應用於三維繪圖系統之
低複雜度切割演算法與高能源效率幾何引擎設計

Power Efficient Geometry Engine Using Low-Complexity Subdivision Algorithm for 3D Graphics System

研 究 生：許籐耀

指導教授：范倫達　博士

中 華 民 國 九 十 七 年 十 月

應用於三維繪圖系統之

低複雜度切割演算法與高能源效率幾何引擎設計

A Power Efficient Geometry Engine Using Low-Complexity Subdivision Algorithm for 3D Graphics System

研　究　生：許籐耀　　Student：Ten-Yao Sheu

指導教授：范倫達博士　　Advisor：Dr. Lan-Da Van

國 立 交 通 大 學
資訊科學與工程研究所
碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

Oct 2009

Hsinchu, Taiwan, Republic of China

中華民國 九 十 七 年 七 月

應用於三維繪圖系統之
低複雜度切割演算法與高能源效率幾何引擎設計

學生：許籐耀　　　　　　　　　　指導教授：范倫達 博士

國立交通大學
資訊科學與工程研究所

摘　　要

在本篇論文中，我們提出了一個低複雜度的三角形切割渲染演算法與具有高能源效率的幾何引擎架構。所提出的切割演算法與架構能提供近似馮氏渲染的繪圖品質，同時能藉由調整切割層級達到有動態調整繪圖品質與功率消耗的能力。目前可支援三種不同層級的切割模式。

本文所提出的幾何引擎架構運用了數項不同的技術來對功率消耗、效能、面積進行最佳化。例如使用所提出的前向差分切割、邊函數修正、雙空間切割、三角形濾除與頂點快取記憶體等技術可以有效減少幾何轉換與打光運算。針對複雜的運算，我們提出了可重組的資料路徑架構並藉由將資料路徑重組成特定的結構來加速複雜運算的處理。由於重組架構使用相同的硬體進行不同的運算，晶片面積與成本也能有效的減少。與傳統的切割式渲染演算法相比，分別對於一層與兩層的切割渲染運算，我們提出的方法可以減少記憶體存取次數達 44.44% 與 68.88%，同時幾何轉換乘法運算量也能分別減少 50% 與 80%。此設計已使用 UMC 90 奈米製程實現，從晶片模擬結果顯示 我們所提出幾何引擎可以達到 16.978 MVertices/(s•mW)之高能源效率。

A Power Efficient Geometry Engine Using Low-Complexity Subdivision Algorithm for 3D Graphics System

Student：Ten-Yao Sheu Advisor：Dr. Lan-Da Van

Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University

ABSTRACT

In this thesis, a power efficient geometry engine (GE) using a low complexity three-level subdivision algorithm is presented. The proposed subdivision algorithm and architecture is capable of providing low power, scalable and near-Phong shading quality. The forward difference, edge function recovery, dual space subdivision, triangle filtering schemes and post-TnL vertex are employed to alleviate the redundant computation for transforming and lighting of the proposed algorithm and architecture. Due to the three-level subdivision algorithm, one reconfigurable datapth is proposed to reduce the area since the same set of processing elements (PEs) is reused for different operations for the GE. Compared with the conventional subdivision algorithm, the reduction of the number of memory accesses can be attained by 44.44% and 68.88% for level-1 and level-2, respectively. The reduction of the number of multiplications for transforms can be attained by 50% and 80% for level-1 and level-2, respectively. From the implementation results in UMC 90nm, the proposed geometry engine can achieve the power efficiency of 16.978 Mvertices/(s•mW).

誌　　　　謝

感謝指導教授范倫達老師。在碩士班這段時間老師提供了我各方面的協助，也花了很多時間審查我的論文和思考改進我的研究，使我可以確立並完成我的論文研究，讓我在研究所的期間收穫豐碩，因此在這邊對老師表達由衷感謝。同時，亦感謝周懷樸教授、陳紹基教授、簡韶逸教授三位口試委員給予精闢的審查意見。

其次，我也要感謝 VIPLab 的同學、學長姐、學弟妹們，姵妤學姐，特別是旭昇學長在 EDA 軟體提供了許多協助，以及同窗迪優、宗哲、宗融、晉豪與得安在研究上給予我很多建議。最後就是可愛的學弟學妹們丞蔚、思翰、庭維、坤隆、家育、盈里、建勳、曉霜、家榮、睿峻、泊硯，與你們相處的回憶對我來說都是相當珍貴的。

最後，我要感謝我的父母親還有祖母，你們是我心靈上最大的支柱，你們的關心和鼓勵都是我最大的動力來源，你們讓我能順利完成學業和此篇論文，非常感謝你們。

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nowadays, 3D graphics functions are integrated into the wireless- and wired-multimedia terminals such as mobile devices and 3D TV systems [1]. 3D graphics system is composed of geometry engine (GE) and rasterization engine [2]. In GE, Gouraud shading [3] with per-vertex lighting is widely used because it only applies reflection model [4] on the vertices of the polygons and uses bilinear interpolation to obtain the intensities for the pixels inside the polygons. Although Gouraud shading has less computation complexity than other approaches, it suffers from Mach band effects and produces polygonal highlights on the rendered objects. Phong shading [5] uses bilinear interpolation to obtain the normal vectors for the internal pixels and applies the reflection model on each pixel. Phong shading can produce more smooth and accurate highlights than Gouraud shading. However, it needs to re-normalize the normal vector and computes the light intensity for every pixel inside the polygon. Phong shading possesses high shading quality, but consumes much more power because of the huge computation requirement.

Recently, low computation, satisfactory quality, and power efficiency are the important research issues for hardware design. In order to have near-Phong shading quality with low computation, several approximate Phong shading schemes have been proposed as follows. The Taylor expansion [6] is used to approximate Phong reflection model. The average computation cost is high for the scenes with small polygons or multi-light sources. Spherical interpolation algorithms [7][8] aim to avoid re-normalizing the normal vectors, but the setup must be performed for each scan line and for each light source. Thus, the setup cost is expensive for thin polygons and the multi-light source scenes. The mixed shading [9][10] combines two shading methods. When the highlight covers the polygon, it is rendered using Phong shading. Otherwise, Gouraud shading is employed. Although deferred shading [2] removes the lighting operations on the hidden pixels, the lighting equation is still applied to the visible pixels. To completely eliminate per-pixel lighting quadratic interpolation, the work in [11][12] uses a quadratic function to interpolate light intensities between six points. The quadratic scheme would incur Mach band effect on the edge if the triangle is too large. Therefore, an error control scheme is proposed in [11] to solve this problem. Subdivision scheme [10][13][14][15][16] is another approach to approximate Phong shading. It subdivides a triangle into smaller ones and renders them individually with Gouraud shading. Compared with other per-pixel lighting approximate schemes, only vertices are lit. One attractive feature of subdivision scheme is its ability to scale shading quality dynamically. If higher shading quality is demanded, more small triangles are generated. Otherwise, fewer triangles are generated to reduce the processing time and power consumption. From another point of view, the power can be used more efficiently if the shading quality is scalable.

1.1 Motivation

Although the conventional subdivision algorithm inherently provides scalable and near-Phong shading quality, the computational complexity and power consumption are still large for GE. Thus, we are motivated to propose a low complexity subdivision algorithm and the corresponding power efficient and scalable-quality geometry engine in the thesis.

1.2 Thesis Organization

The rest of the thesis is organized as follows. The proposed subdivision algorithm and the corresponding complexity analysis are described in Chapter 2. In Chapter 3, the proposed GE architecture is presented. The comparison results and chip implementation are addressed in Chapter 4. Last, a brief statement concludes the presentation of this thesis.

Chapter 2

Proposed Low Complexity Subdivision Algorithm

In this chapter, a low complexity subdivision algorithm to approximate Phong shading is proposed. To reduce the redundant memory accesses, the forward difference technique is used to subdivide triangles in the proposed algorithm. Since the forward difference technique is numerical instable, there may be rasterization anomalies on the rendered objects. Hence, an edge function recovery scheme is proposed to remove the rasterization anomalies. In the subdivision-based approximate Phong shading algorithm, the increased number of triangles becomes a potential problem to the computation and power consumption. In order to reduce the complexity of the proposed algorithm, the dual space subdivision scheme, triangle filtering scheme and the triangle setup variable sharing scheme are also presented. The proposed algorithm and schemes are described in detail in the following subsections.

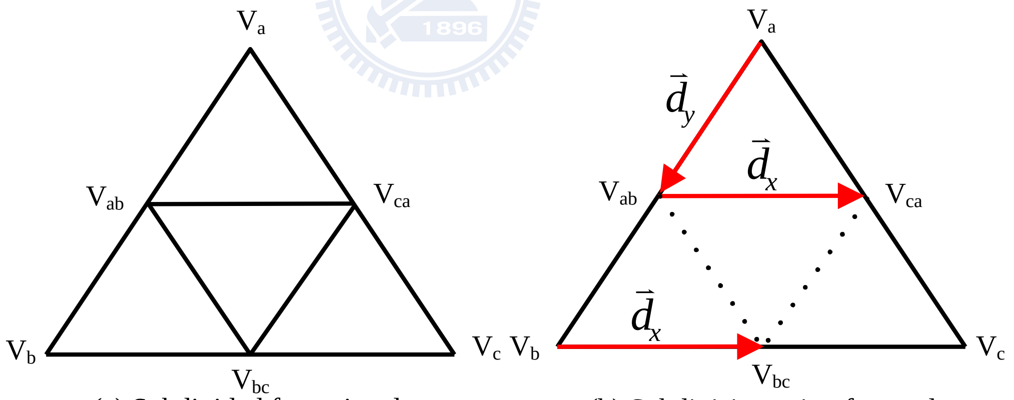
2.1 Subdivision Using Forward Difference

Forward difference [13] method is widely used to evaluate the polynomial function. Herein, use it to reduce the memory accesses for triangle subdivision. An example is illustrated in Fig. 2.1. To subdivide the triangle $\Delta\, V_aV_bV_c$ in Fig. 2.1 (a), the intermediate vertices: V_{ab}, V_{bc}, V_{ca} are computed. Then these new vertices together with the original vertices will be packed and new triangles are generated as: *Δ* $V_aV_{ab}V_{ca}$, *Δ* $V_{ab}V_{bc}V_{ca}$, *Δ* $V_{ab}V_bV_{bc}$ and *Δ* $V_{ca}V_{bc}V_c$. These new triangles will be output for next-stage processing. The forward difference method is used to compute the intermediate vertices. The first step is to compute the difference vectors $\vec{d}_x$ and $\vec{d}_y$ in horizontal and vertical direction using Eq. (2.1) and Eq. (2.2).

$$\vec{d}_x = (V_c - V_b) / N_S \tag{2.1}$$

$$\vec{d}_y = (V_b - V_a) / N_S \tag{2.2}$$

, where $N_S = 2^L$ denotes the number of the segments on each edge of the original triangle and L is a non-negative integer. Without loss of the generality, we set the N_S = 2 as shown in Fig. 2.1(a).



(a) Subdivided four triangle

(b) Subdivision using forward difference

Fig. 2.1. Illustration for subdivision using forward difference.

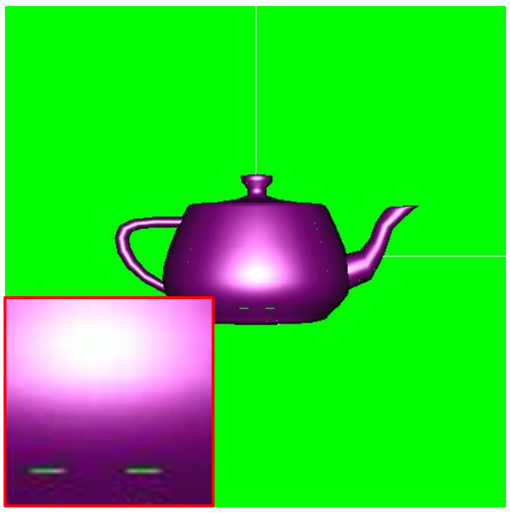
Once the difference vectors are computed, the intermediate vertices can be generated by Eqs. (2.3), (2.4) and (2.5) as shown in Fig. 2.1 (b).

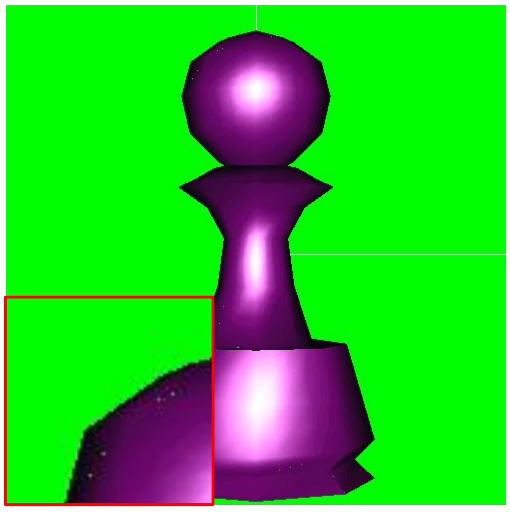
$$V_{ab} = V_a + \vec{d}_y \tag{2.3}$$

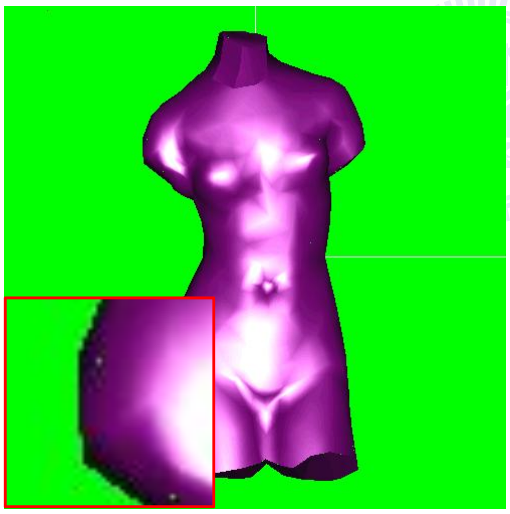
$$V_{ca} = V_{ab} + \vec{d}_x \tag{2.4}$$

$$V_{bc} = V_b + \vec{d}_x \tag{2.5}$$

Computing the intermediate vertices using the forward difference method is more efficient than other methods because generating one intermediate only needs one memory access to store the vertex. Compared with the conventional recursive-based subdivision algorithms [10][13][14][15][16], the forward difference method is stack free and hence the number of memory accesses can be reduced. In other words, the power can be alleviated. However, the subdivision algorithm using forward difference would result in the rasterization anomaly where pixels are lost on the rendered object. As shown in Fig. 2.2(a), (b), (c), (d), the empty pixels on the teapot, pawn, Venus, and couch are the lost pixels. The cause of the anomaly is the numerical instability of subdividing the triangle using the forward difference scheme. An example is illustrated in Fig. 2.3, where two adjacent triangles are subdivided using forward difference. In Fig. 2.3 (a), V_m denotes one intermediate vertex on the sharing edge of two triangles. It can be obtained from subdividing either the left triangle or the right triangle if the calculation has no error. In Fig. 2.3 (a), the vertex V_c is the intermediate vertex in the subdivided left triangle and is computed from the vertex V_b using the difference vector $\vec{d}_x$ twice. The vertex V_c has the same coordinate as the vertex V_m if the calculation has no error. However, the calculation has the quantization error such that the vertex V_c has different coordinate from the vertex V_m. For the same reason, in the right triangle of Fig. 2.3 (a), the vertex V_d computed from vertex V_a with forward difference vector $\vec{d}_y$ has different coordinate from the vertex V_m. As a result, the small triangles defined by vertex V_c and V_d respectively are not adjacent to each other. Fig. 2.3 (b) shows the rasterization result of the sharing edge. Since the pixels are lost on the sharing edge after rasterization, the rasterization anomaly occurs.


