

New 3D display: Algorithms and real time architecture

Benoit KAUFMANN and Mohamed AKIL

ESIEE, Lab. A^2SI ,
2 bd Blaise Pascal, BP99,
93162 Noisy-le-grand, France
<http://www.esiee.fr/~info/a2si/>

Abstract— Three dimensional displays are now commonly used in various domains. But actually, 3D displays do not allow to see objects as if they were real and easily change the observer's point of view. This article will present a now developing 3D display that allows the observers to move according to the displayed objects, without new computing of the displayed pictures. The display will be compatible with any existing 2D and 3D softwares and will allow a large number of simultaneous observers, each of which has his own point of view to the objects.

I. INTRODUCTION

3D imaging is more and more used in various domains. These applications can display different real or virtual data types. Most of the time, the 3D objects are projected onto a screen that can only display 2D graphics. The third dimension is only simulated by the computer using shading, perspective, or else.

To allow the observer to perceive the objects in three dimensions, various systems exist, but none of them are perfect and present various disadvantages.

We are currently working on the implementation of a new display system that allows the observers to look at the represented objects in a very natural way, as if the objects were physically present in front of them.

This article will present a short state of the art of which 3D display systems usable with computers currently exist. Then it will present the principle of the display system we are currently implementing, the architecture we are planing to use for it and the implementation problems we had to solve and the way we solved them.

II. NEW 3D DISPLAY SYSTEM BASED UPON AUTOSTEREOSCOPY

Since Charles Wheatstone, who first described the stereovision principle and a way to use it in a three-dimensional display [10] [11], many three-dimensional systems have been conceived. These systems can be divided into two main families:

- the *stereoscopic vision*-based systems (also called “stereoscopic displays”) that display at least two pictures in a way that each eye of the observer perceives a different one. These display systems compute several images representing objects from different points of view.

These images are then displayed in a way that each eye of the observer sees its own image, different from the one of the other eye. With the difference of the pictures seen by the eyes, the perceived image is in three dimensions. The main differences between the stereoscopic display systems are in the way the different images are shown to an eye and covered to the other: mirrors [10] [11], lenses [1], colored or polarizing filters, mechanical or electronic shutters, etc.

- the “*projection*” systems that make the three-dimensional objects appear as if they were floating in the air, projecting images onto different or onto a rotating semi-transparent screen [5] [8].

Most of the existing 3D display systems use stereoscopy because it offers the best image quality, but many of them need the use of special glasses. So a new principle has been found out to clear up this disadvantage. This principle has been called “autostereoscopy”.

Autostereoscopic display systems project different pictures to different directions. Each eye of the observer perceives a different picture according to its location. So the observer can perceive the 3D objects without need of any additional stuff like glasses: he or she just have to look at the screen. The other main advantage of these systems is that it is possible to increase the number of projected images. The observer can see the displayed objects from more different angles.

The existing autostereoscopic display systems can use various ways to project images to the correct direction by using:

- a “classical” 2D screen (CRT, LCD or else) and vertical lenses to project different pixel columns in one direction [9].
- an LCD display and a partially obscured back light which makes the different columns of the screen visible only from a particular direction. [4]
- two different 2D displays and a moving mirror. A computer analyses the location of the observer and moves the mirror to make each eye of the observer to see a different display. [3]
- different 2D displays and making the observer see them through a lens that makes each of his eyes to see only the display corresponding to its location.
- a fast display that shows the images corresponding to

the different directions and projects these pictures to the correct direction with a shutter or a moving device. [7]

We can resume the previous state of the art by the following array:

	Stereoscopic based systems	“projection” systems	Autostereoscopic systems
Observers don’t need to wear glasses	-	+	+
Multiple simultaneous observers	-	+	+
Different points of view	-	+	+
Infinite observation volume	+	-	+
Objects can be opaque	+	-	+

Our project is an improvement of stereoscopic display systems. It is based on the observation that, in real life, you see an object because each of its points emits or reflects light to every direction and your eyes receive this light. The idea is to reconstitute all the light rays that each object emits. In fact, we do not reconstitute the whole object visible from any direction, but the object as seen through a virtual screen. So we only have to reconstruct the light rays emitted by the object which pass through our display system (see figure 1). Basically the

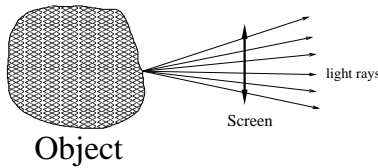


Fig. 1. Reconstructed light rays

system projects a large number of different images in many directions. The observer will be able to change its point of view just moving around the display system: he will be able to move forward, backward and around the displayed objects as if they were physically present in front of him, without any interaction between the observer location and the software. So the number of possible simultaneous observers is virtually infinite.

