REALISATION OF PARALLEL 3D SCAN CONVERSION ON TMS320C80 AND BENCHMARKS*

Tolga Bayik, Mehmet B. Akhan

University of Hertfordshire
Faculty of Engineering and Information Sciences
Department of Electrical and Electronic Engineering
Hatfield, Herts AL10 9AB
United Kingdom
t.bayik@herts.ac.uk, m.b.akhan@herts.ac.uk

ABSTRACT

The need to parallelise the scan conversion stage is explained and the suitability of the TMS320C80 (C80) for parallelising the scan conversion is investigated. Wire frame object rendering is realised on the C80 and the benchmarks are given for the procedure.

Keywords: 3D Graphics, Rendering, Scan Conversion, Benchmarking, C80

1 INTRODUCTION

3D graphics rendering mainly involves two stages: transformation and rasterisation. In order to get an adequate performance, the use of multiple processors for both transformation and rasterisation is a powerful option. Current implementations of graphics pipeline use parallelism at certain stages of the rendering pipeline but there has been no serious attempt to parallelise the scan conversion stage using a powerful Digital Signal Processor (DSP) [Monla94, Hobbs93, Crow88].

In this study, parallelising the scan conversion using 4 parallel processors is aimed to accelerate rendering speed and the C80 has been chosen to be the working platform. Previous 3D graphics implementation using the C80 actually employed only one of the four processors [Poland96]. Also, there are no benchmarking figures available in the literature for parallel scan conversion using the C80. We aim to show how the C80 can be used to implement parallel scan conversion. The effect of increasing the number of parallel processors is also investigated.

* This work is supported by Texas Instruments Europe

2 SCAN CONVERSION

Converting the transformed 3D primitives into 2D pixel information on the screen is called scan conversion [Bayik97, Akhan97]. Scan conversion of 3D objects is implemented by interpolating between two projected end points. We usually have the primitives' vertex coordinates as the information and we want to interpolate between these vertex coordinates.

2.1 Parallel Scan Conversion Using C80

The C80 is used for parallelising the scan conversion stage. Other parts of the graphics pipeline were realised on a PC using MS Visual C++ (Ver 5.0).

The C80 has a RISC Master Processor (MP) with a floating point unit [Timp]. There are also four Parallel Processors (PP) or Advanced Digital Signal Processors (ADSP) which are 32 bit integer units supporting 2 billion operations per second [Tipp]. Interprocessor communication between the processors is implemented by a combination of RAMs that are interconnected via the crossbar to the processors [Tisl]. The transfer controller handles block data movement [Titc].

Data Parallelism is employed and polygons are partitioned among the parallel processors such that each PP runs the same code, hence does the same job. [Bayik97]. In this case, the MP sends commands and organises data flow to each of the parallel processors in a parallel data flow fashion. Figure 1 shows the realisation of 3D graphics pipeline on a PC and the C80 (MP and PPs).

In the next subsections, the implementation shown in Figure 1 is explained.

2.1.1 PC Part

First of all, communication between the PC and the C80 must be established and initialised. This was done using the host interface functions [Tist]. The PC then tells the MP to carry out its own initialisation and the MP starts to wait for data from the PC.

Meanwhile, the PC performs the stages before scan conversion. Whenever the PC finishes the calculations before scan conversion, the MP is signalled to indicate that the processed information is ready and it is passed to the Software Development Board (SDB) for the remaining stages.

