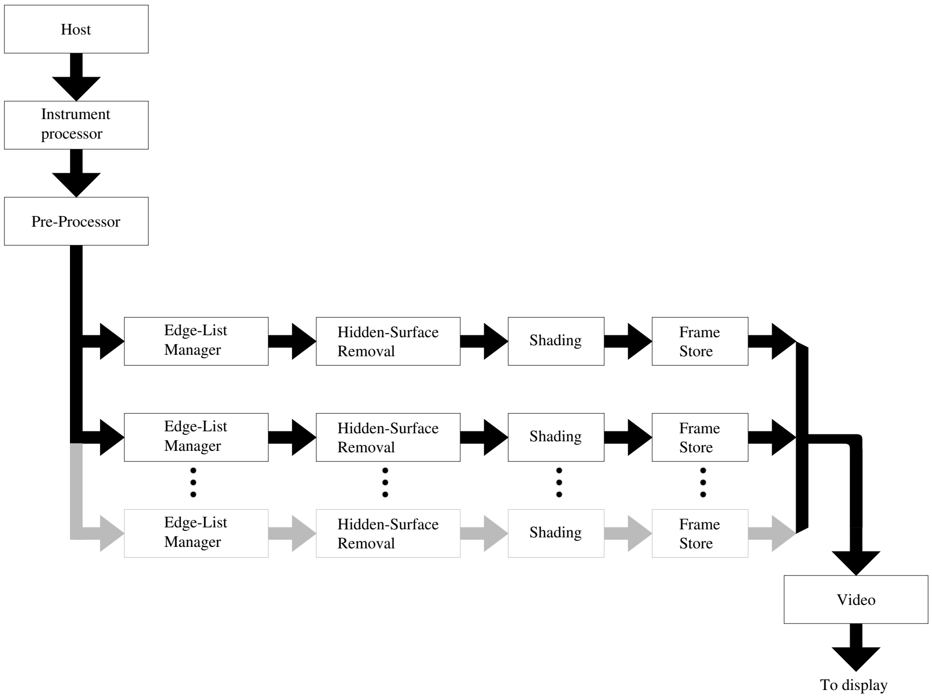

Figure 6.3: The architecture of the Vialls rendering system

The output of the preprocessor is fed to a number of rendering card sets, each consisting of four cards: The edge-list manager card, the hidden-surface removal card, the shading card and the frame store card. The system implements a parallel version of the spanning scan-line algorithm, by distributing scan-lines among the card sets; in a system with ‘n’ card sets every card set renders every nth line. This parallelism allows the required resolution and update rate to be achieved by configuring the system with an appropriate number of card sets for the particular application.

The edge-list manager card takes the list of polygons and edges provided by the preprocessor as input. The processing in the edge-list card consists of two pipelined stages. The first is the ‘setup stage’ which takes the edge list ‘bucket’ sorts the edges on their minimum Y coordinate and performs setup calculations for the colour and depth interpola-

6.1.6 The Vialls Three-Dimensional Graphics Engine

This system was designed specifically for civil aircraft instrumentation and, in particular, the display of ‘Tunnels in the Sky’ overlaid on images of the outside world to show the pilot the direction to fly in when approaching the runway or aborting an approach **[Vial93]**. One particular problem with this application was that opaque tunnels could obscure important detail; for this reason an accurate representation of transparency was deemed essential.

The minimum specification for the system is given in **[Vial93]** as:

- A resolution of 1280 x 1024 pixels
- A 60Hz display refresh rate
- A 30Hz image redraw rate
- 2000 triangular polygons per frame
- Gouraud shading
- Simple, non-refractive transparency

These specifications are rather different from those of commercial rendering systems; in particular the number of polygons per frame is low, yet there is the need for accurate transparency which is difficult to achieve with the Z buffering algorithm. These issues led the designer to choose the spanning scan-line algorithm (described in section **A.2.2.2**).

Figure 6.3 shows the overall architecture of the system[1]. The host in this system provides flight data (such as location, velocity etc.) to the *instrument processor*, which maintains a polygonal model of the instrument which it then provides to the *pre-processor*. This then performs all transformations, clipping, and lighting as well as calculating

1. This diagram is based on figure 2.6 of **[Vial93]**.

object processor pipeline each packet contains a list of contributions from primitives potentially visible at the pixel that packet represents, sorted in front-to-back order.

