Rendering project: Photon mapping glossyness and chromatic dispersion

Goal

It is our goal to improve our photon mapping procedure to take two new aspects into account:

* Glossyness
* Colour Dispersion

Since photon mapping is done in combination with raytracing, the visual effect of a highlight is visible on the spherical surfaces of the cornell box, but it does not cause any weighted photon distribution in the highlight direction, which is known as glossy reflection. This is one of the things we intend to improve in our photon mapper.
Additionally, we intent to add color dispersion, which we hope is a more visually detectable effect than glossyness.
The term chromatic dispersion might sound awesome, but according to colour theory, it does not make sense, since chroma is the saturation of colour, and not the difference, which is registered as hue, so instead of calling it huematic dispersion, we will stick to colour dispersion.

In order to do this we will first clarify on the topics through a theory section which will cover what glossyness and colour dispersion is, and how these mechanics work. Then the method of implementation will be discussed along with the used formulas, and finally the resulting photon mapper will be demonstrated through pictures.

Theory

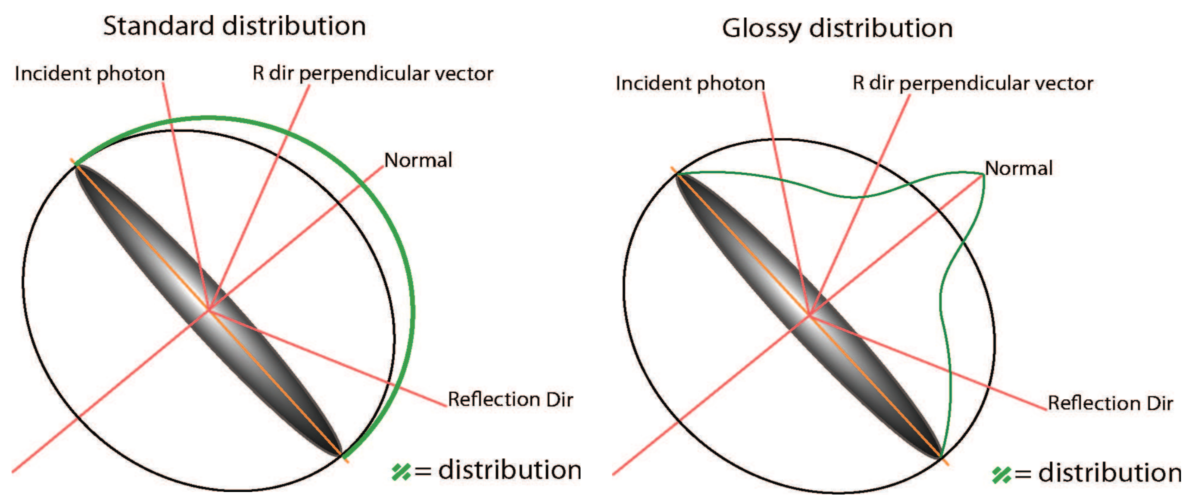
In order to implement the two new features, we must first explore what they are.

Glossyness

In order to add glossyness to photonmapping, it is necessary to perform a different distribution of the diffusely reflected light. The distribution that defines the glossyness is known as a cosine lobe distribution, and a formula has been provided for calculating such a distribution based on two random values between 0 and 1.

The distribution of diffuse reflection with glossyness is pointed along the reflection vector, causing more photons to be shot in the general direction of the reflection.

The difference between normal diffuse and glossy distribution can be seen in figure below.



Instead of generating a random direction on a regular hemisphere we now have to generate a random direction on a unit hemisphere proportional to a cosine lobe. Coordinates on such a hemisphere can be generated with

$$x = \cos(2*\pi*r_1)*\sqrt{1-(r_2)^{\frac{2}{n+1}}}$$

$$y = \sin(2*\pi*r_1)*\sqrt{1-(r_2)^{\frac{2}{n+1}}}$$

$$z = 1-(r_2)^{\frac{1}{n+1}}$$

where *x*, *y* and *z* are the generated coordinates, *r1* and *r2* are random values between 0 and 1 and *n* is the phong exponent.

