Deferred Shading and Rendering

Edward Cao
100697845
edward.cao@ontariotechu.net

Hunter Chu
100701653
hunter.chu@ontariotechu.net

Dhruva Guruprasad
100700271
dhruva.guruprasad@ontariotechu.net

Jiminy Cao
100701335
jiminy.cao @ontariotechu.net

Tavis East
100702011
tavis.east@ontariotechu.net

Mustafa Siddiqi
100703421
mustafa.siddiqi@ontariotechu.net

Gary Liu
10070683
gary.liu@ontariotechu.net

***Abstract*— This paper demonstrates the effects of deferred shading and post processing effects in OpenGL**

***Keywords*— OpenGL, Lighting, Shaders, Deferred Rendering, Post Processing Effects**

I. INTRODUCTION

Lights and lighting effects bring great value to a game and are extremely customisable which allows them to be adjusted to accommodate any game.

II. G-BUFFER

The G-Buffer or geometry buffer is one of the required aspects of deferred rendering. The G-Buffer is made to perform the geometry pass which stores all the information of the geometry in the scene to be passed to the shaders farther down the pipeline when lighting calculations need to be done

III. RENDERING & LIGHTING

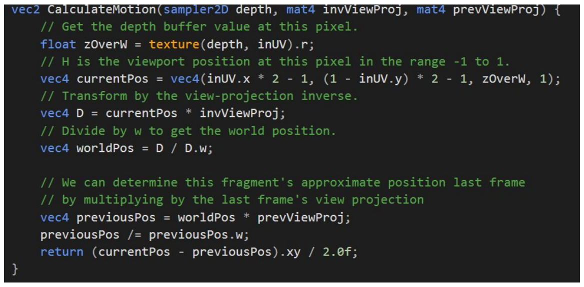
Deferred lighting uses light volumes which represents the area of effect a light has in the scene. Properties on the light such as radius are used to determine what objects a light actually interacts with and needs to perform lighting calculations on. Since the lights are no longer trying to perform calculations on meshes outside of it's area of effect, it allows considerably more lights to be placed in the scene without a heavy reduction in performance.

IV. POST PROCESSING EFFECTS

The two post processing effects that will be discussed in this report are: Motion blur and Toon shading.

A. *Motion Blur*

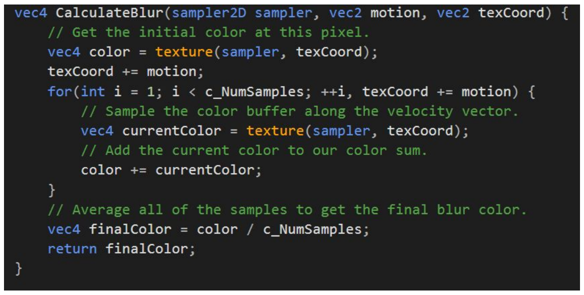
Motion blur is a post processing effect that applies a blur effect to the output image with the intensity and direction of the blur being dependent on the movement of the camera.



```
vec2 CalculateMotion(sampler2D depth, mat4 invViewProj, mat4 prevViewProj) {
    // Get the depth buffer value at this pixel.
    float zOverW = texture(depth, inUV).r;
    // H is the viewport position at this pixel in the range -1 to 1.
    vec4 currentPos = vec4(inUV.x * 2 - 1, (1 - inUV.y) * 2 - 1, zOverW, 1);
    // Transform by the view-projection inverse.
    vec4 D = currentPos * invViewProj;
    // Divide by w to get the world position.
    vec4 worldPos = D / D.w;

    // We can determine this fragment's approximate position last frame
    // by multiplying by the last frame's view projection
    vec4 previousPos = worldPos * prevViewProj;
    previousPos /= previousPos.w;
    return (currentPos - previousPos).xy / 2.0f;
}
```

Figure 1: The code above calculates the change in motion of the current frame by comparing the change in view projections.

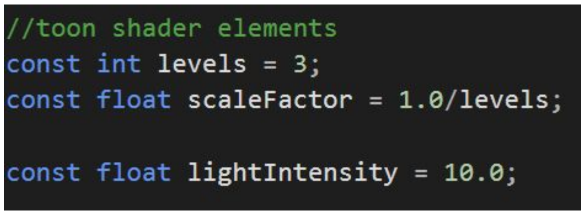


```
vec4 CalculateBlur(sampler2D sampler, vec2 motion, vec2 texCoord) {
    // Get the initial color at this pixel.
    vec4 color = texture(sampler, texCoord);
    texCoord += motion;
    for(int i = 1; i < c_NumSamples; ++i, texCoord += motion) {
        // Sample the color buffer along the velocity vector.
        vec4 currentColor = texture(sampler, texCoord);
        // Add the current color to our color sum.
        color += currentColor;
    }
    // Average all of the samples to get the final blur color.
    vec4 finalColor = color / c_NumSamples;
    return finalColor;
}
```

Figure 2: The motion vector that is returned from the previous function is then brought into the CalculateBlur function which takes the vector and samples pixels in the direction of the vector. The sampled pixels are then averaged out to create the motion blur

B. *Toon Shading*

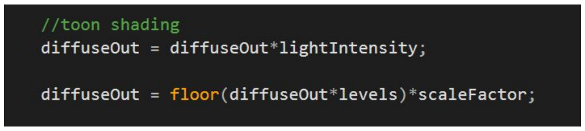
Toon shading is a post processing effect that aims to imitate a cartoon-like look using bands of lighting with sharp drop offs rather than having every pixel be individually shaded. This is done by taking the results of the lighting calculations and sending those into a shade gradient, which then compares the numbers against a table with values that dictate the value of the shade gradient that pixel uses.



```
//toon shader elements
const int levels = 3;
const float scaleFactor = 1.0/levels;

const float lightIntensity = 10.0;
```

Figure 3: In the code above, the variable levels sets the number shading bands the shader will use.



```
    //toon shading
    diffuseOut = diffuseOut*lightIntensity;

    diffuseOut = floor(diffuseOut*levels)*scaleFactor;
```

Figure 4: The code above calculates the output diffuse for the shading bands by multiplying the intensity of the light with the diffuse component. The result is then placed into a shading band and outputted.

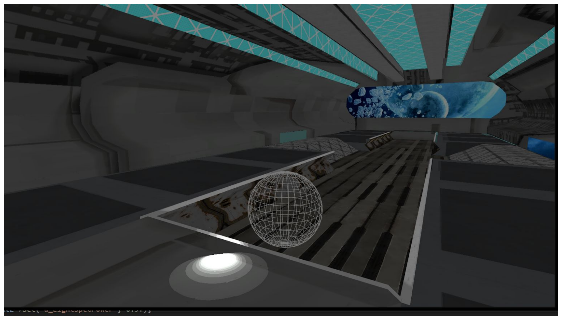


Figure 5: The light's influence on the scene creates the toon shading effect, where three shading bands can be seen as declared in the code