Fundamentals of Computer Graphics
Final Project

Désirée Bellan

1 Introduction

The following report describes my final project for the course " Fundamentals of Computer Graphics" held by prof. Pellacini Fabio during the accademic year 2020/2021.

The project consists in the implementation of two extensions for the Yocto/GL library: the first one is an hair shader built upon the work of Matt Pharr [8], the second one a geometry processing tool based on Monte Carlo estimators [10], used to solve Partial Derivative Equations (PDEs) in a convenient and intuitive way and developed as an alternative to other mesh-based techniques, such as the Finite Elements Method (FEM).

2 Hair Shader

In order to correctly visualize thin hair in a 3 dimensional rendering, it's necessary to compute how the light sources present in the environment interact with its surface, in other words it's necessary to build a *shader*. Bidirectional Scattering Distribution Functions (BSDF) are exactly used for this purpose. They have been conceived to be able to combine BRDF and BTDF - implied respectively in computing the reflected and transmitted light from the surface -, which is ideal when dealing with organic structures as this as hair.

As will be analyzed in depth in the following sections, the scattering function heavily depends on the geometry of the hair, which has been approximated to reduce the computation complexity while maintaining the most relevant properties of it surface and volume.

Since the code implementation have been consistently deduced from the reference paper [8] and only minor changes have been performed, mainly to adapt the authors framework to the one used in `Yocto/GL`, the work done was focused on testing the code on various application and scene, as can be seen in section **Results**.

Finally, the authors provided some test function to ensure the accuracy of the model. While those functions have been implemented and coherently used, they are not discussed in this paper to leave space to the the model description and simulations results.

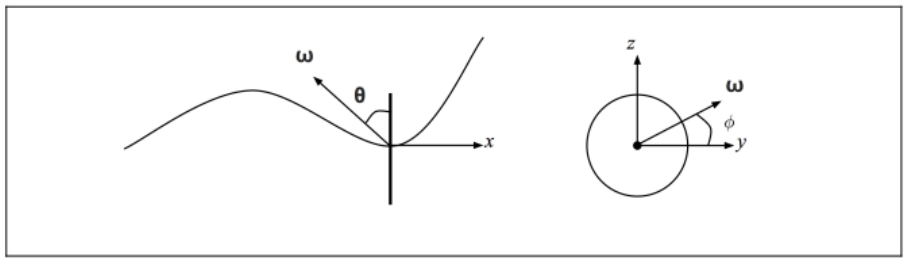

Figure 1: Geometrical Parametrization of incident rays over the hair surface.

2.1 Hair Geometry

Each hair is represented as generalized cylinders, created by spanning a circle, representing the volume of the hair, along a Bezier curve, representing instead its length and curvature. However, for approximation purposes the hair surface is treated in the same manner as a flat ribbon always facing the incident ray. Since in most cases the diameter of the hair is smaller, or comparable, to the pixel width, there is no lost in informations or generality.

Having to define the interaction between light rays and the hair, a parametrization system is needed, so to simplify the algorithm. On the surface incident rays are parametrized with respect to the plane normal to the hair section, called the *normal plane*. The authors used two angles : (i) θ, called *longitudinal angle*, expressing the orientation of the scattering ray ω w.r.t. the *normal plane*, computed in the intersection point, and (ii) ϕ, the *azimuthal angle*, representing the angle aperture of the projection of ω onto the normal plane (see 1). To parameterize the interaction of the ray with the volume is introduced also the parameter h (see 2).

From this first easy parametrization we are able to define hair scattering properties by analizing their organic structure. It usually can be defined by a layered model composed by three sections : (i) the *cuticle*, which is the outer layer, (ii) the *cortex*, the middle and thickest portion, mainly involved in color emission and absorption, (iii) the *medulla*, the inner layer, the most relevant when it comes to light scattering.

To derive a useful model from this definition two main assumption are made : (i) the cortex can be modeled as a dielectric cylinder traversed by homogeneous scales having a constant angle α w.r.t. the surface and (ii) the second and third layer can be condensed into a light absorbing medium (scattering here is not implemented directly).

Finally, in addition the the discussed parameters, other ones are taken into account, mainly to represent the hair roughness and absorption properties : (i) *eta* represents the index of refraction in the cortex and medulla, (ii) σ_a defines instead the light absorption of those internal layers, the roughness is defined on the other end by β_n and β_m, along the azimuthal and longitudinal angle respectively.

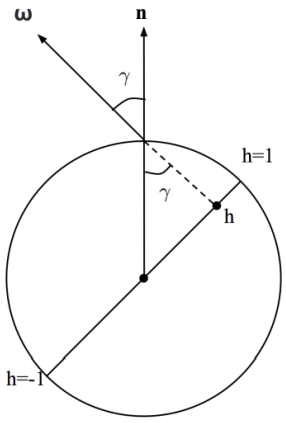

Figure 2: The parameter h indicated the intersection between the ray ω and the hair diameter over the normal plane.

