Hardware accelerator for self adaptive Augmented Reality systems

Tarek FRIKHA, Nader BENAMOR, Kais LOUKIL,
Agnes Ghorbel, Mohamed ABID
CES-Laboratory
Sfax SUD University, National Engineering School of Sfax
Sfax TUNISIA
tarek.frikha@gmail.com

Jean-Philippe DIGUET
Lab-STICC
University Bretagne Sud,
Lorient, FRANCE
jean-philippe.diguet@univ-ubs.fr

***Abstract*: The emergency of multimedia applications particularly in mobile embedded systems puts new challenges for the design of such systems. The major difficulty is the embedded system's reduced energy and computational resources that must be carefully used to execute complex application often in unpredictable environments. So the system architecture must be energy efficient and flexible enough to adapt resources to application requirements to manage the environment architectures and mobile's constraints. The augmented reality is a very promising 3D embedded multimedia application. It's based on the addition of specific 3D's animations on a video flow. In this paper, we describe our concept of flexible architecture and we give implementation results based on Pixel Shader Accelerator. This is the first step of the project and we compare various hardware and software implementation.**

I. INTRODUCTION

The multimedia embedded applications inflate the computer sciences domain. Watching a HD video or a 3D movie is now possible not only with a 3D TV but also possible on small portable systems such as smartphone and tablets.

The design of such systems faces new challenges due to the limited available resources and the external environment fluctuations such as noise, bandwidth fluctuations, available energy…)

To tackle those problems, adaptive systems are a promising solution. Those systems can adjust the used hardware resources according to the application requirement and the environment state. Algorithmic adaptations (based on applications parameter and algorithm tuning) can also be used for energy saving purpose or QoS adaptation.

In this paper, we present preliminary results of the design of an adaptive embedded system based on reconfigurable HW variable units dedicated to augmented reality applications.

AR technique consists to enhance real video sequences with virtual objects. [1] The AR touches many fields such as : medicine (3D organs modeling…), military (Head-Up Display), industrial (total immersion, remote maintenance [2]), marketing and commercial (advertisements, virtual visits…), entertainments (video games and sport events (player numbers, offside virtual lines, WR comparison line, give visual information for TV viewers from hidden angles in sport match [2] …). [4]

Our target AR application (see Figure 1) is the combination of a video flow recorded with a camera and images synthesis.

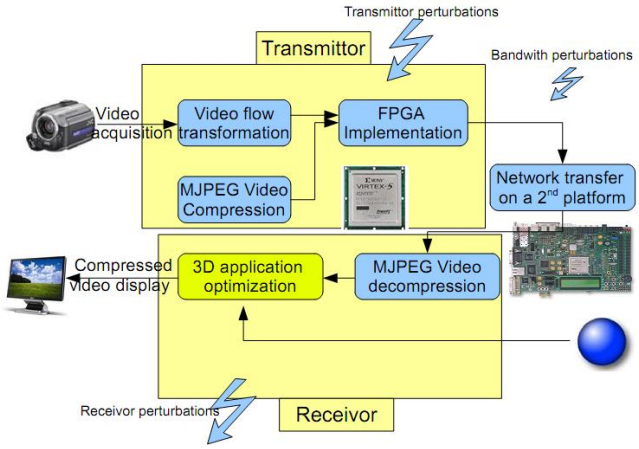

Figure 1: Application demonstrator

Our demonstrator is composed by two parts: a transmitter and a receiver. A camera is used for video acquisition. This camera transmits a video flow to the transmitter. The transmitter is composed of a video flow transformation bloc and a MJPEG coder (embedded in a first ML 507 FPGA board) which is used to compress the video. The 3D animations specifications are multiplexed with the encoded video. They are sent over the TCP IP network using an XML file. At the reception, the video is decoded; 3D animations are computed using XML specifications and mixed with the decoded file.

The figure 2 reperesents the 3D adaptation technique. According to the 3D object characteristics we add the appropriate hardware blocs. The final data are saved on a memory blocs.

