NURBS Interpolation with FPGA Acceleration

Luiz Marcelo Chiesse da Silva
Federal University of Technology - Paraná
Electrical Engineering Department
Cornélio Procópio - Paraná - Brazil
chiesse@utfpr.edu.br

Maria Stela Veludo de Paiva
USP - University of São Paulo
Electrical Engineering Department
São Carlos - São Paulo - Brazil
mstela@sel.eesc.usp.br

Abstract

NURBS are used mainly to represent 3D models, being included in standards like OpenGL (Phigs) and used in computer graphics and manufacturing systems, representing adequately, in a compact way, almost any shape. For this sake, NURBS algorithms are implemented in graphical processing units (GPUs) hardware, but its rigid processing architecture and power consumption makes it inadequate in embedded systems implementations. An alternative to this case, instead of another technologies like microcontrollers or dedicated integrated circuits, is the use of reconfigurable logic with FPGAs, more adequate to embedded systems in relation to GPUs. This work proposes a SoC - System on a chip implementation, using FPGA and the Verilog synthesis language, aiming implement an embedded system for computer graphics NURBS interpolation tools and visualization.

1. Introduction

Curve or surface interpolation is a fundamental task in graphic systems, like CAD/CAM, for example, where the resolution between the design and the manufacturing systems should be adjusted [17]. When a set of points are given or received from a graphical unit, and is essential to fit this set with a curve or straight lines coincident with the given points, is done the interpolation (if the points are not coincident, the fitting is made by approximation) [14]. There are several methods for the interpolation of a set of points, ranging from simple and efficient triangulations [8] to modified methods using RBF [3] and others [7]. Among these methods, NURBS - Non Uniform Rational B-Splines are adopted in graphic standards like IGES [11], STEP [1] and OpenGL [9] (PHIGS) for curve and surface representation between graphical systems. The main advantages of the rational b-splines (affine transformations, for example) make them the most suitable choice for standardization, despite the lack of compression in the representation of conic sections [13], is widely used too in generic mathematical applications. The use of piecewise polynomials require a minimal number of procedures, namely orderly parameterization, linear system solution and the curve/surface fitting. Manufacturing systems, like CNC, and 3D data acquisition systems makes use of NURBS to provide greater efficiency, being implemented in embedded systems [10] based in microcontrollers, DSPs or a application specific integrated circuit. The FPGA technology provide an all-in-one chip solution for the data pre-processing and control in embedded systems, an architecture specified by the system designer, and reconfigurable logic, capable to perform a custom processor [16] for a specific task. This work proposes a Soc in FPGA system, with NURBS local curve and surface interpolation cores, based in the fast Cox-de Boor implementation [18], and a basic graphic pipeline [6], for visualization purposes. Optionally, is included two cores for the generation of straight lines and circles [2]. A Wishbone based bus [12] is used to connect and synchronize the cores, following its conventions of a open source logic bus. The use of FPGAs in the area of video and image processing is consolidated [4], despite the fact of current technologies in this area of application, like the Cell processor [15] [5] integrated circuit, there is a gap in the graphics processing that leads to applications making use of mixed technologies, like GPGPUs and FPGAs working together.

2. NURBS

A NURBS curve of degree $p$ is a piecewise polynomial curve defined as:

$$C(u) = \sum_{i=0}^{n} w_i P_i N_{i,p}(u) \tag{1}$$

where $u$ is the parameter value, $P_i$ form the so called control polygon points, weighted by $w_i$, and $N_{i,p}(u)$, $i=0,...,n$, are the B-spline basis functions defined over the knot vec-
tor U, where:

\[ U = \{ u_0, u_1, \ldots, u_m \} , \ u_i \leq u_{i+1}, \ i = 0, \ldots, m - 1 \] (2)

\[ N_{i,p}(u) = \begin{cases} 1 & \text{if } u_i \leq u_{i+1} \\ 0 & \text{otherwise} \end{cases} \] (3)

\[ N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p}(u) \] (4)

We assume throughout this paper that the knot vector has the following form:

\[ U = \{ a, a, \ldots, a, u_p+1, u_p, \ldots, u_{m-p-1}, b, b, \ldots, b \} \] (5)

where, in most practical applications, \( a = 0 \) and \( b = 1 \). A NURBS surface of degree \((p,q)\) is defined similarly as:

\[ S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} u_{i,j} P_{i,j} N_{i,p}(u) N_{j,q}(v) \] (6)

where \( u \) and \( v \) are the parameter values in the longitudinal and isoparametric directions of surface construction, \( P_{i,j} \), \( i = 0, \ldots, n; j = 0, \ldots, m \), form the so-called control net of the surface. Points between the data points, \( P_{i,j} \), are obtained by the inversion in the matrix form of the NURBS equation (1). For both methods, it is necessary to set up the knots, calculating initial parameters values given by the chord length method:

\[ t_0 = 0; \ t_n = 1 \] (9)

\[ t_k = \frac{1}{L} \sum_{i=1}^{k} |D_i - D_{i-1}| \] (10)

