Integration of Live Video and Computer Graphics for Video Effect Generation

Hans-Josef Ackermann and Utz Osterfeld
Fraunhofer Institute for Computer Graphics
Darmstadt, Germany

Abstract

We present the architecture of a system, based on a PC, covering real-time video processing by integration of live video and computer graphics. The video processing and integration are realized by generation and evaluation of an on-line MIPmap video texture. This new application of a known filtering technique for still textures in the field of live video processing permits an easy and high performance integration of video and graphics to speed up the video processing. The video texture is mapped on surfaces defined by graphics primitives. The graphics primitives are processed by a standard graphics pipeline with special hardware support for the rendering part.

1 Introduction

In the last few years, multi-media became a new keyword and a new dimension in computing and computer interfaces. After sound being integrated, the developers went towards the integration of live video in PCs and workstations. Who ever joint a computer fair recently and watched the fascination of people watching themselves or others on a computer monitor or on a video-phone knows, that the developers hit well. Digitized still images can be fully integrated and enrich all types of documents and advertisements. But thinking of live video, there is no integration with computer-graphics but only a coexistence. This is due to the technical realization of displaying video on computers. The video normally is digitized at video frame rate and stored. During the screen refresh of the graphics image (normally at a higher refresh rate), the video information from the buffer is read out and displayed in a window area of the graphics image as an overlay, separated from the graphics. There is not even a temporal connection between the graphics frames and the video frames. This may be sufficient for some applications, but others like video effect generation and animation cannot profit from live video this way. In our paper we outline a system small enough to fit into a PC but powerful enough to really integrate video and graphics. Moreover the system offers enough rendering performance and texture mapping features to support real-time simulation applications.

2 System Features

Our system unites the features of both video processing systems and graphics systems. The supported video functionality includes:

- temporal conversion of the video
- arbitrary placement of the video on the screen
- video compression and decompression for sequence storage and replay

The integration of graphics and video allows to display any video effects as known from TV and commercials like:

- rotation of the video image with arbitrary angles
- flips, wipes, fly-ins, fly-outs, explosions, etc.
- true perspective mapping of the video on arbitrary objects like cubes or spheres

Moreover, our system contains a high functionality graphics system. The real-time video capabilities allow the following additional features:

- real-time video background for animations and environmental simulations
- mapping of live video on animated objects within the graphics scene
- sequence controlling during the integration of video and graphics

It should be noted, that the main goal of our system is to demonstrate the benefit of the integration of video and graphics. Providing facility for cutting and editing of the video is not the focus. Such features do not influence the hardware architecture of the system. They require an appropriate piece of software for the functionality and the user interface. Furthermore, recording the processed images to video tape requires an additional temporal conversion and some format conversion circuitry.

3 Functional Architecture

Figure 1 shows the functional architecture of our system that realizes the features described above. The system consists of a video module with video preprocessing and video texture generation, a graphics module with fast graphics generation, video texture mapping and picture generation.

3.1 Functional Description of the Video-Module

For a flexible and high quality integration of live video and computer graphics, the video processing has to fulfill several requirements. Displaying video and computer graphics together on the same monitor requires the adaptation of the video frame rate to that of the monitor. In order not to lose the image quality of the video, this adaptation has to be done in real time. That means, that the frame rate of the processed frames has to be equal or higher than the original frame rate of the video. This real time frame rate conversion, in the following referred to as

