Faster Scan Conversion Using the TMS320C80

Authors: M.B. Akhan, T. Bayik, E.G. Bahari

ESIEE, Paris

September 1996
SPRA330

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Figures

Faster Scan Conversion Using the TMS320C80

Abstract

This application report describes the process of parallelizing the scan conversion stage using the Texas Instruments (TI™) TMS320C80 digital signal processor (DSP). Current approaches are discussed and a new approach is suggested to make the graphics pipeline faster, especially at the scan conversion stage. The suitability of the TMS320C80 for the suggested approach is investigated and the design philosophy is explained.

This document was part of the first European DSP Education and Research Conference that took place September 26 and 27, 1996 in Paris. For information on how TI encourages students from around the world to find innovative ways to use DSPs, see TI's World Wide Web site at www.ti.com.

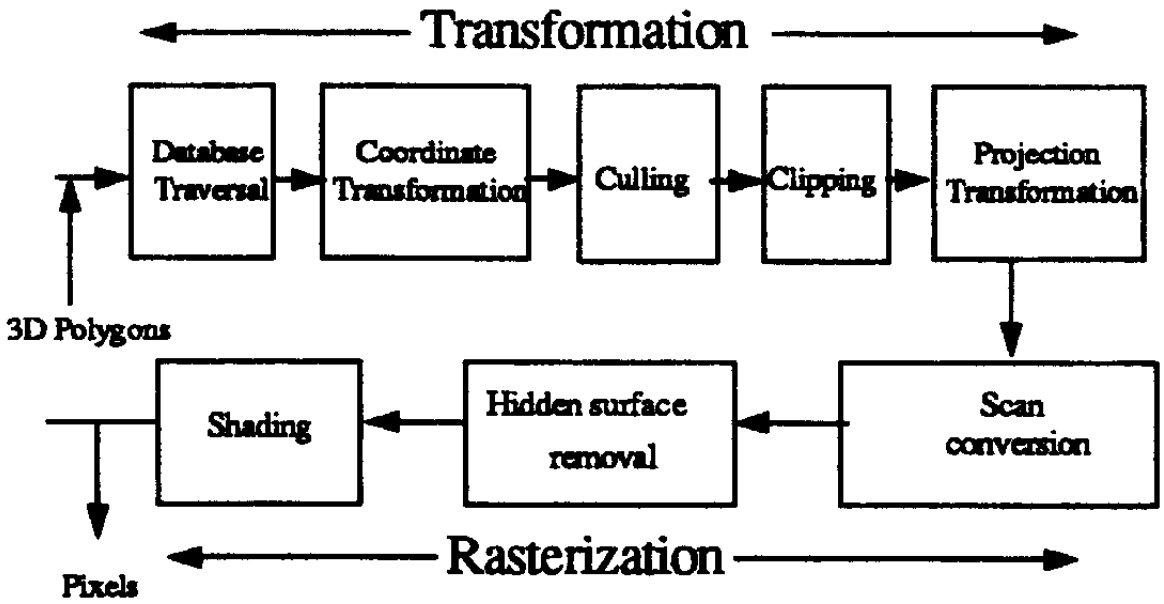
Product Support on the World Wide Web

Our World Wide Web site at www.ti.com contains the most up to date product information, revisions, and additions. Users registering with TI&ME can build custom information pages and receive new product updates automatically via email.

Introduction

Rendering 3D graphics is computationally very intensive, especially for interactive applications. Personal Computers (PCs) usually lack the power required for these calculations compared to workstations. Figure 1 shows a typical rendering pipeline in which 3D polygons are converted to pixel information.

Figure 1. Rendering Pipeline



The objects drawn on a 2D device (such as a screen) are represented as groups of mathematical primitives. Following certain operations on these primitives, a 3D scene is formed. There are two major operations:

- Transformation
- Rasterization

In the transformation operation, the system considers the current viewpoint and mathematically transforms the primitives' vertex coordinates from object space into screen space. In the rasterization operation, the system rasterizes the primitives by examining which screen pixels they overlap and then coloring in the appropriate contribution for each one.

Using multiple processors for both transformation and rasterization is a powerful performance option. Current implementations of the graphics pipeline in some architectures use parallelism at certain stages of the rendering pipeline to increase speed.[1] [2] [3] Although the most time consuming part of rendering pipeline is scan conversion there has been no serious attempt to parallelize this stage to speed up the rendering process.[4]

Scan Conversion

Converting the transformed 3D primitives into 2D pixel information on the screen stage is called scan conversion.

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. Figure 2 shows a polygon to be rendered. Figure 3 shows the final rendered and filled polygon.