2.1. Cox-de Boor Algorithm

The NURBS interpolation could be divided in local and global interpolation methods. The first constructs a curve by rational segments (rational polynomials), in the case of surfaces by rational patches, such that the endpoints of each segment are the given data points. Neighbor segments are joined with some continuity level between the junctions, with the curve construction proceeding segment wise. The global interpolation makes the curve as a whole, using all the given points in a matrix calculation, and the control points are obtained by the inversion in the matrix form of the NURBS equation (1). For both methods, is necessary to set up the knots, calculating initial parameters values given by the chord length method:

\[ u_0 = \ldots = u_p = 0; \ u_{m-p} = \ldots = u_m = 1 \] (11)

\[ u_{j+p} = \frac{1}{p} \sum_{i=j}^{j+p-1} t_i \] (12)

where \( t \) is the parameter in the equation (10). The values \( u_0 \) at \( u_p \) and \( u_{m-p} \) at \( u_m \) reflects the knot multiplicities required for the spline beginning and end conditions. The property of local control allows the local interpolation, given by the support region of the basis functions, restricting the influence of the basis functions only in a limited number of piecewise polynomials. So, many polynomials segments can be computed concurrently to generate the final curve, given the desired continuity at the polynomials junctions (knots), feature exploited by the de Cox-de Boor algorithm given in equation (13), where \( t \) is the parameter, \( u \) are the knots, \( P \) the control points for each layer (data points in the first layer), \( j \) the control point index in the layer, \( k \) the order of the polynomial segments and \( j \) the layer number.

\[ C_i'(t) = \left( 1 - \frac{t - u_i}{u_{i+k-j} - u_i} \right) P_{i+1}^{j-1} + \left( \frac{t - u_i}{u_{i+k-j} - u_i} \right) P_i^{j-1} \] (13)
The control point in a b-spline curve is the convex combination of another two control points in the previous layer, as illustrated in figure 2, and its influence in the curvature is given by the term in brackets in equation (13). If a control point and the respective knot are repeated, the curvature tends towards its position, until the curve pass through the point, resulting in the interpolation.

3. Graphics Pipeline

A basic visualization pipeline is used, and by the sequential nature, this process is divided in serial stages, whose number could vary between implementations, but follows the general arrangement given by fig 3. In computer graphics systems, the last three stages are managed by a API - Application Programming Interface. The vertex reader reads 3D data obtained from a cloud of points, in a form of a triple indicating the euclidean coordinates of the points. This data is stored in a RAM memory and, according to the primitive drawer cores, the required points are buffered in the FPGA. In the viewport transformation stage, the computation of each transformation matrix could be parallelized, but between them are serialized. The primitive drawer include the NURBS, straight line and circle generation cores. The video memory is made in a built-in RAM, for the purpose of different display resolutions suport and faster data transfer, executing the scanning to a D/A video converter independently of the remaining systems. The vertex reader is the data input of the pipeline, obtaining the data in the form of coordinates in euclidean space, from another built-in memory of the FPGA. The data could be inserted in the synthetization process, mapped from a graphical user interface to a memory initialization file. The primitive drawer is responsible for the effective data processing to originate the graphics, remaining to the viewport transformation maps the data to fit the video memory. The video memory writer send the data in 10 bits size for the video resolution of XGA (1024 columns by 768 lines), generating the timing synchroniza-

