CS 348B Final Report

Yao Chen, Xinru Hua
yaochen1@stanford.edu, huaxinru@stanford.edu

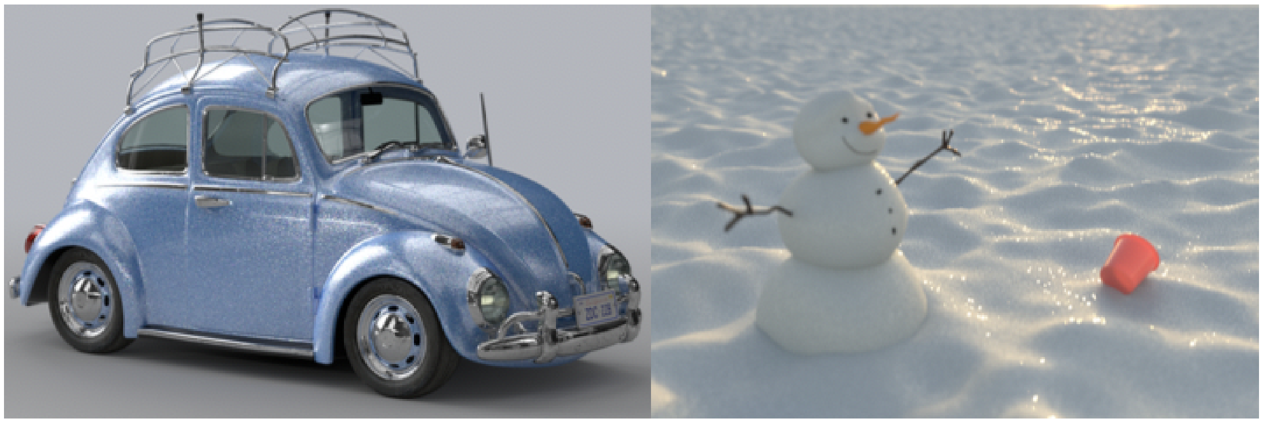

Figure 1: Target image: glittery beetles car (left), shiny snowman scene(right)

1 Introduction

We want to render an image that contains objects like snow or beetle toy as shown in Fig 1 which exhibit small-scale phenomena observed as bright sparkling or glittering surface features as shown in the above images. The beautiful glittering effect under narrow angle illumination could be achieved by implementing the microfacet-based surface model that contains a large finite number of mirror-like flakes.

Our method is based on Atanasov and Koylazov's paper "A Practical Stochastic Algorithm for the Rendering of Mirror-Like Flakes" from Siggraph 2016 [1]. This paper improves a previous stochastic algorithm of rendering mirror-like flakes presented in Jacob's paper [2]. The new method in Atanasov's paper introduces a memory-efficient traversal and applies better importance sampling, and also outputs as good results as the original paper. Generating flakes is hard, computation and storage space expensive, since we want the number of flakes and their normals consistent for all the positions on the surface. The previous method has a 4d hierarchy data structure storing all these information, and that slows down the rendering process. The new paper fastens the process by only calculating the number of flakes and generating random normals.

We created two class in pbrt, a new material class in PBRT called Flake, and a new BXDF class called FlakeBxDF. Class Flake would use the surface interaction, takes in parameters of roughness, gamma for reflection cache, approximate the number of flakes for the interacting ray, and then pass these parameters to the constructor of Glitter. Class Glitter inherit the BxDF class, we overwrite the f, sample f, PDF function to implement the multi-scale discrete microfacet distribution and optimal importance sampling.

2 Method

2.1 Texture Query

We want to estimate the number of normals for each ray when it hits the surface. This could be done by a texture query on footprints A, using a quad-tree, stack-based, depth-first traversal. We first use **dudx**, **dudy**, **dvdx**, **dvdy** to calculate the footprint parallelogram A of a ray. The ray's uv coordinate is drawed from surface interaction si. For example,

$$\mathbf{P_1} = si.x - dudx * \epsilon - dudy * \epsilon$$

$\mathbf{P_1}$ is the upper-left corner of A, as shown in Fig.2; and $\epsilon = 0.01$.

