Advanced Graphics – Physically Based Rendering

Jainesh Pathak
June 2023
School of Computing Science
Newcastle University
J.P.Pathak2@newcastle.ac.uk

ABSTRACT

A video game without lighting is never going to look pretty and will rather look uninteresting. Real world lighting calculation are complex and expensive for video games to handle. Many modern video games make use of a lighting calculation technique called Physically Based Rendering (PBR) which mimics the real-world lighting in a digital world and makes it look realistic. PBR works on microfacet surface model, energy conversations and Bidirectional Reflective Distribution Function (BRDF) to make it happen.

This report explains the process to achieve the approximation of realistic views using PBR and future for the project in context of video game graphics.

KEYWORDS

PBR; BRDF; Lighting; Lights; Irradiance; Billboards; Shading; Diffuse; Specular

1 INTRODUCTION

In computer graphics, lighting plays a vital role of making the player/viewer believe the realism of the digital worlds. In old arcade style games like Pac-Man, Wolfenstein 3D, lighting was non-existent, and player can see the whole level or map even at huge distance (E.g., A corridor in Wolfenstein 3D). Even if expensive lighting algorithms existed, the CPUs of that era were not powerful enough to handle those lighting calculations in real time.

Later, as CPUs and GPUs become more powerful; "close-to" realistic lighting also became possible.

2 RELATED WORK

2.1 Lighting in old video games

One of the greatest games of all time – Doom and others like Heretic, Hexen did show some sort of dynamic lighting although not realistic which is called Sector-based lighting. In Doom game, the world is divided into sectors which are made by connecting line segments to form a polygon. Every sector had different properties like floor, ceiling, texture and most importantly the light levels. Every sector had a light level ranging from 0-255 with 0 being dark and 255 being very bright.

Lighting attenuation computations were done such that the light travel from one sector to its adjacent sectors and based on the distance of how much it traveled, the light level of current sector will be slowly attenuated. It did support additional light sources like lamp for doing light computation for that sector and its adjacent ones.

2.1.1 Light Diminishing

Another fine addition to lighting Doom introduced in "Light Diminishing". Where the brightness of the area from the player's point of view slowly decreases as the distance from the player increases. This did not create any realism as well, but it did create impressive scary atmosphere for the game. This same mechanism was also used to simulate fogs [1].

2.1.2 Colourmap implementation

Before explaining Colourmap, I need to first explain what colour quantization is.

2.1.2.1. Colour quantization

Quantization in general is an image compression technique which is used to narrow down certain range of values to a single value. This also helps in reducing the file size. Moreover, colour quantization works by reducing colours of the image such that the compressed is visually like its original.

In the PC port of original doom engine, it made use a Colourmap which is a colour quantized gradient texture which starts from a different colour starting at top and slowly fades to black colour as moving down. It is a precomputed lookup table which was used to fit in the game's 256-colour palette. The first 32 levels in the colourmap were purely dedicated for implementing lighting. See Figure 1 for example.

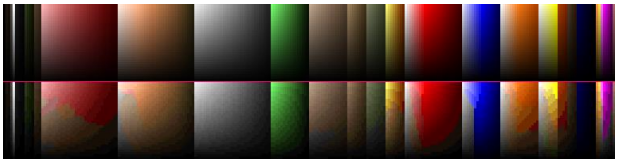

Figure 1: Top – Pure gradient. Bottom – Doom's quantized colourmap light levels

Overall, the lighting calculations in Doom was simple and not realistic, but it did bring immersive gameplay which made the maps look and feel believable.

2.2 Lighting in mid video games

Games from Doom 3, Unreal Tournament onwards started introducing dynamic lighting and shadow casting. Different light sources were introduced like Point Light, Directional Light and Spotlight.

To quickly summarize these different light sources:

2.2.1 Point Light:

This type of light source emits lights in all directions. In technical details it contains the position in 3D world, colour, intensity value and range.

2.2.2 Directional Light:

This light emits light in the single direction, and it has no attenuation and light travel infinitely in the game world. Just like a sun, however the sun in real life is one big point light in the solar system.

2.2.3 Spotlight:

This type of light emits light just like directional light, and it falls in a certain radius. Surfaces outside the radius are not lit at all. A good example of spotlight is flashlight which are common to use in horror games.

