Real-time Graphical Techniques

https://youtu.be/8PBi2Nxfmbo

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

ABSTRACT

Lighting physics in the real world is complex and too much expensive for the processor to handle such computations. The department of computer graphics has undergone numerous transformations and revolutionized the way computer generated images are produced and perceived from drawing 2D lines to drawing complex 3D geometry with the help of fast parallel processing of the GPU.

One such transformative shift is the introduction and widespread adoption of Physically Based Rendering (PBR) which tries to bridge the gap between virtual and physical reality, and which is basically the approximation of the real-world lighting physics in the computer world. PBR’s core components works on the utilization of microfacet theory, Fresnel equations and a special function called BRDF and doesn’t violate the law of energy conservation. For PBR to work at perfectly, the image needs to gamma corrected. While generating images from different light sources is good, it can be further augmented using Image-based Lighting or IBL which takes the light information from a single texture.

Although, PBR is part of lighting calculations which generates beautiful images. Those images can be further enhanced using the term called Post Processing. There are quite many post processing effects whose functions differ from other effects like Bloom, Vignette, Ambient Occlusion, Motion Blur, Auto Exposure, Depth of Field, etc. Post Processing is heavily used in static images video games and visual effects of movies.

This report serves as a demonstration of PBR explaining the algorithms in-depth, investigating factors like roughness, material surface properties, view direction, and few Post Processing effects like Bloom, SSAO, Vignette. Lastly, it will also do comparison of PBR with other well-known shading effects used in the industry like Disney’s Principled BSDF, Oren-Nayar and Phong shading.

**KEYWORDS**

PBR; BRDF; Lighting; Irradiance; Shaders; Diffuse; Specular; Post-Process

**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/room 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 became more powerful, the simulation of realistic lighting also became possible by exploiting the parallel processing power of the GPUs.

The foundations of PBR needs to be properly explained before diving straight into the rendering equations. Shirley [1] explains those basic foundations by first stating that PBR normally has three distinct goals:

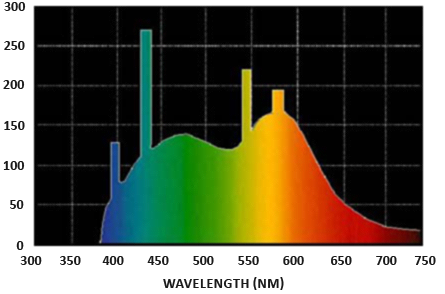
* Predictive Rendering: The computer image result matches the scene which is used in design and simulation applications.
* Plausible Rendering: The image result is close to real world but can be bit incorrect and is used in video games and movies.
* Visually Rich Rendering: Visual richness of reality can be stylized.

For this report, the focus is on video games. So, Plausible Rendering is sufficient.

**1.1 Light**

Lights are called Photon. In real-life, a camera has lens and sensors. The sensor has minute elements called pixels which respond differently depending on the direction and wavelengths of photon. However, in rendering to measure to real light properties like its wavelength both single or multiple and mass is not possible. Wavelengths between 390nm and 700nm are part of visible light spectrum that the human eye can see. For this report, it is assumed that a photon contains some energy, wavelength, time, position and direction.

Directly taking these different wavelengths measures in graphics is not practical. Hence, graphic programmers make use of light colour encoded in the RGB component (Red, Green, Blue) in the range of 0 to 1 for simulating the light. Figure 1 shows the different wavelength of daylight.



**Figure 1: Wavelength of a daylight**

**1.1.1 Light Sources**

Different light sources were introduced like Point Light, Directional Light and Spotlight.

To quickly summarize these different light sources:

1.1.1.1 Point Light:

This type of light source emits lights in all directions. The intensity gradually attenuates as the distance from surface and light increases. In technical details it contains the position in 3D world, light colour, intensity value and range.

1.1.1.2 Directional Light:

This light emits 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.

1.1.1.3 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.