Figure 2. Polygon to be Rendered

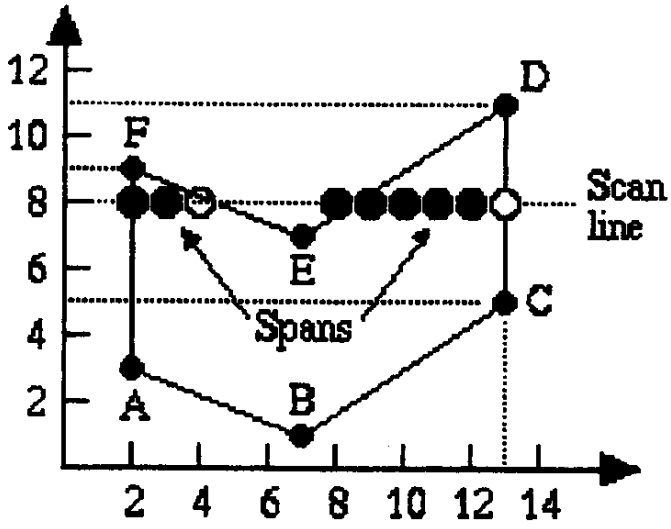
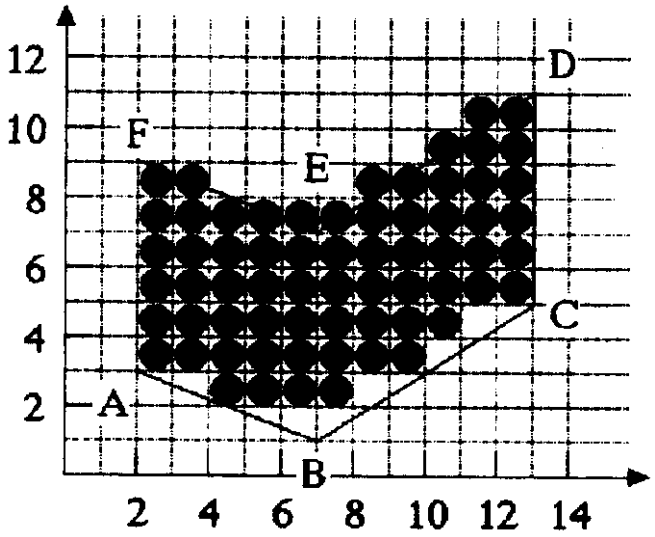


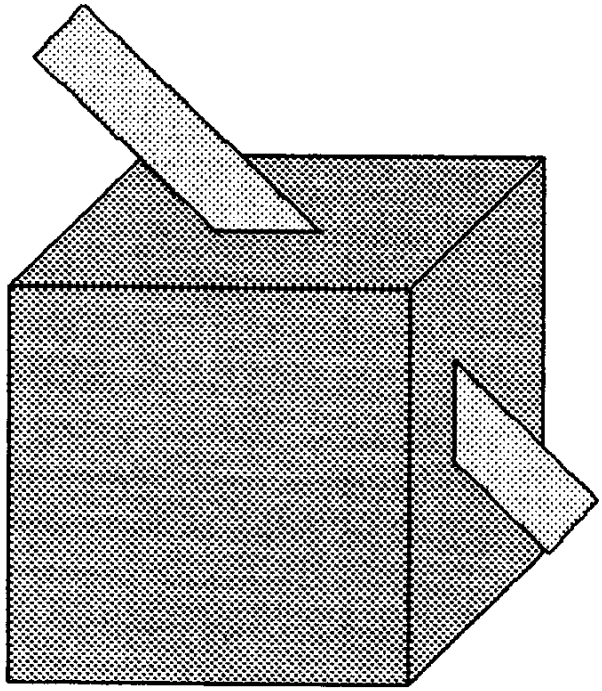
Figure 3. Final Rendered Polygon from Figure 2



After scan conversion, the hidden surface removal stage is performed. The purpose of the hidden surface removal is to remove any surfaces blocked by another object or surface by taking depth information into the account.

Figure 4 shows that the cube obstructs some parts of the rectangular plane where the hidden surface removal stage eliminates the obstructed parts. There are several algorithms for hidden surface removal but the most widely used one is the *z-buffer algorithm*.[5 6 7]

Figure 4. Hidden Surface Removal



In the z-buffer algorithm, for each pixel on the display, we keep a record of the depth of the primitive in the scene closest to the viewer, plus a record of the intensity that should be displayed to show the object. When a new polygon is to be processed, a z-value and intensity value is calculated for each pixel that lies within the boundary of the polygon.

