Feasibility of Making E-blocks for Computer Graphics

Theo M. Verelst*
www.theover.org

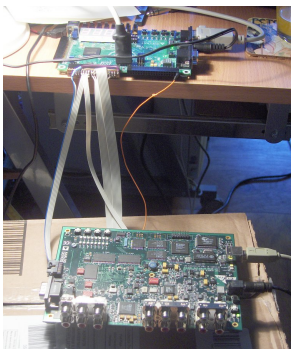

Figure 1: Two demo boards with DSP and FPGA connected.

Abstract

In line with the quest for chips which can perform graphics computation acceleration we experimented with and implemented some working graphics computation blocks and tested their function and to some extend their speed using readily available DSP and FPGA (programmable logic) demo boards, which are connected over a self made bus. It is made credible that with inexpensive means sensible graphics acceleration can be achieved which probably can compete with or add to state of the art consumer graphics computer systems. The used graphics computation primitives are chosen in line with the work of [Verelst 1991] in terms of thinking about Bezier surfaces as alternative graphics primitive throughout design and rendering phases and is direct extension of the ideas presented by [Pulleyblank and Kapenga 1987], though in parallel (not serial) unit computation sense.

CR Categories: C.3 [Computer Systems Organization]: Special-Purpose and Application-Based Systems—Real-time and embedded systems; C.3 [Computer Systems Organization]: Special-Purpose and Application-Based Systems—Signal processing systems G.2 [Mathematics of Computing]: Discrete Mathematics—Applications

Keywords: graphics hardware, prototypes, bezier, subdivision, FPGA, DSP

1 Introduction

Many graphics machines, interfaces and programming methods already exist, most current machines are based on polygonal rendering acceleration or general purpose computers. Other systems are usually exotic or out-dated because of the pace in which electrical engineering allows new hardware to keep up with Moore's law.

So it pays to look for ways to make interesting graphics (and sound) rendering architectures last with time, so that the latest chip technology with existing software expands the horizon of graphics machines.

Open and Free Software have proved strong for keeping general audience computer standards up, open hardware is also emerging. Standard blocks like adders and multipliers, dividers, transform units are becoming available on the internet in standard hardware description languages like VHDL and Verilog.

In the time of microcontrollers and special purpose chips the electrical engineering world is showing a tendency to like E-blocks: units with clear connections and function which can be connected together.

This is a bit of a weak spot in the world of electronics, computers and computer hardware: interfaces, drivers, and standard hardware blocks are not so readily available for strong or advanced use unless maybe for commercial R&D funding levels, but the strength of the Free and Open Source thought is clearly being proven in Linux and many projects, also in graphics (OpenGL/Mesa, X.org, Radiance, POVRay, OpenSceneGraph, etc.) to provide a large audience with quality programs, and counter the closing up and claiming attitudes of giants like Microsoft.

At fairly low budget, we've looked at the combination of a Blackfin DSP and a Spartan-3 Field Programmable Gate Array both on demonstration boards which can readily be purchased. The main target was to ascertain the communication can work with a self made bus connection, and that a relevant graphics computation can be accelerated on the FPGA, in this case a Bezier subdivision with result selection and bounding box data, at significant speed.

The design tools come with the demo boards and are used without special licenses or extra tools for programming and debugging. In addition the main algorithms to render images with subdivision of Bezier curves are shown to work in a C program on cygwin (windows) and on Linux (64 bit).

*e-mail: theover@tiscali.nl

2 Bezier curves and surfaces and their application

Powerful graphics hardware for fair prices allows quite fast rendering of polygonal surfaces when curved surfaces are approximated on them, errors are often visible. Ray tracing and radiosity computations with ray tracing operations are usually not well accelerated by standard graphics engines, and require significant computing power even for fast machines and that doesn't get better with curved surfaces.

Bezier curves and surfaces , in this case cubic, are pleasant curved primitives, and are also suitable for interpolation.