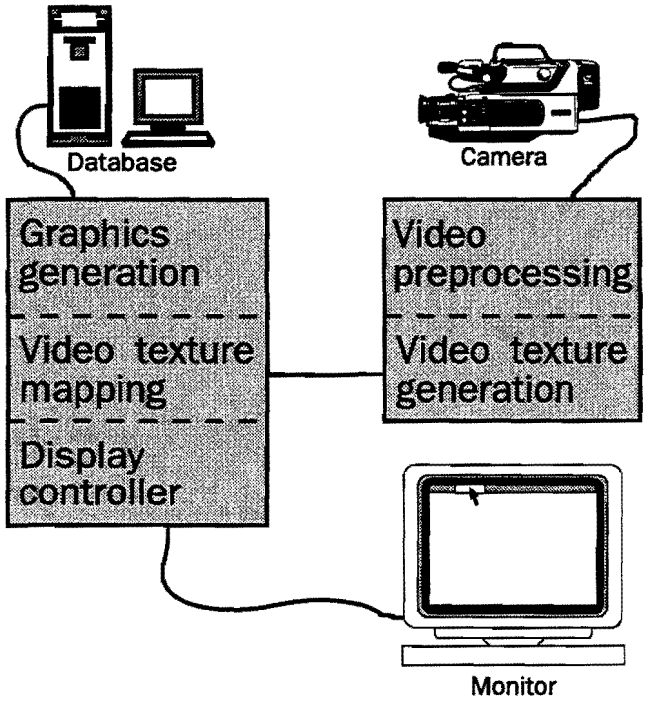

Figure 1: Functional Architecture

video-preprocessing, is realized in most video overlay boards for personal computers or workstations. These overlay boards allow to display the video in a separate window as overlay to the graphics. To achieve a more flexible integration of video and computer graphics, we generate one or several textures from the video. Then, this textures can be mapped on a surface of a graphics object like an ordinary still texture. To integrate live video as video texture, some video processing is required, that generates the textures in real time.

3.1.2 Video Preprocessing

Standard video cameras or video cassette recorders deliver images as analog video signals coded according to NTSC or PAL format. To integrate these video images and the graphics images, the analog video signals have to be digitized first. Afterwards, they are temporally filtered.

The temporal filtering is necessary to preserve the image quality. Without filtering, motions in video images appear jerky. This is due to the different sampling rates of video and graphics. NTSC coded video scenes are sampled at 30 frames per second, while for example advanced PC monitors display this images at 70 frames per second. The way the frames are generated is different as well. A video frame consists of two fields (odd and even field), which contain odd and even lines. The fields are displayed at a repetition rate of 60 Hz. Although the information of a whole frame is displayed at 30 Hz only, the 60 Hz field repetition rate reduces the flicker of the video without increasing the necessary bandwidth for transmission. This method is called interlacing. Most graphics adapters however use a full frame or non-interlaced mode to display the images. This is the reason why uniform motions, sampled in TV norm, mostly do not appear as uniform motions when displayed on a graphics monitor. Temporal filtering works as a conversion between video frame rate and graphics frame rate and is done by interpolation between different input frames.

The digitizing of the analog video signals is performed at a resolution of up to 768 x 576 pixels per image with 24 bits o color depth. Up to three subsequent video frames may be store and evaluated for the following temporal filtering. The vide images are stored with 24 bits of color depth or dithered t reduce the color depth to 8 bits. The dithering algorithm that i processed in real-time, is known as ordered dithering.

The temporal filtering converts the frame rate of the video images to the frame rate of the video texture.

The texture frame rate can be programmed, to combine the video- and graphics raster-image of the PC with a minimum o temporal aliasing. To display the video textures on graphics monitors, the texture frame rate can be selected from 25 frame per second to 38 frames per second. The conversion algorithm can be programmed as well. The frame rate conversion can be done with or without interpolation between subsequent vide frames. To generate the texture frames, the conversion uses all input frames without loss of information.

Conversion without interpolation is done by selecting a video frame that will become the next texture frame. Conversion with interpolation is done by weighting the two color values of the same pixels from subsequent video frames with coefficients and adding the results. The values of the coefficients represent the temporal function of the interpolation, which may be linear or non-linear. This conversion algorithm provides a good trade-off between flexibility of generating different graphics frame rates and the necessity of additional hardware for the interpolation.

The image quality depends on the algorithm, the image information and on the accuracy of the coefficients. Filtering without interpolation or with non-linear interpolation offers highest quality if fast motion of high contrast moving edges exists in the video. Linear interpolation offers best quality with slow motion of moving edges in the video. The accuracy of the coefficients determines how often per picture the weighting can be adapted. We achieved best image quality when the weighting is adapted at least after eight rows.

3.1.3 Video Texture Generation

The temporally filtered frames are one part of the video texture. The whole texture is generated as MIPmap structure [6]. MIPmapping is a filtering technique for still textures known from computer graphics. In this representation, besides the image with the intrinsic resolution, filtered images with lower resolutions are available. The filtering is done by simply accumulating the color values of four adjacent pixels of the higher resolution image and averaging the result to compute one pixel at lower resolution. In this way, the size of the different levels of detail, as the filtered images are called, decreases about a factor of four from one level to the next. The MIPmap filtering of textures is normally done off-line. The exact, position-dependent scale factor for the mapping can be calculated on-line by interpolating between two pixel values of the two nearest levels of details. The advantage of the off-line MIPmap prefiltering is the predictable and short generation time for the on-line scaling.

