FRAGMENT REDUCTION ON MOBILE GPU WITH CONTENT ADAPTIVE SAMPLING

Chia-Yang Chang[1], *Yu-Jung Chen* [1],*Chia-Ming Chang*[2], *Shao-Yi Chien*[1]

Media IC and System Lab
[1]Graduate Institute of Electronics Engineering and Dept. of Electrical Engineering
[2]Graduate Institute of Networking and Multimedia
National Taiwan University
cychang@media.ee.ntu.edu.tw, sychien@cc.ee.ntu.edu.tw

ABSTRACT

Fragment shaders in a graphics pipeline are used to compute the color for each pixel, where lighting, texture loading, and other calculations are involved. The required computing power is proportional to the number of input fragments. In order to improve the power efficiency of mobile GPUs, a content adaptive sampling scheme is proposed to reduce the fragments. The proposed scheme is based on tile-based traversal. For each 4x4 tile, only parts of the fragments are sampled and rendered with the original shader program, and the values of other fragments are interpolated from these rendered fragments if the content checking condition can be passed. With this approach, the sampling patterns are decided adaptively, where more samples are employed for complex regions to avoid quality degradation. Experimental results show that about 30% – 50% fragments can be reduced where high image quality can be still maintained. The proposed scheme can be employed to reduce the power consumption and increase the frame rate for mobile GPUs.

Index Terms— Mobile Graphics, Graphics Rendering, Mobile Graphics Hardware, GPU

1. INTRODUCTION

Graphics Processing Unit (GPU) is a dedicated hardware that accelerates floating and vector computation for the real-time rendering pipeline defined in OpenGL and Direct3D. With the growth of computer-graphics based entertainment, real-time rendering system had been brought into mobile devices. Mobile GPU has been proposed [1] [2] to carry out and accelerate 3D applications, like games and some gorgeous User-Interface. However, there are some limitations in mobile devices because of the battery capacity and the small form factor for portable. Therefore, a low-power and low-cost mobile GPU is required.

The traditional rasterization-based graphics pipeline contains four main stages including application, geometry, rasterization, fragment. In application stage, primitive data(e.g. triangle, line) is sent from application. Vertices of primitive are processed in geometry stage and then primitives are placed into virtual 3D world coordinate. The raster stage continues to transform primitives into screen-space coordinate, so one fragment or more are generated for each pixel of screen in this stage and are sent to fragment stage. After the colors of fragments have been computed in fragment shader and blended by depth sort, they will be written into frame-buffer and show in screen panel. Based on this hardware pipeline as shown in Fig. 1, fragment shader computes the colors of **one or more fragments per pixel.** With resolution of mobile screen panel increment (QVGA: 240x320 to HD: 1280x720), the amount of fragments which should be processed raise rapidly. But, as the previous paragraph we describe, the battery capacity and the small form factor limit the number of shader cores that process vertex or fragment task. The number of shader cores in mobile GPU is far smaller than desktop GPU.

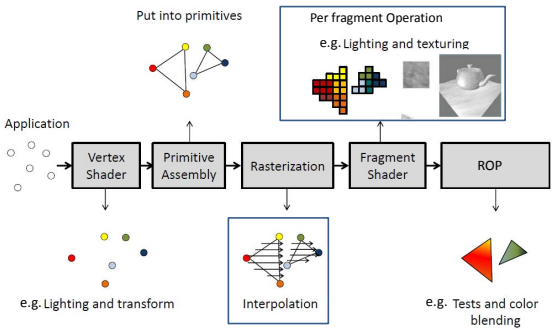


Fig. 1: The hardware pipeline of rasterization-based GPUs.

In this paper, a sampling scheme is proposed to reduce fragment shader workloads for mobile GPUs. Our basic strategy is finding out content adaptive sampling patterns and using interpolation instead of executing shader program for some fragments optionally. This paper is organized as follows. Section 2 introduces some related works. Next, in Section 3, the details of the proposed scheme are shown, and the objective and subjective experimental results are shown in Section 4. Finaly, a conclusion is provided in Section 5.

