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

Lan-Da Van, *Member, IEEE*, and Ten-Yao Sheu

***Abstract*—In this paper, a power-area 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 complexity, high power-area efficiency, scalable and near-Phong shading quality. The forward difference, edge function recovery, dual space subdivision, triangle filtering, and triangle setup coefficient sharing schemes are employed to alleviate the redundant computation for the proposed algorithm. According to the low-complexity subdivision algorithm, one reconfigurable datapath is proposed to save the area since the same set of processing elements (PE) is reused for different operations of GE. Compared with the conventional subdivision algorithm, the proposed subdivision algorithm reduces the number of memory/register accesses for subdivision by 40% and 60.32% for level-1 and level-2 subdivision, respectively. In terms of the number of multiplications for transforms, the reduction can be attained by 27.5% and 60.27% for level-1 and level-2 subdivision, respectively. From the implementation results, the proposed GE can achieve the power-area efficiency of 518.8 $\mathrm{Kvertices/(s\bullet mW\bullet mm^2)}$ for level-1 subdivision.**

***Index Terms*—Forward difference, geometry engine, near-Phong shading, power-area efficient, subdivision.**

I. Introduction

NOWADAYS, 3-D graphics functions are integrated into the wireless and wired terminals such as mobile devices and next generation TV systems [1]. 3-D 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, Phong shading needs to re-normalize the normal vector and computes the light intensity for every pixel inside the polygon. Thus, Phong shading consumes more power because of the huge computation requirement.

Manuscript received July 01, 2010; revised October 29, 2010 and January 08, 2011; accepted February 10, 2011. Date of publication April 05, 2011; date of current version September 14, 2011. This work was supported in part by the National Science Council Grants NSC-99-2220-E-009-029, NSC-99-2220-E-009-071, and NSC-98-2220-E-009-012. This paper was presented in part at 2009 ISCAS. This paper was recommended by Associate Editor A. Strollo.

The authors are with the Department of Computer Science, National Chiao Tung University, Hsinchu, 300, Taiwan, R.O.C. (e-mail: ldvan@cs.nctu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2011.2123430

Recently, low computation, power-area efficiency, and satisfactory quality are the important research issues for hardware design. In order to have near-Phong shading quality with low computation, several existing 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 multilight 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 each light source. Thus, the setup cost is expensive for thin polygons and the multilight source scenes. The mixed shading [9] combines two shading schemes. When the highlight covers the polygon, the polygon is rendered with Phong shading. Otherwise, Gouraud shading is employed. Although the 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 [10], [11] uses a quadratic function to interpolate light intensities between six points. The quadratic scheme would incur Mach band effect on the edge when the triangle is too large. Therefore, an error control scheme is proposed in [10] to solve this problem. The subdivision scheme [12]–[15] is another approach to approximate Phong shading. It subdivides a triangle into smaller ones and renders them individually with Gouraud shading. Compared with other approximate per-pixel lighting schemes, only vertices are lit. One attractive feature of the subdivision algorithm is the ability to scale shading quality dynamically. If higher shading quality is demanded, more small triangles are generated. Although the conventional subdivision algorithm inherently provides scalable and near-Phong shading quality, the computational complexity and power consumption are still large for GE. On the other hand, the area-efficient VLSI architecture and implementation of the scalable subdivision algorithm has not been explored. Thus, we are motivated to propose a low-complexity subdivision algorithm and the corresponding power-area efficient and scalable-quality geometry engine in the paper. Note that, in this paper, we just focus on the triangle subdivision rather than the surface/mesh subdivision [16]–[18] such that the proposed design cannot change the surface/mesh geometry.

The rest of the paper is organized as follows. The proposed subdivision algorithm and the corresponding complexity analysis are described in Section II. In Section III, the proposed GE architecture is presented. The comparison results and chip layout are addressed in Section IV. Last, a brief statement concludes the presentation of this paper.

II. Proposed Low-Complexity Subdivision Algorithm

In this section, a low complexity subdivision algorithm to approximate Phong shading is proposed. To reduce the redundant

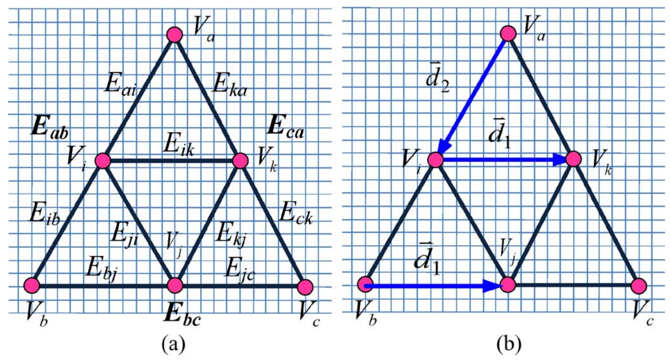


Fig. 2.1. Illustration for subdivision using forward difference.

memory/register accesses for subdivision, 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 order to reduce the complexity, the dual space subdivision scheme, the triangle filtering scheme and the triangle setup coefficient sharing scheme are also presented. The proposed algorithm and schemes are described in detail in the following subsections.

A. *Subdivision Using Forward Difference*

In computer graphics, the forward difference method [16], [17] has been widely used to evaluate the surface/mesh. Herein, we apply this scheme to the triangle subdivision to achieve the reduction of the memory/register access. Without loss of generality, we set $N_S = 2$ as shown in Fig. 2.1(a), where $N_S = 2^L$ denotes the number of the segments on each edge of the original triangle and L denotes a subdivision level number (i.e., level-L). In order to subdivide the triangle $\Delta V_a V_b V_c$, the generated vertices V_i, V_j, V_k are computed. Then these new vertices together with the original vertices will be packed and new triangles are generated as: $\Delta V_a V_i V_k$, $\Delta V_i V_j V_k$, $\Delta V_i V_b V_j$ and $\Delta V_k V_j V_c$. The forward difference method is used to compute the generated vertices. The first step is to compute the difference vectors $\vec{d}_1$ and $\vec{d}_2$ in horizontal and top-right to bottom-left direction using (2.1) and (2.2), respectively

$$\vec{d}_1 = (V_c - V_b)/N_S \tag{2.1}$$
$$\vec{d}_2 = (V_b - V_a)/N_S \tag{2.2}$$

where V_a, V_b, V_c have the corresponding coordinates in different spaces. Once the difference vectors are computed, the generated vertices can be obtained by (2.3), (2.4), and (2.5) in Fig. 2.1(b)

$$V_i = V_a + \vec{d}_2 \tag{2.3}$$
$$V_k = V_i + \vec{d}_1 \tag{2.4}$$
$$V_j = V_b + \vec{d}_1 \tag{2.5}$$

Computing the generated vertices using the forward difference method is more efficient than other methods because one generated vertex only needs one memory/register access to store the vertex. Compared with the conventional recursive-based subdivision algorithms [12]–[14], the forward difference method is stack free, and, hence, the number of memory/register accesses can be decreased. In other words, the power incurred by large number of memory/register data accesses can be alleviated. However, the subdivision algorithm using forward difference would result in the rasterization anomaly due to the numerical instability. As shown in Fig. 2.2(a), (b), (c), and (d), because the numerical instability incurs lost pixels on the rendered object, the empty areas on teapot, pawn, Venus, and couch are referred to as rasterization anomaly. In detail, as illustrated in Fig. 2.3, two adjacent triangles are subdivided using forward difference. In Fig. 2.3(a), V_m denotes one generated vertex on the sharing edge of two original triangles. It can be obtained by subdividing either the left triangle or the right triangle if the calculation has no error. In Fig. 2.3(a), the vertex V_q is the generated vertex in the left subdivided triangle and is computed from the vertex V_p using the difference vector $\vec{d}_1$ once. The vertex V_q has the same coordinate as the vertex V_m if the calculation has no error. However, the calculation has the quantization error. While the accumulated quantization error is large enough, the coordinate of the vertex V_q is different from that of the vertex V_m. For the same reason, in the right triangle of Fig. 2.3(a), the vertex V_i computed from the vertex V_a with the difference vector $\vec{d}_2$ has different coordinate from the vertex V_m. As a result, the small triangles defined by vertex V_q and V_i respectively are not adjacent to each other. In Fig. 2.3(b), since the pixels surrounded by the sharing edges are lost after rasterization, the rasterization anomaly occurs.

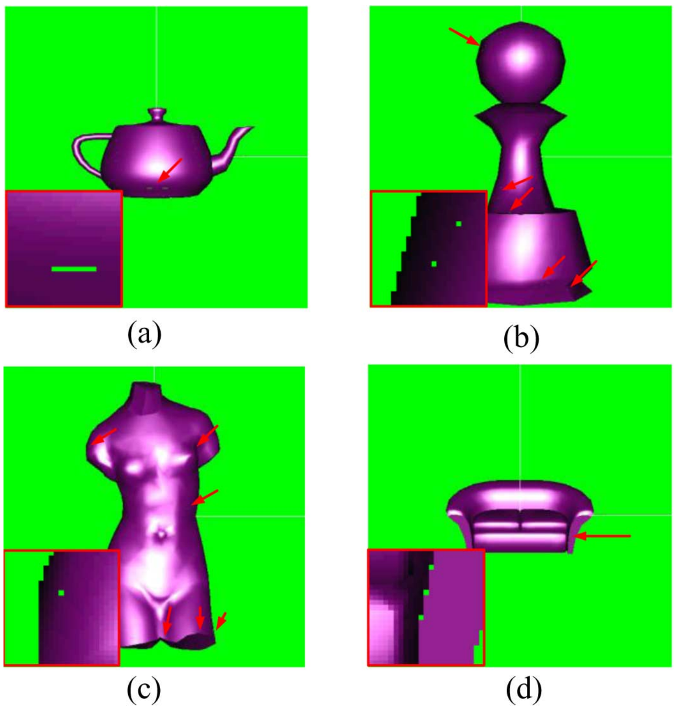


Fig. 2.2. Examples of rasterization anomaly. (a) Teapot; (b) pawn; (c) Venus; (d) couch.