2.2 Scattering Model

Using this preliminary knowledge it's possible to build a semi-seperable scattering model, represented by the equation 1.

$$f_p(w_o, w_i) = \frac{M_p(\theta_o, \theta_i) A_p(w_o) N_p(\phi)}{|\cos \theta_i|} \tag{1}$$

Here w_o and w_i are the outgoing and incoming rays, respectively, M_p is the *longitudinal scattering* component, which depends on the angles θ of the ray w_o and w_i, $A_p(w_o)$ is the attenuation function, accounting for light absorption by the ineer medium, and N_p is the *azimuthal scattering* function. Each scattering is repeated p times, to account for ray transmittance and reflection inside the medium, and ultimately the p contributions are summed to obtain the final result.

Longitudinal Scattering. Influence the specular lobe along the length of hair, the size of which depends on the longitudinal scattering roughness. It's computed here following d'Eon et al. [2] implementation (see equation 2), where $I_o()$ is the modified Bessel function of the first kind [6] and v accounts for the longitudinal longitudinal variance from β_m at each step p. In this case we assume that reflection and transmission are perfectly specular, assumption derived from glossy trasmission.

$$M_p(\theta_o, \theta_i) = \frac{1}{2v \sinh 1/v} e^{-\frac{\sin \theta_i \sin \theta_o}{v}} I_0 \left(\frac{\cos \theta_o \cos \theta_i}{v} \right) \tag{2}$$

Absorption in fibers. The attenuation function A_p accounts for the quantity of light affected by each scattering modes p. It heavily depends on the

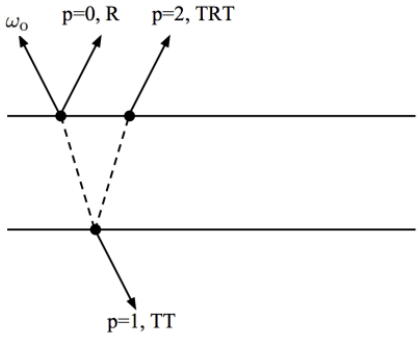

Figure 3: Representation of the p scatteering modes of the light ray inside the hair, derived from sequence of transmittion and reflection.

qualities of the internal layers of the hair, consequently it affects its color.

A_0 corresponds to the light reflected on the outer layer and is therefore equal to the Fresnel dielectric refraction, computed in `Yocto/GL` with the function `fresnel_dielectric(eta, 0, 0, 1, {0, 0, cosTheta});`, that take as arguments a parameter *eta*, the normal to the surface (in the hair frame) and the outgoing ray, here computed though the cosine of θ.

A_1, corresponding to the term $p = 1$ (see 3) depends on two transmittance inside/outside the hair, computed as $(1 - fresnel)$ and a trasmittance T inside the hair. T depends on the distance traveled by the light inside the hair, which can be efficiently computed using the projection of the ray on the longitudinal and azimuthal planes, namely using the angles θ_t and γ_t, respectively. Having those two terms, obtained by applying the Snell's Law and the refraction index, T is derived from the equation 3, where $l = 2cos(\gamma_t)/sin(\theta_t)$ is the distance travelled inside the hair and σ_a is an absorption parameter that can be either given with the model or derived from the eumelanine and pheomelanine parameters, which in turns defines the hair color, weighted by a concentration or a reflectance parameter.

$$\mathcal{T} = e^{-\sigma_a l} \tag{3}$$

$$A[1] = sqrt(1 - f) * T \tag{4}$$

Since the length of the travelled path inside the hair remains constant, we can reuse the term T to compute all the successive p modes of A (see 5). To account for all the infinite possible paths, we compute the sum of the infinite series and store it inside $A[p_{max}]$ (see 6). The choice of p_{max} determines the accuracy of the model, but fortunately it converges very quickly, so $p_{max} = 3$ is perfectly fine for a good quality result.

$$A[p] = (1 - f)^2 * T^p * f^{p-1} \tag{5}$$

$$A[p_{max}] = A[p_{max} - 1] * f * T/(1 - T * f) \tag{6}$$

Azimuthal Scattering. This model is based on two steps : (i) compute an azimuthal direction (always assuming specular reflection and transmission) and (ii) define a distribution of directions around the center.

(i) The change in direction is given by a function Φ that depends on the angles γ_o (azimuthal angle of the reflected ray) and γ_t (angle of the transmitted ray), both computed using the parameter h previously defined, and the mode p (equation 7

$$\Phi(p, h) = 2p\gamma_t - 2\gamma_o + p\pi \tag{7}$$

(ii) The contribution of the center to the scattering depends on a roughness parameter that can be evaluated using the *Logistic Function*. In particular, the authors used a modified version of said distribution function, normalized and defined over a finite range, called *Trimmed Logistic*. Inn equation 9 $L(x, s)$ is the logistic function, depending on a scaling factor s (derived from the roughness parameter β_n), while $L_{CDF}(x, s)$ is the cumulative distribution function of L.

$$TL = L(x, s)/(L_{CDF}(b, s) - L_{CDF}(a, s)) \tag{8}$$

Finally Np term is obtained by applying the trimmed logistic distribution to the angular difference between ϕ and Φ.