(a) Teapot


(b) Pawn


(c) Venus

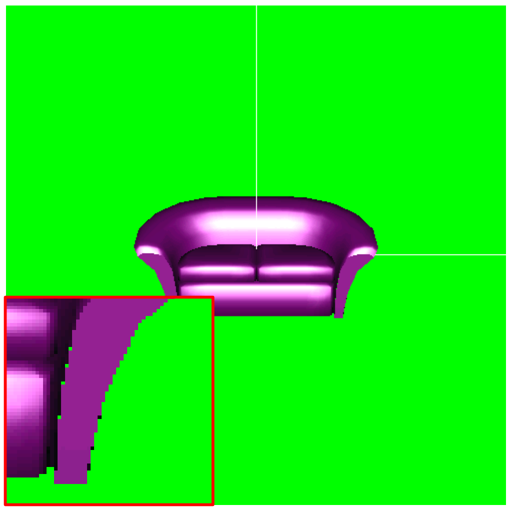
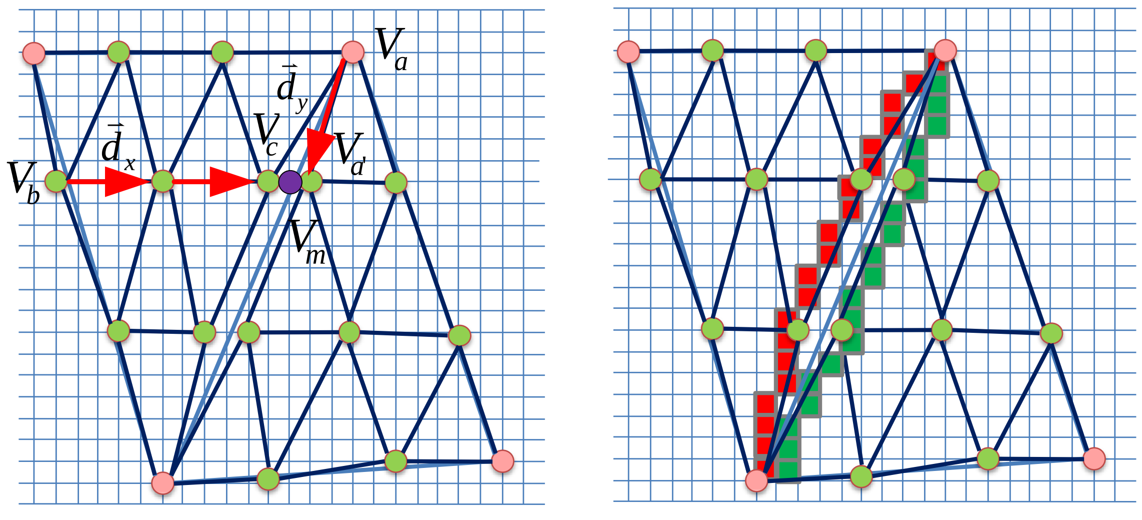

(d) Couch

Fig. 2.2. Examples of rasterization anomaly.



(a) After subdivision (b) Rasterization result

Fig. 2.3. Illustration of the rasterization anomaly.

2.2 Edge Function Recovery Scheme

In order to remove the rasterization anomaly, a recovery scheme based on the edge function method is proposed. The edge function method [16] is used in some raster engine to decide whether a pixel is in the triangle. The edge function is a line equation through the two vertices of the triangle edge. For example, in Fig. 2.4 (a), the edge function E_{ab} of the left triangle defined by vertices V_a and V_b is expressed in Eq. (2.6), where (x_a, y_a) and (x_b, y_b) are the coordinates of vertex V_a and V_b.

$$E_{ab}(x,y):\ A_{ab}x + B_{ab}y + C_{ab} = 0 \qquad (2.6)$$

where $A_{ab} = (y_a\text{-}y_b)$, $B_{ab} = (x_b\text{-}x_a)$ and $C_{ab} = x_a y_b\text{-}x_b y_a$.

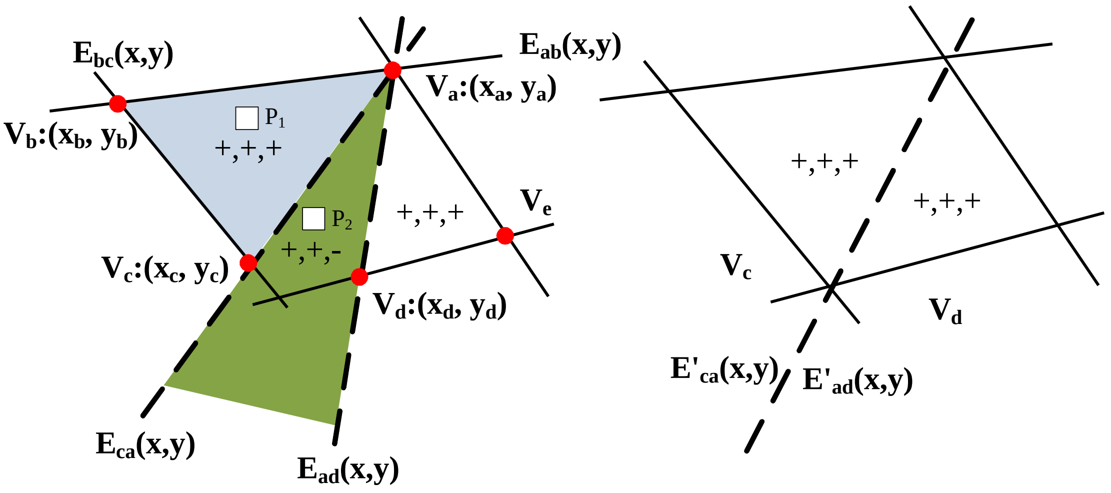
The other two edge functions E_{bc} and E_{ca} can also be similarly derived as follows.

$$E_{bc}(x,y):\ A_{bc}x + B_{bc}y + C_{bc} = 0 \qquad (2.7)$$

where $A_{bc} = (y_b\text{-}y_c)$, $B_{bc} = (x_c\text{-}x_b)$ and $C_{bc} = x_b y_c\text{-}x_c y_b$.

$$E_{ca}(x,y):\ A_{ca}x+B_{ca}y+C_{ca}=0 \tag{2.8}$$

where $A_{ca}=(y_c\text{-}y_a)$, $B_{ca}=(x_a\text{-}x_c)$ and $C_{ca}=x_c y_a\text{-}x_a y_c$.



(a) Before recovery (b) After recovery

Fig. 2.4. Illustration of edge functions.

To test whether a pixel is in a triangle, the coordinate of the pixel is substituted to three edge functions. If the signs of the three calculation result are all positive, the pixel is regarded as an internal point in the triangle. For example, in Fig. 2.4 (a), the pixel P_1 inside the blue triangle has three positive signs of all the edge functions E_{ab}, E_{bc} and E_{ca}.

As demonstrated in Fig. 2.3 (a), the intermediate vertices V_c and V_d of the two triangles have different coordinates. Therefore, they define two different edge functions E_{ca} and E_{ad}, respectively. The different edge functions E_{ca} and E_{ad} are shown in Fig. 2.4 (a). During rasterization, the pixel, for example, P_1 is regarded as an internal pixel of the left triangle because it locates in the blue region which is the positive region for all the edge functions E_{ab}, E_{bc} and E_{ca}. Therefore, P_1 will be rendered correctly. The pixel, for example, P_2 in the green region has negative value for both the edge functions E_{ca} and E_{ad} and is regarded as outside of both the triangles. As a result, the pixels in the green region will be discarded from the pipeline and not be rendered. Therefore, the rasterization anomaly occurs. To eliminate the anomaly, the edge function E_{ca} derived from the left triangle and the E_{ad} derived from the right triangle must be the same. As illustrated in Fig. 2.4 (b), the pixels inside the green region in Fig. 2.5(a) are located at one of the triangles because E'_{ab} and E'_{ad} are the same.

To obtain the same edge function derived, it is improper to use the coordinate of the vertex V_c and V_d for the calculation in Fig. 2.3 (a). Therefore, an edge function recovery scheme is applied to correct edge function calculation. The proposed scheme takes the advantage of linear property of line equation and computes the edge functions for the generated triangles. In Fig. 2.5 (a), a triangle is subdivided into four triangles. After subdivision, the edge functions of the small triangles can be computed in the following steps.

Step 1: Compute the edge functions: E_{ab}, E_{bc}, and E_{ca} of the original triangle using Eqs. (2.6), (2.7) and (2.8).

Step 2: Compute the constant difference values: ΔC_{ab}, ΔC_{bc} and ΔC_{ca} in Eqs. (2.9), (2.10), and (2.11). The slopes of the three edge functions are expressed in the following.

$$\begin{aligned}\Delta C_{ab} &= \frac{1}{2}((x_c - x_b)(y_b - y_a) + (y_c - y_b)(x_a - x_b)) \\ &= \frac{1}{2}(B_{ab}A_{bc} - A_{ab}B_{bc})\end{aligned} \tag{2.9}$$

$$\begin{aligned}\Delta C_{bc} &= \frac{1}{2}((x_a - x_b)(y_c - y_b) + (y_a - y_b)(x_b - x_c)) \\ &= \frac{1}{2}(B_{ab}A_{bc} - A_{ab}B_{bc})\end{aligned} \tag{2.10}$$

$$\begin{aligned}\Delta C_{ca} &= \frac{1}{2}((x_b - x_a)(y_a - y_c) + (y_b - y_a)(x_c - x_a)) \\ &= \frac{1}{2}(B_{ca}A_{ab} - A_{ca}B_{ab})\end{aligned} \tag{2.11}$$

Step 3: Compute the edge functions including E_{ai}, E_{ik}, E_{ka}, E_{ib}, E_{bj}, E_{ji}, E_{kj}, E_{jc}, E_{ck}

of small triangles in Fig. 2.5 with the use of the computed original edge functions and the difference values. For example, E_{kj} can be computed using Eq. (2.12).

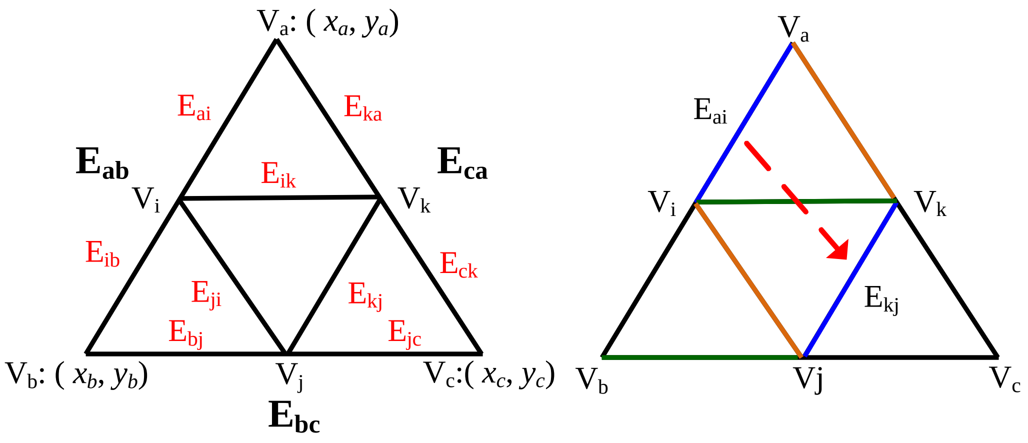
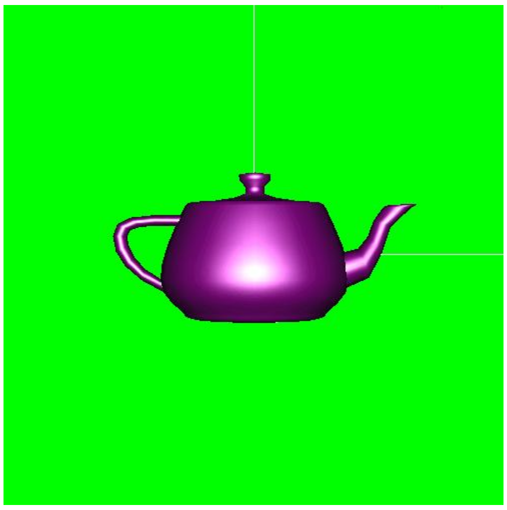


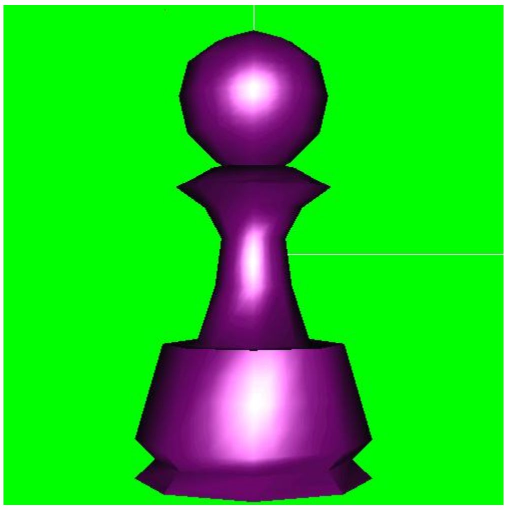
Fig. 2.5. Illustration of computing the edge functions for small triangles.

$$E_{kj}: \ A_{kj}x + B_{kj}y + C_{kj} = 0 \tag{2.12}$$

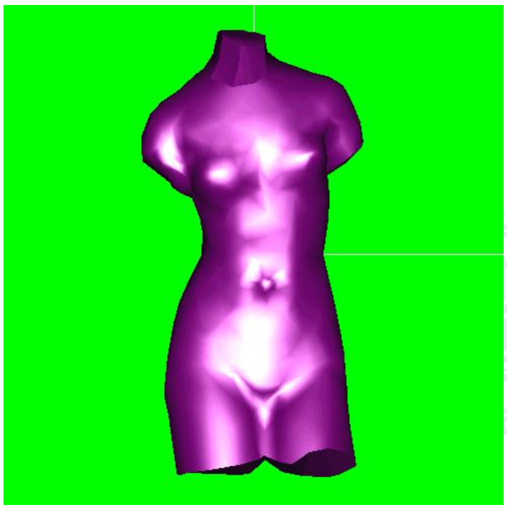
, where $A_{kj} = A_{ab}$, $B_{kj} = B_{ab}$, $C_{kj} = C_{ab} + \Delta C_{ab}$. The constant term C_{kj} can be derived from the constant term C_{ab} of the edge function E_{ab} by adding the difference value ΔC_{ab} in Eq. (2.9). The other edge functions can be computed in the similar behavior. Finally, the small triangles can be rendered with these edge functions. By the proposed method, the derived edge functions on the sharing edge of any adjacent triangles are the same. Therefore, the rasterization anomaly can be eliminated. The rendering results using the proposed edge function recovery scheme are shown in Fig. 2.6 (a), (b), (c), (d).



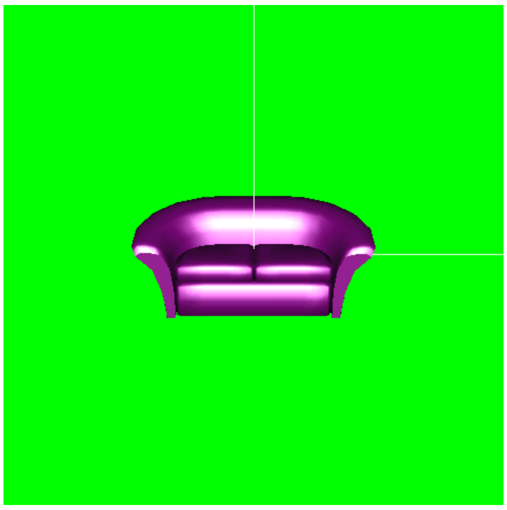
(a) Teapot



(b) Pawn



(c) Venus



(b) Couch

Fig. 2.6. Rendering results with the proposed edge function recovery scheme.

In Eq. (2.6), evaluating one edge function requires three subtractions and two multiplications. For a subdivided triangle with N_s segments on each edge, there are total $3N_S$ edge functions to be computed and computation requires $3N_S$(2 muls + 3 subs) = $6N_S$ muls + $9N_S$ subs = $6N_S$ muls + $9N_S$ adds (subtraction is regarded as addition). The proposed recovery scheme computes each edge function for the subdivided triangle by adding one difference values. Therefore the computation complexity can be reduced to 3(2 muls + 3 adds) + 3(2 muls + 1 add) + ($3N_S$ - 3)(1 add) = 12 muls + ($3N_S$ + 9) adds. Thus, the edge function recovery scheme implies an efficient method for computing the edge functions of subdivided triangles.

2.3 Dual Space Subdivision Scheme

In the geometry engine, a sequence of transforms is applied to the vertices. A flow chart of the transforms is shown in Fig. 2.7.

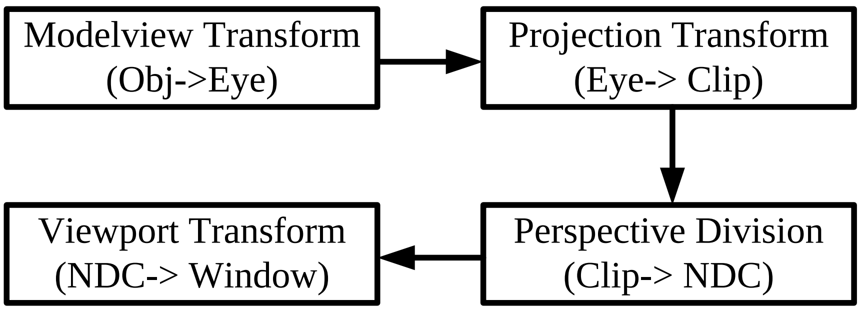


Fig. 2.7. Flow chart of the transforms in the geometry engine.