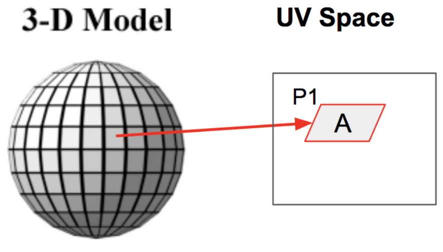

Figure 2: UV Transformation

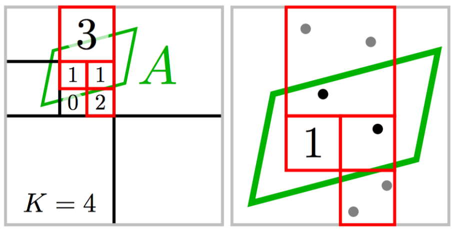

Figure 3: Quad-Tree

We then subdivide the unit texture square and traverse the space by a quad-tree. As shown in Fig.3, the red boxes are the quads, and the total flake number in A is 3. In our case, the space has $\mathbf{N} < 2^{31}$ flakes in total and is subdivided into 4 quads until hitting a leaf. The probability of a flake falling in each quad is a multinomial distribution with $\mathbf{p} = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$. The maximum depth of traversing ($\mathbf{K}$) is 15 and a leaf ($\mathbf{T}$) is a quad that has less than 16 flakes. When implementing this, we used a normal distribution to approximate the multinomial distribution. The data structure is a quad-tree; flakes are kept randomly generated into each quad and the quads that do not intersect with footprints are ignored. We also consider the probability of {flake falls in A given it is in the quad $= \mathbf{n_A}$}, and it is calculated as

$$p(\mathbf{n_A}) = \frac{area_{footprint}}{area_{quad}}$$

.

When implementing the top-down texture query, it traverses to the leafs and we want to find the number of the overlapping bounding boxes. We first bound the parallelogram by a bounding box (a square), and then calculate how many quads that this bounding box would overlap with. For these quads overlapped, we approximate flake numbers in each quad by a normal distribution and add the numbers up. Then the flake number in parallelogram is the flake number in bounding box times the ratio of area(A) and area(boundingbox).

After querying the texture, we pass this number to the Bxdf constructor. With the queried number, normals and normal's positions are all randomly generated. This saves the storage space of a 4d hierarchical data structure of the normals and their positions, compared to Jacob's paper[2].

2.2 Discrete BRDF Model

Like the microfacet model, we treat the surface as a collection of randomly oriented microfacets. Different set of flakes is seen at each pixel, which results in the glittery surface. Texture query now gives us an approximate number of microfacet surfaces for a footprint and it would serve as an start

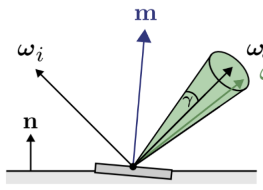

Figure 4: Planar view for a microfacet

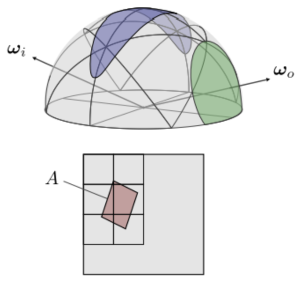

Figure 5: Texture Query and microfacet reflection

point in justifying the specific subset of flakes that actually reflect from a source to a receiver under a particular illumination and viewing geometry.

Jacob et al. [2] proposed a new BRDF model for Multiscale microfacet distribution as follows:

$$f_r(A, \omega_i, \Omega) = \frac{(\omega_i \cdot \omega_h) F(\omega_i \cdot \omega_h) \hat{D}(A, \Omega_h) G(\omega_i, \omega_o, \omega_h)}{a(A) \sigma(\Omega_o)(\omega_i \cdot n_x)(\omega_o \cdot n_x)}$$

Where F is the Fresnel term, D is microfacet distribution, G - shadow-masking function i-incident light direction, o - reflected light direction, h = $\frac{\mathbf{i+o}}{|\mathbf{i+o}|}$, n - the surface normal [3]. It models the radiance that region A (marked in red) scatters into a solid angle around ω_o marked in green in Fig 9 and 3.2. For a particle to reflect light from ω_i into the cone of radius γ around ω_o, its normal must lie in the highlighted blue set. The smaller the angle γ, the fewer particles will be found, though the $\sigma(\Omega_o) = \pi(1$ - $cos\gamma)$ term in the denominator of f_r compensates for this by weighting each particle higher.