These can also be expressed in spherical coordinates, but they are not used in this project.

The coordinate we generate should be produced relative to the reflection direction, which means that there is going to be a gap between the cosine lobe hemisphere and the surface. For this gap normal hemisphere coordinates are generated.

The rotation of the cosine lobe hemisphere is an important step, as it directs the glossy light towards the reflection instead of aiming along the normal vector.

Implementation

As stated above we need to differentiate between two different cases: In the first case we are in the gap between the cosine lobe hemisphere and the surface and here we need to calculate normal spherical coordinates. We check if this is the case we check if the dot product between the normal and the reflection vector is above zero; if that is not the case we continue using the the hemispherical coordinate *out.* If that dot product is above 0 (less than 90 degrees) then we need to calculate coordinates for the cosine lobe hemisphere.

When calculating the cosine lobe hemisphere coordinates we first need the two random variables from the formula. The *magic* variable holds a calculation that is common for the calculation of the x and y coordinate and is simply there for convienience. The formula is applied to obtain the three coordinates. Then the coordinate is rotated relative to the reflection vector instead of the world up vector.

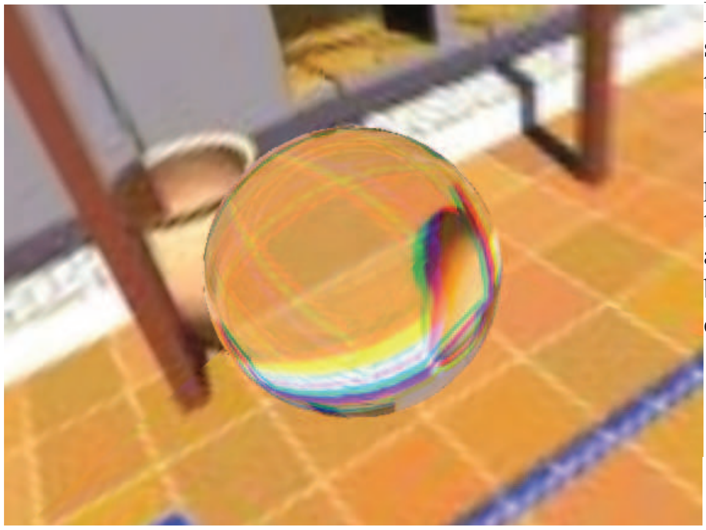
It is possible to generate a coordinate which is beyond 90 degrees from the normal, which is undecierable. Therefore we check for this by doting it with the normal and if the result is less than or equal to zero it is a faulty coordinate and we generate a new one; hence the *do while*-loop.

```
//Glossyness
Vec3f normal = photon_ray.intersected()->get_normal(photon_ray.get_position());
Vec3f refVec = (2*dot((-1)*photon_ray.get_direction(), normal)*normal-(-1)*photon_ray.get_direction());
float dotOutNormal;
if (dot(out, refVec) > 0.) {
        do {
                float r1 = (rand()/((float)RAND_MAX + 1));
                float r2 = (rand()/((float)RAND_MAX + 1));
                float n = photon_ray.intersected()->getSurf()->phong_exponent;

                float hlx, hly, hlz, magic;
                magic = sqrt(1.-pow(r2, (float)(2./(n +1.))));
                hlx = cos(2. * M_PI * r1)* magic;
                hly = sin(2. * M_PI * r1)* magic;
                hlz = pow(r2, (float)(1./(n + 1.)));

                z = refVec;
                x = Vec3f(0., 0., 0.); y = Vec3f(0., 0., 0.);
                orthogonal(z, x, y);
                out = hlx*x + hly*y + hlz*z;
                dotOutNormal = dot(out, normal);
        }
        while (dotOutNormal <= 0.);
}
```