The output of the object processor pipeline passes into the shading pipeline; this consists of a number of shading processors, each one calculating the diffuse or specular component of one light source. An implementation of the shading processors, capable of performing Phong shading is described in **[Clau92]**. The filter stage combines the contributions to generate a final pixel colour using subpixel masks to perform antialiasing

A problem with PROOF and similar systems is that the limited number of object processors can limit the overall complexity of the scene. One solution to this problem is to feed the output of the pipeline back through the pipeline with a different set of object information loaded into each processor and thus effectively double the number of objects which the system can handle. If such a system has sufficient processors for the number of primitives being rendered, and a limit is placed on the number of contributions to each pixel, it is possible to guarantee that an image is rendered in a given time. If the pipeline is fast enough it will generate pixels at the rate needed to refresh the screen directly and thus not require a frame buffer; such systems do not require the large amount of frame buffer memory needed by conventional rendering systems and offer a reasonable latency.

PROOF and similar systems require that the transformations for all objects assigned to processors are completed before rendering can begin and this imposes extra latency. In addition the redraw rate of per-object rendering systems is more predictable than conventional rendering systems since, as long as the inter-processor communication bandwidth is great enough and the shading and filter stages are powerful enough, the size of primitives does not affect the rendering time.

e.g. texture coordinates. The interpolated values are passed to the functions processor which is another DSP executing the shading algorithm. This process is relatively slow but highly flexible. The *span Gouraud shader* is a separate span interpolator used to accelerate the shading of simple Gouraud shaded polygons.

The use of a DSP for the functions processor and the microcoding of the interpolation processor makes this system exceptionally flexible without using vast amounts of processing on the host; for simple Gouraud shaded primitives, the span Gouraud shader eliminates the performance penalty often associated with flexible systems.

6.1.5 PROOF

The PROOF (Pipeline for Rendering in an Object Oriented Framework) architecture described in **[ScCl91]** consists of an object processor pipeline, a shading pipeline and a filter system. The object processor pipeline consists of a number of object processors; each processor is responsible for the scan conversion and parameter interpolation for one primitive.

An empty packet is inserted at the head of the pipeline for each pixel on the display; as it passes through an object processor, the packet is modified to incorporate contributions from the primitive which that object processor is responsible for. Each object processor consists of a primitive processor and a comparator. The primitive processor performs scan conversion and coverage calculations while the comparator incorporates contributions from this primitive into the stream of contributions coming from previous object processors. If the primitive does not intersect the pixel represented by the packet from the previous processor, the entire packet is passed on to the next processor. If the primitive does intersect the pixel the contribution of the primitive is combined into the packet; all contributions in front of the new primitive are passed onto the next stage. If one of the contributions (either incoming or generated at this processor) covers the pixel completely no contribution behind it is forwarded to the next processor. At the end of the

6.1.4 The Pöpsel and Tikwinski Multipurpose hardware shader

Like the Image chip, the *hardware shader* performs scan conversion and interpolation in a conventional Z buffer based system. It does not provide support for texture mapping directly and complex shading algorithms, but allows a separate dedicated processor to perform these functions.

The architecture of the 'hardware shader' is shown in Figure 6.2. The extended frame buffer holds colour and depth information and also holds texture maps; the depth comparator is built into the extended frame buffer. The frame buffer is interfaced to the *universal interpolator* and *functions processor* which actually perform shading.

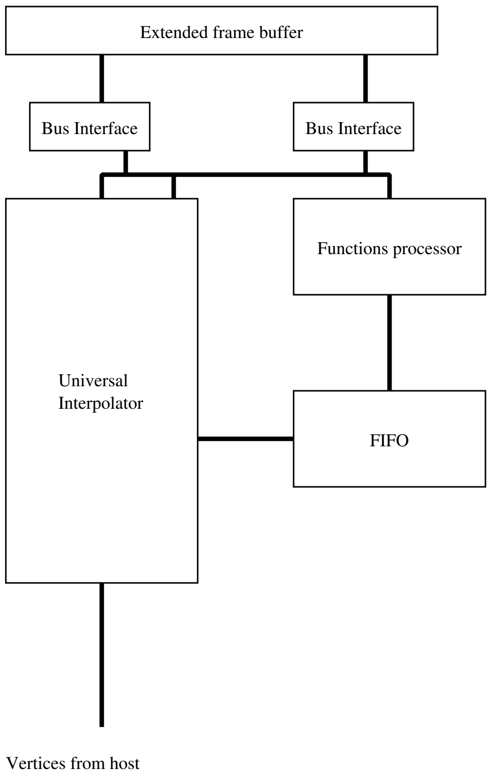

Figure 6.2: The Pöpsel and Tikwinski Multipurpose hardware shader

The universal interpolator consists of a DSP used to perform setup calculations, a microcoded *interpolation processor* and a *span Gouraud shader*. The interpolation processor is used to interpolate any of the parameters needed by the present shading model;

from one chip, while the same design can also be used to produce a workstation level rendering system using multiple Image chips. The Image chip consists of three layers:

