General Rendering of Grooved Surfaces

Carles Bosch, Xavier Pueyo, Stéphane Mérillou[1], and Djamchid Ghazanfarpour[1]

Research Report IIiA06-10-RR

Institut d'Informàtica i Aplicacions
Universitat de Girona

[1]Laboratoire MSI, Université de Limoges, France

Abstract

The majority of objects in the real world exhibit different kinds of surface detail, and grooves is one of the most common. Although a great variety of models have been proposed to consider this type of details, such as anisotropic BRDFs, scratch models, or macro-geometric techniques like bump mapping and displacement mapping, they have many limitations on the type of groove that can be simulated, especially on the distributions and sizes of grooves. In addition, masking, shadowing, or inter-reflections are often neglected. In this paper, we present a general method for rendering grooves that is both fast and accurate, without any limitation on their distribution, size, or geometry. It uses a unique representation for the grooves that requires little memory storage and, due to an appropriate antialiasing method, allows correct smooth transitions among different scales. Our method also includes masking, shadowing, and specular inter-reflections/transmissions inside and between the grooves, and special cases are proposed for intersections between grooves and for groove ends as well. Implemented under a software renderer based on ray tracing, different results are shown demonstrating benefits and possible applications. Some of the surface features that could be simulated with our new method are: scratches, fractures, tiles or bricks, ridges, among others.

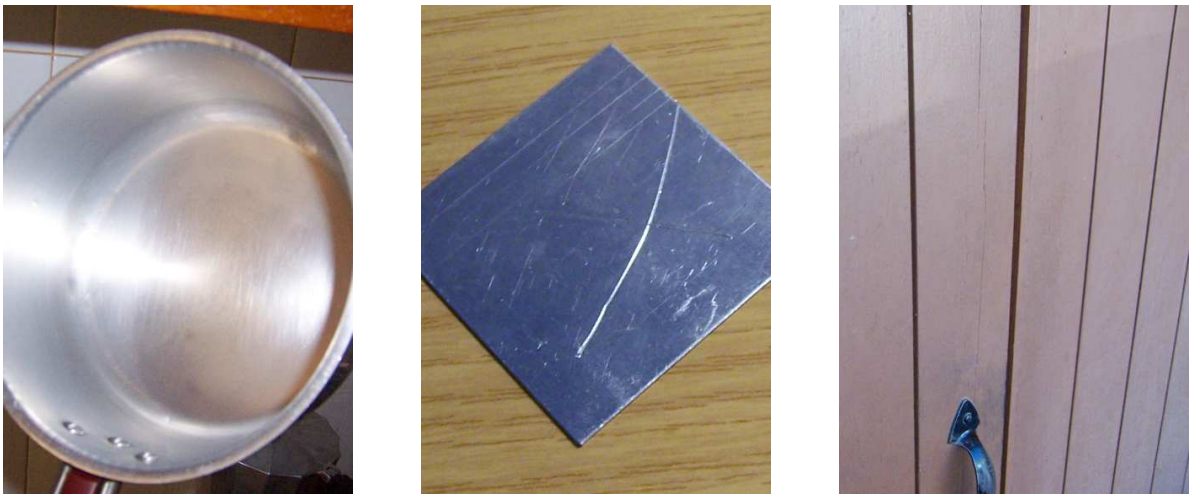


Figure 1: Pictures of real surfaces exhibiting a different type of groove. From left to right: a polished pan with lots of micro-grooves, a scratched surface with isolated scratches, and a door with big grooves between the wooden planks.

1 Introduction

Grooves are one of the most common details of real world objects. They are present in lots of human manufactured objects, such as engraved objects, polished metals, or even on tiled surfaces. Grooves are often called sweep features [CGF04], because they can be characterized by a cross-section swept along a path over the surface, and are also considered as surface features, since they appear in the surfaces of objects. In Computer Graphics a great variety of models have been proposed to simulate them on synthetic images. These can be classified into three main categories, depending on the type or size of the groove that is simulated (see Figure 1): anisotropic models, scratch models, and macro-geometric models.

First models simulate very small grooves distributed along the overall surface, which are not individually perceptible but provide an homogeneous anisotropic aspect to the surface, due to their preferred orientation. Typical examples can be found on brushed or polished surfaces. Due to its microscopic nature, this kind of grooves is usually modeled by the local reflection model or BRDF, where general anisotropic models are used [Kaj85, PF90, War92].

Scratch models, on the other hand, simulate small isolated grooves that are individually visible. Although the reflection of scratches is individually perceptible, their geometry still remains invisible, i.e. their width or cross-section is smaller than half the pixel size, thus they are simulated using a combination of texture and BRDF models [BB90, MDG01, BPMG04].

Finally, macro-geometric models are general models that allow the simulation of different kinds of surface details. Most common techniques are bump mapping [Bli78] or displacement mapping [Coo84]. They are especially useful to simulate bigger surface features, such as grooves found on engraved wood or on tiled walls. Naturally, grooves could also be directly included into the geometry model of the objects using none of the aforementioned techniques. This is only suitable for big details too, and is commonly done, for example, in interactive applications such as the editing of surfaces [CGF04, BIT04].

Each of the previous groups is intended to simulate a different type of groove, with limitations on the distribution or size of the grooves. Concerning the distribution of grooves over the surface, current anisotropic BRDF models only treat parallel and identical micro-grooves or specific statistical distributions, while scratch models do not allow more than a groove per pixel. In addition, special cases such as intersections between grooves or groove ends are considered by none of them. Concerning the size of grooves, anisotropic BRDFs are only valid for very small micro-grooves, assuming the pixel contains many of them. Scratch models allow greater grooves, but with their size limited to half the pixel size. Instead, macro-geometric techniques are more suitable for grooves bigger than the pixel size. For smaller grooves these require good antialiasing or filtering methods, which can be very time consuming. This is especially true for highly detailed surfaces or for algorithms based on over-sampling the scene, like ray tracing rendering techniques.

Note that the size of grooves, however, is relative to the pixel size. This means that it does not depend only on the size of the groove, but also on its distance to the camera, the angle of vision, or the camera resolution. Macro-grooves can become micro-grooves, or the opposite, if any of these parameters is sufficiently changed. Thus, some general multi-

resolution methods have attempted to combine BRDF with bump mapping or displacement mapping, trying to use the best model depending on some of those parameters [Fou92, BM93]. Nevertheless, smooth transitions among the different representations are difficult to carry out, and some geometric effects are neglected, such as masking, shadowing, or inter-reflection, because most of the aforementioned models do not consider them.

In this paper, we propose a general method to render grooves of any geometry, size or distribution. It takes into account the particular properties of grooves to render them both accurately and quickly. Furthermore, it uses a unique and continuous representation of the grooves that requires little memory storage too. This representation, together with the antialiasing methods proposed here, results in a resolution-independent technique that allows correct smooth transitions of grooves among different scales, distances, or other factors. Our method also takes into account occlusion effects such as masking and shadowing, as well as specular inter-reflections/transmissions inside or between the grooves. Finally, specific solutions are also proposed for intersections between grooves and for groove ends. Since our method does not modify the geometry model of objects, it can be easily applied to any kind of object geometry in order to simulate, not only grooves, but also other similar surface features: scratches, fractures, wrinkles, bricks, ridges, and many others. These kinds of surface details are present in a lot of real world scenes, thus a great set of applications can be imagined for our method.

2 Previous work

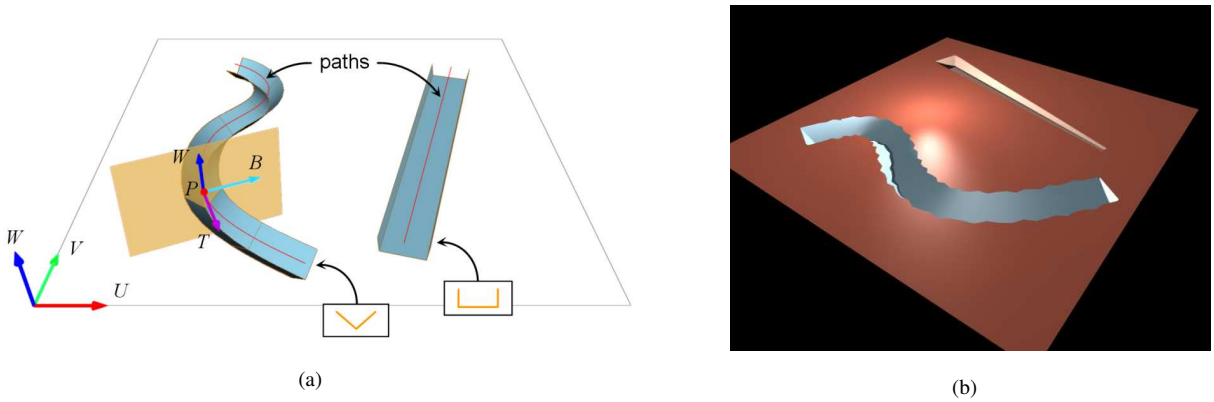
2.1 Anisotropic BRDFs

Many anisotropic reflection models have been proposed over the past years to model surfaces containing micro-grooves. Most of them are empirical models that are chosen when the micro-geometry is not known [War92, Ban94, AS00]. Physically-based models are also available, but only for certain types of micro-geometry, like parallel cylinders [PF90] or random Gaussian surfaces [Sta99]. For general types of geometry, brute force methods precompute the reflection for a subset of directions and store it in tables [Kaj85] or approximate it by spherical harmonics [CMS87, WAT92], but these are most suitable for periodic patterns. Ashikhmin et al. [APS00] also propose a method to generate reflection models from arbitrary normal distributions using a simple occlusion term, and they give an example for ideal V-shaped grooves. The method that we introduce does not have any restrictions neither on the distribution of the grooves nor on their geometry, and occlusion effects are accurately computed.

2.2 Scratches

For the simulation of isolated scratches, the existing methods combine a 2D texture with an anisotropic BRDF model. The texture specifies the position and path of each scratch over the surface, and is represented as a 2D image with the scratches painted on it [BB90, BL99, MDG01] or as a collection of curves defined in texture space [BPMG04]. The BRDF then models the specific light reflection on each scratch point. To compute this reflection, Becket and Badler [BB90] simply apply a random intensity to each scratch, while Buchanan and Lalonde [BL99] use a Phong-like model that adds to the surface base reflection the maximum highlight due to the scratches at the current position. Other authors have proposed a physically-based approach that takes into account a specific cross-sections for each scratch, and where the BRDF is computed according to the cross-section at the current position [MDG01, BPMG04]. However, these models only consider one scratch at a time: very close or intersecting scratches are not considered, as well as scratch ends. Furthermore, scratches are limited to the pixel size.

Together with general anisotropic BRDFs, many authors have also proposed to store the frame or tangent of the surface onto a texture map [Kaj85, PF90, Sta99, KS00]. This allows the variation of the anisotropic effect along the surface, but could also be used to simulate isolated scratches if tangents are mapped only at scratched points. Kautz and Seidel [KS00] proposes a similar approach for shift-variant BRDFs, storing anisotropic parameters at these points and isotropic ones at the rest of points. Scratch models usually map a scratch index instead, which is later used to determine the parameters of scratches, such as the cross-section and the reflection properties [MDG01]. In our case, we model the grooves using the continuous representation of [BPMG04] instead of a texture map, because it is more suitable for a resolution independent approach.



(a) (b)

Figure 2: (a) Grooves are represented in texture space by means of paths (in red) and cross-sections (in orange). (b) Rendering from a different point of view after applying the grooves onto a plane. This plane is the only geometric primitive. Different BRDFs were used for each groove, as well as different perturbation functions: a sinusoidal function for the curved groove and a shrinking function for the straight one.

2.3 Macro-geometry methods

With regard to big surface features, these are commonly simulated using general techniques such as bump mapping [Bli78], displacement mapping [Coo84, LP94, PH96, SSS00, WWT+03], or relief mapping [OBM00, POC05], among others. Another possibility is to modify the geometric model of the object to include these details, which is usually done for interactive sculpting or editing of surfaces. Many recent works deal with subdivision surfaces [BMZB02, CGF04], CSG models [MOiT98], or volumetric models [BIT04]. However, these macro-geometry methods are rarely suitable for small or distant details.

2.4 Multi-resolution models

To account for details of all sizes, a common solution is to perform smooth transitions between different representations of the surface details, one for each scale or distance. Becker and Max [BM93] address the transition among displacement mapping, bump mapping, and BRDF. Due to the inconsistencies of the different representations these transition needs to be approximated, however, and shadowing or inter-reflections are not considered. Other authors have suggested the use of a single representation based on normal distribution maps [Fou92, Sch97, KHS01], but only the normals are used in these techniques, not the geometry. Thus, geometric effects like shadowing or masking between the different surface details are not directly taken into account unless precomputing them [Max88, HDKS00]. Our method uses a single representation too, but shadowing, masking, and inter-reflections effects can be handled due to the availability of the geometries of the grooves.

3 Representation overview

The objective of this paper is to deal with the simulation of grooves and similar surface features. This kind of features, sometimes referred as sweep features [CGF04] or curve-like features [KS99], are usually characterized by means of a cross-section swept along a path over the surface. Both the cross-section and the path are modeled by a curve or by a piecewise line, and the cross-section is often allowed to change or be perturbed along the path [CGF04, BPMG04], to account for non-uniform features.

As depicted in Figure 2(a), here grooves are modeled using a similar representation. Since our method works in texture space, this modeling is done in the space characterized by vectors U, V, and W, where UV is the texture plane. Each groove is first described by a unique path and cross-section: the path is specified as lying on the UV texture plane and the cross-section as lying on the BW plane of the path's local frame. For each point P on the path, such a frame is defined by the tangent T at P, the binormal B perpendicular to T, and the texture space vector W (see Figure 2(a)).