In real life, when a light falls onto an object, some of it get bounced or reflected and some get absorbed by the surface. The light which gets reflected leaves a white shining effect depending on the object's shininess. This shining effect is called Specular highlights.

In computer graphics, the most popular shading model for specular highlights in the Phong shading model which he has explained in detail [2]. His work is influenced from previous shading model such that of Warnock's [3] and Newell, Newell, Sancha's shading model [4] and Gouraud shading model which needs to be explained first.

2.2.4 Warnock's Shading:

Light source and camera position are placed at the same position and function returns the sum of two terms: Normal and Specular reflection. Surfaces facing towards the light source are always bright compared to its adjacent which were shaded with different light intensities. Also, discontinuity can be easily seen on curved surfaces as seen in Figure 2.

2.2.5 Newell, Newell, Sancha's Shading:

These people showcased an idea for creating specular highlights on transparent object. From experiments in real world, they found out that specular highlights are not created from a light source, but also from other objects as well which is highly depends on if the object has high reflective surfaces or transparent like glass. The application of this type of shading unfortunately is highly limited to transparent object and the specular highlights created on glass is like Warnock's shading as it can be seen in Figure 2.

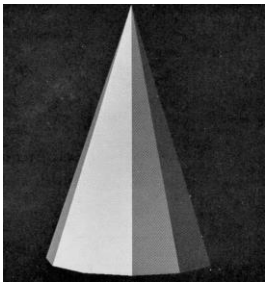

Figure 2: Warnock's Shading destroys smoothness on curved surfaces

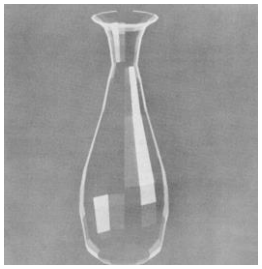

Figure 3: Newell, Newell, Sancha's Shading showing highlights on transparent object

2.2.6 Gouraud Shading:

Gouraud [5] developed an algorithm which computes the specular highlights on curved surfaces of each vertex of the triangle. From the surface's curvature, reflection intensity is calculated and is linearly interpolated towards zero between its edges connecting the neighbour surfaces. This generated visually appealing shading effect and retained the smoothness of curved surfaces as it can be seen in Figure 4.

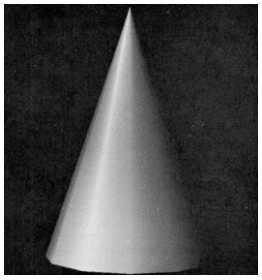

Figure 4: Gouraud Shading effect on low-polygon cone

Unfortunately, the algorithm was still not perfect, and it still created same discontinuity issue if the object has low polygons. If the surface of the object contains high amount of specular reflection, then the highlights are often irregularly shaped because it depends on the shape of the polygons approximate curved surface and not the curved surface of object. In computer generated films, the frame-by-frame discontinuities can be seen even more when the object is in motion, in one frame the faces which are orthogonal to direction of light ray takes a uniform shade and in the next frame, those same faces are in different orientation towards the light source. This same effect happens, when the object's orientation is changed.

This problem can be solved if the number of polygons is increased, but this is not an efficient solution, because of high memory amount required. Here's where enters the Phong shading model.

2.2.7 Phong Shading:

Bui Tuong Phong [2] introduced new shading model based on the Gouraud shading. It consists of three components viz. Ambient, Diffuse and Specular.

Ambient lighting is where Object is never dark, there is still some amount light in the world. It is a simple constant value that is added to object's surface colour. In simple term, Ambient light is omni-present even when there is not a single light source.

Diffuse lighting is an important which directly simulates the light ray on the object's surface. Surfaces that face the light source are brighter than the ones that are not facing.

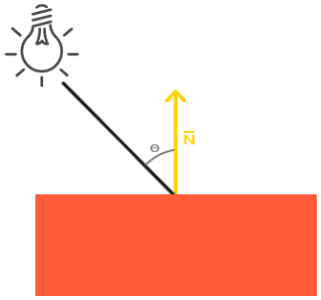

Figure 5: Diffuse Light Ray and surface normal N used to calculate the cosine angle

It is calculated by taking an incident ray or light ray direction vector, the normal vector which perpendicular to the object's surface and thereby calculating the dot product or the cosine angle of these two vectors. If the angle is greater than zero means the surface of object is facing the light source and vice versa. That is why it is best to check if dot product is always greater than zero beforehand.