In our implement, we only take into account Fresnel term F and microfacet D, since bidirectional shadowing mask function G has relatively little effect on BSDF shape except on near glazing angle on very rough surfaces. Fresnel term would be adjusted according to the material.

2.3 Microfacet Distribution D

Since the surface is made up of randomly oriented microfacets, distribution could be calculated as

$$D = \frac{4}{a(A)\sigma(\Omega)(\mathbf{i} \cdot \mathbf{h})\hat{D}\cdot}$$

It denotes the fraction of flakes which are contained in A and reflect o in omega. $(\mathbf{i} \cdot \mathbf{h})\hat{D}$ could be accurately approximated as $\tilde{D} = \frac{1}{N} \sum \mathbf{i} \cdot \mathbf{m}$, m_j is the normal of flakes, $\mathbf{N}$ is total number of flakes.

To compute D delta, we first randomly generate n microfacet normals for n footprint flakes accumulated from the texture query respectively. By the law of specular reflection, microfacet normals $\mathbf{m}_j$ would reflect light along the outgoing direction ω_j into a cone of radius γ centered at $\mathbf{r}_j$. One important thing is to check the validity of r_j, flip the direction if $\mathbf{r}_j$ and ω_o are not in the same hemisphere. At the same time, we define per-flake weight as $w_j = \mathbf{m}_j \cdot \mathbf{r}_j$. The reflected direction r_j, along with the weight w_j form a reflection pair (r_j, w_j), we store all n reflection pairs in a reflection cache. Then we compute an accurate approximation of microfacet distribution which contains the incident light direction $\mathbf{i}$.

$$\tilde{D} = \frac{1}{N} \sum_{\mathbf{i} \cdot \mathbf{r}_j \geq cos\gamma} \omega_j$$

2.4 Optimal Importance Sampling

To effectively use our new BRDF model in a modern rendering system, we require a way of sampling directions from a distribution that is close to the our BRDF model. To do so, we made use of the reflection cache which contains n reflection pair $\mathbf{w}_j$, mentioned in Section 2.3 to build a cumulative weight table containing n + 1 elements.

$$c_o = 0, c_j = c_{j-1} + w_j, j = \overline{1, n}$$

We randomly choose $\zeta \in (1, c_n)$, then binary search the cumulative weight table to find the target weight and its corresponding reflection cone direction based on the reflection cache. You could also interpret the cumulative weight table as a line containing n segments of weights, segment length (weight) denotes the contribution of each reflected light cone in generating scattered radiance. The longer the segment, the larger the possibility of the reflected cache direction being chosen. We then generate the BRDF $\mathbf{i}$ direction by choosing a uniform random direction inside the chosen reflection cone of radius γ. Since the solid angle of the visible cone is $\sigma = \pi(1 \text{ - } cos\gamma)$, the geometric probability is $\frac{1}{\pi(1-cos\gamma)}$. The probability of generating i would be:

$$p(\mathbf{i}) = \frac{k}{\pi(1 - cos\gamma)c_n}, \quad k = \sum_{\mathbf{i}\cdot\mathbf{r}\geq cos\gamma} \omega_j$$

Optimal importance sampling enables us to efficiently sample wide and heavy-tailed microfacet distribution.

2.5 Combining Smooth and discrete BRDF

When just a few facets participate, they would appear strong glittery appearance due to considerable variation from pixel to pixel. When the number of flake increases or a large amount of flakes shares aligned direction, variation would be less drastic. As a result, surface reflectance appears smoother and more like a traditional BRDF and it would be computationally expensive. Therefore, we combine smooth BRDF and discrete BRDF in our model. If the number of flake exceeds a value, we would compute only the discrete microfacet distribution. Otherwise, we would implement the smooth microfacet distribution. Combining Smooth and discrete BRDF helps us improve the efficiency.

2.6 Color Table

To add more color details to the final image, we create a color table where each flake could randomly pick a color from the table. There are 60 colors in our table and they are spanning the hue parameter in HSL space (Hue, Saturation, Lightness). H is sampled within range $[0, 360]$ degree uniformly; in our case, it is $6 * [0, 59]$, so it is within $[0, 354]$ and spanning every 6 degree. S and L are both fixed. For the scene of Christmas balls, we use $S = 1$ and $L = 0.7$. After generating a random color in HSL, the color is transformed to RGB space and be assigned to the flake.