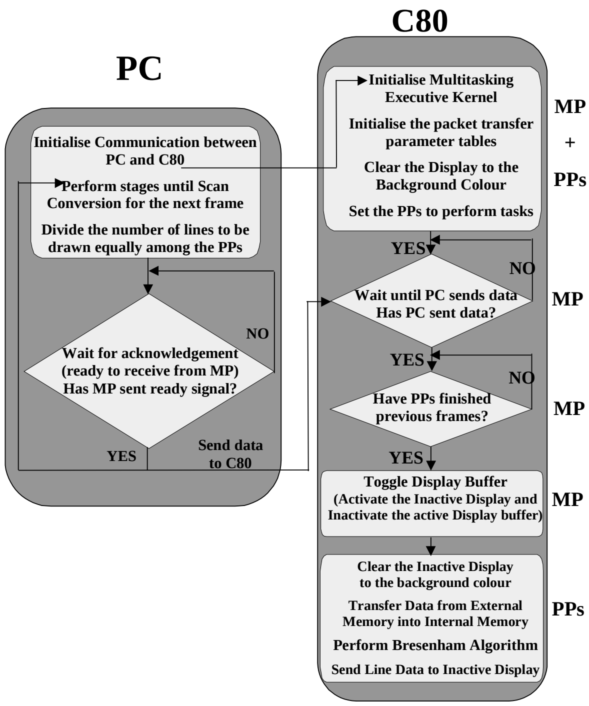

Figure 1 3D Graphics Pipeline Realisation

2.1.2 Transferring Data from PC to C80

The processed data is passed to the SDB for further processing. The SDB is a Peripheral Component Interconnect (PCI) plug-in card that contains the C80 processor and some other hardware components. It includes 8 Mbytes of Dynamic Random Access Memory (DRAM), 2 Mbytes of Video Random Access Memory (VRAM), Audio Codec, Video Capture circuitry, Video Display Circuitry, Memory Controller and the PCI Interface [Tipu97].

If the SDB is working properly, then the code to be run on the C80 is loaded and run. Then the transformed coordinates of the vertices are loaded to the C80 external memory starting from 0x80000000 using the PCI Bus as shown in Figure 2. All PCI FIFO accesses are 32 bits wide and require 2 cycles [Tipu97]. Therefore, the maximum PCI transfer rate turns out to be 80 Mbytes/s for a 40 MHz C80.

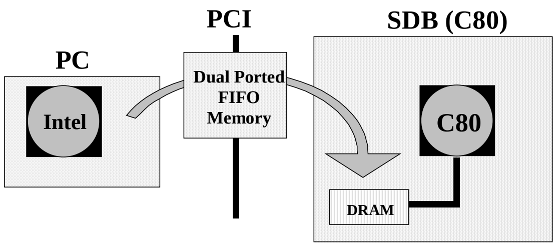

Figure 2 Transfer of Data to the C80

Address 0x80000000 actually corresponds to the C80's external memory where the DRAM resides on the SDB board. The DRAM is used for program and data storage generally.

The graph in Figure 3 shows the time required to transfer a certain amount of data to the C80's external memory. If the number of bytes transferred is small, then the transfer rate is very low. When the transferred number of bytes becomes 64KB, the transfer rate is almost constant and it does not change any more. There is a bottleneck discovered here as the expected rate can be as high as 80 MBytes/s. The library supplied by the C80 is used to interrupt the C80 and transfer the data to its external memory and should be improved for a better rate.

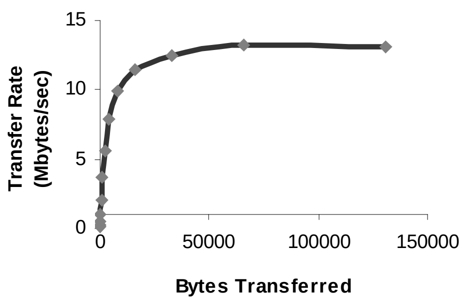

Figure 3 PCI Transfer Rate

2.1.2 MP Part

After transferring the data from the PC to the C80 external memory on the SDB, the MP takes control.

The MP initialises the multitasking executive, the command and argument buffers during the early initialisation stages. It then waits for a signal from the PC as shown in Figure 1. Whenever the PC has some data available to pass to the C80, the PC informs the MP. The PC tells the MP what to do and what the arguments for these operations are. Then depending on the arguments placed in the server command list through the PC, the MP issues the necessary commands to the PPs for further processing. It then waits for the PPs to finish their task. Whenever all the PPs finish their task, control goes back to the MP and it waits again for data or command to be sent by the PC.

The MP is also programmed to be responsible for toggling the display buffer. There are two display buffers on the SDB located at 0xC0000000 and 0xC0100000. At any given time, one of these buffers is programmed to be active and the other one inactive. Figure 4 through Figure 6 show how the display buffers are toggled and double buffering is employed.

