Design and Implementation of Texture Mapping in Parallel

Wen Xu
College of Computer science and Technology
Xi'an University of Posts and Telecommunications
Xi'an, China
xwen1988@126.com

Jungang Han
College of Computer science and Technology
Xi'an University of Posts and Telecommunications
Xi'an, China
hjg@xupt.edu.cn

***Abstract*—Texture mapping is an important part of the realistic graphics rendering process. In this paper we parallelize the algorithm for traditional texture mapping to improve the running speed of the program. The experimental results show that, the speed-up of the program can reach to 1.92 in average.**

Keywords-computer graphics; parallel algorithm; texture mapping.

I. INTRODUCTION

In computer graphics, texture mapping is an important part of realistic graphics rendering process, it can more accurately reflect the realistic graphics and is widely used in computer animation design, 3D games and advertising design. Texture mapping [1], is the process of mapping the existing texture image to the surface of the object, so as to realize to increase surface details of the objects. Using texture mapping technology can greatly improve the fidelity of image. In real time rendering the parallelization [2] of the algorithms in a multi-core architecture is one way to improve the speed of the texture mapping.

In this paper, we use the PAAG (Polymorphous Array Architecture for Graphics) array simulation system [3] [4], which contain a complete compilation systems and the debugging environment to get the running statistical analysis and evaluate the system performance of the programs.

II. THE DESIGN OF TEXTURE MAPPING

A. The theory of texture mapping

Texture mapping can be divided into one-dimensional, two-dimensional and three-dimensional. The most widely used mapping is two-dimensional texture mapping. In this paper, we will focus on two-dimensional texture mapping.

Texture mapping process involves the correspondence between vertexes [5], we have to set the coordinates for objects, image of the texture and texture coordinates respectively. The texture coordinates ranges between 0.0 and 1.0, the corresponding relations among them is shown in figure 1:

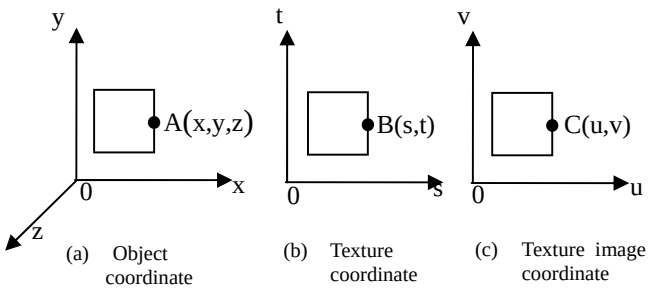

Figure 1. Texture mapping structure

Mapping texture space to object space is usually expressed by the following linear functions with parameters:

$$\begin{cases} u = f_u(s,t) = a_u s + b_u t + c_u \\ v = f_v(s,t) = a_v s + b_v t + c_v \end{cases} \quad (1)$$

B. Texture filtering technology

Different filtering methods have different computing complexity and computing methods and the effect is also different. There are two kinds of commonly used simple filtering technology, one is Nearest Point Sampling, another is Bilinear .

In the Nearest point sampling method a point P is selected, which is not necessarily just a sampling point 'texel' to the corresponding texture, so it will choose the nearest texels in the sampling area, which means it will select the abscissa and ordinate which is close to the point P as the sampling point.

Bilinear filtering takes the texture coordinates which corresponding to the pixel as the center, four texel pixels around the texture coordinates are taken to calculate the color of the points. The easiest way is using the average value of four surrounding coordinates color as samples, that is to say, taking the average weight of the 2×2 texture unit which is close to the center of the pixel block. The usual method is according to the space as the weight of the four coordinates to calculate the pixel of the point (i.e. the texture color value). Bilinear filtering is shown in figure 2.