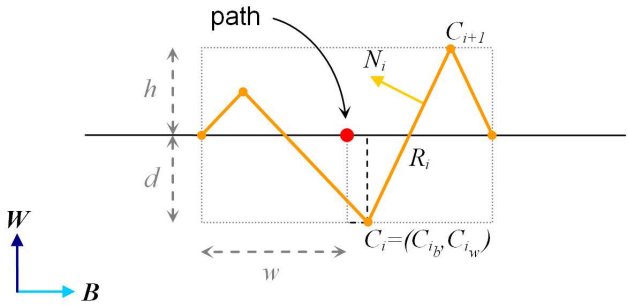


Figure 3: Cross-section modeled using a piecewise line, composed of many points C_i and facets R_i lying on the BW plane. A common bounding box is computed for all the cross-sections from their maximum width w, height h, and depth d.

Paths are modeled by means of curves or line segments. These are directly specified in texture space or in 3D world space, by defining the paths onto the object surface and then projecting these onto texture space. Cross-sections are modeled using a piecewise line (polyline). This allows easy and quick computations of reflections and occlusions of grooves. Thus, curved cross-sections are approximated by polylines. Figure 3 shows an example of such a piecewise cross-section. It is composed of many points $C_i = (C_{i_b}, C_{i_w})$ that lie on the BW cross-section plane and that are defined with respect to the path, with C_{i_b} being the distance from this point to the corresponding path in the UV texture plane, and C_{i_w} its height from the base surface. Depending on the sign of C_{i_b}, the point lies on one side or the other of the path, and depending on the sign of C_{i_w}, the point is penetrating into, or protruding from the surface. Grooves usually penetrate the surface, thus C_{i_w} is usually negative, but they can also have peaks, for which C_{i_w} will be positive. For first and last cross-section points, $C_{i_w} = 0$, as they rely on the UV plane. Finally, every pair of two consecutive points define a facet $R_i = [C_i, C_{i+1}]$, whose normal is $N_i = (N_{i_b}, N_{i_w})$.

The cross-section associated to a groove can also change along the groove path if desired. These changes are done according to a function $f(t)$ that can be independently specified for each groove. For a given point on the path and its parametric position $t \in [0..1]$, it returns a value that alters the cross-section points. Here, we have considered that this value affects the overall scale of the cross-section, multiplying every cross-section point by the scale factor, but other types of perturbations could be used if preferred. Figure 2(b) shows a plane with two grooves rendered using our method, each one using a different perturbation function: a sinusoidal for the curved groove and a shrinking function for the straight one.

For each groove, apart from specifying its path, cross-section, and optional perturbation, we can also choose specific reflection and transmission properties. This is necessary only when these properties are different from the ones of the base surface. In that case, they can be specified for the entire cross-section or for each facet if desired. The latter can be useful, for example, to simulate a bricked wall by means of grooves, where some of the facets will have the properties of the bricks and others the properties of mortar. Different models of reflection (BRDF) or transmission (BTDF) can also be used for each one. In the example of Figure 2 each groove uses a different BRDF.

Once the grooves have been modeled, we must store some information about the grooves for later use in the rendering stage. This information mainly consists in a sort of cross-section bounding box and in occlusion information. For the bounding box, three common parameters are stored: maximum width w, height h, and depth d (see Figure 3). These three parameters are obtained from the dimensions of every cross-section, taking into account the maximum perturbation applied to it, if any. Concerning the occlusion information, this consists in determining which facets can be occluded due to other facets from the same cross-section, and in storing a list of the possible blocking facets for each one. Mainly, blocking facets will be those lying at the same height or higher than the current facet, i.e. with one of their points higher than the lowest point of the current facet, and this will be used during the rendering of intersections between grooves and groove ends (see Section 5.4). Since here perturbation only affects the cross-section scale, this occlusion information will be maintained along the paths.

After modeling the grooves and storing this information, the next phase will consist in their rendering using our approach. This is detailed in the following section.

4 Rendering grooves

This section describes the main contribution of the paper, which relates to the rendering of the grooves using a general and resolution independent approach. Here, we only treat the simple case of isolated or parallel grooves, without accounting for intersections nor ends of grooves. These two special cases will be treated later in Section 5.

When a scene being rendered contains grooved surfaces, modeled using the previous representation, our method will be executed each time a pixel or ray intersects them. In texture space, the intersection of a pixel or ray with a surface gives a footprint as a result, often called the filter shape, which determines the texture area to be sampled for the current pixel or ray [Hec86]. Our rendering procedure is called after the computation of this footprint.

The main idea of our analytical approach is to treat all the grooves contained in the current footprint, analyzing each of their facets individually to compute each groove contribution to the final rendering pass. All of this is done in texture space, thus we suppose that the view E and light L directions, also referred to as reflecting and incoming directions, have been transformed into this space, as well as normalized. In most applications, the footprint shape is usually considered to be a quadrilateral or an oriented ellipse for high quality results. Here, both shapes can be used.

Given a texture footprint, the reflection coming from a grooved surface is computed using the following algorithm:

1. Find the grooves affecting the current footprint (Section 4.1).
2. If no groove is found $\Rightarrow$ Compute the reflection using the base surface BRDF.
   Else $\Rightarrow$

   For each groove:

   For each cross-section facet:
   (a) Clip the footprint to the facet (Section 4.2).
   (b) Subtract masking and shadowing effects due to neighboring facets (Section 4.3).
   (c) Compute facet reflection and add it to the accumulated reflection (Section 4.4).

This algorithm only accounts for direct illumination; its extension in order to include inter-reflections and transmissions will be explained in Section 6. In the following sections, we describe the details of the previous algorithm and its different steps.

4.1 Finding grooves

In the first step of the algorithm, the objective is to quickly determine which grooves affect the current footprint. This includes grooves contained by the footprint as well as grooves casting shadows inside the footprint. In ray tracing, a common practice to accelerate ray-object intersection tests is to perform some sort of space subdivision. Here, we use the same principle but only for the paths of grooves, not for their entire geometry, because it requires less memory and precomputation time. Furthermore, the spatial subdivision does not have to be recomputed when cross-sections or their perturbation functions are modified.

As done by Bosch et al. [BPMG04] for scratches, the type of subdivision we use for paths is a uniform space subdivision, which is simple and gives a good performance in our case. To obtain a uniform subdivision, the UV plane is subdivided into a grid of equally sized cells, saving in each cell a list of its crossing paths (see Figure 4). This is precomputed just after the edition of the paths. Then, during the rendering stage, in order to find the paths inside the footprint, we determine the dimensions of the footprint onto this grid and take the cells inside. These dimensions are obtained by computing the bounding box of the footprint onto the UV plane.

This bounding box could be enough for very narrow or distant grooves, because the footprint will usually contain the paths of such grooves. However, for wider or close grooves, the footprint can contain only a portion of a groove without containing the corresponding path (see Figure 4). If the path is missed then the groove will be also missed for its further processing. This is due to the fact that only the linear paths have been stored inside the grid, without taking into account the dimensions of each groove. Nevertheless, these dimensions can be easily considered at this stage too, by accordingly enlarging the footprint's bounding box before taking the grid cells. Remember that in Section 3 we stored a sort of maximum bounding box for all the cross-sections, represented by a width w, height h, and depth d. These dimensions are used to enlarge the footprint's bounding box.

In Figure 4, for example, the footprint is affected by two grooves whose paths lie outside its bounding box. When the footprint or part of the footprint lies inside the bounds of a groove (left groove of Figure 4), its path can be found

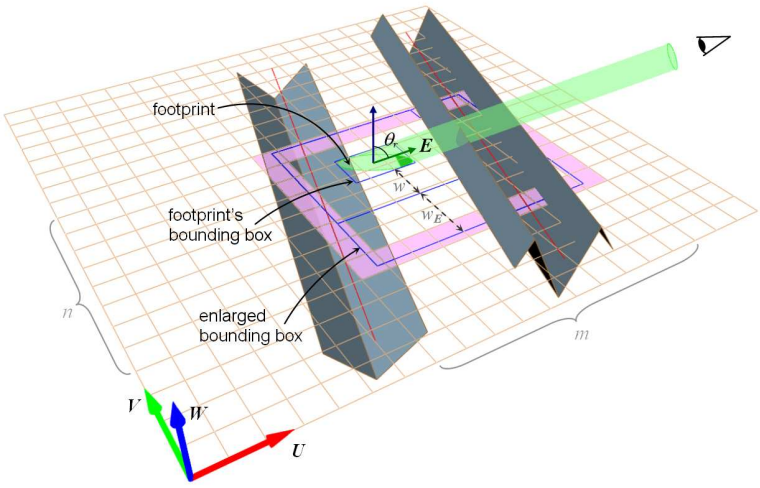


Figure 4: The UV texture plane is uniformly subdivided in order to quickly find the grooves affecting a footprint. The current footprint contains two grooves, one of them being viewed far from its bounds due to a peak. To find their paths into the grid, the footprint bounding box is computed and then enlarged according to the width w and projected height w_E of all the grooves.

by simply enlarging the footprint's bounding box according to the maximum width w. This would be sufficient if no groove protrudes from the surface, but this is not the case for the right groove. That groove has two peaks that can be viewed or shadow far away from the bounds (width) of the groove. In such cases, we must consider the height value h too, specifically its projection according to the view/light angle, and the bounding box must be enlarged in these directions. For the view direction $E = (E_u, E_v, E_w)$, which is the case shown in the figure, the box is enlarged according to

$$w_E = h \tan\theta_r \,,$$

where θ_r is the view angle, and $\tan\theta_r$ is obtained by

$$\tan\theta_r = \frac{\sqrt{1-E_w^2}}{E_w} \,. \tag{1}$$

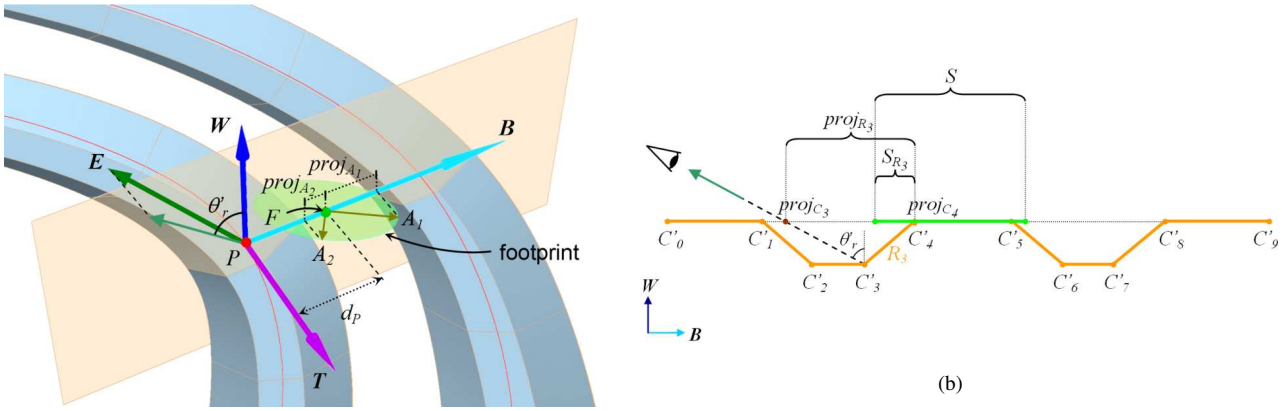
A similar value is computed for each light source direction L too. The previous equation, however, must be computed only for vectors above the surface, i.e. $E_w > 0$ and $L_w > 0$.

Due to the use of the global values w and h, the footprint's enlarged bounding box may contain paths of grooves not really affecting the reflection of the footprint. Those grooves will be discarded during the clipping step (see Section 4.2), but usually, grooves tend to have a similar cross-section size and are not highly perturbed, thus these values rarely differ too much from the real bounds of the grooves.

Once the final bounding box of the footprint is obtained, we get the grid cells inside it and a list of grooves for each cell. Their union finally gives the grooves affecting the current footprint. If the bounding box exceeds the limits of the grid, we also consider the cells from the other sides of the grid, which is suitable for texture repeats, for example. Since grooves are rarely shorter than a footprint, instead of taking all the cells within the bounding box, we only consider the cells lying on the boundary of the box (see pink cells of Figure 4). This reduces the number of cells to examine from $m \times n$ to $2m + 2(n-2)$ without loosing any groove, where m and n are the dimensions of the box onto the grid. The number of cells to consider could be further reduced by only using the cells lying on the two central axes of the bounding box. Nevertheless, we have found that some groove ends can be missed if the corresponding paths do not reach the footprint axes, so we do not use this last option.

4.2 Footprint clipping

For every groove found in the first step, the following pass is to determine, for each of its facets, the portion contained in the footprint. For grooves following straight paths, this requires a projection and clipping operation with a flat facet.



(a) (b)

Figure 5: (a) Footprint containing two parallel curved grooves in texture space. Its two axes are projected onto the common cross-section plane defined by P, B, and W, to find the best segment representing it. (b) The different cross-sections are merged into a unique cross-section in the cross-section plane, and then projected onto the base line according to the viewing angle. Here, the footprint segment S is clipped to a projected facet $proj_{R_3}$, resulting in S_{R_3}.

For curved paths, instead, these operations must be performed with curved facets, which is more expensive. This can be further complicated if the cross-section is perturbed along the portion covered by the footprint. To simplify it, we approximate curved paths as straight lines and assume a constant perturbed profile on each footprint, thus the operations can be easily done with flat facets too. If grooves have high curvatures or high frequency changes along the path, the footprint can be oversampled if necessary, by subdividing it into several small footprints.