2.3 Importance Sampling

Similarly to the approach used in `Yocto/GL`, the distribution sampling is crucial also for the hair scattering model. To increase the number of samples provided, the authors used a method to extract two samples from each incoming sample, used in the function `compute_ap_pdf()`, which returns a discrete PDF with probabilities for sampling each term A_p according to its contribution relative to all of the A_p terms, given the outgoing ray longitudinal angle θ_o. This is accomplished by extracting the luminance from the A_p values and then normalize it. While the value sampled from A_p is an estimate of the actual distribution, M_p and N_p can be sampled directly.

2.4 Results

In this section are illustrated some of the results obtained through the algorithm described, using 3D models of straight and curly hair.

Firstly it's possible to appreciate how the material parameters affect the rendering results, in quality, realism and variety of variety. When is not otherwise specified, the parameter values are :

- $\alpha = 2$
- $\beta_n = 0.3$
- $\beta_m = 0.6$
- eta = 1.55

Figure 4: Straight hair with different values of α. Left : $\alpha = 2$, right : $\alpha = 0$.

Figure 5: Curly hair in different colours

In figure 4 are compared two values of α, defining the scales angle on the hair cuticle. As it can be observed, higher values of α provide a higher light scattering effect, which is more similar to the real result, while a lower values generate a dull velvety effect.

In figure 5 has been tested a range of 3 σ_a values, which affect the absorption and consequently the color of the hair. In figures 6 and 7 have been tested how different values of β_m and β_n, respectively, affect the results. Finally, the straight hair model have been tested on a common hair-less model know as "bald man". As it can be seen the result is quite natural even at a relatively low resolution.

Figure 6: Curly hair with different values of $beta_m$. Left : $beta_m = 0.1$, middle : $beta_m = 0.3$, right: $beta_m = 0.9$

Figure 7: Curly hair with different values of $beta_n$. Left : $beta_n = 0.3$, middle : $beta_n = 0.6$, right: $beta_n = 0.9$

Figure 8: Bald man model with straight hair.

3 Monte Carlo Geometry Processing

In computer graphics many complex problems are commonly solved though Partial Differential Equations (PDEs), which contitue a powerfull tool but rise two fundamental issues : (i) they usually requires heavy computational preparations of the model to be solve, mostly consisting on *mesh generation*, which can be quite computationally expensive, particularly when dealing with finite and complex models. (ii) suffer from unavoidable approximation error. Traditionally, PDEs are solved using Finite Element Methods (FEM), which are extremely robust and constantly improved, but remain trapped inside the constraints of these two limits.

In 2020 R. Sawhney and K. Crane [10] proposed a Monte Carlo based alternative to FEM that promised to solve both problems, while opening possibilities to many exciting fields, not only in computer graphics.

The goal of this project has been to implement and test the approach proposed by the authors on some of the applications proposed, in order to appreciate the advantages and qualities of the method.

3.1 Walk on Spheres

Even though monte carlo based methods have been existed and applied for decades, their field of application seemed to be limited to rendering, while in contrast the research around their use in geometry processing is scarce.

The conception at the base of the method is that an integral of an integrable function f over a confined domain Ω can be approximated by the expected value of the weighted sum of the function over N random points inside the domain (see eq. 9, where $\mathcal{X} \sim \mathcal{U}(\Omega)$ is a random sample over a uniform distribution).

$$I = \int_{\Omega} f(x)dx \to F_N := |\Omega| \frac{1}{N} \sum_{\Omega} f(\mathcal{X}_\rangle) \tag{9}$$

The error of F_N represents the *variance* of the estimator, and may depends on many different factors, such as the sampling distribution and the number of samples. A common technique to improve monte carlo estimators is sample from a distribution that takes into account the domain or the function profile.

Linear elliptic PDEs can be decomposed into two contribution : (i) the boundary condition, dictating how the function behave along the surface of the domain and (ii) the source term, determining on the other hand its behaviour on the inside of the volume. These two contribution can be estimated though *harmonic* functions, the *Laplace equation* and the *Poisson equation*, respectively. Harmonic functions, in other words twice continuously differentiable functions with zero value laplacian on the border of the domain, follow two principles : (i) the Kukatani's principle and (ii) the mean value property.

The *Kukatani's principle* allows us to estimate the value of the solution $u(x)$ at any point inside the surface with the value $g(y)$ at the boundary, where y is the point reached with a random walk starting in x. This concept allows us to

