Efficient Cloth Modeling and Rendering

Katja Daubert[†], Hendrik P. A. Lensch[†], Wolfgang Heidrich[‡], and Hans-Peter Seidel[†]

[†)] Max-Planck-Institut für Informatik, [‡)] The University of British Columbia [1]

Abstract. Realistic modeling and high-performance rendering of cloth and clothing is a challenging problem. Often these materials are seen at distances where individual stitches and knits can be made out and need to be accounted for. Modeling of the geometry at this level of detail fails due to sheer complexity, while simple texture mapping techniques do not produce the desired quality.
In this paper, we describe an efficient and realistic approach that takes into account view-dependent effects such as small displacements causing occlusion and shadows, as well as illumination effects. The method is efficient in terms of memory consumption, and uses a combination of hardware and software rendering to achieve high performance. It is conceivable that future graphics hardware will be flexible enough for full hardware rendering of the proposed method.

1 Introduction

One of the challenges of modeling and rendering realistic cloth or clothing is that individual stitches or knits can often be resolved from normal viewing distances. Especially with coarsely woven or knitted fabric, the surface cannot be assumed to be flat, since occlusion and self-shadowing effects become significant at grazing angles. This rules out simple texture mapping schemes as well as bump mapping. Similarly, modeling all the geometric detail is prohibitive both in terms of the memory requirements and rendering time. On the other hand, it is probably possible to compose a complex fabric surface from copies of individual weaving or knitting patterns unless the viewer gets close enough to the fabric to notice the periodicity. This leads to approaches like virtual ray-tracing [5], which are more feasible in terms of memory consumption, but still result in long rendering times.

Fig. 1. Woolen sweater rendered using our approach (knit and perl loops).

In this paper we present a fast and memory-efficient method for modeling and rendering fabrics that is based on replicating weaving or knitting patterns. While the rendering part currently makes use of a combination of hardware and software rendering, it is conceivable that future graphics hardware will be flexible enough for full hardware rendering.

Our method assumes we have one or a small number of stitch types, which are repeated over the garment. Using a geometric model of a single stitch, we first compute

[1] Part of this work was done during a research visit of the first author to the University of British Columbia

the lighting (including indirect lighting and shadows) using the methods described in [3]. By sampling the stitch regularly within a plane we then generate a view dependent texture with per-pixel normals and material properties.

Before we cover the details of this representation in Section 3, we will briefly summarize related work in Section 2. We then describe acquisition and fitting of data from modeled micro-geometry in Sections 4, and 5. After discussing the rendering algorithm in Section 6 we finally present our results in Section 7.

2 Related Work

In order to efficiently render replicating patterns such as cloth without explicitly representing the geometry at the finest level, we can choose between several different representations. The first possibility is to compose global patterns of parts with precomputed illumination, such as light fields [13] and Lumigraphs [6]. However, these approaches assume fixed illumination conditions, and expanding them to arbitrary illumination yields an 8-dimensional function (which has been called the *reflectance field* [4]) that is too large to store for practical purposes.

Another possibility is to model the patterns as volumes [7, 14] or simple geometry (for example, height fields) with a spatially varying BRDF. Hardware accelerated methods for rendering shadowing and indirect illumination in height fields have been proposed recently [8, 16], as well as hardware algorithms for rendering arbitrary uniform [9, 10] and space-variant materials [11]. However, the combination of space-variant materials with bump- or displacement maps is well beyond the capabilities of current graphics hardware. This would require an excessive number of rendering passes which is neither practical in terms of performance nor in terms of numerical precision.

For high-performance rendering we therefore need to come up with more efficient representations that allow us to simulate view-dependent geometric effects (shadowing and occlusion) as well as illumination effects (specularity and interreflection) for space-variant materials in a way that is efficient both in terms of memory and rendering time.

In work parallel to ours, Xu et al. [18] developed the *lumislice*, which is a rendering method for textiles that is more tailored for high-quality, off-line rendering, whereas our method uses more precomputation to achieve near-interactive performance. In fact, the lumislice could be used as a way to precompute the data structures we use.

The method we propose is most closely related to bidirectional texture functions [2] and virtual ray-tracing [5]. As we will discuss below, our representation is, however, more compact and is easy to filter for correct anti-aliasing. Our approach is also related to image based rendering with controllable illumination, as described by Wong et al. [17]. Again, our representation is more compact, easier to filter and lends itself to partial use of graphics hardware. Future hardware is likely to have enough flexibility to eliminate the remaining software steps, making the method suitable for interactive applications.