- Input processing layer

  This stage performs address decoding to interface with the host processors bus and holds control registers for the rendering process.

- Incremental processing layer

  This layer consists of nine interpolator units; three used for the geometry, three for colour, one for depth, and two for window clipping. Each interpolator has a different width datapath to accommodate the different range and accuracy requirements of the parameters being interpolated.

- Output processing layer

  The output processing layer reformats the depth and colour information before it is output to the memory system; it also includes an antialiasing unit which generates either a subpixel mask or generates a coverage value from current values of the edge functions and edge slopes.

The Image chip does not provide texturing directly, or any of the usual higher complexity functions, however it can provide scan conversion facilities to the host processor. In this mode Image stores pixels in the hosts memory; the host can then perform complex shading functions (such as Phong shading, solid texturing and reflection mapping) in software.

The Image chip provides a reasonable degree of flexibility due to both its ability to be used in parallel configurations and its ability to accelerate complex shading algorithms. The disadvantage of performing complex shading on the host processor is that the host is already heavily loaded in producing setup and control information for the Image chips. Any post-process shading technique is also likely to increase the overall latency of the system.

Y coordinates (aided by a split register file), a special branch instruction (and register) to speed up Sutherland-Hodgman clipping, and a “kitchen sink” instruction which performs a number of house keeping tasks.

Since LeoFloat is so general it is possible to include support for dynamic tessellation of NURBS surfaces; this removes the need for the host to perform computationally expensive tessellation and, the level of tessellation can be made view dependent.

The output of LeoFloat is passed to five LeoDraw chips which perform scan conversion, colour interpolation and hidden surface removal. The frame buffer is distributed among the LeoDraw chips; each chip is connected to every fifth column of the frame buffer. All primitives are broadcast to all LeoDraw chips which render the pixels which they are responsible for. Each LeoDraw chip has an edge walker (which generates the spans) operating on the same data; this was chosen in preference to a system with one edge walker and a routing mechanism, since it reduces the pin counts and the complexity of the buses on the board. LeoCross reads the data from the frame buffer and provides the necessary conversions and timing to provide video output.

The rasterisation methods used by *Leo* are fairly conventional, however LeoFloat is much more flexible than the fixed geometry pipeline used in the Silicon Graphics VGXT.

6.1.3 The ‘Image’ chip

The Image chip implements the scan conversion and interpolation sections of rasterisation **[DWLG92]** using Pineda’s algorithm (see appendix section **A.5**) to perform scan conversion; this was chosen since it allows the entire rasterisation process to be integrated into one chip. The ability to build a system with a single chip was felt to be important since it allows a simple accelerator card for a personal computer to be built

> *"Use of commercially available DSP (Digital Signal Processing) chips for this work has two major drawbacks. First, most such processors require a considerable number of surrounding glue chips, especially when they are deployed as multi-processors. These glue chips can easily quadruple the board area dedicated to the DSP chip, as well as adversely affecting power, heat, cost, and reliability. Second, few of these chips have been optimised for 3D graphics."* **[DeNe93]**.

The effect of the glue logic on board area was a particular problem for the designers of *Leo* since it had to fit on an SBus expansion card which is only 5.7 x 6.7 x 0.6 inches[1]. The solution employed in LeoFloat is to create a microcoded microprocessor which, although completely general, is specifically optimised for the task. Some of the optimisations are:

- Specialist input/output acceleration

  LeoFloat includes extra hardware to interface each chip to the other stages; it packages data and formats it ready for transmission; this relieves the cost of interacting with the rest of the system from the LeoFloat software and eliminates much of the glue logic.

- Register optimisation

  Very few DSPs have enough registers to accommodate efficiently the algorithms used in the 3D graphics pipeline. The designers of LeoFloat wrote the most important algorithm LeoFloat was to execute and then counted the number of registers needed. This algorithm (to process triangles) used 288 registers, and that is the number available to LeoFloat programs.