978-0-7695-4729-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ICMEW.2012.115

2. RELATED WORKS

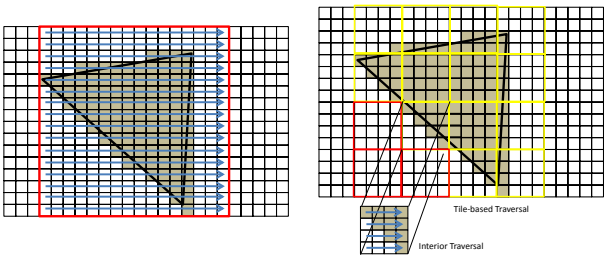
There are some methods which had been proposed to reduce the number of fragment shaders in temporal or spatial domain. In [3], their method considers geometry information to interpolate fragment colors. They compute and store the low resolution buffer **L** that contains color, depth, and normal vector in first pass. In second pass, only depth buffer and normal vector of each pixel should be computed and stored in high resolution buffer **H**. Finally they using joint bilateral filter to compute high resolution color. The contribution of each pixel in L depends not only on their distance in the image domain but also with the different between their respective scene depths and surface normal. They claim that this method could control the frame rate when each fragment computational time is too long, but needs two passes in geometry processing and huge buffer.

Corresponding to the spatial upsampling, temporal upsampling is another popular option. Nehab [4] and Sitthi-amorn [5] propose temporal coherence methods to eliminate redundant fragment shader between two frames. The main concept of their method is re-projecting primitive to the depth buffer of previous frame. If the difference between current depth data and buffer depth data is smaller than threshold they configured, the previous color would pass to current buffer. But this architecture needs other information from application layer.

Our method is inspired by multi-sampling anti-aliasing (MSAA), which reuses one computed fragment color to fill all the fragments in the same pixel . It is a balanced method compared to super-sampling anti-aliasing (SSAA). Although result of SSAA is more authentic, the amount of the fragment shader MSAA has to computed is much less than SSAA.

Some other algorithms like MSAA try to reduce the amount of fragments, and **decoupled sampling** [6] is one of them. Decouple sampling uses sample-sharing method to do not only anti-aliasing but motion blur and defocus blur. The common feature of the above-mentioned application is that they need lots of fragment samples at just one pixel, but those samples sometimes have similar colors. So they buffer the computed fragments results and copy those colors to similar fragments instead of really shading them.

Differ from decoupled sampling, we want to bring sample sharing into pixel level. If the amount of fragments is too huge to be processed by mobile GPU which computational resource is been limited, we will use the corner pixel of blocks to interpolate the others. Chang [7] is close to our method, they use one shading color to fill 2x2 or 4x4 block pixels. But it can't maintain final image quality when the complexity of image or texture is high.



(a) Bounding box traversal (b) Tile-based traversal

Fig. 2: This figure simply describes the difference between two rasterization schemes. The red blocks in (b) are only traversed in tile level, and the interior traversals in fragment-level are saved.

3. PROPOSED CONTENT ADAPTIVE SAMPLING SCHEME

Generally, rasterization stage is finding out a bounding box and examining all of fragments in the bounding box whether those fragments are inside the projected triangle. This scheme is called bounding box traversal as shown in Fig. 2a. There are some improved methods, and Sun [8] had proposed an efficient algorithm which combines of bounding box traversal and tiled traversal as shown in Fig. 2b. It is easy to see that tile-based traversal is more efficient than only bounding box traversal. In addition, tile-based traversal would assure that fragments processed are usually at a tile region. This feature could support our proposed content adaptive sampling scheme.

3.1. Concept overview

In this section, we give the high level overview of our modified graphics architecture as shown in Fig. 3. First, the **tile information** is gotten from interior traversal of tile-based traversal scheme, and the order of fragments outputting would be rescheduled. In fragment shader stage, we add some control logics and buffers to implement our approach — interpolate the fragment colors instead of really computing by fragment shader. See in Fig. 3, what this scheme should modify or add are rasterization and fragment shader stage. We would explain the modified parts and the added units in later sections.