The idea is to display the previously rendered frame on the screen by making it the active display buffer, while writing the data to the inactive display buffer as shown in Figure 4. There is no write to the active display buffer at any time and it is the buffer that is shown on the display.

Whenever the next frame is rendered completely in the inactive buffer as shown in Figure 5, it is time to toggle the buffers.

Then, as shown in Figure 6, the inactive buffer is made active and shown on the screen. At the same time, the active buffer is deactivated and drawing the next frame starts on this inactive buffer.

This double buffering scheme ensures that the object is not shown on the screen until all the data is rendered. If the double buffering scheme were not employed, the object would be drawn on the screen as soon as the pixel locations for that particular parts were calculated, which would cause flicker.

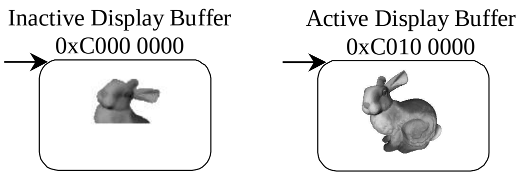

Figure 4 PPs are writing to the Inactive Buffer while Active Buffer is shown on the screen

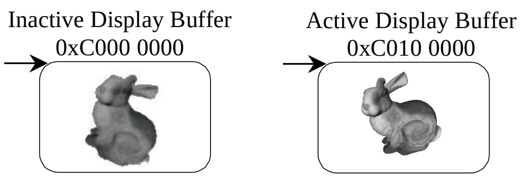

Figure 5 PPs have finished writing to the Inactive Buffer. Toggle the buffers now

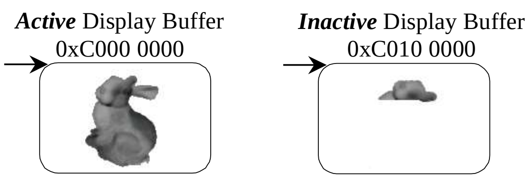

Figure 6 PPs are writing the next frame to the Inactive Buffer while the Active Buffer is shown on the screen

The ANSI C compiler that is provided with the C8x development environment is used for MP programming [Ticg].

2.1.3 PP Part

There are several functions written to be run on the PPs, which include clearing the screen, transferring the external data into the PPs' local RAMs, drawing lines and wireframe objects using Bresenham's Line Drawing Algorithm and submitting a packet transfer request for sending the scan converted data to the display buffer.

The C8x assembly language is used for programming the PPs.

Clearing the screen to the background colour using PPs

The display buffer needs to be initialised to the background colour before doing anything. This is achieved using the fill-with-value packet request, which is a special transfer mode issued to the TC [Titc].

Transferring External Data into PPS' local RAMS

After the screen is cleared, the data processed by the PC and put into the C80's external memory is brought into internal data RAM of each PP. The data neds to be in the internal memory of the C80 for faster access to data. A dimensioned packet request is used to bring external data into internal data RAM of each PP [Titc].

Drawing Lines and Wireframe Objects Using Bresenham's Line Drawing Algorithm

Bresenham's Line Drawing algorithm is modified to employ integers rather than fractions as the PPs are fixed point processors.

For drawing lines and wireframe objects, the fixed patch delta guided transfer type is used. In fixed patch delta guided transfers, the start address of each patch is guided from a guide table located in on-chip memory as shown in Figure 7 [Titc]. Data RAM0 and data RAM1 of each PP are used to store the delta guide tables in this case.

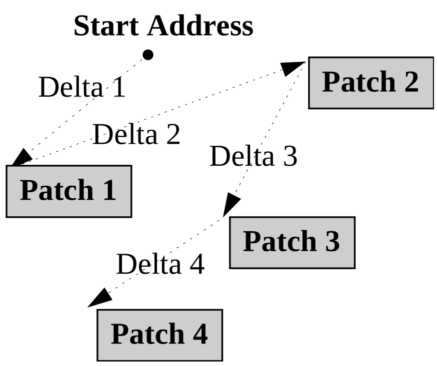

Figure 7 Fixed Patch Delta Guided Transfers

The PPs read the endpoints of each line and calculate the individual pixel locations using Bresenham's line drawing algorithm. Individual pixels are calculated using an offset from the previous pixel location. The first pixel to be illuminated is taken as the first end point of the line. Then, the calculated offsets are used to form the fixed patch delta guided table as shown in Figure 8.

