Computer Graphics and the MOLEN paradigm: a survey

Humberto Calderón and Stamatis Vassiliadis
Computer Engineering Laboratory
Faculty of Electrical Engineering Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2600 GA Delft, The Netherlands
Phone: +31 (15) 2783664 Fax: +31 (15) 2784898
E-mail: {H.Calderon|S.Vassiliadis}@ewi.tudelft.nl

Abstract— **Focusing in the advantages and drawbacks on the FPGA implementations vs. ASIC and pure software, this paper surveys the development of computer graphics. We start with the description of the theoretical problems related to computer graphics. Consequently, we present the most relevant industrial and academic solutions categorizing them from the point of view of their contribution in the speed up of Graphics Pipeline. Finally we introduce the MOLEN reconfigurable computer paradigm project and how the reconfigurable organizations based on this architecture could help in the establishment of an integral solution for computer graphics processing.**

Keywords— **Computer graphics, graphics pipeline; accelerators; MOLEN; reconfigurable computing; 3D pipeline; VirtexII PRO**

I. INTRODUCTION

Since the introduction of the sketchpad interactive drawing system in the sixties by Sutherland [1], computer graphics has evolved with the creation of new algorithms and supporting hardware for this new functionalities and capabilities. Historically, the sixties and seventies saw the creation of the elementary and essential algorithms like the efficient scan converting lines [2], ray tracing [3], Catmull's Z buffering [4], shading developed by Gouraud [5] and Phong [6], the characterization hidden-surface by Sutherland [7] and innovations and improvements presented by Blinn [8]. The eighties came with the introduction of accelerators for support 3D graphics primitives, improving in this way the algorithmic run time [9]. The film industry uses computer graphics into the creation of new effects [10], since those days, innovations has been created and virtual reality scenarios are part of currently films [11]. Nineties have had inherently the idea of photo realistic rendering, and the widespread of a more complex Application Specific Integrated Circuits (ASIC) [12] [13] for rendering acceleration. The massive introduction of hardware for the support of computer graphics into a personal computer (PC) world, oxygenated the research and development of computer graphics hardware, and the gap between PC rendering and the specialized graphics computer are diminishing every day [14][15].

Currently computer graphics are part of our life, we inhabit multimedia environments in the work, home and entertainment, even handheld devices like cellular phones and PDAs are being produced with specialized graphics processor and low power consumption characteristics like the pioneer Z3D [16] and the RAMP-IV 3D mobile graphics IC [17]. The continuous creation of new functionalities diminishes the cycle of life of the computer graphics hardware due his obsolescence. More flexible and adaptable hardware for different functionalities could be achieved with Field Programmable Logic (FPL)[18] technologies using the Reconfigurable Computing paradigm [19].

A reconfigurable computing machine called MOLEN ρμ-coded processor [20] was recently implemented in the Xilinx Virtex II PRO [21]. This configurable platform is intended for the development and improvements of processing including graphics.

Our currently research involves the mapping of graphics functions to the MOLEN processor platform. The remainder of this paper discusses the basics in computer graphics pipeline, surveying some traditional researches and the MOLEN framework; consequently we present computer graphics and reconfigurable computing, and conclude with some future work directions.

II. Background

Performing a scene or rendering involves a series of tasks beginning with the creation of basic objects or primitives like points, lines, and triangles. Primitives are specified in a world with the use of homogeneous coordinates, a three-dimensional representation, and have to be rendering in a screen that can represent the data in two dimensions. A pipeline [22] processes the initial data, and converts through several mathematical transformations and additions into a representation of picture elements (pixels) to be displayed in a computer screen; this pipeline is known as graphics rendering pipeline or simply graphics pipeline [23][24][25].

The three fundamental stages of the graphics-pipeline are: 1) Application, 2) Geometry and 3) Rasterizer as is shown in figure 1.

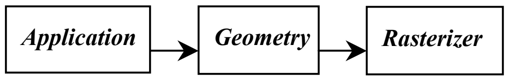

Figure 1: The basic Rendering Graphics Pipeline

Computer Graphic primitives have inherently a lack of data-dependence, for this reason the pipeline can be implemented by a group of parallel pipelines in a space parallelism manner [26]. In the following we will describe the main stages of the pipeline.

A. Application Stage

The scene construction is achieved with the use of an Application Programs Interfaces (APIs), like OpenGL [27] and Direct3D [28], helping us in the creation fundamental primitives, which are base of more complex objects. APIs functionalities includes: objects transforming, orientation modifications, re-sizing, viewing perspectives, projections creation, lighting scenes with different types of lights and, texturing and shading of objects in order to show them in a more realistic way. Summarizing, the previously described actions, establishes the tasks that will be carried out by the rest of the graphics pipeline.

The following code is an illustrative example of the quadrilateral creation with the OpenGL API [27].

```
#include "model3.h"

void Draw_A_Frame(void) // Draw  frame
{
glBegin(GL_QUADS) ;
  glColor3f(0.0,1.0,0.0) ;
  glVertex2f(0.25,0.25) ;
  glColor3f(1.0,1.0,0.0) ;
  glVertex2f(0.25,0.75) ;
  glColor3f(1.0,0.0,0.0) ;
  glVertex2f(0.75,0.75) ;
  glColor3f(0.0,0.0,1.0) ;
  glVertex2f(0.75,0.25) ;
glEnd() ;
}
```

As can be seen in the above code, object's vertices has associated three fundamental colors Red-Green-Blue (RGB-domain)[29], these primary colors are the base to construct the infinite palette available in the nature. Other functionalities not presented in the above code should be established, these includes the texturing coordinates and the computation of vertices normal-vectors. The primitives created in the application stage, are described in the affine space representations [23][24] for facility transformation purposes. Some systems are using another representations, like Euler [24] and Quaternion [30], accelerating in this way the rotations and orientations. After establishing the application framework, the Geometry Stage initiates the processing of the described data.

B. Geometry Stage

The following five stages compose the geometry pipeline (figure 2). A description of these stages are presented

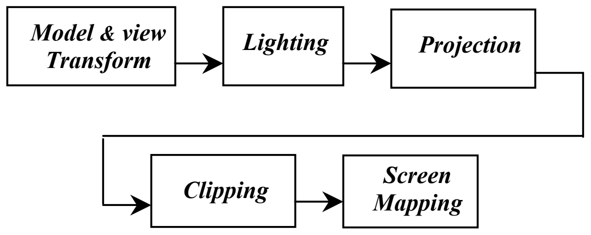

Figure 2. Geometrical Pipeline

Model and view transformation. The elaboration of a scene starts with the creation of the fundamental objects described in a *model space* with its own coordinates. In order to put all the objects that participate in the same scene into a common reference model, the *model space* has to be transformed into a *world space.* Some times, the eye of the viewer of the scene has to be positioned in different locations; in this case the *world space* has to be transformed into the *eye space*, the resultant model is ready to feed the next stage [24]. The mathematical operations for the transformations of the objects, and for the transformation between different models are carried out in the model and view transformation stage. Several scalar-vector multiplications between the vertices and the compound matrices transformation are performed, using a representation of floating-point numbers. Follows, three fundamental matrixes for translation, rotation and scaling are presented.

The object's movement is achieved with the translation matrix (1)

$$T(t) = T(x, y, z) = \begin{pmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (1)$$

where tx, ty, tx represent the amount of translation in each Cartesian coordinate.

Rotations matrices, help us to rotates the body an angle ϕ in a three-dimensional environment, this movements matrixes are depicted by equations (2), (3) and (4):

$$Rx(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (2)$$

$$Ry(\phi) = \begin{pmatrix} \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (3)$$

$$Rz(\phi) = \begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 & 0 \\ \sin(\phi) & \cos(\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (4)$$

The scale matrix (5), change the object relation into the x, y, and z coordinates.

$$S(s) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (5)$$

A detailed explanation of other matrices like shearing and compound can be found in [23][31]

Lightning. More realistic rendering are obtained using lights, different nature of the lights helps us to change the color of the objects in the modeled world; these colors are the result of the interaction of the light sources and the material that it impacts. The complete model of interaction of the light and the bodies that impacts, it is highly complex [32] and seems non-realistic for real time rendering and interactive environments. We use an equation instead of the approximation of the real behavior of the light; this equation is denominated the lighting equation (6).

$$I = K_a * I_a + K_d * I_l * \left(\vec{N} \bullet \vec{L}\right) + K_S * I_l * \left(\vec{N} \bullet \vec{H}\right)^{N_S} \qquad (6)$$

where K_a, K_d and K_s are ambient, diffuse and specular object reflectance respectively. I_l denotes the incident light, N is the normal to the surface, H is indicating vector of maximum highlight direction, and N_s is the glossiness factor [33]. The first term in (6) model the *Ambient Light* and represents a far away light from the scene that irradiates in all directions; an example of this kind of light is the sun, this light it is also knew as global light [27] or directional light [24]. The second term modeled describes the interaction of the light with a diffuse reflection body, and the third term approximates the specular reflection of light [6].