**1.2 Radiance**

Radiance is used to measure the energy emitted, reflected or received by the surface of the object when looking at different angles. Shirley [1] explains that radiance is discussed across many literatures and has come up with few properties of radiance:

* Radiance travel in constant straight line.
* Radiance can be colours that is seen from the direction.
* Camera sensors measure the moments of radiance.
* All light measurements can be derived from radiance.

Number 1 and 2 are similar in the sense that photons never lose any energy as they travel, but this easily violates the light attenuation law.

Therefore, radiance can be defined as a function over space:

*Radiance = L(position, direction, wavelength, time)*

Where *L* is the Spectral Radiance. Spectral Radiance is the emitting radiance spread across different wavelengths. In other words, it tells us how much light energy is transmitted off the surface at which wavelengths.

This function is adopted almost everywhere except for surfaces where reflection/refraction happens. As a result, there are two kinds of radiance:

* Incoming Radiance: Incoming direction of the light of what the surface sees.
* Outgoing Radiance: Direction that the camera/eye sees when looking at the surface.

**1.3 Irradiance**

The continuous sum of all radiance is called Irradiance at point x. The integral or constant function is denoted as H which is the density of light coming from all directions landing on a surface.

The irradiance equation for incoming radiance is given as:

Here, *ω* is the incoming light direction and *θ* is the angle or dot product between surface normal and light direction.

The integral function is the same for outgoing radiance:

**1.4 Lambertian Surface**

A *matte* is a surface that is not reflective or shiny and most of the surfaces are *matte*. They are usually common in computer graphics and are made with the help of *Lambertian Surface* principles.

The basic property of a *Lambertian Surface* is that the brightness of a surface to the camera/eye is same regardless of the angle of view. The surface normal are isotropic or evenly distributed.

The Lambertian diffuse reflectance is the most used in computer graphics due its easy formula:

*C* is the albedo or flat colour of the surface. Here, π is used to satisfy the energy conservation law which means the surface should emit same amount of energy it receives.

Once, the Lambertian result is achieved, it is finally multiplied with the irradiance equation *H*.

**1.5 Rendering Equation**

The rendering equation especially for computer graphics was first introduced by Kajiya [2] in 1986. It tells about the radiance of the surface to the incoming radiances. The general rendering equation is given as:

In summary, the rendering equation tells that light that comes in for all directions is either reflected or refracted. Here, *p* is the Bidirectional Reflectance Distribution Function (BRDF) and it depends on the surface properties like roughness. Also, *i* and *o* are the incoming and outgoing radiances and *Le* is the emitted part.

*θ* is the angle or dot product between surface normal and light direction. This equation is further explained in detail later in the report.

**1.6 BRDF**

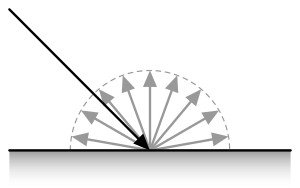
A Bidirectional Reflectance Distribution Function (BRDF) is a function that shows how a light radiance is reflected from an opaque surface. It takes inputs *ωi* and *ωo* as incoming and outgoing light and surface normal *n*. It returns ratio of light direction along the viewing direction and irradiance on the surface from the incoming light direction.

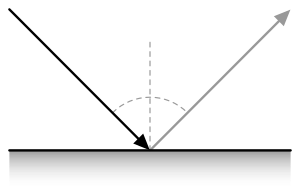
BRDFs comes in two types:

* Diffuse BRDF (Refraction)
* Specular BRDF (Reflection)

The diffuse BRDF takes care of how incoming light energy is absorbed or refracted into the surface. The most famous diffuse BRDF is the lambert which is explained above in section 1.4. Lambert diffuse is famous for its simplicity but produces incorrect results and doesn’t work on micro surface level and it is not the only BRDF. There are other diffuse BRDF like the Oren-Nayar and Burley BRDF which is implemented and will be explained later in this report.

The second type is called the Specular BRDF which shows how the light is reflected depending on the roughness property of the surface. It requires more computations than the diffuse BRDFs. There are several variants specular functions like the Cook-Torrance, Beckmann, GGX, Gaussian, etc. Some variants are accompanied with Fresnel effects. The previous mentioned BRDFs are also implemented and is further explained in the report. Figure 2 shows light reflections in gray colour depending on the type of surface.