If the z-value at a pixel indicates that the polygon is closer to the viewer than the z-value in the z-buffer, the z-value and the intensity values recorded in the buffers are replaced by the polygon's values. After processing all polygons, the resulting intensity buffer can be displayed.

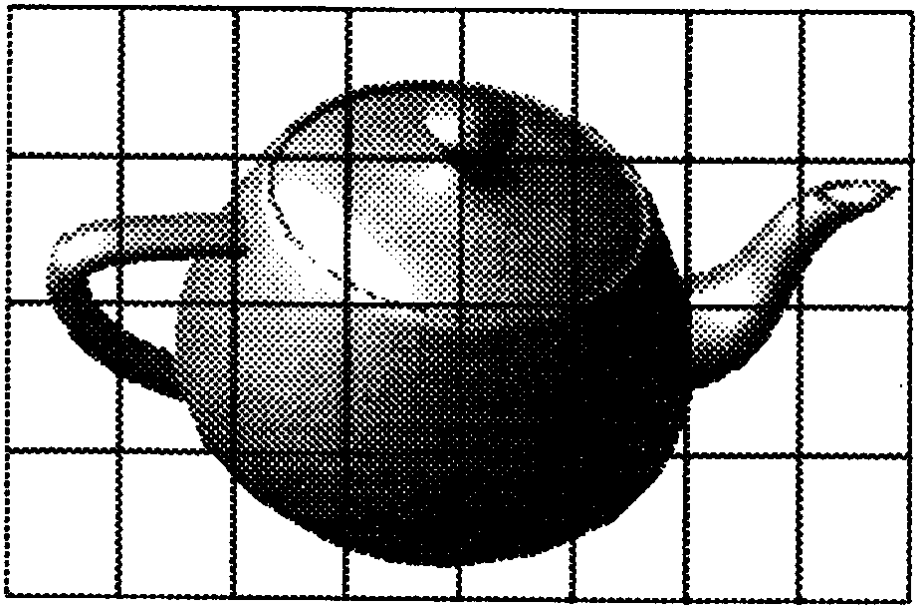
Z-buffer algorithm is easy to implement but the problem with this algorithm is the amount of memory that it requires since we also have to keep a record of depth information as well as intensity.

Parallel Scan Conversion

One methods proposed for parallel scan conversion is to divide the image into smaller portions and distribute these small regions to some parallel processing elements to be scan converted.[3]

Some problems encountered with this approach are the load imbalance because of the nonexistence of apriori knowledge about the regions and aliasing, especially at the borders of the small regions rendered separately and reassembled.[5] Operating on small regions and assembling them to be rendered clearly causes some problems at the borders of the sub-regions when regions are merged. The programmer must decide how to split up and recombine the workload among parallel processors. Figure 5 shows an image broken into 32 regions.

Figure 5. Data Partitioning



Because many of the regions happen to be empty, the processors assigned to those regions go idle right away. The number of regions should be decided in such ways that idle processors have plenty of things they can be reassigned to do.

The approach that we use is entirely different in that each processing element operates on a polygon rather than contiguous subimages. Since 3D databases are already made up of polygons, this strategy does not have the additional overhead of dividing up the image. When several parallel processors operate independently on a large number of polygons, which form the scene, it is expected to result in a substantial increase in rendering speed.

Using the TMS320C80 DSP

We use the TI TMS320C80 DSP to parallelize the scan conversion stage. The TMS320C80 has a RISC master processor (MP) with a floating point unit. Four parallel advanced digital signal processors (ADSPs) that are 32 bit integer units support 2 billion operations per second. Interprocessor communication between the processors is implemented by a combination of RAMs that are interconnected via the crossbar to the processors. The transfer controller handles block data movement.

We do not aim to compete with dedicated graphics accelerators. Our goal is to create a platform capable of multiple image processing and graphics functions, video conferencing, compression, and communication. The processing power of the TMS320C80 enables us to combine several applications into a single platform.