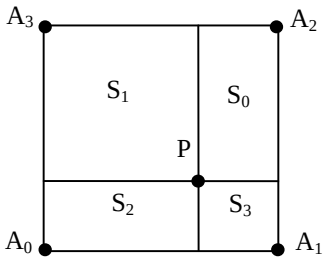

Figure 2. Bilinear filtering

Among them, corresponding to the vertex coordinates of point P, the surrounding 4 integer coordinates areA0, A1, A2 and A3,their weight values are $\frac{S_0}{S}$, $\frac{S_1}{S}$, $\frac{S_2}{S}$ and $\frac{S_3}{S}$, where $S = S_0 + S_1 + S_2 + S_3$,the pixel value of point P (i.e. texture color values, for RGBA format, the R calculation, G, B and A are the same)is shown as follows:

$$A_p = \frac{S_0}{S} \times A_0 + \frac{S_1}{S} \times A_1 + \frac{S_2}{S} \times A_2 + \frac{S_3}{S} \times A_3$$

C. Design and implementation

In this paper, the compilation process mainly involves two major functions:

glTexEnv () and glTexParameter(), and we assign an address and storage to their parameters.

The flow chart of texture mapping design is shown in figure 3:

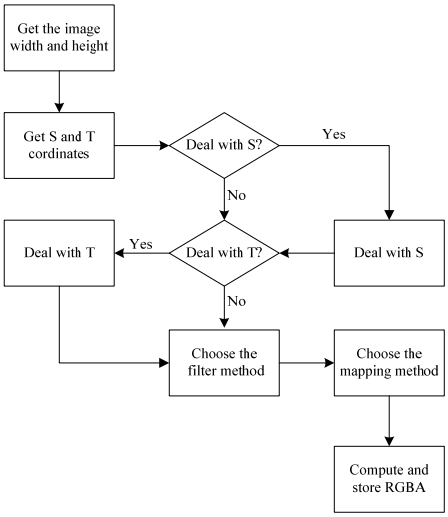

Figure 3. Flowchart of procedure.

1) Processing for texture coordinates

Since texture coordinates are limited between 0.0 and 1.0, we must deal with the texture coordinates which outside the range of values, S and T coordinates are handled as follows (The initial state is for all of S and T behavior to be that given by REPEAT.):

- If TEXTURE _WRAP_S or TEXTURE_WRAP_T is set to REPEAT, then the GL ignores the integer part of s or t coordinates, respectively, using only the fractional part. (For a number f , the fractional part is $f - \lfloor f \rfloor$, regardless of the sign of f ;
- If it is set to CLAMP, it will cause s or t coordinates to be clamped to the range $[0,1]$;
- CLAMP TO EDGE clamps texture coordinates at all mipmap levels such that the texture filter never samples a border texel. The color returned when clamping is derived only from texels at the edge of the texture image. Texture coordinates are clamped to the range $[\min, \max]$. The minimum value is defined as $\min = \frac{1}{2N}$ where N is the size of the two-dimensional texture image in the direction of clamping. The maximum value is defined as $\max = 1 - \min$;
- CLAMP TO BORDER clamps texture coordinates at all mipmaps such that the texture filter always samples border texels for fragments whose corresponding texture coordinate is sufficiently far outside the range [0, 1]. The color returned when clamping is derived only from the border texels of the texture image, or from the constant border color if the texture image does not have a border. Texture coordinates are clamped to the range $[\min, \max]$. The minimum value is defined as $\min = \frac{-1}{2N}$, where N is the size (not including borders) of the two-dimensional texture image in the direction of clamping. The maximum value is defined as $\max = 1 - \min$.

2) Choose for texture mapping mode

There are a variety of texture mapping modes: GL_REPLACE 、 GL_MODULATE 、 GL_BLEND 、 GL_DECAL and GL_ADD, the methods of their specific processing are shown in Table 1:

TABLE I. FUNCTION OF TEXENV()

Internal format	GL_RGBA
GL_REPLACE	$C = C_s \quad A = A_s$
GL_MODULATE	$C = C_f C_s \quad A = A_f A_s$
GL_DECAL	$C = C_f (1 - A_s) + C_s A_s$ $A = A_f$
GL_BLEND	$C = C_f (1 - C_s) + C_c C_s$ $A = A_f A_s$
GL_ADD	$C = C_f + C_s \quad A = A_f A_s$

The index values in the table stand for the following meanings:

- s stand for the texture source color;

- f stand for the new fragment value;
- C stand for a GL_TEXTURE_ENV_COLOR assigned value, which is set to a constant value;
- If the subscript is not assigned, then the result is the final calculation of the color values.