In order to implement such device, we will need a screen of which every single voxel is able to project different light rays in different directions. The picture perceived by the observer will then only depend on his location. His two eyes being at different locations, they will see two different pictures, so the observer will see in 3D and will be able to move according to the displayed objects, as shown on figure 2.

This will be possible by using a very fast display system and time-multiplexing the different pictures in order to project them to the appropriate direction.

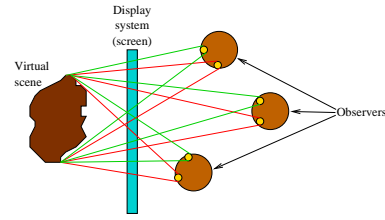


Fig. 2. Characteristics of the 3D display model

III. 3D IMAGE PROCESSING

To control and make this device working, we need to generate appropriate images. These images will be generated from 3D data sent by a computer to the 3D image processing chain. This processing chain (represented on figure 3) will compute all these data and generate the pictures that will be displayed.

It is very similar to the OpenGL processing chain: First of all, the data are introduced into the pipeline from the host computer through a communication device. They are separated according to their nature and sent to the correct pipeline. The pipeline we are interested in is the 3D pipeline. But there are two other ones:

- one of them processes the 2D data: it acts like a 2D video card and stores the 2D data that can be useful like window frames and titles, pop-up menus, etc. The stored pixels are then turned into voxels and included into the 3D scene as planes that are parallel to the screen.
- the second one controls the device: setting the resolution of the screen, setting the processing chain parameters, etc.

3D data are stored in a buffer and then sent to the first step of the 3D pipeline: geometry. This step computes the projection of the points that compose the 3D object to be displayed onto the screen. Its result is the X and Y coordinates of the pixel where each point must be displayed.

The coordinates of the pixels are then computed by the rasterization step. This step transforms the coordinates of the points of the facets into a set of voxels composing it and determines the color of each of them, according to the texture of the object, light sources, shadows, etc.

At last, 3D and 2D images are merged to be displayed by the “voxels processing” step and stored in the frame buffer. If two or more pixels have to be displayed at the same location of the screen, the closest is chosen as done by the Z-buffer algorithm [2]. Then, the display control copies the stored pictures to the display at the appropriate time.

But the size of the needed memory, due to the number of different projection angles is very large. Until now, the 3D systems displayed between 2 and 16 pictures at the same time. In that case, it is possible to memorize all the images in a buffer. But our system allows to display a very large number of different views. So the size of the buffer needed to memorize all the pictures grows very fast. As an example, with full color images, each pixel uses 24 bits. If we consider an