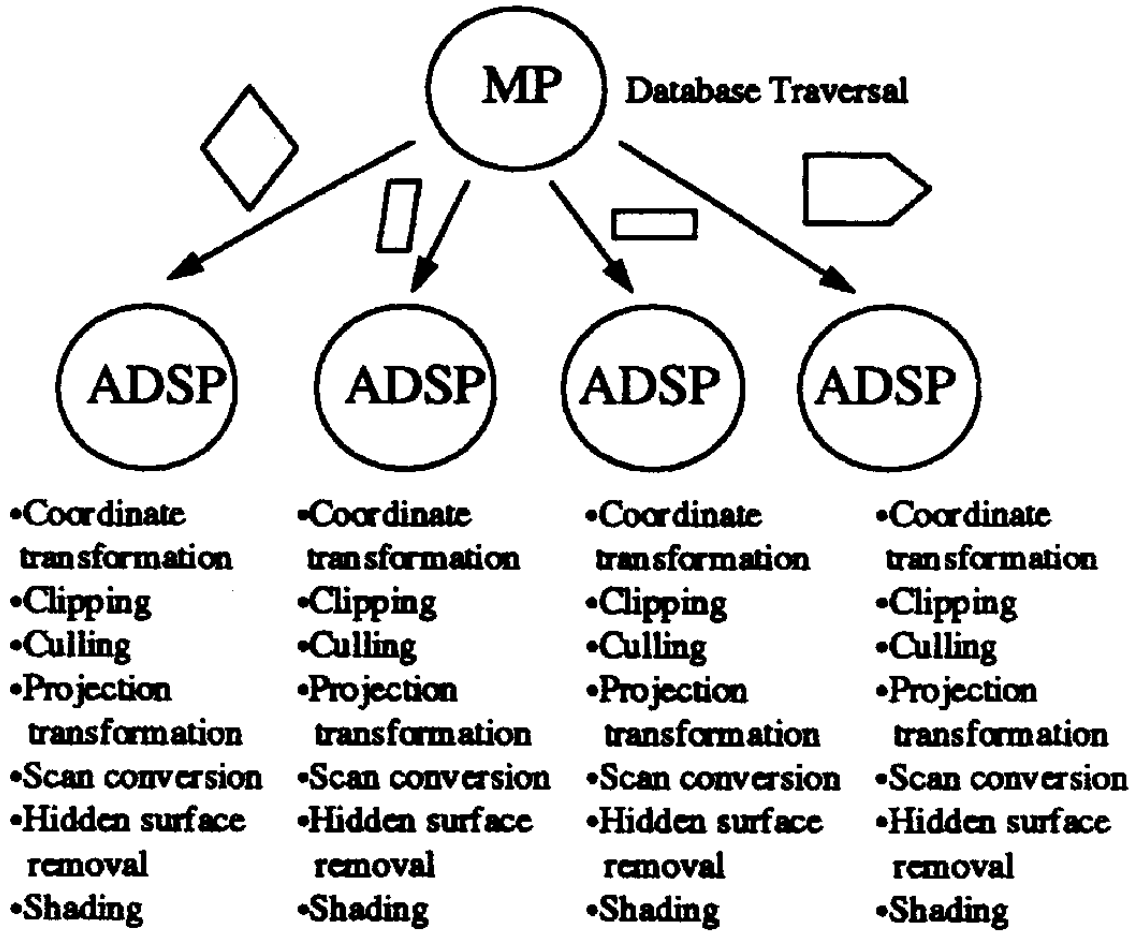
For example, the platform could be used in telemedicine in such a way that the picture of the bone of a patient could be sent to another expert through video conferencing. Then the expert can get another bone structure from his 3D database and compare both bones. A number of other applications, such as medical imaging, advance photocopiers and multimedia applications, will require a single board with the functionality of graphics, imaging and image processing, audio, communication, and compression. Acquisition, graphics and image processing, audio, communications, and compression algorithms can be implemented on the TMS320C80, which allows us to use a single board for a number of areas of applications.

The TMS320C80 ADSPs are used mainly in two ways:

- Partition the data among parallel processors so that each ADSP runs the same code and hence does the same job. In this case, the MP sends commands to each parallel processor in a parallel data flow fashion. Figure 6 shows how the MP partitions the data and sends a small portion of it to the ADSPs.

  For example, MP performs database traversal and sends polygons to the ADSPs. Each ADSP then performs coordinate transformation, clipping, culling, projection transformation, scan conversion, hidden surface removal, and shading on a single polygon it takes from the MP.