The modelview transform transforms the vertex from object space to eye space by multiplying a 4x4 modelview matrix below.

$$\begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{bmatrix} = \begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix} \begin{bmatrix} x_{obj} \\ y_{obj} \\ z_{obj} \\ w_{obj} \end{bmatrix} \tag{2.13}$$

In the projection transform, the eye space coordinate is transformed to clip space by multiplying a 4x4 projection matrix below.

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 \\ p_4 & p_5 & p_6 & p_7 \\ p_8 & p_9 & p_{10} & p_{11} \\ p_{12} & p_{13} & p_{14} & p_{15} \end{bmatrix} \begin{bmatrix} x_{eye} \\ y_{eye} \\ z_{eye} \\ w_{eye} \end{bmatrix} \tag{2.14}$$

After clipping, the vertices in the clip space will be projected to the projection plane by dividing the *w* component below. After the perspective division, the normalized device coordinate (NDC) of each component in the range of [-1, 1] can be expressed in Eq. (2.15).

$$\begin{bmatrix} x_{NDC} \\ y_{NDC} \\ z_{NDC} \end{bmatrix} = \begin{bmatrix} x_{clip} / w_{clip} \\ y_{clip} / w_{clip} \\ z_{clip} / w_{clip} \end{bmatrix} \tag{2.15}$$

Finally, through the viewport transform (viewport mapping), the NDC will be transformed to the window (screen) coordinate.

$$\begin{bmatrix} x_{window} \\ y_{window} \\ z_{window} \end{bmatrix} = \begin{bmatrix} x_{scale} \cdot x_{NDC} + x_{offset} \\ y_{scale} \cdot y_{NDC} + y_{offset} \\ z_{scale} \cdot z_{NDC} + z_{offset} \end{bmatrix} \tag{2.16}$$

The conventional subdivision-based algorithm subdivides the triangles in the object space or the eye space. As illustrated in Fig. 2.8, the subdivision is performed at the early stage of the pipeline. Because the subdivision generates a large number of vertices, theses vertices bring overhead to the computation and the power consumption to the later stages of pipeline. To reduce the complexity, the dual space subdivision is proposed.

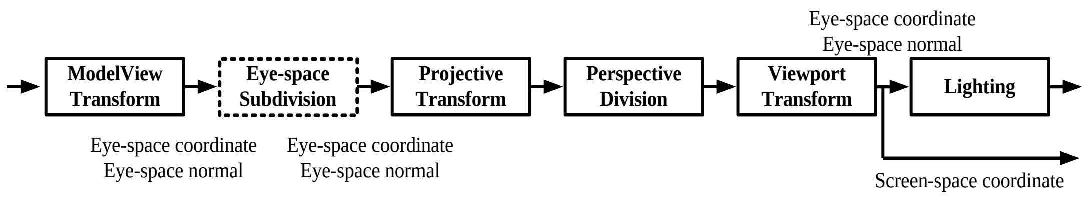


Fig. 2.8. Data flow of eye space subdivision.

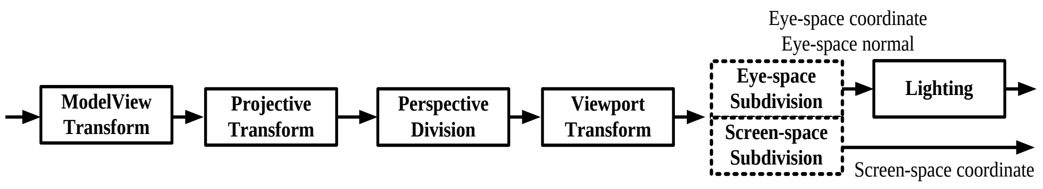


Fig. 2.9. Data flow of dual space subdivision.

As illustrated in Fig. 2.9, the subdivision of the proposed scheme is performed after the viewport transform of the pipeline. It subdivides both the coordinates in eye space and window space. The eye space coordinate is required for point-light calculation and the screen space coordinate is used for edge function calculation and other geometry operations. By skipping these transforms including projection transform, perspective division and viewport transform, the computational complexity is remarkably reduced.

The complexity analysis of the eye space subdivision of a single triangle is given in Table 2.1. The left column lists the operations of subdivision and the corresponding complexity is listed in the right column. N_G is defined as the number of the generated intermediate vertices during subdivision. After the triangle is subdivided, there are (N_G+3) vertices including the original three vertices. First, the triangle is subdivided in eye space. Each step of the subdivision algorithm involves two vector-additions for eye coordinate (x_E, y_E, z_E) and normal vectors (x_N, y_N, z_N) with total six additions. Therefore, the addition complexity of subdivision is 6(N_G+2) additions where two is the number of steps to calculate the difference vectors. After subdivision, all (N_G+3) vertices will be transformed by projection transform, perspective division and viewport transform. As described in this subsection, the projection matrix is a 4x4 matrix and the computational complexity of the projection transform is equal to 16(N_G+3) muls + 12(N_G+3) adds. The perspective division for a vertex requires three multiplications and one inverse and therefore the total computational complexity is 3(N_G+3) muls + (N_G+3) invs for (N_G+3) vertices. The viewport transform requires three multiplications and three additions for each vertex. The computational complexity is 3(N_G+3) muls + 3(N_G+3) adds for (N_G+3) vertices.

Table 2.1: Complexity analysis of the eye space subdivision

Operations	Computational Complexity
Subdivision for 6 components : Eye coordinate: (x_E, y_E, z_E) Normal: (x_N, y_N, z_N)	6(N_G+2) adds
Projection transform for N_G+3 vertices	16(N_G+3) muls + 12(N_G+3) adds
Perspective division for N_G+3 vertices	3(N_G+3) muls + (N_G+3) invs
Viewport transform for N_G +3 vertices	3(N_G+3) muls + 3(N_G+3) adds
Total	(22N_G+66) muls (21N_G+57) adds (N_G+3) invs

Compared to the eye space subdivision, the dual space subdivision subdivides triangles after the viewport transform. Thus, the projective transform, perspective division and viewport transform are performed for the three vertices. The complexity is listed in Table. 2.2. After the viewport transform, the eye coordinates, normal vector and the window coordinate will be subdivided. To have perspective correct eye coordinates and normal vectors for the intermediate vertices, a setup for perspective correction is performed by dividing the eye coordinates and normal vectors by the w_{clip} term. The computational complexity of the setup is six multiplications for each vertex. After setup, the subdivision is performed for the coordinates in two spaces and the normal vector and the computational complexity is 10(N_G+2) additions. The final step is perspective correction which divides the eye coordinates and normal vectors by the $1/w_{clip}$ term of intermediate vertices. Since there are N_G intermediate vertices and six divisions are required for each perspective correction, the computational complexity is $6N_G$ muls + N_G invs.

Table 2.2: Complexity analysis of the perspective correct dual space subdivision

Operations	Computational Complexity
Projective transform for 3 vertices	3x16 muls + 3x12 adds
Perspective division for 3 vertices	3x3 muls + 3 invs
Viewport transform for 3 vertices	3x3 muls + 3x3 adds
Setup for perspective correctly subdivision	3x6 muls
Subdivision for 10 components: Eye coordinate : $(\frac{x_E}{w_{clip}}, \frac{y_E}{w_{clip}}, \frac{z_E}{w_{clip}})$ Normal: $(\frac{x_N}{w_{clip}}, \frac{y_N}{w_{clip}}, \frac{z_N}{w_{clip}})$ Screen coordinate: $(x_w, y_w, z_w, \frac{1}{w_{clip}})$	$10(N_G+2)$ adds
Perspective correction	$6N_G$ muls + N_G invs
Total	$(6N_G+84)$ muls $(10N_G+65)$ adds (N_G+3) invs

The dual space subdivision is able to provide the perspective correct eye coordinates and normal vectors for light intensity calculation. The computation can be further reduced while the perspective incorrectly subdivision is used and the setup and correction can be skipped. This perspective incorrectness of the intensity on the rendered object can be neglected because human eye is not sensitive to the light intensity of small difference. The complexity of the proposed perspective incorrectly dual space subdivision scheme is listed in Table. 2.3.

Table 2.3: Complexity analysis of the perspective incorrect dual space subdivision

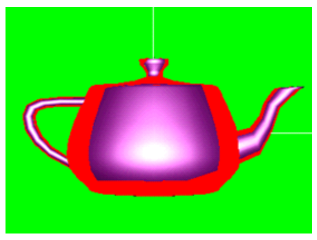
Operations	Computational Complexity
Projective transform for 3 vertices	3x16 muls + 3x12 adds
Perspective division for 3 vertices	3x3 muls + 3 invs
Viewport transform for 3 vertices	3x3 muls + 3x3 adds
Subdivision for 10 components: Eye coordinate: (x_E, y_E, z_E) Normal: (x_N, y_N, z_N) Screen coordinate: $(x_w, y_w, z_w, \frac{1}{w_{clip}})$	$10(N_G+2)$ adds
Total	66 muls $(10N_G+65)$ adds 3 invs

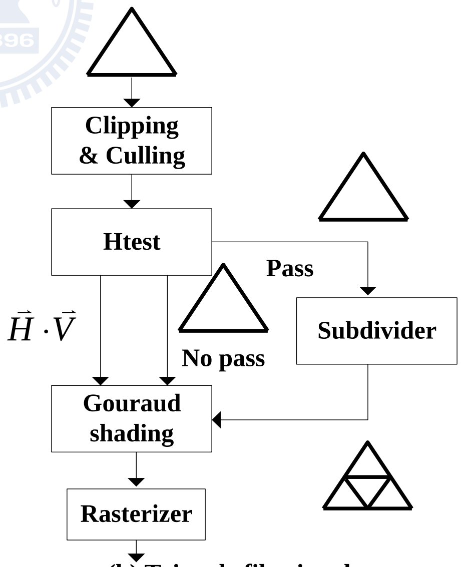
2.4 Triangle Filtering Scheme

To reduce the computation for primitive-level operations, the filtering scheme as shown in Fig. 2.10 is added to the proposed algorithm. The filtering scheme is a hybrid scheme that combines culling/clipping before subdivision and highlight test.

The backface culling in the graphics pipeline is used to test whether a triangle is a backface to the eye direction by the sign of the inner product of the face normal vector and eye direction vector. If a triangle is a backface, it will be discarded and not rendered. In the subdivision algorithm, a triangle will be subdivided into small ones. Performing culling test for these triangles individually brings significant overhead to the computation and power consumption. Because the generated triangles and the original triangle are on the same plane, the face normal vectors are parallel to each other. Therefore, the inner products of these face normal vectors and the eye direction vector will be the same. The statement implies that there is no need to perform backface culling test for each generated triangle since the results will be the same. Hence, in the proposed algorithm, the subdivision is performed after culling test. If the original triangle is culled, the subdivision is unnecessary. Otherwise, all generated triangles are rendered without culling test. Clipping is another primitive level operation in the pipeline. Since the subdivision is performed after clipping, the generated triangles of the clipped original triangle are guaranteed to be inside the view frustum. Therefore, it is not necessary to re-clip these triangles.

To reduce the redundant subdivision, the subdivision-based algorithm usually includes the highlight test scheme. In the proposed algorithm, the mixed-shading [9][10] for the highlight test is adopted. The scheme tests the $\vec{H} \cdot \vec{V}$ term of the original three vertices. While one of the $\vec{H} \cdot \vec{V}$ term is greater than the threshold value, the triangle will be subdivided. If all $\vec{H} \cdot \vec{V}$ terms are smaller than the threshold value, we bypass the subdivision and render the triangle with Grouaud shading. Thus, the redundant primitive operations can be reduced.


(a) H test passed region



(b) Triangle filtering data flow

Fig. 2.10. Data flow of the triangle filtering scheme.

2.5 Triangle Setup Variable Sharing Scheme

To reduce the triangle setup and the unnecessary subdivision for vertex attributes, a triangle setup variable sharing scheme is exposed in this section. The concept of the setup reusing result has been shown in [15]; however, the detailed description is not given. During rasterization, the vertex attributes are linearly interpolated for each pixel. These attributes include screen coordinates, texture coordinates, depth values, fog factors, light intensities and etc. The interpolation usually makes the use of the plane equation [17]. An example is given in Fig. 2.11, where (x_i, y_i) is the window coordinates of the triangle and u_i is the attribute to be interpolated. The attribute plane defined by u_i is obtained by solving Eq. (2.17).

$$\begin{cases} u_0 = A_i \cdot x_0 + B_i \cdot y_0 + C_i \\ u_1 = A_i \cdot x_1 + B_i \cdot y_1 + C_i \\ u_2 = A_i \cdot x_2 + B_i \cdot y_2 + C_i \end{cases} \tag{2.17}$$

After solving the plane coefficients A_i, B_i and C_i, the attribute u_i for any pixel in the triangle can be obtained by substituting the coordinate into Eq. (2.17). Generally, Eq. (2.17) can be recast in Eq. (2.18) and Eq. (2.19).

$$[u_0 \quad u_1 \quad u_2] = [A_i \quad B_i \quad C_i] \begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.18}$$

$$[A_i \quad B_i \quad C_i] = [u_0 \quad u_1 \quad u_2] \begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \tag{2.19}$$

In Eq. (2.19), the coefficients [A_i ,B_i, C_i] of any attribute plane can be computed by multiplying a 3x3 inverse matrix, where the inverse matrix is composed of the coordinates [x_i, y_i, 1] of the triangle. Therefore, once the inverse matrix is available, it can be used to compute any coefficient for interpolating the attributes of the same triangle. Thus, the cost for setup one attribute is generally a 3x3 matrix multiplication.

The generated triangles of the subdivision algorithm increase the complexity for triangle setup. Because the generated triangles are on the same plane, they define the same attribute plane for each attribute. The coefficients of the attribute planes can be shared by the generated triangles without re-computing these coefficients. As illustrated in Fig. 2.11, the triangle is subdivided into four small triangles and therefore the original setup cost for one vertex attribute of these triangles are four 3x3 matrix inversions and four 3x3 matrix multiplications. With the setup variables sharing scheme, the setup only requires one 3x3 matrix inversion and one 3x3 matrix multiplication because the pre-computed variables are shared by the small triangles. Reusing these coefficients eliminates the subdividing and the setting up vertex attributes for the small triangles. Most rasterization algorithms start rasterization from a pixel with initial attribute values and evaluate the attribute values of next pixel in an incremental manner. It is necessary to compute the initial attribute values for each generated triangles in Eq. (2.20). It takes three multiplications to re-setup for each generated triangle in tile-based traversal scheme [16].

$$u = A_i \cdot \Delta x + B_i \cdot \Delta y + C_i = [A_i \quad B_i \quad C_i] \begin{bmatrix} \Delta x \\ \Delta y \\ 1 \end{bmatrix} \tag{2.20}$$

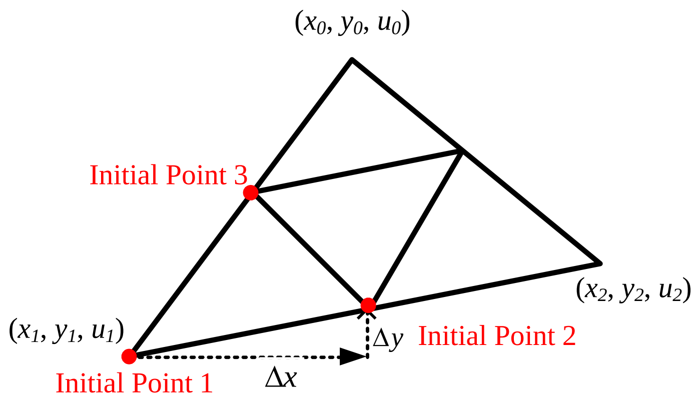


Fig. 2.11. Illustration of the triangle setup variable sharing.

Chapter 3

Proposed Geometry Engine Architecture

In this chapter, a power efficient geometry engine (GE) architecture for 3D graphics pipeline architecture is proposed. Several kernel blocks including the primitive input control (PIC), the primitive processing unit (PPU), vertex processing unit (VPU) and vertex cache management unit (VCMU) are proposed to optimize the power consumption and to support the scalable quality mechanism via the proposed subdivision algorithms. The proposed GE supports the scalable quality mechanism via the proposed subdivision algorithm. The users can choose the most efficient configuration for the graphics processing according to the requirements of the shading quality and the power budget. The supported scalable quality levels are level-0, level-1 and level-2. The overall architecture of the proposed GE is depicted in Fig. 3.1 and the detailed descriptions of each block are given in the following subsections.

