Fuzzy reflections rendering.

Anastassiya S. Kulikova
Kirill Dmitriev
Moscow State University,
Faculty of Computational Mathematics
Moscow, Russia

Abstract

The method proposed in the article adds the possibility of rendering fuzzy reflections to the existing ray tracing systems. It is based on the idea of special blurring . Depending on the roughness of the reflecting surface, the diffuse component is blurred. Even such special filtering does not require much time, besides it can be optimized to concrete processor architecture. Even realization of the method in C++ (without any assembler code) resulted only in 2 % decrease in speed of the ray tracer.

Keywords: Ray Tracing, *Fuzzy reflections, Rendering.*.

1. INTRODUCTION

Due to lack of literature on this subject I will mention only draft versions of fuzzy reflection implementation. There have been several approaches to rendering fuzzy reflections. The first algorithm perturbs the local normal at surface point where backward ray hit it (in fact, in every screen pixel). Second approach is to trace multiple rays inside of a cone, constructed around the direction of ideal specular reflection, and then calculate average. The latter approach is very slow because too many rays from every point are traced. The first approach is relatively fast, but it is not qualitative because it results in grainy look of surface.

2. METHOD DESCRIPTION

The approach suggested also treats the surface as a set of random micro-facets, assuming that their size is much less than pixel size, thus no *granularity* is visible. Under some conditions this approach is physically accurate (it depends on the reflecting properties of materials); in other cases it is expected to give a good approximation sufficient for a photo-realistic appearance.

This method is implemented via two-pass rendering. The 1st pass is the usual backward ray tracing assuming *specular* reflection. This pass is used to fill in image buffer where all pixel contains:

1. Physical luminance L_0 calculated for perfect specular reflection (because only "specular" component will be subsequently blurred).
2. 3D coordinates of intersection point ***a*** in the glossy surface
3. 3D coordinates of the "end of ray" ***b***, that is the point in the scene which we would see reflected in the pixel where the reflecting surface ideally specular.

In case of the so-called "subsampling" mode, when some pixels are not traced but interpolated, we calculate all the above values with bi-linear interpolation.

In the second pass, the above image is "filtered", that is, for those pixels whose intersection point belongs to the glossy surface,

$$L(x,y) = \frac{\sum_{y',x'} L_0(x',y')\, f(x,x',y,y')}{\sum_{y',x'} f(x,x',y,y')}$$

where $L_0(x,y)$ is the luminance of the original image at pixel (x,y), and $L(x,y)$ is the resulting luminance which represents fuzzy reflections. Here $f(x, x^t, y, y^t)$ are weight coefficients depending on the material properties and on the ϑ angle. As in our renderer the reflective properties of material were characterized by the shininess and shin strength, the following formula were suggested

f = 2 * ShinStrength * pow (cos (ϑ), Shininess) (2)

In renderers where specular characteristics of material are determined by Phong coefficient (glossiness), the following formula is suggested

f = pow (cos (ϑ), p) (3)

Where p is Phong coefficient and $\vartheta(x,x',y,y')$ is the angle between direction from intersection point to the end of "its" ray (=specular ray) and ray fired from this point to the end of ray for ***neighbour*** pixel (x',y'):

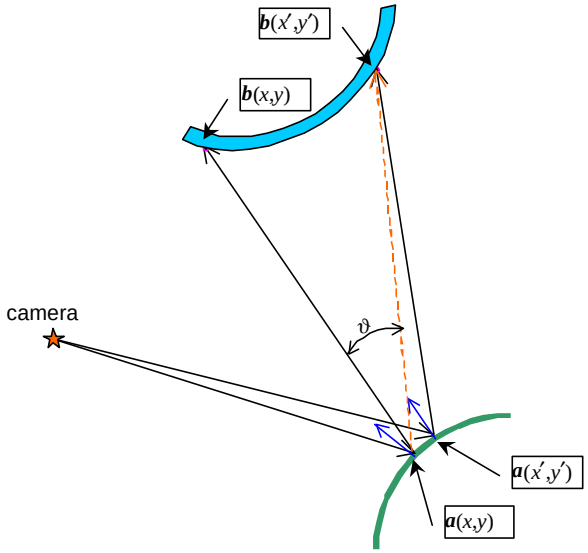

Figure 1: The thick green line is glossy reflector; thick blue line is the object reflected in it; blue arrows show local normals and solid black arrows show specularly reflected rays which were traced in the 1st pass; the dashed arrow is the ray representing fuzzy reflections and ϑ is the angle between it and specular direction.