To locally approximate a curved path with a straight one we must determine its tangent T at the current point (see Figure 5(a)). This point P will be the point on the curve nearest to the footprint center F. The straight path will then be specified by the line passing through P in the direction of T. If the path is already defined as a line segment, any of its end points can be used for P, and its direction vector for T. As stated before, the frame of a groove is then specified by the following normalized vectors: the tangent T, the binormal B perpendicular to it, and the W texture vector. Provided that the grooves inside the footprint are parallel, the same frame is shared by all of them. The only difference between their paths is their signed distance d_P to the current footprint, which is computed from $F = (F_u, F_v)$, $B = (B_u, B_v)$, and P using the implicit line equation as:

$$d_P = B_u F_u + B_v F_v - B \cdot P = B \cdot (F - P) \,. \tag{2}$$

Before the clipping operation, the projection of the footprint onto the different facets must be determined. Since we have considered that the shape of a groove does not change inside the footprint and is represented by its current 2D cross-section, it can be easily performed by projecting the cross-sections onto the UV texture plane, where the footprint lies. First, the different cross-sections are translated according to their distance d_P to the footprint, $C'_i = C_i + d_P$. Then, they are arranged and merged into a unique cross-section, where the surface around the different grooves, if any, is included by adding ground facets between grooves (see Figure 5(b)). Parallel grooves usually share the direction T, but if any has an opposite T, then d_P and its cross-section should be inverted before the arrangement. The projection of this unified cross-section will then be performed according to the viewing angle θ'_r, which is the angle due to the projection of the viewer onto the cross-section plane, as shown in Figure 5. For each point of the unified cross-section $C'_i = (C'_{i_b}, C'_{i_w})$, the following expression is used:

$$proj_{C_i} = C'_{i_b} + C'_{i_w} \tan\theta'_r \,,$$

where

$$\tan\theta'_r = \frac{-B \cdot (E_u, E_v)}{E_w} \,.$$

Note that the angle θ'_r is signed here. If the viewer and B directions remain on the same side then the angle will be negative, and positive otherwise.

The obtained one-dimensional points $proj_{C_i}$ lie on the line defined by the binormal vector B. The footprint must now be clipped to the facets of this profile. The usual way would be to intersect the footprint with each of these parallel facets in the 2D UV texture plane, but it could be easily performed in 1D if the footprint is also projected onto B as a line segment, that is, approximating the original shape of the footprint with a single line segment representing the overall footprint. We have performed many tests and the loss of accuracy due to this approximation is imperceptible, thus it is the approach used here.

In order to determine the projection of the footprint onto the binormal vector, we project its two main axes A_1 and A_2 (see Figure 5(a)), which is faster than projecting its entire shape. This is done with the following expressions:

$$proj_{A_1} = A_1 \cdot B,$$
$$proj_{A_2} = A_2 \cdot B.$$

Then, the largest of these projections will correspond to the axis that better represents the footprint on the cross-section plane,

$$proj_{max} = \max(|proj_{A_1}|, |proj_{A_2}|) ,$$

and the resulting footprint segment S is

$$S = [-proj_{max}, proj_{max}] ,$$

defined with respect to the footprint center, as with the projected cross-section. In the example of Figure 5(a) the largest projection is $proj_{A_1}$, thus $S = [-proj_{A_1}, proj_{A_1}]$.

The segment $S = [S_0, S_1]$ must be finally clipped with each projected facet $proj_{R_i} = [proj_{C_i}, proj_{C_{i+1}}]$, using the following expression:

$$S_{R_i} = \begin{cases} null & \text{if } S_0 > proj_{C_{i+1}} \text{ or } S_1 < proj_{C_i} \\ [\max(S_0, proj_{C_i}), \min(S_1, proj_{C_{i+1}})] & \text{otherwise} \end{cases} \quad (3)$$

The first case occurs when S is completely outside of the current facet $proj_{R_i}$, and the second case, when S is partially or totally inside the facet. In Figure 5(b), for example, the current footprint segment S is clipped with the facet $proj_{R_3}$ resulting in a partially inside segment S_{R_3}.

4.3 Occlusion

Occlusion effects such as masking or shadowing appear when the way from the viewer or the light source to the current facet is blocked by another facet, so that a portion of the facet is not visible, not illuminated, or both. In order to determine the actual contribution of each facet to the final reflection off the footprint, only the visible and illuminated portions should be considered. Here we show how to compute these portions.

Masking occlusion can easily be taken into account during the projection of the unified cross-section in the previous step. This is done by treating the cross-section points C'_i in the same order that are viewed from the observer, and then looking their final order after the projection. The order in which these points are treated will be given by the sign of the view angle θ'_r. For example, if the viewer has a positive θ'_r angle, like in Figure 6, the cross-section points are treated in the normal order, from left to right. After the cross-section has been projected, the occlusion of each facet $proj_{R_i}$ is determined using the following expression:

$$proj_{R'_i} = \begin{cases} null & \text{if } proj_{C_{i+1}} \leq proj_{C_j}, \text{ with } j \leq i \\ [proj_{C_j}, proj_{C_i+1}] & \text{if } proj_{C_i} < proj_{C_j} < proj_{C_{i+1}}, \text{ with } j < i \\ proj_{R_i} & \text{otherwise} \end{cases} \quad (4)$$

By comparing the order in which the different projected points lie on the base line, we can quickly determine which facets are completely masked (first case), partially masked (second case), or not masked (third case). A facet is completely masked if its last projected point $proj_{C_{i+1}}$ lies before its first point $proj_{C_i}$, producing self-masking, or before a previously projected point $proj_{C_j}$ with $j < i$. In Figure 6 for instance, facets R_2 and R_5 are self-masked, and R_3 is completely masked by R_1. A facet is partially masked if the previous situation happens for the first point of the facet but not for its last one. In such a case, the visible portion must be determined. In Figure 6, facet R_4 is partially masked by R_1, where its projected visible portion is $[proj_{C_2}, proj_{C_5}]$, and R_6 by R_4. Finally, a facet is completely visible, if none of its points is covered, like R_0 and R_1 in the figure.

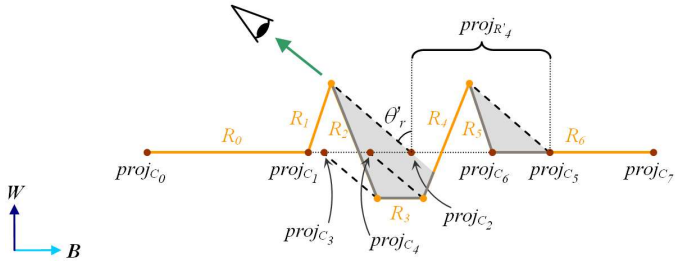


Figure 6: Masked facets are found by projecting the cross-section points in the order that are viewed. In this case, R_2 and R_5 are self-masked, and R_3 is completely masked by R_1. R_4 is partially masked by R_1 too, and R_6 by R_4.

According to this process, masking can be easily achieved by discarding the occluded facets and modifying those that are partially visible prior to the clipping operation. In order to account for the shadowing effect, this kind of projection can be done similarly for each light source. After these projections, facets found to be completely shadowed are discarded too, even when they are visible and inside the footprint. The illuminated portions of the partially shadowed facets are determined as before. These are then intersected with the corresponding visible segment/portion using an expression similar to (3). With this approach, if any groove has some facets protruding from the surface, the occlusion produced to the other grooves is directly considered.

In the case of grooves with curved paths, the obtained occlusion will be an approximation of the exact solution, because these are treated as straight grooves, as mentioned above. The accuracy of this occlusion depends on two factors: the groove curvature at the current point and the occlusion direction, i.e. the view or light direction. The lesser curved the groove is and the closer the direction to the cross-section plane, the more accurate the occlusion is. If at a certain point the groove is highly curved, the viewer or light source is far from its cross-section plane, and occlusion appears, then the computed effect can be very different from that of a straight path, because the projection of the blocking facet to the occluded facet is far from being parallel. In such cases, oversampling can suffice for the clipping, but probably not for the occlusion, so a better method should be considered for these cases. Likewise, this can happen with highly perturbed grooves, since the occlusion due to depth changes along the path is not considered.

4.4 Reflection contribution

The last step of the algorithm consists in computing the reflection of each visible and illuminated facet, adding it to the accumulated reflection of the footprint. This reflection will be computed in groove space, defined by T, B, and W. Each facet normal N_i in this space is $N'_i = (0, N_{i_b}, N_{i_w})$, and the incident L and reflecting E directions are transformed into L' and E', using the rotation matrix M_{rot} defined as:

$$M_{rot} = \begin{pmatrix} T_u & B_u & 0 \\ T_v & B_v & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{5}$$

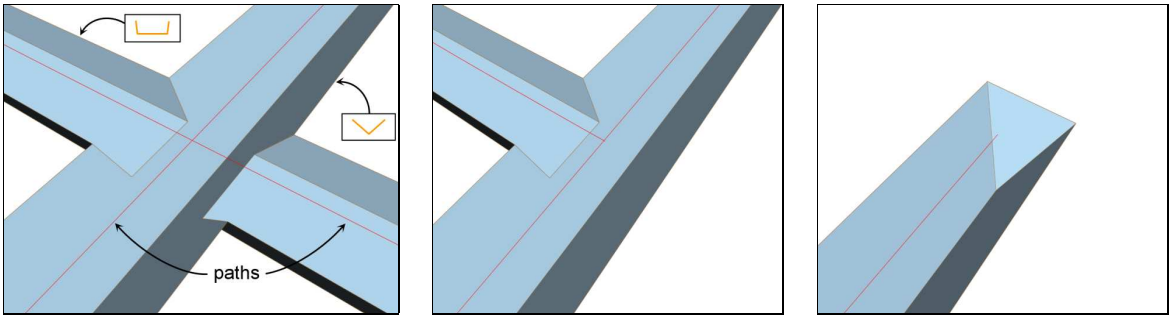
From these three vectors and the BRDF ρ_i of the current facet, the reflection contribution is computed by

$$contribution\ of\ facet = r_i\ \ \rho_i(N'_i, E', L')\ , \tag{6}$$

where r_i is the ratio of footprint area occupied by the facet. This ratio is determined from the size of the final facet segment $S_{R_i} = [S_{R_{i_0}}, S_{R_{i_1}}]$ after clipping and occlusion operations, and the size of the original footprint segment S, using the following expression:

$$r_i = \frac{S_{R_{i_1}} - S_{R_{i_0}}}{S_1 - S_0}.$$

Note that when the current facet is one of the extra facets used to represent the surrounding surface (see Section 4.2), the previous BRDF ρ_i will be the one specified for the base surface.



(a) Intersection (b) Intersected end (c) Isolated end

Figure 7: A different approach is used for such special situations.

5 Intersections and ends

Groove intersections and groove ends (see Figure 7) are special situations that require a different approach from the one proposed in the previous section. The different steps of the algorithm need to be modified to take them into account, and an extra step must be included between steps 2a and 2b, that is, between the clipping and occlusion operations. This new step is just used to compute the intersection between facets.

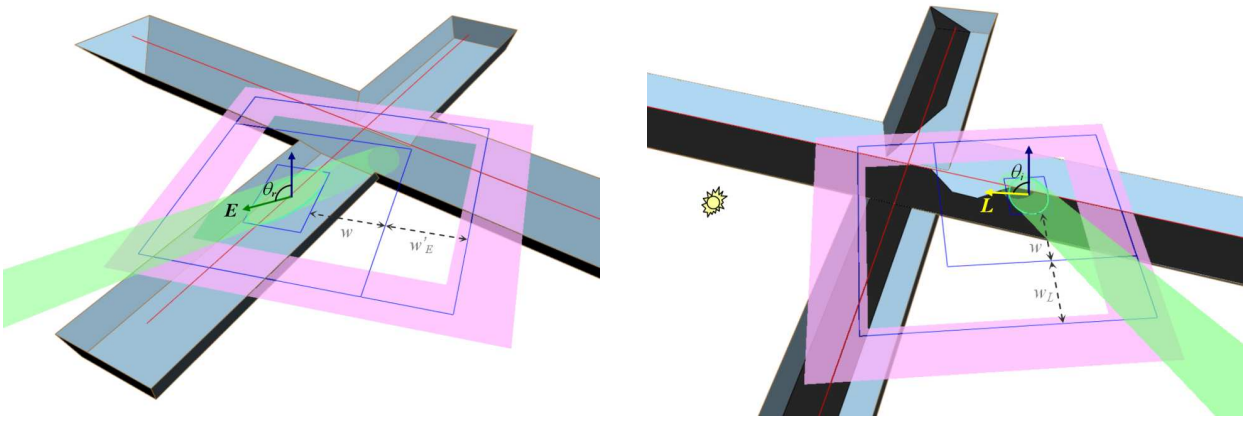
Concerning the groove ends, two kinds of situations are treated, these being the most common cases: intersected ends and isolated ends. The former represents grooves ending in the middle of some other groove, and can be considered as half-intersections (see Figure 7(b)). The latter represents common ends, appearing when a groove ends without being affected by another groove (see Figure 7(c)).

First of all, we must know how to determine if a footprint contains or is affected by some intersection or end. This must be performed for each footprint after the first algorithm step, where the grooves affecting it are found. If the footprint does not contain any intersection nor end it means that it contains an isolated groove or many parallel grooves, and the algorithm from Section 4 is used. Otherwise, the algorithm from this section is applied.

To quickly determine if the footprint contains a possible intersection (see Figure 7(a)), we simply test if two or more grooves inside the footprint's bounding box are not parallel, i.e. if their tangent directions T differ. In the case of ends (see Figure 7(b)-7(c)), for each groove we test if any of the two end points of its path lie inside the bounding box. If several grooves are present and some of them are ends, we must then test if the ends are isolated or intersected. Between an ending groove and a non-ending one, this is done by computing the distance between the end point of the former and the path of the latter, using an expression similar to (2). If this distance is less than the sum of their widths then we consider that they intersect, and they will be simulated as an intersected end. Depending on the directions of the two grooves, they could partially intersect even if their distance is greater than this value, but we will simulate them separately as an isolated end and a normal groove, in order to avoid strange intersections. If there is more than one end in the footprint's bounding box, we must also compute the distance between every pair of ends. When two ends are found to intersect, then they form a corner that, although not explained here, is a special case that can be simulated as a combination of intersected and isolated end.

In order to simulate isolated ends, many shapes could be considered for their final facets. Here we consider that groove ends are straight, and that the number and shape of their final facets is given by the cross-section of the groove, like the one shown in Figure 7(c). This kind of end will be simulated using a special case of intersection, by considering that the current groove is intersected by a sort of perpendicular groove with half the same cross-section. Other types of isolated ends, such as piecewise or curved ends, could be simulated using a similar approach, considering a collection of temporary half-grooves intersecting from different positions and directions.