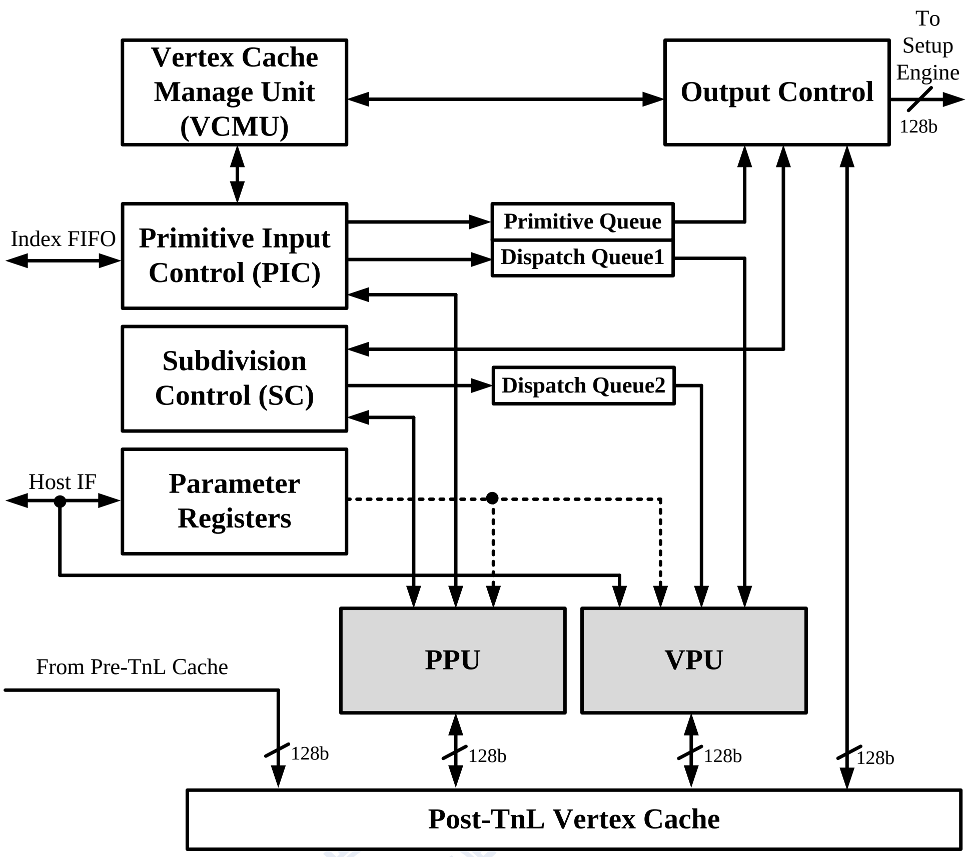


Fig. 3.1. Overall architecture of proposed GE architecture.

3.1 Primitive Input Control (PIC)

The primitive input control (PIC) processes the input primitive information from the host. The PIC reads one index from index FIFO at a time and accesses cache tag to check whether the vertex with the index exists in the vertex cache. Once the cache misses, the PIC requests fetching the vertex data (object coordinate and normal vectors) from the pre-TnL cache. The vertex data returned from the pre-TnL cache will be stored in the post-TnL cache. If the cache hits, the vertex data are not fetched because it is already in the post-TnL cache. The triangles defined by the indices are assembled in PIC and then the backface culling test is issued for the assembled triangles. If the triangle is backface, it will be discarded from PIC. Otherwise, the triangle is pushed to the primitive queue (PQ) and the vertices that belong to the triangle are push the dispatch queue 1 (DQ1) for further processing.

3.2 Primitive Queue (PQ)

The primitive queue (PQ) is a FIFO that buffers the triangles for processing. Each entry of PQ stores the cache entries of three vertices of a triangle. The triangle that passed the culling test is pushed to PQ by PIC. After all vertices of the triangle are transformed and lit, the output control pops the triangle from PQ and read the vertex data (window coordinate and light intensity) of the triangle from vertex cache memory and then output to the setup engine.

3.3 Dispatch Queue (DQ)

The dispatch queue (DQ) is used to keep the vertices under processing. As illustrated in Fig. 3.2, the dispatch queue contains two vertex-cache-entry buffers. The vertex-cache-entry buffer contains the entry addresses of the vertices in the cache. The VPU can access the vertex data with this information. When the vertices in buffer 1 in Fig. 3.2 are processed in VPU, the PIC is able to continue pushing unprocessed vertices into buffer 2 in Fig. 3.2. After all vertices of buffer 1 are processed and buffer 2 is full, the buffers swap. Then, the VPU processes the vertices in buffer 2 and the PIC pushes the unprocessed vertices to buffer 1. With the ping-pong buffer architecture, the PIC and VPU can operate concurrently and thus the performance is increased. In DQ, the size of each buffer is six which is the optimized size for the three-level subdivision algorithm.

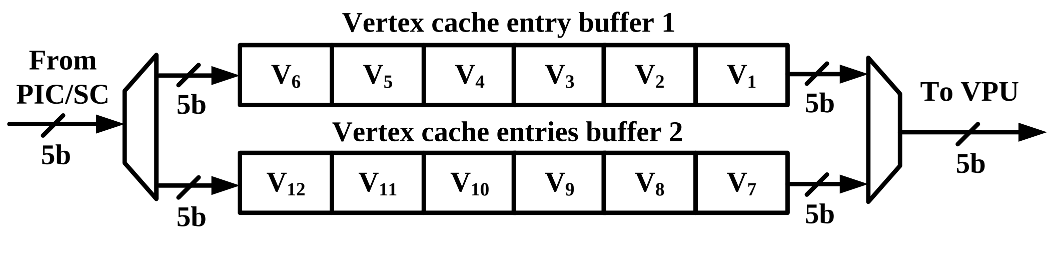


Fig. 3.2. Illustration of the dispatch queue.

3.4 Vertex Cache Management Unit (VCMU)

The vertex cache manage unit (VCMU) is a vertex cache tag unit for the post-TnL vertex cache. The post-TnL cache contains 16-tag entries and each entry has seven fields as illustrated in the first entry in Fig. 3.3. Compare with other works [19], the tag entry contains a reference count field to trace the number of references to the vertex in the tag entry. The primitive processing unit (PPU), VPU, and PIC can operate concurrently. The reference count field is necessary to prevent the data of the vertices which are under processing from being replaced by the incoming vertices. When the PIC requests the VCMU to check whether a vertex exists in the cache, the searched vertex index is compared with the index field of each tag entry. When the index matches one of the valid tag entries, the Entry_hit signal of the tag entry asserts and the VCMU returns hit signal. Since one more vertex enters the pipeline and refers to the data in the cache entry, the value of the reference count field of the tag entry is added by one. The entry address of the vertex is obtained by encoding the hit_vector in Fig. 3.3 and returned to PIC. If the index does not match any tag entry, the VCMU returns miss signal. Before PIC requests fetching the vertex data, the VCMU searches for one free tag entry and allocates it to the vertex. A tag entry is available to be allocated when the valid field is 0 or the reference is 0. When these conditions are met, the Entry_free signal asserts. The allocated entry address for the vertex is obtained by encoding the free_vector in Fig. 3.3 and returned to PIC. In Fig. 3.3, the reference count in the tag entry subtracts by one when a vertex referring to it exits the pipeline. When the cache hits, the in_pipe and lit fields indicate that the vertex is processed in the pipeline and lit, respectively. When one of the two fields is set to 1, the vertex is not pushed into DQ since it is already been transformed and lit. The window coordinate and intensity can be read from the cache directly. The Htest_result field stores the result of the highlight test for the subdivision algorithm. With this field, the power can be reduced because the highlight test for the stored vertex is only performed once and the result can be reused by the triangles the vertex.

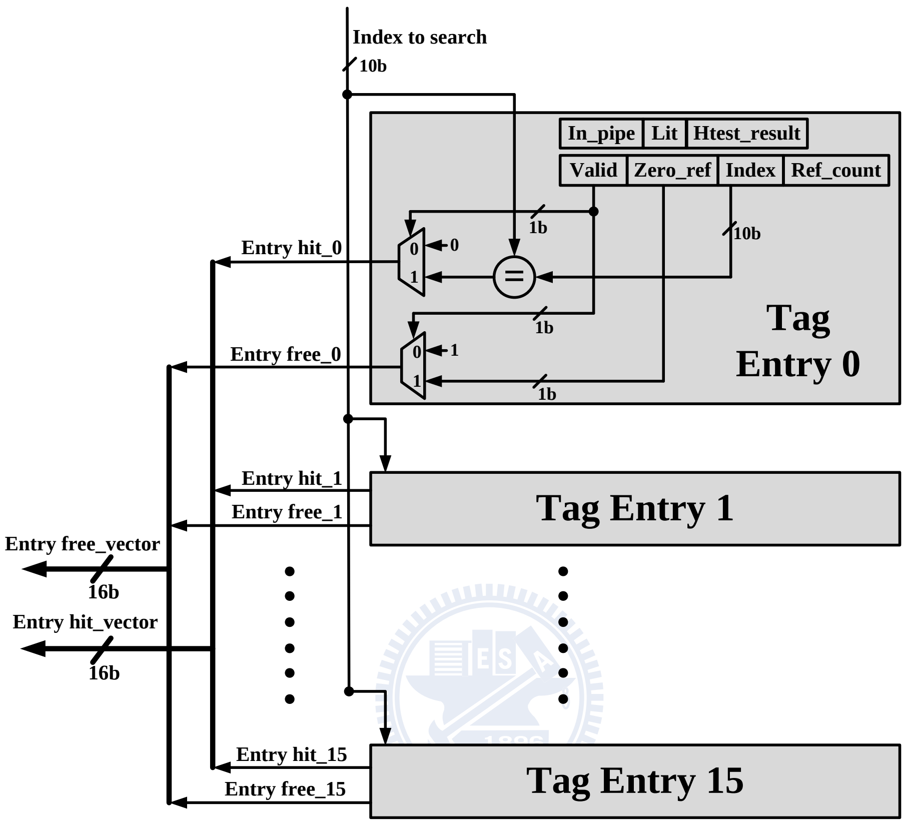


Fig. 3.3. Illustration of the vertex cache management unit.

3.5 Primitive Processing Unit (PPU)

The primitive processing unit (PPU) performs primitive-level operations including backface culling and subdivision algorithm. The backface culling is performed in object space [20] to remove the unnecessary transforms for the vertices of the culled triangles. The subdivision algorithm makes the use of the forward difference to subdivide the triangles. These operations are similar such that the datapath architecture can reused for area efficiency. The block diagram of the proposed PPU architecture is depicted in Fig. 3.4, where the bit with of each node has been marked for clear representation. As illustrated in Fig. 3.4, before culling or subdivision starts, the controller loads the data of the three vertices of the triangle in the cache into the input buffers. The eye position in the object space is stored in the eye position buffer which is set by the host when the eye position is updated. The PPU is able to write the data into the vertex cache because the subdivision algorithm generates intermediate vertices. These vertices are written back to the cache and be read by the VPU for further processing.

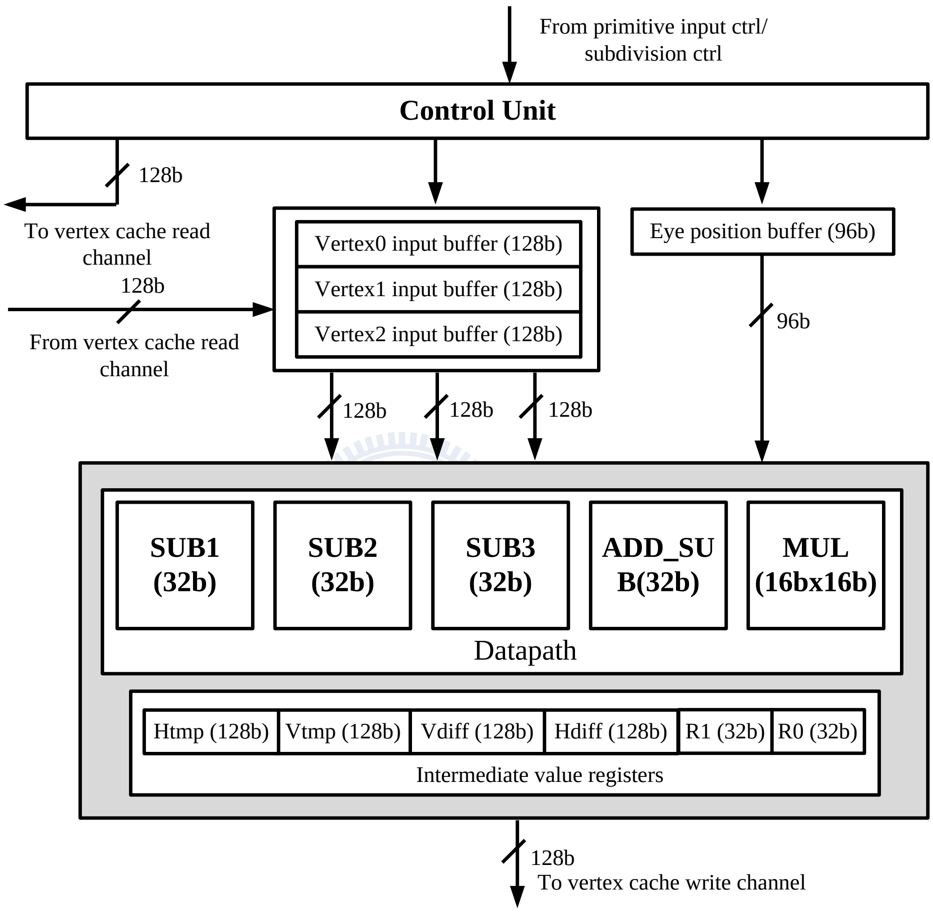

From primitive input ctrl/ subdivision ctrl
Control Unit
128b
To vertex cache read channel
128b
From vertex cache read channel
Vertex0 input buffer (128b)
Vertex1 input buffer (128b)
Vertex2 input buffer (128b)
Eye position buffer (96b)
96b
128b
128b
128b
SUB1 (32b)
SUB2 (32b)
SUB3 (32b)
ADD_SUB(32b)
MUL (16bx16b)
Datapath
Htmp (128b)
Vtmp (128b)
Vdiff (128b)
Hdiff (128b)
R1 (32b)
R0 (32b)
Intermediate value registers
128b
To vertex cache write channel

Fig. 3.4. Block diagram of the primitive processing unit.

3.6 Vertex Processing Unit (VPU)

The vertex processing unit (VPU) performs vertex-level operations including vertex transformation and lighting operation. The operations covers modelview transform, projection transform, perspective division, normal transform, viewport transform, vector normalization and Blinn-Phong reflection model. The Blinn-Phong reflection model [4] can be formulated in Eq. (3.1).

$$I = I_a + (\vec{N} \cdot \vec{L}) I_d + (\vec{N} \cdot \vec{H})^n I_s \tag{3.1}$$

Where I_a, I_d, I_s, $\vec{N}$, $\vec{L}$, $\vec{H}$ denote the ambient intensity, diffuse intensity, specular intensity, normalized normal vector, normalized light vector, and normalized halfway vector, respectively. The halfway vector $\vec{H} = \frac{\vec{L} + \vec{V}}{2}$ is the vector between the light direction vector $\vec{L}$ and the viewing vector $\vec{V}$. In Fig. 3.1, so as to maximize the performance, the VPU is designed to process a batch of vertices in DQ at the same time. The block diagram of VPU architecture is depicted in Fig. 3.5, where the bit width of each node has been marked for clear representation. The vertex data are read from the read channel of the vertex cache. Then, they are transformed and lit in the reconfigure datapath (RDP). The register file stores the intermediate values for the vertices under processing. The constant memory stores the matrix parameters and the light parameters for transforms and lighting, respectively. The content of the constant memory is set by the host before the GE starts. Finally, the vertices are read from the register file and written back to vertex cache when all vertices in the batch are transformed and lit.

