Virtual and Augmented Reality: Final Project

Yang Gao
N15410269
yg2754@nyu.edu

Abstract

Achieving Phong shading with interactions to view a 3D object, implemented by Unity and C.

1. Introduction

Shading is a powerful tool to use in the graphics design process and mimic reality. There are many shading techniques that could be implemented to achieve this. Shading technique like lambertian shading is view independent where the color of a surface does not depend on the direction from which the user look. On the contrary, Phong shading does. Phong shading is proposed by Phong and later updated by Blinn to the form most commonly used today. The idea is to produce reflection that is at its brightest when viewing ray and light ray are symmetrically positioned across the surface normal, which is when mirror reflection would occur. The reflection then decreases smoothly as the vectors move away form a mirror configuration. Many real surfaces would show certain degree of shininess or highlights or we would call specular reflections, that appear to move around as the viewpoint change. This project would follow the guidelines on unity 3d manual, to implement a Phong shader and item rotation.

2. Reading Summary of Related Articles/Papers

In order to achieve a Phong shader [2], a light source and a camera is needed. Since this is a Virtual Reality project, instead of a camera, we would utilize gaze as the camera. Therefore, we can have the ray tracing from the eye to the element. The item in this project would potentially utilize the scene in the HelloCardboard, provided by google. Once the light source, camera, and item has been determined, Phong shader can be implemented by using the formula shown later. There are three main components in the Phong shading, sample image is shown in figure 1. Ambient component is the reflection of ambient light source form/in all directions. Diffuse component is the diffuse reflection of the light source in all directions. Specular component is the "glossy" reflection creating specular highlights.

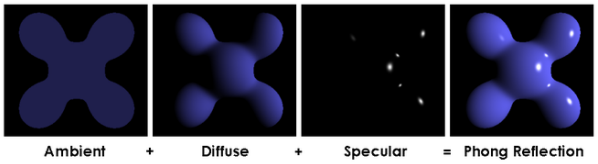

Figure 1. Phong shading

$I_o = I_{amb} + I_{diff} + I_{spec}$
$= k_a * I_a + k_d * I_i * (\overrightarrow{l} * \overrightarrow{n}) + k_s * I_i * (\overrightarrow{r} * \overrightarrow{v})^n$

- k_a is the ambient coefficient and I_a is the intensity of the ambient light
- k_d is the diffuse coefficient, I_i is the incident light and $\overrightarrow{l}$ is the light direction
- k_s is the specular coefficient and $\overrightarrow{r}$ is reflection of light direction $\overrightarrow{l}$ at the surface

3. Design and Implementation

As shown in the image below(Figure 2), here is a main structure of my scene. Player with camera as the component, representing the player in realistic. Inside the camera, I created a script called camerapointer.cs, which would help on eye tracking and recording the gaze from the user. If gaze is on the object, message would send out for other scripts to use to achieve more functionality. Nothing special for light source, it would be needed to calculate the diffuse and specular component in Phong shading. The 3Dobject, sphere, is the main component in this project. As shown, it has a material wrapped around the object. For the material, it would apply the shader that I created for it, PhongShader. In addition, objeccontroller.cs is another script that used for object interaction.

3.1. Shader

Inside the PhongShader, there are three main steps inside the shader script. The first step is the statistic set up,