Concerning the intersections, the previous test only works for intersections between different grooves. If self-intersecting grooves are allowed, a special case should be added for them. Before the rendering stage, we should first determine which grooves are self-intersecting and at which points. Then, if such a groove is contained inside the footprint as well as the intersection point, the groove should be simply treated as two independent grooves with different directions.



(a) (b)

Figure 8: (a) At intersections, some penetrating facets can be contained in the footprint far from the boundary of the groove. To find their path into the grid, the footprint bounding box must be enlarged according to w'_E, which is the depth of grooves projected in the inverse viewing direction. (b) Similarly, facets shadowed due to an intersected blocking facet can now remain illuminated. The path of the intersecting groove can be found by enlarging the footprint bounding box according to the depth of grooves projected in the light source direction w_L.

5.1 Finding grooves

In the first step of the algorithm, the objective is to determine the grooves affecting the current footprint. In Section 4.1 a special case was added for grooves protruding from the surface, in order to take into account facets seen/illuminated possibly far from the boundary of grooves onto the UV texture plane. Now that grooves can intersect each other, another case must be added for grooves penetrating the surface too. When an isolated groove is viewed from a grazing angle, its penetrating facets are usually masked by one of its neighboring facets. However, when this groove is intersected by another one, those facets might not be masked due to the intersection geometry and remain visible (see Figure 8(a)). To take these facets into account, their associated path should be found inside the footprint's bounding box too. This can be achieved by enlarging the bounding box according to the groove's maximum depth d and the viewing angle θ_r, using the following value:

$$w'_E = d \tan\theta_r \, .$$

For intersecting grooves, instead of enlarging the box following the viewing direction, this enlargement must be done in the inverse direction, as shown in Figure 8(a).

The case of shadowing is slightly different. If a penetrating facet is shadowed due to one of its neighboring facets, some portions might not be shadowed if another groove intersects the blocking facet (see Figure 8(b)). To account for this effect, we need to find the intersecting groove here too. This is similarly done using the depth d and the illumination angle θ_i, but here the bounding box must be enlarged in the same direction of the light source, like in Section 4.1. Thus, the enlargement due to protruding facets and intersecting grooves can be merged into a unique expression

$$w_L = (h+d) \tan\theta_i \, .$$

In short, when grooves can protrude from the surface and intersect each other, possibly masking grooves are found by enlarging the bounding box following the viewer direction and its inverse, and possibly shadowing grooves, by enlarging it only following the illumination direction. In Figure 8, we show how the bounding box is enlarged for each case. The pink cells represent the cells lying on the boundary of the enlarged box, which are used to find the paths. The grid cells are not shown for clarity.

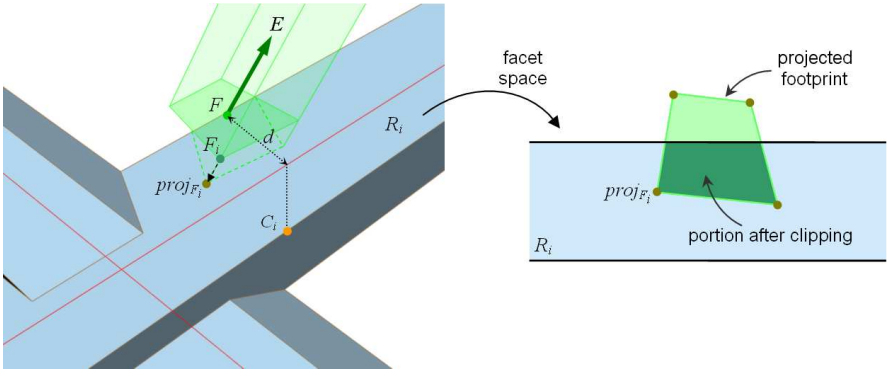


Figure 9: Left: footprint is projected onto each facet in 3D groove space, following the view direction. Right: once in 2D facet space, the footprint is clipped to the bounds of the facet, represented by two horizontal lines.

5.2 Footprint clipping

Most of the assumptions made in Section 4.2 regarding the projection and clipping of the footprint do not apply for intersected grooves or ends. In such situations, the shape of the grooves changes along their paths. They can not be well characterized only with the 2D cross-sections; we need to consider the local 3D shape of the grooves. Similarly, the footprint can not be represented by a single line segment. Neither the grooves nor their different edges inside the footprint will be parallel, if only a single segment is used some facets might not be well represented and others missed.

To correctly clip the footprint at intersections and ends, we propose to use the footprint's original shape. This shape will be represented by a polygon, defined by means of a set of 3D points F_i. If the footprint is originally represented by an elliptical shape, we approximate it by a rhombus or a quadrilateral, which are easier to intersect. The points of such a polygonal shape initially lie on the UV plane ($F_{i_w} = 0$), and these are defined relative to the footprint center F (see left part of Figure 9).

In order to compute the intersections and occlusions in 2D facet space, the footprint's polygonal shape is then projected onto each facet. This projection is done using common line-plane intersections in 3D groove space. This means that the footprint, the facet plane, and the direction of projection, must be previously transformed into this space for each of the grooves. First, the footprint points F_i must be rotated according to the matrix M_{rot} and translated by its distance d_P to the groove, resulting in $F_i' = F_i\, M_{rot} + (0, d_P, 0)$. The matrix M_{rot} is obtained from equation (5), using the T and B vectors of the groove path. The plane of a facet R_i is described by means of a normal and a point on it. The normal is the facet normal in groove space N_i', and the point is one of the cross-section points $C_i = (C_{i_b}, C_{i_w})$ of the facet, represented in such 3D space by $C_i' = (0, C_{i_b}, C_{i_w})$. Finally, the direction of projection is the view direction in groove space E'.

Next, given all the parameters, the projection of a point F_i' onto the facet plane in the direction of E' is computed by

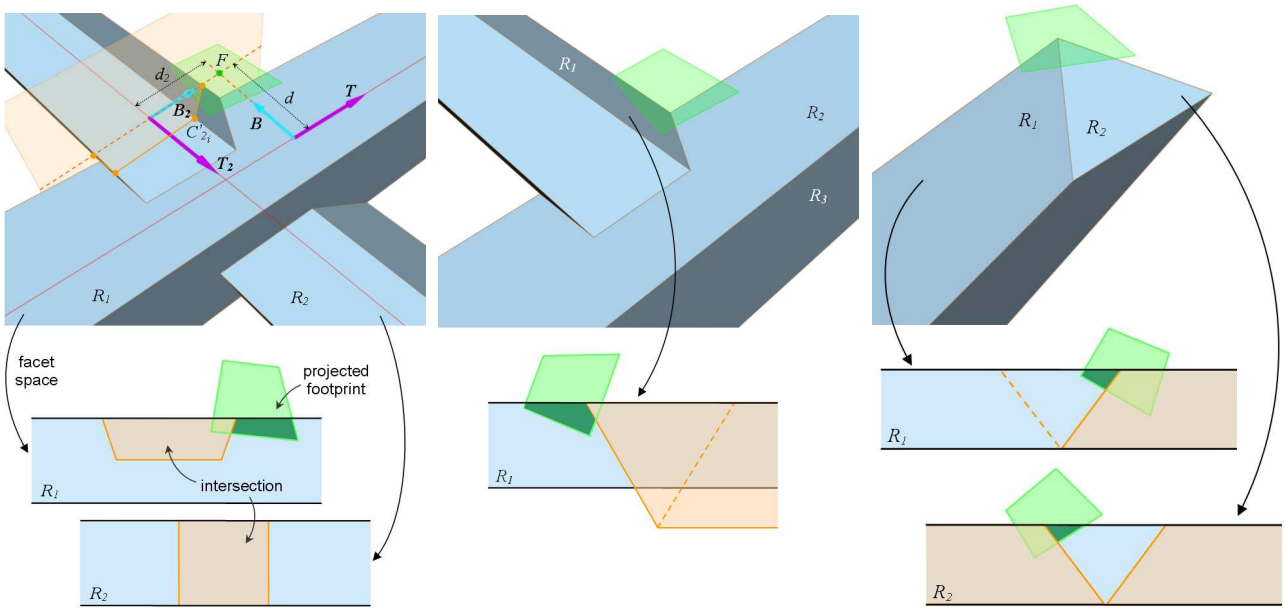
$$proj_{F_i} = F_i' + t\, E' \,, \tag{7}$$

where

$$t = \frac{N_i' \cdot (C_i' - F_i')}{N_i' \cdot E'} \,. \tag{8}$$

These equations must be computed only if the denominator of (8) is positive. When the dot product $N_i' \cdot E'$ is negative the facet is self-masked, and when it is zero, the facet plane is parallel to the viewing direction. In both cases, the facet does not contribute to the final reflection and is thus neglected. Otherwise, the resulting point $proj_{F_i}$ is a 3D point lying on the facet plane. In order to simplify computations, instead of transforming this point into the 2D facet space, we use the common approach of dropping the most representative coordinate of both the plane and the point. As a result, the clipping will be easily performed in an axis-aligned plane, while keeping the proportions of the facet and the footprint. The dropped coordinate is chosen to be the component for which the plane normal has the largest absolute value. In our case, it can be one of the two components corresponding to the cross-section BW space, since $N_{i_t}' = 0$.

To clip the footprint to the boundaries of the axis-aligned facet we can first perform a fast test to determine if the footprint falls totally inside or outside the facet. Since the facet plane is bounded by two horizontal lines (see right part of Figure 9), it can be easily done by comparing the height values of the footprint points with each boundary height. The



(a) Intersection (b) Intersected end (c) Isolated end

Figure 10: The new algorithm step projects the cross-sections of the intersecting grooves to the current facet in 3D groove space (top), and then intersects the footprint with these projected cross-sections in 2D facet space (bottom). For ends, the projected cross-sections must be extended before its intersection with the projected footprint. The dashed lines in the bottom of (b) and (c) show the originally projected cross-section.

footprint will be neglected if all the points $proj_{F_i}$ fall outside one of the boundaries, and used as is if all of them fall inside the boundaries. Otherwise, the footprint is clipped and its exact intersection with the boundaries is determined. This is done using a 2D polygon intersection.

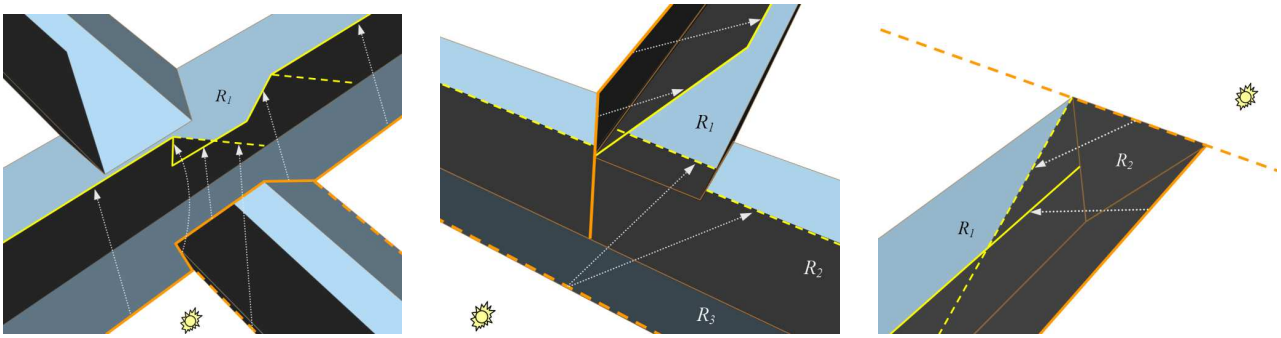
With respect to the surface around the grooves, remember that in Section 4.2 this base surface was included by adding ground facets between the grooves. Here, only a ground facet is necessary. In addition, since this facet is unbounded and lies on the UV texture plane, like the footprint, we neither need to project the footprint onto the facet nor to clip it. In the following intersection and occlusion steps we will determine which portion really belongs to the surface and which belongs to the grooves.

5.3 Intersecting facets

This new step has been included to account for the portion lost during an intersection. When two or more grooves intersect, part of their facets is lost, and it should be removed from the polygon obtained in the previous step. As shown in Figure 10, this portion is the cross-section of the intersecting groove once projected onto the current facet, and it can be computed using the previous expressions (7) and (8) too, using the groove direction T_2 as the projecting direction and its points C_{2_i} as the points to be projected. Each point $C_{2_i} = (C_{2_{i_b}}, C_{2_{i_w}})$ must be previously transformed to the current groove space, according to the binormal vector $B_2 = (B_{2_u}, B_{2_v})$ and the footprint's distance to each groove d_P and d_{P_2} (see top of Figure 10(a)), as

$$C'_{2_i} = \left(B_{2_u}(C_{2_{i_b}} - d_{P_2}),\ B_{2_v}(C_{2_{i_b}} - d_{P_2}) + d_P,\ C_{2_{i_w}} \right) ,$$

The cross-sections of all the grooves found in the footprint must be projected onto the current facet using this approach, except the current cross-section and those for which the groove is parallel to the current one, for example when two parallel grooves are intersected by a third one. We must also take into account the particular cases, like when the current facet is parallel to the UV texture plane, such as R_2 in Figure 10(a) or the facet representing the surrounding surface, because the projection direction is parallel to the facet and the denominator of Equation (8) becomes zero. In these cases, the



(a) Intersection (b) Intersected end (c) Isolated end

Figure 11: During occlusion the blocking facet (continuous orange profile) and the prolongation of the intersecting grooves (dashed orange segments/lines) must be projected onto the current facet (R_1 in these examples). Here the shadowing effect is computed by projecting them following the light direction. Once in facet space, the final shadowing profile is obtained after unifying the projected blocking facet (continuous yellow profile) with the projected prolongations (dashed yellow segments/lines).

intersected portion will be defined by a set of parallel lines. These lines pass through the points where the cross-section intersects with the current facet's height, and they follow the groove direction. In Figure 10(a), for example, the intersected portion of R_2 is defined by two vertical lines, because the cross-section of the intersecting groove crosses the facet twice and the grooves are perpendicular.

