DIVISION-FREE RASTERIZER FOR PERSPECTIVE-CORRECT TEXTURE FILTERING

Donghyun Kim and Lee-Sup Kim

Dept. of EECS, KAIST, 373-1 Guseong-dong, Yuseong-gu, Daejeon, 305-701, Republic of KOREA

ABSTRACT

Perspective-correct texturing for correct 3D graphics images requires the per-pixel division, but the division can be avoided by midpoint algorithms. Previous approaches using midpoint algorithms for perspective-correct texturing evaluate the integer texture coordinates, which are not suitable for trilinear filtering used in the most applications. In this paper, we propose a rasterizer architecture which separates the evaluation of integer part and the computation of fraction parts. The fraction part of a texture coordinate is computed per cycle in spite of short pipeline depth, consequently much smaller cost than divider. The implemented parallel rasterizer runs at 100 Mhz clock and evaluates 400M pixels and 800M texture coordinates per second.

1. INTRODUCTION

Every pixel on the display in widespread mobile phones or PDA systems must be rendered in higher quality because the average eye-to-pixel distance is very short. In order to improve 3D graphics image with no visual pixel defects, every textured pixel on the display must be rendered with perspective-correct methods. [1] However, usual implementations for perspective correct rendering of planar polygons require a division per pixel. [1] Division requires long latency and large area, so division per pixel makes it difficult to achieve high throughput of pixel-fill rate with small hardware cost.

In order to avoid division per pixel, there have been several studies. Some techniques to approximate hyperbolic curve with linear and quadratic interpolation have been presented. [5] These polynomial algorithms use only adders instead of dividers, but they are approximations with possible errors. There have been other approaches that use midpoint algorithms [2,3], which are well known for the trace of hyperbolic functions. These approaches which induce mid-point algorithms for perspective correct texturing are well summarized and published by Barenbrug [3]. He demonstrates that the midpoint algorithm makes exact integer texture coordinates for point sampling, only using several additions instead of a division. However, point sampling does not provide good image quality, and then trilinear filtering instead of point sampling is usually used in the most applications. Usual filtering methods such as trilinear filtering require the fraction part of texture coordinates as filtering coefficients. Fixed point representations including fraction parts can be scaled such that they take integer values only, as mentioned in [3], but the number of iteration are too increased to be used in hardware.

In this paper, we modify midpoint algorithms to be suitable for hardware and present its hardware architecture and implementation. The fraction part of texture coordinates are evaluated without iteration overhead. We also demonstrate mipmap level switching architecture using the internal variables for midpoint algorithms.

This paper is organized as follows. First, perspective-correct texturing and midpoint algorithms are reviewed. After that, fraction evaluations and mipmap switching are explained in detail. And then hardware architecture is described. Finally, discussion and summary are offered in conclusion.

2. BACKGROUND

2.1. Perspective-correct texture mapping

The perspective texture coordinates of a certain pixel *(u, v)* are calculated as follows. For triangles, the screen coordinates and the texture coordinates are connected by a homogeneous linear transformation. The relation among the screen coordinates *(x, y)*, the homogeneous 2D screen coordinates *(x', y', w')* and the texture coordinates *(u, v)* is given by:

$$(x,y)=\left(\frac{x'}{w'},\frac{y'}{w'}\right) \qquad \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} K & L & M \\ N & P & Q \\ R & S & T \end{bmatrix} \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix}$$

Therefore, texture coordinates *(u, v)* can be represented in terms of screen coordinates *(x, y)* in the next relation.

$$(u,v)=\left(\frac{Kx+Ly+M}{Rx+Sy+T},\frac{Nx+Py+Q}{Rx+Sy+T}\right) \qquad (1)$$

As shown in (1), two division operations or one reciprocal and two multiplications are required in calculating exact texture coordinates of one pixel.