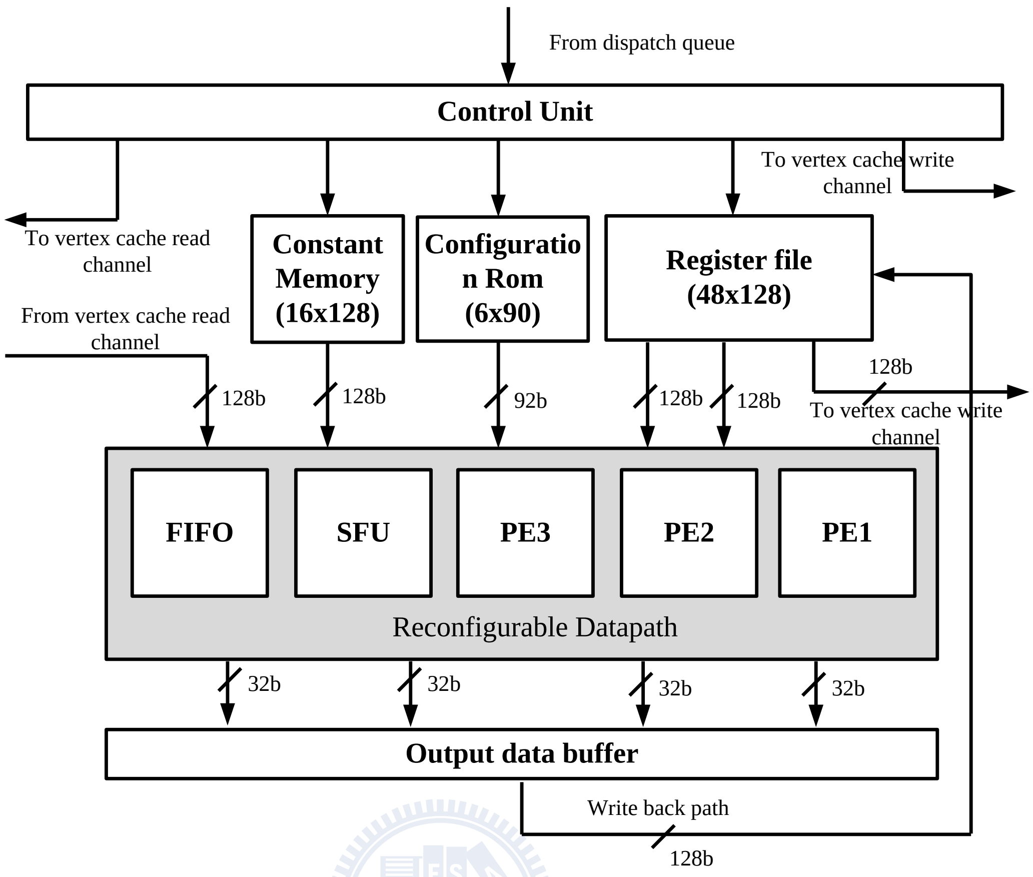


Fig. 3.5. Block diagram of the vertex processing unit.

Considering the trade-off for the power, area and vertex processing performance of the GE architecture, the operations listed in Chapter 3 are disassembled into the simpler atomic operations. For example, the modelview transform involves a 4x4 matrix multiplication which can be achieved by four dot product operations. The un-disassembled operations and the atomic operations define the minimum set of operations supported by the RDP. The RDP can be configured to different modes to achieve these operations. These configuration modes are summarized in Table 3.1. The RDP is a pipelined SIMD datapath architecture for high performance vertex processing. The RDP is composed of three processing elements (PEs), one special function unit (SFU) and one FIFO as shown in Fig. 3.5. By reconfiguring three PEs, the SFU and the interconnection between these PEs, the RDP can realize all the transform and lighting operations listed in Table 3.1. For a complicated operation such as the vector normalization, the RDP is configured to be an efficient pipelined datapath. By processing a batch of vertices at the same time and filling the pipeline, the average cycle for the operation is reduced compared with other architectures that process one vertex one at one time. The detailed descriptions about the RDP are given in the following subsections.

Table 3.1: Configuration modes for RDP

Configuration Mode	Function Description
trans_dp	Dot product for transform
light_dp	Dot product for lighting
vec_norm	Vector normalization
Pd	Perspective division
Pow	Powering
vec_sub	Vector subtraction

3.6.1 Processing Element (PE)

The architecture of the processing element (PE) is illustrated in Fig. 3.6, where the PE is a three-stage pipeline. At the first stage, the 32-bit fixed-width Booth multiplier multiplies two numbers and generates two partial products. The 32-bit fixed-width Booth-based squarer [21] is used to perform squaring operation. A dedicated squarer consumes less power dissipation than that of a general-purpose multiplier. The outputs of the squarer are two partial products. At the second stage, the 32-bit 4-2 compressor is used to add four inputs and generates two partial products. Finally, at the last stage, the adder-subtractor unit adds or subtracts two numbers and produces the final output. The function of the adder-subtractor is controlled by the MODE signal in Fig. 3.6. The multiplexers in PE control the data flow for different operations. The PE can be configured to perform multiplication (MUL), square (SQR), multiplication-accumulation (MAC), addition (ADD) and subtraction (SUB) as shown in Figs. 3.7, 3.8, 3.9, and 3.10, respectively.

The datapath of multiplication (MUL) operation is illustrated in Fig. 3.7 and marked with dashed lines. The first stage of MUL generates the partial products by multiplying two numbers of the input registers REG_B and REG_C. The partial product outputs are registered in the pipeline registers REG_F and REG_G and then are summed up in the adder-subtractor unit. The datapath of square (SQR) operation is illustrated in Fig. 3.8. The squarer squares the number in the input register REG_D and generates two partial products. The partial products are registered in the pipeline register REG_H and REG_I and then are summed up in the adder-subtractor unit. The datapath of the multiplication-accumulation (MAC) operation is illustrated in Fig. 3.9. For the MAC operation, the number in the input register REG_B is multiplied by the number in the input register REG_C and the result is added to the number in the input register REG_A to produce a result of MAC. At the first stage, the numbers in REG_B and REG_C are multiplied and the partial products are registered in the pipeline register REG_F and REG_G. The number in the register REG_A is directly passed to the pipeline register REG_E. At the second stage, the partial products in REG_F and REG_G and the number in REG_E are added with the 4-2 compressor and the resulting partial products are registered in the REG_J and REG_K. At the last stage, the partial products in register REG_J and REG_K are summed up in the adder-subtractor unit to produce the result. The datapath of addition (ADD) and subtraction (SUB) operations are illustrated in Fig. 3.10. The pipeline registers REG_J and REG_K are configured to be the input registers for the ADD and SUB operations. The numbers in REG_J and REG_K are added or subtracted according to the target operations.

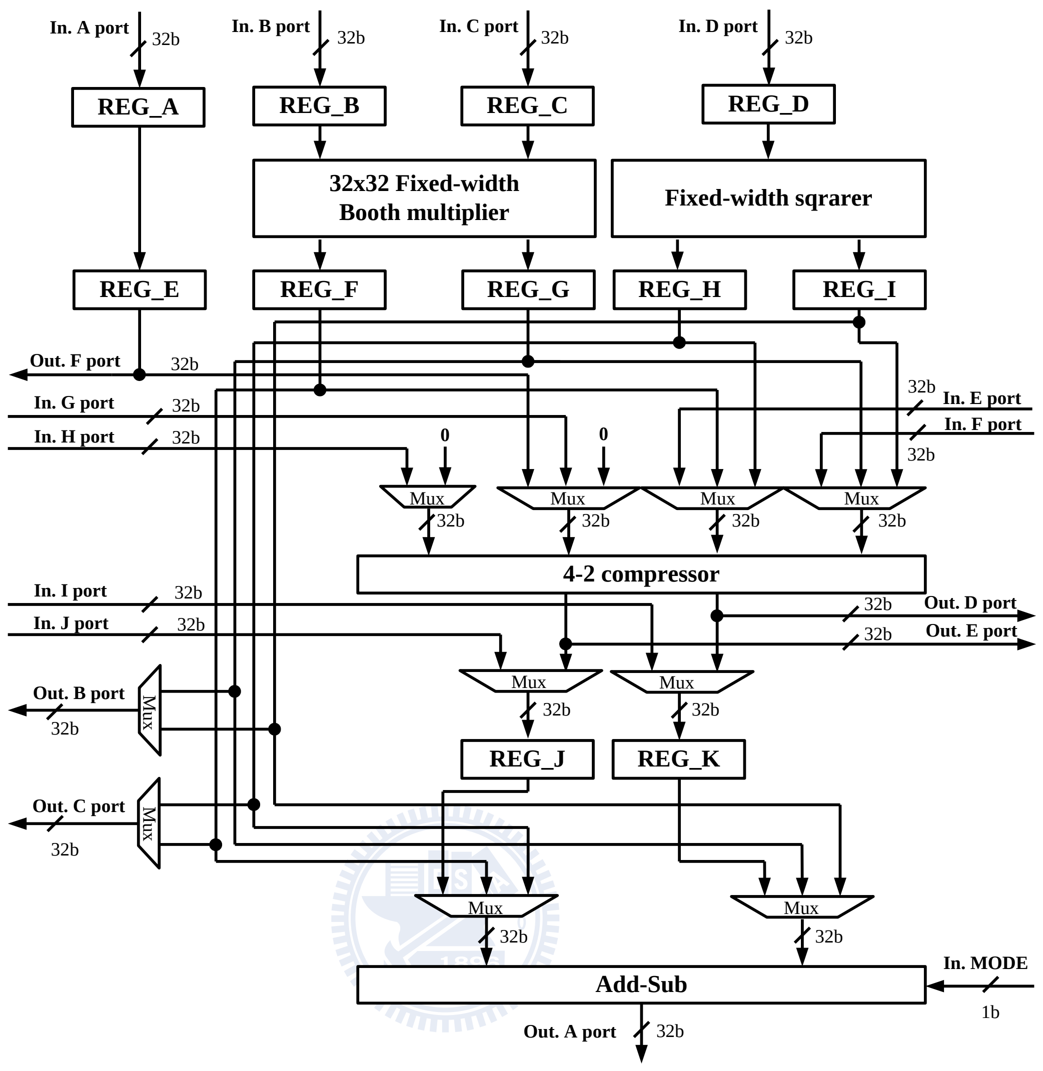


Fig. 3.6. Block diagram of the processing element.

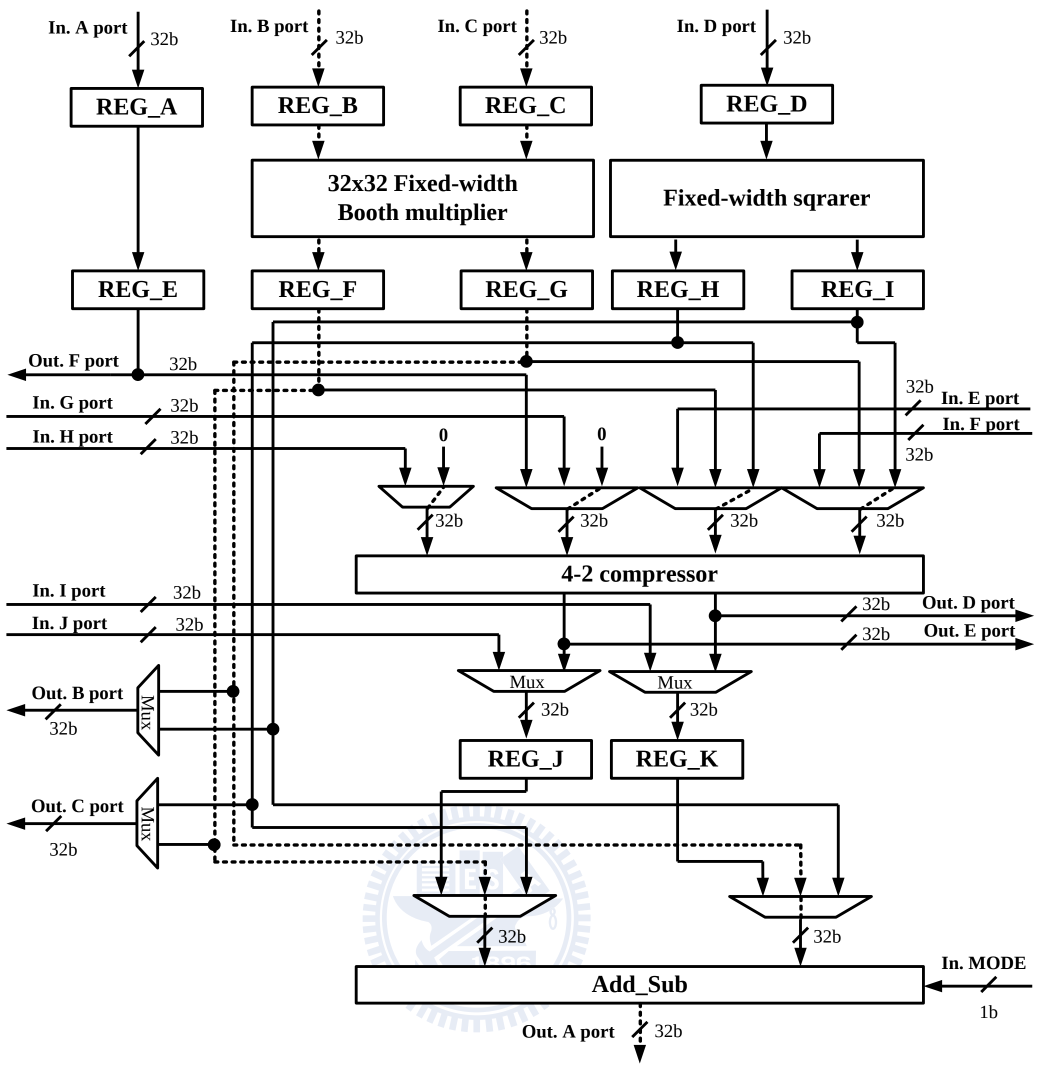


Fig. 3.7. Illustration of multiplication operation.

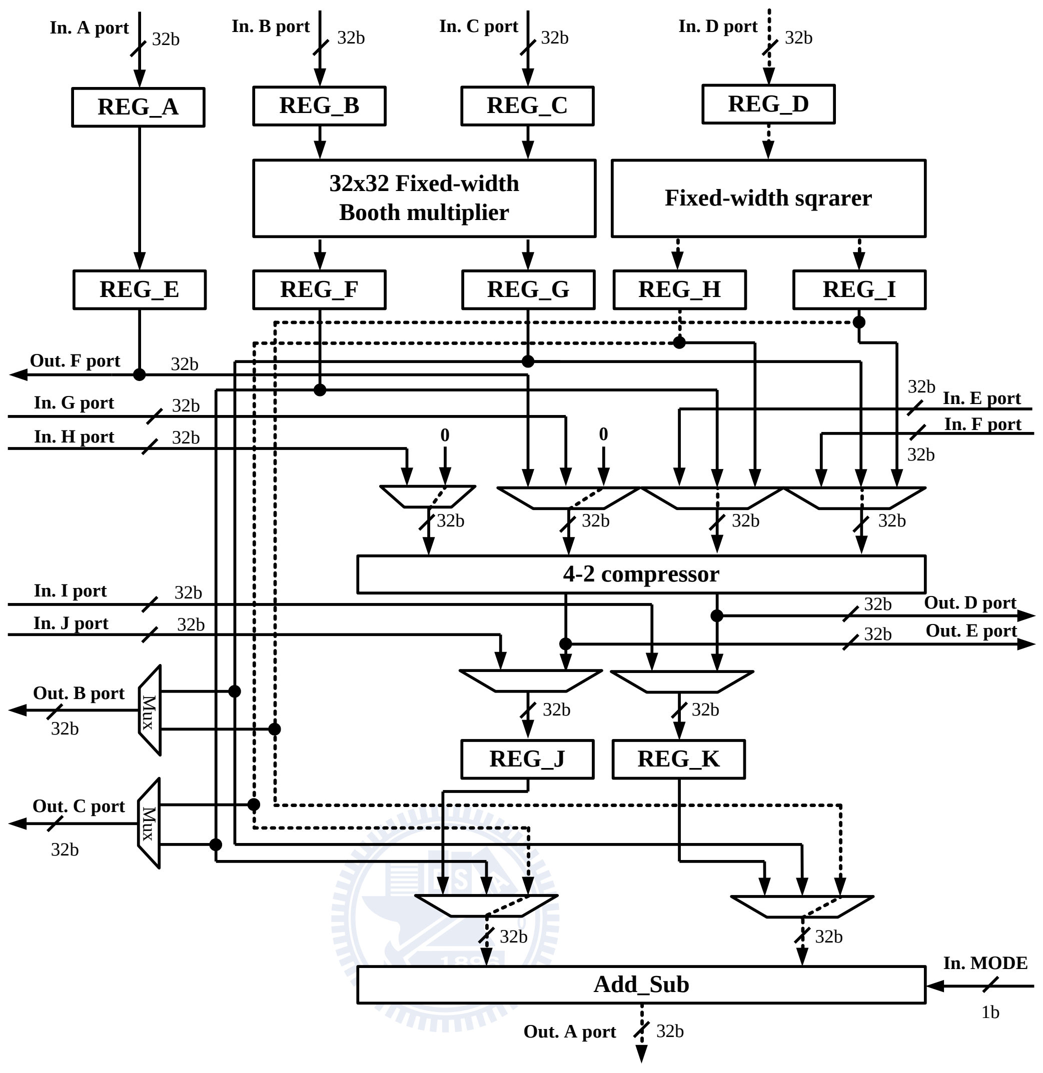


Fig. 3.8. Illustration of square operation.