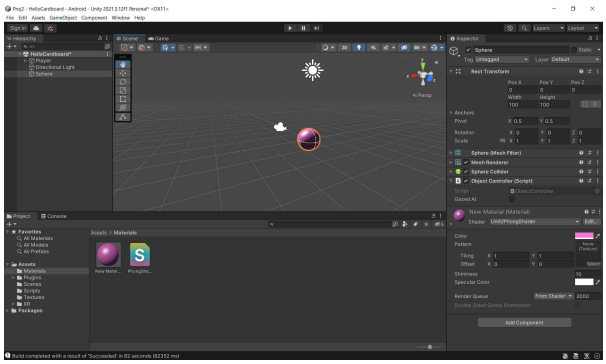

Figure 2. Shader Implementation in Unity

```
struct vInput
{
    float4 vertex : POSITION;
    float3 normal : NORMAL;
    float2 uv : TEXCOORD0;
};

struct vOutput
{
    float4 pos : POSITION;
    float3 normal : NORMAL;
    float2 uv : TEXCOORD0;
    float4 posWorld : TEXCOORD1;
};

vOutput vert(vInput v)
{
    vOutput curr;

    curr.posWorld = mul(unity_ObjectToWorld, v.vertex);
    curr.normal = normalize(mul(float4(v.normal, 0.0), unity_WorldToObject).xyz);
    curr.pos = UnityObjectToClipPos(v.vertex);
    curr.uv = TRANSFORM_TEX(v.uv, _Tex);

    return curr;
}
```

Figure 3. Vertex Shader

```
fixed4 frag(vOutput finput) : COLOR
{
    float3 normalDirection = normalize(finput.normal);
    float3 viewDirection = normalize(_WorldSpaceCameraPos - finput.posWorld.xyz);

    float3 vert2LightSource = _WorldSpaceLightPos0.xyz - finput.posWorld.xyz;
    float oneOverDistance = 1.0 / length(vert2LightSource);
    float attenuation = lerp(1.0, oneOverDistance, _WorldSpaceLightPos0.w); //Optimization for spot lights. This isn't needed if you're just go
    float3 lightDirection = _WorldSpaceLightPos0.xyz - finput.posWorld.xyz * _WorldSpaceLightPos0.w;

    float3 I_amb = UNITY_LIGHTMODEL_AMBIENT.rgb * _Color.rgb; //Ambient component
    float3 I_diff = attenuation * _LightColor0.rgb * _Color.rgb * max(0.0, dot(normalDirection, lightDirection)); //Diffuse component
    float3 I_spec;

    // sphere transformation
    if (dot(finput.normal, lightDirection) < 0.0) //Light on the wrong side - no specular
    {
        I_spec = float3(0.0, 0.0, 0.0);
    }
    else
    {
        //Specular component
        I_spec = attenuation * _LightColor0.rgb * _SpecColor.rgb * pow(max(0.0, dot(reflect(-lightDirection, normalDirection), viewDirection)),
    }

    float3 color = (I_amb + I_diff) * tex2D(_Tex, finput.uv) + I_spec; //Texture is not applied on I_spec
    return float4(color, 1.0);
}
```

Figure 4. Fragment Shader

for example, object color, light position, light color, and coefficient for shininess and specular. Some information are set up inside unity, where can use "UnityCG.cginc" package to bring in predefined variables and helper functions ([1]). The second step is vertex shader. Vertex Shaders transform shape positions into 3D drawing coordinates. The third step is Fragment Shaders compute the renderings of a shape's colors and other attributes.

In figure3, it illustrates how I specified my vertex shader. First of all, I define the input and output for my vertex

```
public void Update()
{
    // Casts ray towards camera's forward direction, to detect if a GameObject is being gazed
    // at.
    RaycastHit hit;
    if (Physics.Raycast(transform.position, transform.forward, out hit, _maxDistance))
    {
        // GameObject detected in front of the camera.
        if (_gazedAtObject != hit.transform.gameObject)
        {
            // New GameObject.
            _gazedAtObject?.SendMessage("OnPointerExit");
            _gazedAtObject = hit.transform.gameObject;
            _gazedAtObject.SendMessage("OnPointerEnter");
        }
    }
    else
    {
        // No GameObject detected in front of the camera.
        _gazedAtObject?.SendMessage("OnPointerExit");
        _gazedAtObject = null;
    }

        // Checks for screen touches.
    if (Google.XR.Cardboard.Api.IsTriggerPressed)
    {
        _gazedAtObject?.SendMessage("OnPointerClick");
    }
}
```

Figure 5. Camera raycast

shader where how I transpose my position to global position for each vertex on the sphere. In addition, normal vector and texture information for each vertex is also been send to my fragment shader. In figure4, it demonstrate how I implement my fragment shader.