2.2. Midpoint Algorithm

The idea using midpoint algorithm in perspective texture rendering was proposed in [3]. The key idea of [3] is that repetition of additions can follow hyperbolic curves within given precision as shown in Figure 1. Let's assume that u and x are represented by only integers. We do not need to know the exact value of $f(x_i)$ for given x_i. The exact function value $f(x_i)$ must be taken in limited precision, so the nearest integer u_i is acceptable for $f(x_i)$. The difference between u_i and $f(x_i)$ is smaller than the precision error. Therefore, u_i is the exact value allowed by given precision.

The acceptable value u_i for x_i must be an integer inside acceptable region shadowed in Fig. 1. Considering both of x and y in (1), this condition is formulated as follows.

$$\frac{Kx_i+Ly_i+M}{Rx_i+Sy_i+T}-0.5<u_i\le\frac{Kx_i+Ly_i+M}{Rx_i+Sy_i+T}+0.5 \qquad (2)$$

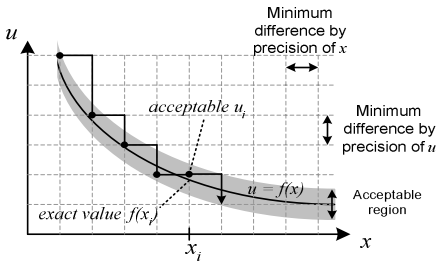

Figure 1: Integer points in the acceptable region for a hyperbolic curve are evaluated by a midpoint algorithm.

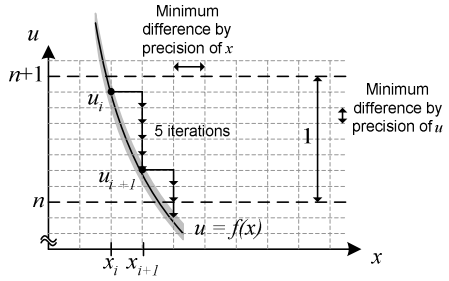

Figure 2: When three bits are used to represent the fraction part, the acceptable region width and the minimum movement of iteration (6) are not 1 but 1/8.

As proposed in [3], several variables are introduced:

$$\begin{aligned} d(x,y) &= 2(Rx+Sy+T) \\ E(x,y,u) &= ud(x,y)-(R+2K)x-(S+2L)y-(T+2M) \end{aligned} \tag{3}$$

Assuming $d>0$, to substituting (3) simplifies (2) to (4).

$$-d(x_i,y_i) < E(x_i,y_i,u_i) \le 0 \tag{4}$$

A new variable A is introduced for simplicity. When x increases by 1, d and E are changed as shown in (5).

$$\begin{aligned} A(u) &= (2u-1)R-2K \\ d(x+1,y) &= d(x,y)+2R \\ E(x+1,y,u) &= E(x,y,u)+A(u) \end{aligned} \tag{5}$$

When u increases by 1, A and E are modified as follows.

$$\begin{aligned} E(x,y,u+1) &= E(x,y,u)+d(x,y) \\ A(u+1) &= A(u)+2R \end{aligned} \tag{6}$$

Therefore, when E and d change by increase or decrease of x or y, E can be restored to satisfying inequality (4) by adjusting u with iterative additions or subtractions in (6).

However, In order to trace a hyperbolic curve as shown in Fig. 1, an increment by 1 in x or y may require multiple increments or decrements of u depending of partial derivatives of the hyperbolic curve. The iteration of additions or subtractions does not have upper bound generally, but mipmap [4] limits the derivatives such as $\partial u/\partial x$ to two. But still, when using texture filtering requires fraction parts of texture coordinates, much more iterations must be done as shown in Fig 2. the example in Fig 2 shows the case that 5 iterations of computing (6) and checking (4) are required to obtain correct u_{i+1} from u_i. To calculate u_{i+2}, 3 iterations of (6) and (4) must be computed. If three bits are used to represent the fraction of u, $2\times2^3=16$ iterations must be performed in the worst case in order to obtain texture coordinates of one pixel. This long latency by the iteration wastes clock cycles in the hardware implementation, like divisions. This problem can be solved by the proposed architecture that separates integer part interpolation and fraction part computation.