III. PARALLELIZATION OF TEXTURE MAPPING AND SIMULATION IMPLEMENTATION

The main problem to achieve the parallelism of texture mapping is how to break down the data and allocate the task. Task decomposition is to divide the task into several sub-tasks which can be performed simultaneously, according to this method, we are able to classify a large number of independent tasks, if one of the two tasks can run at the same time, the schedule can be formed between two concurrent execution; data decomposition sucks a relatively large data set to be processed, such as dividing an array into several subsets, which can implement the operation for the different parts of the member at the same time.

In order to increase the speed of texture mapping, it can be found that there exists parallel algorithm in the texture mapping algorithms, we can use the parallelism to realize the optimization of the algorithm, the algorithm can divide into a plurality of modules in parallel processing, and the program uses the pipeline means for execution.

A. Simulation Environment

PAAG (Polymorphic Array Architecture for Graphics and Image Processing) system is an 4×4 array, which contains 16 PEs, it can support 16 programs running at the same time; the program can be performed with data interaction. At the runtime, PAAG can be modeled per PE to achieve the communications between different modules. More details about PAAG can be found in reference [3].

PAAG simulation system design has a blocking and non-blocking mode, which can ensure the correctness of data transmission. It supports a PE and 4 PE to interact at the same, PAAG system structure is shown in figure 4:

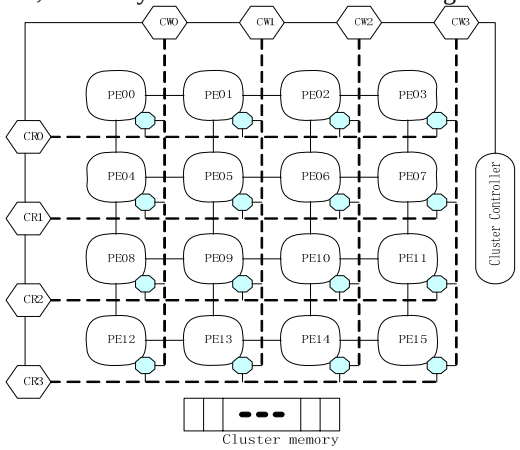

Figure 4. PAAG Structure

B. Parallel Implementation

Considering the limitation of the serial algorithm, which cannot have an effective use of the resources, in order to make full use of the resources and achieve the load balance, we found that there may be two places which can be parallelism in the texture mapping algorithm, one is the S and T coordinates processing algorithms, because the treatment methods of S and T coordinates are the same and the process has no data interaction, they can be processed in parallel; another one is R, G, B and A processing algorithms, their treatment will not affect each other, so they can process in parallel.

The experiment used 2 PEs, 3 PEs and 5 PEs (structure diagram for the 7 PE, in fact the effective use of PE is 5, two of which are used as data transmission, without any other operation) to deal with the texture mapping algorithm, the specific structure is shown in figure 5:

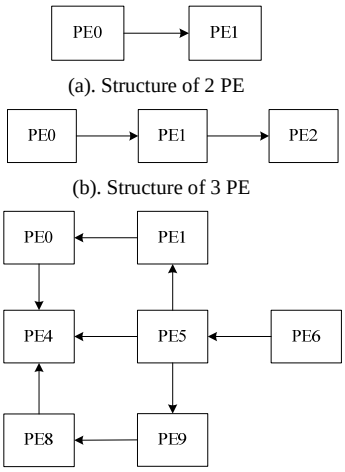

Figure 5. Structure

- In Fig. 5(a), PE0 performs the S coordinate processing, and PE1 achieves the processing of T coordinate, R, G, B and A.
- In Fig. 5(b), PE0 performs the S coordinate processing, and PE1 achieves the processing of T coordinates, R and G, PE2 achieves the processing of B and A.
- In Fig.5(c), PE6 achieves the S coordinate processing, PE5 achieves the processing of T coordinate and R, after the operation on PE5 is finished, the process will pass the results of S and T coordinates to PE9, PE4 and PE1. PE9, PE4 and PE1 realize the processing of G, B and A, after processing ,PE5 passes the R-value to PE4, PE9 passes the G-value to PE4 through PE8, PE1 passes the A-value to PE4 through PE0, finally the results summarizes in PE4 and we can get the final RGBA values on PE4.