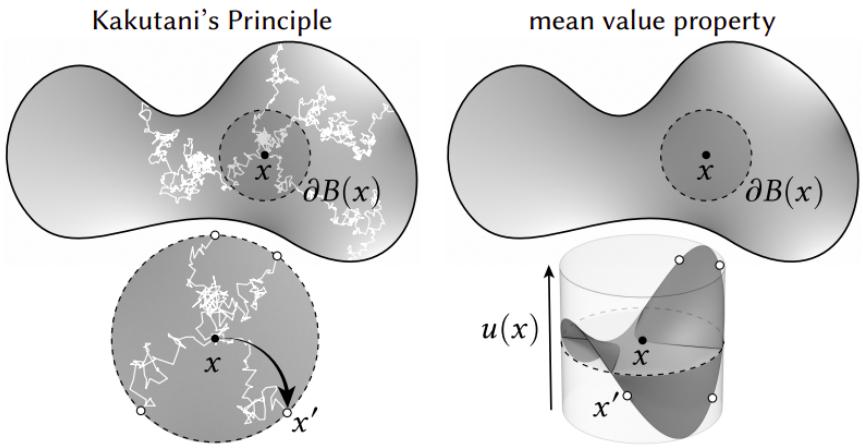

Figure 9: Visual representation of the Kukatani principle (left) and mean value property (right). They can be used as a starting point and a nice demonstration of the Walk on Sphere algorithm.

easily compute every value inside the domain and on the border by executing a random walk which, on the other hand, can be quite tricky and expensive to compute on a 3 dimensional and complex domain.

For this very reason we reach for the *Mean Value property*, stating that the value $u(x)$ is equal to the mean value of u over the boundary of any ball $B(x)$ centered in x and fully contained inside the domain Ω.

The duality of these two concept can be Incorporated in what is known as *Walk on Sphere* [7], i.e. since a random walk is equally likely to reach any points on the surface of a ball $B(x)$, we can uniformly sample a point on the surface of the biggest ball centered in x and fully contained inside the domain and recursively repeat this process until we reach the surface $\partial\Omega$ (see figure 9). When the walk is close enough to the surface, meaning the radius of the ball is smaller than a tolerance parameter ϵ, we can evaluate the boundary condition on the nearest point on the surface. After N walks, the mean value of the sum of these estimated contributes, thanks to Kukatani's and the Mean value properties, will represent an estimate of the real solution.

3.1.1 Estimators

These technique, which represents the keystone of the methodology, can be applied to every PDEs that can be represented by harmonic functions. In this papers are discussed *Laplace equations*, *Poisson equations* and *Biharmonic equations*.

Laplace Equation. The Laplace equation solves the basic system :

$$\begin{array}{cc} \Delta u = 0 & \text{on } \Omega \\ u = g & \text{on } \partial\Omega \end{array} \tag{10}$$

which can be estimated using the described form of the Walk on Spheres method :

$$\hat{u_0}(x_k) := \begin{cases} u_0(x_{k+1}) & \text{on } \Omega \\ g(x_k) & \text{on } \partial\Omega \end{cases} \tag{11}$$

After N walks the i-th walk contributes are averaged to obtain the final result $\hat{u_0} = \frac{1}{N}\sum_{i=1}^{N} u_0(x_0)$.

Poisson Equation. The poisson equations introduce a source term f on the inside of the domain (see equation 12). To account for this contribution a second term y is sampled for each ball during the walk from a distribution over the *inside* of said ball, rather than the boundary.

$$\begin{array}{cc} \Delta u = f & \text{on } \Omega \\ u = g & \text{on } \partial\Omega \end{array} \tag{12}$$

The contribute of $f(y)$ is added to the solution after been weighted by a distribution coefficient G depending on the distance between the center x_k and the sampled point y_k. This is the *Green's Function* over the ball of radius R and it's role is to concentrate the contribution of the source term around the center of the ball. Finally, both the boundary and source contribution can be approximated in the same way as the Laplace equation (see equation 13).

$$\hat{u_f}(x_k) := \begin{cases} \hat{u_f}(x_{k+1}) + |B(x_k)| f(y_k) G(x_k, y_k) & \text{on } \Omega \\ g(x_k) & \text{on } \partial\Omega \end{cases} \tag{13}$$

A common variation of the Poisson equation is the *Screened Poisson Equation* (see equation 14) , which introduces a screening term c to the equation and reduces the diffusion effect of the source inside the domain. It's computation is basically the same as the Poisson equation, with the exception that the Green's Function is substituted by the *Yukawa equation* G_c and the boundary value is weighted by a constant C depending on the screening parameter and the dimension of the domain.

$$\begin{array}{cc} \Delta u - cu = f & \text{on } \Omega \\ u = g & \text{on } \partial\Omega \end{array} \tag{14}$$

Biharmonic Equation. Biharmonic equations, used for example for shape deformation, introduce a second derivative element to the equation , that cane be trivially solved by nesting a laplace equation inside a poisson equation, performing a model semplification without lost of generality (see equation 15).

$$\begin{cases} \Delta^2 u = 0 & \text{on } \Omega \\ u = g & \text{on } \partial\Omega \\ \Delta u = h & \text{on } \partial\Omega \end{cases} \rightarrow \begin{cases} \Delta u = v & \text{on } \Omega \\ u = g & \text{on } \partial\Omega \end{cases} \text{U} \begin{cases} \Delta v = 0 & \text{on } \Omega \\ v = h & \text{on } \partial\Omega \end{cases} \tag{15}$$