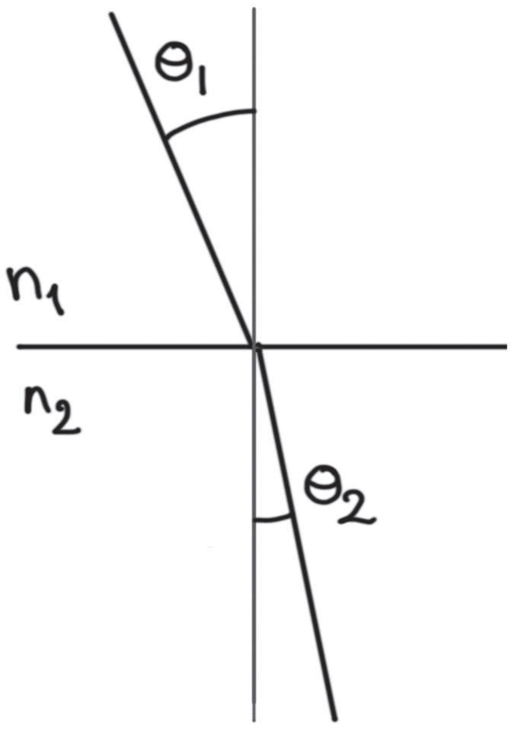
Unfortunately since we did not perform monte carlo integration the scene appeared very course and the it seems to be difficult to see the effect of the glossyness.

Colour dispersion

While it is not as pronounced as with a prism, most materials have a refractive index that varies with the wavelength of the light. The variation in refractive index allows for some pretty colours and it is the "what" that creates rainbows (refraction of light in water droplets).
The refractive index is used to calculate the angle used when transmitting light through a material, just as done with the exercises from the rendering course. So if more colours were present and colour dispersion was enabled, the light would simply be cast in different directions due to the varying refractive index with colours. [T01]



Example of colour dispersion on a glass sphere with one refraction (the backside of the sphere does not refract the light on the picture).
Picture is from a computer graphics project, and in photon mapping we expect the colours to mix once they hit the patches and distribute their colour, so we expect a better variation of colours as a result of the chromatic dispersion.



The refraction angle is calculated as such:

$$\frac{\sin\theta_1}{\sin\theta_2}=\frac{n_2}{n_1}\Rightarrow\theta_2=asin\left(\frac{\sin\theta_1}{n_2/n_1}\right)$$

Where n1 and n2 are the refractive index values of the materials.

Fortunately, this was implemented when performing the photon mapping exercise, and this code can easily be recycled with different refractive indices.
In order to create chromatic dispersion, the light must be split when it is transmitted (and thereby refracted) for the first time.

While in practice, it does not make sense to talk about a white photons, we assume that each photon is a beam of light, and since we cannot represent the entire spectral curve, we just assume that each photon represents the average light distribution of the light source in 3 values, red green and blue.

We must be careful to avoid the same ray being split more than once, so to keep track of the color of light, we will introduce an integer value to keep track of the color of the light. That way if light is white, we can safely split it, but if it is red, green or blue, it should simply be refracted according to wavelength.

Chromatic dispersion only happens when the light is transmitted in our photon mapper, so all that is needed is to introduce a new mechanic in the transmitted light calculation.