To calculate light ray direction is simple by subtracting the light's position and surface position and normalizing it.

The normal vector is calculated from a triangle of a mesh. It is stored with the mesh data file as it is not required to calculate it every frame. To calculate the normal vector, we first take a difference vector from the two vertices to the one vertex and normalizing it and taking the cross product of those difference vectors gives the normal vector.

Once the angle is retrieved, it is simply multiplied with the light colour and lastly added with the ambient component to create a believable image. This can be seen in Figure 6 with ambient lighting.

Figure 6: Diffuse + Ambient Lighting on the mesh

Specular lighting computes the bright spot on the surface of the object depending on the shininess of the surface.

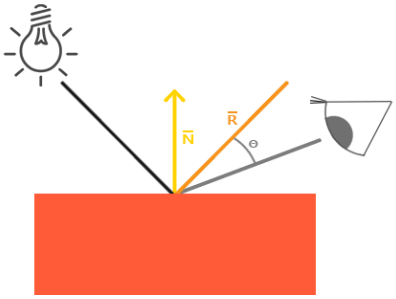

Figure 7: Specular lighting introduces View direction vector

It takes same input from diffuse i.e. the light direction which is a normalized difference vector of light position and surface position in world space and surface normal. Additionally it takes the third input which is a direction vector which is obtained by normalized difference of camera/eye position and surface position again in world space. Then the reflection vector is calculated from the light ray direction and surface normal.

Lastly the dot product is done from the view direction and reflection vector and is again made sure that its value is positive. This dot product is then raised by the shininess value constant. A high shininess value tightens the bright spot. See figure 8.

Figure 8: Phong Shading with shininess power of 128

2.3 Lighting in modern video games

2.3.1 Physically Based Rendering:

While Phong or Blinn-Phong shading does improve the visual quality of the image frame, it is still not close to being realistic. Physically based rendering (PBR) is combination of different rendering techniques that performs lighting calculations that closely matches the real world. It was introduced by Yoshiharu Gotanda [7] at SIGGRAPH 2010. PBR also makes the work of the artists a lot easier as they no longer use any kind of hacks like increase the colour range more than 1.0 and other stuff.

PBR works on the microfacet surface model which means that the surface normals of any object are not uniform or smooth and it contains rough surfaces when magnified.

It also promotes energy conservation, which means the amount of reflected light is always the same as the incoming light except for surfaces with emission. [8] explains the PBR pipeline is more details.

2.3.1.1. Microfacet model:

Microfacets are little perfect mirrors on a microscopic scale. The alignment of these microfacets depends on the roughness the surface. More the roughness, the light will reflect in random direction depending on the microscopic surface normal. Since, microscopic surface normal is not possible on per-pixel basis, a simple roughness parameter in the range of 0.0-1.0 is introduced to average the roughness of the object. The ratio of the microfacets can be calculated from the halfway vector:

$$h = \frac{l + v}{||l + v||}$$

Where l is the light direction and v is view direction. If the microfacet is aligned close to then it will create strong specular reflection.

2.3.1.2. Energy Conservation:

When the light falls on the surface of object, it either gets reflected which means the light rays clearly bounces off and never enters the surface or it gets refracted meaning some portion the light is absorbed by the surface. The light that gets reflected is called Specular Reflection and the one which gets absorbed is called Diffuse Reflection.

In real physics, the diffuse light reflection is not always absorbed, but rather the refracted light is bounced inside, where some lights may get reflected out and some don't. This however depends on the type of the surface the light is interacting with. If the surface is completely metallic, then the diffuse light is immediately absorbed and only specular light is reflected. In computer graphics PBR pipeline, it introduces a new parameter along with roughness which is called Metallic, and it also ranges from 0.0-1.0. Pure Metallic surfaces will never show diffuse colours.

2.3.1.3. Rendering Equation:

The rendering equation is an integral equation used in computer graphics for determine how exactly light interacts with surfaces and makes sure how the final object will look like after light falls on it. The equation looks like:

$$Lo(p,\omega o)=\int\Omega F(N, V, L)\, Li(p,\omega i)\, n\cdot\omega i\, d\omega i$$

Here, L denotes the term called radiance. Radiance is used to measure the strength of the incoming light from the single direction. A term called Radiant Flux is used to measure the energy sent from a light source in Watts.