B. *Edge Function Recovery Scheme*

In order to remove the rasterization anomaly, a recovery scheme based on the edge function method [19] is employed, where the edge function method is used in some raster engines to decide whether a pixel is inside the triangle. Assume that (x_a, y_a), (x_b, y_b) and (x_c, y_c) are the coordinates of vertex V_a, V_b, and V_c, respectively, the edge functions E_{ab}, E_{bc}, and E_{ca} in Fig. 2.1 defined by vertices V_a, V_b, and V_c can be expressed in (2.6), (2.7), and (2.8), respectively

$$E_{ab} : A_{ab}x + B_{ab}y + C_{ab} = 0 \tag{2.6}$$

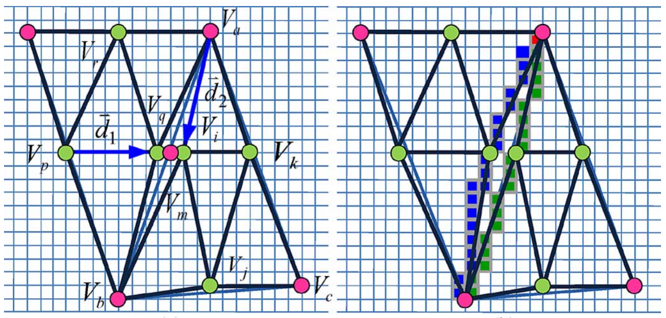


Fig. 2.3. Illustration of the rasterization anomaly with $Ns = 2$. (a) After subdivision; (b) rasterization result.

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

$$E_{bc} : A_{bc}x + B_{bc}y + C_{bc} = 0 \tag{2.7}$$

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

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

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

In order to eliminate the anomaly, the edge functions of the small subdivided triangles in Fig. 2.1 can be recovered in the following steps.

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

Step 2) Compute the constant difference values: ΔC_{ab}, ΔC_{bc} and ΔC_{ca} in (2.9), (2.10), and (2.11), respectively

$$\Delta C_{ab} = \frac{1}{2}(A_{bc}B_{ab} - A_{ab}B_{bc}) \tag{2.9}$$

$$\Delta C_{bc} = \frac{1}{2}(A_{ca}B_{bc} - A_{bc}B_{ca}) \tag{2.10}$$

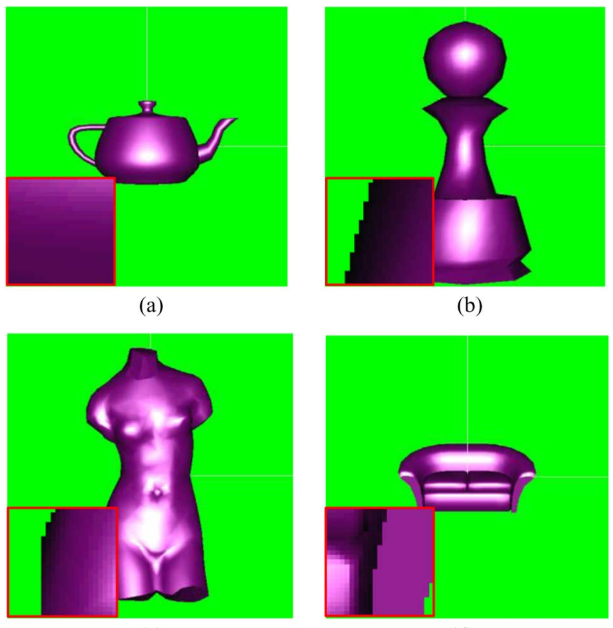
$$\Delta C_{ca} = \frac{1}{2}(A_{ab}B_{ca} - A_{ca}B_{ab}) \tag{2.11}$$

Step 3) Recover 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. 2.1 with the use of the original edge functions and the difference values. For example, E_{kj} can be obtained by (2.12)

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

where $A_{kj} = A_{ab}, B_{kj} = B_{ab}, C_{kj} = C_{ab} + \Delta C_{ab}$. The constant term C_{kj} can be obtained from the constant term C_{ab} of the edge function E_{ab} plus the difference value ΔC_{ab} in (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 edge function recovery scheme, 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. 2.4(a), (b), (c), (d). It



(d)

Fig. 2.4. Rendering results with the edge function recovery scheme. (a) Teapot; (b) pawn; (c) Venus; (d) couch.

is worth mentioning that although the anomaly will not happen if the sub-vertex on the edge is generated by only two end-vertices of the edge, this scheme will result in larger number of temporary registers to store the processed and on-processing vertices to calculate other vertices while number of subdivision levels increases. The processed vertex in the register can be discarded until the other vertices depending on this vertex are calculated. On the other hand, the forward difference scheme has the constant number of temporary registers for the larger subdivision level. The forward difference scheme and edge function recovery scheme can be easily applied to the larger subdivision level; however, the sub-vertex scheme generated by only two end-vertices of the edge needs more efforts to arrange the vertex processing sequence.

In (2.6), evaluating one edge function requires two multiplications and three subtractions. For a triangle with N_s segments on each edge, there are total $3N_S$ edge functions to be computed and computation requires $3N_S(2\text{ muls}+3\text{ subs}) = 6N_S$ muls+ $9N_S$ subs $=6N_S$ muls + $9N_S$ adds. Herein, one subtraction can be regarded as one addition. The proposed edge function recovery scheme computes each edge function for the subdivided triangle by adding one difference value. Using the proposed edge function recovery scheme, the original three edge functions require 3 (2 muls + 3 adds) and the rest $(3N_S - 3)$ edge functions require $(3N_S - 3)(1\text{ add})$ due to difference values. The three difference values in (2.9), (2.10) and (2.11) require 3(2 muls + 1 add). Thus, the computation complexity can be reduced to $3(2\text{ muls} + 3\text{ adds}) + (3N_S - 3)(1\text{ add}) + 3(2\text{ muls} + 1\text{ add}) = 12\text{ muls} + (3N_S + 9)\text{ adds}$. Thus, the edge function recovery scheme implies an efficient method for computing the edge functions of the subdivided triangles. Note that while $N_s = 1$ (i.e., $L = 0$), we do not need to produce generated vertices. Thus, the calculation for difference values can by bypassed and the result will be equal to 3(2 muls + 3 adds). Although the conventional algorithm and the proposed algorithm

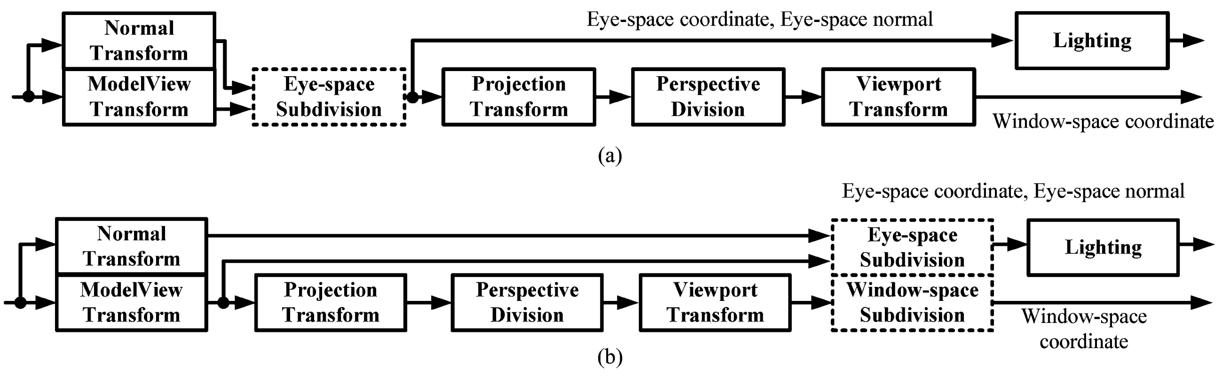


Fig. 2.5. (a) Data flow of eye space subdivision, and (b) data flow of the proposed dual space subdivision.

need the edge functions for all triangles, the proposed edge function recovery scheme can reduce the edge function computation. The contributions of the proposed edge function recovery scheme are as follows. One is to eliminate the anomaly due to forward difference calculation and the other is to reduce the edge function computation.

C. Dual Space Subdivision Scheme

In the geometry engine, a sequence of transforms including modelview transform, projection transform, perspective division, viewport transform is applied to the vertices. The modelview transform transforms the vertex from object space to eye space by multiplying a 4×4 modelview matrix [20, 4] in (2.13)

$$\begin{bmatrix} x_{\text{eye}} \\ y_{\text{eye}} \\ z_{\text{eye}} \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{object}}/w_{\text{object}} \\ y_{\text{object}}/w_{\text{object}} \\ z_{\text{object}}/w_{\text{object}} \\ 1 \end{bmatrix} \tag{2.13}$$

where $(x_{\text{object}}, y_{\text{object}}, z_{\text{object}}, w_{\text{object}})$ and $(x_{\text{eye}}, y_{\text{eye}}, z_{\text{eye}})$ denote the object-space coordinate and eye-space coordinate, respectively. In the projection transform, the eye-space coordinate is transformed to the clip-space coordinate by multiplying a 4×4 projection matrix [19, 4] in (2.14)

$$\begin{bmatrix} x_{\text{clip}} \\ y_{\text{clip}} \\ z_{\text{clip}} \\ w_{\text{clip}} \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_{\text{eye}} \\ y_{\text{eye}} \\ z_{\text{eye}} \\ 1 \end{bmatrix} \tag{2.14}$$

where $(x_{\text{clip}}, y_{\text{clip}}, z_{\text{clip}}, w_{\text{clip}})$ denotes the clip-space coordinate and r, l denote the x-coordinate component of the right edge and the left edge, respectively, of the near clipping plane and t, b denote the y-coordinate component of the top edge and the bottom edge, respectively, of the near clipping plane, and n, f denote the distances to the near clipping plane and far clipping plane, respectively. Next, through the perspective division, the vertices in the clip space will be projected to the projection plane by dividing w_{clip} and the normalized device coordinate (NDC) of each component in the range of $[-1, 1]$ can be expressed in (2.15)

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

where $(x_{NDC}, y_{NDC}, z_{NDC})$ denotes the normalized device coordinate. Finally, through the viewport transform, the NDC will be transformed to the window- (screen-) space coordinate in (2.16)

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

where $(x_{\text{window}}, y_{\text{window}}, z_{\text{window}})$ denotes the window-space coordinate, $x_{\text{scale}}, y_{\text{scale}}, z_{\text{scale}}$ denote scaling factors and $x_{\text{offset}}, y_{\text{offset}}, z_{\text{offset}}$ denote offset values. In addition, the normal transform is required to transform the normal vector of each vertex from the object space to the eye space by multiplying a 3×3 matrix. In this paper, we set $w_{\text{object}} = 1$ in our experiment. If w_{object} is not equal to one, in the homogeneous space [4] and above four transformation matrices, $(x_{\text{object}}, y_{\text{object}}, z_{\text{object}}, w_{\text{object}})$ can be scaled as $(x_{\text{object}}/w_{\text{object}}, y_{\text{object}}/w_{\text{object}}, z_{\text{object}}/w_{\text{object}}, 1)$ without affecting the window-space result after proceeding four transforms. We only need one time to precompute the scaled coordinate for the input model. While the precomputation is available, we can reuse this model for other graphics operations. As illustrated in Fig. 2.5(a), the conventional subdivision algorithm subdivides the triangles in the eye space. On one hand, because the subdivision generates a large number of vertices, theses vertices bring overhead to the computation for the later stages of the pipeline. On the other hand, for the triangle with non-large depth range, because the human eye is not sensitive to the light intensity with small difference, the perspective correctness computation of the generated vertices can be passed over. Based on the above reasons, the dual space subdivision as shown in Fig. 2.5(b) is proposed to reduce the complexity for the triangle with non-large depth range compared with the eye-space subdivision in Fig. 2.5(a). As illustrated in Fig. 2.5(b), the proposed subdivision scheme is performed to subdivide the eye-space coordinate and window-space coordinate after the modelview transform and viewport transform, respectively. The eye-space coordinate is required for point-light calculation and the window-space coordinate is used for edge function calculation and other geometry operations. By skipping three transforms including projection transform, perspective division and viewport transform for generated vertices, the computational complexity is reduced.