In general Bezier curves, and by tensor construction also surfaces, are spanned by a number of control points, which form the curve by using Bernstein polynomials which multiply each point:

$$\mathbf{B}(t) = \sum_{i=0}^{n} \mathbf{P}_i \, \mathbf{b}_i^n(t), \quad t \in [0,1] \tag{1}$$

Where the Bernstein polynomials are defined as:

$$\mathbf{b}_i^n(t) = \binom{n}{i} t^i \, (1-t)^{n-1} \tag{2}$$

Written out for cubic case (n=3):

$$\mathbf{B}^3(t) = \mathbf{P}_0(1-t)^3 + 3\mathbf{P}_1 t(1-t)^2 + 3\mathbf{P}_2 t^2(1-t) + \mathbf{P}_3 t^3 \qquad t \in [0,1] \tag{3}$$

The *control points* can be of any dimension, 2 for plane curves, 3 for 3D curves, or 4 for homogeneous coordinates, which can to great advantage be used to create mathematically correct circular shapes.

The end point of such curve can be seen to coincide with P_0 and P_3, while the tangents at those points correspond with the connection lines to point 1 and 2.

Using tensors gives us 3 dimensional surfaces by multiplying two curves and seeing them as spanning the u and v surface coordinate space.

3 Subdivision

Bezier subdivision according to De Casteljau's algorithm is a fast operation to make two Bezier curves or surfaces from one. Every coordinate is divided with the same scheme, where each lower line element is the average of the elements left and right above it as is shown in figure 2.

```
i0   i1   i2   i3
   a    b    c
     d    e
       f
```

Figure 2: Bezier subdivision scheme.

so for instance b = (i1+i2)/2 , where i is incoming data. The resulting curves made from the coordinate components corresponding to (i0,a,d,f) and (f,e,c,i3) or surfaces (where the subdivision is 4 fold)

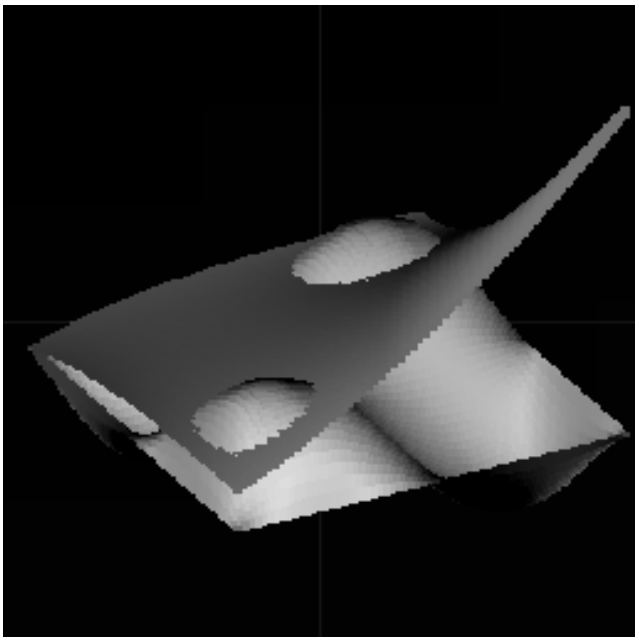

Figure 3: Rendering by flat projection with bounding boxes in 3D with z-buffer using 16 bits overall accuracy.

can be proven to together represent mathematically the same curve or surface as the original.

Figure 3 has been made with a C program based on repeated Bezier subdivision of 3D coordinates of cubic patches, in this case only 2: the top sheet and the bottom bulges.

The following is a piece from the C program to compute repeated 3D subdivisions to render a orthogonally projected image (available as Open Source software), where a single subdivision is computed in a loop:

```
a = (in[0][i][0][j] + in[0][i][1][j])/2 ;
b = (in[0][i][1][j] + in[0][i][2][j])/2 ;
c = (in[0][i][2][j] + in[0][i][3][j])/2 ;
d = (a+b)/2;
e = (b+c)/2;
f = (d+e)/2;
```

Coarse benchmarking of the programs was done on two machines, on a Dual Core Pentium machine (3GHz, dual bank memory, XP / Cygwin gcc C compiler version 3.4.4, using only one core) the image as shown took about 0.53 sec to compute, which contains 524256 patch subdivisions, which is 4 (u direction) x 8 (resulting 2 in v direction) x 3 (x,y,z coordinates) = 12 one dimensional subdivision steps. When increasing subdivision depth, 55.7 seconds is measured for 33554400 patch subdivisions, equaling about 400 million subdivision operations. On an Athlon 64 3.3GHz machine (moderate chipset, Fedora Core 4 64 bit / gcc C compiler 4.0.0), respectively 0.65 sec. and 56.5 seconds were measured. All these figures contain also time for flat projection and file reading and writing, but subdivisions consume most of the time.