- Specialist instructions and architectural modifications

  LeoFloat has a number of specialist instructions, which implement sections of the algorithm that are particularly expensive to implement using the conventional instruction set. These include a method of sorting vertices based on their

1. By contrast the Silicon Graphics VGX is built on four triple-height EUROcard boards each 14 x 12 inches.

complex model hierarchies which would require many modifications to the current transformation matrix.

- Windowing support

  Although not a direct disadvantage, the windowing support is unneeded for most VR applications and therefore adds to the cost unnecessarily.

6.1.2 The Sun ZX rendering accelerator (Leo)

The *Leo* system (**[DeNe93]**), (marketed as the Sun ZX rendering accelerator), features a number of interesting architectural enhancements to the conventional rendering pipeline. *Leo* is implemented using four different types of custom integrated circuit: LeoCommand, LeoFloat, LeoDraw, and LeoCross.

LeoCommand interfaces the system to the host system's expansion bus, acquires data from the host, and performs format conversion and vertex reordering. LeoCommand can perform direct memory access (DMA) and fetch geometry from memory under the host's control, without the need for the host to perform the transfer of each data item. LeoCommand performs format conversions and distributes the primitives to one of four LeoFloat chips, which perform all geometric and setup calculations on the primitive. LeoCommand manages the parallelisation of the four LeoFloat chips, both in distributing the primitives and ensuring that the resulting primitives enter the rasterisation stage in the correct order.

The approach taken in implementing the geometric and setup calculations in *Leo* is perhaps its most interesting feature. In many systems (such as the VGXT) the geometric and setup processing for one primitive is split among many different stages of the pipeline; in *Leo* each LeoFloat chip performs all the computation for one primitive before passing it onto the rasterisation stage. Commercially available Digital Signal Processor (DSP) chips were considered for this job but were found to be unsuitable:

The VGX and VGXT are both designed to operate with a window based user interface; for this purpose they incorporate a window identification field in the frame buffer which allows objects to be clipped to window boundaries with very little penalty.

The VGX suffers from a number of problems when used in Virtual Reality applications:

- Multiple image generation

  The VGX provides support for frame sequential stereo but produces only one channel of output and is thus unsuitable for use with head mounted displays Multiple channel output can be added through the use of the *Video Splitter Option*, which is expensive and adds an extra frame of latency to the system.

- Host processor loading

  The command processor must be passed each command and vertex individually by a host action. Since the system is based around the immediate mode of operation the host must also perform model traversal. Analysis of a number of applications executing on a Silicon Graphics Crimson workstation fitted with a VGXT rendering system, indicate that despite a fast processor many applications were limited by host performance.

- Flexibility

  The stages of the pipeline are tightly bound to specific tasks and thus it is not possible to optimise them for different classes of application[1]. The frame buffer is made more flexible by the inclusion of a set of *stencil planes*, which can be used to implement constructive solid geometry rendering.

- Changes in the current transformation matrix

  The VGXT is capable of changing the current transformation matrix less than 34 000 times per second (while otherwise idle). Many VR applications have