In figure4, it demonstrates the phong shading implementation. As discuseed in section 2 about phong shading formula. Viewing direction, light direction, and reflection direction is calculated. Within parameters setup in step 1, output in step2, and three vectors just calculated. Phong shading is able to compute now.

3.2. Interaction

In order to viewing the phong shading more precise, where we can see the specular spot and ambient at the back. We are allowing the object to move. Meanwhile, itself is rotating while gaze on the object.

Before discussing about object interaction. Another essential component of this program is the script inside the camera, called camerapointer.cs script (figure 5) to send message on whether the object been gazed. This part is achieved by raycastHit component inside the unity, which is like a lookat function in openGL. Camera location, camera facing direction, and a ray out from the camera, and maximum distance for the ray to reach, are the four parameters for the raycast to call. If it hit something which is not the last one it hit, and it is in front of the camera a, a message with information to "onPointerExit" for last object and "onPointerEnter" for current object. If the boolean is false, then "onPointerExit" is sent to the system. In addtion, google cardboard has a trigger on the right side, it can help to click on the screen. Thus, if the trigger been hit, then "onPointerClick" message will send to the system.

After explaining how the camera work and transfering the data, the interaction part will be easy to explain. As shown in figure 6, it shows how the program initiated with

```
public void OnPointerEnter()
{
    gazedAt = true ;
}

/// <summary>
/// This method is called by the
/// </summary>
public void OnPointerExit()
{
    gazedAt = false;
}

/// <summary>
/// This method is called by the
/// is touched.
/// </summary>
public void OnPointerClick()
{
    translation();
}
```

Figure 6. Start and update scene

```
public void translation()
{
    Vector3 newPos = new Vector3(0,5,0);
    transform.localPosition = newPos;
}

public void roation()
{
    transform.RotateAround(transform.position, transform.up, Time.deltaTime * 90f);
}
```

Figure 7. Transformation function

gaze off. Once scene start updating, the gaze boolean will change, and different functions call would occur. Three types of messages send from camera, now can be used inside the object control. Obviously, "onPointerEnter" and "onPointerExit" will handle the gaze boolean with rotaion function call. "onPointerClick" will control the translation function call. These two functions detail is shown in figure 7.

4. Demonstration

The result is simple to view. In figure8, it shows when user gaze on the object with. The rotation did not achieve well, since no texture has been applied to the object. However, the main part from phong shading, the specular spot and gradully from diffuse to ambient is demonstrated perfectly. Once the trigger been hit from the cardboard(figure 9), it would move up, so that we can see the bottom of the ball which should be ambient.

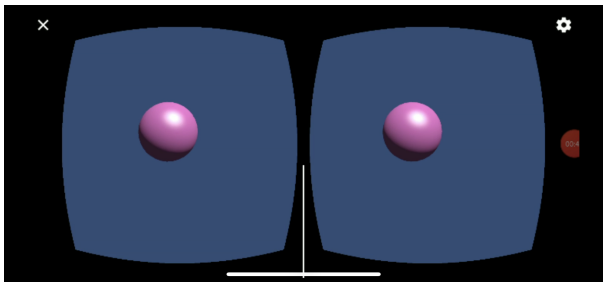

Figure 8. Demo: Gaze on the object

```
public void translation()
{
    Vector3 newPos = new Vector3(0,5,0);
    transform.localPosition = newPos;
}

public void roation()
{
    transform.RotateAround(transform.position, transform.up, Time.deltaTime * 90f);
}
```

Figure 9. Demo: translation

References

[1] Unity user manual. http://docs.unity3d.com/Manual/index.html. 2

[2] Steve Marschner and Shirley Peter. Fundamentals of computer graphics. CRC Press LLC, 2015. 1