Some helpful sources of light are the *Point Light*; this multidirectional light is located in some point of the scene. Another positional light in the *Spot Light;* instead of a multidirectional illumination characteristic this light has a conic irradiance

Projection. This stage delimits the scene to be rendered; the modeled world is transformed and delimited by cubic representation by means of the application of the *orthographic projection,* implemented with translation and scaling transformations. Another more elaborated projection is called *perspective projection,* the resultant geometrical volume is not cubic instead of that, is a truncated pyramid volume, denominated *frustum*; this particular shape is caused by the representation of the far away objects with small sizes. Detailed information of projection matrices could be found in [27][23], and an introduction of projections and viewing transformations are established in [34].

Clipping. Objects outside the projection volume are dismissed because they are not visible; this action diminishes the processing time avoiding the scan conversion of not visible objects. One of basic algorithms developed by Cohen-Shutherland [35][23][25] computes the intersections of the lines and the viewing window, determining in this way the necessary information for the clipping; the searching area is split in nine regions, and the intersection of the object in a particular area is compared in a binary way.

An improvement of 36 % in processing time was achieved by Liang-Barsky algorithm, with the description of clipping in an exact mathematical form using a parametric representation [36]. Since then, have been developed specialized algorithms for the clipping of points, lines, polygons, texts and other objects [37][23][25]. The following figure schematizes the clipping action.

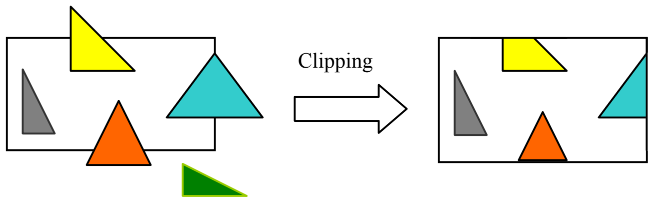

Figure 3. Clipping

Screen Mapping. Finally in the last stage of the geometry pipeline, the primitives are mapped into the screen coordinates, usually these coordinates are expressed in an integer format, then a normalization should be achieved in order to operate with integer representation. The following figure schematize the mapping operations indicated in equations 7-9 [25].

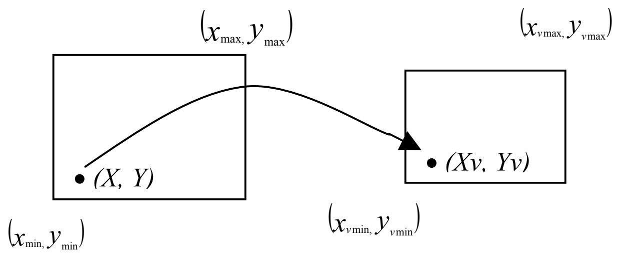

Figure 4. Mapping to screen coordinates.

$$x_v = x_{v\min} + (x - x_{\max})\frac{x_{v\max} - x_{v\min}}{x_{\max} - x_{\min}} \qquad (7)$$

$$y_v = y_{v\min} + (y - y_{\max})\frac{y_{v\max} - y_{v\min}}{y_{\max} - y_{\min}} \qquad (8)$$

$$z_v = z_{v\min} + (z - z_{\max})\frac{z_{v\max} - z_{v\min}}{z_{\max} - z_{\min}} \qquad (9)$$

The previous exposition of the *Geometry Pipeline* evidences his intensive floating-point data processing. Several improvements have been done in order to accelerate the processing time. The scalar-vector multiplication operations have been studied in [38] avoiding some unnecessary calculations. Others researchers, diminishes the latency of multiply-add (MAC) operations [39][40][41], a commonly used operation into the processing; while a more aggressive innovations merge the arithmetic logic in the double MAC unit [42], accelerating in this way the processing time.

Poor flexibility, high costs and rapid obsolescence are characteristic of ASICs, for this reason more specialized programmable processors have been created. These processors are using parallelism in time and space, achieving good performances like the represented ones of the following table.

Table I

Geometry Processors - Coprocessors

Name	Parallelism	ISA Optimized	Performance
Geometry Engine [43]	12 units in Time Parallelism.	Matrix, clipping, projection & others	5 MFLOPS
TGPx4 [44]	LIW	Geometry Stage & others	80 MFLOPS 40 MHz
FLOVA [45]	VLIW	SIMD Geometry Stage	500 MFLOPS 100MHz
Emotion Engine [46]	VLIW	SIMD Geometry Stage	5.52 GFLOPS 300MHz
VPU1 [47]	2 way VLIW	2 SIMD Geometry Stage	2.5 GFLOPS 250MHz
Four Way - VLIW Processor [48]	VLIW	SIMD Geometry Stage	2.5 GFLOPS 312MHz

Cost-effective oriented studies suggest the use of currently non-specialized processors for graphics processing [49] [50] [51] [52]. Sacrificing performance instead cost, this processor incorporates new functionalities to his ISA, the new functionalities in some processor includes special units for reciprocal and square root calculations [53]. Additional information of the challenges to combine GPP and multimedia can be found in [54].

The processing of computer graphics with the low power constraints, lead us to diminish the use of big floating-point units and emerge the idea of use different sized data [55]. Going further, other studies use integer arithmetic for real numbers representations [56], optimizing in this way the computer graphics bandwidth and consumed power [57]. Some commercial processors are actually using this approach of processing [58], a complementary information is found in [59]

C. Rasterization Stage

The Rasterization stage is implemented with a pipeline that converts the primitives into an image, determining the final color of the pixels. The description of each part of the pipeline of the figure 5 is presented.

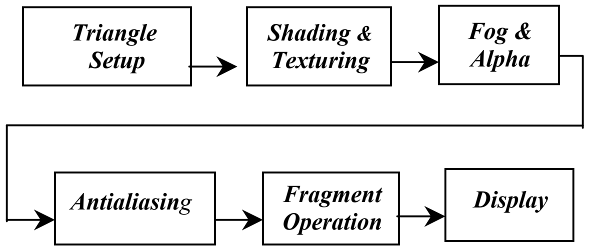

Figure 5. Rasterization Pipeline.

Triangle Setup. The first step in the rasterization stage executes a back face-culling test [27]. This test computes the sign area of the triangle in order to discards the not visible primitives. This test can also be made calculating the normal of the projected polygon [24]. The next step approximates the mathematical description of the primitive. The simplest and widely applied algorithm uses the edge function (equ.12)

$$\Delta_y = y_1 - y_0 \qquad (10)$$

$$\Delta_x = x_1 - x_0 \qquad (11)$$

$$E(x,y) = (x - x_0) * \Delta_y - (y - y_0) * \Delta_x \qquad (12)$$

Based on this equation Pineda [60] proposes an easy way to increment and update the edge with equations 13 and 14.

$$E(x,y+1) = E(x,y) - \Delta_x \qquad (13)$$

$$E(x+1,y) = E(x,y) + \Delta_y \qquad (14)$$

For scan conversion phase, different algorithms for traverse the triangle have been established. The Digital Differential Analyzer (DDA) [23] is one of the traditional algorithms used in scan conversion; others variations for traversing are also proposed and studied in [61][24][27].

Finalizing this stage, the interpolation of colors, and depth values for each pixel that has been created recently in the scan procedure is carried out, merging in this way the Triangle Set-up and the Shading functionalities.

Shading. Depending on the used technique some lightning model is evaluated, the main three fundamental techniques widely used to shading are:

1. **Flat Shading** [23]. The fastest and simple to implement, establishes a common color to the triangle, this color is obtained after averaging the colors associated with the triangle vertices.

2. ***Gouraud Shading*** [5]. Interpolates color across the triangle, taking into account the colors associated with the vertices computed with equation (6). The first interpolation occurs between the vertices and after a second interpolation occurs between the edges lines resulting in a more realistic scene compared with *Flat Shading*.