3.2. Tile-based traversal rescheduling

Our method is also benefited from tile-based traversal. When it goes to **Interior traversal** as shown in Fig. 2, we could determine the adaptive sampling pattern and reschedule the sequence of output fragments. The content adaptive sampling patterns are chosen by four **line information**. It is shown in Fig. 4. If the all fragments of the line are valid, we set this line

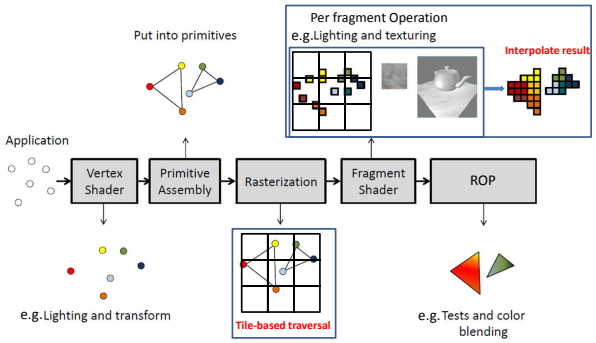


Fig. 3: Modified hardware pipeline of rasterization-based GPU with the proposed content adaptive sampling scheme.

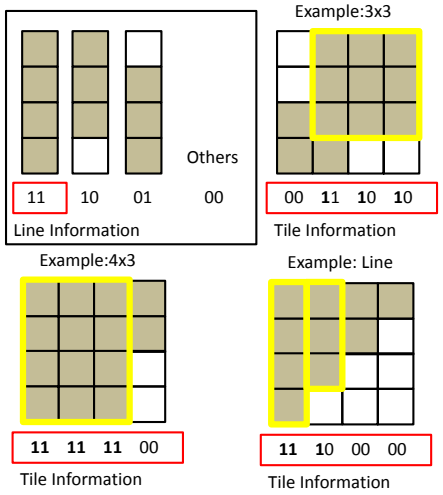


Fig. 4: Tile information for one tile can be recorded by only 8 bits, where 2 bits are used for each column. The yellow block denotes the "sampling pattern," which is the largest interior rectangle in each examples.

information as "11". And if only the top three fragments are valid, the line information would be "10". On the contrary, the line information is "01" if only the bottom three fragments are valid. If one line is not the above situation, line information of this line would be "00". Finally, tile information would gather four line information together. It could see that the tile information only needs 8 bits, and the content adaptive sampling pattern could be found by this tile information.

After determining adaptive sampling pattern, we first output the **tile information** which is used in next stage. And if the adaptive sampling pattern is 3x3, 4x3, 3x4, or 4x4 block, the corner fragments would be sent in prior to the other fragments. If one tile has only line-type pattern, it would send side fragment of lines first. Fragments which are not inside adaptive sampling pattern are sent last. The all sending priority is shown in Fig. 5.

We would discuss in the next section that our content

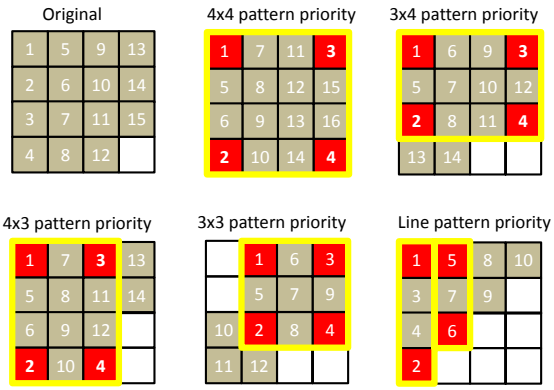


Fig. 5: The priority of all sampling patterns. In each sampling pattern denoted by yellow block, the fragment operations for red fragments would be executed first, and the operations for other fragments may be skipped and replaced by interpolation operations to save the computation.

adaptive pattern sometime cause serious artifacts if the complexity of real colors is too high. Beside the rescheduling mechanism, a **checking point** is added in our pattern, which is located at the center of the block. Bug, as mentioned in [8], the varyings of fragments can be interpolated using their coordinate and plan equation. As follow:

$$\begin{aligned} s(x,y) &= -\frac{a}{c}x + \frac{b}{c}y - \frac{d}{c} \\ &= Ax + By + C \end{aligned} \quad (1)$$

If the raster are in the direction of x or y, we can find that:

$$s(x+1,y) = s(x,y) + A \quad (2)$$

$$s(x,y+1) = s(x,y) + B \quad (3)$$

Because **checking point** is located the center of yellow block as shown in Fig. 5, the varyings of which would be interpolated in following equation:

$$s(x+0.5,y+0.5) = s(x,y) + A/2 + B/2 \quad (4)$$

It would use floating point division in raster engine. So we want to set the check point as the safety mode, which could be configured by programmer. If the object is important in the scene, it could use the checking point mechanism to avoid artifacts. We would discuss this in the last section.