The complexity analysis of the eye-space subdivision of a single triangle is given in Table 2.1, where the left and right columns list the operations of subdivision as well as transform and the corresponding complexity, respectively. Note that, in

TABLE 2.1
COMPLEXITY ANALYSIS OF THE EYE SPACE SUBDIVISION FOR ONE ORIGINAL TRIANGLE

Operations	Computational Complexity
Modelview transform for 3 vertices	3x9 muls + 3x9 adds
Normal transform for 3 vertices	3x9 muls + 3x6 adds
Subdivision for 6 components : Eye coordinate: $(x_{eye}, y_{eye}, z_{eye})$ Normal: (x_N, y_N, z_N)	$6(4^L$-1) adds
Projection transform for N_{GV}+3 vertices	$5(N_{GV}$+3) muls + $3(N_{GV}$+3) adds
Perspective division for N_{GV}+3 vertices	$3(N_{GV}$+3) muls + $(N_{GV}$+3) invs
Viewport transform for N_{GV}+3 vertices	$3(N_{GV}$+3) muls + $3(N_{GV}$+3) adds
Total	(11 N_{GV}+87) muls (6 N_{GV}+6x4^L+ 57) adds $(N_{GV}$+3) invs

TABLE 2.2
COMPLEXITY ANALYSIS OF THE PROPOSED DUAL SPACE SUBDIVISION FOR ONE ORIGINAL TRIANGLE

Operations	Computational Complexity
Modelview transform for 3 vertices	3x9 muls + 3x9 adds
Normal transform for 3 vertices	3x9 muls + 3x6 adds
Projection transform for 3 vertices	3x5 muls + 3x3 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: $(x_{eye}, y_{eye}, z_{eye})$ Normal: (x_N, y_N, z_N) Window coordinate: $(x_{window}, y_{window}, z_{window})$ Reciprocal of depth: $(1/w_{clip})$	$10(N_{GV}$+2) adds
Total	87 muls (10 N_{GV} +83) adds 3 invs

this paper, we do not count the multiplication while one variable times the constant values of 0 or $\pm 2^i$, where i denotes the integer value. N_{GV} is defined as the number of the generated vertices during subdivision. After the triangle is subdivided, there are $(N_{GV} + 3)$ vertices, where 3 denotes three vertices of the original triangle. In the conventional subdivision algorithm, the original triangle is only subdivided in eye space. For the level-L case, the addition complexity of the recursive subdivision is $6(4^L - 1)$ additions. The reason is that, in each recursion, the number of generated vertices is $3(4^{L-1})$ for $L \geq 1$; therefore, the summation of the number of generated vertices for all recursions is $(4^L - 1)$. Since each new generated vertex requires 6 adds for eye-space coordinate and normal vector, the total number of additions is $6(4^L - 1)$. After subdivision, only $(N_{GV} + 3)$ vertices are used to assemble the small triangles and transformed by projection transform, perspective division and viewport transform. As described in this subsection, through the projection matrix with a 4 $\times$ 4 matrix, the computational complexity of the projection transform is equal to $5(N_{GV} + 3)$ muls $+ 3(N_{GV} + 3)$ adds. The perspective division for a vertex requires three multiplications and one inverse, and, therefore, the total computational complexity is $3(N_{GV} + 3)$ muls + $(N_{GV} + 3)$ invs for $(N_{GV} + 3)$ vertices. The viewport transform requires three multiplications and three additions for each vertex to scale and offset the normalized device coordinates (NDC). The computational complexity is $3(N_{GV} + 3)$ muls $+ 3(N_{GV} + 3)$ adds for $(N_{GV} + 3)$ vertices.

Compared to the eye-space subdivision, the proposed dual space subdivision scheme only needs to process three vertices instead of $(N_{GV} + 3)$ vertices for latter stage transforms. Thus, for each vertex, the eye-space coordinate with three components, normal vector with three components, window-space coordinate with three components, and one reciprocal of depth will be subdivided in two spaces with the forward difference scheme such that the computational complexity is $10(N_{GV} + 2)$ additions, where 2 is the number of operations to calculate the difference vectors. The total complexity of the proposed dual space subdivision scheme listed in Table 2.2 is 87 muls $+ (10N_{GV} + 83)$ adds + 3 invs. Note that while $N_{GV} = 0$ (i.e., $L = 0$), we do not need to produce generated vertices. Thus, the calculation for $10(N_{GV} + 2)$ adds can be bypassed and the result will be equal to 87 muls + 63 adds + 3 invs.

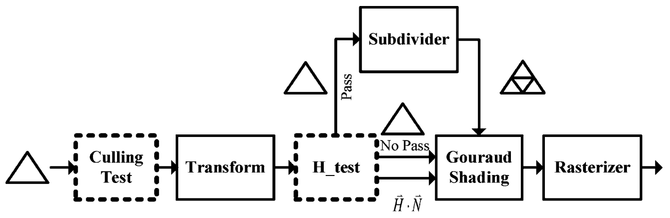


Fig. 2.6. Data flow of the triangle filtering scheme.

D. Triangle Filtering Scheme

To reduce the computation for primitive-level operations, the filtering scheme as shown in Fig. 2.6 is added to the proposed algorithm. The proposed triangle filtering scheme is a hybrid scheme that combines culling test and highlight test before subdivision [12], [13]. 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 dot product of the face normal vector and eye direction vector. If a triangle is a backface, it will be discarded and not rendered. Performing culling test for these subdivided 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 dot products of these face normal vectors and the eye direction vector will be the same. The result implies that there is no need to perform the backface culling test for each generated triangle since the results will be the same. Hence, in the proposed algorithm, the subdivision is performed after the culling test. If the original triangle is culled, the subdivision is unnecessary. Otherwise, all generated triangles are rendered by reusing the same culling test result of the original triangle. For the level-L case, N_s^2 triangles are generated, and, therefore, N_s^2 culling operations are required while the culling test is operated after the subdivider. With the triangle filtering scheme, only one culling operation is needed since the N_s^2 new generated triangles can reuse the culling result.

To reduce the redundant subdivision, the subdivision-based algorithm usually includes the highlight test scheme [13]. In the proposed algorithm, the highlight test [9] is adopted after the culling test. The scheme tests the $\vec{H} \cdot \vec{N}$ term of the original three vertices, where $\vec{N}$ and $\vec{H}$ denote the normalized normal vector and normalized halfway vector, respectively. While one of the $\vec{H} \cdot \vec{N}$ terms is greater than the threshold value, the triangle will be subdivided. If all $\vec{H} \cdot \vec{N}$ terms are smaller than the threshold value, we bypass the subdivision and render the triangle with Gouraud shading. Thus, the redundant primitive operations can be reduced. In our experiment, the threshold value is equal to 0.7 for level-1 and level-2 subdivision. The guideline of threshold value selection is described as follows.

Rule 1) The threshold value is between zero and one.

Rule 2) Determine which threshold value can result in the satisfactory lighting quality from the scene simulation for each level.

Rule 3) Choose the higher threshold value to reduce the number of triangles under satisfying the lighting quality for each level.

E. Triangle Setup Coefficient Sharing Scheme

To reduce the triangle setup and the unnecessary subdivision for vertex attributes, a triangle setup coefficient sharing scheme is exposed in this section. The concept of reusing setup result has been shown in [14]; however, the detailed processes are not addressed much. During rasterization, the vertex attributes are linearly interpolated for each pixel. These attributes include eye-space coordinate, texture coordinate, depth values, fog factors, and light intensities. The interpolation usually makes the use of the plane equation [21]. Given three attribute values u_0, u_1, and u_2 for three vertices, the coefficients of the attribute plane are obtained by solving (2.17)

$$[A_i \quad B_i \quad C_i] = [u_0 \quad u_1 \quad u_2] \begin{bmatrix} x_0 & x_1 & x_2 \\ y_0 & y_1 & y_2 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \tag{2.17}$$

where (x_i, y_i) is the window-space coordinate of the triangle. After obtaining 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 coordinates.

The generated triangles using the conventional subdivision algorithm increase the complexity of the triangle setup. Because the generated triangles are on the same plane, the coefficients of the attribute plane can be shared by the generated triangles without re-computing these coefficients. This sharing scheme is referred to as the setup coefficient sharing scheme. In Fig. 2.1, the triangle is subdivided into four small triangles, and, therefore, the conventional setup cost for one vertex attribute of these triangles is four 3×3 matrix inversions and four 3×3 matrix multiplications. With the setup coefficient sharing scheme, the setup only requires one 3×3 matrix inversion and one 3×3 matrix multiplication because the precomputed coefficients are shared by the small triangles. After obtained all attribute planes, the attribute value of a vertex is obtained by substituting the vertex coordinate into the attribute plane equation. The complexity of substitution is 2 muls + 2 adds and that is about 1/3 3×3 matrix multiplications. Hence, the complexity of the proposed coefficient sharing scheme is $(\lceil (1/3) N_A N_S^2 \rceil + N_A)$ 3×3 matrix multiplications and one 3×3 matrix inversion, where N_A denotes the number of attributes and $\lceil x \rceil$ denotes the ceiling operation for x.