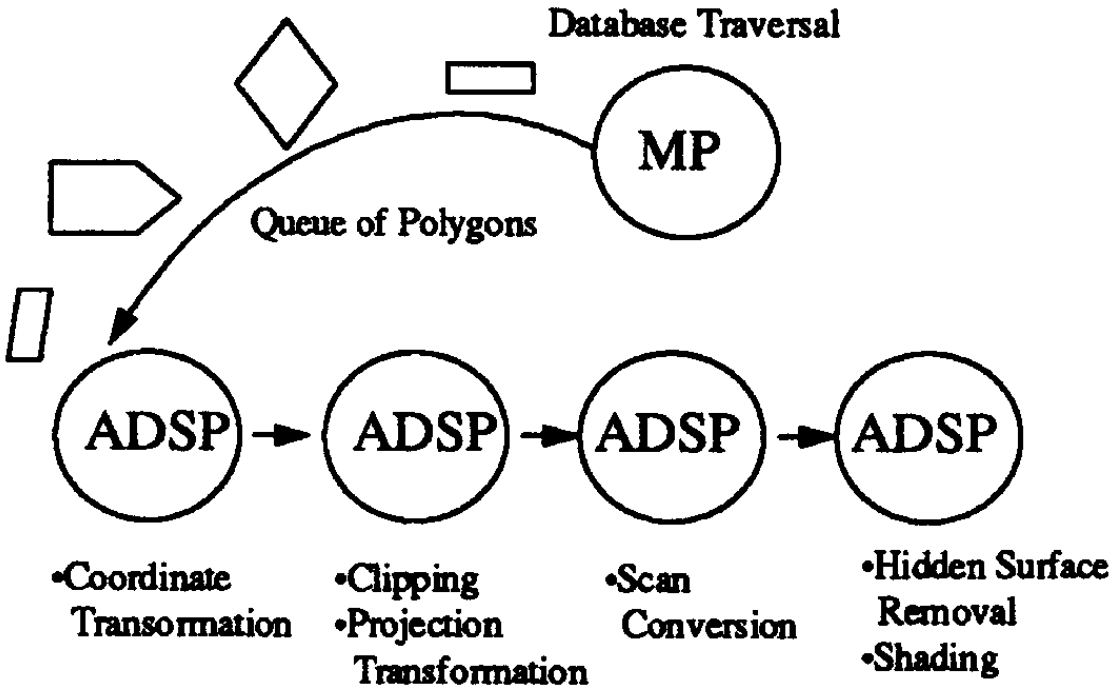
Figure 6. ADSPs Running the Same Code



- Partition the tasks to be performed so that each ADSP runs a different code. In this case, the MP sends commands to one of the processors and the output of the data is passed to another processor to be processed. While the second ADSP is performing its job, the first ADSP can start performing the next bit of data. Figure 7 shows how the MP partitions the data and sends a small portion of it to the ADSPs. Each ADSP performs a different stage of the graphics pipeline.

  For example, while the first ADSP is performing coordinate transformation, the second one performs clipping and projection transformation, the third one performs the scan conversion, and the fourth one performs the rest of the pipeline.

Figure 7. Each ADSP Running a Different Code



It should be clear that the latter case is not suitable for parallelizing the graphics rendering because it is nothing but pipelining; hence, we use the first scheme. Although hybrid combinations of both approaches are possible, we believe the first method is the best solution to increase the speed of the scan conversion stage of the graphics pipeline. This scheme avoids memory and crossbar contentions caused when ADSPs try to read/write to the same data locations. Our initial experiments also support this belief. Each of the ADSPs are assigned a polygon to be scan converted from the polygon list and each polygon is rendered by a single ADSP until rendering is complete.

We must avoid the load imbalance that can be caused by the uneven distribution of the polygons over the ADSPs, which is vital to obtain the maximum throughput. If the sizes of the polygons are quite different from each other, the MP should take care of it and chop the polygons down to smaller polygons or trapezoids.

Summary

This application report explained the need to parallelize the scan conversion stage. It was suggested that that the ADSP should work on a single polygon rather than on a small contiguous region of data. The MP converts polygons into smaller polygons if necessary. ADSPs are used in parallel to increase the speed of the scan conversion stage of the graphics pipeline. The MP distributes tasks and organizes the data flow.

Our initial work, which is capable of rendering wireframe objects, shows that 3D rendering significantly benefits from parallelization. The final aim of this work is to build a general-purpose 3D-application program interface (API).

Various strategies to provide equal load balance on parallel processors for rendering will be tested in the immediate future. We expect to improve the rendering performance even further with a purpose built multitasking executive.

Acknowledgments

The authors would like to thank Dr. Barry Jefferies, Head of Division, Electronic and Electrical Engineering, University of Herfordshire for his support of this work.

References

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

[2] 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

[3] F. C. Crow, "Parallelism in Rendering Algorithms", *Proceedings of Graphics Interface '88*, pp.87-96

[4] T. Lee and C.S. Raghavendra, *Parallel Processing for Graphics Rendering on Distributed Memory Multicomputers*, School of Electrical Engineering and Computer Science Washington State University Pullman, WA

[5] Foley J., A. van Dam, *Computer Graphics Principles and Practice*, Addison Wesley, 1990.

[6] Rogers, D.F., *Procedural Elements for Computer Graphics*, McGraw Hill, 1988.

[7] Watt, A., *3D Computer Graphics*, Addison Wesley, 1994.