**Figure 2: Clockwise from top left: Diffuse, Glossy, Mirror**

**2 BACKGROUND AND RELATED WORKS**

2.1 Lighting in Doom

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 is Light Diminishing or Distance Fading. 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].

Figure 3 shows the distance fading effect. Bandings can be clearly seen on walls and floor. The reason for banding that can be clearly seen on floor and wall is explained further in colourmap implementation.



**Figure 3: Light Diminishing. Banding effect is seen on floor and walls.**

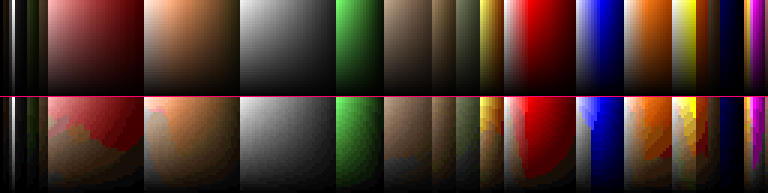
2.1.2 Colourmap implementation

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 of 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 4 for example.



**Figure 4: Top – Pure gradient. Bottom – Doom’s quantized colourmap light levels.**

**2.1.3 Fake Contrast**

Fake contrast is a feature in Doom engine where the walls parallel to east-west axis are made darker compared to walls facing north-south. Although, this feature dated back in older games like Hovertank 3D. However, this was not perfect because it works only on orthogonal geometry.

**2.1.4 Coloured Lighting**

Coloured Lighting is another feature in Doom engine where the ambient lighting in a sector can be multiplied with a tint colour. This feature was heavily used in PlayStation port of Doom.

Doom 64 had more sophisticated coloured lighting where every sector has a maximum support of five different coloured lights and each of them affect only one type of entity like sprites, floor, ceiling, etc.

*2.1.4.1 PlayStation Special Lighting*

The PlayStation port of Doom had a feature of special lighting effects that can affect only walls or floors. It makes use of a table of 256 32-bit binary colour values which are stored in RGBA order. Every sector in PS1’s map format has a single byte property which stores the index pointing to the colour table for determining the special lighting effect. Figure 5 shows the light colour information.



**Figure 5: PS1’s Special Lighting colour table.**

In conclusion, the lighting calculations in Doom was simple with only support of ambient lighting and with some special lighting effects did contribute to bring immersive gameplay and visually impressive atmosphere.

Later, as the engine was released as open-source, several third-party or fan-made projects were released with enhances and integration of modern graphics API like OpenGL, Vulkan with dynamic lightings.

2.2 Shading Models

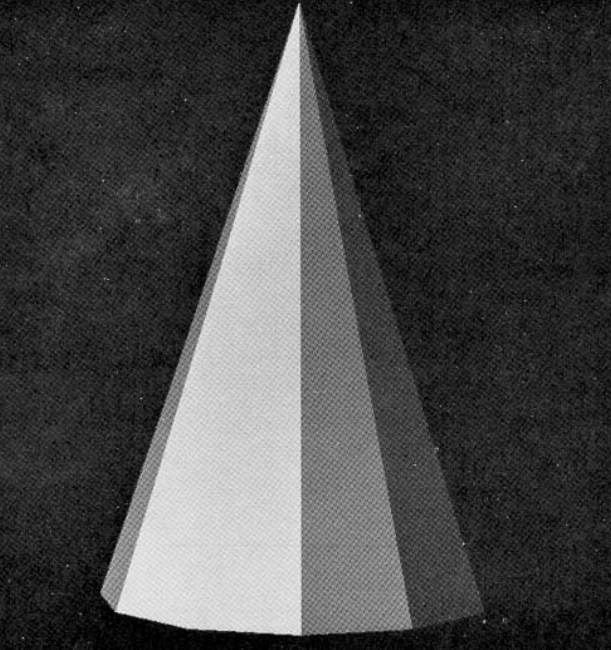
In computer graphics, the most popular shading model for specular highlights in the Phong shading model which he has explained in detail [2]. His work is influenced from previous shading model such that of Warnock’s [3] and Newell, Newell, Sancha’s shading model [4] and Gouraud shading model which needs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