III. Proposed Geometry Engine Architecture

In this section, a power-area efficient geometry engine (GE) architecture for 3-D graphics pipeline architecture is proposed. Several kernel blocks including the primitive processing unit (PPU), vertex processing unit (VPU) and vertex cache management unit (VCMU) are proposed to reduce the power-area consumption and to support the scalable quality mechanism via the proposed subdivision algorithm for GE. The supported scalable quality levels are level-0 ($L = 0$), level-1 ($L = 1$) and level-2 ($L = 2$). The overall architecture of the proposed GE is depicted in Fig. 3.1, where the triangle clipping function is not realized in the proposed architecture. The detailed descriptions of each block are given in the following subsections.

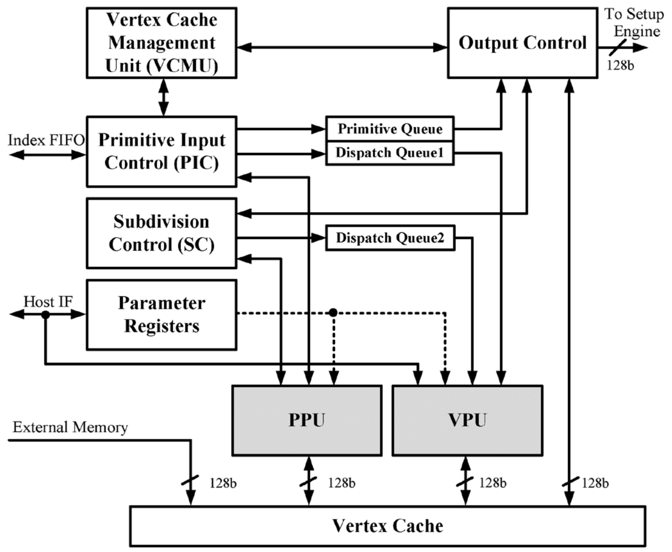


Fig. 3.1. Overall architecture of the proposed GE architecture.

A. Primitive Input Control (PIC)

The main task of the primitive input control (PIC) is to process the input primitive information. 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-space coordinate and normal vector) from the external memory. The vertex data returned from the external memory will be stored in the vertex cache. If the cache hits, the vertex data are not fetched from the external memory because it is already in the vertex cache. The triangles defined by the indices are assembled by PIC and then the backface culling test operated in PPU is issued for the assembled triangles. If the triangle is a backface, it will be discarded from PIC. Otherwise, the cache entries of the three vertices of the triangle are pushed to the primitive queue (PQ) and the un-processed vertices that belong to the triangle are pushed into the dispatch queue 1 (DQ1) for further processing. The processed vertices will not be pushed to DQ1 because they are already transformed and lit.

B. Primitive Queue (PQ) and Dispatch Queue (DQ)

The primitive queue (PQ) is responsible for buffering the cache entries of three vertices of a triangle for processing. 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 unit pops the triangle from PQ and reads the vertex data (window-space coordinate and light intensity) of the triangle from vertex cache and then outputs data to the setup engine. The main task of the dispatch queue (DQ) in Fig. 3.2 consisting of two vertex-cache-entry buffers is to keep the cache entries of the un-processed vertices. The VPU can access the vertex data with this information. With the exchangeable buffer architecture, the PIC and VPU can operate concurrently and thus the performance is increased. In DQ1 and DQ2, the size of each buffer is six for the three-level subdivision algorithm.

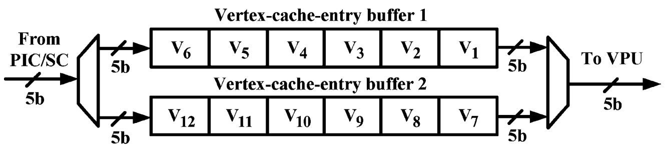


Fig. 3.2. Illustration of the dispatch queue.

C. Subdivision Control (SC)

The main task of the subdivision control (SC) is to control PPU to subdivide the triangle which passes the highlight test. Whenever the three vertices of the triangle are lit, the output control checks the test results of the triangle and delivers the primitive information of the triangle to SC if the test is passed. The subdivision process in SC is accomplished with two phases. The normal vector and the two-space coordinates are subdivided at the first and second phase, respectively. At each phase, the SC requests PPU to perform subdivision and provides PPU the primitive information including the attribute to be subdivided and the subdivision level. After the two phases are completed, the SC pushes the cache entries of the new generated vertices to the dispatch queue 2 (DQ2). These vertices will be lit in VPU. When all the generated vertices are lit, the SC requests the output control to output the vertices of the subdivided triangle.

D. Vertex Cache Management Unit (VCMU) and Vertex Cache

The vertex cache management unit (VCMU) is responsible for supporting the subdivision algorithm, where a vertex cache tag unit is needed. The vertex cache contains 16-tag entries and each entry has seven fields including vertex_index, tag_entry_available, zero_vertex_count, vertex_count, vertex_in_pipe, vertex_lit, vertex_Htest fields as illustrated in the first entry in Fig. 3.3, where the detailed field descriptions are listed in Table 3.1. When PIC requests VCMU to check whether a vertex exists in the cache, the searched vertex index is compared with the vertex_index field of each tag entry. If the vertex_index matches one of the valid tag entries, the entry_hit signal of the tag entry asserts and VCMU returns hit. The entry address of the vertex is obtained by encoding the entry hit_vector in Fig. 3.3 and returned to PIC. If the vertex_index does not match any tag entry, VCMU returns miss and PIC will fetch the vertex data from the external memory. Before PIC fetches the vertex data, the PIC requests VCMU to allocate one tag entry for the vertex. A tag entry can be allocated if the tag_entry_available field is 0 or the zero_vertex_count field is 1. When one of conditions is met, the entry_free signal asserts. The entry address of the available tag entry is obtained by encoding the entry free_vector in Fig. 3.3. The vertex_count field is necessary to prevent the on-processing vertices from being replaced by the incoming vertices. When the cache hits, the vertex_count field is added by one since a new vertex enters the pipeline and refers to the existing vertex in the cache. The vertex_count field is subtracted by one when a vertex exits the pipeline. When the vertex_in_pipe field or vertex_lit field is set to 1, the vertex is not pushed into DQ since the vertex has already been transformed and lit. With the two fields, the vertex cache can act as a post cache such that the processed vertices can be reused/read from the cache without extra computation.

The vertex_Htest field stores the result of the highlight test for the subdivision algorithm. With this field, the computation can be reduced because the highlight test for the shared vertex is only performed once and the result can be reused by the triangles with the shared vertex. According to the cache analysis of [22], the 16-entry vertex cache is selected for the post transformation and lighting operation, where the cache size is 448 bytes in this paper. Using the 16-entry vertex cache, the four hit rates are $1945/3072 = 63.3\%$, $1731/2886 = 60\%$, $2316/4254 = 54.4\%$, $489/912 = 53.6\%$ for teapot, pawn, Venus, and couch, respectively, where numerator and denominator denote the number of hits and vertex accesses, respectively. From the simulation results, the average hit rate is 57.8%.

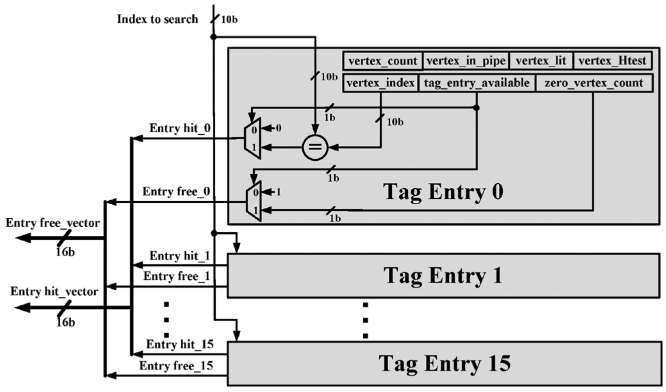


Fig. 3.3. Illustration of the vertex cache management unit.

TABLE 3.1
GLOSSARY OF FIELDS USED IN VCMU AND THE CORRESPONDING DETAILED FIELD DESCRIPTIONS

Field	Detailed field descriptions
vertex index	Indicate the vertex index
tag_entry_available	Indicate which and whether the tag entry is available to allocate
zero_vertex_count	Indicate whether the vertex count reaches zero value
vertex_count	Indicate the number of vertices
vertex_in_pipe	Indicate whether the vertex being processed in the pipeline
vertex_lit	Indicate whether the vertex is lit
vertex_Htest	Indicate whether the vertex passes the highlight test result

E. Primitive Processing Unit (PPU)

The primitive processing unit (PPU) performs primitive-level operations including backface culling [23] and subdivision algorithm. The block diagram of the proposed PPU architecture is depicted in Fig. 3.4, where the bit width of each node has been marked for clear presentation. The vertex input buffers store the primitive data for culling or subdivision operations. Before these operations start, the controller loads the corresponding data into the input buffers from the vertex cache. Because the window-space coordinate has three components and one reciprocal of depth, four subtractors are required for the subdivision operation. For the culling operation, the vector subtraction and dot product operations are involved [4]. The vector subtraction can be implemented with three subtractors and the dot product operation can be implemented with one multiplier and one adder. Considering the area efficiency, three subtractors, one adder/subtractor and one multiplier as illustrated in Fig. 3.4 is capable of handling the above operations.

F. Vertex Processing Unit (VPU)

The vertex processing unit (VPU) performs vertex-level operations including vertex transformation and lighting operation.