3. COMPUTING FRACTION OF TEXTURE COORDINATES

Assuming that m bit precision of the texture coordinate fraction is required as texture filtering coefficients, inequalities (2) are to be modified to (7).

$$\frac{Kx_i+Ly_i+M}{Rx_i+Sy_i+T}-\frac{1}{2^{m+1}} < u_i \le \frac{Kx_i+Ly_i+M}{Rx_i+Sy_i+T}+\frac{1}{2^{m+1}} \tag{7}$$

Instead of (7), let us consider an inequality (8) and Fig. 3.

$$\frac{Kx_i+Ly_i+M}{Rx_i+Sy_i+T}-\frac{1}{2^{m}} < u_i \le \frac{Kx_i+Ly_i+M}{Rx_i+Sy_i+T} \tag{8}$$

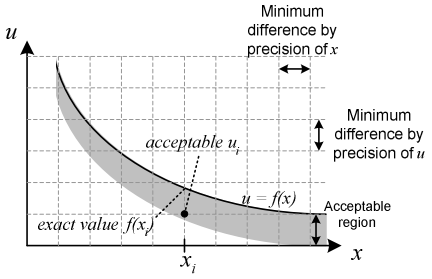

Figure 3: Acceptable region is changed by (8).

Fig. 3 shows the acceptable u_i defined by inequality (8). Inequality (8) evaluates u_i by truncation while inequality (7) rounds off $f(x_i)$. Rounding off computes closer u_i to original curves than truncation, but the difference is negligible for enough precision.

We introduces E' and A' instead of E and A defined in (3) and (6) so that E' is independent of m as follows. Note that the second and the third in the relation (5) are still valid.

$$\begin{aligned} E'(x,y,u) &= ud(x,y)-2(Kx+Ly+M) \\ A'(u) &= 2uR-2K \end{aligned} \tag{9}$$

Therefore, substituting E' in (8) yields:

$$-\frac{d(x_i,y_i)}{2^m} < E'(x_i,y_i,u_i) \le 0 \tag{10}$$

Let's take the sequence $k(u)$ to represent binary figures of fraction part of u as shown in the next relation. $\lfloor u_i \rfloor$ implies the largest integer equal to or less than u_i.

$$u_i = \lfloor u_i \rfloor + \sum_{j=1}^{m} \frac{k_j}{2^j} \qquad k_j \in \{0,1\}$$

$$-d(x_i,y_i) < E'(x_i,y_i,\lfloor u_i \rfloor) \le 0 \tag{11}$$

Naturally, $\lfloor u_i \rfloor$ satisfies (10) in the case of $m = 0$, as shown in (11). Notice that (11) and (4) have the same shape such that $\lfloor u_i \rfloor$ is evaluated as described in section 2.2.

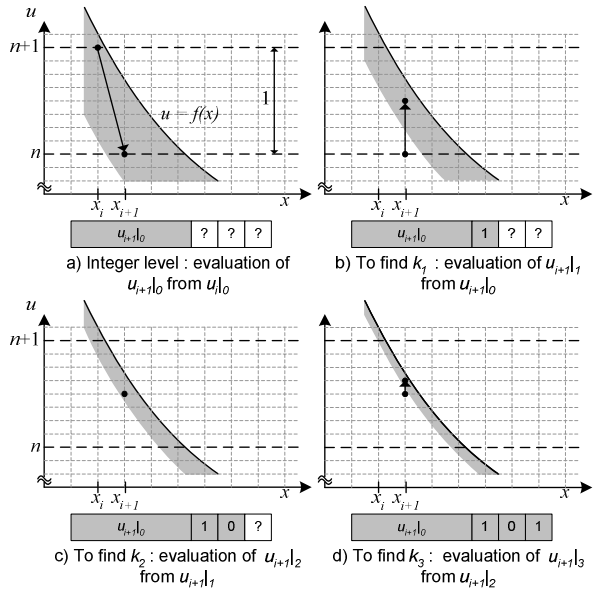

Figure 4: Finding from k_1 to k_m sequentially.

