PBL/IBL

Hiago DeSena

Introduction:

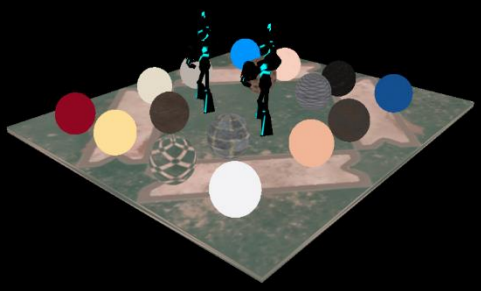
We can further develop our deferred shading pipeline by adding support for physically based and image based lighting. This adds more realistic pleasing visuals that enhance the rendering of our game environment. In recap a deferred shading pipeline is a two-step process. The first step is known as the **GBuffer** stage which processes all geometry in the scene, building the depth buffer and saving out geometry material information to multiple render targets simultaneously to be used for lighting calculation. Finally when all the data has been collected we bind those resources as textures and sample from them and calculate our lighting information based on the light type.

GBuffer Stage:

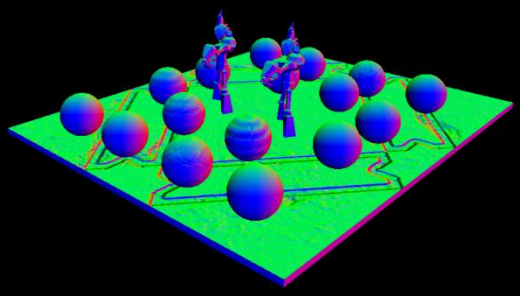
For the GBuffer stage we will allocate two render targets with the format R8G8B8A8 and a depth stencil view with the comparison state of COMPARE_LESS. We will process all opaque geometry in the scene, binding and mapping their material information in a const buffer and writing out the material information in the channels of the render target. The table below shows the material information we will saving in their respected render target channels.

Render Target #	Red	Green	Blue	Alpha
RT0	DiffuseColor.R	DiffuseColor.G	DiffuseColor.B	Metallic
RT1	Normal.x	Normal.y	Normal.z	Roughness

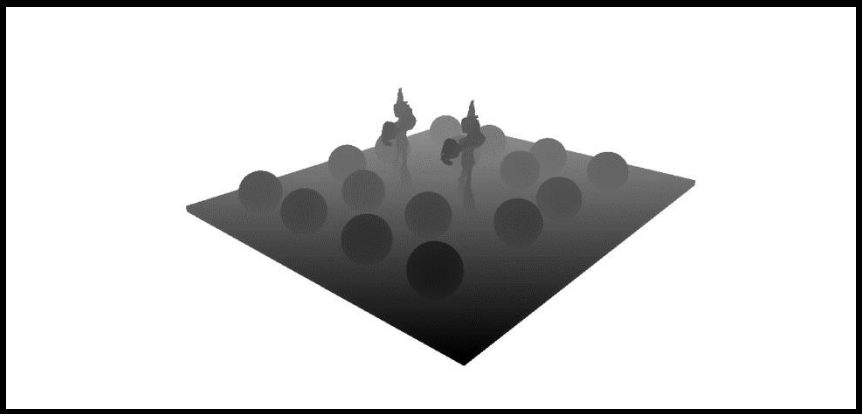
DiffuseColor + Metallic



Normals + Roughness



32-bit Depth Buffer



Lighting Stage:

After all our information has been gathered in our GBuffer we can process our lighting objects in the scene. Each light is represented by different geometry as shown below.

Lighting Equation:

We are using a BRDF lighting model for our light calculation. A BRDF lighting model is a good representation for physically-based lighting. To be physically plausible our equation must have the property of reciprocity and energy conservation.

The equation below is our BRDF diffuse term:

$$Diffuse = \frac{albedo}{\pi}$$

The equation below is our BRDF specular term:

$$f(l,v) = \frac{D(h)F(v,h)G(l,v,h)}{4(n*l)(n*v)}$$

Note: A surface cannot reflect more than 100% of the incoming light energy.

Equation Explained:

The BRDF lighting model has a diffuse and specular contribution. The diffuse part of the equation is just a Lambertian model. The Lambertian model states that all diffuse light is distributed equally about a hemi-sphere. The specular portion of the BRDF can be broken down into three functions. The functions are the micro geometry distribution function D(h), Fresnel reflectance function F(v, h) and Geometry function G(l, v, h). The denominator is a correction factor that accounts for quantities being transformed from local to overall micro facet surface.

For our Distribution function we chose the GGX/Trowbridge approximation.

$$alpha = roughness^2$$

$$D(h) = \frac{alpha^2}{\pi((n*h)^2(alpha^2-1)+1)}$$

The D(h) function determines the size, brightness and shape of the specular highlight. D(h) is a scalar value that as a surface roughness decreases the concentration of micro geometry normal (m) around the overall surface normal (n) increases.