3. ***Phong Shading*** [6]. The third is the most costly algorithm in terms of processing time, can represent effects like spotlight, offering greater realism. Phong algorithm computes the normal vector in each point of the triangle (equation 6), interpolating the normal vectors at the vertices of the triangle, then a second interpolation occurs in the scan lines. This technique is similar as Gouraud; but instead of interpolate colors, the normal vectors are interpolated; finally the shading model is applied obtaining the final color.

Texturing. With the texels (pixels from the texture image) and the lightning equation a more realistic image is create; the following pipeline realizes this process.

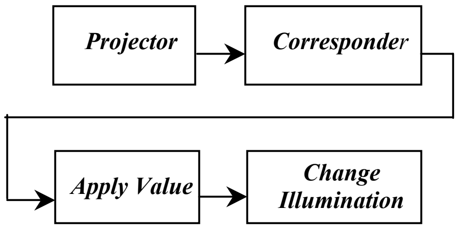

Figure 6. Texture pipeline.

A *Projector function* translates the texture coordinates into a parameter space with values *u* and *v*, this translation is called mapping. With the use of another mapping function denominated C*orresponder function*, the values *u* and *v* are translated to the texture values, which are scaled with the *Apply value function* in order to get finally the texture pixel that modify the characteristic value into the Illumination equation [24].

All these operations are expensive in terms of computation, memory use with the texture images, and bandwidth in order to move this data between the different buffers of the graphics pipeline [26]. After mapping the image gets warped and should be filtered [62]; some of these filters have prohibitive processing times for real time applications. On the other hand, we can use some pre-filtering images as Lance Williams proposes with his Pyramidal data structures called Mip-Mapping [63]. This technique creates multiple copies of the original texture image; each copy has exactly the half resolution on each axis of the previous one in both coordinates *v* and *u*, resulting a one-quarter size filtered image, this procedure follows recursively until the final image has size of one pixel. The obtained copies form a pyramid in which the new coordinate *d* is used to index the most adequate image resolution to be mapped into a desire pixel area. Figure 7 sketches this technique.

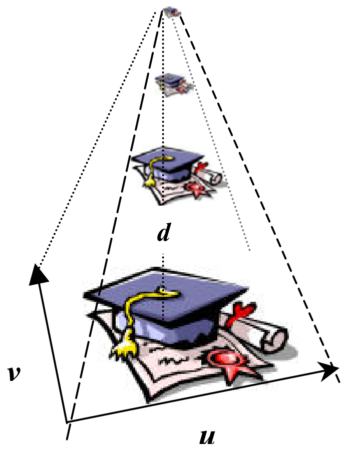

Figure 7 MIP- Mapping

Bilinear and trilateral [63][23] interpolations are been used to obtain an antialiased-texturing pattern to be applied into the polygon.

Fog. In order to create a more realistic image some times it is necessary to apply atmospheric effects like a mist or a heavy fog depending on the scene. Fog is characterized by the C_f color, and a fog factor *F*. If C is the color resident in memory (frame buffer), the fog equation computes the final color C_F:

$$C_F = F \cdot C + (1-F)C_f \qquad (15)$$

the factor F is calculated from:

$$F = e^{-d_F z_P} \qquad (16)$$

where d_F controls the fog density, and z_p is the z value of the pixel [64].

Antialiassing. Data represented into a raster display suffer the jagged effect due the inherent discrete characteristic of this device. But avoiding this unsolvable characteristic, aliased images can be affected by an inadequate sampling of the image made with lower rate than the Nyquist theorem [65] establishes; then it is infeasible to reconstruct an image if the sampling rate is less than twice the highest frequency of the image. There are two fundamental solutions into the image filtering denominated Pre-filtering and Post-filtering.

1. **Pre-filtering.** This technique determines the color of the pixel based on the geometric description of the image; taking into account the description, a precise filter is applied in order to obtain a non-aliased image. Breshman's classical algorithms had been used [66], and also Pitheway [67] developed an improvement in the way of the incrementing of the shading area taking into account the slope of the edge.

2. **Post-filtering.** The classical approach in Post-filtering use the super-sampling [68] of the image, multiple pixel samples are took, and a filter is used to create a new sample by averaging of the samples. Barlett, box, Gaussian, and other discrete filters [62] can also used instead of the average filter in order to eliminate the high frequency components.

Alpha Blending. The RGB components have associated the α factor for the transparency and opacity control of the object; the blending of this factor with the processed color of the pixel determines the final opacity of the rendered pixel:

$$C_{AB} = C_P \cdot \alpha + (1-\alpha)C \qquad (17)$$

In (17) *C* is the color in the frame buffer, C_p and α are the color and alpha value of the incoming fragment to being processed. The final color to be written in the frame buffer is a blending of the actual color and the incoming color with different grades of transparency.

Depth test. With this test it is possible to determine which pixels can be viewed and which are hidden behind objects. With the comparison of the depth value of the incoming processed fragment and the actual value stored in Z buffer (frame buffer) it is possible to discard the hidden object [23][24][25][27].

Fragment Operation. Depending on the API technology, and if some functionalities were enabled, some additional test operations are carried out. Related information on fragment test can be found in [64].

Display. Finally this stage ends the pipeline; a specialized controller is in charge of the display actualization using the frame buffer information. A detailed reference of the functionalities of this stage could be found in [23][24].

The quantity of processed data in the rasterization pipeline is higher than the processed in the geometry stage; the number of fragments produced by the primitive setup exceeds the number of primitives in at least 20 to 1. Even taking into account the use of integer data in the rasterization stage instead of the floating point utilized by the geometry stage, the amount of processing data reaches bigger amounts.

The available technology evidences the use of different kind of parallelism and with different degrees by the geometry and rasterization stages. The following table shows some examples of rendering accelerators and processors.

Table II
Rendering Accelerators and Processors

Name	Parallelism	Operation	Performance
4D/240GTX [69]	Time parallelism in geometry and space in rasterization.	Geometry and Rasterizer Stages	100K lighted quadrilaterals per second
InfinitiReality [70]	4 Geometry Engine MIMD, 4x80 engines in raster boards	Geometry and Rasterizer Stages	710 M textured antialiased pixels/s
Neon [71]	8 Pixel processor per Rasterizer, simgle chip with unified memory	Rendering	4 Mvertices/s
Truga001 [72]	12 graphic processors and 7 functional units in a single chip	MIMD structure Rendering	4 M vertices/s
GeForce FX 5800 [73]	Single chip	Engines for Geometry and Rasterization.	200 Mvertices/s 4 Billon texels/s 500MHz
VISUALIZE fx 6 [74]	Scalable processor, 3 Geometry up to 8, 2 Raster and 2 texture chips	Array of units with Space and Time parallelism	-

The large quantity of data involved in the graphics pipeline has to be taken into account in order to understand the used bandwidths and the necessary computational power. Figure 8 depicts a simplified pipeline with the data amounts moved between the principal stages, and between stages and memories; also are depicted the sort point for parallel organizations.

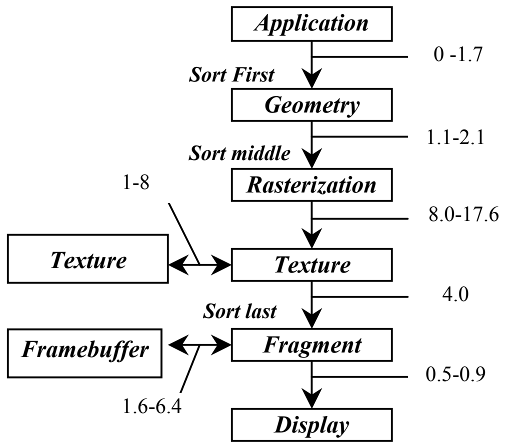

Figure 8 Data bandwidths in GHz for 60MComands of input, 20M Vertices, 400M pixels and 120Msamples[26].

The necessary processing power is achieved by increasing the parallelism and the clock rate, nevertheless more parallelism causes a bandwidth increment [75] and higher clock rates also evidence even more the inherently memory latency problem [22].

Different approaches were made in order to hide the memory latency and manage bandwidths. The use of a texture cache for fast local retrieval of textures is primarily used [76] and different organizations were studied like multilevel texturing instead of simple caching architectures [77], a detail architectural analysis could be found in [78]. Also was shown that merging prefetching and caching [79], improvements in performance are reached. Even better performance was obtained using parallel distributions of different hierarchy for texturing memory [80][81]. The parallelism benefits are also exploited in other stages in different degrees.