2.2.1 Warnock’s Shading:

Back in early computer graphics time, the most common approach for rendering polygons is the Scanline Rendering approach which works on horizontal line-by-line basis. The main objective of this rendering is to provide depth perception which will be draw on 2D screen. For each scanline, it determines the visible parts of polygons and computers the shading for those visible areas. The polygons that are to be rendered are first sorted in Y coordinates and later by scanline checks for any intersection of a scanline to determine which polygon is on front and is added to the sorted list. This sorted list is updated as the scanline moves down the screen. The benefits on this sorting algorithm were that there was no need to do any edges comparison on per-pixel level.

Shading algorithm developed by Warnock [3] works on scanline rendering approach. His algorithm is also often called as Flat-Shading and is one of the foundations for modern shading algorithms.

His algorithm works by putting light source and camera are placed at the same position and function returns the sum of two terms used: Surface Normal and Specular reflection. The reason for putting the light source and camera at same position is to avoid the shadow problem. Surfaces facing towards the light source are always bright compared to its adjacent which were shaded with different light intensities. The polygon surfaces that move away from the light source receive less light intensity and hence appear darker. Also, discontinuity can be easily seen on curved surfaces as seen in Figure 6.



**Figure 6: Warnock’s Shading destroys smoothness on curved surfaces.**

His shading algorithm uses the rule:

Where θ is the angle between light direction and surface normal and R is the measure of distance from the observer and returns the absolute value.

The advantage of this algorithm is that it was efficient enough to draw complex scenes on low-end hardware in less time and provided visually believable rendering for flat surfaces. Unfortunately, his shading assumes that the 3D models are always convex and hence doesn’t work on concave polygons.

2.2.2 Newell, Newell, Sancha’s Shading:

While Warnock’s and other shading works well for opaque and flat-shaded model, they don’t work for cases where object contains transparency.

The Newell-Sancha [4] model is an extension of previous shading models that does the depth sorting in a different manner and handles the shading of transparent objects as well.

Their depth sorting algorithm works by first having a linear array of data items which contains plane equations in screen-space and values of x and y on screen. Additionally, it also has an ordered list which stores the references of faces in order of minimum z values where that minimum z tells the point furthest from the camera. The top of the list is then compared with other points to check whether that other point is drawing over the current face. If the current face is not obscuring, then it is removed from top of the list and added to the screen map used for rendering and moves to the next point and does the same comparison.

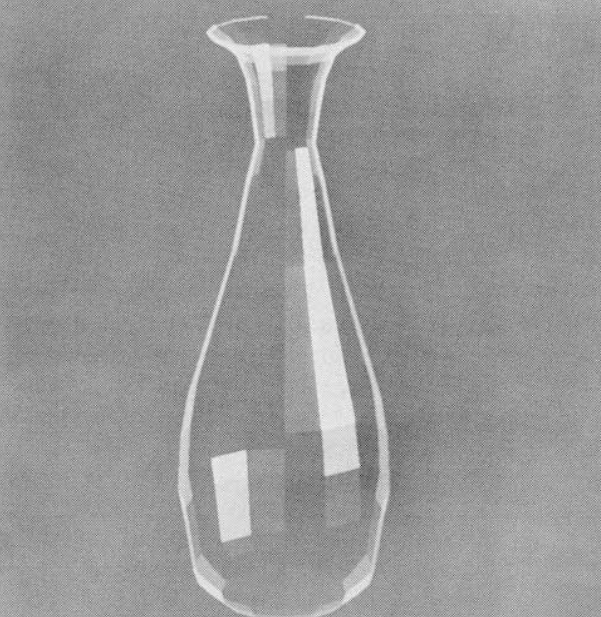
However, there were also edge-cases where the two faces fail to determine there are obscuring or not. In such case, one of the faces is split into two faces and check if the second face lies on both recently split faces of current face. If it does, then second face is also split and is added in the ordered list for depth comparison.