To use this algorithm for scaling and manipulating video images means that for each video frame a MIPmap image has to be calculated. This leads from off-line MIPmap generation, proposed for graphics texture prefiltering, to "On-line MIPmap" generation for live video images [4].

3.2 Functional Description of the Graphics-Module

Our basic approach for the integration of graphics and video is to use the preprocessed video as a texture to be mapped upon graphics objects. As the real-time requirements imposed by the live video are handled by the video preprocessing stage, no major conceptual changes to the classical graphics pipeline are necessary. The graphics module acts as a MIPmap-evaluator to map the video texture, preprocessed by the MIPmap-generator to a surface described in display coordinates forming a 3D-space. Arbitrary manipulations of the surfaces can be achieved through proper transformation in 3D space. The surface upon which the texture is to be mapped is tessellated into triangles. In the most usual case, the surface is a rectangular window. In this case the surface is broken down into two triangles covering the area of the rectangle. A very complex case could be the surface of a sphere. The sphere has to be approximated by triangles as well. Depending on the quality that is required, up to thousands of triangles are necessary to describe the surface.

The graphics pipeline can be subdivided in two blocks with different tasks. The geometry block does all the calculations related to the different transformations of the objects, the clipping and, assuming the Gouraud shading model, the lighting at the vertices of the triangle primitives. These calculations are normally done by the host or a special accelerator using floating-point representations.

The second block does the scan conversion of the primitives including hidden surface removal, shading, blending and texture mapping functions. All these calculations are done on a pixel by pixel base. The number of calculations depends on the number of pixels that have to be actually rendered. As the functions necessary for rasterization are simple and normally done in integer representation, they are often realized using proprietary high speed hardware. The geometry block consisting of programmable processors is functionally flexible. Consequently, the rendering hardware determines the functional and performance differences between different graphics systems. Our system will support the following features:

- Gouraud shading
- z-buffering
- depth cueing
- perspective correct video-texture mapping with MIPmap approach and tri-linear interpolation
- flexible texture formats: 32 bit r,g,b,α, 16 bit r,g,b, 8 bit intensity
- alpha blending for transparency
- intensity manipulation to support lighting effects on textured surfaces
- anti-aliasing
- scaleable performance
- flexible screen resolutions from 640 x 480 to 1280 x 1024

3.2.1 Performance Requirements

The graphics object upon which the video texture is mapped has to be generated at a repetition rate equivalent or greater than the original frame rate of the video. This ensures, that no information from the video is lost, which would result in jerky motion. The computational power required for this task depends only on the number of primitives to be displayed. So its performance is measured in triangles per second.

To ensure the real-time capabilities assuming a certain frame-rate, the graphics module has to fulfill two different requirements:

The geometry block must be capable of calculating a sufficient number of triangles per second. Assuming a frame rate of 30 Hz and a screen complexity of 1000 triangles per frame, a performance of 30,000 triangles per second is required. This sounds not too challenging but to reach this performance using a single processor like the i860, you have to keep the overhead small and optimize kernel routines. Furthermore, if there are curved objects that are approximated by triangles, 1000 triangles are easily spent for only one globe.

The rendering block must be capable of producing the pixels required to displaying the scene including the mapped video in real-time. The number of pixels per second to be generated mainly depends on three factors: the frame rate, the screen resolution (if whole frames are generated) and the medium coverage factor. Assuming a frame rate of 30 Hz, a resolution of 800 by 600 pixels and a coverage factor of 1.4, the required performance is about 20 Mpixel/s. That leaves only 50 ns for the whole calculation of one pixel including texture buffer and z buffer lookup as well as frame buffer and z buffer write.

3.2.2 Algorithmic Approach

The triangles, being the main graphics primitive, can be rasterized in two different ways depending on the anti-aliasing mode. If no anti-aliasing or supersampling is used, the triangles are rasterized using the exact point sampling technique and subpixel addressing in order to avoid artifacts. Details of the algorithm and its hardware realization are described in [1] and [2]. To support a less costly anti-aliasing method, area-sampling using stochastically distributed sampling points can be applied to the triangle edges. The number of triangles that may contribute to a single pixel is flexible but for normal scenes, four (that means four edges meet within a pixel) seems sufficient. Determination of visible surfaces in case of overlapping objects is done by applying a z-buffer algorithm. For native shading, the Gouraud algorithm, that does linear interpolation of the color values between the vertices of a triangle, is realized. The way the texture mapping is done, depends on the quality that is required. According to the interpolated texture addresses, color values are read from the texture memory that contains the MIPmap structure. The interpolated z-value determines the level(s) of detail to be used. According to [3], There are different possibilities to determine the resulting pixel value.