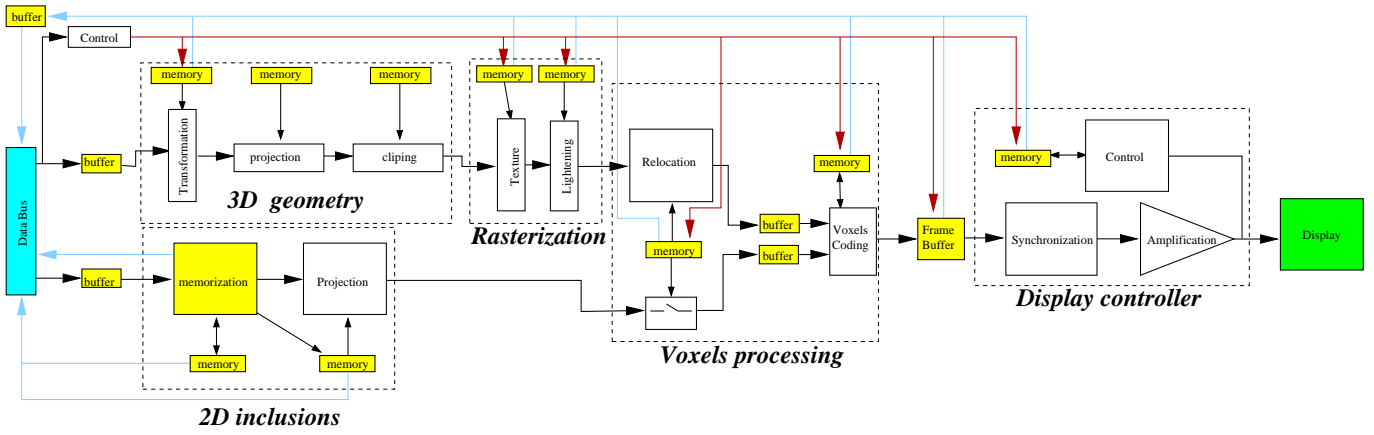


Fig. 3. 3D image processing chain

image of $1,024 \times 768 \times 24$ bits, and want to have 20 different angles of observation, we will need $1,024 \times 768 \times 3 \times 20 = 47,185,920$ bytes ≈ 45 Megabytes for only one 3D picture. This problem has been resolved by an algorithm that stores the 3D data as voxels without repetition and can reconstruct the pictures of all the different directions in real time [6]. What is considered to be real time here is at least 30 frames per second projected in up to 200 different directions, which correspond to $30 \times 200 = 6,000$ images per second.

The image processing pipeline must be adapted to this algorithm:

- the depth of each voxel has to be discretized, so each voxel has its real and its discretized depths.
- the “voxels processing” step of the processing chain cannot just use the z-buffer algorithm, but keeps all the voxels (from 2D and 3D pipelines) and codes them into the frame buffer with the above algorithm.
- the frame buffer does not contain 2D images anymore, so the display controller must decode it in order to reconstitute the displayed pictures.

IV. HARDWARE IMPLEMENTATION OF THE PROCESSING CHAIN

A. Processing

In order to reach a speed suitable for real-time display (up to 30 3D frames per second projected to 200 different directions, which means 6,000 projected images per second), the use of a dedicated hardware is necessary. This hardware will act exactly as a 3D accelerated video card.

The whole processing chain will be controlled by the host computer through the control stage. This control is similar to a memory access: the host will simply read and write three different types of memory data. These types are:

- hardware informations. This type of memory is only readable by the host, not writable. It informs the host about the type of screen (i.e. plug-and-play support) is connected, the size of the frame buffer, the valid screen resolutions, etc.

- registers of the processing chain. These registers modify the comportment of the different stages or the rendering parameters. Typically, these values determines the resolution of the output screen, the refreshment frequency, the depth of the pixels, etc.
- input data. This type of memory allows the access to some of the working memories: input memory, where data are read before processing, result memory where computed data are stored for displaying and the 2D memory where 2D image is stored, which can be accessed for reading and writing. The input memory contains data like transformation matrix of the virtual camera, locations and color of the light sources in the virtual scene, textures to be applied, etc. The result memory can be accessed for reading the result of the computation in the frame buffer or for writing a previously read frame buffer into the hardware frame buffer to display it.

The processing chain will receive informations to be displayed from the host computer from the communication channel (in the far left in figure 3) which will typically be an AGP or PCI bus interface. This sort of communication bus allows a stream up to 2.1 Giga-Bytes (in 8X AGP 3.0 mode) per second, but the most used is 536 Mega-Bytes (in 2X AGP 2.0 mode) per second. This communication channel must provide a fast data transfer in order to allow the fast transfer of all data needed to compute and display the 3D images (objects coordinates, textures, etc.)

The data send to the processing chain (2D and 3D pipelines) are 32 bits. The speed of the whole processing chain should be as fast as the communication channel, but its speed will be limited by the speed of the “voxels processing” stage, depending on the implementation of its algorithm and the complexity of the images.