Molnar [82] proposes three main approaches to exploiting the parallelism: sort first [83], sort middle [84][85] (image-oriented), and sort last [86][87][88] (pixel-oriented), suggested in the figure 8. This basic division establishes a starting point of redistribution of data between the parallel processors at different levels of the pipeline. The first two sort schemes suffer of lack in load balance due to the unknown number and sizes of primitives. The sort last is highly dependent on the functionalities enable in the fragment processing and also suffers of load imbalance. Eldridge presents the sort-anywhere architecture [26], based on a high connectivity between the processing clusters improving the distribution the data in a more balanced way.

Other research lines improves the rendering through the use of *adaptive rendering*, in this sense, Bergman propose the adaptations of the image, generating an image rapidly without so many detail and then refining it when was necessary [89], in this way only the necessary pixels are Phong shaded, the rest of the pixels use the Gourard technique. Following this paradigm, Cho [90] improves the determination of which triangles should be Phong or Gourard shaded. Similar approaches diminish the processing time, using a multi-resolution model [91] and representing the objects with different level of details. Finally, taking into account the improvements in the technology of embedded memory [92][93] with read cycles of 2.9 ns in DRAM [94], we hope the creation of multiple multimedia processors with higher bandwidths and less power consumption characteristics. One example of this approach with a sort middle architecture and a dynamic reconfigurable bus is presented in [95].

III. The MOLEN Paradigm

A. Reconfigurable Computing

The capacity to transform a hardware platform imposed and controlled by the software is denominated Reconfigurable Computing (RC). The reconfigurable computing was introduced four decades ago, but recently the last decade [96] has been the witness of the evolution and growth of this important field in the computer science, the catalyst of this development comes with the improved performance of the Field Programmable Logic, it usually assembles a general purpose core and a field programmable unit. This new hybrid architecture is referred like Field-Programmable Custom Computing Machine (FCCM) [97]. The following figure outlines this rationality [98].

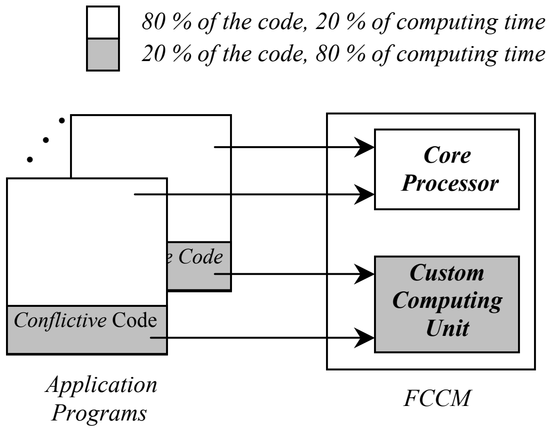
80 % of the code, 20 % of computing time
20 % of the code, 80 % of computing time
Core Processor
Custom Computing Unit
Code
Conflictive Code
Application Programs
FCCM

Figure 9. Rationality behind the FCCM

The FPL technology comes associated with the concept of *Virtual Hardware*, in which any application believes that has a sized engine to run on it, and it is possible to establish the hardware on demand paradigm, overcoming the hardware obsolescence imposed by other technologies.

Reconfigurable technologies demonstrated a great flexibility and a good performance in order to replace the traditional solutions in high demand tasks [99], offering spatial and temporal parallelism characteristics [19] and also the inherently bit level parallelism [98]. Table III summarizes the principal characteristics of RC compared to traditional solutions.

Table III
Reconfigurable advantages and drawbacks compared with other technologies.

	Power	Performance	Flexibility	Time to Market
General Purpose Processor	High	Low	Medium	Low
ASIC	Low	High	Low	High
Re-Configurable Processor	High	Medium	High	Low

Nevertheless, the main drawback of the RC is the necessary configuration time of the new hardware functionality. Several studies show the importance of the Run time reconfiguration [100][101][102], and some researchers looks for the hiding of the configuration latency time. One of this works proposes the use of matched common components for his use in different tasks, sharing in this way the same hardware [103] and diminishing the overall configuration time. Another approach proposes work with different contexts [104]; the dynamic of this solution is based on switching the context on demand, this solution consumes a less time compared with the configuring of the FPL. A similar approach [105] stores different configurations in the internal memory of the FPL, and has the capability to change context in a single cycle. The previously presented solutions constitutes the first approaches in order to hide the configuration time and fulfill the goal, nevertheless they suffer a memory overuse.

FPL technology evolves and the configuration time of the devices are diminishing gradually. Table IV presents the configuration times for Virtex II PRO family, working with a 50MHz clock.

Table IV

Virtex II PRO sizes and Programming times [106]

Device	No. of frames	No. of bits	Config. Time
XC2VP2	884	1,305,440	3,26 ms
XC2VP7	1,320	4,484,472	11,21 ms
XC2VP20	1,756	8,214,624	20,54 ms
XC2VP50	2,628	19,005,696	47.55 ms

Additionally, we must emphasize that new FPL devices support the partial reconfiguration of the logic and routing characteristics. Also, the dynamic configuration is supported; this characteristic gives us the ability to update only a portion of the configuration memory in a FPL with a new configuration without stopping the functionality of other device sections [107].

B. MOLEN Processor

The MOLEN ρμ-coded processor presented in [108] constitutes an FCCM and it is merging a general-purpose processor and a reconfigurable processor. This FCCM uses micro code concept to carry out the configuration process of the augmented CCU, as well for the emulation of the execution of the core processing unit and the control of the execution of the reconfigurable unit.

The microcode is referred as *ρμ-coded*, and is located into the traditional μprogram memory. Figure 10 depicts the general architecture of MOLEN.

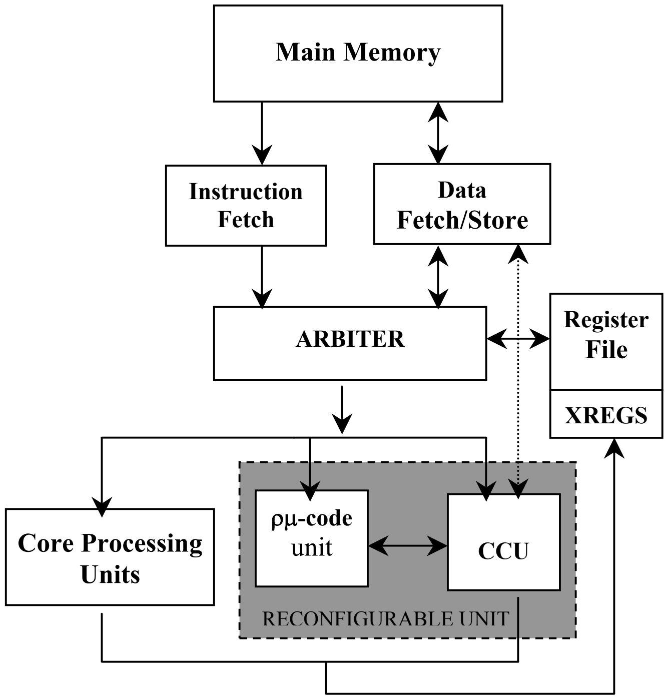

Figure 10. MOLEN Organization

The *ARBITER* partially decodes the fetched instruction, determining where will be issued for his execution; it has two alternatives the *Core Processing Unit* and the *Reconfigurable* units. A precise description involved in control tasks and functionalities of the *ARBITER* could be found in [109]

The currently implementation of MOLEN uses a PowerPC 405 [110] processor as the *Core Processing Unit*. The instructions issued to this unit are decoded and executed in a normal RISC way. This unit uses the Register File for hold the initial and resulting data, finally the Exchange Registers (XREGS) are an architectural support for the parameter passing between the core processing units and the reconfigurable unit.

The *Reconfigurable Unit* consists of a Custom Computing Unit (CCU) and a *ρμ-code* unit. The particular ISA of the MOLEN is composed by three fundamental instructions distributed on the *set phase* and the *execute phase*:

1. **Set Phase.** The set phase it is constituted by two sub phases: The first one is known as the *partially set (p-set),* and the second one is cited as *complete set (c-set).* In *p-set* sub-phase, the *CCU* is partially configured in order to perform common functions; these actions can be made during the loading of the program or even at chip fabrication time. In the second subphase the *c-set*, as its name suggest, the microinstructions establish the final functionality of the *CCU* enabling to perform less frequent functions.

2. **Execute Phase.** Once it has been established the functionality of the hardware, the initiation and regulation of the execution is performed by this instruction. When an instruction is being executed into the *CCU*, the *ARBITER* interrupts the *Core Processing Unit* in order to regulate the bus access.