In last step of distance blending, we switch between our discrete microfacet distribution and a continuous distribution. Here we use a single color for flakes in the discrete distribution and multiply the smooth BRDF by the average color in the table.

When rendering scenes, we want the color of shiny flakes more similar to the color of the objects. For example, in the scene of Christmas balls, if the color of the ball is red, we decrease the probability of glitters being green, so that the ball will not look too noisy.

3 Result

We designed several experiment to investigate the and also to validate our implement.

3.1 Color and Color Table

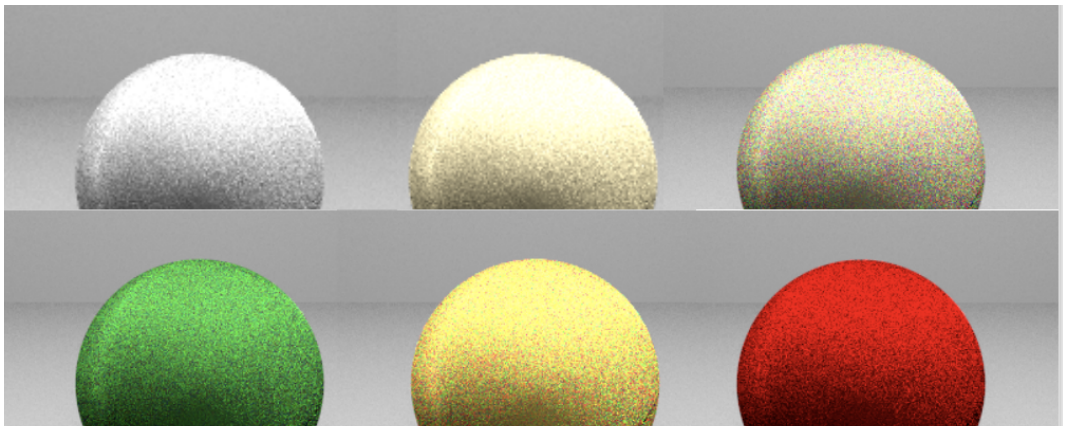

Figure 6: result: Color spheres

Fig 3.1 gives six images of our glittery spheres.

In the first row, comparison between the first and second sphere represents the difference of rendering a glittery surface without and with the specific color. The implementation of color table added more color details to the glittery effect as shown in the third sphere.

The second row gives three beautiful glittering color spheres implemented using the designed color table.

3.2 Visible Angle γ

We rendered the ball with different γ value. From the paper, larger γ means larger reflection cache cone, thus more flakes . The comparison of the images below confirmed that.

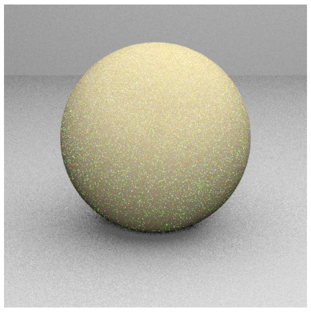

Figure 7: $\gamma = 2$

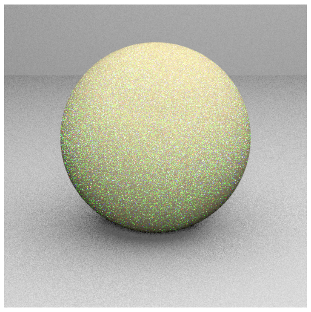

Figure 8: $\gamma = 5$

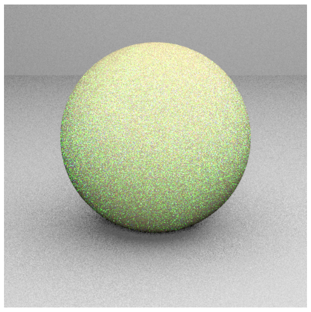

Figure 9: $\gamma = 10$

3.3 Final Image

Our final image is given in Section 5. It took us 855s to render the image on a 4-core 2.6 GHz Intel Core i7 CPU.

4 Challenges

- When writing the transformation from world space to uv space, the four nodes of parallelogram A will be flipped. For example, I am calculating lower-left node, lower-right node in world space and transform them to uv space. The transformation is not affine, so that the two nodes switch position in uv space. Later I add absolute value when I calculate area of A.