In real life, light is sum of energy over a different wavelength graph. Each wavelength gives a particular colour in nanometers with ideal range between 390nm and 700nm. Taking these wavelengths for computer graphics in not possible.

ω here denotes the solid angle. The solid angle tells the size of shape which projected on a sphere with radius of 1. See figure 9.

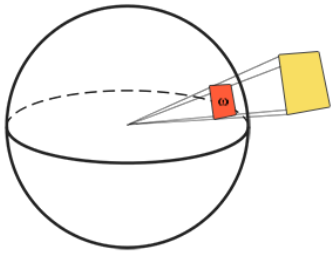

Figure 9: Solid Angle

Radiant intensity is the intensity of the light source over a projected area of the unit sphere. It is calculated using formula:

$$I = \frac{d\Phi}{d\Omega}$$

Here, I is the radiant flux divided by solid angle. With all this, the final radiance L can be calculated using the formula:

$$L = \frac{d^2\Phi}{dAd\omega\cos\theta}$$

Here, A can be represented as point P of the fragment in world space and solid angle ω can be represented the light direction vector and cosine angle is the dot product of the light direction and surface normal N. So, L here denotes the radiance at point P in direction solid angle.

From the rendering equation, Lo is the irradiance which is the sum of all the radiance of all the light sources.

2.3.1.4. Bidirectional Reflective Distribution Function:

It uses a special Bidirectional reflective distribution function or BRDF. The job of BRDF is to determine how a light is bounced or reflected from the surface based on the reflective property of the surface. In computer graphics, this reflective property is called Roughness or Glossiness in some 3D engines. The BRDF function requires three inputs and is denoted as F(N, V, L). Here, N is the

surface normal, V is the direction from camera or eye and L is the direction of light

The most famous BRDF is the Cook-Torrance BRDF which is given by the equation:

$$f = Kd\ flambert + Ks\ fcook - torrance$$

Where, Kd is the light ratio that will get refracted and Ks the light ratio that will be reflected.

2.3.1.5. Diffuse and Specular BRDFs:

Reflections comes in two types: Diffuse and Specular. Diffuse is a reflection where photons enter the surface and moves out after many bounces. Whereas the Specular get reflected or refracted. See figure 9.

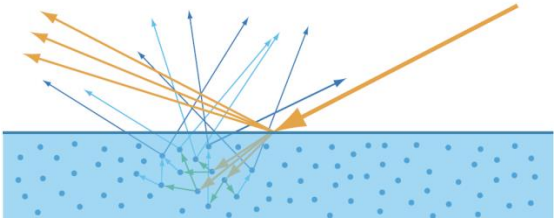

Figure 9: Yellow- Specular reflection, Blue- Diffuse reflection

To start with diffuse BRDFs, the most common the Lambertian diffuse BRDF which is given by the formula:

$$flambert = \frac{c}{\pi}$$

Here, c is the simple albedo colour which can be obtained from any albedo texture and is divided by pi for normalizing the reflected diffuse light. There are other types of diffuse BRDFs like the Oren-Nayar BRDF, Burley BRDF.

The specular BRDF is the Cook-Torrance Specular BRDF which is again given by the formula:

$$fCook - Torrance = \frac{DFG}{4(wo\ .n)(wi\ .n)}$$

It is made from three sub-functions; D is the Normal distribution function (NDF) which approximates the alignment of microfacets along the halfway vector h based on the roughness parameter. Here, Trowbridge-Reitz GGX is used to calculate NDF using the formula:

$$NDF(n,h,r) = \frac{\mathrm{r}^2}{\pi((\mathrm{n}.\mathrm{h})^2(\mathrm{r}^2-1)+1)^2}$$

Here, n is the surface normal, r is the roughness value and h is the halfway vector between the surface normal and light direction. If the roughness is zero, it will create a bright spot and vice-versa as it seen in figure 10.