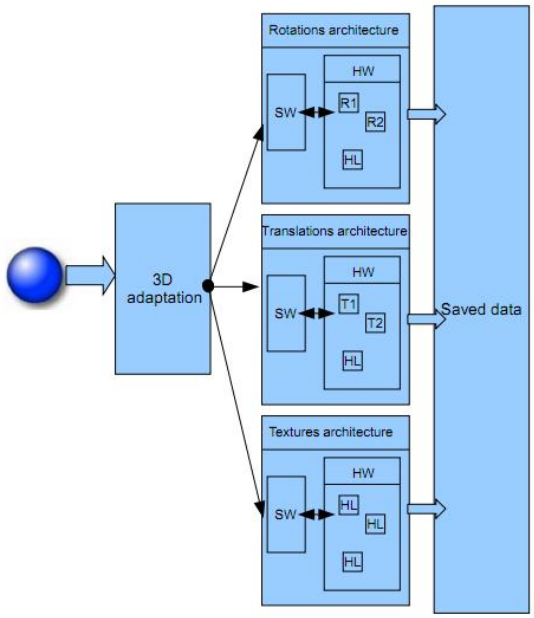

Figure 2: Adaptation technique
SW : Software, HW : Hardware, T Translation, R Rotation, HL: Hline

The paper is organized as follows. Section II gives our work major features and compares it with related works. Section III presents our application design and details our Hardware accelerator. Section IV shows the implementation and the obtained results. Finally, section V concludes the paper with a brief outlook on future works.

II. AUGMENTED REALITY AND USED ARCHITECTURES

The advances in the field of computer vision and mobile computing have made possible the development of complex but one of the main issues remains outdoor application in unknown environment. Applications become more complex, and the environment conditions are unpredictable (sunlight, unrestricted mobility, etc.) and where different types of sensors can be used. [3] In this paper we'll talk about the 3D application implementation.

Used architectures:

3D computation requires high performance architecture. GPP are greedy. A solution would be to adopt computing units (shader, geometric).

To work with complex 3D applications many GPUs are used. This GPUs architecture evolution increased the last years. ATI and NVIDIA leaders of GPUs used different architectures to display 3D images[5].

The architectures presented are based on geometry shaders for different movements (translation, rotation …) and vertex shaders (image textures…). The first architectures consist on using one of each one to do different computation. After that, they increased the number of the shaders and make it work in parallel.

The number and type of shaders used depends on the type of images and objects. With a richly textured video, many vertex shaders are required with moving video, geometry shaders. GPU use a unified shader to display the video. This architecture is an optimal one for the process using but need many resources.

In the next part, we present an alternative solution based on reconfigurable architecture for 3D objects displaying. Our job is oriented to add the 3D object to the embedded video. We need to have a trade-off between the video quality display and the limited FPGA resources.

III. APPLICATION DESIGN AND HARDWARE ACCELERATOR:

A. 3D image synthesis application overview:.

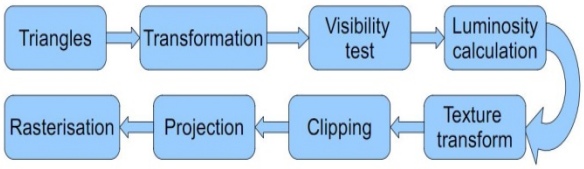

Figure 3: Graphical 3D pipeline

The triangles represent the input of our 3D graphical pipeline (figure 3). The transformation step represents the conversion from local coordinate system to a global one, which is the camera coordinate system. We'll use translations, rotations and homoteties to obtain the final result.

The visibility test consists in identifying which pixel will be viewed and which one will be hidden on the screen using the angle between the vision vector and the hidden one.

The luminosity calculation step gives the luminous intensity attributed to each pixel.

The clipping step consists in eliminating the pixel which will not be on the projected screen but on the computer monitor: if the pixel is a hidden one, it is not displayed.

The projection step is the application of the projective geometry which consists on how displaying a 3D point on a 2D scene.