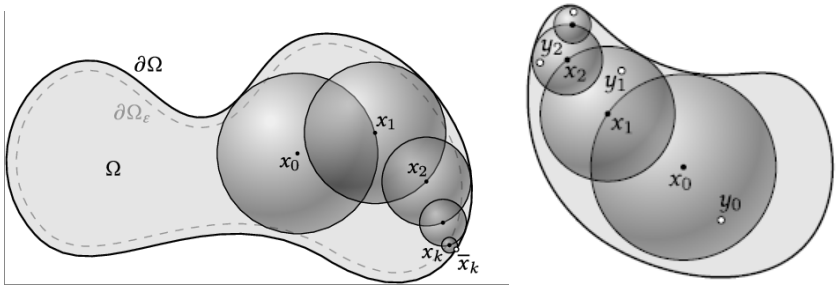

Figure 10: Left : Walk on Spheres to solve Laplace equations. Right: Walk on spheres to solve Poisson equations

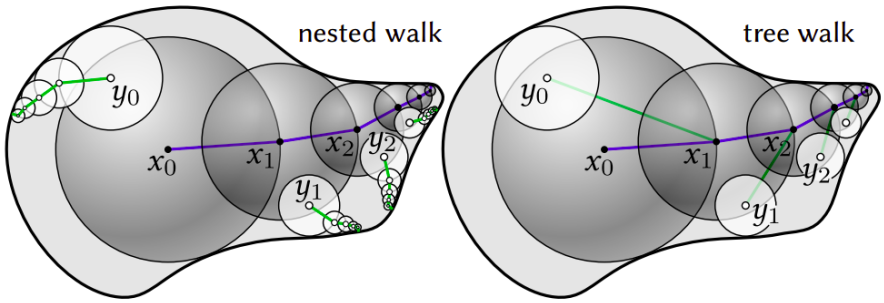

Figure 11: Comparison between biharmonic equations solve using nested equations (left) and tree walking (right).

Using this nested equation, however, is computationally intense and may be a source of variance, considering the number of random walks performed in a single run (see figure 11). A nice improvement, inspired by Bidirectional Path Tracing (BDPT), consists in connecting the source terms to the boundary estimate, in other words the term $h(y)$, previously solved by performing a random walk starting in y, is substituted by $h(x_i)$, where x_i is the next sampled point in the outer walk. By applying this trick not only the computation cost is reduced from $O(N^2S)$, where N is the number of walks and S is the number of steps in a walk, to $O(NS)$, but also allows the walks to be more strongly correlated.

It's relevant to notice that all these estimators can be equally applied to *exterior problems*, i.e. on the domain complementary to Ω, technique relevant when dealing with surface with holes, polygon soups or point clouds, in which the boundary of the domain is not uniquely defined. Following the advise of the author, one can use a *russian roulette* approach [9] to deal with walks that wonder far from the surface. This approach, used also in monte carlo path

tracing, consist in randomly avoiding computing the integral value based on a parameter q. In my implementation I've used a constant value of q for simplicity, but it's relevant to mention that a good value of q should depend on the integral value in some way, perhaps increasing while the integral decrease. The discarded samples are substituted with a constant value c and then weighted to reduce variance (see equation 16).

$$F' = \begin{cases} \frac{F - qc}{1-q} & \epsilon > q \\ c & \text{otherwise} \end{cases} \tag{16}$$

3.1.2 Derivatives

Often times the gradient of a PDE is needed, for example to compute the Helmotz decomposition of a vector field. Luckely it's possible to apply the same concept to gradient computation.

Laplace gradient. Using the mean value property we can derive the gradient of the Laplace solution u as:

$$\nabla u(x) = \frac{1}{|B(x)|} \int_{\partial B(x)} u(y)v(y)dy \tag{17}$$

Here y is a point sampled on the surface, while $v(y)$ is the normal to the ball computed in correspondence to y. This integral can be approximated using the walk on sphere paradigm as :

$$\hat{\nabla u(x0)} = \frac{n}{R} u(x_1)v(x_1) \tag{18}$$

where n is equal to the number of dimensions of the domain, R is the radius of the ball and $x1$ is a point sampled on the surface of the ball B(x0). In practice it's performed a walk on sphere starting from $x1$, and the result is weighted by the normal to the point.

Poisson and Biharmonic gradient. To apply this equation when there is a source term inside the domain we can use the same approach and just add the contribution of $f(y)$ weighted by the gradient of the green's function (yukawa's if the screening parameter c is not zero).

3.1.3 Variance Control

Apart from the Tree Walking approach previously discussed, there exists other ways to improve the model results by decreasing the variance of the estimator.

Firstly we can estimate lower order contribution from the Taylor series around the center of the ball by using running derivate estimates (see equation 19) to reduce the variance around the point x.

$$\hat{(u(x_0))} = \frac{1}{N} \sum_{i=1}^{N} (\hat{u(x_0)} - \bar{\nabla}^{i-1} u(x_0)(x_1^i - x0) \tag{19}$$

Since the gradient estimate is unbiased, it is also statistically independent from the i-th walk.