1. Some stages are microprogrammed, however there is no facility for the programming to be modified.

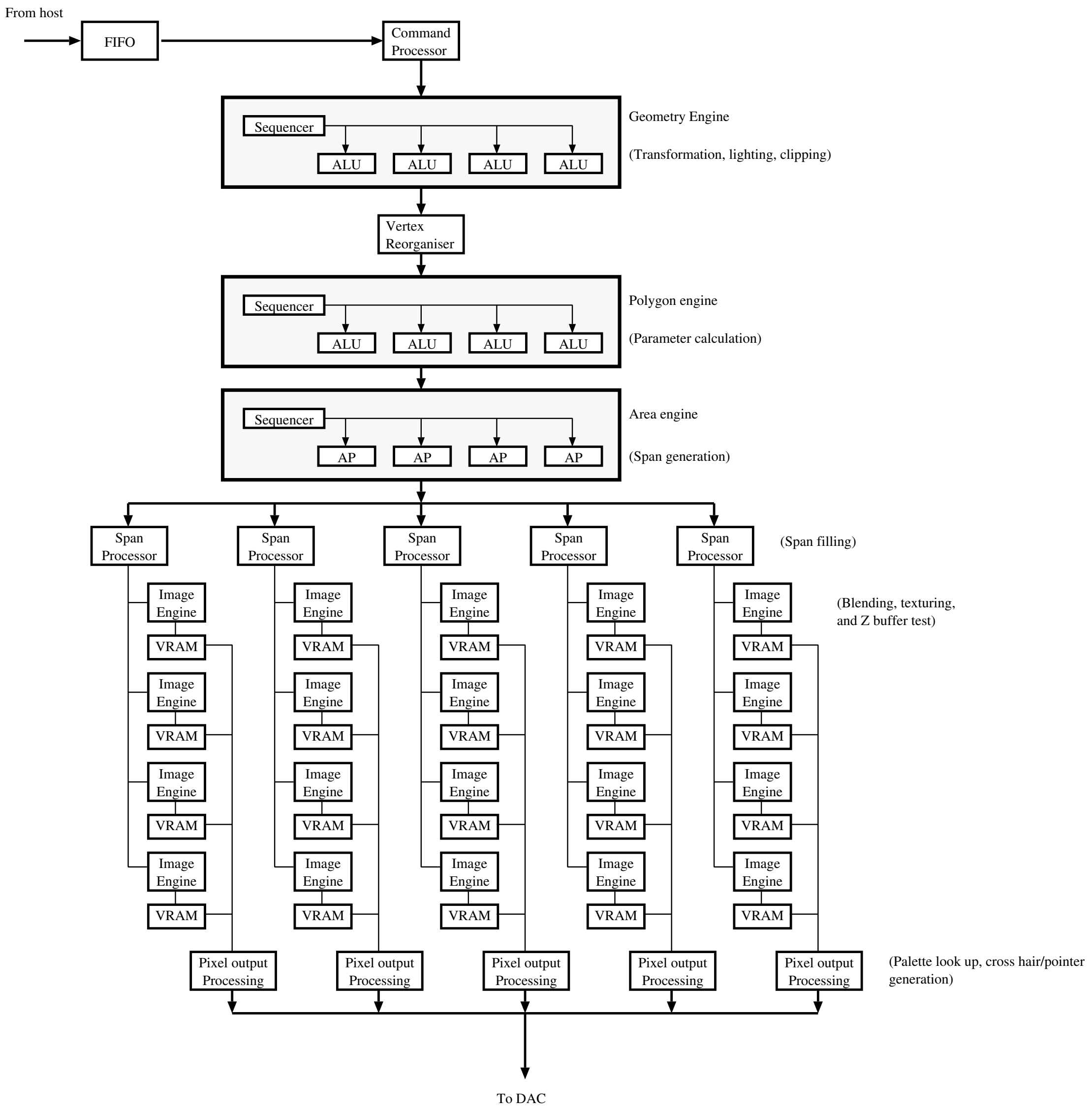

Figure 6.1: The architecture of the Silicon Graphics VGX graphics subsystem

simultaneously. One unusual feature of the VGX is that spans are vertical rather than horizontal collections of pixels.

The VGX supports a number of different texturing algorithms from point sampling to a bilinear approximation to MIP mapping **[Will83]**. The VGXT has more advanced Image Engines which also provides true trilinear MIPmap interpolation. Another difference between the VGX and VGXT is that the VGX must perform perspective correction by subdividing the primitives, while the image engines of the VGXT provide true per-pixel perspective correction.

Chapter 6

Existing systems

This chapter describes existing rendering hardware, software and enabling technology, and analyses their suitability to VR applications.

6.1 Rendering hardware

This section describes a selection of the many hundreds of rendering systems which have been developed over the last couple of decades. Most of the systems described in this chapter were chosen because of an unusual architecture, feature, implementation decision or intended application area; the only exception to this is the Silicon Graphics VGXT which represents a high performance conventional rendering architecture and is included as a point of comparison.

6.1.1 Silicon Graphics VGX/VGXT

The Silicon Graphics VGX is one of a range of rendering subsystems produced by Silicon Graphics for use in their Unix workstations. It has a mainly conventional architecture, designed for use with immediate mode graphics libraries such as Silicon Graphics GL, enhanced by a large amount of parallelism. Figure 6.1 (derived from **[SGI92a]**), shows the architecture of the VGX graphics subsystem. The VGXT is identical in structure and only differs in having improved texture support in the Image Engines.

The VGX architecture is highly parallel. The Geometry, Polygon and Area engines are all implemented as SIMD processing systems. The rasterisation stage incorporates interleaving both horizontally and vertically, and allows up to twenty pixels to be filled