The rasterisation step is very important because it gives the projected 2D objet a 3D visual aspect when it is projected on the screen. Because of the complexity of the 3D application, we accelerate this software application by introducing hardware blocks. Inspired by the GPP architecture, the software bloc communicates with hardware blocks with the FSL bus.

B. Application analysis and profiling

We use a 3D application available as a C code. In this application the object rotates around different axis. Due to its complexity, the software application version can be displayed but are so slow. We need to accelerate the application by creating hardware blocks replacing the heaviest 3D application functions. To know which functions must be transformed on a hardware block, we profile the native C code via the profiling tool of the Nios II embedded processor on Altera Platform.

The table 1 represents the 3D application profiling.

Table 1: 3D's function application profiling

Functions	Time percentage
Hline (Pixel Shader)	69%
Rotation (Geometry Shader)	13%
Scale	5%
Translation (Geometry Shader)	5%
All other functions	< 2%

The profiling result gives us that the polygon filling takes the most important part of 3D application. The 3D application is based on a rotation around an only one axis.

69% of the application time was dedicated to the Hline function which is oriented to fill in each pixel the attributed color value. Each pixel of the triangle contains a value which is an integer that belongs to [0,255]. We must assign the appropriate value to each pixel. However, one may keep in mind that this profiling depends on the benchmark.

All the other application function didn't use less than 13% and that's why we choose the Hline to accelerate it.

The Hline function can be called pixel shading whereas the rotation and translations functions represents the geometry's one.

Our 3D object is formed by a set of triangles. Each triangle is filled using horizontal lines formed with pixels. The Hline function attributes to the pixels of each horizontal line of each triangle the color value according to the Gouraud shading algorithm.

C. Hline hardware accelerator architecture:

The implementation of the hardware accelerator aims to speed up a low frequency low cost architecture to display the 3D object moving on the screen without being heavy.

Our study and implementation is based on Virtex that can be dynamically and partially reconfigured.

Figure 4 represents the accelerator with more details

This accelerator contains two important stages:

- ✓Lines extremities determination
- ✓Pixel color filling.

*a) Segment's extremity determination **(SED)**:*

The line extremities determination is represented by two steps which are represented in the figure 4:

- ✓ Segment's extremity determination.
- ✓ Segment's pixel extremities filling

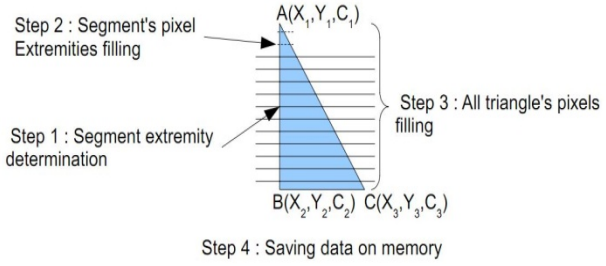

Figure 4: 3D triangle's pixel filling accelerator

Segment extremity determination consists on finding the x coordinate of the pixels which are the extremity of each triangle's line side.

Every triangle is formed by three sides. To fill the entire triangle we have to find the pixels which are on every triangle side. Since the triangle is displayed on the screen our landmark entity will be the pixel not only horizontally but also vertically.

To find the triangle segment's extremity we must find director coefficients of each triangle's side. Using the three top coordinates, we can calculate the director coefficient using the equations system:

$$y = ax + b \quad (1)$$

$$a = \frac{y_B - y_A}{x_B - x_A} \; et \; b = y_A - ax_A \quad (2)$$

Each pixel value will be incremented by the director's coefficient a. We skim from the minimum value of the abscise to the maximum one.

To obtain the coefficient a value, a division is required. This arithmetical operation is done by using the IP core generator of Xilinx.

*b) Segment's pixel filling**(SPF)**:*

Pixel color filling is also based on two steps:

- ✓All triangles pixels filling.
- ✓Saving data on memory.