Another powerful tool to reduce the variance is to perform *importance sampling* over the model. This technique resemble closely path tracing methods for material and light source sampling. In particular, we can either change the sampling distribution of the source point y inside the ball (relevant for Poisson on Bidirectiona equations), which translate directly to material sampling, either sample the source function f to capture otherwise impossible to visualize characteristic, which highly resembles light source sampling in rendering.

Sampling Distribution. A good choice for the sampling distribution is to derive it from the Green's function G_c, so to focus the attention around the center of the ball, where said function has higher values. To sample from this distribution $p_c := G_c / \int G_c$ I've used, following the authors indications and purely for the $c = 0$ case, the following algorithm :

Algorithm 1 Sampling routing for Green's functions 3D

Sample d from a spheric uniform distribution
phi = atan2(d)
r = Use the Ulrich's polar method to get a distance $r/R \in [0, 1]$
$r = \frac{rRsin(phi)}{4\pi}$
return $y = r * d$

It uses the *Ulrich's Polar Method*[3] to sample from a simmetric beta distribution obtained by multiplying the regulating factor $r * sin(\theta)$ to the Green's function G in 3D (tabulated in the reference paper), where θ is the angle of rotation of y along the xy plane.

Source Term Sampling. Since the term f appears only in relation to y, randomly sampled, discrete source terms, e.g. points or curves, won't be sampled at all if there wasn't a specific sampling techniques capable to focus on interesting regions. A relevant example has $f := \delta_z c$ as source function, where c is a constant parameter equal to the value of f on a point $z \in \Omega$, and δ_z is a Dirac's delta centered in z. In this case we can use an importance density $p_z = \delta_z$, in other words if z is inside the ball $B(x)$, $y = z$ and $f = c$. A nice advantage w.r.t. rendering lights, the path tracing counterparts of spot source terms, is that there is no visibility term involved, i.e. z can always be reached by a random walk.

3.2 Boundary Representation

The Walk on Spheres algorithm described is entirely based on *closest point* computation, which is a simple routine that can be addressed for a plethora of models, from clean meshes, to implicit surfaces, even polygon soups and booleans.

3.2.1 Meshes

The main limit to closest point queries is time consumption, since they are linearly dependent to the number of faces. In order to fasten the code execution one can build a *Bounding Volume Hierarchy* (BVH) model over the mesh, in this way the time cost can be reduced to $O(logN)$, with N equal to the number of mesh triangles.

Another important routine is to create the Signed Distance Function (SDF) of the mesh in order to (a) check if a point is inside the boundary domain and (b) combine the mesh with other models though boolean operations. For clean meshes (without any holes, cracks or imperfections) it's possible to compute the *Generalized Winding Number* (GWN) [5] $\omega(\mathbf{p})$, i.e. a signed integer determining the property of a point $\mathbf{p}$ w.r.t. a closed Lipschitz Surface (in 3D), equal to the projection of the surface onto the point: (i) if $\omega < 0.5$ then x is outside Ω, (ii) if $\omega > 0.5$ $x \in \Omega$, (iii) else the value $\omega = 0.5$ roughly follows the surface of the mesh. In case of clean meshes it can be computed exactly in discrete form (see equation 20). Here Ω_f is the *solid angle* of the oriented triangular face f with respect to the point $\mathbf{p}$.

$$w(\mathbf{p}) = \sum_f \frac{1}{4\pi} \Omega_f(\mathbf{p}) \tag{20}$$

The problems of this representations are that (i) it's computationally inefficient, since needs to iterate over all the faces of the mesh, operation quite expensive (O(N)), (ii) does not work well with open manifolds and polygon soups.

The first problem can be easily solved by using a BVH, bringing the cost to $O(logN)$, while the second one cannot be solve directly just using GWN. That's why *Fast Winding Number* (FWN) [1] becomes handy. This routine improves the computation of the winding number while fastening it even further.

Basically it computes the number only if it's *near* enough to the surface, otherwise uses a pre-computed approximated value using the first order derivative of the Taylor series over the surface function. The "nearness" to the surface depends on a constant accuracy parameter β following the condition :

$$w_n(\mathbf{p}) = \begin{cases} \hat{w_n}(\mathbf{p}) & \text{if if } ||p - c|| < \beta r \\ \sum_{child} w_{child}(\mathbf{p}) & \text{otherwise} \end{cases} \tag{21}$$

In other words, for every node of the BVH, if the point $\mathbf{p}$ is far from the center of the node c (distance depending on β and the radius of the node r), then its value is substituted by an approximation $\hat{w_n}(\mathbf{p})$, otherwise corresponds to the sum of the values computed on every of its children.