- There is problem when depth is too large. After traversing, there is 0 flake number in a leaf. We also noticed that and tuned the parameter depth T and total flake number N.
- Debugging the bxdf class took a lot of time. We went into the error check failed many times. We spent some time printing and reading the return values of function f, samplef. After talking with Feng, we realized we need to write many checks to make sure the functions do not return negative values for both PDF and sample function, like probability<0 would lead to check failure.

5 Final Image

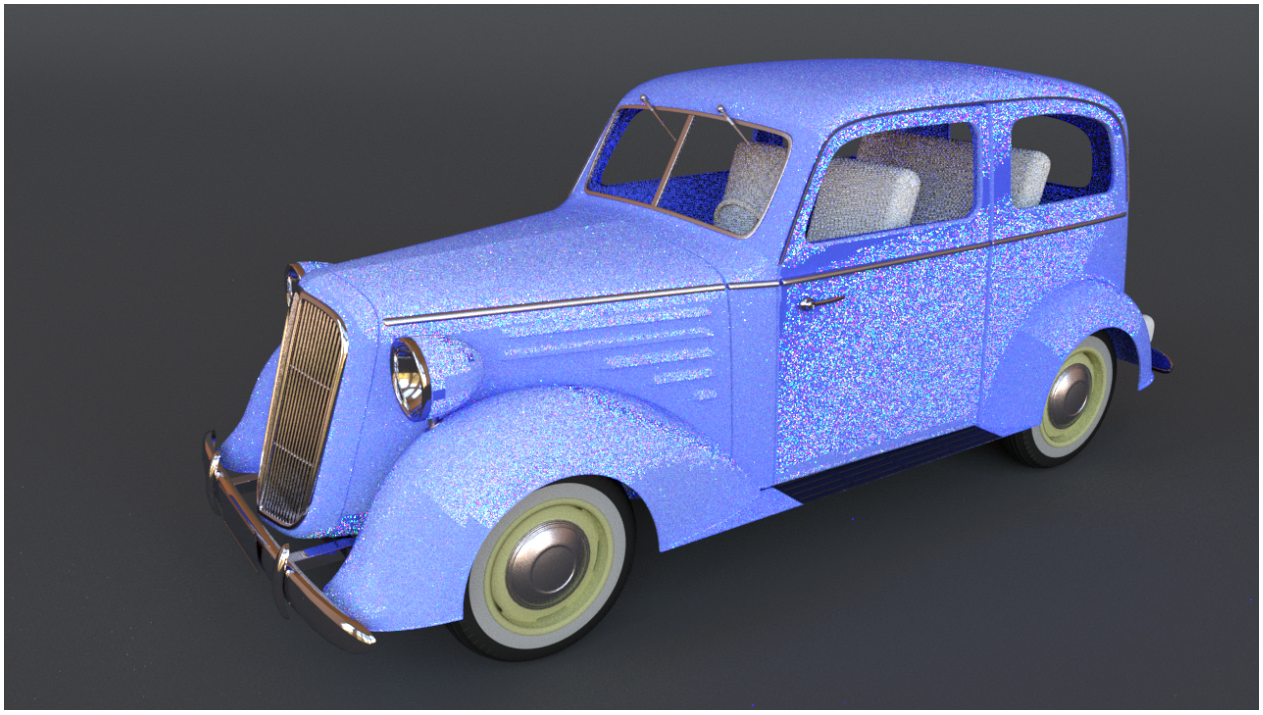

Figure 10: Final Image

6 Contribution

- Xinru: Implemented class Flake (texture query and color table).
- Yao: Implemented class FlakeBxDF (microfacet distribution, optimal importance sampling and distance blending).

7 Acknowledgement

We really appreciate the time and help from professor Hanrahan, professor Pharr, and course assistant Feng. Especially, Feng explained us the paper in details and gave us a lot of tips on debugging. With all the help, we could finish our project and render such a great final scene.

8 Link to Code Repository

https://github.com/EmmaYChen/beetles-carcar

References

[1] A. Atanasov and V. Koylazov, “A practical stochastic algorithm for rendering mirror-like flakes,” in *ACM SIGGRAPH 2016 Talks*, p. 67, ACM, 2016.

[2] W. Jakob, M. Hašan, L.-Q. Yan, J. Lawrence, R. Ramamoorthi, and S. Marschner, “Discrete stochastic microfacet models,” *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 115, 2014.

[3] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance, “Microfacet models for refraction through rough surfaces,” in *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pp. 195–206, Eurographics Association, 2007.