Other special cases are intersections between protruding and penetrating grooves or intersections where one of the groove predominates over the other, which sometimes happens with scratches. These cases, although not considered here, could be easily included into our method if desired.

Concerning the ends, some special considerations must be examined too. These depend on the type of end and on the groove where the current facet belongs to. In the case of intersected ends, when treating a facet belonging to the ending groove, we must modify the projected cross-section previously obtained. As shown at the bottom of Figure 10(b), this projected cross-section is extended following the ending direction. On the other hand, when treating a facet belonging to the non-ending groove, it depends on the side where the facet lies. If it lies on the intersected side, like R_2 in top of Figure 10(b), then we act as in a normal intersection, using the projected cross-section as it is. If it lies on the opposite side, like R_3 in the same figure, then no intersection happens and nothing needs to be done.

In the case of isolated ends, when treating a facet belonging to the ending groove, we need to extend the projected cross-section here too, but without including the cross-section in itself (see R_1 in Figure 10(c)). For facets belonging to the temporary half-groove, the projected cross-section must be extended on both sides, but not including the cross-section this time, like R_2 in the same figure.

All these approaches with minor changes also apply for facets protruding from the surface.

5.4 Occlusion

After intersecting the facets, the next step is the computation of the occlusion effect. Occlusions occurring on isolated and parallel grooves are simpler than the ones occurring on intersected or ending grooves. In the former, provided that the scene does not contain area light sources, occlusions appear as straight lines following the groove direction, so these can be easily computed using the approach from Section 4.3. In the latter, the same lines must include the cross-sections of the intersecting grooves and their prolongations, thus further projections are required. These are shown in Figure 11 in orange. As can be seen from 11(a) to 11(c), the blocking facet and the prolongation lines that must be considered depend on the kind of situation. Prolongations represent straight open-ended segments starting at the highest points of an intersecting groove (peaks) and following the direction of this groove. For the shadowing case of the figure, these segments and the cross-section are projected onto the current facet R_1 following the light source direction. The result of these projections is shown in yellow. Once in 2D facet space, the projected segments and cross-section will be unified

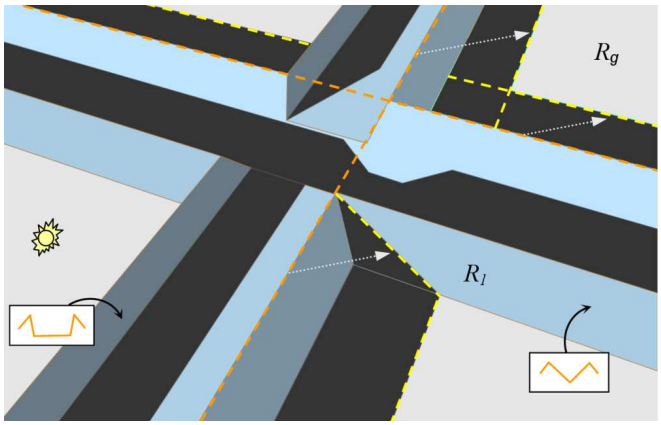


Figure 12: If some facets protrude from the surface, these facets and the surrounding ground facet can then be occluded. To take into account their occlusion, only the prolongations of grooves need to be projected.

to determine the final shadowing profile, which needs to be removed from the footprint polygon obtained in the previous step.

Usually, it is not necessary to perform entirely this set of projections for all the facets. If a facet is self-shadowed with respect to a light source, no shadowing must be further computed due to this light, and if this happens for all the light sources, not even masking is necessary and the facet is discarded. In groove space, for a given light source with direction L', the current facet will be self-shadowed if $N_i' \cdot L' < 0$. Here, self-masking tests are not required, since they were performed during the clipping step.

Once self-shadowing is tested, we must check if the current facet can be blocked by another facet of the same groove and from which one. Remember that this information was precomputed in a previous stage (see Section 3), thus we simply must retrieve it. If no facet can block the current one, there will be no occlusion and we jump to the next algorithm step. Otherwise, the above mentioned projections should be performed for each of its blocking facets and for each viewing and lighting direction, but only if the blocking facet is self-occluded in this direction. When a facet is not self-occluded, it can not cause occlusion to the other facets.

In the case of grooves with facets protruding from the surface, occlusion can also appear on the external facets of grooves as well as on the surrounding surface, but this can be taken into account by just projecting the prolongations, as shown in Figure 12. In this example, the external facet R_1 is shadowed by one of the prolongations/peaks of the intersecting groove, while the ground facet R_g is shadowed by two of them, each one from a different groove.

As in the previous step, further considerations must be taken into account for ending grooves, also depending on the type of end. In the case of intersected ends, when computing the occlusion for a facet on the ending groove, we must only project half of its blocking facet and one of its prolongations (see orange profile in Figure 11(b) for facet R_1), although the latter is a line instead of a segment. For a facet on the non-ending groove, this projection depends on the blocking facet: if the blocking facet is intersected, its projection is done as in a normal intersection, otherwise a single line is projected, as on isolated grooves. In Figure 11(b), R_3 is not intersected, thus the occlusion onto R_2 results in a single parallel line.

In the case of isolated ends, no blocking facet contains a cross-section, as can be seen on Figure 11(c), therefore projections are simpler. For a facet belonging to the ending groove, such as R_1 in the figure, only the line coming from its opposite blocking facet and the one from the temporary half-groove must be projected. For facets belonging to the half-groove, such as R_2, their occlusion can be determined by just projecting the line given by one of the possibly blocking facets from the ending groove. In the example of Figure 11(c), however, this projection is not required for R_2, because it is already self-shadowed.

5.5 Reflection contribution

At this point, the result of the previous steps is a polygon or a collection of polygons. These represent the portion of the current facet contained in the footprint that is visible and illuminated. The contribution of this facet to the final footprint's reflection is computed using (6) as before. The only difference is that the area ratio r_i here is determined from the area of the polygons. Being A_F the area of the footprint polygon once projected onto the facet, and A_R the area of this polygon

after clipping, intersection, and occlusion, the ratio will be

$$r_i = \frac{A_R}{A_F} .$$

5.6 Efficiency issues

The algorithm presented in this section requires several projections and intersections of polygons that can be very time consuming, especially if a surface contains a lot of intersecting or ending grooves. Here we show how this algorithm can be improved depending on the situation.

On close views, for example, most of the facets from the grooves at the current footprint do not contribute to its final reflection, but are also treated by the algorithm. To avoid it, we can simply start processing the facets lying on the same side as the footprint and stop when the contribution of the footprint is fulfilled, i.e. when $\sum r_i \approx 1$. A further improvement is obtained if reusing the projected cross-sections from one footprint to another, using a cache structure. This is especially useful when cross-sections and projection directions do not vary between them. With these improvements, the algorithm performs less projections but the speed up is not very important in our implementation ($\approx 10\%$), because the most consuming part is the polygon intersection.

To further improve this, we have replaced the polygons by a set of representing segments, as done for isolated and parallel grooves in Section 4. For intersections, one line segment would not be enough as previously stated (see Section 5.2), thus we have decided to use two perpendicular segments. The axes A_1 and A_2 of the footprint are used for this purpose. Each axis is considered as a 3D oriented segment, and is independently treated. It is projected onto each facet, clipped, and subtracted with the intersection and occlusion profiles, as done with the polygons. After processing a facet, the result is another segment or a collection of segments. The total length of these segments divided by the original length of the segment gives its area ratio r_i. From this ratio the reflection of the facet is computed and added to the accumulated reflection, also as before. After all the facets have been processed, and thus the algorithm, the result is a reflection value for each of the two segments.

Once the two values for the segments are obtained, we must know how to combine them in order to obtain the best representation of the footprint reflection. Some years ago, Jones and Perry [JP00] proposed a method based on line sampling but performed in screen space instead of texture space. The authors used two similar perpendicular segments, which were combined according to a weighting. Each segment weight was computed as the sum of a set of $\sin^2 \alpha$ values, where α was the angle between the line segment and each of the intersected edges. Such a weighting gives more importance to the segment intersecting the edges at angles close to the perpendicular than those close to zero.

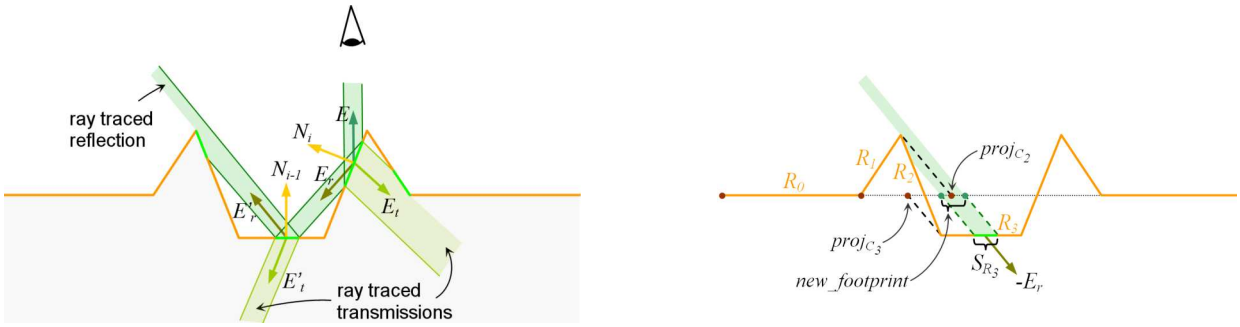
Here, we have decided to use the same weighting, computing the α angle during the intersection of our segments with the edges of the facets or the projected profiles. Using two line samples the algorithm speeds up considerably ($\approx 40\%$) and the reflection values are quite similar to the ones obtained using polygons. However, as the viewer moves away from the surface, some facets are missed by the lines and the error can be considerable too. In such cases, one solution is the use of more line samples, as proposed by Jones and Perry, but a polygon could perform better antialiasing with a similar computation time. Furthermore, a grooved surface usually contains few intersections and ends, so the increase in computation time is nearly imperceptible when using polygons.

6 Indirect illumination

On the previous sections, the illumination of a grooved surface is determined only taking into account the light arriving directly from the light sources, that is, the direct illumination. In order to achieve more realistic results, the indirect illumination should be also considered, which is due to the multiple reflection or transmission of the light on the same object and to the light coming from the other objects of the scene. Depending on the behavior of the involved surface, this light is reflected or transmitted in a different set of directions. This is commonly separated into diffuse and specular behaviors.

6.1 Specular reflections and transmissions on the same object

Specular reflections and transmissions can be easily included in our algorithm. It mainly consists in recursively executing the algorithm for each bounce of the light, but following the eye direction, similar to a beam tracing approach [HH84]. The idea is to re-run the algorithm for each visible facet found for the current footprint, using the visible portion of the



(a) Inter-reflections and transmissions on a groove

(b) Computing an inter-reflection of an isolated groove

Figure 13: (a) Cross-section view of a groove undergoing inter-reflections and transmissions. These are taken into account by recursively calling the algorithm for each bounce of the light, starting from the eye. At each bounce, the facet's visible portion and the reflection vector E_r replace the initial footprint and vector E, respectively. The refraction direction E_t is used for transmissions. When the new footprint is not entirely projected onto the facets, we ray trace the scene to include the illumination from the other objects. In this example, only some bounces are shown for the initial footprint. (b) Computing the illumination of facet R_3 given the reflection direction $-E_r$. The visible portion of R_3, S_{R_3}, is previously projected onto the base texture plane to become the new footprint. Here, the same groove contributes to the illumination. The only facets that are considered are those lying in the opposite side or $-E_r$, here R_0 to R_2. Since it is an isolated groove and $-E_r$ comes from under the surface, masking tests must be inverted. R_2 is not self-shadowed, for example, even though its two points lie in the opposite order once projected.

facet as the new initial footprint, and the reflection or transmission direction as the new view vector E (see Figure 13(a)). For this, an extra step has been included at the end of the algorithm after the computation of the direct illumination in step (2c). This new step is computed using the following code:

if ($depth > 0$) **then**
  if (*facet_reflects_light*) **then**
    E_r = ComputeReflectionVector(N_i'', E)
    new_footprint = ProjectFootprintOntoSurface(*clipped_and_masked_footprint*, E_r)
    I_r = Algorithm($-E_r$, *new_footprint*, $depth - 1$)
    $I = I + r_i\, k_r\, I_r$
  endif
  if (*facet_transmits_light*) **then**
    E_t = ComputeRefractionVector(N_i'', E, *index_of_refraction*)
    new_footprint = ProjectFootprintOntoSurface(*clipped_and_masked_footprint*, E_t)
    I_t = Algorithm($-E_t$, *new_footprint*, $depth - 1$)
    $I = I + r_i\, k_t\, I_t$
  endif
endif

As it can be seen, the recursive execution of the algorithm is first guided by the parameter *depth*. This parameter is specified by the user, and represents the maximum number of light bounces that will be considered, i.e. the maximum depth of recursion. If the current depth is 0 the recursive execution is stopped. Otherwise, we test if the facet reflects or transmits light, according to its BRDF and BTDF. If this is the case, we determine the new viewing vector and footprint, we re-execute the algorithm with them, and the result is added to the current accumulated reflection color I.

The new view vector, E_r for reflection and E_t for transmission, is computed with the classical ray tracing expressions [Gla89], using the current vector E and the facet's normal N_i'', both in texture space. E_t also uses the index of refraction of the facet. Since the algorithm expects a vector pointing towards the viewer, the new vector will be inverted before calling the algorithm. In the case of the new footprint, it is just the portion of the current facet visible by the current

footprint, i.e. once clipped, masked, and intersected. When the current footprint contains isolated or parallel grooves, the portion, and thus the new footprint, will be represented by a segment (see Section 4.2). When it contains intersections or ends, the portion will be represented by a polygon (see Section 5.2). Before recursively executing the algorithm, the portion must be projected back from the facet to the texture plane, because the algorithm expects an initial footprint lying on the UV plane. This is done by the function *ProjectAndTransformFootprint*.