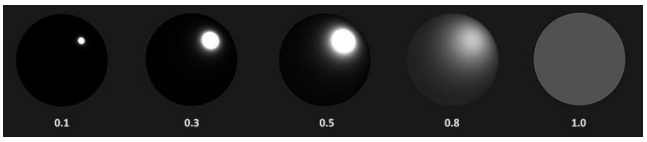

Figure 10: NDF Reflections based on roughness strength

The second sub-function is Geometry function whose purpose to calculate the shadow factor of the microfacets which again depends on the strength of the roughness or glossiness. Here, the Schlick-GGX formula is used with normal, view and roughness as inputs.

$$Gschlick - GGX(n,v,k) = \frac{n.v}{(\mathrm{n}.\mathrm{v})(1-\mathrm{k})+\mathrm{k}}$$

Where, k is remapping of roughness r based on direct or image-based lighting. For doing approximation of geometry, the Smith method is used for with Schlick-GGX which takes light direction vector l and view direction vector v and remapped roughness k being common for both. Figure 11 shows the Schlick-GGX method with different roughness strength.

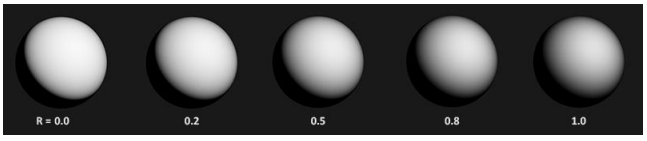

Figure 11: Geometry Shadowing based on roughness strength

There are other G functions like Neumann, Keleman for approximations of shadowing.

The third and last sub-function is the Fresnel which is most important function because in real life everything has shine when looked from a certain angle. Even a brick object with highest roughness strength will show some specular effect when looked at almost 90-degree angle. Fresnel-Schlick is the most common approximation used in films and video games which has its formula:

$$Fschlick(h,v,F0) = F0 + (1-F0)\big(1-(h.v)\big)^5$$

One good thing about Fresnel-Schlick approximations is that it takes account of both metallic and non-metallic or dielectric surfaces. Just like roughness strength of range 0.0-1.0, game engines also make use metallic strength which defines if the surface is pure metal or dielectric with 1.0 being pure metal and 0.0 being dielectrics.

F0 is the base reflectivity of the surface which is achieved using the Index of Refraction (IOR) depending on the type of surface of object like metal, water, plastics, etc. Fresnel is seen more when the angle between the view vector and halfway vector reaches almost 90-degrees. See figure 12.

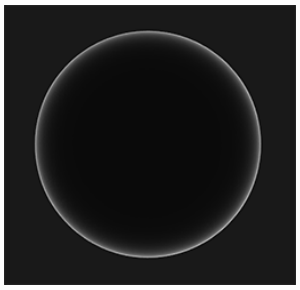

Figure 12: Fresnel is seen at angle reaching approximately 90-degrees

Unreal Engine 4 [9] uses a modified Schlick approximation which uses the Spherical Gaussian approximation to replace the power of 5 in the above formula which gives minor performance boost.

$$Fschlick(h, v, F0) = F0 + (1 - F0)2\text{^}(-5.55473(v.h) - 6.98316(v.h)$$

3 PROJECT APPROACH

3.1 Using OpenGL:

After discussing various mentioned algorithms, for starters I have decided to continue to use the NCLGL framework provided in this master's course and using OpenGL as the main graphics API.

The reason I chose OpenGL over other graphics API like DirectX and Vulcan is because it itself is a big graphics API with many features to learn and use it.

3.2 Material System:

Although the material system is not directly related to PBR. I want to create a material system for my own personal knowledge to learn how the material system works inside the graphics engine. This was inspired by the current generation game engine like Unity, Unreal engine.

The current project directly makes use of shader and only one shader can be at runtime. With the help of the material system, this problem can be mitigated, such that materials can be changed at runtime simply by changing the index which will bind material and its shader at the current index and whatever other properties like colour, textures are sent to the GPU.

3.3 Post Processing:

Post processing really interests me a lot because it gives the power to change the final image from one colour to another. For this, I am planning to do only two post processing effects i.e., the Bloom and Screen Space Ambient Occlusion (SSAO).

3.3.1 Bloom:

For bloom, I plan to use a technique where the frame image only consisting of bright colours is first down scaled up to 8 times such that it looks blurry and rightly up scaled scale 8 times that the up scaled image looks closely like the original bright colour image. This kind of Bloom effect was shown in Call of Duty: Advanced Warfare game. The bloom down scaling and up scaling will be done in the compute shader.