1. The easiest way is to just read one value from one level of detail (pointsampling). This algorithm can produce visible artifacts when objects move within the scene.

2. To increase precision, up to four values from one map can be weighted according to the fractional parts of u and v addresses (bi-linear interpolation). This reduces artifacts when objects move in x and y direction.

3. To reduce artifacts when objects move in z direction as well, values from the two nearest levels of detail can be interpolated using weighting factors depending on the fractional part of the z-address (point sampling, MIPmap).

4. The most complex algorithm is the combination of 2. and 3. In this case four values from two different levels are used to determine the resulting pixel (tri-linear interpolation). This produces the best result but requires eight reads, calculation of 10 weighting factors, 10 multiplications, 7 additions and 3 shift operations for each separate color value (r,g,b,α) of each pixel.

Textures can represent a certain intensity distribution that modulates the intrinsic color and the intensity as result of the lighting. In this case an 8 bit value seems sufficient. Where

texture is used to generate a photo-realistic image, like in environmental simulation, the texture consists of are true color values. Lighting effects are modeled assuming, that the surface, on which the texture is to be mapped, has an intrinsic white color and calculating the effects of the light sources for this surface. The texture value and the value from the lighting calculation are then blended together. For the video textures it depends on the application whether they are mapped unchanged or blended with the result of a lighting calculation. Both possibilities can be realized.

While texture mapping helps to gain a more realistic image of a scene without the necessity of very complex models, there exists one problem. If the scene consists only of a few big surfaces, perspective distortions will be visible. The distortions increase with the difference of the extreme z-values within a surface where the z-values are linearly interpolated. That is due to the fact, that the linear approximation of the perspective equation, which requires to calculate the reciprocal of z, is rather bad where the derivation of the function is big. Subdividing the area covered by a triangle in a way, that the range of its z-values is small, decreases visible artifacts but burdens the geometry unit by the higher amount of triangles. An other way is to approximate the perspective equation with a quadratic interpolation. This approach avoids division and works better than linear interpolation. The effort in hardware is medium. However significant errors occur in critical cases. To gain the most precise results, the exact perspective equation can be realized. This however requires a division and some multiplications/accumulations per pixel which means a big expense of hardware.

4 Hardware Architecture

Figure 2 shows the block diagram of our system architecture. Our two modules are connected to the system bus of the host to provide an interface for control data. The graphics-module reads texture data using a separate 32 bit connection to the texture-buffer. The camera is connected to the analog interface of the video module. It accepts PAL and NTSC coded FBAS signals as well as separate luminance and chrominance signals for SVHS quality. The monitor is connected to the r,g,b outputs of the display controller included in the graphics module. Depending on the configuration of the system, the output of the graphics module displays both system information and processed application image (single display configuration) or a separate monitor for system messages and user interface is connected to the system display adapter (dual display configuration).

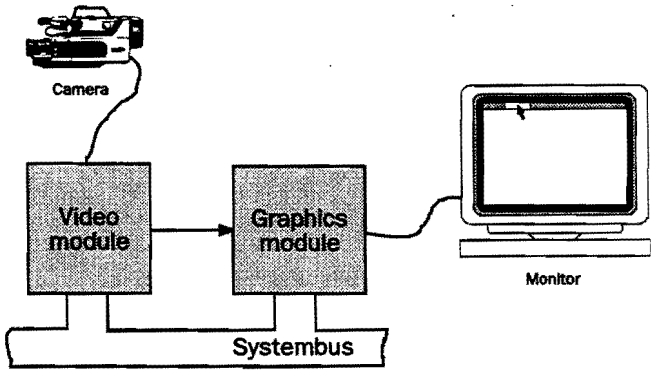

Figure 2: Hardware Architecture

4.1 Video Module

The video module consists of the video preprocessing and the texture generation. The video preprocessing comprises the video input circuit, the video buffer and the temporal filtering circuit. The texture generation comprises the MIPmap generation circuit and the MIPmap double buffer. Figure 3 shows the connection of the units and the signal flow through the circuits. The video module is realized using standard logic components and programmable logic devices.