During the recursive execution of the algorithm, some considerations must be taken into account. In the first step of the algorithm, for example, the found grooves affecting the new footprint could be the same of those in the previous footprint or new ones. If the new vector E comes from above the surface, this works as usual, enlarging the footprint's bounding box an amount of w_E in the direction of E (see Section 4.1) or an amount of w'_E in the inverse direction (see Section 5.1). However, reflection or transmission vectors can potentially arrive from under the surface (see Figure 13). In such cases, E_w and the value of $\tan\theta_r$ obtained in (1) are negative. Nevertheless, this only means that the bounding box must be enlarged with the absolute amounts in opposite directions. This also affects the masking step of Section 4.2. When $E_w < 0$, we must invert the order in which the different cross-section points are projected and then tested for masking. As shown in Figure 13(b) for R_2, if E_r comes from under the surface then a facet will not be self-shadowed, for example, when $proj_{C_{i+1}} < proj_{C_i}$, but when $proj_{C_{i+1}} > proj_{C_i}$. Finally, another consideration must be taken into account during the recursive runs of the algorithm. If the illumination reflected or transmitted by the current facet comes from a facet on the same groove, only the facets lying on the visible side of the groove must be considered. This must be done in order to avoid the incorrect masking of such facets from the other ones. For example, in Figure 13(b) the illumination of R_3 coming from E_r is partially due to a facet on the same groove, R_2. When the algorithm is re-executed to account for this reflection, the same groove will be found during the first step, but at the other steps, only its facets R_0 to R_2 from the left will be processed, since the others do not contribute for the current E_r direction.

After the algorithm is recursively executed, the resulting illumination value, I_r or I_t, is added to the current accumulated reflection color, as stated before. Each value is previously multiplied by the corresponding specular term from the BRDF or BTDF, k_r or k_t, as well as by the current r_i value. Remember from Section 4.4 and 5.5 that this value is the ratio of footprint area occupied by the current facet. However, unlike with the direct illumination, here it is computed not including the shadowed portions of the facet.

6.2 Illumination from other objects

The new step introduced before along with the aforementioned changes are used to take into account inter-reflections and transmissions occurring on the same grooved surface. If the light arriving from the other objects of the scene wants to be included, as well as the light transmitted by other parts of the same object, i.e. side or back faces, another step must be introduced to our algorithm. This new step will be executed at each recursion level and at the end of the algorithm, when all the current grooves and their facets have been completely processed. If at this point, the contribution of the footprint has not been fulfilled, i.e. when $\sum r_i < 1$, it means that part of the reflected or transmitted footprint does not project onto the neighboring grooves, but bounces towards the rest of the scene (see Figure 13(a)). In such cases, the contribution from the other objects is computed using ray tracing, tracing a ray through the scene in the corresponding direction:

if ($\sum r_i < 1$) **then**
  E_c = TransformFromTextureToCameraSpace(E)
  I_R = RayTrace(E_c)
  $I = I + (1 - \sum r_i)\, I_R$
endif

The incoming illumination returned by the ray tracer will then be added to the final footprint reflected illumination, taking into account that its contribution weight is $1 - \sum r_i$.

6.3 Glossy and diffuse reflections

Although not considered in this paper, glossy and diffuse reflections or transmissions could be included using the same approach described above. The main difference will consist in recursively executing the algorithm using a set of randomly chosen directions for E_r or E_t, instead of only using the perfectly specular direction. Then, the illumination due to each of these directions should be weighted and averaged, as done in distribution ray tracing [CPC84] or path tracing [Kaj86].

In the case of glossy or blurred reflections, an easy way to approximate them is to simply enlarge the footprint before re-executing the algorithm. However, the result might not be very realistic.

	Our method		Geometry		Gain	
	Time	Memory	Time	Memory	Time	Memory
Figure 14	26	16 (15)	289	656	11.1	41
Figure 15	47	14 (13)	202	4232	4.3	302
Figure 16 (a)	20	14 (13)	71	4232	3.5	302
Figure 16 (b)	13	14 (13)	71	4232	5.4	302
Figure 17	46	16 (15)	387	656	8.41	41
Figure 18 (a)	258	9251 (283)	–	–	–	–
Figure 19 (top-left)	31	486 (124)	–	–	–	–
Figure 19 (top-right)	43	486 (124)	–	–	–	–
Figure 20 (a)	12	149 (49)	–	–	–	–
Figure 20 (b)	164	149 (49)	–	–	–	–
Figure 20 (c)	290	149 (49)	–	–	–	–
Figure 20 (d)	382	149 (49)	–	–	–	–
Figure 21 (a)	81	863 (679)	–	–	–	–

Table 1: Performance of our method for each scene. Rendering times are in seconds. Memory consumptions are in kilobytes, and represent the total memory requirements of the geometry model. In parentheses we show the consumption due to our representation, without considering the underlying mesh. For the first four figures, we compare the performance between the scene modeled and rendered with our method and the one entirely modeled with geometry and ray traced using adaptive oversampling. The gain of our method is shown in the last two columns, which is computed by dividing the amount of time and memory for the geometry by those for our method.

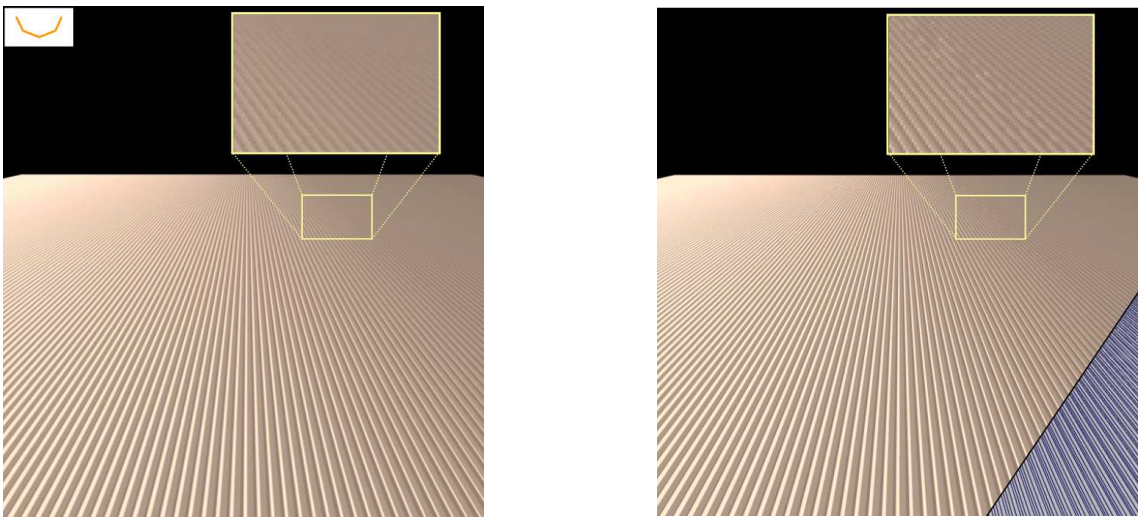
7 Results

Our method has been implemented as two plug-ins for the Maya® software. A procedural texture helps to place the grooves onto the surface, while a shader renders them at run-time. All the images and timings in this section have been obtained on a Pentium 4 processor at 1.6 GHz, using Maya's software renderer based on ray tracing. The images using our method have been ray traced using only 1 sample per pixel. The cross-section used for the grooves is included on the upper left of these images.

The timings and memory consumption are included in Table 1. Notice that the memory required by our model, shown in parentheses, mainly depends on the resolution of the grids that is used to store the paths. For the images in this section, we have mainly used grid resolutions of 100×100. Although this usually depends on the distribution of the grooves over the surface, it is difficult to determine which is the best resolution for the grids. Low resolutions make the footprint's bounding box to include grooves far from the footprint that do not affect its reflection, thus the rendering time tends to increase. High resolutions, instead, are more accurate when finding the grooves and less grooves need to be finally treated. Nevertheless, the savings in time tends to be in the order of a few seconds, for the rendering parameters used in our executions, while the memory increase can be noticeable. In our tests, we have found that grid resolutions of 100×100 or 200×200 have low-memory consumption and good rendering times.

First, we compare our method to a method based on adaptive oversampling. This will allow us to test both the performance of our method and the accuracy of the results. In the former, grooved surfaces are modeled using the representation of Section 3, by means of paths and cross-sections. In the latter, they are entirely modeled with geometry, that is, with the grooves included in the geometry model of the surfaces. In order to easily model the grooved meshes, we have made the comparisons using relatively simple models, where meshes have been modeled with some thousands of polygons. Although maybe the memory amounts stated here are not very important, notice that these increase rapidly for complex grooved surfaces, like those containing curved grooves. When such surfaces must be modeled with meshes or by displacement mapping the geometry, the resulting number of polygons can be very important, and thus the required memory.

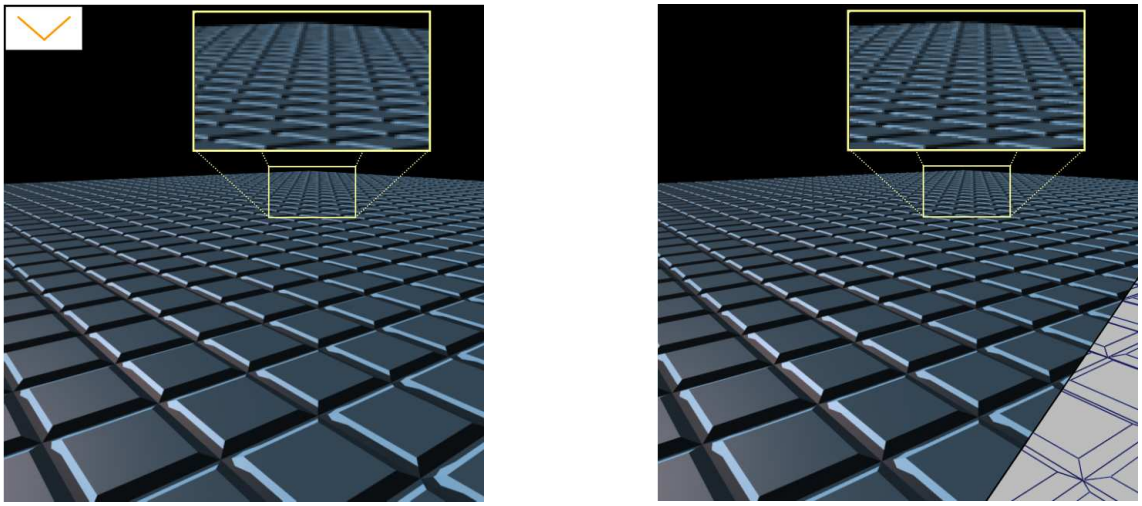
The first comparison is made with a surface containing parallel grooves, in order to test the method of Section 4. Figure 14 shows a surface containing lots of parallel grooves, each having the same cross-section and without any space between them. Figure 14(a) has been modeled using a flat surface, 250 parallel paths, and a unique cross-section. Then, it has been rendered using our method, only taking into account direct illumination. The image shows a smooth transition from near to distant grooves, i.e. from macro-geometry to micro-geometry. The difference in color from left to right on



(a) Our method

(b) Geometry with adaptive sampling

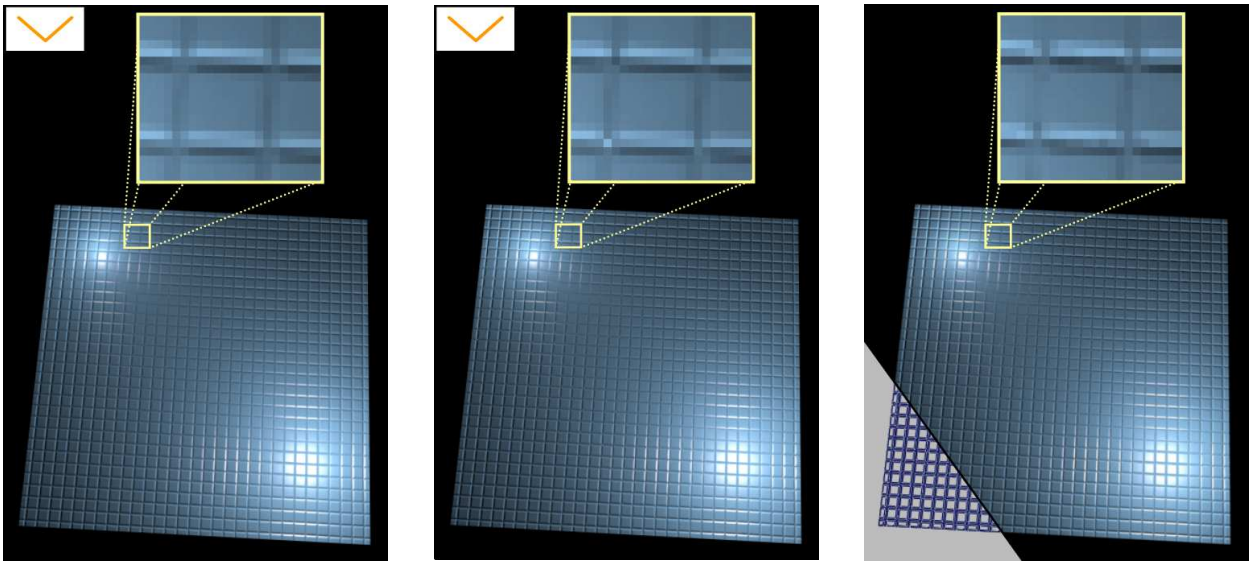
Figure 14: Surface containing lots of parallel grooves showing: smooth transitions from near to distant grooves, masking, and shadowing. (a) Modeled and rendered using our method. (b) Modeled onto the surface geometry model and rendered using adaptive oversampling. The mesh model is shown on the bottom right.