We will show that evaluation of u_i in fraction level can be separated from the evaluation of $\lfloor u_i \rfloor$. u_i can be evaluated from only $\lfloor u_i \rfloor$ regardless of u_{i-1}, and $\lfloor u_i \rfloor$ is evaluated from $\lfloor u_{i-1} \rfloor$ by one or two iterations by (6).

Let $u|_n$ imply the number with n fraction bits equal to or less than u. Then next relations are induced easily.

$$
\begin{aligned}
& u_i\big|_{n+1} = u_i\big|_n + \frac{k_{n+1}}{2^{n+1}} = \lfloor u_i \rfloor + \sum_{j=1}^{n+1} \frac{k_j}{2^j} \\
& u_i\big|_0 = \lfloor u_i \rfloor \qquad\qquad u_i\big|_m = u_i \\
& E'(x_i, y_i, u_i\big|_{n+1}) = E'(x_i y_i, u_i\big|_n) + \frac{k_{n+1}}{2^{n+1}} d(x_i, y_i) \\
& -\frac{d(x_i, y_i)}{2^n} < E'(x_i, y_i, u_i\big|_n) \le 0
\end{aligned}
\tag{12}
$$

The conditions of sequence k are induced from (12) as shown in (13).

$$
\begin{aligned}
& -\frac{d(x_i, y_i)}{2^{n+1}} < E'(x_i, y_i, u_i\big|_n) \le 0 \qquad \rightarrow\ k_{n+1} = 0 \\
& -\frac{d(x_i, y_i)}{2^n} < E'(x_i, y_i, u_i\big|_n) \le -\frac{d(x_i, y_i)}{2^{n+1}} \rightarrow\ k_{n+1} = 1
\end{aligned}
\tag{13}
$$

The sequence k is obtained from k_1 to k_m step by step as shown in Fig. 4. These sequential evaluations can be pipelined and the required hardware resource for one pipeline stage is only one comparator for checking (13) and one adder for updating E'.

4. LEVEL OF DETAIL AND MIPMAP

Mipmap is pyramidal data structure of pre-processed texture data. [4] Trilinear filtering using mipmap provides better image quality than point sampling. Mipmap also limits the derivatives so that the iteration limit is bound in midpoint algorithms.

Because of its pyramidal structure, the texture coordinate u goes to the half when a mipmap level is increased by one. Therefore, we rewrite (8) and (9) considering mipmap level $\lfloor \lambda \rfloor$ as following expressions. When a mipmap level is changed, internal variables E', A' do not need to be calculated again. Only one bit shift operations solve new E', A', When a mipmap level is increased by one, E and A go half and hence, one bit right shift operation. In the case of level-down, E' and A' become double, and one bit left shift operations are used.

$$
\frac{1}{2^{\lfloor \lambda \rfloor}} \frac{Kx_i + Ly_i + M}{Rx_i + Sy_i + T} - \frac{1}{2^m} < u_i \le \frac{1}{2^{\lfloor \lambda \rfloor}} \frac{Kx_i + Ly_i + M}{Rx_i + Sy_i + T}
$$

$$
E'(x, y, u) = ud(x, y) - \frac{2}{2^{\lfloor \lambda \rfloor}} (Kx + Ly + M)
$$

$$
A'(u) = 2uR - \frac{2}{2^{\lfloor \lambda \rfloor}} K
$$

A mipmap level is determined by level-of-detail (LOD) value evaluated as expressed as follows. [3,4] B' is defined for y similarly as A'. $\Delta\lambda$ implies the difference of the LOD value and the current mipmap level.