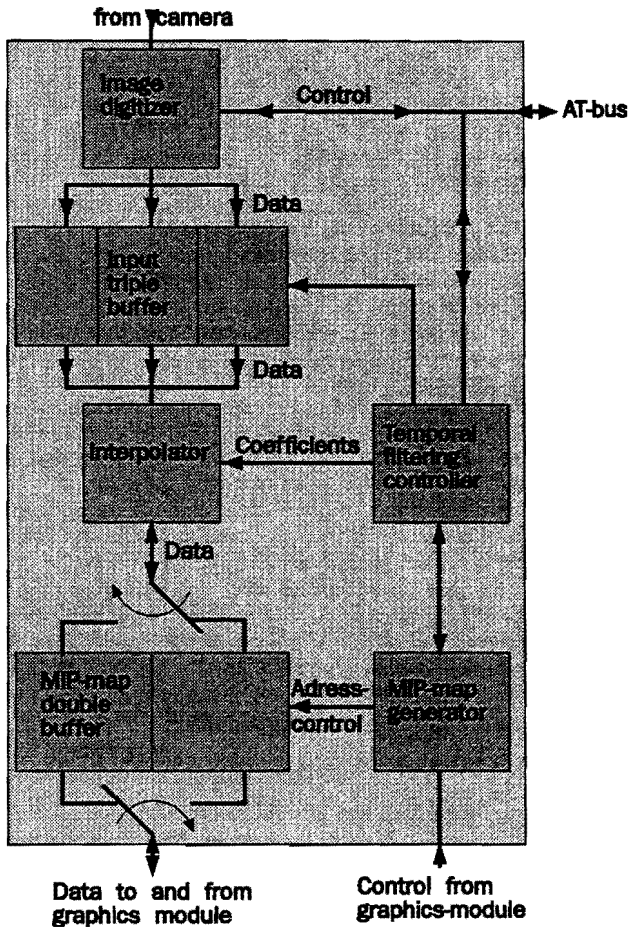

Figure 3: Architecture of the Video Module

The functionality the video module provides is:

- support of standard video formats (PAL, NTSC) in quality up to SVHS
- support of 24 bit color depth for the video images
- optional dithering of the video images to 8 bit color depth
- generation of on-line video textures with formats up to 768x576 and 35 Hz non-interlaced
- generation of MIPmap textures

4.1.1 Video Input Circuit

To implement the video input circuit, we chose a chipset from Philips Semiconductors, which offers high quality raster image generation and programmable digital color filtering with only a few components. The analog camera signals, FBAS, or luminance chrominance are digitized, digitally filtered to YUV and color converted to RGB24.

The chipset is programmable to digitize NTSC or PAL coded video. For software controlled initialization, the chipset is connected to the AT-bus of the PC. The optional real time dithering to 8 bit (3 bit R, 3 bit G, 2 bit B) is done in hardware. The raster images of 768x567 size are stored in the video buffer. The video buffer connects the video input unit and the temporal filtering unit. Because the video buffer is written and read asynchronously from different units, it requires independent interfaces for read and write operations. As best choice for the memory, we found field memories by Texas Instruments. These field memories have independent and asynchronous interfaces for read and write operations and, since they work like FIFOs, they do not need an address generation.

The video digitizing runs asynchronously to the temporal filtering. The stored video images have to be accessed independently from the digitizing unit and the temporal filtering unit.

4.1.2 Temporal Filtering Circuit

The write control signals of the video buffer are derived from the video synchronization signals. The read control signals of the video buffer are generated by the temporal filtering controller. The interpolation for the temporal filtering is done by the interpolator. The interpolator reads the color values of two video pixels, stored in the video buffer, weights this pixels with coefficients, adds the result and writes one pixel to the texture buffer. The coefficients to weight the color values are stored in a coefficient memory. The addresses for the coefficient memory are generated by the temporal filtering controller. The interpolator is realized with special multiplier accumulator chips from TRW LSI Products. The temporal filtering controller is realized using programmable logic devices from AMD and Lattice. The temporal filtered image, stored by the interpolator in the texture buffer, is the first level of detail of the MIpmap texture.

4.1.3 MIPmap Generation Circuit

The lower levels of detail are generated successively from the higher ones. This means, the second level of detail is calculated from the first, the third is calculated from the second, and so on. All levels of detail are stored in the MIpmap buffer. As described, the calculation of one pixel at lower resolution is done by interpolation among 4 pixels from a level of detail with higher resolution. This interpolation is done by the interpolator of the temporal filtering unit and controlled by the MIPmap controller. This controller includes a state machine and an address generator to control the read and write operations of the texture buffer. The MIPmap controller is realized using programmable logic devices from AMD.