(a) Our method

(b) Geometry with adaptive sampling

Figure 15: Surface containing many intersecting grooves. (a) Modeled and rendered using our method. (b) Modeled onto the surface geometry model and rendered using adaptive sampling. The mesh model is shown on the bottom right.



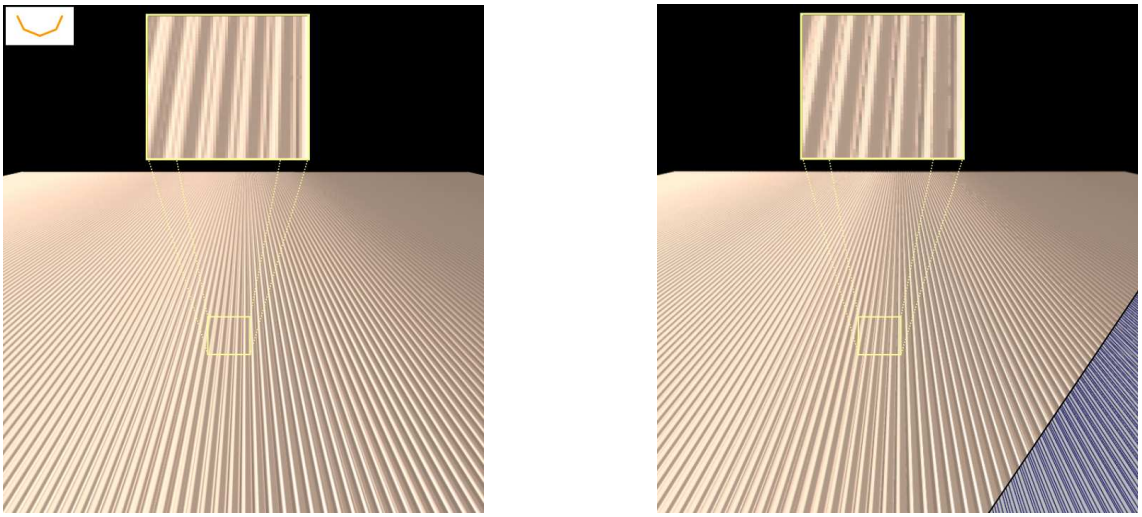
(a) Our method using polygon sampling (b) Our method using line sampling (c) Geometry with adaptive sampling

Figure 16: Models of Figure 15 illuminated with two point light sources and rendered from a different point of view. (a) Using our method based on line sample for isolated grooves, and on polygons for intersections. (b) Using 2 line samples for intersections. (c) Mesh geometry ray traced with 16 adaptive samples. The mesh model is shown on the bottom left.

the far side is due to the masking effect, and shadowing effect is present in all the grooves due to a light source placed far away on the right part of the scene. Figure 14(b), on the other hand, has been modeled using 1000 polygons, and then ray traced with 1 shadow ray and 8 adaptive samples. As can be seen, the result obtained with our method is as accurate as with adaptive oversampling. However, our method is one order of magnitude faster and requires less memory (see Table 1). Aliasing is visible at some places in both images. In our case, it is due to the fact that we use a simple box filter for antialiasing. This aliasing, however, could be reduced using other filter shapes, such a Gaussian filter. Although not implemented here, the summed-area table proposed in [JP00] could be easily included in our method. Nevertheless, as shown in the close views on the top of both Figures 14(a) and 14(b), the aliasing is less perceptible with our method.

In Figure 15 we compare the two methods using a tiled surface consisting of many crossing grooves, with transitions from near to far grooves, masking, and shadowing effects too. Figure 15(a) has been modeled and rendered using our approach. When the footprint only contains a groove or many parallel grooves, the line segment approach from Section 4 is applied. At intersections, the polygon approach from Section 5 is used instead. Figure 15(b) has been modeled with a mesh of 4681 polygons and ray traced using 1 shadow ray and 8 adaptive samples, as before. The two images are nearly indistinguishable. Since polygons require more computations and many intersections are present, it is not as faster as with only parallel or isolated grooves. However, the one rendered with our method is 4 times faster than using adaptive sampling, as shown in Table 1, and also requires less memory. Note that the image is correctly antialiased even for nearly grazing angles, because we consider oriented footprints. In Figure 16 the same surfaces are illuminated using two near point light sources and rendered from a distant point of view. In this case, for the grooved surface modeled with our representation, we compare its rendering using our polygon approach (see Figure 16(a)) with a rendering using the 2 line samples approach proposed in Section 5.6 (see Figure 16(b)). For the image in Figure 16(c), the mesh has been rendered using up to 16 samples per ray here, in order to obtain less aliasing artifacts with respect to the other two images. The times in Table 1 show that the line sampling approach is faster than the polygon one, but as shown in the top of the figures, antialiasing is better performed by the polygon approach, because it samples the entire footprint area.

Figure 17 shows the same images of Figure 14 but including inter-reflections inside the grooves, thus obtaining brighter surfaces. In Figure 17(a), we have used the recursive approach of Section 6, while in Figure 17(b) the mesh has been ray traced including reflection rays. For both images, only one light bounce has been considered. As shown in Table 1, the rendering time has increased in both cases, still being faster in our method. Naturally, these timings would further increase if more bounces were considered. The close view on top of Figure 17(b) shows that even more aliasing artifacts



(a) Our method

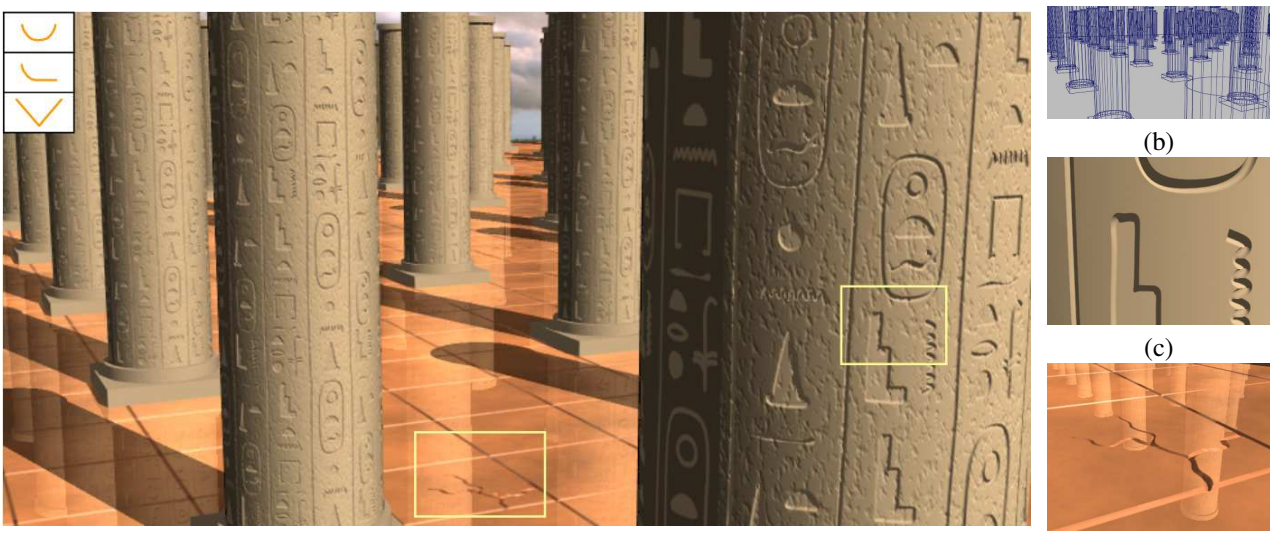
(b) Geometry with adaptive sampling

Figure 17: Models of Figure 14 rendered taking into account inter-reflections inside the grooves. Only one bounce of light is considered.

are present when reflection is adaptively sampled, thus more samples should be used in this case. Our method, instead, correctly samples inter-reflections (see top of Figure 17(a)).

The following examples show different kinds of scenes containing grooved surfaces and similar details, that can profit from our method. In Figure 18 we can see a scene composed of many stone columns with hieroglyphic details over a tiled floor. Hieroglyphics are simulated with different kinds of paths and two curved cross-sections (see the two upper left cross-sections in Figure 18(a)). Narrow details are simulated using the first cross-section. In some cases, for example in the snakes, the cross-section changes along the path. Wider details such as circles or triangles are simulated using a contouring path and half the same cross-section, which is easier than using a straight path and highly perturbing the previous cross-section. To simulate the erosion of columns we have also applied a bump map onto the surfaces. This modifies the initial surface normal which is later used by our method, thus it affects the grooves too. The tiled floor is composed of many straight grooves intersecting perpendicularly, using the last cross-section (see the third upper left cross-section in Figure 18(a)). It also includes a fracture on one of its tiles that uses a similar cross-section. A closer view of this fracture is shown in Figure 18(d). The floor has specular properties and includes inter-reflection too. It also shows smooth transitions from near to distant grooves, as well as the columns. The rendering time and memory consumption for this image are shown in Table 1. The time is mainly due to the ray tracing of the scene for the inclusion of the shadows between the different objects as well as the reflections of the columns on the floor. If shadow and reflection rays are neglected, the scene is rendered in 77 seconds, in front of the 252 seconds in Table 1. The memory requirements of this model are mainly due to the underlying mesh of the columns, composed of $\approx$7000 polygons altogether (see Figure 18(b)). In Figure 18(c) we show a closer view of some hieroglyphics on the near column. This image exhibits how the curvature of the paths affects the computation of their occlusion, as explained in Section 4.3. Bump mapping is not included to better see these effects. The shadowing for the upper narrow groove is correctly simulated, because of the smooth curvature of its path, but for the lower groove, it is not correctly simulated at the curved corners, due to the high curvature of these points. This could be solved by considering the curvature during the computation of the occlusion.

Figure 19 shows another example of a grooved surface: a vinyl. Although vinyls are made of a single groove with a spiral path, here we have modeled them using lots of small concentric grooves, which is easier to model. We have not left any space between the different grooves, except for the separation between tracks or songs. Concerning the cross-section, each vinyl on the top and lower left of Figure 19 uses a different cross-section, shared by all the grooves. The last vinyl, instead, has been modeled using three different cross-sections, randomly applied to the grooves (see right and middle images on bottom of Figure 19). The images on the left and middle columns have been rendered from a far viewpoint, so that each footprint is projected onto many grooves. This results on an anisotropic reflection effect on the overall vinyl.



(a) (b) (c) (d)

Figure 18: (a) Scene composed of many grooved surfaces. The hieroglyphics of the columns and the tiled floor have been simulated with our method, as well as the fracture on the floor. Bump mapping is included to simulate the erosion on the columns. The image shows different kinds of grooves and situations: narrow and wider grooves, perturbed grooves, intersections, and ends. It also shows smooth transitions on both types of surfaces and inter-reflections on the floor. (b) Wireframe of the underlying mesh. (c) Closer view of some hieroglyphics without bump mapping, showing how shadowing is not well simulated on highly curved corners. (d) Closer view of the fracture.

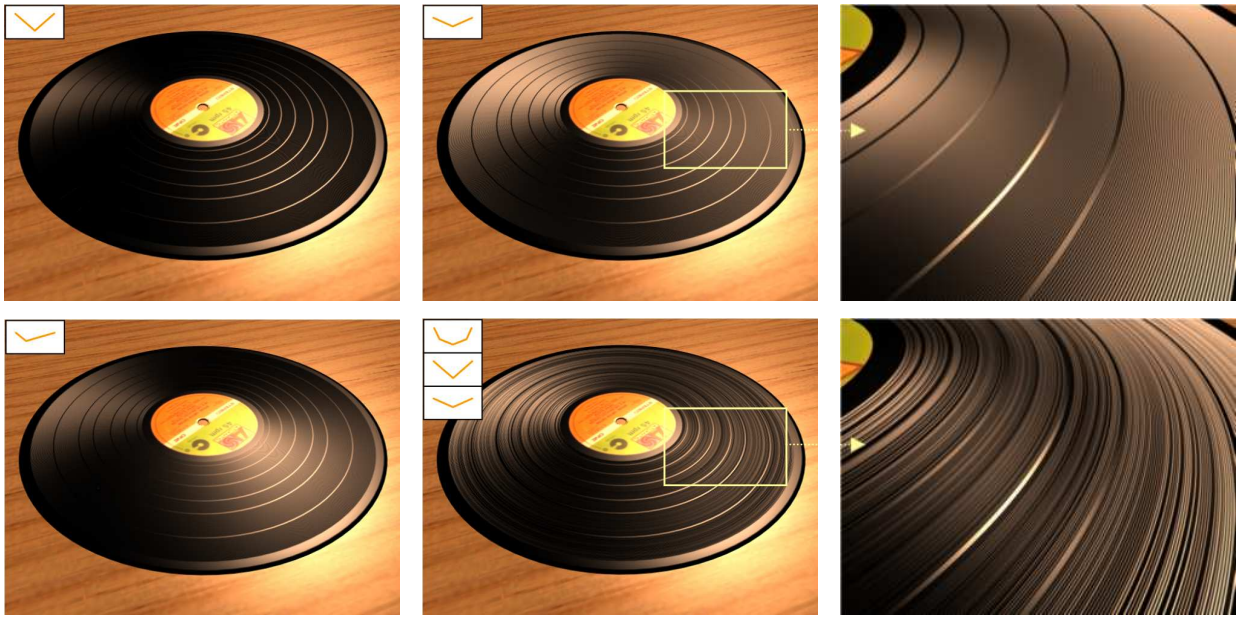
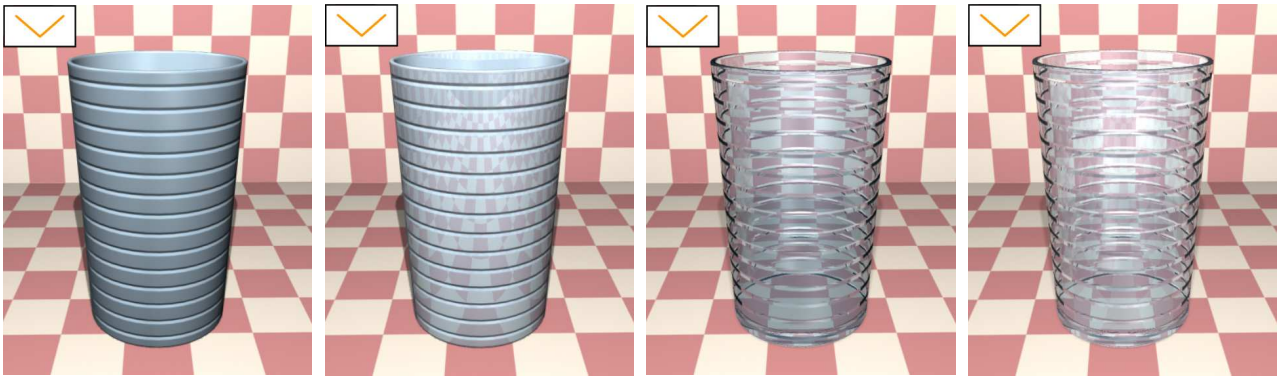


Figure 19: Vinyls are modeled using lots of small concentric micro-grooves. A different cross-section is used for each vinyl, shared by all of its grooves, except for the middle and right vinyls on the bottom, which use three cross-sections randomly applied. Right images represent closer views of the vinyls in the middle column.