Through the reconfigurable instructions utilization it is possible to control the whole MOLEN organization. Taking into account the instruction format of the chosen Core Processing Unit (CP), it has been created the reconfigurable instruction with the format presented in figure 11, which is congruent with the PowerPC Instruction Set Architecture.

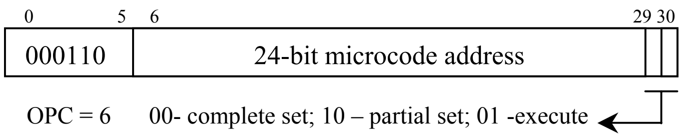

Figure 11 Reconfigurable instruction encoding: *R-form*

The delay introduced by the ARBITER in the decoding of the instructions is insignificant [106]. Therefore, encouraged in the functionalities of the *set* and *execute* instructions, we can create multiples CCU, and to control the executions of instructions for these units with the same delay and without increasing the decoder complexity. This is achieved through the use of the 24-bit address of the reconfigurable instructions *R-form*, which constitute it the first μinstructions address of the current reconfigurable instruction service.

Additional information regarding the functionalities of MOLEN including the memory hierarchy, CACHE facilities and also an accurate description of the architecture and paradigm could be found in [108]

IV. Related Work in Computer Graphics and Reconfigurable Computing

The utilization of the FPL technology into the development of particular stages of the computer graphics pipeline was foreseeable. The intrinsic characteristic of FPL technology in fast prototyping and rapid changes into the design without highest costs encouraged the development of several computer graphics stages using FLP [111] [112] [113] [114][115][116].

The survey shows that few works were presented relating computer graphics and reconfiguration computing. One interesting work utilize the M1 reconfigurable system [117] for mapping the geometrical transformations; an algorithm distributes the load in order to perform the vector-scalar operations in the array of reconfigurable units that conform this architecture, improvements of this approach with respect to the traditional processing were reported in [118].

Finally we want to mention another interesting work, in the line of adaptive shading, power-aware 3D computer graphics, and relationship of energy and perceptual quality. This research reports an -energy efficient rendering environment-, achieved with the use of low quality algorithms when Human Visual Perceptions (HVP) is less than a pre-established threshold. [119]

V. Conclusions and Future Work

From our point of view, reconfigurable computing and the MOLEN paradigm became in the framework for the development of the new fully adaptable and reconfigurable graphics pipeline.

Graphics pipeline under the MOLEN paradigm can arise a flexible architecture, as much as the environment of processing requires. Custom computing machines in MOLEN can be adaptable and the different clusters into the pipeline will be created when be necessary, enabling the data *sort in a demand paradigm*. Also the embedded memory can be managed in a dynamic way, regulating in this way the bandwidth and resources. The following figure sketches this view.

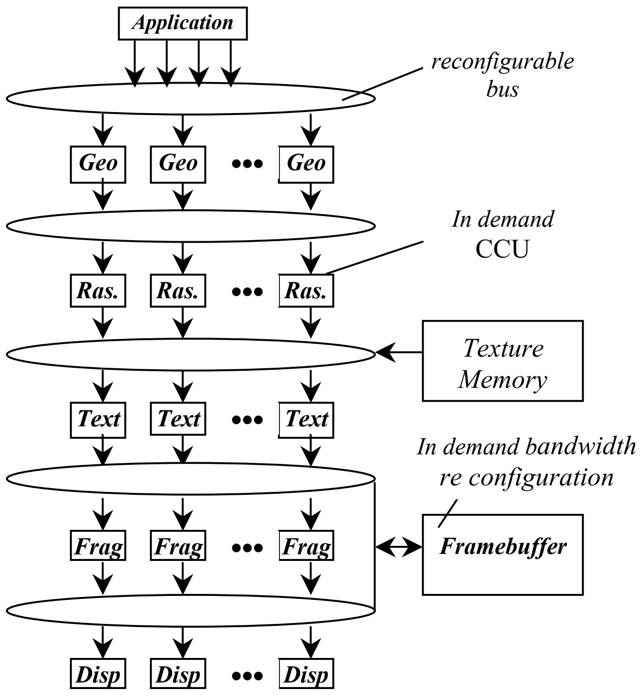

Figure 12 Sort in demand paradigm support architecture

Finally, in order to reach the *sort in demand paradigm* the following researches and studies should be necessary:

1. The evaluation and quantification of the necessary amount of computing parallelism into the graphics pipeline in principal stages and sub stages for different applications, and systems.
2. Study of the performances of different memory hierarchies in systems with embedded memory like the Virtex II PRO.
3. The necessary grade of connectivity of the reconfigurable buses in order to support the

paradigm.

4. A classification of similar functionalities and the establishment of common basic reconfigurable hardware entities for the service of multiple functionalities.
5. Determining of the appropriate granularity of reconfiguration in order to increase the functionalities of the MOLEN architecture.

We believe that the MOLEN paradigm can help into the creation of a less expensive and adaptable processing to the multimedia environments.

REFERENCES

[1] Ivan E. Sutherland, Sketchpad A Man-Machine Graphical Communication System, *Proceedings –Spring Joint Computer Conference,* pag. 507- 524, USA 1963,

[2] J. E. Bresenham, Algorithm for computer control of a digital plotter, *IBM System Journal*, Volume 4, Number 1, Page 25 (1965), Non topical Issue

[3] Appel, A, Some Techniques for Shading Machine Renderings of solids, *1968 SJCC*, pp 37-45

[4] E. Catmull, A Subdivision Algorithm for Computer Display of Curved Surfaces, *PhD thesis, University of Utah*, 1974.

[5] Gouraud H, Continius Shading of curved surfaces, *IEEE Transactions on Computers*, C-20 (6), 623-629, June 1971

[6] Bui Tuong Phong, Illumination for Computer Generated Pictures, *Communications of the ACM*, Volume 18 Issue 6, June 1975.

[7] Evan E. Sutherland, Robert F. Sproull, Robert A. Schumacker, A characterization of ten hidden-surface algorithms, *ACM Computing Surveys (CSUR)*, Volume 6 Issue 1

[8] James F. Blinn, Martin E. Newell, texture and reflexion in computer Generated images**,** *Communications of the ACM*, Volume 19 Issue 10, October 1976.

[9] H. M. Levy, Vax Station, A General Purpose Raster Graphics Architecture, *ACM Transactions on Graphics (TOG)*, Volume 3 Issue 1, January 1984.

[10] Richard Rouse, Columns: Gamming and Graphics: looking for some art amidst the technology, *ACM SIGGRAPH Computer Graphics*, Volume 33 Issue 1, February 1999.

[11] Tom Porter, Galyn Susman, On Site: creating lifelike characters in Pixar movies, January 2000, *Communications of the ACM*, Volume 43 Issue 1, Pages: 25-29

[12] Juhola, T.; Tenhunen, H.; Nielsen, I.R.; Adoption and utilization of ASIC technologies in European SMIs, ASIC Conference and Exhibit, 1994. *Proceedings., Seventh Annual IEEE International* , 19-23 Sept. 1994 , Page(s): 240 -244

[13] Michael F. Deering, Scott R. Nelson, Leo: a system for cost effective 3D shaded graphics, *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, September 1993

[14] Macedonia, M.; The computer graphics wars heat up, *Computer, Volume: 35* Issue: 10, Oct. 2002 Page(s): 97 –99

[15] Mark J. Harris, Greg Coombe, Thorsten Scheuermann, Anselmo Lastra, Physically-Based Visual Simulation on Graphics Hardware, *Proceedings of the conference on Graphics hardware 2002*, September 2002

[16] Masatoshi Kameyama, Yoshiyuki Kato, Hitoshi Fujimoto, Hiroyasu Negishi, Yukio Kodama, Yoshitsugu Inoue, Hiroyuki Kawai, *3D Graphics LSI Core for Mobile Phone Z3D, 2003 Graphics Hardware Workshop*, July 26 to July 27, 2003, in San Diego, California

[17] Ramchan Woo, Sungdae Choi, Ju-Ho Sohn, Seong-Jun Song, Young-Don Bae and Hoi-Jun Yoo, A Low-Power and High-Performance 2D/3D Graphics Accelerator for Mobile Multimedia Applications, *HotChips 2003*; http://ssl.kaist.ac.kr/ramp/

[18] Krupnova, H.; Saucier, G.; FPGA technology snapshot: current devices and design tools, Rapid System Prototyping, 2000. *RSP 2000. Proceedings. 11th International Workshop on*, 21-23 June 2000, Page(s): 200 -205

[19] DeHon, A.; Wawrzynek, J.; Reconfigurable computing: what, why, and implications for design automation, *Design Automation Conference*, 1999. Proceedings. 36th , 21-25 June 1999, Page(s): 610 –615

[20] Stamatis Vassiliadis, Stephan Wong and Sorin Cotofana, The MOLEN ρμ-coded processor, Proceedings of the *11th International Conference on Field-Programmable Logic and Applications 2001 (FPL2001)*, Belfast, Northern Ireland, UK, August 2001

[21] Xilinx, Virtex II Pro Platform FPGA Handbook, Xilinx Inc. UG012 (v1.0) January 2002.USA.