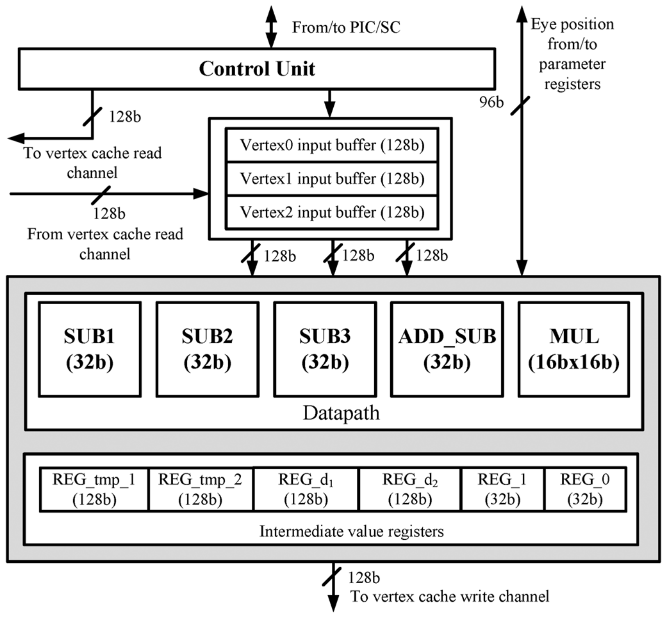

From/to PIC/SC
Control Unit
Eye position from/to parameter registers
128b
96b
To vertex cache read channel
Vertex0 input buffer (128b)
Vertex1 input buffer (128b)
Vertex2 input buffer (128b)
From vertex cache read channel
SUB1 (32b)
SUB2 (32b)
SUB3 (32b)
ADD_SUB (32b)
MUL (16bx16b)
Datapath
REG_tmp_1 (128b)
REG_tmp_2 (128b)
REG_d1 (128b)
REG_d2 (128b)
REG_1 (32b)
REG_0 (32b)
Intermediate value registers
To vertex cache write channel

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

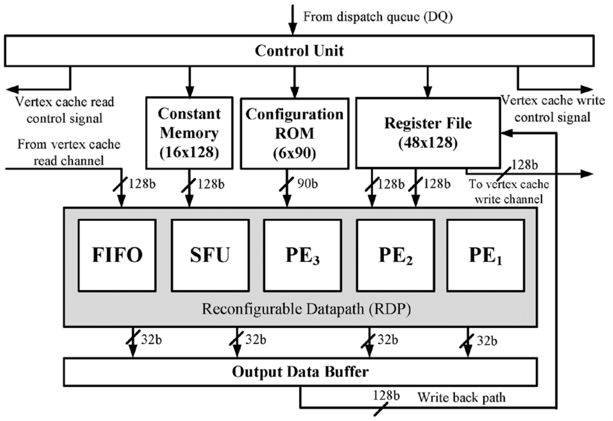


Fig. 3.5. Block diagram of the vertex processing unit.

The operations cover modelview transform, projection transform, perspective division, viewport transform, normal transform, vector normalization and Blinn-Phong reflection model. The Blinn-Phong reflection model [4] can be formulated in (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 direction vector, and normalized halfway vector, respectively. The halfway vector $\vec{H} = (\vec{L} + \vec{V})/2$ is the vector between the light direction vector $\vec{L}$ and the viewing vector $\vec{V}$. The block diagram of the proposed VPU architecture is depicted in Fig. 3.5, where the bit width of each node has been marked for clear presentation. The vertex data are read from the read channel of the vertex cache. These vertices are transformed and lit in the reconfigurable datapath (RDP). The register file stores the intermediate values for the on-processing vertices. The constant memory stores the lighting parameters and the matrix parameters for the transform matrix. After all vertices in the batch are transformed and lit, the vertices are read from the register file and written back to vertex cache.

Due to the proposed low computational complexity algorithm, the requirement of the number of processing units for vertex processing can be alleviated. Considering the trade-off for the power, area and vertex processing performance of the GE architecture, the vertex operations in Table 3.2 including dot products for transform and lighting, vector normalization, perspective division, powering, and vector subtraction are implemented by the proposed RDP architecture. The proposed RDP composed of three processing elements (PEs), one special function unit (SFU) and one FIFO in Fig. 3.5 is a pipelined SIMD datapath architecture and can be reconfigured to six configuration modes. Due to reconfigurability [24] and SIMD datapath architecture, the proposed RDP can improve the area efficiency and performance. That means the proposed GE belongs to the reconfigurable engine due to RDP rather than the programmable processors. The detailed descriptions about the RDP are given in the following subsections.

TABLE 3.2
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

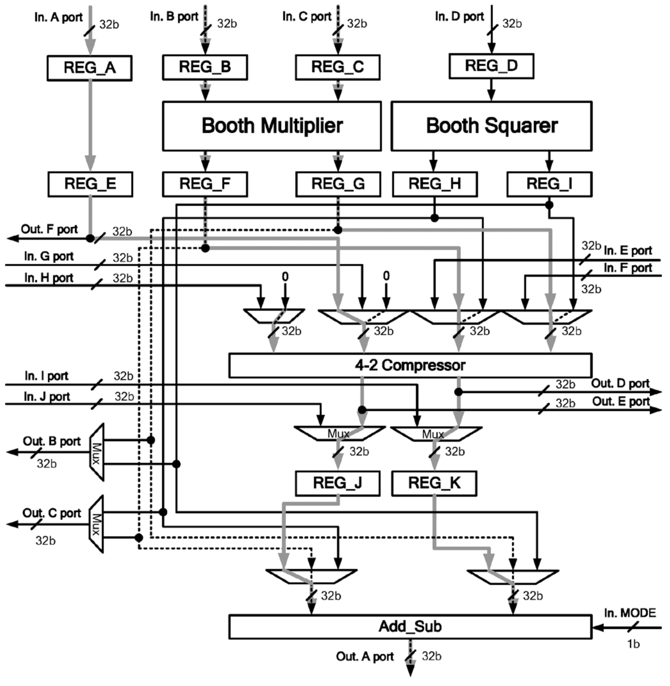


Fig. 3.6. PE configured as multiplication in dash lines or MAC operations in gray lines.

1) Processing Element (PE): The processing element (PE) with three-stage pipeline can be configured to perform multiplication in Fig. 3.6, square, multiplication-accumulation (MAC) in Fig. 3.6, and addition/subtraction such that the configuration modes in Table 3.2 can be supported. In PE as shown in Fig. 3.6, at the first stage, the fixed-width Booth multiplier multiplies two 32-bit numbers and generates two 32-bit partial products. The Booth-based squarer [25] is operated in fixed-width format, where the outputs of the squarer at the first stage are two 32-bit partial products. At the second stage, the 32-bit 4-2

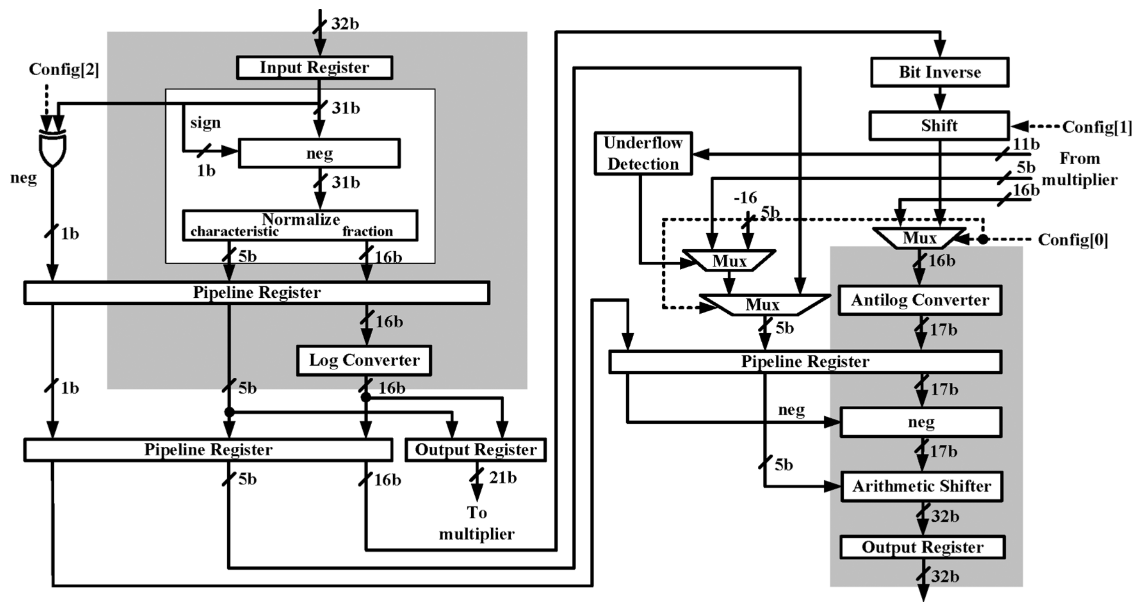


Fig. 3.7. Block diagram of the special function unit.

compressor is used to add four inputs and generates two partial products. At the last stage, the adder-subtractor unit adds or subtracts two numbers and produces the final 32-bit result. The function of the adder-subtractor is controlled by the In. MODE signal in Fig. 3.6. The multiplexers in PE control the data flow for different operations.

The datapath of multiplication operation is illustrated in Fig. 3.6 with dashed lines. The first stage of multiplication 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 operation is similar to that in Fig. 3.6. 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. 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. The datapath of the MAC operation is illustrated in Fig. 3.6 with gray lines. For the addition/subtraction operation, the pipeline registers REG_J and REG_K are configured to be the input registers, where two inputs come from In. I port and In. J port. The numbers in REG_J and REG_K are added or subtracted according to the target operations. Other interconnection wires of the PE as discussed in next two subsection will be used to connect to other PEs for vertex operations.

2) Special Function Unit (SFU): The main task of the SFU is to support the vector normalization, perspective division, and powering modes in Table 3.2. In [26], the special function unit (SFU) is capable of providing various elementary functions. In this paper, since the power-efficient arithmetic operations are taken into account for the proposed subdivision algorithm, the SFU adopts the logarithmic number system (LNS) [24], [27], [28] to realize the inverse, inverse-square-root and power (POW) operations.

The block diagram of the proposed SFU architecture is depicted in Fig. 3.7, where the bit width of each node has been marked for clear presentation. In Fig. 3.7, Config[0] controls the source for the antilogarithmic converter, Config[1] controls the behavior of the shifter, and Config[2] controls the negating unit at the final stage. For the inverse and the inverse-square-root operations, the logarithmic converter as shown in the left gray region of Fig. 3.7 converts the input number m to the corresponding logarithmic number M. Then, the number M takes one's complement for each bit through the bit inverse block to produce the result $-(M+1)$. In order to reduce the bit transition, $-M$ is approximated by $-(M+1)$. In the shift block, the number $-M$ is shifted right one bit to obtain $(-M) \gg 1$ for inverse-square-root operation or directly bypassed for inverse operation. The behavior of the shift block is controlled by the Config[1]. The output logarithmic number $(-M) \gg 1$ or $-M$ of the shift block is then converted to the corresponding ordinary fixed-point number $1/\sqrt{m}$ or $1/m$ by the antilogarithmic converter as shown in the right gray region of Fig. 3.7. For the POW operation m^n, the number m is first converted to the logarithmic number M. Next, the multiplier is required to compute nM, where the processing element (PE) of the RDP in Fig. 3.6 can be configured to be a multiplier to compute nM. In Fig. 3.7, the logarithmic number M is outputted to a PE which is configured as a multiplier and is multiplied by the number n. Finally, the result nM is returned from the PE and converted to its ordinary number m^n. Thus, the real multiplier is not needed in the SFU to achieve power-area efficiency.