For the larger number of subdivisions, the different PC's both do about 20 million subdivisions per second instead of 30 million, probably due to memory access over a larger memory range (less caching).

For the programmable logic, the basic subdivision block is in 16 bit form shown in figure 4.

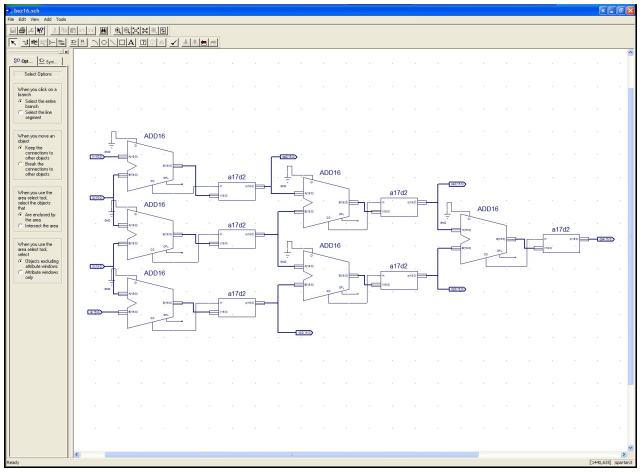

Figure 4: A 16 bit bezier subdivision unit

The wedge shaped units are 16 bit adders, which are followed by a shift right block, which also reads the carry bit from the adders. Data flows from left to right, all external connections are 16 bits wide.

4 Quick min/max in FPGA

To determine the size in each dimension a Bezier surface after subdivision the convex hull property can be taken in a looser way by using rectangular bounding boxes, which exactly encloses all control points, and are aligned with the coordinate axes. Every point on the surface can be proven to be inside the 3D bounding box.

To compute the bounding box, the minimum and maximum of the control points in each dimension (for instance xmin, xmax, ymin, ymax, zmin, zmax) is needed, for which a comparison operation is needed, which is possible reasonably efficient in terms of logic circuits, at least by doing a subtraction and sign check (equals a inversion, carry in and binary addition and MSB bit test) or, as here, a specialized binary number compare circuit of limited complexity.

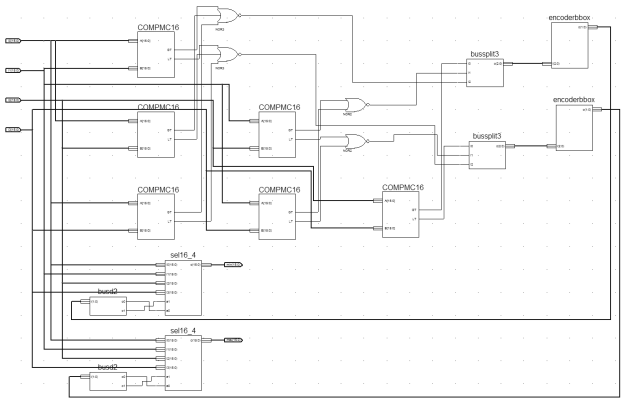

Figure 5: The Bounding Box computation unit for one dimension.

Figure 5 shows the designed one dimensional bounding box computation unit made of 6 comparators, 2 one out of four 16 bit wide selectors, and a priority encoder-like circuit.

The maximum length of the data path is a comparison, a selection and relatively fast combinatory logic circuit, so the result is available very quickly after the input is offered. The output is a minimum and a maximum 16 bit number, which is a selection from the inputs. The unit tests correct.

5 Developing DSP program, hardware blocks, and communication

The VisualDSP environment allows programming in C and debugging the dual core DSP on the evaluation board, and access to its memory and registers.

The (free) Webpack software has an integrated project manager, schematic digital block editor, VHDL block facility, and programs the demo board using a cheap and easy to use parallel port JTag interface, either for repeatedly programming the FPGA or burning the configuration in flash memory.

A bus connection has been prototyped between the DSP asynchronous bus interface and some twenty pins of the FPGA board with fitting connectors and series resistors on the side of the DSP to make sure the prototyping and testing process wouldn't blow anything up, this however has a speed impact.