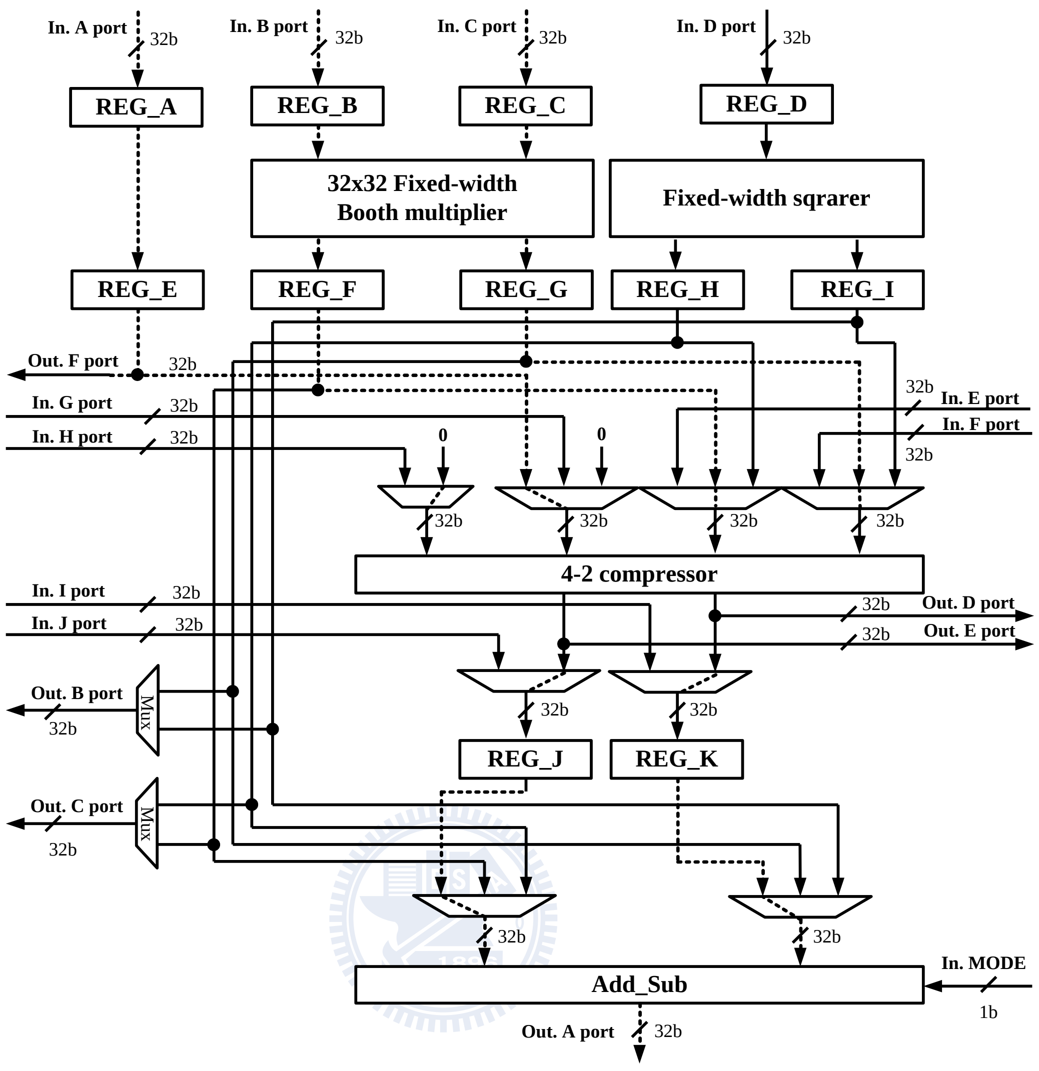


Fig. 3.9. Illustration of MAC operation.

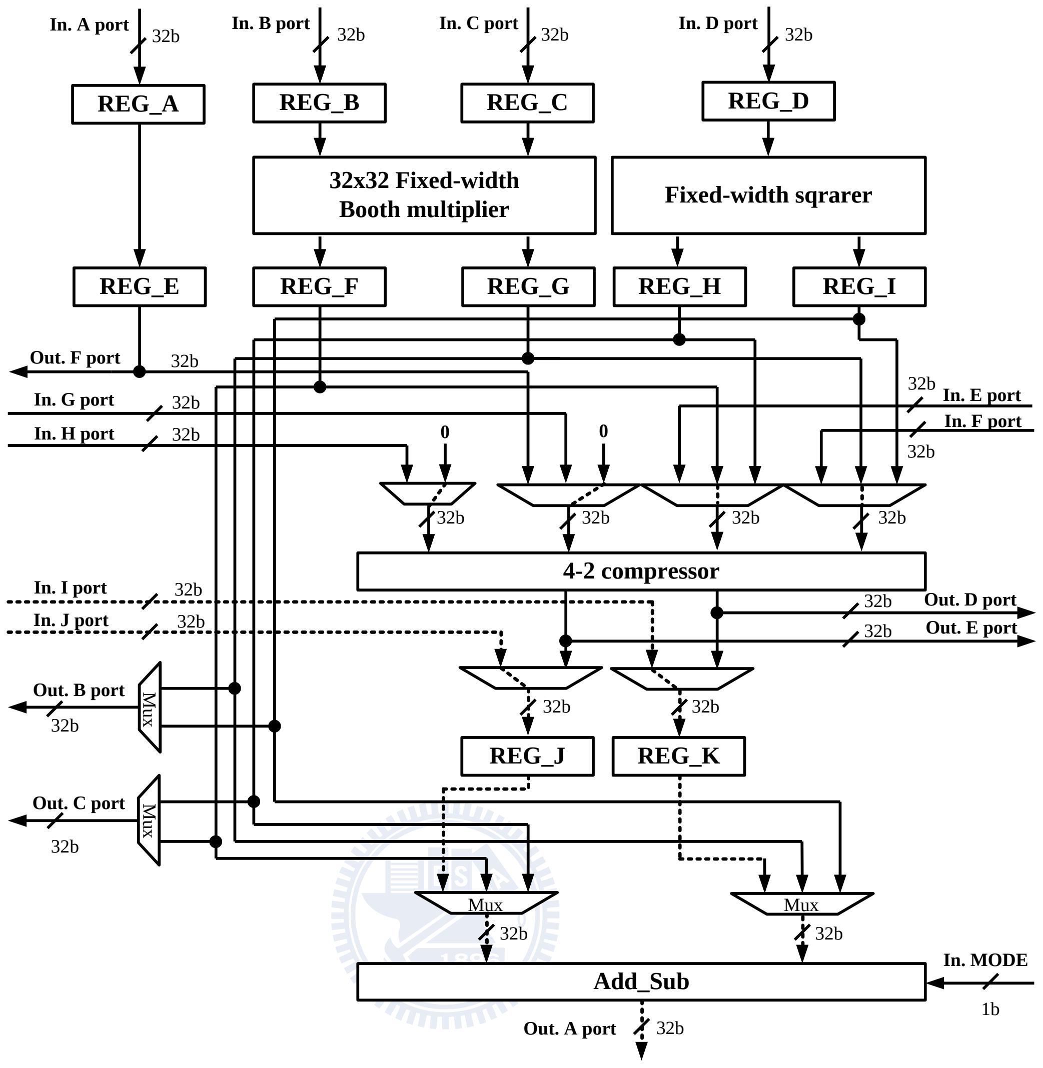


Fig. 3.10. Illustration of addition/subtraction operation.

3.6.2 Special Function Unit (SFU)

The special function unit (SFU) provides various arithmetic operations including the inverse (INV), inverse-square-root (InvSqrt) and power (POW) operations. These operations are used for vertex processing. To achieve low-power arithmetic operations, the SFU adopts the logarithmic number system (LNS) [22-24] where the complicated arithmetic operations are replaced by the simple arithmetic.

The architecture of SFU is depicted in Fig. 3.11, where the bit width of each node has been marked for clear representation. For the INV and the InvSqrt operations, the logarithmic convertor as shown in the top gray region of Fig. 3.11 converts the input number *m* to its logarithmic number *M*. Then, the number *M* is inversed through the Inv block to produce the result ~*M*. In the shift block, the number ~*M* is shifted right one bit to obtain (~*M*)>>1 for InvSqrt operation or directly bypassed for Inv operation. The behavior of the shift block controlled by the Config[1] port. The output logarithmic number (~*M*)>>1 or ~*M* of the shift block is then converted to their ordinary fixed-point number $\frac{1}{m}$ or $\frac{1}{\sqrt{m}}$ by the antilogarithmic convertor as shown in the bottom gray region of Fig. 3.11.

For the POW operation m^n, the number *m* is converted to its logarithmic number *M*. To compute nM, the multiplier is required for multiplication. However, the real multiplier is not included in SFU to achieve area and power-efficient feature. Because the processing element (PE) in the RDP shown in Fig. 3.6 can be configured to be a multiplier to compute nM. In Fig. 3.11, the logarithmic number *M* is outputted to a PE which is configured as a multiplier and multiplies to the number *n*. The result nM is then returned from the PE and converted to its ordinary number m^n. The Config[2] controls the source for the antilogarithmic convertor.

3.6.3 FIFO

As mentioned above, the RDP constructs an efficient pipeline datapath for complicated operations. In some configuration modes, some of the input data are used in the later stage of the pipeline. However, bypassing these data with pipeline registers stage by stage is not efficient for power consumption. To avoid the unnecessary data transfers between the pipeline registers, the FIFO is included in the RDP.

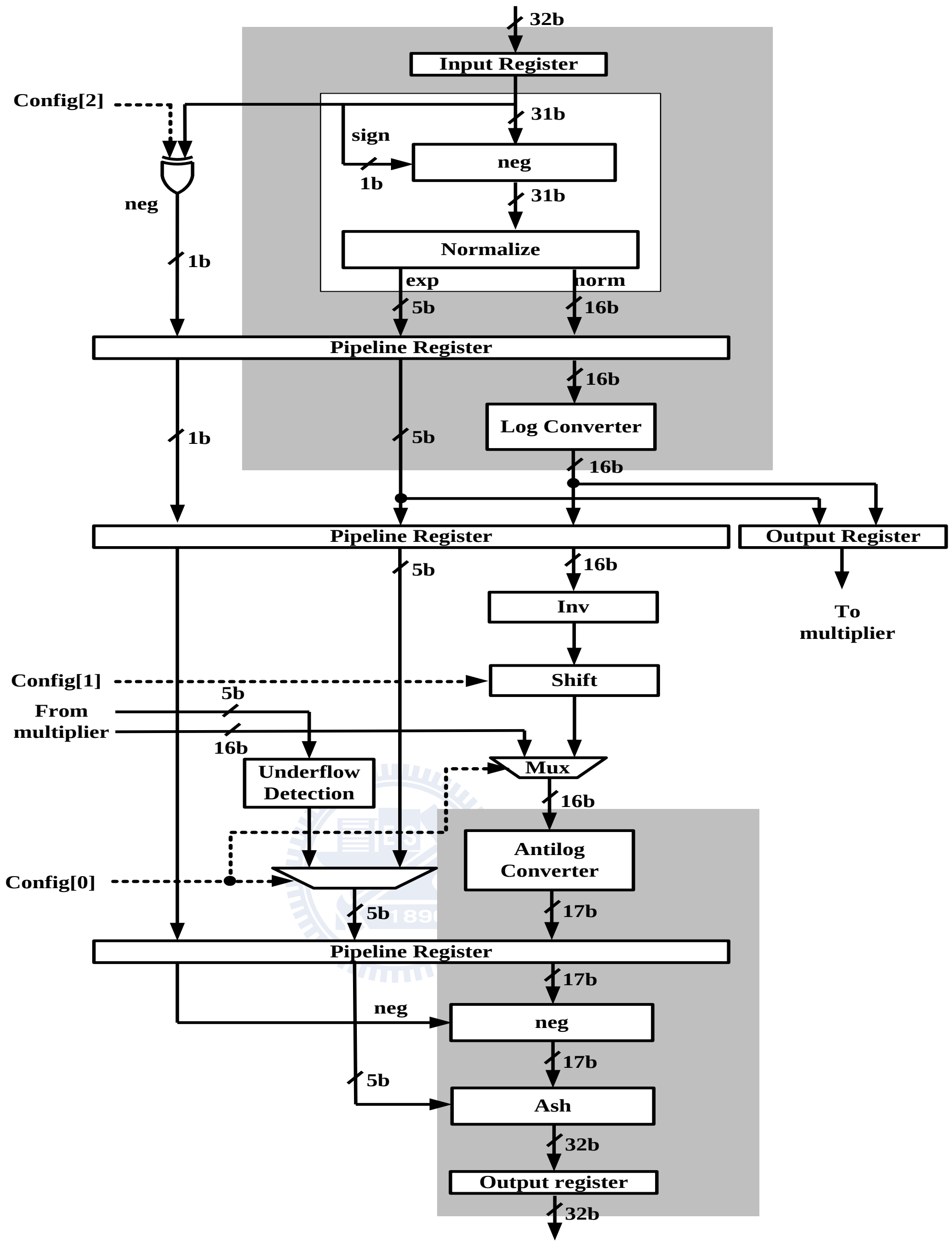


Fig. 3.11. Block diagram of the special function unit.

3.6.4 Interconnection of Configuration Modes

In this section, the interconnections between the building blocks for different configuration modes are described. For clearly explanation, the block diagram of the processing element (PE) is simplified. The geometry transforms in Eq. (2.13) and Eq. (2.14) multiply a 4x4 matrix by a 4x1 column vector. The matrix-vector multiplication can be replaced by four 4-component inner product operations. In Eq. (3.2), the 4-component inner product calculation employs four multiplications and therefore requires four multipliers.

$$\begin{bmatrix} x_1 & y_1 & z_1 & w_1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ w_2 \end{bmatrix} = x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2 + w_1 \cdot w_2 \tag{3.2}$$

Because the term w_{obj} in the column vector in Eq. (2.14) is always one and the projection matrix in Eq. (2.15) is a sparse matrix, these transforms can be achieved by the 3-component inner products and the additions as expressed in Eq. (3.3). The datapath for the operation in Eq. (3.3) is composed of three processing elements (PE) and the interconnections between PEs are illustrated in Fig. 3.12. At the first stage, the three multiplications are performed using the partial-product multiplier in the PEs, respectively. The addend w is directly passed to the next stage. At the second stage, the partial products and the addend w are compressed by the 4-2 compressor. Finally, the resulting partial products are summed up in the adder-subtractor unit of the central PE to produce the result.

$$\begin{bmatrix} x_1 & y_1 & z_1 & w_1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2 + w_1 \tag{3.3}$$

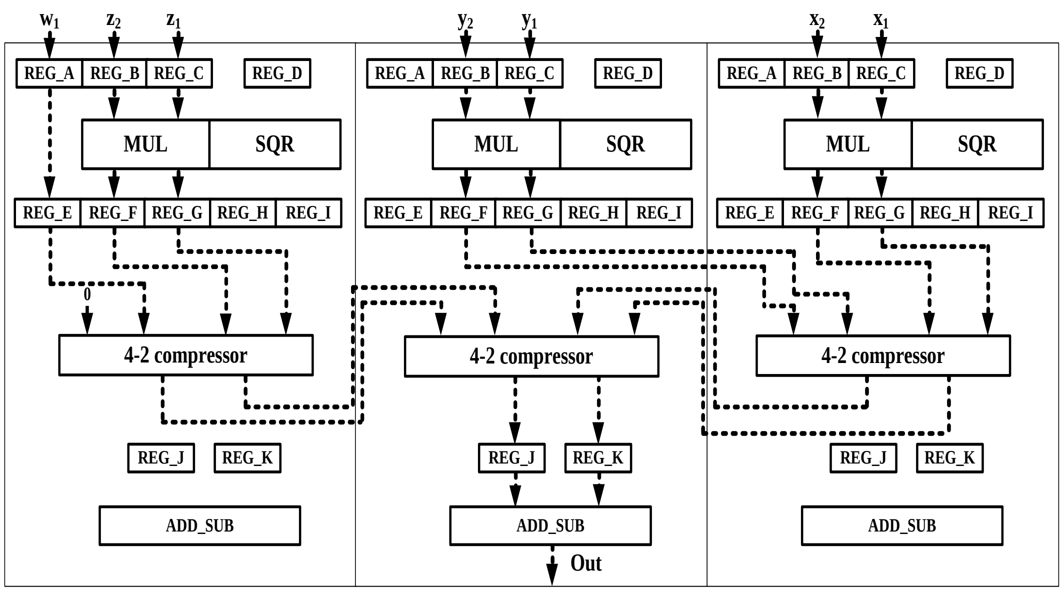


Fig. 3.12. Interconnection of the transform dp configuration mode.