[22] John L. Hennessy and David A. Patterson, Computer Architecture: A Quantitative Approach, *Morgan Kaufmann Publisher*, 1990 USA

[23] James D. Foley, Andries van Dam, StevenK. Feiner and John F. Hughes, Computer Graphics Principles and Practice, *Addison-Wesley Publishing Company*, 1997, USA ISBN: 0201848406.

[24] Tomas Moller and Eric Haines, Real Time Rendering, *A K Peters 1999*, USA, ISBN: 1-56881-101-2

[25] Edward Angel, Interactive Computer Graphics a Top-Down Approach with OpenGL, *Addison Wesley, 2000*, USA, ISBN: 0-201-38597-X

[26] Matthew Eldridge; Designing Graphics Architectures Around Scalability and Communication, *Ph.D. dissertation, Stanford University*, June 2001.

[27] Mark Segal,Kurt Akeley, The OpenGL Graphics System: A Specification (version 1.4) , *Silicon Graphics Inc*., 2002,

[28] http://msdn.microsoft.com/library/

[29] Michael W. Schwarz, William B. Cowan, John C. Beatty, An experimental comparison of RGB, YIQ, LAB, HSV, and opponent color models, *ACM Transactions on Graphics (TOG)*, Volume 6 Issue 2, April 1987.

[30] Ken Shoemake, Animating Rotation with Quaternion Curves, *ACM SIGGRAPH Computer Graphics*, Proceedings of the 12th annual conference on Computer graphics and interactive techniques, Volume 19 Issue 3, July 1985

[31] F.S. Hill, Computer Graphics using OpenGL, *Prentice Hall, 2001*,USA, ISBN: 0-20-354856-8

[32] Wolfgang Heidrich, High-quality Shading and Lighting for Hardware-accelerated Rendering, *Ph.D. dissertation, University of Erlangen - Nurnberg*, 1999

[33] Hyun-Chul Shin; Jin-Aeon Lee; Lee-Sup Kim; A hardware cost minimized fast Phong shader, *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Volume: 9 Issue: 2, April 2001, Page(s): 297 –304

[34] Ingrid Carlbom, Joseph Paciorek, Planar Geometric Projections and Viewing Transformations, *ACM Computing Surveys (CSUR)*, Volume 10 Issue 4, December 1978

[35] Newman, W. M. and R. F. Sproull, Principles of Interactive Computer Graphics, 2nd ed. *McGraw-Hill*, New York, 1979.

[36] You Dou Liang and Brian A Barsky, A New Concept and Method for Line Clipping, *ACM Transactions on Graphics (TOG)*, Volume 3 Issue 1, January 1984

[37] Tina M. Nicholl, D. T. Lee, Robin A. Nicholl, An new efficient Algorithm for 2D line clipping: its development and analysis. *ACM SIGGRAPH Computer Graphics, Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, Volume 21 Issue 4, August 1987.

[38] Karagianni, K.; Stouraitis, T.; A vector processor for 3-D geometrical transformations, Circuits and Systems, 2001. ISCAS 2001. *The 2001 IEEE International Symposium on*, Volume: 4, 6-9 May 2001, Page(s): 482 -485 vol. 4

[39] An embedded 32-b microprocessor core for low-power and high-performance applications, Clark, L.T.; Hoffman, E.J.; Miller, J.; Biyani, M.; Luyun Liao; Strazdus, S.; Morrow, M.; Velarde, K.E.; Yarch, M.A.; Solid-State Circuits, *IEEE Journal of, Volume*: 36 Issue: 11 , Nov. 2001, Page(s): 1599 -1608

[40] Lang, T.; Bruguera, J.D.; Floating-point fused multiply-add with reduced latency,Computer Design: VLSI in Computers and Processors, 2002. *Proceedings. 2002 IEEE International Conference on, 16-18 Sept. 2002*, Page(s): 145 –150

[41] Michael J. Flynn and Stuart F. Oberman, Advanced Computer Arithmetic Design, *John Wiley & Sons*, Inc, 2001 USA, ISBN: 0-471-41209-0

[42] N. Yaday, M.J. Schulte, C. J. Glossner, Parallel Saturating Fractional Arithmetic Units, *Proceedings of the Ninth Great Lakes Symposium on VLSI*, pp. 214-217, Ypsilanti, MI, USA, March 1999, Projectcode: ET01-05

[43] James H. Clark, The Geometry Engine: A VLSI Geometry System for graphics, *Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, July 1982.

[44] Awaga, M.; Ohtsuka, T.; Yoshizawa, H.; Sasaki, S.; 3D graphics processor chip set, *Micro, IEEE*, Volume: 15 Issue: 6, Dec. 1995, Page(s): 37 -45

[45] Sang-Joon Nam; Byoung-Woon Kim; Yeon-Ho Im; Young-Su Kwon; Jun-Hee Lee; Young-Wook Cheon; Sung-Jae Byun; Dae-Hyun Lee; Chong-Min Kyung; FLOVA: A four-issue VLIW geometry processor with SIMD instructions and lighting acceleration unit, *Custom Integrated Circuits Conference, 2000. CICC. Proceedings of the IEEE 2000*, 21-24 May 2000, Page(s): 551 -554

[46] Kunimatsu, A.; Ide, N.; Sato, T.; Endo, Y.; Murakami, H.; Kamei, T.; Hirano, M.; Ishihara, F.; Tago, H.; Oka, M.; Ohba, A.; Yutaka, T.; Okada, T.; Suzuoki, M.; Vector unit architecture for emotion synthesis, *Micro, IEEE*, Volume: 20 Issue: 2 , March-April 2000 , Page(s): 40 -47

[47] Suzuoki, M.; Kutaragi, K.; Hiroi, T.; Magoshi, H.; Okamoto, S.; Oka, M.; Ohba, A.; Yamamoto, Y.; Furuhashi, M.; Tanaka, M.; Yutaka, T.; Okada, T.; Nagamatsu, M.; Urakawa, Y.; Funyu, M.; Kunimatsu, A.; Goto, H.; Hashimoto, K.; Ide, N.; Murakami, H.; Ohtagu, A microprocessor with a 128-bit CPU, ten floating-point MAC's, four floating-point dividers, and an MPEG-2 decoder, *Solid-State Circuits, IEEE Journal of*, Volume: 34 Issue: 11, Nov. 1999, Page(s): 1608 –1618

[48] Kubosawa, H.; Higaki, N.; Ando, S.; Takahashi, H.; Asada, Y.; Anbutsu, H.; Sato, T.; Sakate, M.; Suga, A.; Kimura, M.; Miyake, H.; Okano, H.; Asato, A.; Kimura, Y.; Nakayama, H.; Kimoto, M.; Hirochi, K.; Saito, H.; Kaido, N.; Nakagawa, Y.; Shimada, T.; A 2.5-GFLOPS, 6.5 million polygons per second, four-way VLIW geometry processor with SIMD instructions and a software bypass mechanism, *Solid-State Circuits, IEEE Journal of, Volume: 34 Issue: 11*, Nov. 1999, Page(s): 1619 –1626,

[49] Lempel, O.; Peleg, A.; Weiser, U.;Intel's MMX technology-a new instruction set extension, Compcon '97. *Proceedings, IEEE*, 23-26 Feb. 1997 Page(s): 255 –259

[50] Peleg, A.; Weiser, U.; MMX technology extension to the Intel architecture, *Micro, IEEE*, Volume: 16 Issue: 4, Aug. 1996, Page(s): 42 -50

[51] Thakkur, S.; Huff, T.; Internet Streaming SIMD Extensions, *Computer, Volume: 32* Issue: 12, Dec. 1999 Page(s): 26 –34

[52] Diefendorff, K.; Dubey, P.K.; Hochsprung, R.; Scale, H.; AltiVec extension to PowerPC accelerates media processing, *Micro, IEEE*, Volume: 20 Issue: 2 , March-April 2000, Page(s): 85 -95

[53] Oberman, S.; Favor, G.; Weber, F.; AMD 3DNow! Technology: architecture and implementations, *Micro IEEE*, Volume: 19 Issue: 2, March-April 1999, Page(s): 37 –48

[54] Conte, T.M.; Dubey, P.K.; Jennings, M.D.; Lee, R.B.; Peleg, A.; Rathnam, S.; Schlansker, M.; Song, P.; Wolfe, A.; Challenges to combining general-purpose and multimedia processors, *Computer*, Volume: 30 Issue: 12, Dec. 1997, Page(s): 33 –37

[55] Lee, R.B.; Accelerating multimedia with enhanced microprocessors, *Micro, IEEE*, Volume: 15 Issue: 2, April 1995, Page(s): 22 –32.