Since the underflow of the power operation will incur the discontinuity of light intensity, the underflow detection unit in the SFU is provided. While the underflow occurs, the saturation scheme with minimum value representation can be applied to compensating the underflow value. The datapath can be correctly operated for four-benchmark simulations. On the other hand, the overflow saturation scheme probably leads to not exact enough for the vertex coordinate calculation during the matrix multiplication. In the proposed geometry engine, we can modify the matrix parameters in (2.13), (2.14) and (2.16) or lighting parameters in (3.1) to avoid the overflow such that test scenes can be run correctly. In terms of precision, the accuracy of representative cases of the inverse, inverse square root, power, and multiplication are listed in Table 3.3, where the overflow cases are excluded.

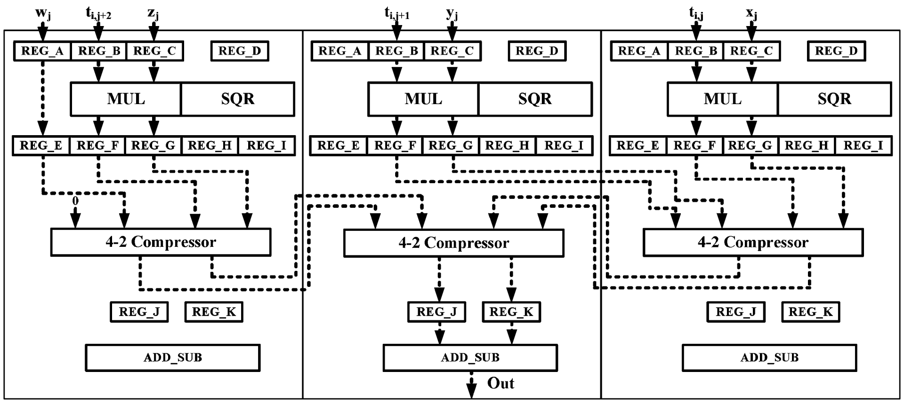


Fig. 3.8. Configuration mode interconnection of the dot product for transform.

TABLE 3.3
PRECISION OF EACH FUNCTION UNIT

Function	Max Error	Average Error
Inverse	8.25	$7.1\text{x}10^{-6}$
Inverse square root	$9.16\ \text{x}10^{-2}$	$6.5\ \text{x}10^{-6}$
Power	$1.83\ \text{x}10^{-3}$	$5.82\ \text{x}10^{-5}$
Multiplication	$3.91\ \text{x}10^{-3}$	$3.14\ \text{x}10^{-4}$

3) Interconnection of Configuration Modes: In this subsection, the interconnections among the building blocks for different configuration modes are described. For convenience of explanation, the block diagram of the PE is simplified. The modelview and projection transforms in (2.13) and (2.14), respectively, are achieved by multiplying a 4 × 4 matrix and a 4 × 1 column vector. Because the term w_{object} in (2.13) is one and the projection matrix in (2.14) is a sparse matrix, the dot products of these transforms can be achieved by three multipliers and three adders. The datapath for the transform operation is composed of three PEs and the interconnections between PEs are illustrated in Fig. 3.8. At the first stage, the three multiplications are performed by the partial-product multipliers in PEs, respectively. The addend w_j is directly passed to the next stage. At the second stage, the partial products and the addend w_j are added 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.

Since the dot product for lighting has three multiplications, the datpath is similar to the datapath of dot product for transform as illustrated in Fig. 3.8, 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 leftmost PE are forced to be zero. In the lighting equation, the normalized normal vector, normalized light vector and normalized halfway vector are with the unit length before computing dot products. The vector normalization is expressed in (3.2)

$$[\overline{x_j}, \overline{y_j}, \overline{z_j}] = \left[\frac{x_j}{\text{Length}}, \frac{y_j}{\text{Length}}, \frac{z_j}{\text{Length}}\right] \tag{3.2}$$

where $Length = \sqrt{x_j^2 + y_j^2 + z_j^2}$. As illustrated in Fig. 3.9, the solid-line datapath evaluates the length of the input vector. The square operations are performed in the dedicated squarers of the PEs and the output partial products are added with the 4-2 compressor. Because all add-subtractor units in the PEs are occupied by the dashed-line datapath, one additional adder is included to sum the two outputs of the compressor in the central PE. The produced length square value is passed to SFU to evaluate the inverse square root of the length square. Then, the input vector is multiplied by the inverse of the length value to obtain the normalized vector. The dashed-line part in Fig. 3.9 shows the operation of vector scaling.

In the perspective division in (2.15), the clip-space coordinate, $(x_{\text{clip}}, y_{\text{clip}}, z_{\text{clip}})$, is divided by the term w_{clip}, where $1/w_{\text{clip}}$ can be obtained by SFU. The perspective division can be realized by three PEs as shown in Fig. 3.6, where each PE multiplies one clip-space coordinate and $1/w_{\text{clip}}$.

It is worth mentioning that the proposed engine only needs three PEs rather than four PEs from (2.13)to (2.16), but the additional cycles are needed for one-time precomputation of the scaled coordinate to save one PE. While the precomputation is available, it is beneficial that we can reuse this model for other graphics operations with one PE saving. In addition, we have simulated the minimized depth difference in z-axis between two triangles for the proposed engine architecture, where the simulation environment setting is the same as that of four test scenes. The minimized depth difference between two triangles is 0.00037. The overlapping does not happen while the depth difference is greater than or equal to the minimized depth difference. Otherwise, the overlapping will occur. If the overlapping occurs, we can add the offset to z value of one triangle in the rasterization part with software solution to avoid this overlapping. In summary, the major architecture differences from the previous papers [29]–[32] are as follows. 1) Propose the reconfigurable datapath (RDP) consisting of three PEs, one SFU, one FIFO to execute six modes in Table 3.2. 2) Propose the reconfigurable PE to configure as multiplication, square, MAC, or addition/subtraction. 3) Propose the SFU to save one multiplier by reusing the multiplier configured by the PE. 4) Propose the subdivision control (SC) and seven-field VCMU for subdivision management.

IV. COMPARISON RESULTS AND CHIP LAYOUT

In this section, the comprehensive comparison results in terms of complexity for the proposed and conventional subdivision algorithms and power-area efficiency (PAE) index among the existing geometry engines are addressed.

A. Complexity and Quality Comparison Results

The complexity comparison to Phong shading algorithm and the conventional subdivision algorithm is listed in Table 4.1 in terms of number of memory/register accesses, computation for edge functions, computation for transforms and subdivision, number of culling test operations, number of 3 × 3 matrix operations of setup, computation for normalization and Blinn-Phong

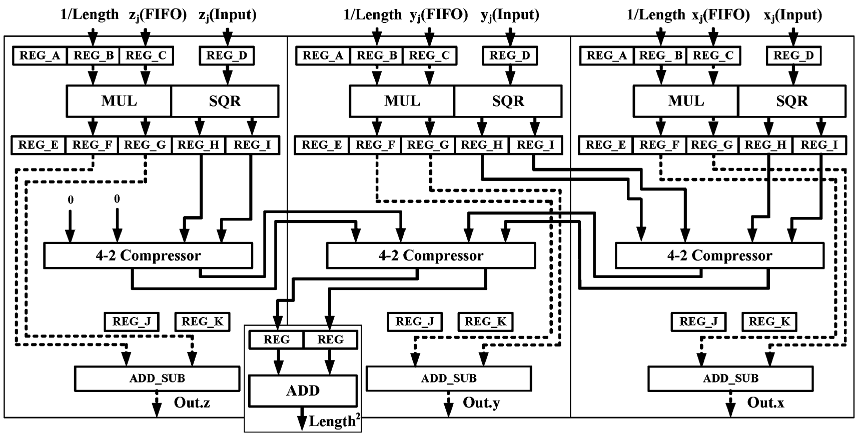


Fig. 3.9. Configuration mode interconnection of the vector normalization.

TABLE 4.1
COMPLEXITY COMPARISON RESULTS IN GENERAL REPRESENTATION AMONG PHONG SHADING ALGORITHM, CONVENTIONAL SUBDIVISION ALGORITHM AND PROPOSED SUBDIVISION ALGORITHM

		Phong-shading algorithm	Conventional subdivision algorithm			Proposed subdivision algorithm		
			L=0	L>0	Used Schemes	L=0	L>0	Used Schemes
Number of memory/ register accesses for subdivision		*0*	*0*	$(4^{L+1}-1)N_T$	Recursive subdivision	0	$(2N_{GV}-2^L+5)N_T$	Forward difference
Computation for edge functions	Muls	$6N_T$	$6N_T$	$6N_SN_T$	Without edge function recovery	$6N_T$	$12N_T$	Edge function recovery
	Adds	$9N_T$	$9N_T$	$9N_SN_T$		$9N_T$	$(3N_S+9)N_T$	
Computation for transforms and subdivision	Muls	$87N_T$	$87N_T$	$(11N_{GV}+87)N_T$	Eye space subdivision	$87N_T$	$87N_T$	Dual space subdivision
	Adds	$63N_T$	$63N_T$	$(6N_{GV}+6\text{x}4^L+57)N_T$		$63N_T$	$(10\,N_{GV}+83)N_T$	
	Invs	$3N_T$	$3N_T$	$(N_{GV}+3)\,N_T$		$3N_T$	$3N_T$	
Number of culling test operations		$1N_{OT}$	$1N_{OT}$	$1N_{OT}$	Triangle filtering	$1N_{OT}$	$1N_{OT}$	Triangle filtering
Number of 3x3 matrix operations for setup	Matrix Muls	N_AN_T	N_AN_T	$N_A\,N_S^2N_T$	Without setup coefficient sharing	N_AN_T	$(\lceil(\frac{1}{3})N_AN_S^2\rceil+N_A)N_T$	Setup coefficient sharing
Computation for Normalization	Muls	$6(\sum_{i=1}^{N_T}N_{Pi})$	$18N_T$	$18N_S^2N_T$	Normalization	$18N_T$	$6(N_{GV}+3)N_T$	Normalization
	Adds	$2(\sum_{i=1}^{N_T}N_{Pi})$	$6N_T$	$6N_S^2N_T$		$6N_T$	$2(N_{GV}+3)N_T$	
	Invs	$1(\sum_{i=1}^{N_T}N_{Pi})$	$3N_T$	$3N_S^2N_T$		$3N_T$	$1(N_{GV}+3)N_T$	
	Sqrt	$1(\sum_{i=1}^{N_T}N_{Pi})$	$3N_T$	$3N_S^2N_T$		$3N_T$	$1(N_{GV}+3)N_T$	
Computation for Blinn-Phong reflection model	Muls	$12(\sum_{i=1}^{N_T}N_{Pi})$	$36N_T$	$36N_S^2N_T$	Lighting	$36N_T$	$12(N_{GV}+3)N_T$	Lighting
	Adds	$10(\sum_{i=1}^{N_T}N_{Pi})$	$30N_T$	$30N_S^2N_T$		$30N_T$	$10(N_{GV}+3)N_T$	
	Power	$1(\sum_{i=1}^{N_T}N_{Pi})$	$3N_T$	$3N_S^2N_T$		$3N_T$	$1(N_{GV}+3)N_T$	