The color value of each pixel obtained is the difference between the vertices colors divided by the difference between Y_B and Y_A as mentioned in equation (3):

$$c_{inc} = \frac{c_B - c_A}{y_B - y_A} \quad (3)$$

Once all triangles are found, all these values will be saved on the FPGA external dual port BRAM memory [8]. We use a dual port BRAM memory because we need to

save the SPF data on the memories. These data will be used as the input of the All triangle pixels fillings blocs.

Finally, we will focus on the accelerator which is the Pixel color filling.

c) All triangles pixels filling:

After filling the color value of the triangle's side, we'll find the value of the color of each pixel interior to our triangle. We'll use the same method used for the triangle extremities values.

Then, we'll save the pixels filling values on a BRAM bloc.

d) Saving data on the memory:

The last step of our accelerator will be to save the data obtained on a BRAM memory bloc.

This bloc generated by the Xilinx IP core generator contains also every pixel abscise value and the color that we'll affect to this pixel. These values will be used when we'll obtain the entire image to display on the screen after the entire treatment.

The figure 5 represents the entire 3D accelerator.

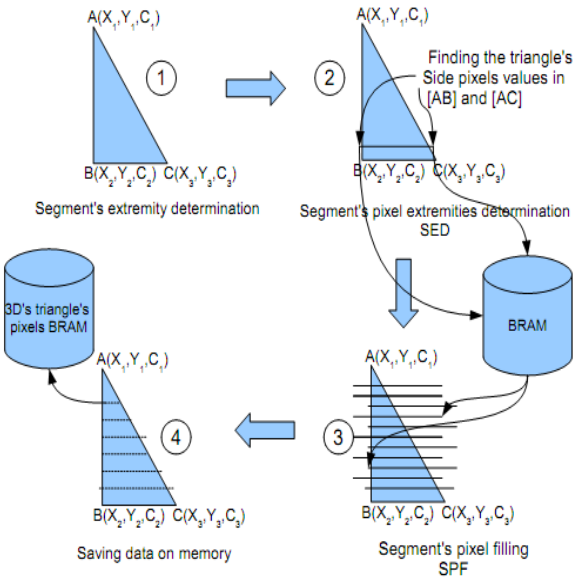

Figure 5: 3D triangle's pixel filling accelerator steps

Global flexible architecture :The whole project lies on a concept of self-adaptive architecture based on a softcore (Microblaze) [6] enhanced with a set of reconfigurable VHDL accelerators (see Fig 6). Based on various profiling we can observe that Geometry Computation and Pixel Shaders (Hline) accelerators must be considered. However the number of accelerator of each kind can be adapted according to application needs. The Hline accelerator integration is detailed in the next section.

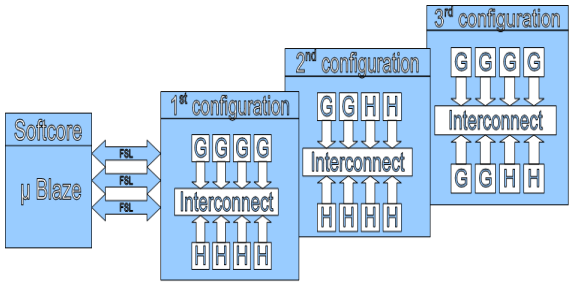

Figure 6: Global architecture: G: Geometry, H: Hline

D. Hline architecture :

The interface between Microblaze and accelerators is based on FSL bus, since FSL FIFOs are provide fast communication between hardware blocks and the processor.

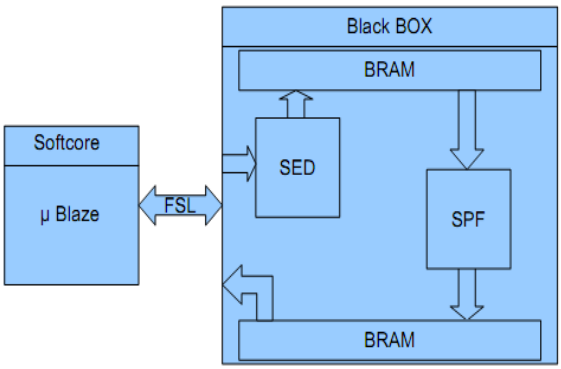

Figure 7: Hline architecture

Figure 7 represents the accelerator's architecture. The Microblaze, communicates with a black Box [7] containing the two VHDL functions which write data on BRAMs.