[56] Daniel Menard, Daniel Chillet, François Charot, Olivier Sentieys, Automatic Floating-point to Fixed-point Conversion for DSP Code Generation, *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems*, October 2002, New Orleans-USA

[57] Tang, K.C.; Wu, A.K.M.; Fong, A.S.; Pao, D.C.W.; Integrated partition integer execution unit for multimedia and conventional applications, *Electronics, Circuits and Systems, 1998 IEEE International Conference on*, Volume: 2, 7-10 Sept. 1998, Page(s): 103 -107 vol.2

[58] Clark, L.T.; Hoffman, E.J.; Miller, J.; Biyani, M.; Luyun Liao; Strazdus, S.; Morrow, M.; Velarde, K.E.; Yarch, M.A.; An embedded 32-b microprocessor core for low-power and high-performance applications, *Solid-State Circuits, IEEE Journal of*, Volume: 36 Issue: 11, Nov. 2001, Page(s): 1599 –1608

[59] Gopi K. Kolli, Kyle Fox, Haim Barad and Stephen Junkins, 3D Graphics Optimizations for Intel PCA Applications Processors with Intel XScale Technology, *Solutions Journal*, Volume 3, spring 2002, www.intel.com/pca/developernetwork.

[60] Juan Pineda, A Parallel Algorithm for Polygon Rasterization, ACM SIGGRAPH Computer Graphics, *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, Volume 22 Issue 4, June 1988

[61] W. Jack Bouknight, A Procedure for Generating of Three-dimensional Half –toned Computer Graphics Presentation, *Communications of the ACM,* Volume 13 Issue 9, September 1970.

[62] Alan V. Oppenheim, Discrete Time Signal Processing, *Prentice Hall PTR*, I1999 USA, SBN: 0137549202

[63] Lance Williams, Pyramidal Parametrics, *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, July 1983

[64] Keith Cok, Roger Corron, Bob Kuehne and Tomas True, Developing efficient graphics Software, *SIGGRAPH 2000* Course, http://www.sgi.com/events/siggraph00/gfxapps

[65] B.P Lathi, Modern Digital and Analog Communications Systems, *Oxford University Press*, 1998, New York. ISBN 0-19-511009-9.

[66] Don P. Mitchell, Generating Antialiased Images at low sample densities, *ACM SIGGRAPH Computer Graphics, Proceedings* of the 14th annual conference on Computer graphics and interactive techniques, Volume 21 Issue 4, august 1987.

[67] M. L. V. Pitteway, D. J. Watkinson, Bresenham's Algorithm with gray Scale, *Communications of the ACM*, Volume 23 Issue 11, November 1980.

[68] Norman P. Jouppi, Chun-Fa Chang, Z3: an economical hardware technique for high-quality antialiasing and transparency, *Proceedings of the 1999 Eurographics/SIGGRAPH workshop on Graphics hardware*, July 1999

[69] Akeley, K.; The Silicon Graphics 4D/240GTX superworkstation, *Computer Graphics and Applications, IEEE*, Volume: 9 Issue: 4, July 1989, Page(s): 71 –83

[70] John S. Montrym, Daniel R. Baum, David L. Dignam, Christopher J. Migdal, InfiniteReality: a real-time graphics system, *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, August 1997

[71] Joel McCormack, Robert McNamara, Christopher Gianos, Larry Seiler, Norman P. Jouppi, Ken Correll, Todd Dutton, and John Zurawski, Neon: A Big, Fast, 3D Workstation Graphics Accelerator, *WRL Research Report 98/1*, July 1999.

[72] Ikedo, T.; Ma, J.; The Truga001: a scalable rendering processor, *Computer Graphics and Applications*, IEEE, Volume: 18 Issue: 2, March-April 1998, Page(s): 59 –79

[73] Technical Brief, NVIDIA GeForce FX GPUs Cinematic Computing for every User: http://www.nvidia.com/object/overview_tb.html

[74] Noel D. Scott, Daniel M. Olsen and Ethan W. Gannett, An Overview of the VISUALIZE fx Graphics Accelerator Hardware, *The Hewlett-Packard Journal* Article 4 • 1998

[75] Burger, D.; Goodman, J.R.; Kagi, A.; Limited bandwidth to affect processor design, *Micro, IEEE*, Volume: 17 Issue: 6, Nov.-Dec. 1997, Page(s): 55 –62

[76] I. Antochi, B.H.H. Juurlink, A. G. M. Cilio, P. Liuha, Trading efficiency for energy in a texture cache architecture, *Proceedings of the 2002 Euromicro conference* on Massively-parallel computing systems, pp. 189-196, Ischia, Italy, April 2002

[77] Michael Cox, Narendra Bhandari, Michael Shantz, Multi-Level Texture Caching for 3D Graphics Hardware, *ACM SIGARCH Computer Architecture News, Proceedings of the 25th annual international symposium on Computer architecture*, Volume 26 Issue 3, April 1998

[78] Ziyad S. Hakura, Anoop Gupta, The Design and analysis of a Cache Architecture for Texture Mapping, *ACM SIGARCH Computer Architecture News, Proceedings of the 23rd annual international symposium on Computer architecture*, Volume 24 Issue 2, may 1992

[79] Homan Igehy, Matthew Eldridge, Kekoa Proudfoot, Prefetching in a Texture Cache Architecture, *Proceedings of the 1998 EUROGRAPHICS/SIGGRAPH workshop on Graphics hardware*

[80] *Alexis Vartanian, Jean-Luc Bechennec, Nathalie Drach-Temam,* The Best Distribution for a Parallel OpenGL 3D Engine with Texture Caches, High-Performance Computer Architecture, 2000. HPCA-6. *Proceedings. Sixth International Symposium on*, 8-12 Jan. 2000, Page(s): 399 –408

[81] Homan Igehy, Matthew Eldridge, Pat Hanrahan, Parallel Texture Caching, Proceedings of the 1999, *Eurographics / SIGGRAPH workshop on Graphics hardware*, July 1999

[82] Molnar, S.; Cox, M.; Ellsworth, D.; Fuchs, H.; A sorting classification of parallel rendering, *Computer Graphics and Applications, IEEE*, Volume: 14 Issue: 4, July 1994, Page(s): 23 –32

[83] Mueller, C.; Hierarchical graphics databases in sort-first, Parallel Rendering, 1997. PRS 97. *Proceedings. IEEE Symposium on*, 20-21 Oct. 1997, Page(s): 49 -57, 117

[84] Vartanian, A.; Bechennec, J.-L.; Drach-Temam, N.; The best distribution for a parallel OpenGL 3D engine with texture caches, High-Performance Computer Architecture, *2000. HPCA-6. Proceedings*. Sixth International Symposium on, 8-12 Jan. 2000, Page(s): 399 -408

[85] Henry Fuchs, John Poulton, John Eyles, Trey Greer, Jack Goldfeather, David Ellsworth, Steve Molnar, Greg Turk, Brice Tebbs, Laura Israel, Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories, *ACM SIGGRAPH Computer Graphics, Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, Volume 23 Issue 3, July 1989

[86] Tong-Yee Lee; Raghavendra, C.S.; Nicholas, J.B.; Image composition schemes for sort-last polygon rendering on 2D mesh multicomputers, *Visualization and Computer Graphics, IEEE Transactions on*, Volume: 2 Issue: 3, Sept. 1996, Page(s): 202 –217

[87] Don-Lin Yang; Jen-Chih Yu; Yeh-Ching Chung; Efficient compositing methods for the sort-last-sparse parallel volume rendering system on distributed memory multicomputers, *Parallel Processing, 1999. Proceedings. 1999 International Conference on*, 21-24 Sept. 1999, Page(s): 200 –207

