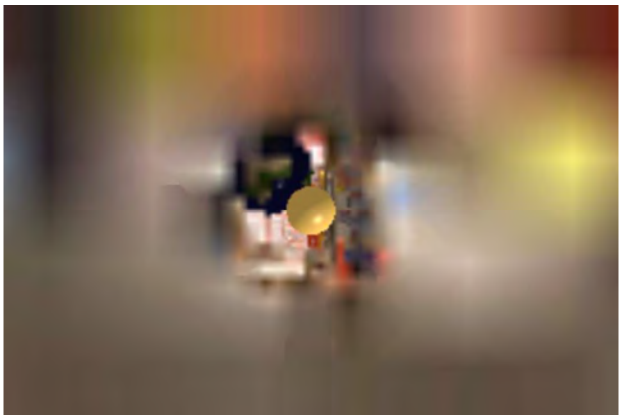
An Acceleration Method for Color Computation Using Coarse Pixel Shading

Takashi Imagire

Tokyo Polytechnic Univ., 2-9-5 Honcho, Nakano, Tokyo 164-8678, Japan



ABSTRACT

Coarse pixel shading is a method of reducing the color computation load in scenes by partially changing the size of pixels. Although coarse piece plotting is one of the speeding up techniques, the current GPU architecture has no standard function of partially changing the size of output pixels, nor has it been incorporated into real-time applications. In our method, by selecting a fragment to be processed by the fragment shader, which is a processing for each pixel, before inputting it to the fragment shader, color computation by the coarse pixel shading completed by a graphics pipeline not causing unnecessary color computation Is realized.

1. INTRODUCTION and RELATED WORKS

The GPU consists of shader units and fixed functions. Regarding the fixed function, it is possible to configure predetermined functions, and restrict the processing which can be realized utilizing. One thing that cannot be realized is partial resolution change in an image. In rendering with VR, the degree of detail of the peripheral vision part is unnecessary, so a method to speed up the color computation has been proposed [1]. However, since the current GPU can only render an object with a constant pixel density, it is necessary to render images with multiple resolutions and combine them in generating an multi resolution image, as well as computing the same fragments over and over again.

Vaidyanathan et al. [2] proposed the Coarse Pixel Shading. For fragments that perform lighting calculations, multiple pixels are grouped together as a square fragment of a large size of 2 without associating fragments and pixels one to one It is a method to speed up the rendering by calculating. Sathe and Janczak [3] adapted Deferred Coarse Pixel Shading as a method that can be implemented with game engines that are standard in current commercial games, adapting coarse fragment drawing to differed rendering. However, the Deferred CPS is not closed in the rendering pipeline of the GPU, it uses a compute shader to implement CPS that is used for perform General-Purpose GPU computation, and the purpose as a rendering method for load reduction It is not suitable.

In this paper, we propose a method to realize CPS only by rendering pipeline without compute pipeline, using early culling function with mipmap structure as well as texture mipmap structure

2. OUR METHOD

Our method extends the deferred shading [4]. Figure 2 shows the rendering pipeline. In standard deferred shading, after generating G-buffers, color computation is performed using the information of the G-buffers. In our method, after generation of the G-buffers, a hierarchical stencil buffer (stencil pyramid) is created, and by performing the early stencil test using the stencil pyramid, rendering only the necessary fragments to the render targets of the mipmap structure is realized. Finally, one image is generated by combining the results rendered as mipmap textures.

2.1 G-buffer generation

First, information for color computation are outputted to G-buffers. Normally, albedo, specular reflection rate, roughness, normal map, etc. are recorded in the G-buffers, in our method, the degree of detail of coarse pixel shading (LOD buffer) is added to the G-buffers. In the LOD buffer, record the level L of the fragment for calculating each pixel. A fragment of level L is a square with the number of pixels of one side is 2L. In this paper, we use the phong reflection model for the specular reflection as the color computation with applying the LOD buffer, our method can be applied to other color computation models.

