Non photorealistic rendering with tile based deferred shading on WebGL

Sijie Tian
University of Pennsylvania

Yuqin Shao
University of Pennsylvania

Figure 1. WebGL NPR Tile-based Deferred Shading

Abstract

Deferred shading, a screen-space shading technique, which enables fast and complex light resource management, has been more and more widely used in game industry. However, it seems this technique has much less use on WebGL. In this project, we are trying to implement an advanced deferred shader on WebGL as well as to achieve some non-photorealistic rendering effects.

1. Introduction

As we all know, to solve the problems in forward shading, the deferred shading passes and stores all the positions, normals, colors, depth and others information in the Geometry buffer.Then, we can draw our scene on this 2D textures. In this way, deferred shading avoids calculating the light on each face of the object and can culling lights efficiency using screen space bounding box. However, there is still a problem in deferred shading is the overhead of bandwidth. In deferred shading, we draw every light

 ISSN 2331-7418

by reading the information in G-buffer. To reduce the bandwidth usage in deferred shading, we use an approach called tile based deferred shading. Tile based is just like the name is, it separate the screen in small tile e.g. 32x32, 16x16, and groups lights in each tile. The NPR effects we are trying to achieve in this project is the Chinese Painting Effects. Generally, what we do includes two steps which are silhouette and stroke simulation as well as interior ink shading.

1.1. Related Work

There are bunch of previous works related to deferred shader and Chinese painting NPR we could refer to.

Tile-based deferred shading has been described in several talks and paper[Olsson and Assarsson 2011][Olsson et al. 2012]. It is very suitable for multiply light scene. All of these deferred shading are implemented on Cuda or DX11, their performance shows a great improvement on tile based deferred shading compare to originally method.

Silhouette and stroke simulation There are two classic solutions for this topic in earlier works. The first one is the image based silhouette extraction which works as frame-buffer process, by tracking discontinuities of shape features. And the other one is object based method which detects the feature edges in object space and renders lines as mesh separately. We mainly refer to[Yeh 2002] which is a object based stroke solution.

Ink shading effects There are also two main categories of methods to solve this issue. The first one is physically-based Image-Based algorithm. This method could give create Chinese ink painting style easily without using any strokes. However, it is quite slow. The other one is non-physically based method, which could get a remarkable speed-up with ignorable artifact. Since we are doing WebGL project which has a very high requirement for speed, thus we use the non-physically-based method here for the interior shading. However, we will still have to use the object based silhouette extraction because of the un-precise problem of image-based one. We mainly refer to[Lixing Do 2012], which gives a non-physically based solution.

2. Tile based deferred shader

To implement this on WebGL, first we need several WebGL extensions, including OES_texture_float,OES_texture_float_linear , WEBGL_depth_texture and WEBGL _draw_buffers. To use these extensions, you need to turn on the on Enable WebGL Draft Extensions your browser.

In order to group lights in each tile, we first need to know the screen space bounding box of each light. In our project, we consider each light as a sphere and objects outside the light radius are not affected by this light. So we are able to cull light base

on theirs position and radius.. To compute the boundary of each light, we first use camera view direction and view matrix to calculate the up and left vector of camera. Then, using the light position, radius and these two vectors, we can easily get the left and top position of the boundary of the light in world space. Lastly, we transform these values to screen space and get the boundary of the light.

After having the bounding box of each light, we can insert our light to the tile. For each light, we know the top, left, bottom and right position in the screen space, so we can divide by the size of tile to know which area of tiles are affected by this light. Therefore, each tile stores all the affected lights' position, radius and color. However, to facilitate and easily look up in the shader, our program using some other data structures to store this information. Firstly, Instead passing array, we pass array as texture to shader. Secondly, we have three arrays to store suitable information for shader. First array is the light list, which stores all the lights' position, radius and color. The second array is the light index list which stores the light index for each tile. The third one is the light tile array which stores the offset in light index array and number of lights in each tile. After constructing these three array, we pass it to the shader as 2D texture. Then we can access the light information in the shader easily.

Figure 2. This is a rendered result with 300 lights

Figure 3. This is a rendered result with 500 lights

3. Chinese Painting NPR

3.1. Silhouette Extraction

A silhouette edge is an edge adjacent to one front-facing and one back-facing polygon. A polygon is defined as front-facing if the dot product of its outward normal and a vector from the camera position to a point on the polygon is negative. Otherwise the polygon is back-facing.