While the leaves values are computed either as solid angles (with the same computation used for GWN) or as the winding number over the *area weighted dipole* of the triangle, the estimated values depends on the areas of the triangles, their normals and progressive gradient order of the Green's function G (see equation **??**). The levels of approximation can vary from 1 to 3, with increasing

accuracy, but should be noticed that even using just the first component of Taylor expansion the result are satisfying (see chapter "Results").

$$
\begin{aligned}
w(\mathbf{q}) \approx & \left(\sum_{t \in \text{triangles}} \int_t \hat{\mathbf{n}}_t dA \right) \nabla G(\mathbf{q}, \tilde{\mathbf{p}}) \\
& + \left(\sum_{t \in \text{triangles}} \int_t (\mathbf{x} - \tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}_t dA \right) \nabla^2 G(\mathbf{q}, \tilde{\mathbf{p}}) \\
& + \left(\sum_{t \in \text{triangles}} \int_t (\mathbf{x} - \tilde{\mathbf{p}}) \otimes ((\mathbf{x} - \tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}_t) dA \right) \nabla^3 G(\mathbf{q}, \tilde{\mathbf{p}})
\end{aligned} \tag{22}
$$

where :

$$\int_t \hat{\mathbf{n}}_t dA = a_t \hat{\mathbf{n}}_t \tag{23}$$

$$\int_t (\mathbf{x} - \tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}_t dA = a_t \left(\frac{1}{3} (\mathbf{x_i} + \mathbf{x_j} + \mathbf{x_k}) - \tilde{\mathbf{p}} \right) \otimes \hat{\mathbf{n}}_t \tag{24}$$

$$\int_t (\mathbf{x} - \tilde{\mathbf{p}}) \otimes ((\mathbf{x} - \tilde{\mathbf{p}}) \otimes \hat{\mathbf{n}}_t) dA = \frac{1}{3} a_t C_t \otimes \hat{\mathbf{n}}_t \tag{25}$$

Here a_t is the area of the triangle, $\tilde{\mathbf{p}}$ is the centroid of the triangle, corresponding to the center of its bounding box when working with AABB, and C_t is :

$$
\begin{aligned}
C_t = & \left(\frac{1}{2} (\mathbf{x_i} + \mathbf{x_j}) - \tilde{\mathbf{p}} \right) \otimes \left(\frac{1}{2} (\mathbf{x_i} + \mathbf{x_j}) - \tilde{\mathbf{p}} \right) \\
& + \left(\frac{1}{2} (\mathbf{x_j} + \mathbf{x_k}) - \tilde{\mathbf{p}} \right) \otimes \left(\frac{1}{2} (\mathbf{x_j} + \mathbf{x_k}) - \tilde{\mathbf{p}} \right) \\
& + \left(\frac{1}{2} (\mathbf{x_i} + \mathbf{x_k}) - \tilde{\mathbf{p}} \right) \otimes \left(\frac{1}{2} (\mathbf{x_j} + \mathbf{x_k}) - \tilde{\mathbf{p}} \right)
\end{aligned} \tag{26}
$$

Since in my project I've been using only the implementation for triangular faces, the dipole computation for point clouds is not present here.

3.2.2 Implicit Surfaces

The proposed algorithm works on implicit surfaces as well as on meshes, since it does not need to perform mesh generation to perform over the model. Closest point queries can be solve easily and efficiently using the SDF of the surface directly (when computing the largest ball around a point contained inside the surface) or through *Sphere Tracing* algorithm [4] that allows us to (i) find the closest point on the surface (ii) find the intersection point with a ray during rendering routines. Furthermore, having the SDF of the surface is quite trivial to compute booleans over models, such as intersections, unions, subtractions, but also smooth booleans.

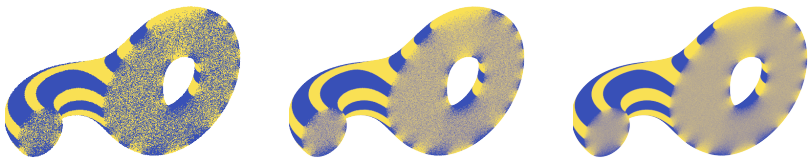

Figure 12: Laplace equation solved with three different number of walks: 1 (left), 10 (center) and 100 (right).

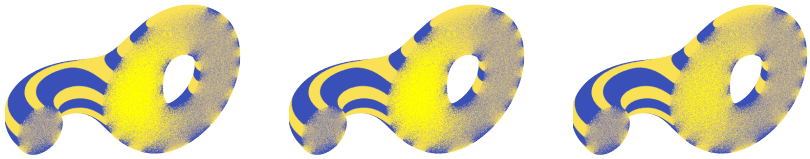

Figure 13: Poisson equation solved with three different values of screening: c = 0 (left), c = 100 (center) and c = 1000 (right).

3.3 Results

In this section are collected the most relevant results obtained with the various techniques defined previously. If not otherwise stated, the boundary function $g(x)$ defines horizontal stripes on the boundary surface (see equation **??**), the tollerance ϵ is set to 10^{-5}, the max number of steps is 128 and the number of walks is 10. The models used are (i) an implicit surface created through the smooth intersection of two equal and perpendicular to each other tori and (ii) a complex triangular mesh of a bunny, to which a surface it's been subtracted. Both models are cut by a plane to visualize the behaviour of the function inside the domain.

In figure 12 can be observed how the number of walks affect the results.

In figure 13 it's been solve a poisson equation with a spheric source term, with various values of screening.