3.3.2 Screen Space Ambient Occlusion (SSAO):

I have also planned to do another post processing effect of Screen Space Ambient Occlusion (SSAO) for creating soft shadows around corners of the mesh.

4 PROJECT PLAN

I have extended the NCLGL library with abstract classes like the Frame Buffers, Texture, Uniform Buffer Objects. Making frame buffer objects will help me do post processing effects like Bloom as well as HDR with Tone Mapping and Image Based Lighting (IBL).

4.1 Uniform Buffer Objects:

Having discussed the approach, I have implemented the forward lighting in the fragment shaders, but this time with small changes. Previously, I used to just send all the lighting relevant data via their uniforms location every frame to the GPU, but now after learning the use of Uniform Buffer Objects (UBOs), I don't have to send the lighting data every frame.

With additional integration of Dear ImGui tool which is an awesome tool for debugging or changing a value at runtime. Whenever, any property of a light is changed using ImGui, only a small block of the whole UBO light object is changed by providing the offset and size of the data that was changed. This small optimization helps.

Multiple light sources like point, directional and spotlight are also implemented with maximum amount of 100.

The same UBO object was also used for storing the projection and view matrix of the camera object instead of sending then every frame.

4.2 Textures:

Texture class was created to get flexibility on type of texture is loaded like RGBA, RED, etc. and get its width, height dimensions of the image whenever required along with options like what internal format to give before loading the image. With this I have also created more texture classes which inherits from normal texture class like HDR texture class for HDR and gamma correction and a Cube Map texture class for converting the equirectangular HDR image to a 6-sided cube map texture.

4.3 Tone Mapping:

For Tone Mapping to work, the colour texture format attached to the frame buffer need to be of 16-bit or 32-bit float format for it work. Such frame buffers are called floating point framebuffers. Tone Mapping uses simple Reinhard tone mapping algorithm. There are plans to use other tone mapping algorithms in the future.

4.4 Billboarding:

This is not related to lighting in the project. Billboard is a simple flat quad mesh object which will always face the camera or any target it is given. The direction always changes in runtime when the camera or the billboard object moves. Referring to [6], I will explain the type of billboards and its implementations. Billboards comes in three types:

4.4.1. Point Sprites:

Point Sprites billboards are common billboards where the center pivot point is in the middle of the quad mesh and is rotated around that pivot point. The current project utilizes the point sprites billboards.

To implement the point sprite billboards, it requires special billboard matrix, which is like a normal model matrix, but for the billboards.

The first step is to get the look direction vector from billboard to camera which is obtained by simply subtracting the camera position and billboard position in world space and normalizing it. This look vector is required for any type of billboard.

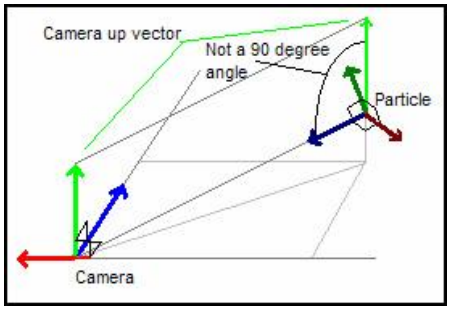

Figure 13: Point Sprites

After getting the look vector, we need to calculate the right vector of the billboard. For this, a temporary camera up vector is required and simply doing the cross product of camera up and look vector will give the right vector of the billboard. Lastly, to get up vector of billboard, we again do the cross product of the look vector and right vector we got in previous step.

This billboard type can be seen in game engine as gizmos like Unity, Unreal, etc.

4.4.2. Axis Aligned:

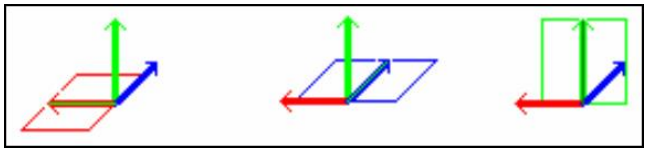

Figure 14: Axis Aligned Billboards locked Y axis

Axis aligned billboard is a lot simple as it doesn't the temporary camera up vector. AA billboards always rotate around the locked global Y axis or the up axis. In OpenGL it is a simple up vector of (0, 1, 0). The look vector works as forward vector and right vector calculated by doing the cross product of the look and up vector.