For performing light shading operations, they followed same method as Warnock’s by keeping the light and camera at same position with the reason to avoid shadow problem. Object faces were shaded using different light intensities using the existing simple function:

Here, *r* is the light intensity range, n being some random power, *a* is angle between the incoming light and face normal and *b* is the ambient light level. With *n* being 1, it simulated diffuse reflection, but as power is increased for example 20, then it leads to objects appear much darker that are barely facing the light and those facing appears the brightest.

Transparent faces are simulated by having the line segments added to the screen map instead of overwriting it and using a different intensity function:

Where I1 and I0 are the intensities of newly created line segment and the one that is overdrawn, and *w* is the weight factor. Figure 7 shows the shading effect of transparent object.



**Figure 7: Newell, Newell, Sancha’s Shading.**

The intensity functions for transparent faces were more of a hack than simulating real world lighting on non-opaque objects. But they were efficient enough to run on limited hardware.

Overall, the shading of Newell-Sancha model was still flat-shaded with a fine addition of handling transparent surfaces. They were still not good enough for curved surfaces and the problem of Mach banding effect still occurs. The Mach banding effect problem is explained further next section.

2.2.3 Gouraud Shading:

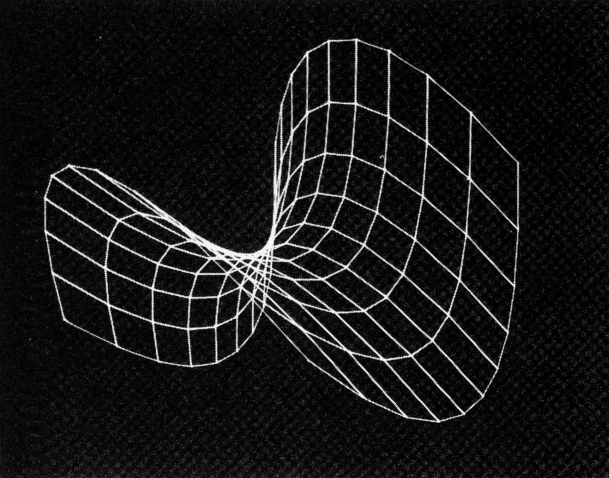
Warnock and Newell-Sancha shading were visually impressive but were prone to problem of Mach bands effects when it comes to curved surfaces.

*2.2.3.1 Mach Band Effect:*

Mach bands [MACH BAND] is a psychological illusion that triggers the edge detection in the human visual system. It is dependent on the intensity of light. Every neuron that it receives light information communicates with its neighbour and modifies the performance. It visualizes discontinuities as seen in Figure 3, 6, 7. In other words, the banding effect occurs when there is a jump in contrast values between two surfaces.

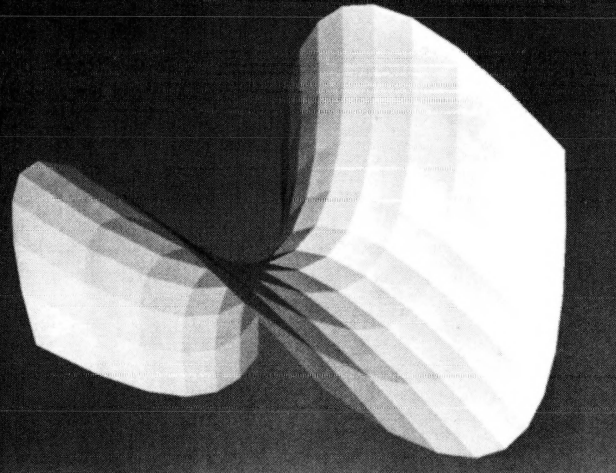
*2.2.3.2 Curved Surfaces:*

Coons [COONS PATCH] introduced the rendering the polygons with curved surfaces. It also known as Coons patch. The rendering of polygons using the Coons patch were grid-based or wireframe and didn’t have any support of depth sorting and as a result, back surfaces were also rendered. Several variants were introduced using Coons patch as a base layer. Figure 8 shows an example of Coons patch curved surface.