C. *Simulation results.*

To realize the texture mapping algorithm for the above structure, we obtain the following results, which is analyzed and compared, the total clock of the serial program is 819 , the speed-up is defined as:

$$a = \frac{T_s}{T_p}$$

T_s is the running time for the serial program, T_p is the running time for the parallel program.

TABLE II. TIME SCALE OF 2 PE

PE	Execute time	Block time	Total time	Speed-up
0	178	0	178	
1	630	0	0	1.3

TABLE III. TIME SCALE OF 3 PE

PE	Execute time	Block time	Total time	Speed-up
0	173	0	173	
1	487	12	499	1.59
2	490	25	515	

TABLE IV. TIME SCALE OF 5 PE

PE	Execute time	Block time	Total time	Speed-up
1	214	188	402	
4	220	207	427	
5	405	0	405	1.92
6	191	0	191	
9	194	186	380	

By the results of the statistics, we can visually observe the change of the speed-up of different number of PEs from Figure 6. The horizontal axis shows the number of PE used for the execution and the vertical axis shows the speed-up.

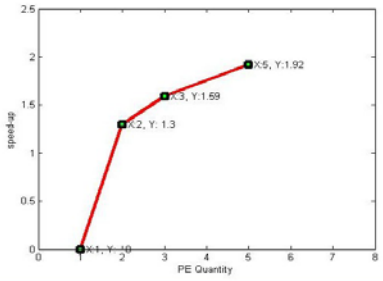

Figure 6. Speed-up of different number of PE

IV. CONLUSION

In this paper, we briefly introduce the texture mapping algorithm and texture filtering techniques. Meanwhile, we analysis the algorithm for existing parallelism, the S and T coordinates processing algorithms and the R, G, B and A for processing algorithm can divide into a plurality of parallel processing modules, which can greatly improve the speed of texture mapping, the speed-up of final processing can reach to 1.92 in average, the processing speed is increased by 91 percent.

ACKNOWLEDGMENT

The authors would like to thank the many staff and graduate students at Xi'an University of Posts and Telecom for participating in this project. This support of the China Natural Science Foundation (grant 61136002) is acknowledged.

REFERENCES

[1] E.A.Bier, K.R.Sloan. Two-Part Texture Mapping[J]. IEEE Computer Graphics Application,1986(6):40-53

[2] Jorge L. C. Sanz. Computing Image Texture Features in Parallel Computers, Proceedings of the IEEE. 1988,76(3):292-294

[3] T. Li, "PAAG: A Polymorphic Array Architecture for Graphics and Image Processing", PAAP'2012, IEEE Computer Society CPS, pp. 242-249, Dec. 2012

[4] Hucai Huang, Tao Li, Jungang Han, Simulator Implementation and Performance Study of A Polymorphous Array Computer, 2013 12th IEEE International Conference on Trust, Security and Praivacy in Cmputing and Communications, Melbourne, Australia, pp1848-1855

[5] L. Dong, J. Han, Design and Simulation of Lighting and Texture Mapping in GPU.Computer Science, 2011, 02

[6] Thomas Rauber, Gudula Runger, Parallel Programming, Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2010

[7] Huijian Han, Hengwu Li. Texture Mapping Based on Coefficient Map and Basic Texture Map, Academy Publisher, 2009,4(6):412-418

[8] Dave Shreiner,The Khronos OpenGL ARB Woking Group, OpenGL Programming Guide: The Official Guide to Learning OpenGL, Addison-Wesley Educational Publishers Inc

[9] S.W. Keckler, W.J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the future of parallel computing", IEEE Computer, Vol.44, No.9, pp7-17, 2011

[10] T. Li, H. Du, L. Zhang, J. Han, "Design and analysis of a class of simple networks", 13th Int. Conf. Parallel and Distributed Computing, Applications and Technologies (PDCAT 2012), IEEE press, 2012, pp285-289.

[11] K. He .OpenGL Programming Technology , Chemical Industry Press, 2010.

[12] Donald Hearn, M.Pauline Baker, OpenGL Graphics,2010

[13] Chas Boyd, "Data Parallel Computing", ACM QUEUE, March/April,2008, pp30-39.