2.2 Stencil Pyramid construction

Next, a hierarchical stencil buffer is constructed to perform early culling. Prepare a stencil buffer as many as the maximum number of detail levels. The magnitude of each stenciff buffer is made smaller by the power of 2 of detail level. Copy the LOD buffer to each stencil buffer. For this time, we used the OpenGL GL_ARB_shader_stencil_export extension to write the LOD buffer directly into the stencil buffer. As an implementation of the proposed method, it is possible to use a depth buffer instead of a stencil buffer,Because it is difficult to compare floating point number matching and floating point number and fixed decimal number conversion in the depth buffer processing, it is difficult to extract matching fragments, so in this method the stencil buffer was used for the premature ring.

2.3 Color computation

After that, perform color computation using G-buffers. Prepare a mipmaped render target with a size of power of 2 up to the maximum level of detail level like the stencil buffer and perform color computation with early stencil testing for each mipmap level. In the stencil test, the detail level of the render target to be compute is compared with the value of the stencil buffer, and only fragments with the same value are drawn.

2.4 Final Image construction

Finally, the mipmap texture of the computed result is gathered for each level to generate the final image. For each pixel, reading the texture at the mipmap level of the value of the LOD buffer and the image of the standard CPS is generated. As the results of this standard CPS stand out for block artifacts, we reduce artifacts by our custom interpolation.

3. RESULTS

For verification of this method, we used a notebook PC (Razer Blade Stealth) equipped with Radeon RX 560 as an external GPU. We use OpenGL for 3D API and GLFW 3.2.1 for framework. In the proposed method, the GL_ARB_shader_stencil_export extension is necessary. In environments where this extension is not supported, such as NVIDIA GPU, this demo does not work. Figure 1 shows the results of the proposed method. Color computation is performed with 9 Levels of Details from 1 × 1 to 256 × 256. In the demo the number of light sources set to 200, in a normal color computation with one pixel as one fragment, it is executed at 36.4 FPS, the result of our method is 60.1 FPS. The execution speed of 65% Improvement was obtained.

4. FUTURE WORKS

In our method, the final color is derived by performing linear combination of the color of the neighborhood fragments, not the sampling of the nearest neighbor pixels, but the discontinuity is observed in the bounaries where the level of detail changes. Since this problem is noticeable as the level of detail decreases, it is necessary to suppress the upper limit of the detail level or to improve the sampling method.

As an example of this case, we applied the algorithm to generate 2D planar images, but we want to realize the result of improving the user experience by applying it to the use of CSP as VR sickness countermeasure [5].

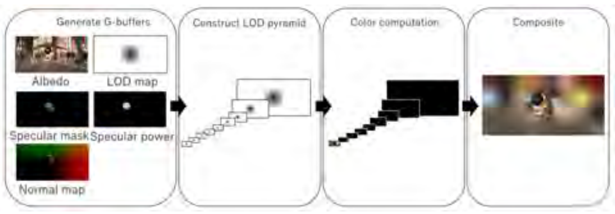


Fig. 2 Rendering Process

5. REFERENCES

[1] B. Guenter, M. Finch and S. Drucker, D. Tan and J. Snyder, "Foveated 3D Graphics", ACM Trans. Graph. 31(6), pp.164:1-164:10 (2012).

[2] K. Vaidyanathan, M Salvi, R. Toth, T. Foley, T. Akenine -Moller, J. Nilsson, J. Munkberg, J. Hasselgren, M. Sugihara, P. Clarberg, T. Janczak and A. Lefohn, "Coarse Pixel Shading", Proceedings of High Performance Graphics 10 pp.9-18 (2014).

[3] R. P. Sathe and T. Janczak, "Deferred Coarse Pixel Shading", GPU Pro 7, CRC Press, pp.145-153 (2016).

[4] M. Valient, "Deferred rendering in killzone 2", in Develop Conference, http://www.guerrilla-games.com/ publications/dr_kz2_rsx_dev07.pdf (2007).

[5] K. Tachibana and T. Imagire, “A Fast Rendering Technique for VR Sickness Prevention using Coarse Pixel Shading”, in I3D 2018 Posters (2018).