Delta Guide Value 1
Delta Guide Value 2
Delta Guide Value 3
⋮
Delta Guide Value n

Figure 8 Fixed Patch Delta Guide Table for Line Drawing

After the line information is calculated and the guide tables are prepared, the lines are drawn on the screen using fill-with-value to fixed patch delta guided packet transfers [Titc].

2.2 Scan Conversion Using a PC only

To be able to realistically see the improvement achieved by using the C80, a graphics pipeline was implemented using a PC (Pentium 120 MHz) only without any DSP help. The following factors were compared:

- The time required to do coordinate transformation on the PC and the time required to do scan conversion on the C80 as shown in Figure 9.

- The time required to do both coordinate transformation and scan conversion on the PC as shown in Figure 10.

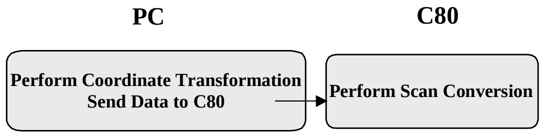

Figure 9 System 1 – Using Parallel Processing

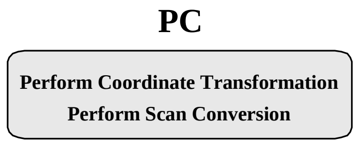

Figure 10 System 2 – No Parallel Processing

The performance figures based on these two systems will be presented in the next section.

3 RESULTS

The timings are found using the timer library function of MS Visual C++ (Ver 5.0) as shown in Figure 11.

```
start=clock();
……
……
stop=clock();
benchmark= (stop - start) / CLOCKS_PER_SEC;
```

Figure 11 Time Measurement

The clock time is assigned to start before the operations that are to be benchmarked and to stop after the procedure is complete. Then the difference is divided by the frequency of the system clock.

Figure 12 summarises the performance figures in terms of polygons rendered per second with varying object complexity and varying number of parallel processors.

Using a large range of objects, it has been observed that a PC can outperform the power of parallel processing only if one parallel processor is used. The performance of PC comes close to that of two PPs at around 20k polygons but it then decreases again.

One important thing to note is that the performance hardly depends on the complexity of the object when the PC and C80 are used together. The performance of the system when parallel processing is used does not depend on the complexity of the object as it did in the case of without parallel processing. This means that when the C80 is used for parallel processing, the performance will be quite robust and will not differ a lot for different objects of different complexity.

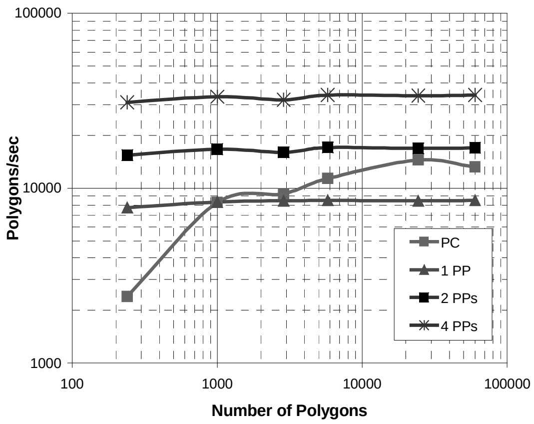

Figure 12 Performance Comparisons

4 CONCLUSIONS AND FUTURE WORK

The C80 was used for parallelising the scan conversion stage using data parallelism. Each PP worked on a single polygon and the PPs were used in parallel to make the scan conversion stage faster. The master processor distributed tasks and organised the data flow. Wire frame object rendering was realised and the benchmarks were given for the procedure.

Our work that is capable of rendering wireframe objects shows that 3D rendering will significantly benefit from parallelisation. As the number of parallel processors increases, the speed of the rendering is increased almost linearly. However, this may not be true after increasing the number of parallel processors beyond a certain number. Using multiple C80 boards and more parallel processing power, the optimum number of processors should be found.