3.3. Condition checking and interpolate

In fragment shader stage, we first get the **tile information**. It is used to check whether the fragment is inside the sampling pattern, and the interpolate coefficients are modified by this information. And the corner fragments of block pattern or side fragments of line pattern would be sent into shader core to compute, we call these fragments as reference fragments

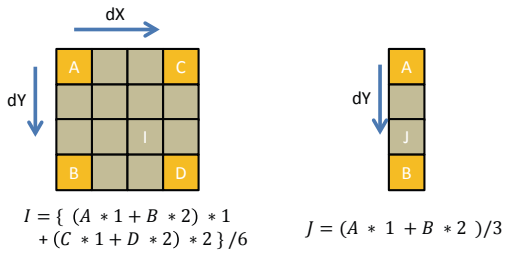


Fig. 6: Bilinear or linear interpolation in a sampling pattern.

which could be seen in Fig. 5 as red colors. After **reference fragments** were computed, we buffer their RGBA color data in an extra **tile buffer**. Tile buffer capacity we need is 16 bytes, which contain only four reference fragments computed results in practice.

After computing and buffering the colors of reference fragments. Our strategy is using linear or bilinear interpolation to approximate the colors of the other fragments. Sometimes the true colors are interpolated by vertex attributes of primitive and added some lighting terms. In this situation, our method could estimate the color accurately. Even if lighting equation has the complexity of quadratic or higher, the error could still be ignored in small region. But the fragment colors are often sampled from texture image, and those could produce some uncertain errors. We could compute the color distance between two reference fragments as:

$$\begin{aligned} ColorDistance(fragA, fragB) &= (R_{fragA} - R_{fragB})^2 \\ &+ (G_{fragA} - G_{fragB})^2 \\ &+ (B_{fragA} - B_{fragB})^2 \end{aligned} \quad (5)$$

And set the threshold to avoid interpolating if the maximum of color distances is larger than threshold, this examination could ensure that the interpolating fragment colors would not cause serious artifact.

Finally, if this block passes the condition checking. The other fragments at block pattern would be bilinear interpolated by [equation.6]from reference fragment as shown in Fig. 6 left. And the fragments at line pattern are only linear interpolated as shown in Fig. 6 right. We would cull these fragments and send the interpolate results to save the workload of shader cores. And even if the test is failed, we send the other fragments into fragment shader. It just causes a little timing delay if the interpolation does not happen. And the timing delay could be hidden by pipeline in practice.

$$\begin{aligned} Colors = \{ & [A * (3 - dY) + B * dY] * (3 - dX) \\ & + [C * (3 - dY) + D * dY] * dX \} / 6 \end{aligned} \quad (6)$$

3.4. Implementation

The proposed scheme is implemented in a graphics pipeline with the control flow shown in Fig. 7, where the checking condition is shown as follows.

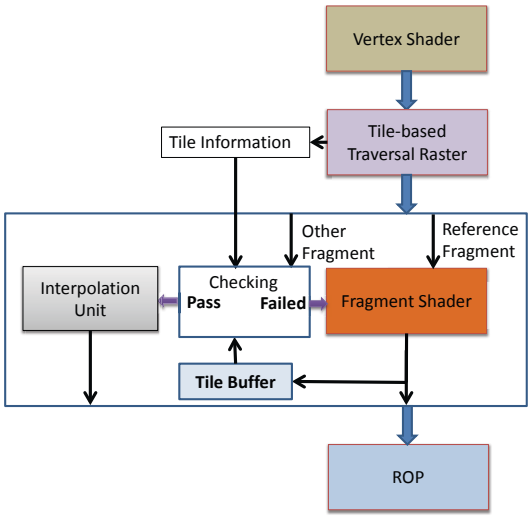


Fig. 7: The detailed control flow diagram of the proposed scheme.