The microblaze communicates with Black box through a FSL bus. The inputs of the black box are processed by the SED module. The SED results are saved in the first BRAM block.

This data are the input of the SPF module which fills all the triangles color in the second BRAM block.

The resulted data are sent via the FSL to the Microblaze to end the process. This data are sent for the VGA controller to display it.

We test this architecture with the 3D application. We do the test with an only one hardware accelerator then with 4 accelerators in parallel.

Aiming the AR applications, we'll not project the 3D object on the screen but a smallest representation of it on the screen. The obtained results of a 50*80 3D object will be shown on the next part.

IV. IMPLEMENTATION AND EXPERIMENTATION RESULTS:

The test of pure software 3D object code and the mixed Software/Hardware one is exposed in the Table 2.

Table 2: Software / Hardware comparison

Version	Time (second)	Frames/second (fps)
Software	6.5	0.15
Mixed HW/ SW		
Using 1 HW Block	0.273	4
Using 4 HW Blocks	0.075	14

The hardware accelerator allows for a speed up of almost 24x. The use of multiple accelerators gives implementation almost 87 times faster than the software version. This improvement was possible due to the architectural parallelism. A frame rate of 14 fps is achieved with 4 HW blocks, with an acceleration of geometric functions and improvement of the current standard C software implementation the objective of 24fps will be obtained.

The FPGA occupation after the hardware implementation is described in the table 3. The use of 4 hardware blocks need 4 times more hard design use.

We increase the FPGA use to obtain a better result.

Table 3: Single Hline accelerator

Device utilization summery	Number	% ML 507
Slice Registers	5,790	12
Used memory	204	1
External Memory (kb)	1,080	20

V. CONCLUSION / PERSPECTIVES:

We have presented our concept of flexible architecture for AR systems and detailed the main accelerator dedicated to pixel shading. The second accelerator, dedicated to geometric operations will be presented, in a following paper. The combination of both accelerators will guaranty a 24fps frame rate with a 100MHz clock frequency. This is the first step of the project. But, as previously mentioned, resource requirements are strongly data dependent in 3D applications (geometry vs pixel shading, size and number of objects). So, the second step is the implementation of self-adaptation as a software function on the Microblaze that controls online the dynamic configuration of hardware accelerators, according to application needs.

REFERENCES

[1] Cemil Azizoglu, Ph. D, “High Performance Graphics on Android”, Khronos group, 2010

[2] B. Thomas, B. Close, J. Donoghue, J. Squires, P. De Bondi, M. Morris, and W. Piekarski. “ARQuake :an outdoor/indoor augmented reality first person application”. In The Fourth International Symposium on Wearable Computers, 2000.

[3] Gerhard Reitmayr, Tom W. Drummond “ Going out: Robust Model-based Tracking for Outdoor Augmented Reality”.

[4] Sturman, D.J. and Zeltzer, D., A survey of glove-based input, Computer Graphics and Applications, IEEE , V14 #1, Jan. 1994,30 -39

[5] D.Luebke, SIGGRAPH 2008,Beyond programmable shading in action“GPU Architercutre : Implications & Trends”. NVIDIA Coorporation 2007.

[6] K. Loukil, N. Ben Amor, M. Abid “ HW/SW Partitioning Approach on Reconfigurable Multimedia System l'Art ” on ChipInternational Journal of Engineering (IJE) 2011 avril 2011 volume5, Page 568..

[7] www.xilinx.com/support/documentation/sw_manuals/xilinx11/mb_ref_guide.pdf, MicroBlaze Processor Reference Guide.

[8] www.xilinx.com/support/documentation/ip_documentation/fsl_v20.pdf, LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c)