(a) Direct illumination (b) Reflection (c) Refraction (d) Reflection and Refraction

Figure 20: Glass with grooves in the outside, rendered with different types of illumination.

As can be observed, the reflection of the vinyl clearly depends on the type of cross-section that is used. The parallel and symmetrical grooves of the upper vinyls could be simulated using the anisotropic reflection model of [PF90], by approximating the profiles with cylinders. However, this would not work for the asymmetrical cross-section of the bottom left image, nor for the variability of grooves present on the bottom middle image. These can not be simulated only using an anisotropic BRDF model. If shift-variant anisotropic BRDFs were used, an approximate effect could be probably obtained. Nevertheless, both kinds of methods are only valid for a certain distance of the viewpoint. Our method, instead, can render the grooves accurately no matter the distance or the point of view, as demonstrated by the images on the right of Figure 19. These represent the vinyls on the middle seen from a closer view, showing the details of their grooves.

In Figure 20 we have rendered a grooved glass by considering different types of illumination. Figure 20(a) first shows the glass rendered only taking into account the direct illumination. In Figure 20(b) we have included the indirect illumination due to inter-reflections, which mainly arrives from the other objects of the scene: the floor and the walls. The algorithm has been recursively executed with a depth of 2, and for each depth, the ray tracing to the other objects has been performed using 8 adaptive samples of the same depth. In Figure 20(c) we have considered only transmitted indirect illumination, after giving transmission properties to the glass. It shows refraction through the glass, according to an index of refraction of 1.5 that is typical of such a material. In this case, a depth of 3 is used for our algorithm and a depth of 6 for the scene ray tracing, also with 8 adaptive samples. Figure 20(d) finally shows the combination of both reflection and transmission. The rendering times for this figure and Figure 20(b) are included in Table 1. As in Figure 18, the high rendering time is basically due to the ray tracing of the scene, in order to account for the reflection and transmission coming from the other surfaces. For the same images, if we only consider the inter-reflections and transmissions coming from the grooves on the same surface, the rendering time is 17 seconds.

Finally, in Figure 21 we have used our method to simulate a more complex scene, showing a house and its surroundings. The underlying mesh geometry only contains 88 polygons (see Figure 21(b)). Almost all the surfaces contain grooves. Such a scene entirely modeled with geometry could consist of $\approx$20,000 polygons or more. The rendered image includes reflections mainly on the swimming pool, which does not contains grooves, and on the windows. Its rendering time is just 81 seconds (see Table 1), 26 if only considering shadows and reflections between the grooves of the same surface, i.e. without ray tracing shadows or reflections between the objects. Figure 21(c) shows a closer view of the chimney made of bricks. These are simulated by means of grooves placed on the space between the different bricks and using the middle cross-section shown in Figure 21. Each groove uses a different BRDF for each of its facets. The lowest facet belonging to the mortar has a white color reflection, while the others use the same BRDF of the red surface. This image shows many intersecting ends including shadowing and masking effects. Figure 21(d) shows a closer view of a window, entirely modeled with grooves over a flat surface too. It is made of 5 vertical and 3 horizontal protruding grooves that use the lower cross-section of Figure 21, with all of their facets protruding from the surface. Each groove has a different associated cross-section scale, which is visible at the intersections. Occlusion effects as well as inter-reflections are included. Inter-reflections include the reflection of the grooves on the base surface, i.e. the windowframe onto the glass, as well as the reflection of the swimming pool and the floor, computed by ray tracing.

(a) (b) (c) (d)

Figure 21: (a) Complex scene fully modeled and rendered using our approach. (b) The mesh model only consist of 88 polygons. (c) Chimney bricks are simulated as mortar grooves penetrating into the base surface, using a different BRDF for the lower facet. (d) Windows are simulated using protruding grooves of different scales. The base surface (glass) includes inter-reflections due to the near grooves (the windowframe) and the objects of the scene (the swimming pool and the floor).

The previous scenes represent different examples of the kind of situations where our method could be applied. Grooves and similar features are very common in real world scenes, and many applications could profit from the approach presented here.

8 Conclusions and future work

In this paper, we have presented a general method to render grooved surfaces of all kinds. The method is fast and accurate, and accounts for smooth transitions between different scales, occlusion effects, and inter-reflections or transmissions on grooves. In contrast with other methods, ours has no restrictions on the distribution, size, or geometry of the grooves, thus it can simulate many kinds of surfaces and details, such as scratches, fractures, tiles, bricks, or ridges. It also serves for the simulation of micro-grooved surfaces, correctly simulating anisotropic reflections. Mainly, two different approaches have been presented in this work: a line sampling approach for isolated or parallel grooves, and a polygon or area sampling approach for special situations like intersections between grooves or groove ends. Both approaches are based on a single and compact representation of the grooves. This representation allows the easy modeling of these by means of paths and cross-sections, and has an important memory performance in front of surface detail modeled with geometry.

Finally, the good performance and the quality of the results have been demonstrated against an adaptive oversampling approach, commonly used on ray tracing algorithms. We have also shown many different examples of grooved surfaces modeled and rendered with our approach, which gives an idea of the many possible applications for this method. In architectural applications, for example, many surfaces present grooves or similar details, especially bricked or tiled walls or floors. Industrial applications are also often interested in the realistic simulation of scratches [BPMG04], and none of the existing methods accounts for general distributions of scratches, intersections, ends, or inter-reflections. Non-photo realistic rendering of grooved surfaces is another possible application too.

For the method proposed here, further research is still needed. First of all, it should be adapted for grooves with

highly curved paths. As stated before, occlusions may not be correctly simulated if the path is highly curved, and this may also happen for inter-reflections. Similarly, surface curvature has not been considered here. The base surface for the grooves is approximated by its local tangent plane, specified by the current surface normal. At grazing angles of vision or illumination, occlusions and inter-reflections between grooves should be modified according to the surface curvature. Silhouettes are not considered either, that is, grooves affecting the silhouette of the objects. This is usually observed on the boundaries of objects seen from close views. And last but not least, a transition towards geometry or BRDF is also of interest. Depending on the situation it could be convenient to use a different model than the one presented here. For close views of grooved surfaces, for example, if silhouettes want to be correctly simulated one could use a geometry model for the boundaries and our model for the rest, then performing smooth transitions. With our method, it should be easy to perform such smooth transitions towards other representations, because the exact geometric information is always preserved and the rendering is as accurate as using geometry or BRDFs, as demonstrated.

Acknowledgements

We would like to thank Frederic Pérez for his always very useful comments and suggestions. This project has been funded with grant numbers AP2001-1639 and TIN2004-07672-C03-01 from the MEC of the Spanish Government, and 2005SGR-00002 from the DURSI of the Catalan Government. Maya software has been used under agreement with Alias®.

References

[APS00] Michael Ashikhmin, Simon Premoze, and Peter S. Shirley. A microfacet-based BRDF generator. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 65–74, July 2000.

[AS00] Michael Ashikhmin and Peter S. Shirley. An anisotropic phong BRDF model. *Journal of Graphics Tools*, 5(2):25–32, 2000.

[Ban94] David C. Banks. Illumination in diverse codimensions. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, pages 327–334, July 1994.

[BB90] Welton Becket and Norman I. Badler. Imperfection for realistic image synthesis. *Journal of Visualization and Computer Animation*, 1(1):26–32, August 1990.

[BIT04] Pravin Bhat, Stephen Ingram, and Greg Turk. Geometric texture synthesis by example. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 41–44, 2004.

[BL99] John W. Buchanan and Paul Lalonde. An observational model for illuminating isolated scratches. In *Proceedings of the Western Computer Graphics Symposium 1999 (SKIGRAPH'99)*, March 1999.

[Bli78] James F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 78)*, volume 12, pages 286–292, August 1978.

[BM93] Barry G. Becker and Nelson L. Max. Smooth transitions between bump rendering algorithms. In *Proceedings of SIGGRAPH 93*, Computer Graphics Proceedings, Annual Conference Series, pages 183–190, August 1993.

[BMZB02] Henning Biermann, Ioana M. Martin, Denis Zorin, and Fausto Bernardini. Sharp features on multiresolution subdivision surfaces. *Graphical Models*, 64(2):61–77, 2002.

[BPMG04] Carles Bosch, Xavier Pueyo, Stéphane Mérillou, and Djamchid Ghazanfarpour. A physically-based model for rendering realistic scratches. *Computer Graphics Forum*, 23(3):361–370, September 2004.

[CGF04] Chiara Eva Catalano, F. Giannini, and B. Falcidieno. Introducing sweep features in modeling with subdivision surfaces. *Journal of WSCG*, 12(1):81–88, 2004.

[CMS87] Brian Cabral, Nelson Max, and Rebecca Springmeyer. Bidirectional reflection functions from surface bump maps. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, volume 21, pages 273–281, July 1987.

[Coo84] Robert L. Cook. Shade trees. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, volume 18, pages 223–231, July 1984.

[CPC84] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, volume 18, pages 137–145, July 1984.

[Fou92] Alain Fournier. Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination*, pages 45–52, May 1992.

[Gla89] Andrew S. Glassner. *An Introduction to Ray Tracing*. Academic Press Ltd., 1989.

[HDKS00] Wolfgang Heidrich, Katja Daubert, Jan Kautz, and Hans-Peter Seidel. Illuminating micro geometry based on precomputed visibility. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 455–464, July 2000.

[Hec86] Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics & Applications*, 6(11):56–67, November 1986.

[HH84] Paul S. Heckbert and Pat Hanrahan. Beam tracing polygonal objects. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, volume 18, pages 119–127, July 1984.

[JP00] Thouis Jones and Ronald Perry. Antialiasing with line samples. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 197–206, June 2000.

[Kaj85] James T. Kajiya. Anisotropic reflection models. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, volume 19, pages 15–21, July 1985.

[Kaj86] James T. Kajiya. The rendering equation. In *Computer Graphics (Proceedings of SIGGRAPH 86)*, volume 20, pages 143–150, August 1986.

[KHS01] Jan Kautz, Wolfgang Heidrich, and Hans-Peter Seidel. Real-time bump map synthesis. In *2001 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 109–114, August 2001.

[KS99] Andrei Khodakovsky and Peter Schröder. Fine level feature editing for subdivision surfaces. In *Proceedings of the 5th ACM Symposium on Solid Modeling and Applications*, pages 203–211, 1999.

[KS00] Jan Kautz and Hans-Peter Seidel. Towards interactive bump mapping with anisotropic shift-variant BRDFs. In *2000 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 51–58, August 2000.

[LP94] J. R. Logie and J. W. Patterson. Inverse displacement mapping in the general case. *Computer Graphics Forum*, 14(5):261–273, December 1994.

[Max88] Nelson L. Max. Horizon mapping: shadows for bump-mapped surfaces. *The Visual Computer*, 4(2):109–117, July 1988.

[MDG01] Stéphane Mérillou, Jean-Michel Dischler, and Djamchid Ghazanfarpour. Surface scratches: Measuring, modeling and rendering. *The Visual Computer*, 17(1):30–45, 2001.

[MOiT98] Shinji Mizuno, Minoru Okada, and Jun ichiro Toriwaki. Virtual sculpting and virtual woodcut printing. *The Visual Computer*, 14(2):39–51, 1998.

[OBM00] Manuel M. Oliveira, Gary Bishop, and David McAllister. Relief texture mapping. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 359–368, July 2000.

[PF90] Pierre Poulin and Alain Fournier. A model for anisotropic reflection. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, volume 24, pages 273–282, August 1990.

[PH96] Matt Pharr and Pat Hanrahan. Geometry caching for ray-tracing displacement maps. In *Eurographics Rendering Workshop 1996*, pages 31–40, June 1996.

[POC05] Fábio Policarpo, Manuel M. Oliveira, and João L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, pages 155–162, 2005.

[Sch97] Andreas Schilling. Toward real-time photorealistic rendering: Challenges and solutions. In *1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pages 7–16, August 1997.

[SSS00] Brian Smits, Peter S. Shirley, and Michael M. Stark. Direct ray tracing of displacement mapped triangles. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering*, pages 307–318, June 2000.

[Sta99] Jos Stam. Diffraction shaders. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 101–110, August 1999.

[War92] Gregory J. Ward. Measuring and modeling anisotropic reflection. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, volume 26, pages 265–272, July 1992.

[WAT92] Stephen H. Westin, James R. Arvo, and Kenneth E. Torrance. Predicting reflectance functions from complex surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, volume 26, pages 255–264, July 1992.

[WWT+03] Lifeng Wang, Xi Wang, Xin Tong, Stephen Lin, Shimin Hu, Baining Guo, and Heung-Yeung Shum. View-dependent displacement mapping. *ACM Transactions on Graphics*, 22(3):334–339, July 2003.