4. FPGA Implementation

Given the set of data points to be interpolated, the initial parameters for the knots are calculated by equations (9-10), with one core for euclidean distance between the points, and the knot vector is calculated by the core of equation (11-12), as demonstrated in the diagram given by figure 6, excluding the enabling gates from the bus signals, wich increments in two clock cicles each core processing (the blocks between dashed lines are parallelized). For each parameter \( t \) is given a core that recursively calculates the point \( C \) in the curve by the Cox-de Boor algorithm, with the recurrence for interpolation of the data points. In figure 7 there is an example for the generation of one point by the Cox-de Boor algorithm for a degree 3 NURBS, resulting in 3 cores layers, showing that the degree determines the number of layers,
and this core is repeated for each parameter (point) generated in the curve. If the control points and the respective knot vector are given instead of the data points to be interpolated, is used only the Cox-de Boor core to generate the NURBS curve, without the repetition for interpolation. A wishbone bus is used to connection, with a round-robin arbiter to control the cores requesting. Two cores are included for straight line and circle generation, that has compact and efficient representations in relation to NURBS. For the FPGA system working as a accelerator, the data networking is made by a Nios II embedded processor core, i.e., a core acting as a general purpose microcontroller. The FPGA IDE environment provides a interface specific to custom processor design, allowing data bus width, performance and peripherals configuration, being used to set the communication with an ethernet interface. The Nios II is designed with a 8 instruction set, only for the data communication over a point to point ethernet link, in this case for the FPGA system and a computer. An C++ console application was developed and used for the data transfer between a PC computer and the FPGA system, being possible too constrain the number of parameters sent to the FPGA due to limitation in the number of logic elements.

5. General Purpose GPU

The Graphics Processing Units was created with the multicore processing capability, but the first ones was made for the computer graphics processing applications only. Actually, it has an architecture made for the multidata processing, open to the programmers and suitable for high performance computing, so that the manufacturers allows actually GPGPUs with more than 500 cores devoted to general purpose applications. The use of GPU here is based in the CUDA implementation of NURBS, that consists in a multithread processing of the Cox-de Boor and knot addiction algorithm to the generation for each parameter in the curve/surface, despite the devoted circuits in the board. The performance is compared with the FPGA implementation, up to 16 cores, being made in CUDA-C language. The CUDA implementation follows these items:
- dividing the task in blocks, each of them consisting of a thread, defining 8 to 32 threads for each block;
- passing the serial processing to the computer processor;
- 16 bits data size (compatible with the FPGA system).

The GPU used has 2 multiprocessors with 8 CUDA cores each (enabling 16 processors). The NURBS is implemented multithreading each layer of the Cox-de Boor algorithm, just like the FPGA implementation, regarding the hardware limitation.
Table 1. Total clock cycles for 32 interpolation points and 100 parameters (p is the NURBS degree).

<table>
<thead>
<tr>
<th></th>
<th>CPU Vectorized code</th>
<th>GPU CUDA</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>NURBS interpolation</td>
<td>47p(p+1)</td>
<td>16p</td>
<td>8p</td>
</tr>
<tr>
<td>Visualization pipeline</td>
<td>12p+2</td>
<td>4p</td>
<td>10p+20</td>
</tr>
<tr>
<td>Knot addiction (16 knots)</td>
<td>10p</td>
<td>16p</td>
<td>4p</td>
</tr>
</tbody>
</table>

Table 2. Cores and number of logic elements.

<table>
<thead>
<tr>
<th>Core</th>
<th>LEs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex reader</td>
<td>57</td>
</tr>
<tr>
<td>Viewport transformation</td>
<td>110</td>
</tr>
<tr>
<td>NURBS interpolation</td>
<td>148</td>
</tr>
<tr>
<td>Straight line</td>
<td>24</td>
</tr>
<tr>
<td>Circle</td>
<td>48</td>
</tr>
<tr>
<td>Video memory writer</td>
<td>62</td>
</tr>
<tr>
<td>Wishbone</td>
<td>54</td>
</tr>
</tbody>
</table>

6. Results

The NURBS local interpolation algorithm and the visualization system for FPGAs, is compared with a single GPU by the number of clock cycles for each core. The GPU processing still results in a more efficient manner to deal with the data interpolation, being a dedicated circuit, designed to optimally perform graphic functions. The cores are synthesized to perform similar functionality like the GPU, with 16 simultaneous threads and an independent clock counter synchronized with the beginning and end of the process.

7. Conclusions

The use of FPGAs in computer graphic is still incipient, being confirmed despite the fact that circuits dedicated to this aim leading to the following items:

1. while the GPU is a highly specialized processor that can get great performance (for a specific subset of the problems), actually most of them are not suitable for embedded applications in respect to FPGAs due to the power dissipation, sometimes requiring more cooling than computer processors;
2. the GPU is limited by the built-in hardware and firmware, despite of the multiprocessing power;
3. the processors based in the traditional computer architecture, are restricted by the demand of a higher clock frequency, giving arising to multicore processors. PLAs technologies didn’t reach the higher frequency, being developed with higher densities too.

Actually, General Purpose GPUs (GPGPUs) provide more flexibility to the system designer, still locked to the hardware architecture, being some operations, like fixed point operations, efficiently done in FPGAs. Future works are devised to match reconfigurable systems with GPGPUS.

References