3.1.1. Silhouette Culling

Since we are drawing silhouette as separate mesh. It is impossible for us to use WebGL's depth buffer for back silhouette culling. Thus, we will have to write it by ourselves. Based on [Yeh 2002] method, they first render the depth buffer of the original triangle mesh into a framebuffer. And based on the exact pixel location of both endpoints in silhouette edge, they compare their depth value to the depth buffer on current rendering frame. Finally the eliminate edges that didn âĂŹt pass the depth test. And get all of those edges rendered out again. However,[Yeh 2002] method is written in opengl and they used the readpixels in opengl to get all of those depth values back from framebuffer. Unfortunately, this method is not able to receive FLOAT type value in WebGL. Thus, we finally gave up this method and found another solution to visually solve the problem. In our method, we need three framebuffer textures for the final result. First two textures stroe values of silhouette edges color pass(without culling) and silhouette edges' depth buffer values respectively. The third pass we store the depth buffer value of the original triangle mesh. In the final post fragment shader, before we set the final fragment color, we first compare the depth value of the edges' depth texture and the original meshes' depth texture and only render out the color that pass the depth buffer.

3.1.2. Stroke rendering

In section 3.2, we get the silhouette edges of mesh. In stroke simulation step, we use the texture rendered out from section 3.2 to get the final stroke effects. Firstly, since the silhouette edge is rendered with DRAW_LINE in WebGL. Thus,it is only one pixel width. What we do in the first step is to make the silhouette edges thicker. Basically, in the fragment shader, we make pixels that around silhouette edge to be the same color as of silhouette edge. And in the final step, we use Gaussian blur method to blur the current stroke and finally make them blend with the interior color. We are still trying to find a better way to simulate the stroke with different thickness based on edge connection relationships. It is not easy to balance a good result and a perfect performance for WebGL so far.

Figure 4. Stroke Only Rendering

3.1.3. Interior shading

Interior shading could be a separate step from the stroke rendering.[Lixing Do 2012] This step only need us to work on shaders instead of creating complex data structure with javascript. Firstly, based on the diffuse factor calculated by the dot product of normal and light vector, we categorizes object color into four main type. For example, if diffuse factor is within the range of [0 , 0.3] we make it totally black, and if it is within the range of [0.7,1.0], we make it totally white. Second step is to apply a spattering filter to the image we get from first step. We randomly choose a pixel color in current pixel's neighboring area(defined by a radius parameter), and replace current pixel âĂŹs color with the new color. Finally, we use a median filter to suppress the noises brought from the spattering step. Additionally, in order to achieve more realistic Chinese painting effect, we also use a gamma correction formula to increase the contrast of our image.

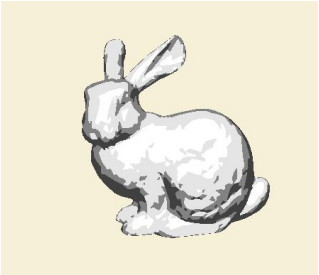

Figure 5. Quantization Step

Figure 6. Spattering Filter Rendered Result

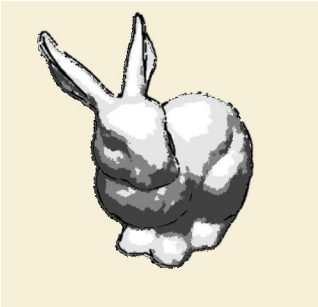

Figure 7. Final Results with Stroke Added in

4. More Result

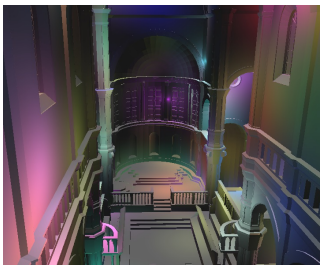

Figure 8. 300 lights rendered result

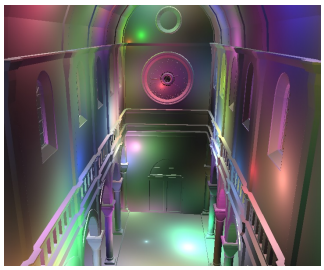

Figure 9. 500 lights rendered result

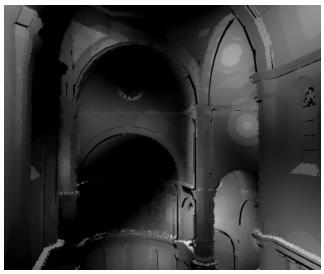

Figure 10. Chinese Painting NPR result

References

LIXING DO, SHUFANG LU, X. J., 2012. Real-time image-based chinese ink painting rendering. 1, 4

OLSSON, O., AND ASSARSSON, U. 2011. Tiled shading. *Journal of Graphics, GPU, and Game Tools 15*, 4, 235–251. 1

OLSSON, O., BILLETER, M., AND ASSARSSON, U. 2012. Clustered deferred and forward shading. In *HPG '12: Proceedings of the Conference on High Performance Graphics 2012*. 1

YEH, J.-W. 2002. *Automatic Rendering of 3D Animal Models in Chinese Painting Style*. Master's thesis, National Taiwan University. 1, 3