4.1.4 Texture Buffer

Since the MIPmap generation and the spatial conversion need fast random access to the texture buffer, the buffer is designed with an interleave of four and built of fast SRAM. To achieve a texture generation- and evaluation in real-time, texture-generation and evaluation have to be done in parallel and so the texture buffer is realized as double buffer.

4.2 Graphics Module

The graphics module comprises three units, the geometry unit, the rendering unit and the display controller unit. Figure 4 shows the architecture of the graphics module.

4.2.1 Geometry Unit

As mentioned above, the geometry calculations are either done on the host CPU or on a special accelerator module. We realized several accelerator modules employing floating point processors like the i860 in a single processor configuration or the TMS320C40 in a multiprocessor configuration with up to four processors. Since the announcement of the TMS320C80 this new generation multiprocessor seems most suitable as features like real-time MPEG decompression fit to our system very well. The geometry stage and the rendering stage are decoupled by a FIFO memory that buffers the triangle data sets necessary for the rendering module. Typical scenes normally consist of very different sized triangles [5]. The FIFO prevents the geometry module from waiting for the rendering device during the processing of big triangles. Vice versa, the rendering module can still render if there are data in the FIFO although the geometry module produces small triangles that take longer to calculate than to render.

4.2.2 Rendering Unit

The main component of the rendering module is an ASIC that does the rasterizing of the triangles and the interpolation of the texture pixels according to the algorithms described above. Moreover it is a rendering and shading processor. It performs Gouraud shading, texture mapping as described above, alpha blending and antialiasing. It receives input and command data through the 32 bit input interface connected to the decoupling FIFO. The results of rasterization and mapping are written to the image buffer. The 32 bit image buffer is a double buffer and consists of Enhanced DRAMs or SRAMs. To achieve a sufficient memory bandwidth, the memories are two times interleaved.