**Figure 8: Coons Patch model of hyperbole.**

Coons patch didn’t have any feature of light shading and running Warnock’s algorithm easily produces Mach bands as it can be seen in Figure 9.

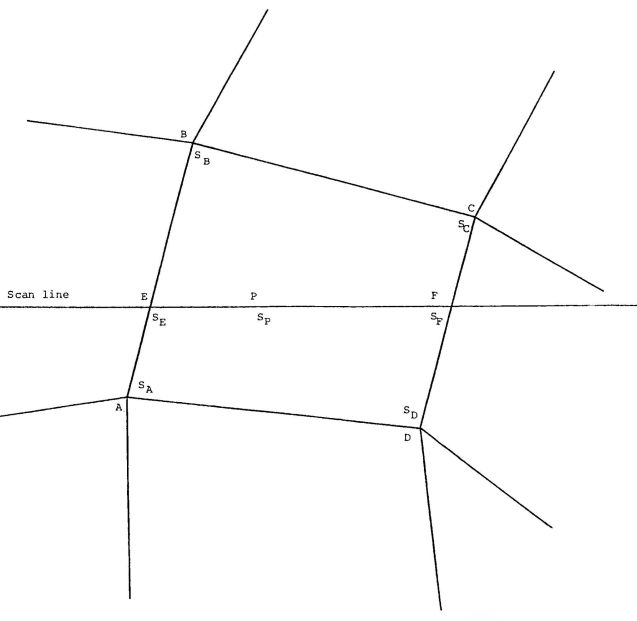


**Figure 9: Warnock’s Shading on Coons Patch.**

Gouraud [5] developed the shading algorithm which computes the specular highlights on curved surfaces of each vertex of the triangle. With the rule of shading that it needs to continuous for polygons and its neighbours. For achieving that continuity, every vertex has one normal which is computed as mean of all normal of each polygon associated with that vertex.

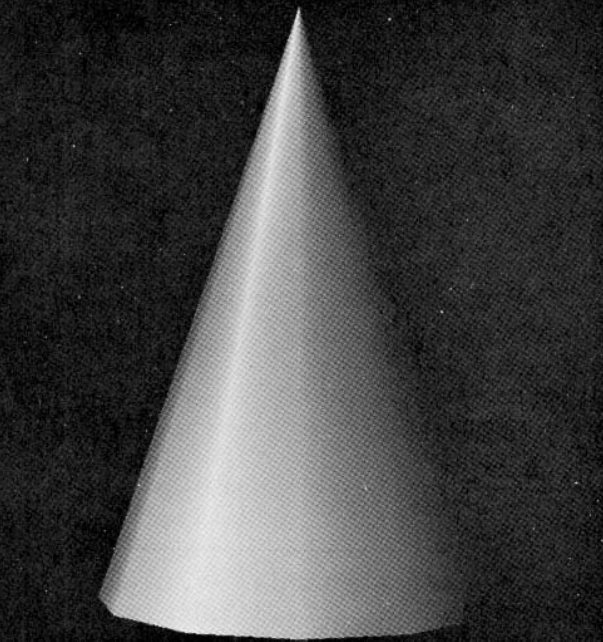
For example, if a polygon has vertices A, B, C, D and projection edges are AB and CD and a scanline (Figure 10) with E, F being the intersection point of scan line with edges AB and CD and P being any arbitrary point on the scanline, then the shading value at point P can be calculated as linear interpolation of SA and SB in formula:

Here, *a* is the coefficient (0 ≤ a ≤ 1) showing the position of E on the edge AB. The same formula can be calculated for SF and SP.



**Figure 10: Projection of a polygon with scanline**

This algorithm was demonstrated using the same scanline rendering approach. This generated visually appealing shading effect and retained the smoothness of curved surfaces as it can be seen in Figure 4.