In figure 14 to the source term it's added a point source with various inten-

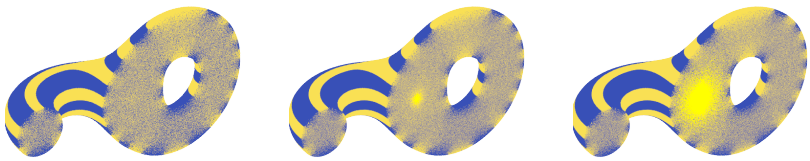

Figure 14: Source sampling with a point source inside the domain. The source values are 10 (left), 100 (center) and 1000 (right).

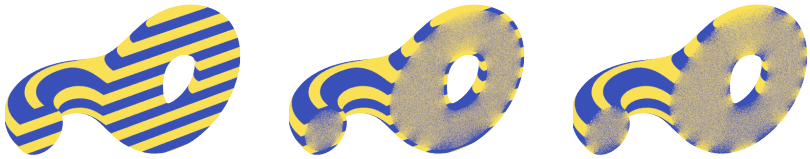

Figure 15: Laplace equation solved with three different values of tolerance: $\epsilon = 0.1$ (left), $\epsilon = 0.01$ (center) and $\epsilon = 0.0001$ (right).

sities.

In figure 15 are compared different values of tollerance.

figure 16 is solved the laplace equation over a complex mesh, to which a sphere has been subtracted. It's visibly noisy since the number of walk used was 1, due to time consumption.

References

[1] Gavin Barill et al. "Fast Winding Numbers for Soups and Clouds". In: *ACM Transactions on Graphics* (2018).

[2] Eugene d'Eon et al. "An Energy-Conserving Hair Reflectance Model". In: *Comput. Graph. Forum* 30 (June 2011), pp. 1181–1187. DOI: 10.1111/j.1467-8659.2011.01976.x.

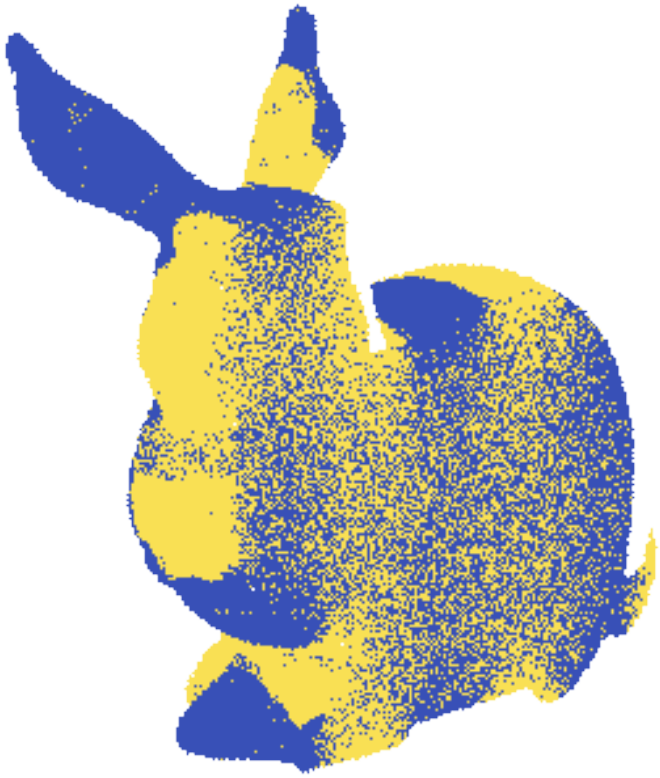

Figure 16: Laplace equation solved over a mesh boolean.

[3] Luc Devroye. “Sample-Based Non-Uniform Random Variate Generation”. In: *Proceedings of the 18th Conference on Winter Simulation*. WSC ’86. Washington, D.C., USA: Association for Computing Machinery, 1986, pp. 260–265. ISBN: 0911801111. DOI: 10.1145/318242.318443. URL: https://doi.org/10.1145/318242.318443.

[4] John Hart. “Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces”. In: *The Visual Computer* 12 (June 1995). DOI: 10.1007/s003710050084.

[5] Alec Jacobson, Ladislav Kavan, and Olga Sorkine. “Robust Inside-Outside Segmentation using Generalized Winding Numbers”. In: *ACM Trans. Graph.* 32.4 (2013).

[6] B. G. Korenev. “Bessel Functions and Their Applications”. In: 2002.

[7] Mervin E. Muller. “Some Continuous Monte Carlo Methods for the Dirichlet Problem”. In: *The Annals of Mathematical Statistics* 27.3 (1956), pp. 569–589. DOI: 10.1214/aoms/1177728169. URL: https://doi.org/10.1214/aoms/1177728169.

[8] Matt Pharr. “The Implementation of a Hair Scattering Model”. In: *Physically Based Renderings*. Ed. by Pharr Matt, Jakob Wenzel, and Humphreys Greg. 2016.

[9] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)* 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Oct. 2016. Chap. 13. ISBN: 9780128006450.

[10] Rohan Sawhney and Keenan Crane. “Monte Carlo Geometry Processing: A Grid-Free Approach to PDE-Based Methods on Volumetric Domains”. In: *ACM Trans. Graph.* 39.4 (2020).