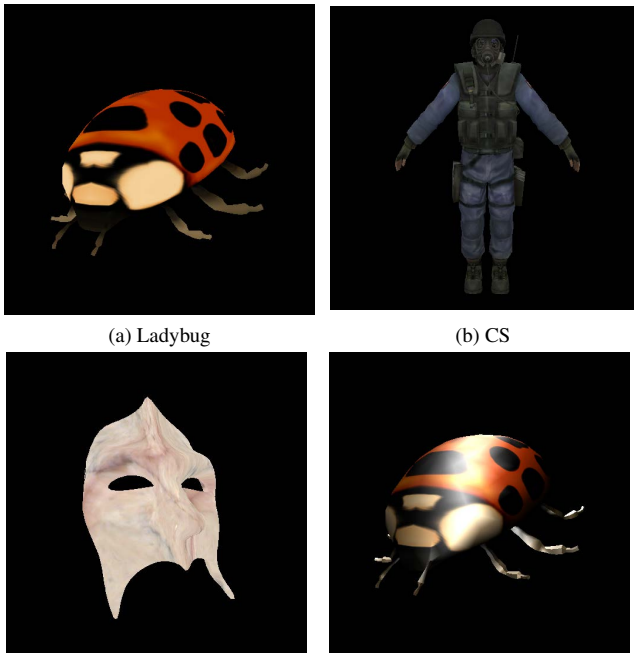
if $Tile\ Information \neq 8'b00000000$ **then**
  Compute $all\ ColorDistance.$
  Find $the\ Maximum\ ColorDistance.$
  if $Maximum\ ColorDistance \geq Threshold$ **then**
    $Shader\ core \leftarrow Fragment.$
    $ROP \leftarrow Computed\ result.$
  else
    $Interpolate.$
    $ROP \leftarrow Interpolation\ result.$
  end if
end if

Note that the threshold could be determined by users according to different usage conditions and application requirements. In traditional graphics pipeline, fragments could be parallel processing by shader cores. In contrast, our scheme would send all fragments of one tile to a shader core. And shader cores would process each tile in parallel.

4. EXPERIMENTAL RESULTS

We use PSNR to indicate the error between our result and the ground truth. Our some examples are shown in Fig. 8. And in the different threshold, PSNR and the saved rate of fragment shaders are statistics in following Fig. 9.

Some detailed comparisons are shown in Fig. 10 and Fig. 11, the left images are ground truth and both the right images are at threshold 8000. Content adaptive sampling patterns distribution is in Fig. 12, where the red region represents 4x4 pattern, where saves 75% fragments. The green region is using 4x3,3x4,3x3 pattern, where saves 55% – 66% fragments. In blue regions, only 33% – 50% fragment are saved by using line pattern. There are no fragments saved in the white region. It could be seen that if the triangles of



(a) Ladybug (b) CS

(c) Mask (d) Ladybug with lighting

Fig. 8: Test cases.

model(e.g. CS, Mask) are smaller, the saved fragment shader rate is worse because it usually use small patterns.

As we mentioned, if there are high frequent texture, some serious artifact would occur as shown in Fig. 13a. So checking points could be used to check this situation in the safety mode. After executing the checking fragment, we could obtain the true colors of check points. If we find that the difference of the interpolate result by reference fragment and the true checking fragment result are too large. We would not interpolate the other fragments. This scheme is effective to improve the high frequency texture as shown in Fig. 13b.

5. CONCLUSION

In this paper, a content adaptive sampling scheme is proposed to reduce the workloads of GPUs by replacing shader program execution with interpolation for parts of the fragments. The experimental results show that this scheme can save about 30%–50% fragments while the image quality is maintained at 45dB in PSNR. For scenes with high-frequency textures, a safety mode is also proposed with checking point method to determine the sampling approaches for different contents. With this scheme, it is expected that the power efficiency of mobile GPUs can be further improved. That is, the rendering quality can be adjusted according to different usage conditions and application requirements.

As shown in this paper, additional buffers, control logics, and interpolate units are required to implement the proposed scheme. The hardware implementation of the whole system will be included in our future work to verify the power reduction ratio and hardware cost overhead.