The bus gives in the test configuration access to 8 memory mapped 16 bit readable and writable locations, currently running at roughly 20 Mega accesses per second, equivalent to 40MB/sec. Those accesses are uncached, and direct, that is unpipelined: the finest granularity is simply one read or write, with a little speed overhead for accesses spreading out over memory banks (but not when the DSP as normal runs from an internal fast memory). The speed is not up to lets say high end graphics buses, but for audio purposes its pretty high, and it can probably be upgraded by taking the current limit resistors out, shortening the wires, doubling the bus width to 32bit, and adjusting the bus timing parameters to the maximum 25 or 33 Mega accesses per second, giving in this case a maximum of over a hundred megabyte per second. The programmable logic should be up to at least 400 Megabyte per second on a 32 bit bus, which starts to get serious in comparison with existing graphics buses.

The DSP debug environment allows access in all kinds of number representation to memory locations, which is used to access the memory mapped addresses 16 bit at the time.

The DSP programming is done in C, which is well readable, and can be reasonable efficient, while care must be taken that compiler and assembler optimization can affect memory access order, and of course external memory access considerably slows down the DSP core(s).

To test the hardware and communicate with it, this is a fragment from the DSP main source code which writes a one dimensional Bezier curve to the Spartan , reads back immediately the outcome of the accelerators computations, in this case a single Bezier subdivision, binary compares it with a DSP based computation of the same formula with the same bit-width and counts the errors:

```
short bezsub1(a1,a2,a3,a4)
short a1,a2,a3,a4;
{
   return((((a1+a2)/2+(a2+a3)/2)/2
          +((a3+a4)/2+(a2+a3)/2)/2)/2);
}

...
    count = 0;
```

```
for (i=0; i<256; i++)
   for (j=0; j<32000; j++) {
      *(p1+0x80)  = 10000+j/2;
      *(p1+0x82)  = 32000-j;
      *(p1+0xa0)  = 67;
      *(p1+0xa2)  = 50+j;
      ssync();
      if (*(p1+0x80)  !=
          bezsub1(10000+j/2,32000-j,67,50+j))
         count++;
   }
* (unsigned short *) 0x2c000000 = 0x101;
printf("count %d\n", count);
```

When it's done with the 256 iterations of 32000 different input numbers and checks, it sets a LED display on the board to 1 using the same interface, and prints on the DSP IDE the number of errors, which after the bus interface speed had been set has only been zero during hundreds of tests, except when on purpose an error is introduced in the formulas.

The sync() is to make sure the DSP compiler or assembler doesn't change the order of memory accesses when packing up to 4 DSP instructions in a pipelined packed instruction, which would make the result mapped memory I/O location read before the multiple inputs are set. In this example, the memory accesses form a major part of the DSP processor time, and the single Bezier subdivision chain block in the FPGA responds in the time it takes to do a read after the last write to its inputs which is in the order of 50 nano seconds, which is probably very easy to keep up with, in practice no errors (because of delay) were observed.

No attempts were yet made to add functionality to the FPGA circuit to autoclock it and determine the upper speed bound. Also, there has not yet been made much use of it memory capabilities, except for multiple 16 bit latches/registers. The bounding box computation unit also has been tested in similar fashion as the above and was also faultless.

To determine the amount of computations that could be done by the FPGA (a 3200 slice one, 200,000 gate equivalent), 12 Bezier-select-bbox units have been put together in 3x4 array which can correspond to 4 repeats subdivision steps on 3 coordinates simultaneously, like in figure 6.

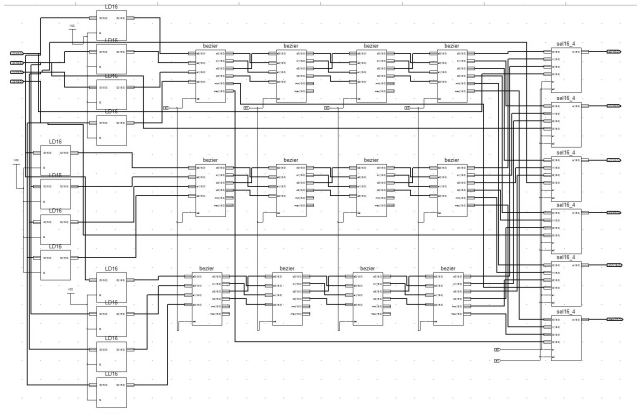

Figure 6: Impression of the structure of a test Bezier subdivision chain set from the actual prototype schematics.