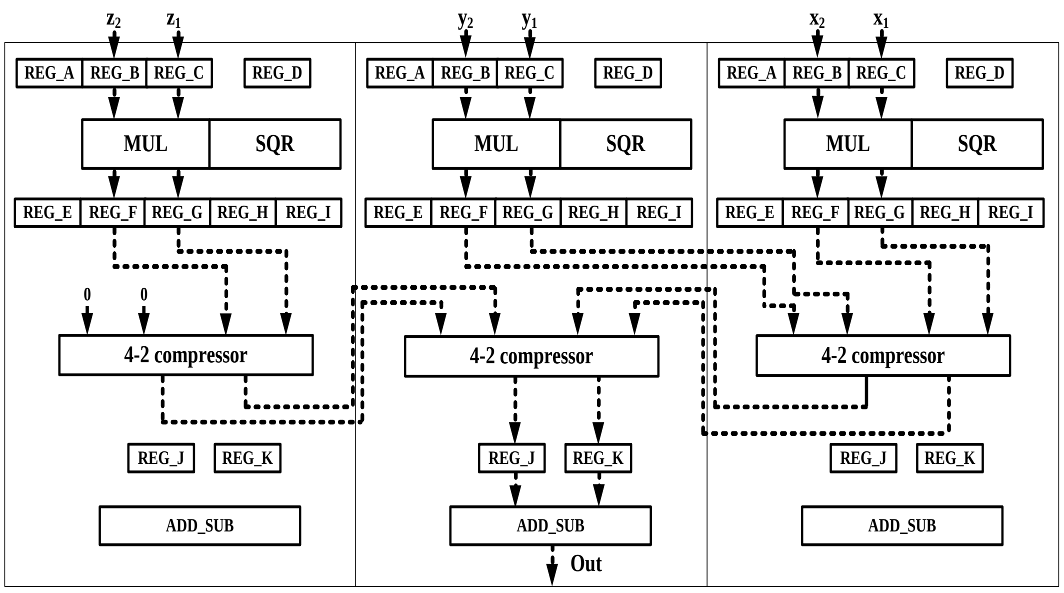


Fig. 3.13. Interconnection of the light dp configuration mode.

The light dp is a general 3-component inner product operation and is used for lighting calculations, for instance normal vector transform, dot product of two vectors. The datpath illustrated in Fig.3.13 is similar to the datapath of transform dp as illustrated in Fig. 3.12 but only the partial products from the multiplier are summed up at the second stage. The unused inputs of the 4-2 compressor in the left PE are forced to be zero.

In the lighting equation, the normal vector, light vector and halfway vector are required to have unit length before computing inner products. The equation of vector normalization is expressed in the Eq. (3.4). The RDP can be configured to accelerate the normalization operation.

$$norm([x_1, y_1, z_1]) = [\frac{x_1}{Length}, \frac{y_1}{Length}, \frac{z_1}{Length}]$$
$$\text{where } Length = \sqrt{x_1^{\ 2} + y_1^{\ 2} + z_1^{\ 2}} \tag{3.4}$$

As illustrated in Fig. 3.14, the solid-line datapath evaluates the length of input vector. The square operations are performed in the dedicated squarer in the PEs and the output partial products are added with the 4-2 compressor. Because all add-subtractors in the PEs are occupied by the dashed line datapath, additional adder is included to sum up the two outputs of the compressor in the central PE. The produced length value is passed to the SFU to evaluate its reciprocal value. Then, the input vector is multiplied by the inverse of the length value to obtain the normalized vector. The datapath of the scale-vector multiplication is the dashed-line datapath illustrated in Fig. 3.14.

In the perspective division in Eq. (2.15), the clip space coordinate x_{clip}, y_{clip} and z_{clip} are divided by the term w_{clip}. The perspective division can be simplified by computing the inverse of the w_{clip} and then multiplying the clip coordinate x_{clip}, y_{clip} and z_{clip} to the $1/w_{clip}$. The datapath of perspective division is depicted in Fig. 3.15. At the first stage, the w component of the input vector is passed to the SFU to compute the inverse $1/w$. After obtaining $1/w$, the x, y and z components of the input vector are multiplied by $1/w$. The multiplications can be achieved by configuring the processing elements to perform multiplication MUL operation.

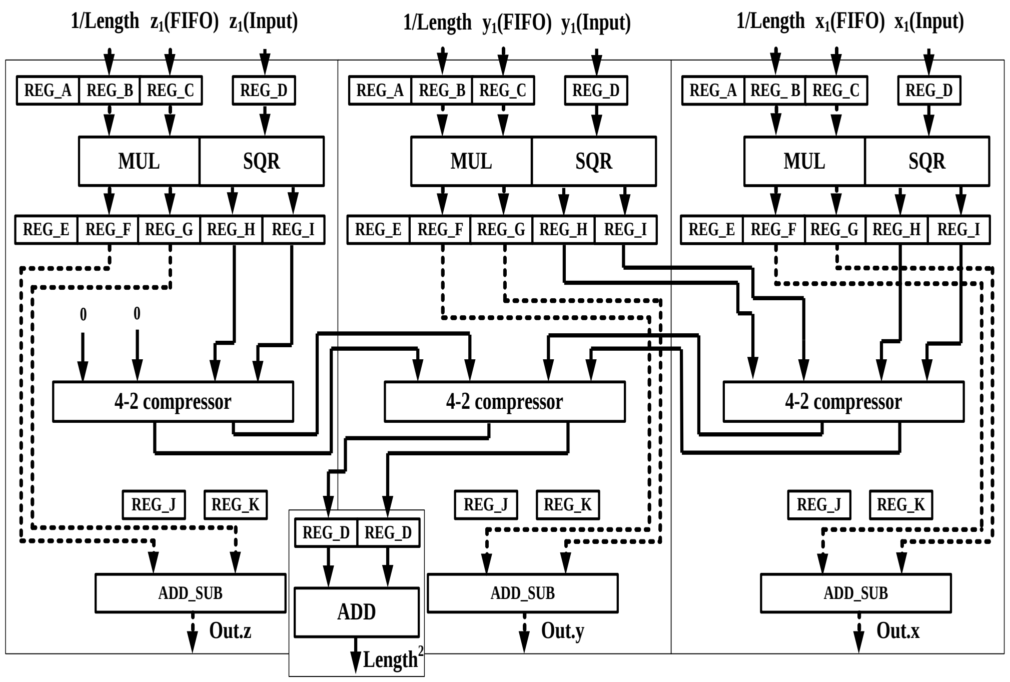


Fig. 3.14. Interconnection of the vector normalization configuration mode.

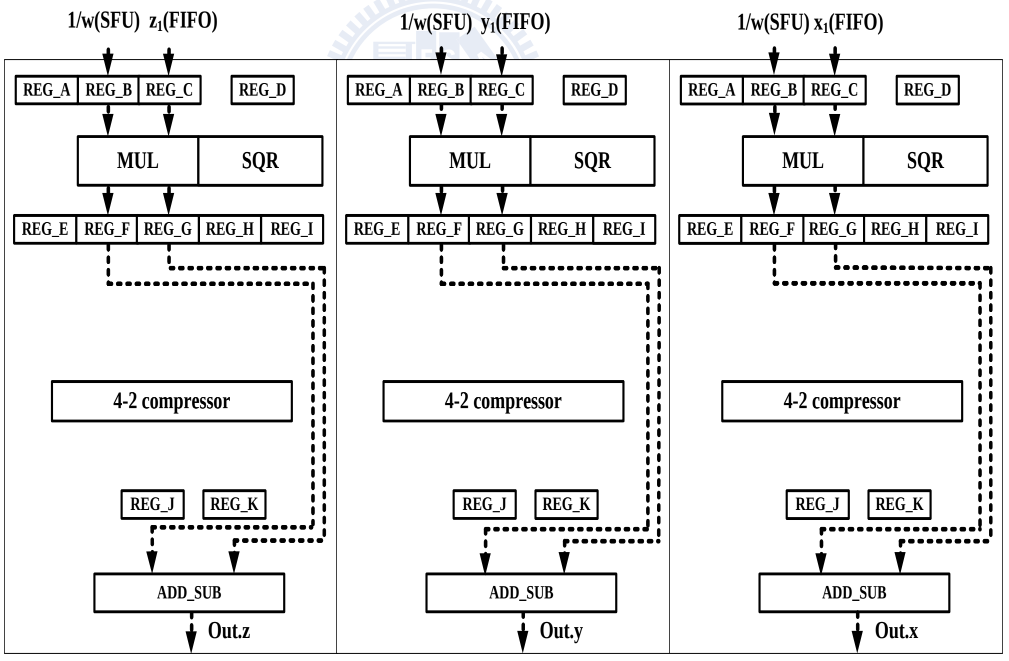


Fig. 3.15. Interconnection of the perspective division configuration mode.

The vector subtraction is used to compute the vector of two points in the 3D space, for example the light direction vector. The equation of vector subtraction is expressed in Eq. (3.4). The datapath is illustrated in Fig. 3.16. The REG_J and REG_K in each PE are configured to be the input registers and the add-subtractor unit is configured to perform subtraction operation.

$$[x_1, y_1, z_1]-[x_2, y_2, z_2]=[x_1 - x_2, y_1 - y_2, z_1 - z_2] \qquad (3.5)$$

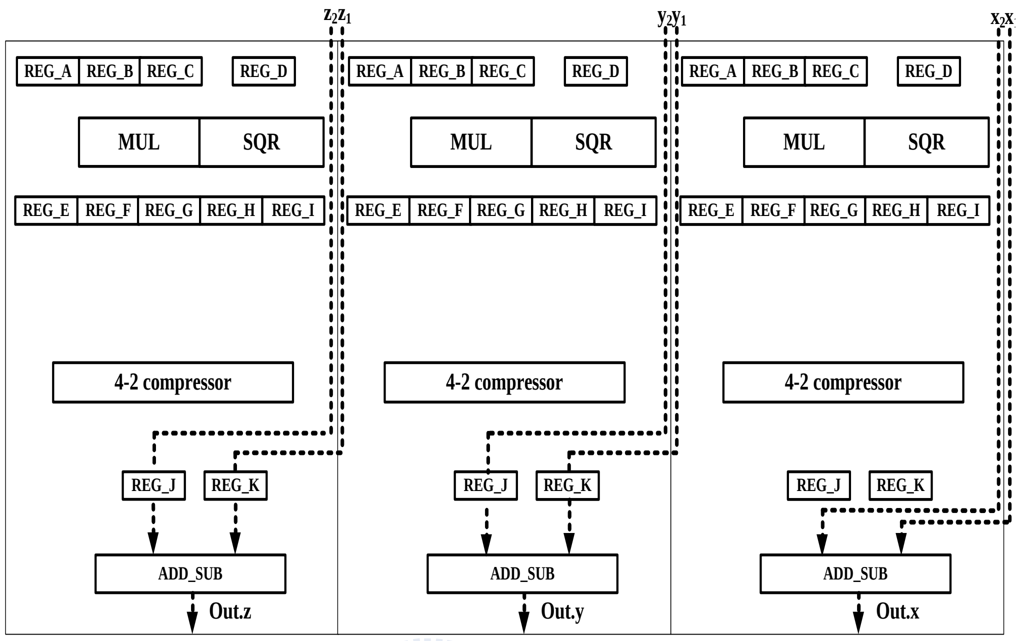


Fig. 3.16. Interconnection of the vector subtraction configuration mode.

Chapter 4

Comparison Results and Chip Implementation

In this chapter, the comprehensive comparison results in terms of complexity for subdivision algorithm and power-efficient index among different state-of-the-art chips for geometry engines are addressed.

4.1 Complexity Comparison Results

The complexity comparison to the conventional subdivision algorithm is listed in Table. 4.1 in terms of number of memory accesses, computation for edge functions, computation for transforms, number of clipping/culling test operations, and number of 3x3 matrix multiplications of setup operation for rasterizaiton. In Table. 4.1, N_T is defined the number of triangles in the scene and N_A denotes the number of vertex attributes of each vertex. For level-1 case and level-2 case, the quantitative comparison is listed in Tables 4.2 and 4.3. The reduction of the number of memory accesses can be attained by 44.44% and 68.88% for level-1 and level-2, respectively. In terms of multiplications for the edge function calculation, the computation can be alleviated by 0% and 50% for level-1 and level-2, respectively. The reduction of the number of multiplications for transforms can be attained by 50% and 80% for level-1 and level-2, respectively. In terms of clipping/culling test operations, the computation can be alleviated by 75% and 93.75% for level-1 and level-2, respectively. The reduction of the number of 3x3 matrix multiplications of setup operation for rasterizaiton can be attained by 40% and 60% for level-1 and level-2, respectively.

Table 4.1: Complexity comparison results in general representation between conventional subdivision algorithm and proposed subdivision algorithm.

		Conventional subdivision algorithm	**Proposed subdivision algorithm**	**Used schemes**
Number of memory accesses			$(N_G+2)N_T$	Forward difference
Computation for edge function	Muls	$6N_SN_T$	$12N_T$	Edge recovery
	Adds	$9N_SN_T$	$(3N_S+9)N_T$	
Computation for transforms	Muls	$(22N_G+66)N_T$	$66N_T$	Dual space subdivision
	Adds	$(21N_G+57)N_T$	$(10N_G+65)N_T$	
	Invs	$(N_G+3)\ N_T$	$3N_T$	
Number of clipping/culling test operations		$N_S^2N_T$	$1N_T$	Triangle filtering
Number of 3x3 matrix multiplications of setup operation for rasterization		$N_AN_S^2N_T$	$(\left\lceil (\frac{1}{3})N_AN_S^2 \right\rceil +N_A)N_T$	Setup variable sharing

Table 4.2: Complexity comparison results for level-1 case ($N_S = 2$, $N_G = 3$, $N_A = 5$).

		Conventional subdivision algorithm	**Proposed subdivision algorithm**	**Complexity reduction percentage**
Number of memory accesses		$9N_T$	$5N_T$	44.44%
Computation for edge function	Muls	$12N_T$	$12N_T$	0%
	Adds	$18N_T$	$15N_T$	16.66%
Computation for transforms	Muls	$132N_T$	$66N_T$	50%
	Adds	$120N_T$	$95N_T$	20.83%
	Invs	$6N_T$	$3N_T$	50%
Number of clipping/culling test operations		$4N_T$	$1N_T$	75.00%
Number of 3x3 matrix multiplications of setup operation for rasterization		$20N_T$	$12N_T$	40.00%

Table 4.3: Complexity comparison for level-2 case ($N_S = 4$, $N_G = 12$, $N_A = 5$).

		Conventional subdivision algorithm	**Proposed subdivision algorithm**	**Complexity reduction percentage**
Number of memory accesses		$45N_T$	$14N_T$	68.88%
Computation for edge function	Muls	$24N_T$	$12N_T$	50.00%
	Adds	$36N_T$	$21N_T$	41.66%
Computation for transforms	Muls	$330N_T$	$66N_T$	80.00%
	Adds	$309N_T$	$185N_T$	40.12%
	Invs	$15N_T$	$3N_T$	80.00%
Number of clipping/culling test operations		$16N_T$	$1N_T$	93.75%
Number of 3x3 matrix multiplications of setup operation for rasterization		$80N_T$	$32N_T$	60.00%

4.2 Chip Implementation and Comparison Results

Concerning the chip implementation of the proposed GE architecture, the cell-based design flow with Faraday standard cell library in UMC 90-nm CMOS process is adopted. The Synopsys Design-Compiler is used to synthesize the RTL design of the proposed architecture and the Cadence SOC-Encounter is adopted for automatic placement and routing (APR) and the Synopsys Prime-Power is used to measure the power consumption for the post-layout simulation.

Table 4.4 summarizes the chip characteristics of the proposed GE architecture and the corresponding chip layout is shown in Fig. 4.1.