**Figure 8: 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.

2.2.4 Phong Shading:

Bui Tuong Phong [2] introduced new shading model heavily inspired from Gouraud shading. It consists of three core 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 9: 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.

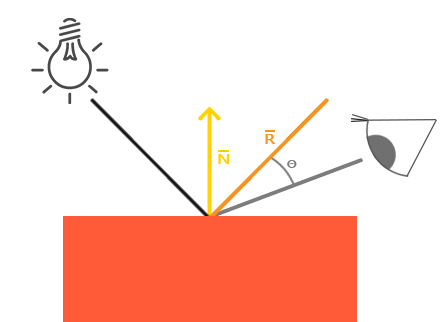
The normal vector is calculated from a triangle of a mesh. It is stored with the mesh data file as it is not required to calculate it every frame. To calculate the normal vector, we first take a difference vector from the two vertices to the one vertex and normalizing it and taking the cross product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

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 10 with ambient lighting.



**Figure 10: Diffuse + Ambient Lighting on the mesh.**

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



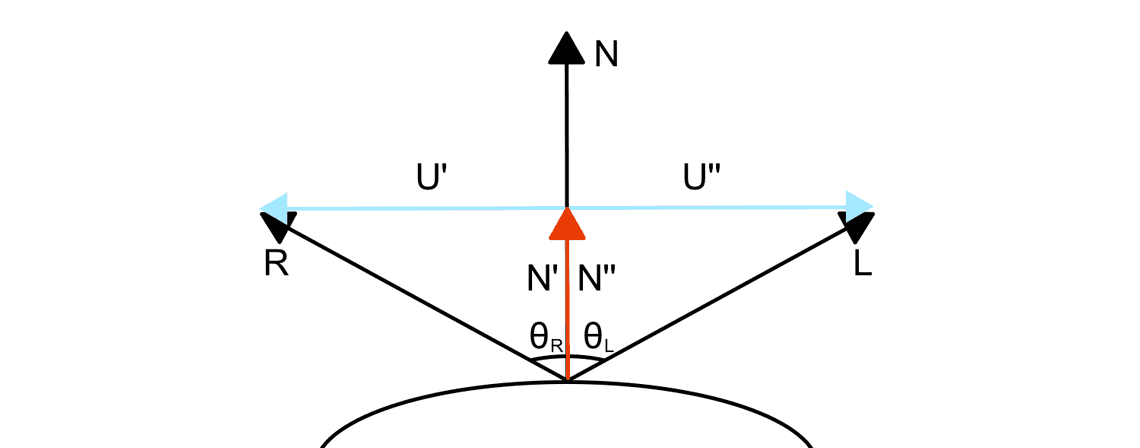
**Figure 11: Specular lighting introduces View direction vector.**

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 a 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 or cosine angle 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 13.

The reflection vector can be calculated using the formula:

Here, R is the reflection vector and N is the surface normal and light is incident light direction. Figure 12 shows the breakdown example.



**Figure 12: Reflection Vector**

Based on the law reflection:

*θR* is the dot product or cosine of angle between vector R and normal N’ and same for vector L and normal N’’. The above formula can be rewritten as:

From figure 12, it is also known that vector U’ = -U’’. Since, U’ is difference between R and N’, so the reflection can be written as:

Solving further, we can substitute the values for calculating reflection vector R as:



**Figure 13: Phong Shading with shininess power of 128.**

**2.2.5 Blinn-Phong Shading:**

Phong model is efficient way for approximation of lighting. However, it’s specular calculation runs into a problem. When the angle between reflection and view doesn’t go over 90 degrees, the specular portion gets cut off. Also, one must continually recalculate the dot product R.V between view direction V and reflected vector R.

This problem was solved by James F. Blinn which tells that instead of creating a reflection vector R using reflection formula, it can be calculated using the sum of viewing vector V and light vector L and normalizing it giving us the two-way vector. This is also known as halfway vector where the unit vector is exactly half between view direction and light direction. As a result, the dot product between surface normal and halfway vector never overshoots 90 degrees. This halfway vector H is calculated as:

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.