As described above, the two first steps of the 3D pipeline (3D geometry and rasterization) are hardly the same as those in the existing commercial 3D accelerated cards. The 3D geometry is the Geometry engine that is present in every 3D graphics accelerator card. It is schematically divided into three

main parts:

- The *transformation* stage which computes the projections of the 3D coordinates of the points forming the facets of the objects. Those points are projected onto the virtual plane that represents the virtual camera.
- The *lightning* stage which computes the illumination of every facet, according to its position inside the virtual scene, the normal vector of each facet and the position of the different light sources of the scene.
- The *clipping* stage which controls if the projected points will be displayed inside or outside the screen and modifies the facet if it is partially visible to make it properly displayed.

These three stages are exactly the same as the ones which are actually used in any hardware 3D graphics accelerator. The only difference is the width of the virtual screen: it is virtually very much larger, to provide the observer to move around the displayed objects and to see properly on the sides of the screen. But this modification can be very easily done by modifying the parameters of the clipping circuit or by modifying the projection matrix of the virtual camera, by scaling the X coordinate. The second stage of the 3D pipeline, the rasterization, is quite identical to the rendering engine that is present in every 3D graphics accelerator card. The only difference is the use of the Z coordinate of the generated pixels which is scaled and kept with the X and Y coordinates to the end of the pipeline, without any Z-buffer. The voxels with their three coordinates (X, Y and Z) are then computed by the voxel processing stage.

In parallel with the 3D pipeline, between the communication bus and the voxel processing, there is another pipeline which assume to memorize and compute all the 2D data to be displayed. The first stage of this pipeline can be considered as a 2D frame buffer which stores the 2D data from the software (windows frames, menus, icons, wallpapers, etc.). This part is seen by the software as a “classical” video interface and all the 2D pixels can be stored directly into this buffer as if it was the actual final frame buffer which reflects what is displayed onto the screen. The second part will convert the pixels into voxels suitable for the 3D display. This conversion can be computed for each pixel when it is stored into the 2D buffer, or in only one operation that will convert all the stored pixels. This second method has a better complexity because the pixels must be merged into patterns to be displayed, so coding an already made pattern is faster than coding separated pixels. This depends on the way the pixels are stored into the 2D frame buffer. In that case, this part of the processing chain can be simply implemented by the next stage of the 2D and the 3D pipeline: the voxel processing.

This stage merges the data from the 2D and the 3D pipelines and codes them as described above to the 3D frame buffer. The coding algorithm [6] is based on the principle of coding the voxels as patterns located at a particular line (of the screen) and depth. The data will be stored in the frame buffer the following way:

For each line of the screen

- Number of used depth (where voxels are present)
- For each depth of this line
 - Depth (signed distance between the screen and the plan)
 - Number of the patterns present in this depth
 - For each pattern
 - Horizontal position (from the left limit of the visible area) of the pattern
 - Number of the voxels present in this pattern
 - For each voxel of the pattern
 - ARGB component of the voxel
 - end For
 - end For
- end For

end For

The received voxels are inserted in the correct line and the correct depth, according to their Y and Z coordinates. Then, their X coordinate determines which pattern they must be inserted into (if it exists) or where a new pattern must be created. After that, a test must be done to check if the modified pattern must be merged with another one. There are two main ways to implement the corresponding coding algorithm. The first one uses chained lists to store depths and patterns. It has a time complexity in $o(n \times m)$ (where n is the average number of patterns in each line of the frame buffer and m is the average size of the patterns) and a size need (for the working memory) of $o(n \times m \times 24 \text{ bits} + n \times m \times l \times s)$ (where n is the number of patterns in all the frame buffer, m is the average size of the patterns, l is the number of patterns plus number of depths for all the lines and s is the size of a pointer). You can also code the voxels inside a patterns as a chained list to fasten the coding, but it increases the needed size. The second one uses directly the final coding. The size of the needed memory is limited to the size of the final frame buffer but the complexity is $o(S^2)$, where S is the size of the frame buffer. This is a simplified version of the algorithm that is defined here. The full version includes a post-processing which erases voxels hidden by other ones and a mechanism to avoid frame buffer overflows. The needed size of the final frame buffer is not determined and does not only depend on the quality of the displayed pictures, but it also depends on the complexity of the scene to be rendered. Its size must be, at least, the size of the 2D frame buffer for a 2D image of the same resolution. For example, a picture of $1,024 \times 768 \times 24$ bits/pixel will take about 2.3 MB (MB = Mega Bytes, $1 \text{ MB} = 8 \times 2^{20}$ bits). So this frame buffer will be of a size of about 10 MB, at least. The voxel processing must store data at $10 \text{ MB} \times 30$ images per second = 300 MB per second at least to assume real time imaging.