reflection model for rasterizaiton. In Table 4.1, N_{Pi}, N_{OT}, N_T denote the number of pixels of the i-th original triangle, the number of original triangles for input model, the number of original visible triangles for subdivision and Phong shading, respectively. Herein, for fair comparison, assume that all original visible triangles are subdivided as our evaluation case. Compared with Phong shading computation in Table 4.1, since N_{Pi} is much larger than N_{GV} or N_S^2 in general cases, it can be observed that Phong shading has highest computation complexity due to the hugest normalization and reflection model computation. Under the same constraint of no anomaly, the conventional subdivision algorithm includes recursive subdivision, conventional edge function without edge function recovery, eye space subdivision, triangle filtering, conventional setup up without setup coefficient sharing, and normalization and reflection model computation, where the triangle filtering scheme similar to Fig. 2.6 moves latter three transforms after the subdivider. Applying the conventional recursive subdivision algorithm to one original triangle, in each recursion, the number of the memory/register accesses is equal to $12(4^{L-1})$ for $L \geq 1$, where initial three accesses for three vertices that written into three input registers are excluded. Therefore, the summation of the number of the memory/register accesses for all recursions plus the initial 3 accesses is $(4^{L+1}-1)$. In the same case, the memory/register access is equal to $(2N_{GV}-2^L+5)$ using the forward difference scheme. It is worth noting that the values of other terms in Table 4.1 have been clearly discussed in Section II and the level-0 subdivision in Table 4.1 is Gouraud shading scheme. For level-1 case with $N_{GV}=3$ and $N_A=5$ and level-2 case with $N_{GV}=12$ and $N_A=5$, the quantitative comparison is listed in

TABLE 4.2
COMPLEXITY COMPARISON RESULTS FOR LEVEL-1 CASE WITH $N_{GV} = 3$ AND $N_A = 5$ AND LEVEL-2 CASE WITH $N_{GV} = 12$ AND $N_A = 5$

		Conventional subdivision algorithm		Proposed subdivision algorithm		Complexity reduction percentage	
Subdivision level number		*Level-1*	*Level-2*	*Level-1*	*Level-2*	*Level-1*	*Level-2*
Number of memory/register accesses for subdivision		*15*N_T	*63*N_T	*9*N_T	*25*N_T	40.00%	60.32%
Computation for edge functions	Muls	*12*N_T	*24*N_T	*12*N_T	*12*N_T	0%	50.00%
	Adds	*18*N_T	*36*N_T	*15*N_T	*21*N_T	16.67%	41.67%
Computation for transforms and subdivision	Muls	*120*N_T	*219*N_T	*87*N_T	*87*N_T	27.50%	60.27%
	Adds	*99*N_T	*225*N_T	*113*N_T	*203*N_T	-14.14%	9.78%
	Invs	*6*N_T	*15*N_T	*3*N_T	*3*N_T	50.00%	80.00%
Number of 3x3 matrix operations for setup	Matrix Muls	*20*N_T	*80*N_T	*12*N_T	*32*N_T	40.00%	60.00%
	Matrix Invs	*4*N_T	*16*N_T	*1*N_T	*1*N_T	75.00%	93.75%

TABLE 4.3
NUMBER OF TRIANGLES AND POWER CONSUMPTION AMONG FOUR TEST SCENES WITH THREE-LEVEL SUBDIVISION

Scenes	N_{OT}	*Level-0*		*Level-1*		*Level-2*	
		N_{VT}	Power (mW)	N_{VT}	Power (mW)	N_{VT}	Power (mW)
Teapot	1024	466	28.3	1075	33.6	3463	43.6
Pawn	304	156	30.7	294	34.2	846	42.3
Venus	1418	712	28.9	2248	37.4	8392	46.9
Couch	962	541	29.2	1454	35.8	5102	45.4
Ave.	927	469	29.3	1268	35.3	4451	44.6

Table 4.2. The reduction of the number of memory/register accesses for subdivision can be attained by 40% and 60.32% for level-1 and level-2 subdivision, 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 subdivision, respectively. The reduction of the number of multiplications for transforms can be attained by 27.50% and 60.27% for level-1 and level-2 subdivision, respectively. In terms of 3×3 matrix multiplications of setup operation for rasterizaiton, the computation can be alleviated by 40% and 60% for level-1 and level-2 subdivision, respectively. From above analysis results, the proposed subdivision algorithm can attain low-complexity computation. So as to observe the distribution of generated triangles for real test scenes, the N_{OT} and the number of visible triangles for output model, N_{VT}, using three subdivision levels are listed in Table 4.3. For each test scene, the larger subdivision level results in larger number of triangles.

In terms of near-Phong shading quality, it is difficult to compare quality in quantitative way. Thus, we compare shading quality with those using Phong shading as shown in Figs. 4.1(a), (b), (c), and (d) by four test scenes. From the simulation comparison results, we can observe that the quality with level-2 subdivision in Fig. 2.4 is close to near-Phong-shading quality. Thus, the perspective correctness computation of the generated vertices can be saved for the triangle with non-large depth range compared with the conventional subdivision algorithm. On the other hand, for the triangle with deeper depth range, the proposed subdivision algorithm may result in large quality difference. It is worth emphasizing that the proposed design does not change the normal vector of the original triangle due to the triangle subdivision. That means the normal vector of the triangle subdivision will be the same as $\vec{N}$ of the corresponding original triangle in the mesh subdivision. Thus, the reflection line will be the same.

B. Chip Layout and Comparison Results

Concerning the chip layout of the proposed GE architecture, the cell-based design flow with Artisan standard cell library in TSMC 0.18-um 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 postlayout simulation. The chip layout of the proposed GE is shown in Fig. 4.2, where 4 core power PADs and 12 IO power PADs are planned and the width of the power ring is 65 um. The gate count reported by SOC-Encounter is 182,779. The rendering results for the teapot benchmark with different subdivision levels including level-0, level-1, and level-2 subdivision are shown in Figs. 4.3(a), (b) and (c), respectively, where the teapot benchmark has 1,024 triangles. In Table 4.3, the three average power consumptions for level-0, level-1 and level-2 subdivision are 29.3 mW, 35.3 mW, and 44.6 mW, respectively. That means the average current density for level-2 subdivision is 0.38 mA/um obtained by 44.6 mW/(1.8 V $\times$ 65 um) at 1.8 V. The postlayout power in different subdivision levels is mainly affected by whether the architecture performs more computations per cycle on average. For one test scene, while increasing the subdivision level, the PPU generates more new vertices of the visible triangles in Table 4.3 and the dispatch queue has higher probability to be full such that VPU can be performed more frequently during the same number of computation cycles. That means VPU performs more computations per cycle on average. Thus, the larger power consumption is induced at higher subdivision level. Note that, in this experiment, the GE operations are implemented in hardware and the rasterization part is realized in software.

The comparison results between prior work [29]–[32] and our work are summarized in Table 4.4. In order to consider the effects of power and area, the power-area efficiency (PAE) metric adopted in [31] is expressed in (4.1)

$$PAE = \frac{\text{Peak Performance of Geomerty Transform(Kvetices/s)}}{\text{Power(mW)}\bullet\text{Core Area(mm}^2)} \tag{4.1}$$

TABLE 4.4
COMPARISON RESULTS AMONG THE EXISTING GEOMETRY ENGINE WORKS

	Sohn *et al.* [29]	Yu *et al.* [30]	Nam *et al.* [31]	Chien *et al.* [32]	This Work *Level-0*	This Work *Level-1*	This Work *Level-2*
Process (nm)	180	180	180	180	180		
Frequency (MHz)	200	100	200	50	100		
Polygon Rate (Mvertices/s)	50	120	141	25*1/12.5*2	50*1/25*2		
Power (mW)	155*3	157	52.4	8.6	29.3	35.3	44.6
Core Area (mm^2)	23*3	16	9.7*3	6.05*4	2.73		
Power-Area Efficiency (Kvertices/(s•mW•mm^2))	14	47.8	227	480.5	625.1	518.8	410.7
Feature	Graphics	Graphics	Graphics	Graphics, DSP	Graphics with scalable-quality hardware support		

*1: With cache hit rate of 50%. *2: With cache hit rate of 0%. *3: Include rendering engine. *4: With the core area of 2.164 mm × 2.797 mm and see acknowledgement.

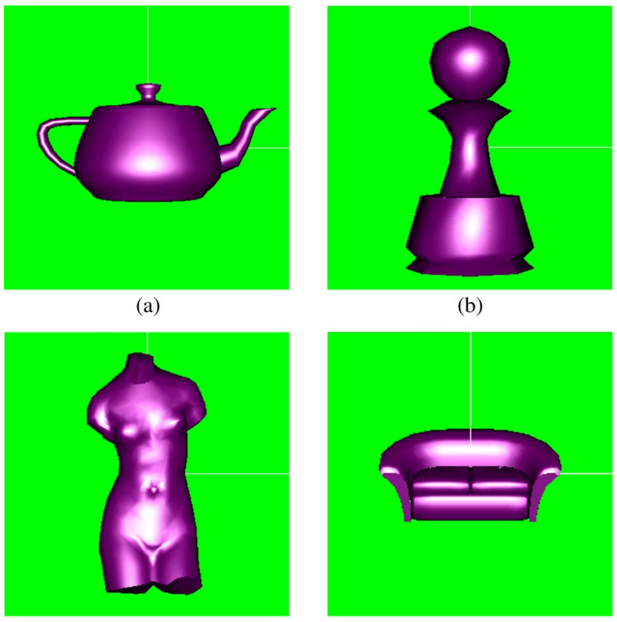


Fig. 4.1. Rendering results with Phong shading algorithm. (a) Teapot; (b) pawn; (c) Venus; (d) couch.