For our Fresnel function we chose the Schlick approximation.

$$F0 = Specular\ Color$$

$$F(v,h,F0) = F0 + (1 - F0)(saturate(1-(v*h)))^5$$

The Fresnel reflectance function computes the fraction of light reflected from an optically flat surface. The function is dependent on the incoming angle between (v) and (h). Note (h = halfway vector).

For our Geometry function we chose a Shlick approximation.

$$G(l,v,h) = G1(v) * G(l)$$

$$G1(v) = \frac{n * v}{(n * v)(1-k)+k}$$

The geometry function is the probability that a surface's points with a given micro geometry normal (m) will be visible from both the light direction (l) and the view direction (v). The function is a scalar value from 0 to 1. This function is essential for the energy conservation part of the BRDF and also applies shadowing and masking at the microscopic level.

$$\frac{active\ surface\ area}{total\ surface\ area}$$

Using our Lighting Equation:

We can now use the BRDF lighting model explained above to calculate our light contribution in our scene depending on the properties of our geometry material information and light type. Our main goal with this deferred rendering technique is to be able to draw three different ranges of materials. Metallic materials such as steel, gold, copper and silver. Rough materials such as concrete, wood and dirt. Finally we want to be able to render anything in between fully metallic and fully rough materials such as plastic, leather, cotton and skin.

Before applying our lighting calculation we need to recalculate our albedo color and our specular color of our object based on the metallic contribution of the material.

$$realAlbedo = diffuseColor - diffuseColor * metallic$$

This equation states that as a material becomes more metallic it loses the diffuse color contribution of the object and becomes fully reflected.

Next we calculate the real specular color of the material.

$$realSpecularColor = lerp(0.03f, diffuseColor, metallic)$$

To calculate the specular color we just lerp from diffuse color to metallic by 0.03f (this is a good value for starting range of dielectrics).

Finally we use all the information stored in our GBuffer and our lighting equations from above to calculate the final color of our lit scene.

$$albedoDiffuse = Diffuse(realAlebdo)$$

$$specular = Specular(realSpecularColor, alpha, dotNL, dotNV, dotVH, dotNH)$$

$$color = lightColor * dotNL * (albedoDiffuse * (1.0 - specular) + specular)$$

Image Based Lighting:

IBL is a rendering lighting technique that uses a cube map to represent the environment to achieve realistic rendering. The main goal for the IBL technique is calculate how much light from the surrounding environment is reflected towards the virtual camera at a given surface. It is impossible to calculate illumination for all direction so we will choose random directions, summing up the result and calculating the average color. We have to calculate the diffuse and specular term for our IBL technique, this produces the amount of light from the environment that effect the scene. For our image based lighting calculation we need to calculate our realAlbedo and realSpecularColor values from the equations explained above. We will first explain the specular calculation portion of our technique. For the specular term we will to iterate depending on our sample count and generate a world normal for each sample with the hammersley sequence, applying our lighting BRDF equation,

calculate the incident light color and divide by probability density function (PDF) to calculate the final average color of that pixel. A PDF is a normalized function that its entire domain is equal to 1.

$$pdf = D * \frac{dotNH}{4 * dotVH}$$

The equation below is the final equation we are solving for after all the necessary variables have been calculated.

$$specularLight \ += \ IndicentLight * (\frac{BRDF}{PDF})$$

To calculate a random direction we first must calculate Phi, cosTheta and sinTheta using the hammersley pair at the specified sample index.

$$phi \qquad = 2 * PI * pair.x$$

$$cosTheta = \frac{sqrt(1 - pair.y)}{(\ 1 + (alpha * alpha - 1) * pair.y)}$$

$$sinTheta = sqrt(\ 1 - cosTheta * cosTheta\ )$$

We then apply spherically mapping using our calculated variables above and also apply two linear transformation that will transform the direction to world-space. The two linear transformation will take the direction vector from specular-space to tangent-space and from tangent-space to world space.

$$H.x = sinTheta * \cos(phi)$$

$$H.y = sinTheta * \sin(phi)$$

$$H.z = \cos(phi)$$

To go from tangent-space to world space introduce a basis formed with the world normal in the GBuffer.

$$Up = abs(normal.x) < 0.999\ ?\ (0,0,1)\ ?\ (1,0,0)$$

$$TangentX = cross(Up, normal)$$

$$TangentY = cross(normal, TangentX)$$

Finally we apply the linear transformation to get the direction from tangent-space into world-space.

$$H = TangentX * H.x + TangentY * H.y + normal * H.z$$

We can now convert the newly calculated light direction in world space into the reflected light direction towards the virtual camera.

$$L = 2 * dotVH * H - V$$

Next we calculate all our angles between each direction vector for our lighting calculation. (dotNV, dotNL, dotNH, dotVH).