$$
\begin{aligned}
\Delta\lambda &= \log_2 \left\{ \max\left(\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial u}{\partial y}\right|, \left|\frac{\partial v}{\partial x}\right|, \left|\frac{\partial v}{\partial y}\right| \right) \right\} \\
&= \log_2 \left\{ \max\left(\left|\frac{Au'}{d}\right|, \left|\frac{Bu'}{d}\right|, \left|\frac{Av'}{d}\right|, \left|\frac{Bv'}{d}\right| \right) \right\} \\
&= \log_2 \left\{ \max\left(|Au'|, |Av'|, |Bu'|, |Bv'| \right) \right\} - \log_2 d
\end{aligned}
$$

If $\Delta\lambda$ is less than 0, the mipmap level will be decreased. If $\Delta\lambda$ is equal to or more than 1, the mipmap level will be increased. We can check this condition without logarithm as follows.

$$
\begin{aligned}
& \Delta\lambda \ge 1 \leftrightarrow \max\left(|Au'|, |Av'|, |Bu'|, |Bv'| \right) \ge 2d \\
& \Delta\lambda < 0 \leftrightarrow \max\left(|Au'|, |Av'|, |Bu'|, |Bv'| \right) < d
\end{aligned}
\tag{14}
$$

5. THE HARDWARE IMPLEMENTATION

5.1. Overall Architecture

In this section, overall architecture of the proposed rasterizer will be described. Fig. 5 shows the block diagram of a basic rasterization unit.

Context Register contains the variables of a current pixel such as position, color, depth and texture coordinates. Note that *Context Register* stores only integer part of texture coordinates.

Pixel Interpolator interpolates the contexts by the controls of FSM. Internal variables E' and d are also updated by (5) in *Pixel Interpolator*.

Texel Interpolator finds correct the integer part of texture coordinates by (6). *Texel Interpolator* has two adder sets to avoid iterations.

Mipmap Switching Detector checks the condition (14) to determine whether a current mipmap level must be changed. When a mipmap level is changed, the outputs of *Pixel Interpolator* and *Texel Interpolator* are not updated to *Context Register*. Instead, the outputs of *Mipmap Switch* are updated. This is one cycle loss to change a mipmap level, but mipmap level shifting is not frequently occurred.

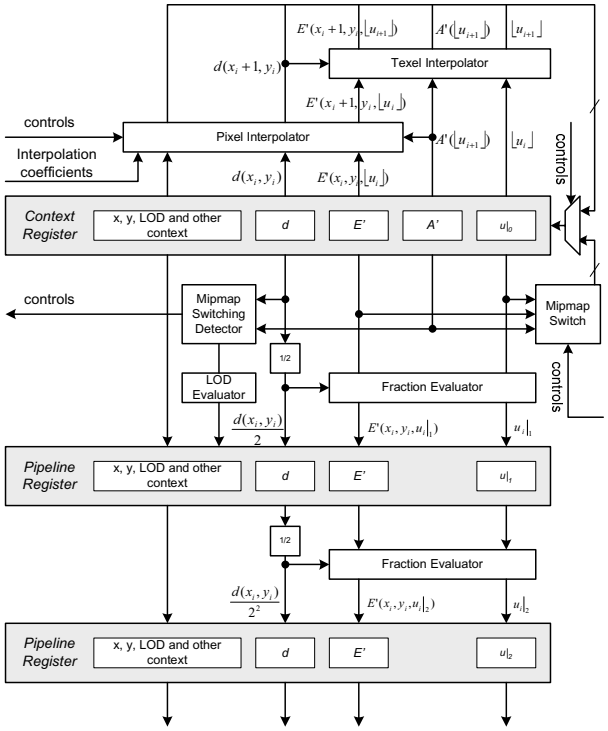

Figure 5: The block diagram of Basic Rasterization Units.

Note that *Mipmap Switching Detector* determines mipmap level shifting by using *A'* of integer texture coordinates. It is mathematically correct to use *A'* of full texture coordinates, but this makes no problem since a precise mipmap level threshold is not critical. *LOD evaluator* generates the fraction of LOD for a trilinear filtering coefficient.