On the left the 3 coordinates x 4 tuples x 16 bit = 192 bit buffered input is shown, which feeds the three subdivide chains with per unit a select between the output curves and a min/max coordinate output.

On the right selectors can simultaneously feed back 6 words to the Blackfin interface circuitry consisting of 4 resulting Bezier control points, and a minimum and maximum of these numbers from each of the three chains.

Main purpose was to check the design software and the xilinx with a larger number of units being used. Improvements are quite possible: selecting can be more efficient because only half the computations are needed, bounding boxes probably aren't needed everywhere like when there is no interest in having achieved a certain accuracy limit, and the chain idea is probably inefficient compared to fast (internal) memory storing intermediate results for more parallel chains.

The design software had to reiterate to fit the circuit with a little less speed and its preferable to use a heavy machine to run it on or interaction experimentation times are replaced by batch FPGA programming. After reiterating 99% of the slices were used and the circuit worked fine.

6 Analysis of actual and possible speed

The speed of the setup has been analyzed by running a DSP program in a loop, which accesses the programmed hardware during the course of execution of the inner loop, and which compares each result from the FPGA, which is read back over the memory mapped bus connection, bit for bit to make sure no computation or communication error has been made during each individual computation in the inner computation loop.

The FPGA can immediately after setting the last memory mapped register be read back for the result, so 50nS later. The memory accesses were tested to be the bottleneck in the DSP program.

According to the FPGA programming and simulation software, the overall longest path was in the order of 10 nSec so 100 Mega accesses per second should be possible when the bus wouldn't exhibit delay, but detailed analysis has not been done.

Bear in mind that the power consumption and form factor of these solutions are very small compared with for instance PCs and high end graphics cards. When well used, the post stamp size chip uses maybe a few watts, without need for extra cooling, and the FPGA is also very small, cheap, and uses (under higher load) maybe a few watts, too.

Efficient use of the Bezier blocks is possible when the FPGA controls the data flow, and can iterate itself, for instance to search intersections, or project using a Z-buffer. It would seem reasonable to assume a single Bezier subdivision and possibly added bounding box test can be done in 10nS. In fact, it is possible this can be even less, multiplies can even be done in less. That would make one subdivide unit faster than a current PC effectively seems to give, though of course parallelism could be used and the PC can compute in higher accuracy. Ten of such units would mean a possible order of magnitude speedup from a (relatively small) FPGA, assuming those units can compute sensible things, and assuming that in that situation like now temperature increase is absent or within limits.

Finally, taking numerical averaging accuracy into account, 16 bits can be subdivided maybe 8 times to still be able to compute normals from the results which are suitable for viewing. Assuming results converge properly, 16 bits can give a small error when the integer

range is well used, but it is likely that not all types of computations are satisfactory in 16 bits.

The design environment for the FPGA, and to an extend also for the DSP are not suitable for immedeate, short learning time use, and require considerable knowledge to use right and effective, though both come with examples. Maybe a good open source project can change that.

7 Conclusion

The setup at a low budget and with easy to obtain tools works, and can even be foreseen to be usable as an Open Source development solution with serious applications.

The subdivision operation is a useful graphics primitive which can be efficiently implemented in hardware, and in various forms be used for accelerating ray tracing or projection computations. It appears worth the effort of making the hardware and using it in practice after some programming and optimization.

The same signal processing with programmable hardware combination can also be used for audio purposes where the bus bandwidth is more than sufficient for relevant applications.

Testability and repeated programmability in combination with a PC are quite acceptable, while (limited) benchmarking indicates enough competitive edge with existing fast PCs.

References

PULLEYBLANK, R., AND KAPENGA, J. 1987. The feasibility of a vlsi chip for ray tracing bicubic patches. *IEEE Computer Graphics and Applications 7*, 3, 33–44.

VERELST, M. T. 1991. *A CAGD System Framework with Rational Cubic Bezier Surfaces as Graphics Primitives*. Master's thesis, Delft University of Technology, dept EE, Network Theory Section.