B. Displaying

After the data have been stored into the 3D frame buffer, they must be read and decoded once for each picture displayed in every direction.

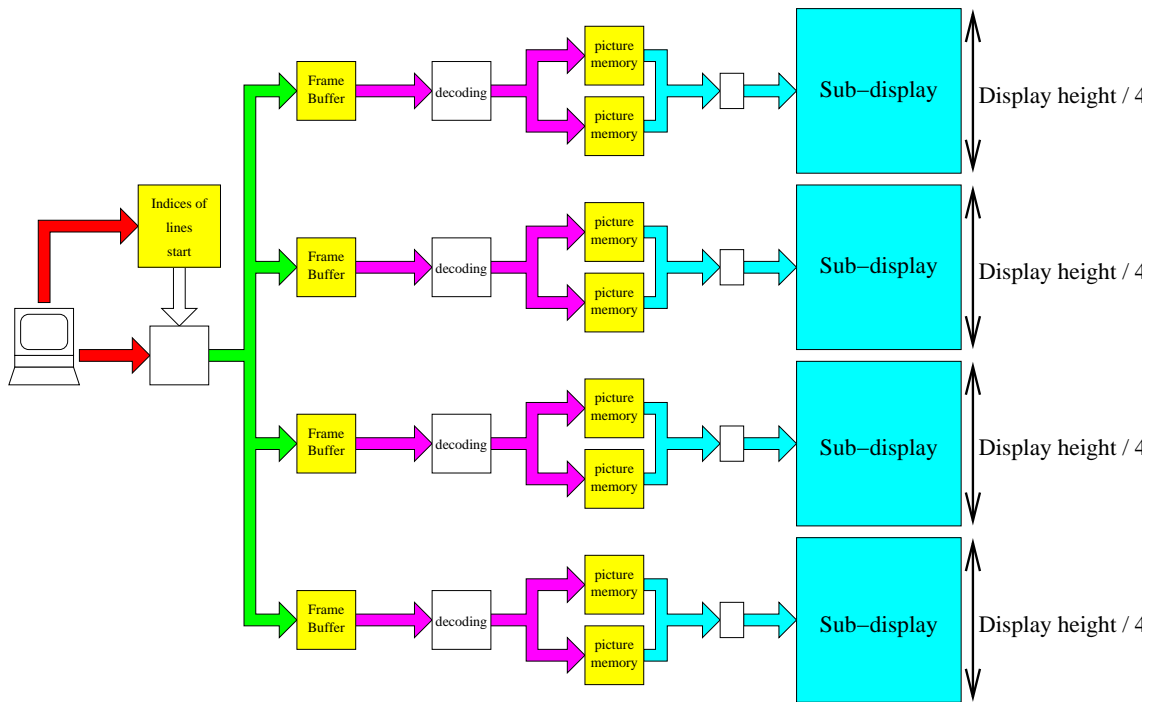


Fig. 5. Frame buffer decoding parallelization example

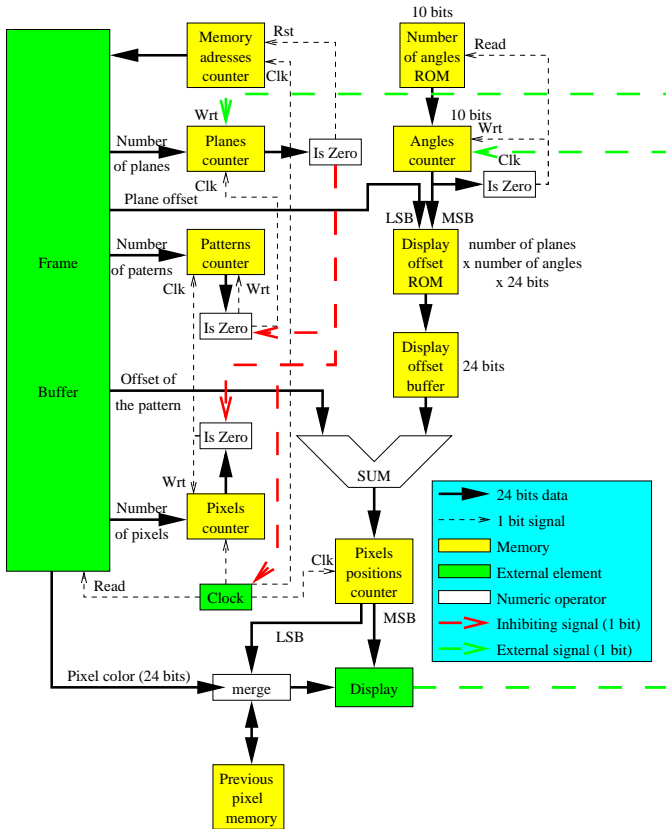


Fig. 4. Frame buffer decoding architecture

Because such great speeds were needed, the decoding algorithm has been made so easy to implement into specific hardware: it can be operated by only a few simple electronic components, as shown on figure 4, which represents the decoder for one of the lines of the display. A quick anti-aliasing has also been added to improve the quality of the displayed pictures. This architecture is composed of three counters that implement three imbricated *for* loops: planes, patterns and pixels. For each pixel, the location in which it must be copied into the display buffer is calculated from the depth of this pixel and the angle into which the current picture will be projected. First, the display sends a signal that initializes the planes counter, setting it to zero. This resets the memory addresses counter to the beginning of the frame buffer and starts the decoding. At each clock signal, a 32 bits value is read inside the frame buffer and the memory addresses counter is incremented (after the value has been read) to be ready to read the next value. The read value is stored inside the planes counter. Those planes are the set of voxels of a line which are at the same depth of the screen, i.e. which have the same Z coordinate. If the first value is not zero, the associated “is zero” comparator unlocks the patterns counter so the next read value will be stored inside this counter. If it is zero (the line is empty), the decoding is blocked on the first value until the line contains voxels to be displayed. After the patterns counter have been set (by a value that cannot be zero), the offset of the first plane is memorized and the display offset is computed and stored in the display offset buffer. This value is the signed X coordinate of the voxel located at position zero in the plane. Then, the pixel counter is unlocked and stores