From the Figure 1 one can calculate that

$$\cos\vartheta(x,x',y,y') = \frac{\big(\boldsymbol{b}(x',y') - \boldsymbol{a}(x,y), \boldsymbol{b}(x,y) - \boldsymbol{a}(x,y)\big)}{\big|\boldsymbol{b}(x',y') - \boldsymbol{a}(x,y)\big| \cdot \big|\boldsymbol{b}(x',y') - \boldsymbol{a}(x,y)\big|} \quad (3)$$

We assume that the reflecting surface is glossy, so that nearly all reflected energy contains in cone $\vartheta \le \Theta < 10°$. In this case contribution of far pixels is negligible, and we can confine the sum in (1) to the neighbourhood of pixel (x, y). The latter comprises pixels (x', y') such that the rays fired to them from camera deviate from ray fired to the central pixel (x, y) be angle less than

$$\alpha = \Theta/(1 + s/r) \quad (4)$$

where s is the distance from camera to intersection point on glossy surface (point $\boldsymbol{a}$) and r is the distance from the latter to the end of ray (point $\boldsymbol{b}$). From the angular size we can easily estimate the radius in pixels:

$$\rho = \alpha \times (\text{image size})/(\text{view angle}) \quad (5)$$

Thus, we obtain the following formula for determining the blurred color

$$L(x,y) = \frac{\sum_{|y'-y|<d, |x'-x|<d} L_0(x',y') f(x,x',y,y')}{\sum_{|y'-y|<d, |x'-x|<d} f(x,x',y,y')} \quad (1')$$

(or equally we can use round instead of rectangular area)

The filter function is evaluated at the centres of pixels. It would be better to evaluate $f(x,x',y,y')$ as *average* over pixel, but that is too expensive.

3. BASIC OPTIMIZATIONS

When realized as described above the algorithm considerably decreases rendering speed. The following optimizations were made . They do not lead to drawbacks in quality, but allow for fast rendering.

The filtering (1) may be expensive in case of large filter size. The following means can be used to accelerate that:

a) Force restriction on the filter size

b) Adaptive interpolation is possible. We can "blur" the image ignoring antialiasing. While blurring we do not split pixel in subpixels.

c) The weight coefficients can be tablulated on a regular mesh. It is done on the first pass of renderer. Then the calculation of arccos is obviated, as well as e.g. raising to power γ in Phong model.

d) While testing the preliminary implementation it was observed that the coefficients calculated in the above-mentioned way are similar to Gauss kernel. Therefore it is possible to use Gauss convolution. The subtle differences in images are usually not seen by pure eye.

e) The filter size can be calculated at each fourth pixel, because it is reasonable to assume that the angles does not change considerably between adjacent pixels. The procedure for calculating filter size can also be simplified.

3.1 RESULTS

Figure 2: Sample scene, no fuzzy reflections

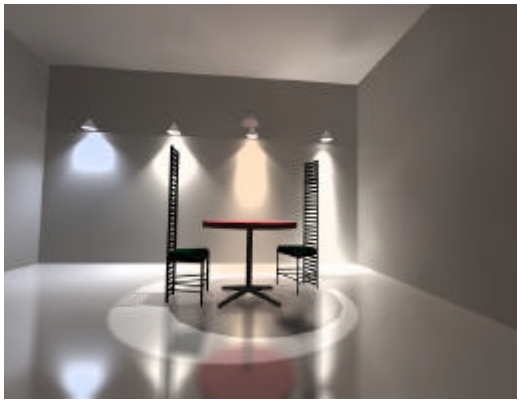

Figure 3: Sample scene, surface roughness 50%.

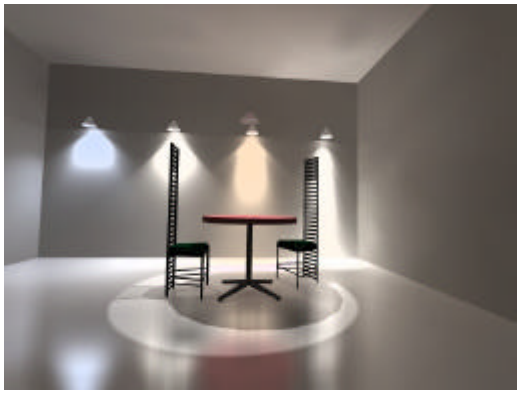

Figure 4: Sample scene, surface roughness 100%.

4. CONCLUSION

The advantages of the method are its speed, physical accuracy. The algorithm can be added to any renderer as a second pass. The following limitations are inherent in the method.

Reflections after reflection by the first encountered glossy surface are not handled. That is the method can not accurate handle the case when e.g. glossy surface reflects ideal mirror which in turn reflects something. This is because the method

assumes that rays form intersection point till ray end are straight lines.

Similarly, if one fuzzy object is reflected in other fuzzy object, the fuzziness of furthest object (first one) will be ignored. This assumption looks reasonable since fuzziness of primary reflection should hide sharpness of secondary one any way.

About the author

Anastassiya S. Koulikova is the student of Computational Mathematics Faculty.
E-mail sergevna@mailru.com

Kirill Dmitriev
kadmitr@gin.keldysh.ru