The data transfer rate from the PC to the C80 should be increased by carefully investigating the library subroutines supplied with the C80 and modifying them.

In this study, only wireframe object rendering is performed. This should be extended to perform full rendering pipeline including shading and texture mapping to add more realism.

ACKNOWLEDGEMENTS

The authors would like to thank University of Hertfordshire Electronics and Electrical Engineering Department and Texas Instruments Europe for their support of this work. Special thanks go to Robert Owen, Texas Instruments, Northampton for his continuous support and encouragement. The authors would also like to express their appreciation to David Lauder for his valuable suggestions, helpful comments and proofreading.

REFERENCES

[Bayik97] Bayik, T. “3D Graphics Acceleration Using Data Parallelism On A Multiprocessor Architecture”, Proceedings of DSP UK’97, London December 1997

[Akhan97] Akhan, M. B. and Bayik, T., “3D Graphics Acceleration On A Multiprocessor Architecture”, Proceedings of ICSPAT ’97, San Diego, USA September 1997

[Clark80] Clark, J. “A VLSI Geometry Processor for Graphics”. Computer, Vol. 13, No. 7, July 1980, 59-68

[Clark82] Clark, J. “The Geometry Engine: A VLSI Geometry System for Graphics”, Computer Graphics, Vol. 16, No. 3, July 1982, 127-133

[Crock] Crockett, T. W. “Parallel Rendering”, Encyclopedia of Computer Science and Technology, Vol. 34, Supp. 19, A. Kent and J. G. Williams, eds.

[Crow88] F. C. Crow, “Parallelism in Rendering Algorithms”, Proceedings of Graphics Interface ‘88, pp. 87-96

[Foley90] Foley J., A. van Dam, “Computer Graphics Principles and Practice”, Addison Wesley, 1990

[Hobbs93] T. Hobbs et. al., “A Parallel Image Processing and Display System (PIPADS) Hardware Architecture and Control Software”, Proceeding of the Twenty Sixth Hawaii International Conference on System Sciences, 1993, pp.106-115

[Lee] T. Lee and C. S. Raghavendra, “Parallel Processing for Graphics Rendering on Distributed Memory Multicomputers”, Washington State University Pullman, WA

[Monla94] S. Monlar et. al., “A Sorting Classification of Parallel Rendering”, IEEE Computer Graphics and Applications”, pp.23-32, July 1994

[Poland96] Poland, S. TMS320C8x (DSP) Fundamental Graphic Algorithms Application Book. Texas Instruments, Digital Signal Processing Solutions, 1996.

[Roger88] Rogers, D.F., “Procedural Elements for Computer Graphics”, McGraw Hill, 1988.

[Ticg] *TMS320C80 (MVP) Code Generation Tools User’s Guide*, Texas Instruments Microprocessor Development Systems, 1995

[Time] *TMS320C80 (MVP) Multitasking Executive User’s Guide*, Texas Instruments Digital Signal Processing Products, 1995

[Timp] *TMS320C80 (MVP) Master Processor User’s Guide*, Texas Instruments Digital Signal Processing Products, 1995

[Tipp] *TMS320C80 (MVP) Parallel Processor User’s Guide*, Texas Instruments Digital Signal Processing Products, 1995

[Tipu97] *TMS320C8x Software Development Board Programmer’s Guide*, Texas Instruments Digital Signal Processing Solutions, *Literature Number: SPRU178, January* 1997

[Tisl] *TMS320C8x System-Level Synopsis*, Texas Instruments Digital Signal Processing Products, 1995

[Tist] TMS320C8x Software Development Board Technical Reference, Preliminary, Texas Instruments Microprocessor Development Systems, 1995

[Tisu] TMS320C8x Software Development Board User’s Guide, Preliminary Texas Instruments Microprocessor Development Systems, 1995

[Titb] *TMS320C80 Multimedia Video Processor (MVP) Technical Brief*, Texas Instruments Digital Signal Processing Products, 1994

[Titc] *TMS320C80 (MVP) Transfer Controller User’s Guide*, Texas Instruments Digital Signal Processing Products, 1995

[Watt94] Watt, A., “3D Computer Graphics”, Addison Wesley, 1994.