the next value read in the frame buffer. At the following clock beat, the offset of the first pattern is read and the offset of the first pixel of the pattern is computed: the offset of the pattern is added to the display offset stored in the display offset buffer and the result is stored in the pixels positions buffer. For the next clock cycles, the values read in the frame buffer are treated as pixels: they are merged with the previous pixel for anti aliasing, according to the less significant bits of the pixel position which determines the percentage of each pixel (new and previous) that must be displayed, and the new pixel (as it was before the merging) is stored to be merged with the next pixel. The result of the merging is then send to the display memory and will be merged with the previous color (background or pixel in a further plane), according to its alpha channel value. Then, the pixels counter and the pixels positions buffer are decremented. If the pixels counter is not zero, we continue for the next voxel. If it is zero, we decrement the patterns counter and, if it is not zero, we store the next pattern offset and size. If the patterns counter is zero, the planes counter is decremented and, if the planes counter is not zero, the decoding continues on the next plane. If the planes counter is zero, the decoding is stopped until the next signal of the display to start the decoding for the next angle of view.

For real time imaging (at least 30 frames per second and 200 projection angles), the number of pictures to be displayed per second is $30 \times 200 = 6,000$. So the frame buffer must be able to be read at least 6,000 times per second by the display control. Because the size of the frame buffer is about 10 MB, this corresponds to a flow speed of $6,000 \times 10 \text{ MB} = 60,000 \text{ MB}$ per second. Because each value read is a 32 bits value, the clock should be about 15 GHz. The display controller must also be able to decode and send $1,000 \times 6,000 = 6,000,000$ pixels per second to the display. This corresponds to a flow speed of 6 MB per second. The very high number of pictures to be computed in a very short time brings up the problem of the speed of the decoding. We proposed the solution of heavily parallelizing the decoding process. There is two ways of doing this:

- We can parallelize the decoding of lines: several separated decoders compute a limited number of lines. This can be done easily because this decoding algorithm decodes the display lines independently, with a reduced need of memory. Figure 5 shows an example of such a parallelization of the decoding stage.
- We can also decode simultaneously the pictures to be projected to different directions, which is quite similar to the previous method.