This kind of billboards can be seen in action in old games like Wolfenstein 3D and Doom.

4.4.3. Arbitrary Axis:

Arbitrary axis billboard works slightly different. The look vector here is not the final look vector, but a temporary one which is used to get the right and up vectors of the billboards. This temporary look vector is not orthogonal to the up vector. The up vector is given by the axis we want to rotate around which can be either X, Y or Z.

To get the right vector, a cross product of the up and temporary look vector is done. Finally, the original look vector can be calculated using the cross product of up and right vector. See figure 15.

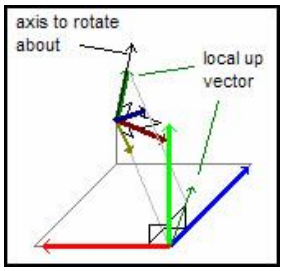

Figure 15: Arbitrary Axis

4.4.4. Billboard Matrix:

Once the entire forward, up and right vectors are done, the billboard matrix can be calculated and is sent to the vertex shader. To calculate the matrix, the three – forward, up and right vectors are put in a 4x4 identity matrix. The billboard matrix looks like this:

$$\begin{bmatrix} R1 & U1 & F1 & Px \\ R2 & U2 & F2 & Py \\ R3 & U3 & F3 & Pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where R is right vector, U is up vector, F is forward or look vector and P is the position of the billboard in world space. This matrix is later multiplied with the view projection matrix in the vertex shader.

4.5 PBR:

The PBR shader is implemented which currently takes the direct lighting data from different light sources. It uses the Cook-Torrance BRDF for calculating specular. But there are plans to use different BRDFs in future.

The next step in PBR pipeline is to do the image-based lighting (IBL). So far, the shaders are implemented to convert the equirectangular HDR texture into a cubemap texture and relevant frame buffers are added.

Current issue I am facing right now is that the cubemap texture that is made from HDR texture is not being rendered and instead is showing the black texture. To debug this, RenderDoc software will be used. Once the issue is solved, the next thing to do is to create the irradiance texture to get the lighting data. Figure 16 shows current PBR with direct lighting in action.

Figure 16: PBR from point and directional light sources with point sprite billboards

EVALUATION

The project will be evaluated for performance benchmarks using the ImGui tool like doing frames per seconds (FPS), startup time, rendering time. For debugging purpose, RenderDoc software will be used if any issues are occurred in graphics or data when sent to the shader. For graphics quality, the PBR scene in the project can be compared with the scene from other engines like Unreal Engine Uber shader and Unity Autodesk Interactive PBR.

REFERENCES

[1] "Light Diminishing" 2022. [Online]. Available: Light diminishing - The Doom Wiki at DoomWiki.org

[2] Bui Tuong Phong, Illumination for computer generated pictures, Communications of ACM 18 (1975), no. 6, 311-317.

[3] Warnock, J.E. A hidden-line algorithm for halftone picture representation. Dep. Of Comput. Sci., U. of Utah, TR 4-15, 1969

[4] Newell, ME.E, Newell, R.G., and Sancha, T.L. A new approach to the shaded picture problem. Proc. ACM 1973 Nat. Conf.

[5] Gouraud, H. Computer display of curved surfaces. Dep. Of Comput. Sci., U. of Utah, UTEC-CSc-71-113, June 1971. Also in IEEE Trans. C-20 (June 1971), 623-629.

[6] Neon Helium Productions, "Billboarding How To". [Online]. Available: https://archive.fo/SLzw2

[7] Gotanda, Martinez, Snow, "Physically-Based Shading Models in Film and Game Production," SIGGRAPH 2010. [Online]. Available: SIGGRAPH 2010 Course: Physically-Based Shading Models in Film and Game Production (renderwonk.com)

[8] Joey De Vries, "PBR Theory". [Online]. Available: https://learnopengl.com/PBR/Theory

[9] Brian Karis, "Real Shading in Unreal Engine 4". Epic Games. [Online]. Available: s2013_pbs_epic_notes_v2.pdf (selfshadow.com)