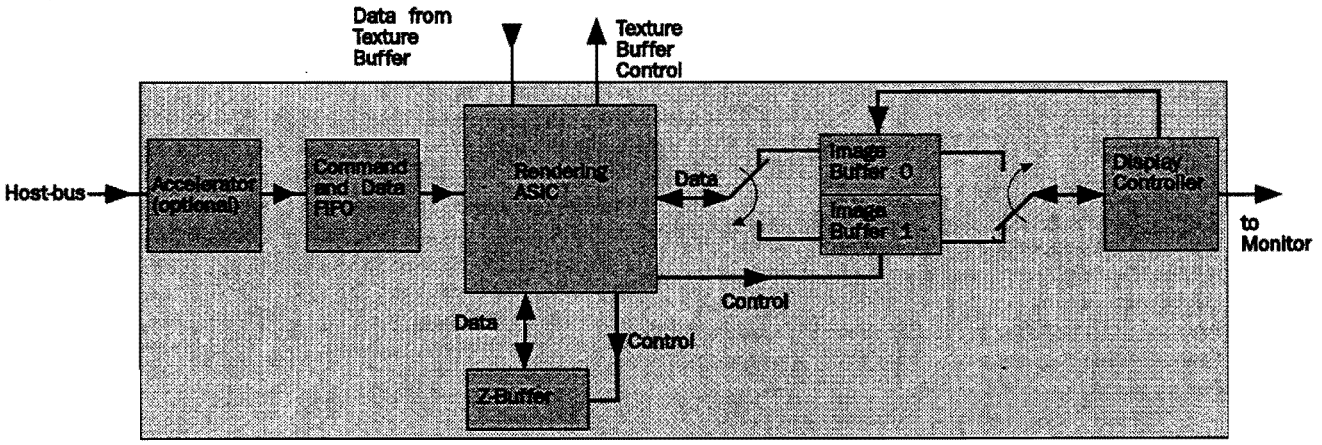

Figure 4: Architecture of the Graphics Module

When the area sampling algorithm is used for anti-aliasing, multiple area information has to be stored per pixel. After the generation of a complete frame, a postprocessing step is required to blend the different colors to the resulting color for the respective pixel. This blending is done by a blending ASIC during copying the frame to the display buffer. The z-buffer is organized similar to the image buffer. It contains 20 bits of z-information. 28 bits that logically belong to the image buffer contain status information, pointer to additional area information and memory space for accumulation. The texture buffer interface is two times 32 bit wide. The texture buffer can either consist of SRAM or of EDRAM. Because a random access is required, the SRAM is preferable but expensive. 16 Mwords of texture buffer can be addressed. For the video textures, a double buffer is required to decouple generation of the video MIPmap from the rendering device. As mentioned above, the textures can range from 8 bits to 32 bits of color. In the 8 bit mode, four values can be read in parallel. To increase performance for 32 bit modes, a preprocessor chip can be installed, which reads eight 32 bit values in parallel and blends them according to the tri-linear interpolation algorithm. The result is then read by the texture interface of the rendering chip.

The number of triangles, which can be processed is determined by the time for loading the initialization data. The actual limit is about 500,000 triangles per second. The rendering chip largely exploits parallelism and pipelining. In best case, it is able to produce one pixel per clock cycle that is 25 ns. A potential bottleneck is the memory access. The need to access more than one z-value or texture value per pixel will drop the performance. This design decision was a compromise between pin count of the rendering chip and performance. To increase system rendering performance, provisions have been taken, to provide support for parallelism on rendering chip level. The rendering chips can be configured to work in an interleaved fashion, each chip producing only certain lines of the resulting image. This image space partitioning requires no overhead in the geometry calculations. Furthermore, as each rendering chip works upon its own local part of the z- and image buffer no memory conflicts occur as with object space partitioning approaches. One drawback is the fact, that each rendering chip requires an own complete copy of the texture buffer because there is no fixed association between texture and image rows. As long as the triangles, that are to be rendered are not too flat, a linear speed-up can be reached. The system can be configured to employ up to 16 rendering chips for one display channel. With such a highly parallel system, a performance of 640 Mpixels per second is possible.

4.2.3 Display Controller Unit

The display controller unit handles the screen refresh, provides standard 2D-graphics functionality and manages the interfacing to a windows environment. We chose a TMS34020 graphics processor as main component. It combines ease of use with high performance and functionality. The host interface of the processor is connected to the AT bus. The local memory interface is connected to two banks of 1K by 1K by 24 bit VRAM display buffer. The data from the image buffer of the rendering unit are transferred to the display buffer using the serial ports of the VRAMs. The processor is provided with 4 Mbytes of DRAM for code and data storage. The controller runs a standard TIGA graphics interface to support MS-Windows and other applications that make use of this high level interface.

5 State of Realization

The video module is completely implemented as evaluatior board and tested. A user interface for the system, under MS Windows, is under development. An integration of tempora filtering controller, interpolator and an integration of the MIPmap controller as ASIC is planned.

Geometry accelerator and display controller are available anc used for code development.

The functional and interface specifications for the rendering ASIC have been finished. The design has been described using VHDL. Simulation and synthesis using the Synopsys VHDL simulator and silicon compiler are on the way. The expected gate count is about 500.000 gate equivalents. For the realization, a 0.6m CMOS standard cell process will be used. First silicon will be available in March 95. We expect a prototype of the whole system running in May 95.

6 References

[1] Ackermann, H.-J., Hornung, Ch.: The Triangle Shading Engine. In R.L. Grimsdale, A. Kaufman, (Eds.): *Advances in Computer Graphics Hardware V*, Springer-Verlag, Berlin, 1991, p.3-13.

[2] Ackermann, H.-J., Hornung, Ch.: An Architecture for a High Performance Rendering Engine. In A. Kaufman, (Ed.): *Rendering, Visualization and Rasterization Hardware*, Springer-Verlag, Berlin, 1993, p.3-13.

[3] Heckbert, P. S.: Survey of Texture Mapping. *Computer Graphics & Applications*, 6(11), November 1986, pp. 56-67.

[4] Jäger, M., Osterfeld, U., Ackermann, H.-J., Hornung, C.: Building a Multimedia ISDN PC, *Computer Graphics and Application*, 13(5), September 1993, pp. 24-33.

[5] Selzer, H.: Dynamic Load Balancing within a High Performance Graphics System. In A. Kaufman, (Ed.): *Rendering, Visualization and Rasterization Hardware*, Springer-Verlag, Berlin, 1993, p.37-53.

[6] Williams, L,: Pyramidal Parametrics. *Computer Graphics*, Proc. SIGGRAPH'83, 17(3), July 1983, pp. 1-11.