[88] Bor-Sung Liang, Yuan-Chung Lee, Wen-Chang Yeh, and Chein-Wei Jen,"Index Rendering: A Hardware-Efficient Architecture for 3-D Graphics", *Proceeding, The 10th VLSI Design/CAD Symposium*, p.p.137-140, Nantou,

[89] Larry Bergman, Henry Fuchs, Eric Grant, Susan Spa,Image Rendering by Adaptive Refinement, ACM SIGGRAPH Computer Graphics , *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, Volume 20 Issue 4, August 1986

[90] Youngkwan Cho; Neumann, U.; Jongwook Woo; Improved Specular Highlights with Adaptive Shading, *Computer Graphics International*, 1996. Proceedings, 24-28 June 1996, Page(s): 38 –46

[91] Paul Heckbert and Michael Garland, Multiresolution Modeling for Fast Rendering, *Graphics Interface'94*

[92] Atwood, B.; Ishii, T.; Osabe, T.; Mine, T.; Murai, F.; Yano, K.; SESO memory: a CMOS compatible high density embedded memory technology for mobile applications, *VLSI Circuits Digest of Technical Papers*, 2002. Symposium on, 13-15 June 2002, Page(s): 154 –155

[93] Yong-Ha Park; Ramchan Woo; Sun-Ho Han; Jung-Su Kim; Se-Joong Lee; Jeong-Hun Kook; Jae-Woon Lim; Hoi-Jun Yoo; 7.1 GB/sec bandwidth 3D rendering engine using the EML technology, *VLSI and CAD, 1999. ICVC '99. 6th International Conference on*, 26-27 Oct. 1999, Page(s): 277 –280

[94] Chong-Lii Hwang; Kirihata, T.; Wordernan, M.; Fifield, J.; Storaska, D.; Pontius, D.; Fredernan, G.; Ji, B.; Tomashot, S.; Sang Dhong; A 2.9ns random access cycle embedded DRAM with a destructive-read, *VLSI Circuits Digest of Technical Papers, 2002*. Symposium on, 13-15 June 2002 Page(s): 174 –175

[95] Chi-Weon Yoon; Ramchan Woo; Jeengheon Kook; Se-Joong Lee; Kangmin Lee; Hoi-Jun Yeo; An 80/20-MHz 160-mW

multimedia processor integrated with embedded DRAM, MPEG-4 accelerator and 3-D rendering engine for mobile applications, *Solid-State Circuits, IEEE Journal of, Volume*: 36 Issue: 11, Nov. 2001, Page(s): 1758 –1767

[96] Reiner Hartenstein, A Decade of Reconfigurable Computing: a Visionary Retrospective, *Proceedings of the Design, Automation, and Test in Europe (DATE '01)*, March 2001

[97] Lew, A.; Halverson, R., Jr.; A FCCM for dataflow (spreadsheet) programs, *FPGAs for Custom Computing Machines, 1995. Proceedings. IEEE Symposium on*, 19-21 April 1995, Page(s): 2 -10

[98] Bernardo Kastrup, Automatic Synthesis of Reconfigurable Instruction Set Accelerators, Promoters: prof. dr. ing. J.A.G. Jess, prof.dr.ir. J.L. van Meerbergen. *TU/e*, ISBN 90-74445-50-0, 22 May 2001, pp. 1-129.

[99] Villasenor, J.; Hutchings, B.; The flexibility of configurable computing, *Signal Processing Magazine, IEEE*, Volume: 15 Issue: 5 , Sept. 1998, Page(s): 67 –84

[100]Jose O. Cadenas, Graham M. Megson and Toomas P Plaks. Quantitative evaluation of three reconfiguration strategies on FPGAs: A case study. In *HPC-Asia 2000. Proc. of the Fourth Int. Conf. on High-Performance Computing in the Asia-Pacific Region, 14--17 May, 2000, Beijing, China.* Vol. I, pages 337--342. IEEE Computer Society Press, 2000.

[101]Heron, J.P., Woods, R., Sezer, S., and Turner, R. H., Development of a Run-Time Reconfiguration System with low reconfiguration overhead, *the Journal of VLSI Signal Processing, Special issue on Re-configurable Computing, vol. 28 (1/2), pp 97-113, May 2001.*

[102]Wirthlin, M.J.; Hutchings, B.L.; Improving functional density using run-time circuit reconfiguration [FPGAs] *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, Volume: 6 Issue: 2, June 1998 Page(s): 247 -256

[103]Michael Flynn, Technology trends and adaptive Computing, FPL 2001,LNCS 2147, pp 1-5, 2001

[104]*Scalera, S.M.; Vazquez, J.R.;* The design and implementation of a context switching FPGA, *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on*, 15-17 April 1998 Page(s): 78 –85

[105]S. Trimberger, D. Carberry, A. Johnson, J. Wong Xilinx Inc., A time-multiplexed FPGA, 5th *IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM '97),* April 16-18 , 1997, Napa Valley, CA.

[106]G.K. Kuzmanov, G. N. Gaydadjiev, S. Vassiliadis, Loading rm-code: Design Considerations, *Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation*, pp. 8-11, Samos, Greece, July 2003

[107]Scott McMillan and Steven A. Guccione. Partial Run-Time Reconfiguration Using JRTR. In R. W. Hartenstein and H. Grunbacher, editors, Field-Programmable Logic and Applications, pages 352-360. *Springer-Verlag*, Berlin, August 2000. Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications, FPL 2000

[108]James Stephen Soe Mein Wong, Microcoded Reconfigurable Embedded Processors, *Thesis dissertation, TUDelft- The Netherlands*, December 2002, ISBN: 90-9016380-8

[109]Georgi Kuzmanov and Stamatis Vassiliadis, Arbitrating Instructions in an ρμ-coded CCM, *13th International Conference on Field Programmable Logic and Applications Lisbon - Portugal, September 1-3, 2003*

[110]IBM, PowerPC 405 Core, Publication Number: SA14-2339-04, Revision Date: 12/03/01, IBM Corporation, 2001.

[111]P. L. Watten, J. P. Ewins, M. White, M. D. J. McNeill and P. F. Lister, *'A Digital Arithmetic ToolKit for Graphics Hardware Design '*, submitted to *The Twelfth International Symposium on Computers and Information Sciences (ISCIS-XII),* Antalya, Turkey, pp85-91, November 1997.

[112]M.MeiBner, U.kanus et al, VIZARD II: A reconfigurable Interactive Volume Rendering System, *Proceedings of the conference on Graphics hardware*, September 2002

[113]Pavel Zemcik, International Conference on Computer Graphics and Interactive Techniques, *Proceedings of the 18th spring conference on Computer graphics Budmerice*, Slovakia. 2002 ISBN:1-58113-608-0

[114]*Shih-Ching Ou; Li-Hong Shiu; Sung-Jung Hsiao; Wen-Tsai Sung;* Accelerate the calculation of NURBS curves and surfaces based on parallel architecture, *Parallel and Distributed Systems, 2002. Proceedings. Ninth International Conference on* , 17-20 Dec. 2002 ,Page(s): 245 –250

[115]Laurent Moll, AlanHeirich and Mark Shand, Sepia: scalable 3D compositing using PCI pamette, *Field-Programmable Custom Computing Machines, 1999. FCCM '99. Proceedings. Seventh Annual IEEE Symposium on,* 21-23 April 1999 Page(s): 146-155

[116]Heirich, A.; Moll, L.; Scalable distributed visualization using off-the-shelf components, *Parallel Visualization and Graphics Symposium, 1999. Proceedings*. 1999 IEEE, 25-26 Oct. 1999, Page(s): 55 –118

[117] Guangming Lu; Singh, H.; Ming-Hau Lee; Bagherzadeh, N.; Kurdahi, F.J.; Filho, E.M.C.; Castro-Alves, V.; The MorphoSys dynamically reconfigurable system-on-chip, Evolvable Hardware, 1999. *Proceedings of the First NASA/DoD Workshop on*, 19-21 July 1999, Page(s): 152 –160

[118]I. Damaj, and H. Diab, "Performance analysis of linear algebraic functions using reconfigurable computing," *The Journal of Supercomputing*, Vol. 24, No. 1, pp. 91-107, 2002.

[119]Jeongseon Euh, Power-Aware 3D Computer Graphics Rendering System, *Ph.D. dissertation, University of Massachusetts Amherst*, August 2002.