Table 4.4: Chip characteristics of the proposed GE architecture

Power Supply	1.0V
Process Technology	UMC 90 nm CMOS
Max. Clock	200 MHz
Max. Power	5.89 mW
Gate Count	195K
Core Area	0.58 mm^2

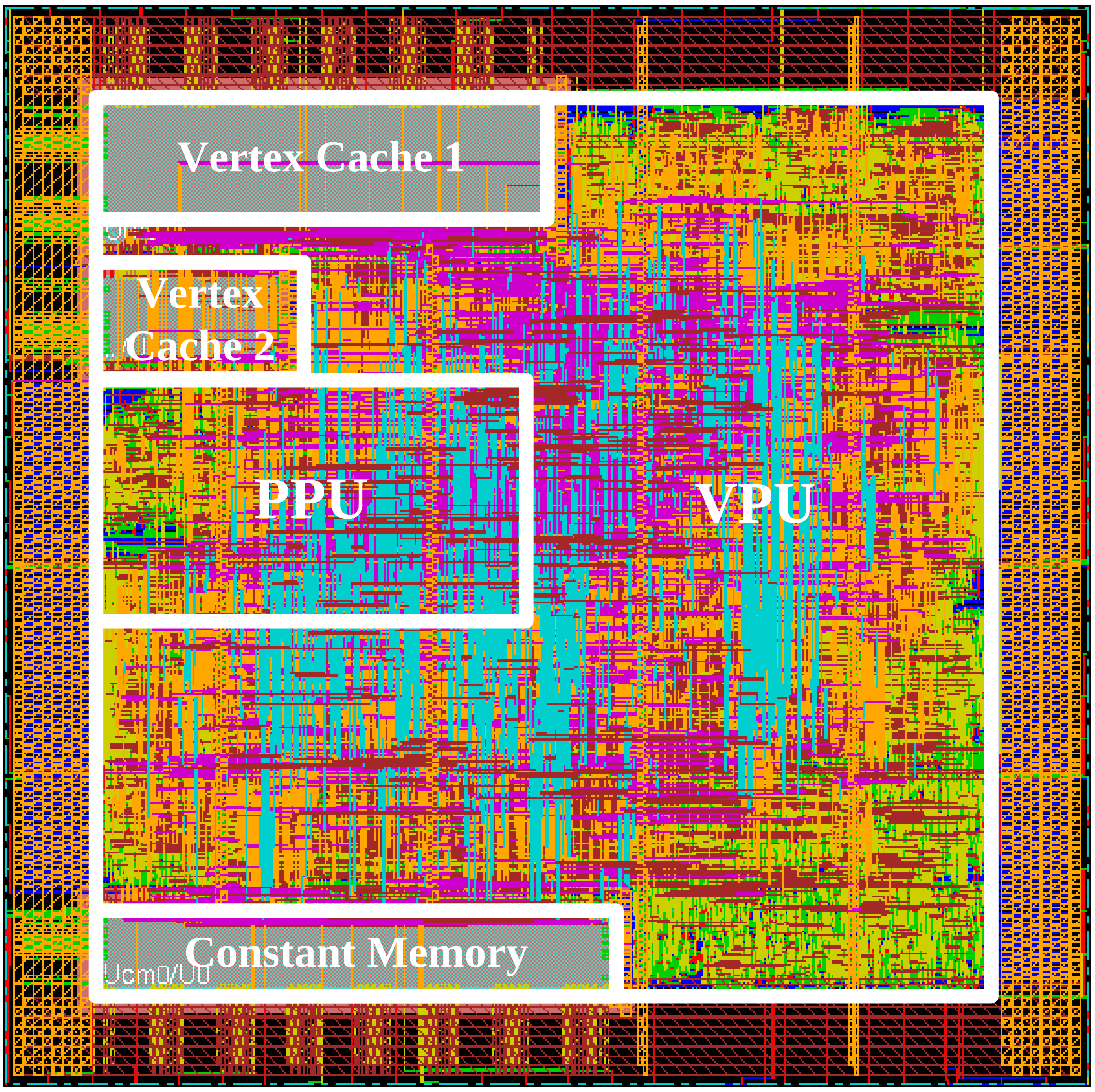
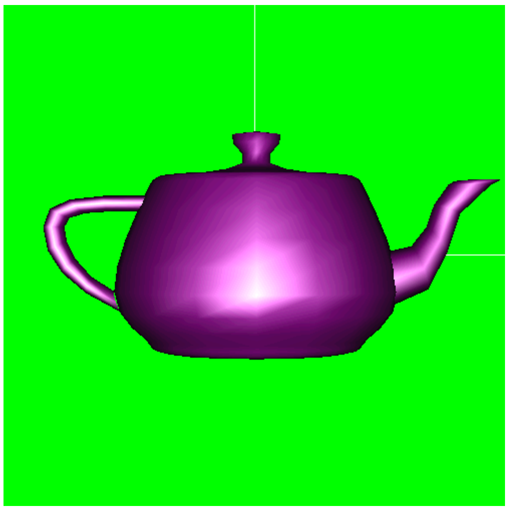
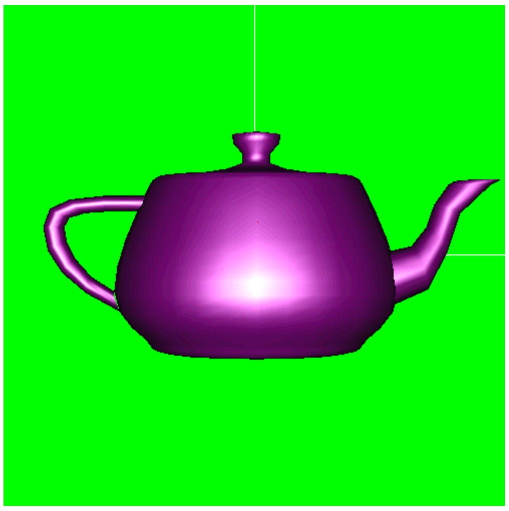


Fig. 4.1. Chip layout of the GE.

The same teapot benchmark is rendered with different subdivision levels including level-0, level-1, and level-2 as shown in Figs. 4.3 (a), (b) and (c), respectively. The power consumption for each subdivision level are measured and illustrated in Fig. 4.4.


(a) level-0


(b) level-1

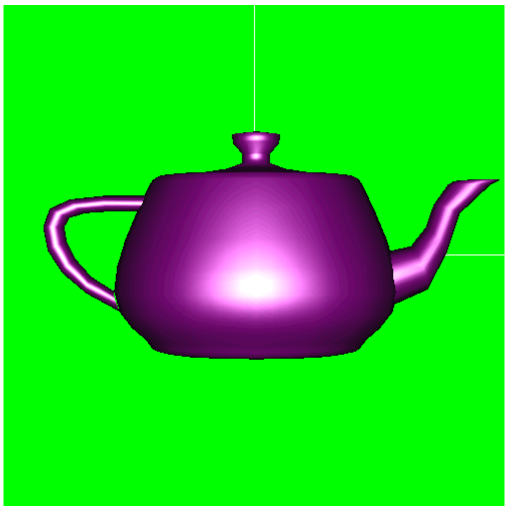

(c) level-2

Fig. 4.2: Rendering result of different subdivision levels.

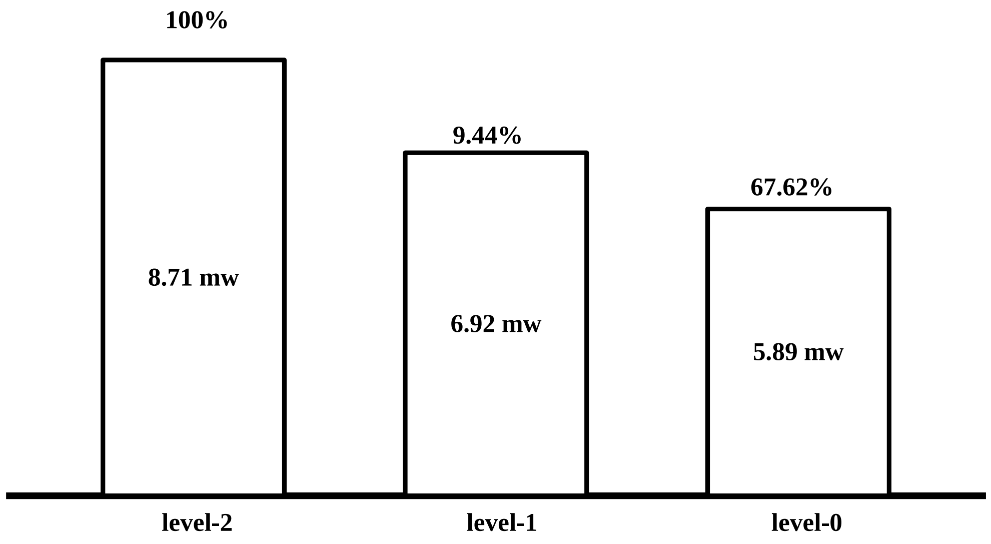


Fig. 4.3: Power profiling of different subdivision levels.

The comparison results between prior work and our work are summarized in Table 4.5. Compared with [25][26][27][28][29], the proposed GE has better power efficient index with 16.978 Mvertices/(s•mW). Moreover, using the proposed subdivision algorithm, the proposed GE can provide near-Phong shading quality.

Table 4.5: Comparison results among the existing work

	ISSCC'04 [25]	JSSC'06 [26]	JSSC'07 [27]	ISSCC'07 [28]	JSSC'08 [29]	This Work
Process (nm)	130	180	180	180	180	90
Frequency (MHz)	400	200	100	200	50	200
Polygon Rate (Mvertices/s)	36	50	120	141	25*1/12.5	100*1/50
Power (mW)	250	155	157	86	8.6	5.89
Core Area (mm^2)	-	23	16	9.7	6.05	0.58
Power Efficiency Mvertices/(s•mW)	0.144	0.323	0.764	1.64	2.907	16.978
Feature	Graphics, DSP	Graphics	Graphics	Graphics	Graphics, DSP	Graphics with scalable quality hardware support.

*1: Assume hit rate is 50%.

*2: The core area is 2.164mmx2.797mm

Chapter 5

Conclusion

In this work, a low complexity subdivision algorithm and a power efficient GE are presented. Five low complexity techniques including the triangle filtering scheme, the dual space subdivision, the setup variable sharing and the edge function recover scheme are proposed to reduce the computational complexity of the subdivision algorithm. The proposed geometry engine employs several techniques to optimize the power, area and shading quality. With the post-TnL vertex cache and the object space culling scheme, the redundant computation for transforming and lighting can be eliminated. With the proposed RDP, the area is reduced since the same set of PEs can be reconfigured for different mode operations. The dedicated hardware supports the scalable and near-Phong shading quality. Three different subdivision levels including level-0, level-1 and level-2 are supported. From the chip implementation results, the proposed geometry engine can achieve the power-efficiency of 16.978 Mvertices/mW.

Bibliography

[1] P. Cesar, P. Vuorimaa, and J. Vierinen, “A graphics architecture for high-end interactive television terminals,” *ACM Trans. Multimedia Comput. Commun. and Appil.*, vol. 2, no. 4, pp.343-357, Nov. 2006.

[2] B.-S. Liang, Y.-C. Lee, W.-C. Yeh, C.-W. Jen, "Index rendering: hardware-efficient architecture for 3-D graphics in multimedia system," *IEEE Trans. Multimedia*, vol.4, no.3, pp. 343-360, Sep. 2002.

[3] H. Gouraud, “Continuous shading of curved surfaces,” *IEEE Trans. Compt.*, pp.623-628, June 1971.

[4] A. Watt, “3D computer graphics,” 3rd Edition, *Addison Wesley,* 2000.

[5] A.T. Phong, “Illumination for computer generated pictures,” *Communications of the ACM*, vol. 18, no. 6, pp.311-317, June 1975.

[6] G. Bishop, and D. M. Weimer, “Fast Phong Shading,” *Proc. Computer Graphics and interactive Technique*, 1986, pp.103-106.

[7] A. A. Mohamed, L. S. Kalos, and T. Horváth, “Hardware implementation of Phong shading using spherical interpolation,” *Periodica Polytechnica*, vol. 44, Nos 3-4, 2000.

[8] T. Barrera, A. Hast, and E. Bengtsson, “Faster shading by equal angle interpolation of vectors,” *IEEE Trans. Visualization and Computer Graphics,* pp.217-223, Mar. 2004.

[9] K. Harrison, D. A. P. Mitchell, and A. H. Watt., “The H-test: a method of high speed interpolative shading,” *Proc. New Trends in CG.*, 1988, pp.106-166.

[10] J. Pöpsel, and Ch. Homung, “Highlight shading lighting and shading in a PHIGS+PEX environment,” *EUROGRAPHICS*, 1989, pp.317-332.

[11] A. A. Mohamed, L. S. Kalos, G. Szijártó, T. Horváth, and T. Fóris, “Quadratic interpolation in hardware Phong shading and texture mapping,” *SCCG’01*, April, 2001, pp.181-188.

[12] T. Barrera, A. Hast, and E. Bengtsson, “Fast near Phong-quality software shading,” *WSCG’06*, January, 2006, pp.109-115.

[13] S. Bischoff, L.P. Kobbelt, and H.P. Seidel, “Toward hardware implementation of Loop subdivision,” *Proc. SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, 2000, pp.41-50.

[14] Y. Cho, U. Neumann, and J. Woo, “Improved specular highlights with adaptive shading,” *Proc. of CG. International*, June, 1996, pp.38-46

[15] Y. Kamen, and L. Shirman, “Triangle rendering using adaptive subdivision,” *IEEE Comput. Graph. Applal.*, Mar. 1998.

[16] T. Y. Sheu, L. D. Van, T. R. Jung, C. W. Lin, and T. W. Chang, "Low complexity subdivision algorithm to approximate Phong shading using forward difference," *ISCAS 2009*, pp. 2373-2376.

[17] J. McCormack and R. McNamara, “Tiled polygon traversal using half-plane edge functions,” *Proc. SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, 2000, pp.15-21.

[18] M. Olano, and T. Greer, “Triangle scan conversion using 2D homogeneous coordinates,” *Proc. SIGGRAPH/EUROGRAPHICS workshop on Graphics Hardware*, August, 1997, pp.89-95.

[19] K.-C., C.-H. Yu and L.-S. Kim, "Vertex cache of programmable geometry processor for mobile multimedia application," *ISCAS 2006.*

[20] C.-Y. Han, Y.-H. Im and L.-S. Kim, "Geometry engine architecture with early

backface culling hardware," *Computers & Graphics*, pp.415-425, 2005.

[21] Antonio G.M. Strollo and Davide De Caro, "Booth Folding Encoding for High Performance Squarer Circuits," *IEEE Trans. CAS II: Analog and Digital Signal Processing*, vol.50, no.5, pp.250-254, May 2003.

[22] K. H. Abed and R. E. Siferd, “CMOS VLSI implementation of a low-power logarithmic converter,” *IEEE Trans. Computers*, vol. 52, no. 11, pp. 1421-1433, Nov. 2003.

[23] K. H. Abed and R. E. Siferd, “CMOS VLSI implementation of a low-power antilogarithmic converter,” *IEEE Trans. Computers*, vol. 52, no. 9, pp. 1221-1228, Nov. 2003.

[24] B.-G. Nam, H.-Kim and H.-J. Yoo, “A low-power unified arithmetic unit for programmable handheld 3-D Graphics Systems,” *IEEE J. Solid-State Circuits*, vol. 42, no. 8, Aug. 2007.

[25] F. Arakawa et al., “An embedded processor core for consumer applications with 2.8 GFLOPS and 36 Mpolygons/s FPU,” *IEEE ISSCC*, Feb. 2004, pp. 334–335.

[26] J. Sohn et al., “A 155-mW 50-Mvertices/s graphics processor with fixed-point programmable vertex shader for mobile applications,” *IEEE J. Solid-State Circuits*, vol. 41, no. 5, pp. 1081–1091, May 2006.

[27] C. H. Yu, K. Chung, D. Kim and L.-S. Kim, "An energy-efficient mobil vertex processor with multithread expanded VLIW architecture and vertex caches," *IEEE J. Solid-State Circuits*, vol. 42, no. 10, Oct. 2007.

[28] B.-G. Nam, J. Lee, K. Kim, S.-J. Lee, and H.-J. Yoo, “A 52.4 mW 3-D graphics processor with 141 Mvertices/s vertex shader and 3 power domains of dynamic voltage and frequency scaling,” *ISSCC 2007*, pp. 278-603.

[29] S.-Y. Chien, Y.-M. Tsao, C.-H. Chang and Y.-C. Lin, “An 8.6 mW 25

Mvertices/s 400-MFLOPS 800-MOPS 8.91 mm^2 multimedia stream processor core for mobile applications,“ *IEEE J. Solid-State Circuit*, vol. 43, issue. 9, pp. 2025-2035, Sept. 2008.

Publication List

International Conference Papers

[1] T. Y. Sheu, L. D. Van, T. R. Jung, C. W. Lin, and T. W. Chang, ”Low complexity subdivision algorithm to approximate Phong shading using forward difference,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May. 2009, pp. 2373-2376, Taipei, Taiwan.

[2] T. R. Jung, L. D. Van, T. Y. Sheu, C. W. Lin, W. C. Fang, “Design of multi-mode depth buffer compression for 3D graphics system,” in *Proc. IEEE Int. Conf. Multimedia and Expo. (ICME)*, July 2008, pp. 789-792, Hannover, Germany.

[3] T. R. Jung, L. D. Van, W. C. Fang, T. Y. Sheu, "Reconfigurable depth buffer compression design for 3D graphics system," in *Proc. Int. Conf. MUE.*, Apr. 2008, pp. 470-474, Busan, Korea.

Biography

Ten-Yao Sheu was born in Changhua, Taiwan, R.O.C, in 1983. He received the B.S. degree from National Pingtung University of Education (NPUE), Pingtung, Taiwan, in 2006, and the M.S degree from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 2009, all in computer science. His research interests are VLSI information processing algorithm and architecture for 3D graphics.