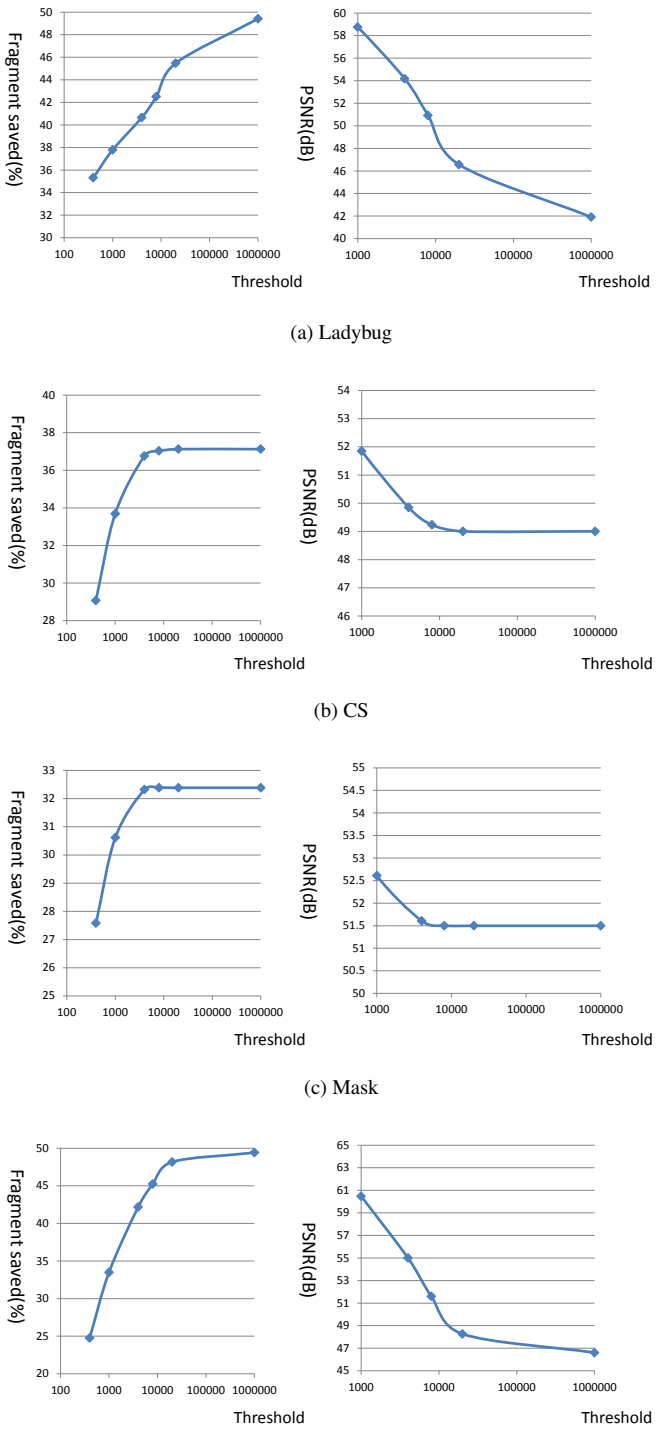
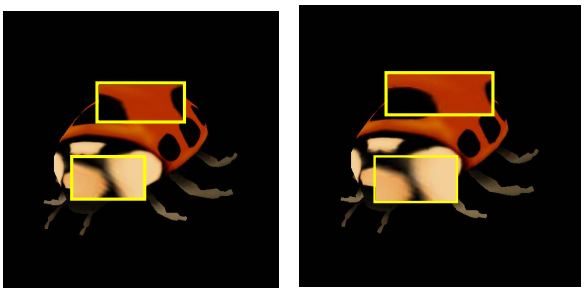
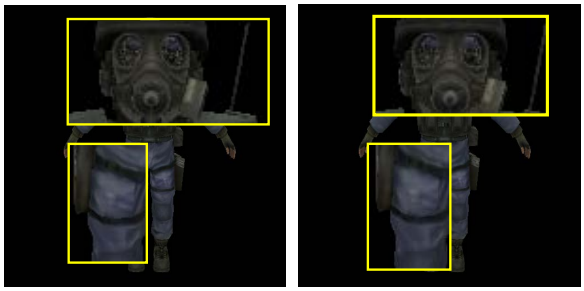


Fig. 9: Result statistics. The left part shows the fragment saving ratio under different threshold settings; the right part shows the corresponding image quality in PSNR.