The parallelizing of the decoding permits to divide the data flow and the clock rate of the decoder by the number of parallel decoders, to reach more realistic values: If we take 10 decoders, the data flow is reduced to 600 KiloBytes per second and the decoder clock rate to 1.5 GHz. If we take 20 decoders, it will be 300 KiloBytes per second and 750 MHz, respectively.

V. CONCLUSION AND PERSPECTIVES

We have presented in this article the principle of the 3D display system we have proposed, which consists of projecting different light rays in determined directions. This system has the main advantage of displaying 3D virtual objects as if they were physically present inside the device, including the possibility of moving according to the system to change the observer's point of view, and allowing several simultaneous observers from several locations, without any software interaction. This new display system will allow anybody to display any 3D object in a very realistic way and allows him to quickly change his point of view without needing to manipulate any peripherals (like mice or keyboards) to do it. The displayed objects will be very easy to manipulate and the prehension of the volumetric form of each of them will be increased. Such a display is particularly adapted to display virtual endoscopy pictures, molecules representations or any type of virtual reality.

We proposed an architecture of the processing chain which satisfies the real time constraint: 30 frames per second with 200 direction of projection = 6,000 pictures per second. We proposed different prototypes of 3D display systems that we are actually evaluating. One of them was based on an MMD and the other based on the use of an analogic memory bench (of which a prototype is actually being manufactured). We are also simulating the architecture of the whole image processing chain using systemC.

REFERENCES

- [1] Sir David Brewster. *The Stereoscope*. 1856.
- [2] Edwin E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Dept. of CS, U. of Utah, December 1974.
- [3] Basil A. Omar Nicolas S. Holliman Jonathan Harrold David Ezra, Graham J. Woodgate and Larry S. Shapiro. New autostereoscopic display system. In *IS&T/SPIE Symposium on Electronic Imaging Science and Technology*.
- [4] Sandin; Daniel J. et. al. Computer-generated barrier-strip autostereography. In *Proc SPIE, Non-Holographic True 3D Display Technologies, 1083*, pages 65–75, Jan. 1989.
- [5] E. Rieper K. Oltmann D. Bahr K. Langhans, C. Guill. Solid felix: A static volume 3d-laser display. *Stereoscopic Displays and Applications XIV, Proceedings of SPIE, Volume 5006*, Santa Clara, CA, 2003.
- [6] Benoit Kaufmann and Mohamed Akil. Video memory compression for multi-view auto-stereoscopic displays. EI 5291A-7, Proceedings of Steroscopic Displays and Applications XV, SPIE 16th annual symposium, San Jose, California, january 2004.
- [7] S. R. Lang G. Martin N. A. Dodgson, J. R. Moore and P. Canepa. A 50" time-multiplexed autostereoscopic display. In *SPIE 3957, SPIE Symposium on Stereoscopic Displays and Applications XI*, 23rd-28th Jan 2000, San Jose, California.
- [8] Actuality systems. web site. <http://www.actuality-systems.com/>.
- [9] C van Berkel and J H A Schmitz. Multiuser 3d displays. presented at Tile 2000, London 10-11 May 2000.
- [10] Charles Wheatstone. "contributions to the physiology of vision. part 1. on some remarkable, and hitherto unobserved, phenomena of binocular vision". pages 371–394. Royal Society of London Philosophical Transactions 128, 1838.
- [11] Charles Wheatstone. "contributions to the physiology of vision. part 2. on some remarkable, and hitherto unobserved, phenomena of binocular vision (continued)". *The London, Edinburgh, and Dublin Philisophical Magazine and Journal of Science, series 4, 3*, pages 504–523, 1838.