The last variable we need to calculate is our incident light color. We sample the environment cube map with our (L) vector and a mip-map level to get our incident light color. We must calculate the right mip level for every sample depending on the roughness of our object and the distribution function D(h).

$$MipIndex = 0.5 * log2\left(\frac{width * height}{SampleCount}\right) - 0.5 * log2(D(H))$$

$$SampleColor = EnvMap.SampleLevel(EnvMapSampler, L, MipIndex)$$

$$IncidentLight \ = \ SampleColor * dotNL$$

Then the final equation operation can now be solved.

$$specularLight \ += \ \ sampleColor * dotNL \ * (\frac{\frac{D * F \ * G}{4 * dotNL * dotNV}}{\frac{D * dotNH}{4 * dotVH}})$$

Which then can be simplified to.

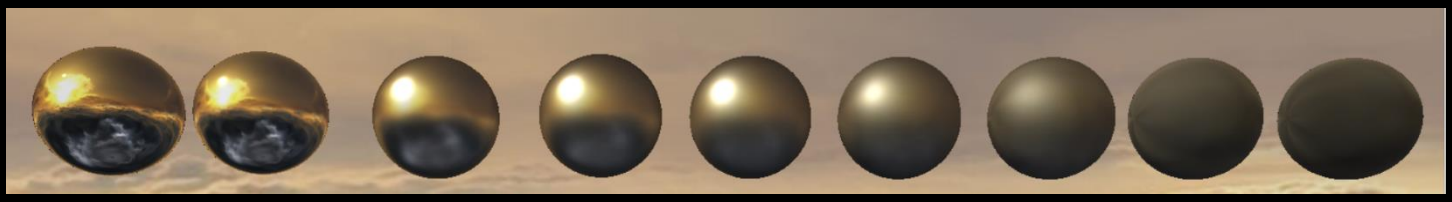
$$specularLight \ += \ \frac{sampleColor \ * F * G * dotVH}{dotNH * dotNV}$$

Lastly once all the samples have been calculated we can find the average by dividing by the sample count. For the diffuse term we just apply the Lambertian function based the real albedo color of the pixel multiplied by the diffuse contribution of the cube map. The diffuse contribution is sampled from an irradiance map that is calculated offline.

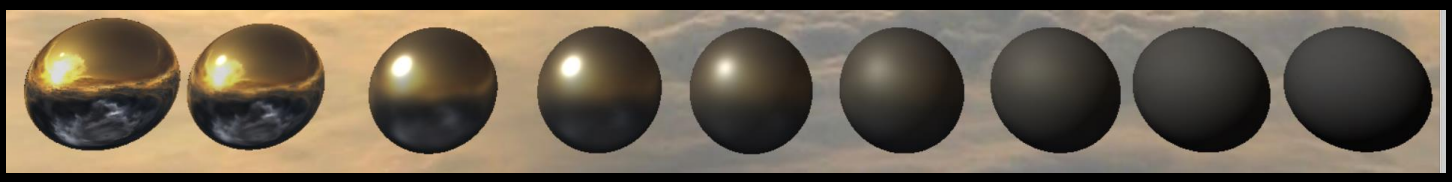
$$Diffuse = Irradiance * \frac{realAlebdo}{\pi}$$

Then we add the diffuse and specular term to achieve the desired result below:

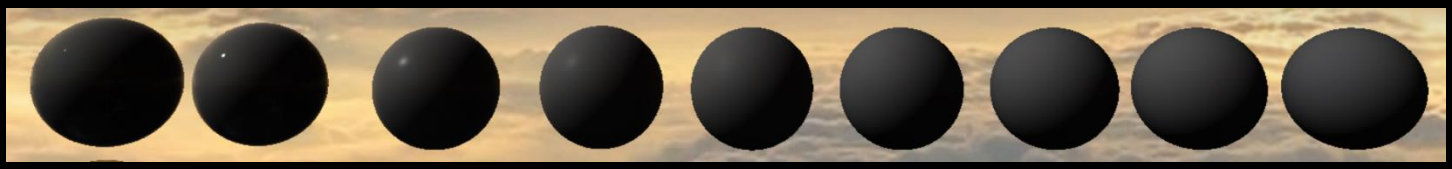
(full metal / varying roughness)



(varying metal / varying roughness)



(no metal / varying roughness)



Conclusion:

In conclusion the deferred rendering technique presented shows how to implement a physical based deferred renderer with imagine based lighting support. Once you have a simple deferred renderer it becomes very easy to change your lighting model and implemented this technique.