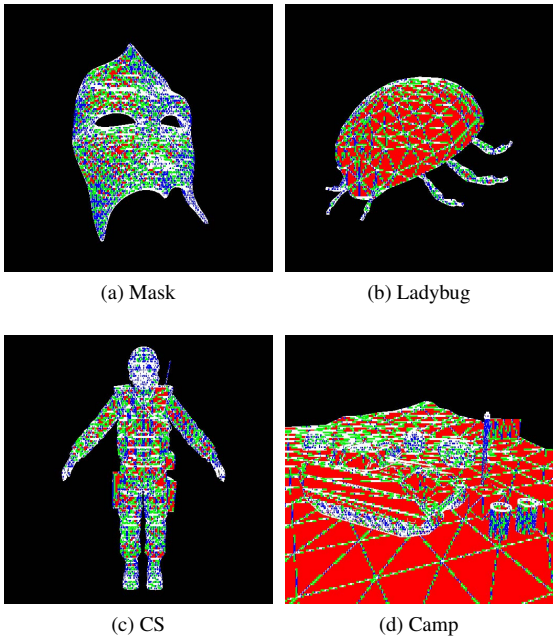
(a) Ground truth (b) Threshold = 8000

Fig. 10: Detailed subjective comparison with Ladybug model.



(a) Ground truth. (b) Threshold = 8000

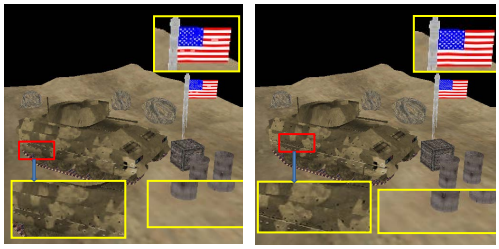
Fig. 11: Detailed subjective comparison with CS model.



(a) Mask (b) Ladybug

(c) CS (d) Camp

Fig. 12: Distribution of different sampling patterns.



(a) Without checking point (b) With checking point

Fig. 13: Test scene with high-frequency textured objects.

6. REFERENCES

[1] M. Kameyama, Y. Kato, H. Fujimoto, H. Negishi, Y. Kodama, Y. Inoue, and H. Kawai, "3D graphics LSI core for mobile phone "Z3D"," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware(HWWS)*, 2003, vol. 2, pp. 60–67.

[2] T. Akenine-Möller and J. Ström, "Graphics for the masses: a hardware rasterization architecture for mobile phones," in *Proceeding of ACM SIGGRAPH*, 2003, SIGGRAPH '03, pp. 801–808.

[3] L. Yang, P. V. Sander, and J. Lawrence, "Geometry-aware framebuffer level of detail," *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering)*, vol. 27, no. 4, pp. 1183–1188, 2008.

[4] D. Nehab, P. V. Sander, J. Lawrence, N. Tatarchuk, and J. R. Isidoro, "Accelerating real-time shading with reverse reprojection caching," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, 2007, GH '07, pp. 25–35.

[5] P. Sitthi-amorn, J. Lawrence, L. Yang, P. V. Sander, D. Nehab, and J. Xi, "Automated reprojection-based pixel shader optimization," in *Proceeding of ACM SIGGRAPH Asia*, 2008, SIGGRAPH Asia '08, pp. 127:1–127:11.

[6] J. Ragan-Kelley, J. Lehtinen, J. Chen, M. Doggett, and F. Durand, "Decoupled sampling for graphics pipelines," *ACM Transactions on Graphics(TOG)*, vol. 30, no. 3, pp. 17:1–17:17, May 2011.

[7] C.-M. Chang, Y.-J. Chen, Y.-C. Lu, C.-Y. Lin, L.-G. Chen, and S.-Y. Chien, "A 172.6mW 43.8GFLOPS energy-efficient scalable eight-core 3D graphics processor for mobile multimedia applications," in *Proceeding of IEEE Asian Solid State Circuits Conference (A-SSCC)*, 2011, pp. 405–408.

[8] C.-H. Sun, Y.-M. Tsao, K.-H. Lok, and S.-Y. Chien, "Universal rasterizer with edge equations and tile-scan triangle traversal algorithm for graphics processing units," in *Proceeding of IEEE International Conference on Multimedia and Expo(ICME)*, 2009, pp. 1358–1361.