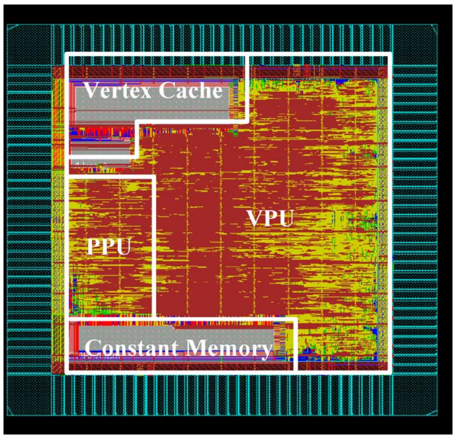


Fig. 4.2. Chip layout of the GE.

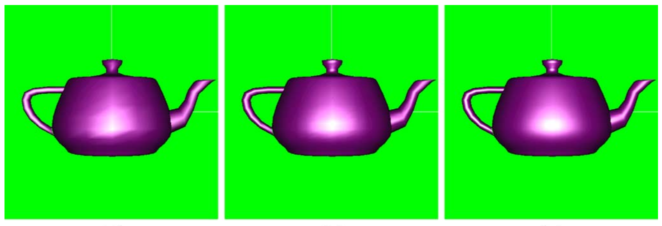


Fig. 4.3. Rendering result with different subdivision levels. (a) Level-0; (b) level-1; (c) level-2.

Compared with [29]–[32], the proposed geometry engine has better power-area efficiency with 518.8 $\mathrm{Kvertices}/(\mathrm{s} \bullet \mathrm{mW} \bullet \mathrm{mm}^2)$ for level-1 subdivision. Compared with work in [32], the proposed geometry engine can increase the power-area efficiency by 30.1%, 8%, and −14.5% with level-0, level-1, level-2 subdivision, respectively. Moreover, using the proposed subdivision algorithm, the proposed GE can provide near-Phong shading quality.

V. CONCLUSION AND FUTURE WORK

In this work, a low complexity subdivision algorithm and a power-area efficient GE are presented. Five low complexity techniques including the forward difference scheme, the edge function recovery scheme, the dual space subdivision scheme, the triangle filtering scheme, and the setup coefficient sharing scheme are adopted/proposed to reduce the computational complexity of the subdivision algorithm. 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 three different subdivision levels including level-0, level-1, and level-2 subdivision to achieve scalable and near-Phong shading quality. From the postlayout results, compared with the work in [32], the proposed geometry engine with level-0 and level-1 subdivision can improve the power-area efficiency by 30.1% and 8%, respectively. In the near future, the proposed design can be embedded into the system configuration in Fig. 1 of [17] to save more power and area cost for the geometry part of the standard graphics pipeline while non-fine mesh subdivision and near-Phong shading is desired. In this configuration, the surface/mesh geometry can be changed for smooth issue.

ACKNOWLEDGMENT

The authors would like to thank the associate editor and anonymous referees for their valuable suggestions to this paper, as well as Prof. S.-Y. Chien for providing the core area information of the chip in [32].

REFERENCES

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

[2] B.-S. Liang, Y.-C. Lee, W.-C. Yeh, and 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. Comput.*, pp. 623–628, Jun. 1971.

[4] D. Hearn and M. P. Baker, *Computer Graphics With OpenGL*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2004.

[5] A. T. Phong, "Illumination for computer generated pictures," *Commun. ACM*, vol. 18, no. 6, pp. 311–317, Jun. 1975.

[6] G. Bishop and D. M. Weimer, "Fast Phong shading," in *Proc. ACM SIGGRAPH Computer Graphics*, Aug. 1986, vol. 20, pp. 103–106.

[7] A. A. Mohamed *et al.*, "Hardware implementation of Phong shading using spherical interpolation," *Periodica Polytechnica Ser. El. Eng.*, vol. 44, no. 3–4, pp. 283–301, 2000.

[8] T. Barrera, A. Hast, and E. Bengtsson, "Faster shading by equal angle interpolation of vectors," *IEEE Trans. Vis. Comput. Graph.*, pp. 217–223, Mar. 2004.

[9] M. H. Lai, M. F. Yu, and S. G. Chen, "An efficient modified Phong shading algorithm & its low-complexity realization," in *Proc. IEEE ISCAS*, vol. 4, pp. 201–204.

[10] A. A. Mohamed, L. S. Kalos, G. Szijártó, T. Horváth, and T. Fóris, "Quadratic interpolation in hardware Phong shading and texture mapping," in *Proc. SCCG*, Apr. 2001, pp. 181–188.

[11] T. Barrera, A. Hast, and E. Bengtsson, "Fast near Phong-quality software shading," in *Proc. WSCG*, Jan. 2006, pp. 109–115.

[12] J. Pöpsel and C. Homung, "Highlight shading: Lighting and shading in a PHIGS+PEX environment," in *Proc. EUROGRAPHICS*, 1989, pp. 317–332.

[13] Y. Cho, U. Neumann, and J. Woo, "Improved specular highlights with adaptive shading," in *Proc. Int. Conf. Computer Graphics*, Jun. 1996, pp. 38–46.

[14] Y. Kamen and L. Shirman, "Triangle rendering using adaptive subdivision," *IEEE Comput. Graph. Appl.*, pp. 95–103, Mar./Apr. 1998.

[15] T. Y. Sheu and L. D. Van *et al.*, "Low complexity subdivision algorithm to approximate Phong shading using forward difference," in *Proc. IEEE ISCAS*, 2009, pp. 2373–2376.

[16] S. Bischoff, L. P. Kobbelt, and H. P. Seidel, "Toward hardware implementation of loop subdivision," in *Proc. SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, 2000, pp. 41–50.

[17] A. del Rio and M. Boo *et al.*, "Hardware implementation of the subdivision loop algorithm," in *Proc. IEEE EUROMICRO*, 2002, pp. 1–8.

[18] Y. Yasui and T. Kanai, "Surface quality assessment of subdivision surfaces on programmable graphics hardware," in *Proc. Shape Modeling Applications*, 2004, pp. 129–138.

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

[20] [Online]. Available: http://glprogramming.com/red/appendixf.html

[21] M. Olano and T. Greer, "Triangle scan conversion using 2D homogeneous coordinates," in *Proc. SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, Aug. 1997, pp. 89–95.

[22] K. Chung, C.-H. Yu, and L.-S. Kim, "Vertex cache of programmable geometry processor for mobile multimedia application," in *Proc. IEEE ISCAS*, 2006, pp. 1908–1911.

[23] C.-Y. Han, Y.-H. Im, and L.-S. Kim, "Geometry engine architecture with early backface culling hardware," *Comput. Graph.*, pp. 415–425, 2005.

[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, pp. 1767–1778, Aug. 2007.

[25] A. G. M. Strollo and D. De Caro, "Booth folding encoding for high performance squarer circuits," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 50, no. 5, pp. 250–254, May 2003.

[26] D. De Caro, N. Petra, and A. G. M. Strollo, "High performance special function unit for programmable 3-D graphics processors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 9, pp. 1968–1978, Sep. 2009.

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

[28] K. H. Abed and R. E. Siferd, "CMOS VLSI implementation of a low-power antilogarithmic converter," *IEEE Trans. Comput.*, vol. 52, no. 9, pp. 1221–1228, Sep. 2003.

[29] 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.

[30] C. H. Yu *et al.*, "An energy-efficient mobile vertex processor with multithread expanded VLIW architecture and vertex caches," *IEEE J. Solid-State Circuits*, vol. 42, no. 10, pp. 2257–2269, Oct. 2007.

[31] B.-G. Nam and H.-J. Yoo, "An embedded stream processor core based on logarithmic arithmetic for a low-power 3-D graphics SoC," *IEEE J. Solid-State Circuits*, vol. 44, no. 5, pp. 1554–1570, May 2009.

[32] 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, pp. 2025–2035, Sep. 2008.

Lan-Da Van (S'98–M'02) received the B.S. (Honors) and the M.S. degrees from the Tatung Institute of Technology, Taipei, Taiwan, R.O.C., in 1995 and 1997, respectively, and the Ph.D. degree from the National Taiwan University (NTU), Taipei, in 2001, all in electrical engineering.

From 2001 to 2006, he was an Associate Researcher at the National Chip Implementation Center (CIC), Hsinchu, Taiwan. In February 2006, he joined the faculty of the Department of Computer Science, National Chiao Tung University, Hsinchu, where he is currently an Assistant Professor. His research interests are in VLSI algorithms, architectures, and chips for digital signal processing and 3-D graphics systems. This includes the design of low-power/high-performance/cost-effective 3-D graphics systems, computer arithmetic, adaptive filters, and transform designs. He has published 45 journal and conference papers and held one U.S. and one Taiwan patent in these areas.

Dr. Van was a recipient of the Chunghwa Picture Tube (CPT) and Motorola fellowships in 1996 and 1997, respectively. He was an elected chairman of IEEE NTU Student Branch in 2000. In 2002, he received the IEEE award for outstanding leadership and service to the IEEE NTU Student Branch. In 2005, he was a recipient of the Best Poster Award at iNEER Conference for Engineering Education and Research (iCEER). From 2009 to 2010, he served as the officer of IEEE Taipei Section. He served as a reviewer for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: EXPRESS BRIEFS, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION SYSTEMS, the IEEE TRANSACTIONS ON SIGNAL PROCESSING, the IEEE TRANSACTIONS ON MULTIMEDIA, and the IEEE SIGNAL PROCESSING LETTERS.

Ten-Yao Sheu was born in Taipei, Taiwan, R.O.C. He received the B.S. and M.S. degrees in computer science from the National Pingtung University of Education (NPUE), Pingtung, Taiwan, and the National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 2006 and 2009, respectively.

His research interests include lower-power 3-D graphics system, computer arithmetic, digital signal processing, and VLSI design.