* + - 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 formula H explained in Blinn-Phong shading above.

Where L is the light direction and V is viewing direction. If the microfacet is aligned close to then it will create strong specular reflection or a bright highlighted spot.

* + - 1. *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.

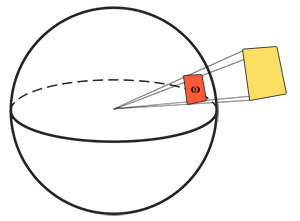
* + - 1. *Rendering Equation:*

The rendering equation was explained in brief in introduction section. It 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:

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:

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

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.

* + - 1. *Bidirectional Reflective Distribution Function:*

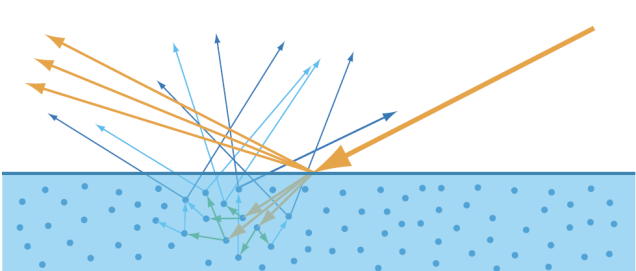
It uses a special Bidirectional reflective distribution function or BRDF. The job of BRDF is to determine how a light is bounced or reflected from the surface based on the reflective property of the surface. In computer graphics, this reflective property is called Roughness or Glossiness in some 3D engines. 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:

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

* + - 1. *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**

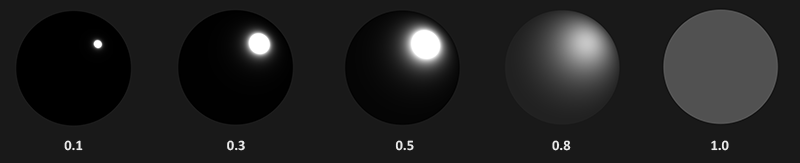
To start with diffuse BRDFs, the most common the Lambertian diffuse BRDF which is given by the formula:

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:

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:

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.

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.



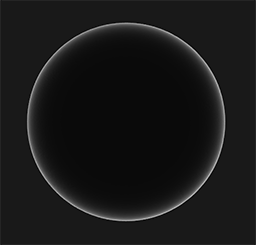
**Figure 11: Geometry Shadowing based on roughness strength**

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:

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.

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:

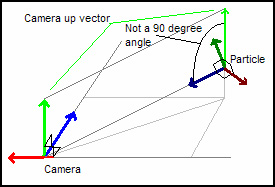
This is not related to lighting in the project. Billboard is a simple flat quad mesh object which will always face the camera or any target it is given. The direction always changes in runtime when the camera or the billboard object moves. Referring to [6], I will explain the type of billboards and its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

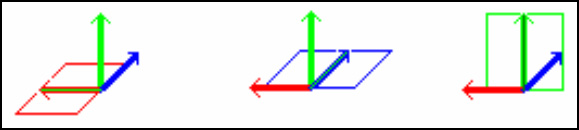


**Figure 13: Point Sprites**

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.

* + 1. *. Axis Aligned:*



**Figure 14: Axis Aligned Billboards locked Y axis**

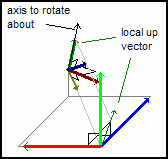
Axis aligned billboard is a lot simple as it doesn’t the temporary camera up vector. AA billboards always rotate around the locked global Y axis or the up axis. In OpenGL it is a simple up vector of (0, 1, 0). The look vector works as forward vector and right vector calculated by doing the cross product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

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

* + 1. *. 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:

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/SLzw2Conference Name:ACM Woodstock conference

[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)Conference Short Name:WOODSTOCK’18

[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)

Conference Location:El Paso, Texas USA

ISBN:978-1-4503-0000-0/18/06

Year:2018

Date:June

Copyright Year:2018

Copyright Statement:rightsretained

DOI:10.1145/1234567890

RRH: F. Surname et al.

Price:$15.00