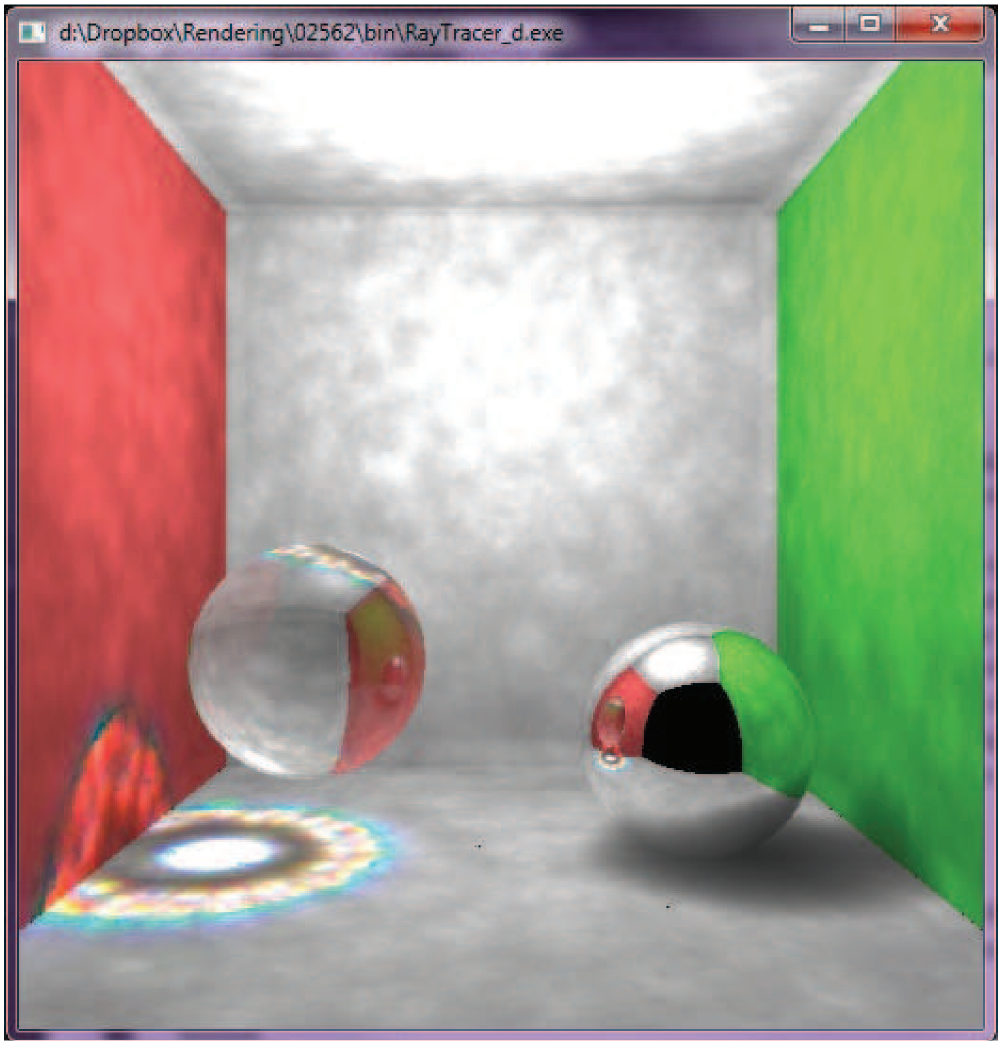
Unfortunately the photon mapper was held together by an infinite loop for each photon. While the infinite loop approach is the easiest to follow in logic, but it introduces some coding issues when more than one sub photon (one for each colour) must be spawned off an existing photon.
To fix this, the code inside the infinite loop has been moved to a new function called photonTracing. The function is meant to be recursively called either when a specific colored photon is to be traced, or when the function reaches EOF. This way, photons can only 'die' from exiting the scene, as intended.
With the new recursive function for photonTracing, we can introduce a calling argument for the color of the photon that is being traced. That way a color will only be applied once, as this is the check that is used to change the color of the photon.

As an invariant, only white photons may have their color changed, and the initially generated photons will all be white.

So with 3 refractive indices, we have decided to use the average of these for the white light refraction.

Results

By combining the above described features, we get the following result:



Further development

The project could be extended be completing the photon mapping algorithm by eliminating the low frequency noise, which is what makes the scene look coarse. This would mean replacing the photon map lookup when shading with Monte Carlo integration of the incident illumination.

This would likely also make glossiness a lot more visible.

The project could also be improved with double refraction would remove looking glass effect that is currently present in the ray tracer rendering performed after photon mapping.

References

[T01] Richard Tilley. *Colour and the optical properties of materials*. Chapter 2. John Wiley & Sons, LTD - 2000.

[ICG] Edward Angel. *Interactive Computer Graphics – A top-down approach using OpenGL.* Chapter 9. Pearson International – 5th Edition.