Fraction Evaluator produces the fraction part of texture coordinate by (13). A *Fraction Evaluator* block produces only one fraction bit by (13), but pipelining multiple *Fraction Evaluator* blocks enable to obtain multiple bits of fraction parts without loss of performance. 4 bit fraction is usually enough for trilinear filtering coefficients. Therefore, 4 pipeline stages produce a texture coordinate per cycle.

Fraction Evaluator is very simple, which is only composed of a comparator and an adder, but the registers to contain context of pixel are quite large. Therefore, it is better to cascading several *Fraction Evaluator* blocks in a pipeline stage to produce multiple fraction bits. This reduces the number of pipeline stages instead of increasing delay.

5.2. Implementation

The proposed architecture was verified by GATE [6] and implemented by Verilog HDL. The implemented rasterizer has 4 parallel basic rasterization units for 4 pixels per cycle throughput. Each basic rasterization unit evaluates 4 texture coordinates to support two level multi-textures. All the internal variables and texture coordinates are represented in 16 bit fixed point representation and the fraction parts of texture coordinates are 4 bits. The rasterizer is pipelined into two stages to evaluate 4 bit fraction. It is verified from static timing analysis in 0.13um process that the rasterizer affords 100 Mhz clock in the worst case. The total gate count is 220k.

6. CONCLUSION

To render perspective-correct textured images, divisions are required per pixels. However, using dividers is very expensive in terms of area and speed. Dividers are also deep pipelined to reduce latency, but the pipeline registers to store full pixel context are very large. Pipelined dividers with short latency have large look-up-tables. [7]

To avoid expensive divisions, the proposed rasterizer uses midpoint algorithms. To use midpoint algorithms instead of divisions require also iterative computations, but the integer parts of texture coordinates are evaluated without iterations due to the characteristics of the 3D graphics mipmap structure.

The fraction parts of texture coordinates are evaluated in pipelined methods. The pipeline depth is determined on the precision of only fraction parts while the pipeline depth of divider is related on full precision. The precision of fraction part is usually much less than the precision of integer part. Therefore, the pipeline depth is much shortened and the area overhead by the pipeline registers becomes small. Besides, one or two pipeline stages as we used in hardware implementation is not overhead in the entire graphics systems. Alpha test or pre-depth tests are located between rasterizer and texture engine and executed in parallel with the fraction evaluation.

We implemented the proposed architecture into real hardware. The implemented parallel rasterizer produces 4 pixels with two texture coordinates for multi-texturing per cycle. The rasterizer operates at 100 Mhz clock and hence gives 400M pixels and 800M texture coordinates per second.

7. ACKNOWLEGEMENT

This work is supported in part by SAMSUNG Electronics and A Collaborative Project for Excellence in Basic System IC Technology.

8. REFERENCES

[1] J.F. Blinn, “Hyperbolic Interpolation”, *IEEE Computer Graphics and Applications*, pp. 89-94, July 1992.

[2] M. Pitteway, et al., “Algorithms for drawing ellipses or hyperbolae with a digital plotter”, *Computer Journal*, 10(3), pp. 282-289, 1967.

[3] B. Barenbrug, et al., “Algorithms for Division Free Perspective Correct Rendering”, *Proceeding of SIGGRAPH /EUROGRAPHICS workshop on Graphics Hardware*, pp. 7-13, 2000.

[4] J. Ewins, et al., “MIP-map level selection for texture mapping”, *IEEE Transactions on Visualization and Computer Graphics*, vol.4, no. 4, pp. 17-29, 1998.

[5] M. Demirer, et al. “Approximation Techniques for High Performance Texture Mapping”, *Computer&Graphics*, vol. 0, no. 4, pp 483-490, 1996.

[6] Inho. Lee, et al, “A hardware-like high-level language based environment for 3d graphics architecture exploration”, *Proceedings of the 2003 International Symposium on Circuits and Systems*, vol. 2, pp 512–515, May, 2003.

[7] P.Hung, et al, “Fast division algorithm with a small lookup table”, *Conference Record of the 33rd Asilomar Conference on Signals, Systems and Computers*, Vol.2, pp 1465-1468, 1999.