3 Data Representation

Our representation of cloth detail is based on the composition of repeating patterns (individual weaves or knits) for which efficient data structures are used. In order to capture the variation of the optical properties across the material, we employ a spatially varying BRDF representation. The two spatial dimensions are point sampled into a 2D array. For each entry we store different parameters for a Lafortune reflection model [12], a

lookup table, as well as the normal and tangent.

An entry's BRDF $f_r(\vec{l}, \vec{v})$ for the light direction $\vec{l}$ and the viewing direction $\vec{v}$ is given by the following equation:

$$f_r(\vec{l}, \vec{v}) = T(\vec{v}) \cdot f_l(\vec{l}, \vec{v}), \tag{1}$$

where $f_l(\vec{l}, \vec{v})$ denotes the Lafortune model and $T(\vec{v})$ is the lookup table[2].

The Lafortune model itself consists of a diffuse part ρ and a sum of lobes[3]:

$$f_l(\vec{l}, \vec{v}) = l_z \cdot \left(\rho + \sum_i \left[(l'_x, l'_y, l'_z) \cdot \begin{pmatrix} C_{x_i} & 0 & 0 \\ 0 & C_{y_i} & 0 \\ 0 & 0 & C_{z_i} \end{pmatrix} \cdot \begin{pmatrix} v'_x \\ v'_y \\ v'_z \end{pmatrix} \right]^{N_i} \right) \tag{2}$$

Each lobe's shape and size is defined by its four parameters C_x, C_y, C_z, and N. Since f_l is wavelength dependent, we represent every parameter as a three-dimensional vector, one dimension per color channel. Before evaluating the lobe we transform the light and viewing direction into the local coordinate system given by the sampling point's average normal and tangent, yielding $\vec{l}'$ and $\vec{v}'$. In order to account for area foreshortening we multiply by l_z.

The lookup table $T(\vec{v})$ stores color and alpha values for each of the original viewing directions. It therefore closely resembles the directional part of a light field. Values for directions not stored in the lookup table are obtained by interpolation. Although general view-dependent reflection behavior including highlights etc., could be described by a simple Lafortune BRDF, we introduce the lookup table to take more complex properties like shadowing and masking (occlusion) into account that are caused by the complex geometry of the underlying cloth model.

Like in redistribution bump mapping [1], this approach aims at simulating the occlusion effects that occur in bump maps at grazing angles. In contrast to redistribution bump mapping, however, we only need to store a single color value per viewing direction, rather than a complete normal distribution. Figure 5 demonstrates the effect of the modulation with the lookup table. The same data, acquired from the stitch model shown in the middle, was used to fit a BRDF model without a lookup table, only consisting of several cosine lobes (displayed on the left cloth in Figure 5) and a model with an additional lookup table (cf. Figure 5 on the right). Both images were rendered using the same settings for light and viewing direction. Generally, without a lookup table, the BRDF tends to blur over the single knits. Also the BRDF without the lookup table clearly is not able to capture the color shifts to red at grazing angles, which are nicely visible on the right cloth.

The alpha value stored in the lookup table is used to evaluate the transparency. It is not considered in the multiplication with f_l, but used as described in Section 6 to determine if there is a hole in the model at a certain point for a given viewing direction. The alpha values are interpolated similarly to the color values.

4 Data Acquisition

After discussing the data structure we use for representing the detail of the fabrics, we now describe how to obtain the necessary data from a given 3D model.

[2]Both $T(\vec{v})$ and f_l are defined for each color channel, so $\cdot$ denotes the component-wise multiplication of the color channels.

[3]The operator a^N is defined to return zero if $a < 0$.

We model the base geometry of our knits and weaves using implicit surfaces, the skeletons of which are simple Bézier curves. By applying the Marching Cubes algorithm we generate triangle meshes, which are the input for our acquisition algorithm.

Now we can obtain the required data. As mentioned in Section 3, the spatial variations of the fabric pattern are stored as a 2D array of BRDF models. Apart from radiance samples $r(\vec{l}, \vec{v})$ for all combinations of viewing and light directions, we also need an average normal, an average tangent, and an alpha value for each viewing direction for each of these entries.

We use an extension of Heidrich et al.'s algorithm ([8]) to triangle meshes ([3]), which allows us to compute the direct and indirect illumination of a triangle mesh for a given viewing and light direction per vertex in hardware (for details see [3]). In order to account for masking and parts of the repeated geometry being visible through holes, we paste together multiple copies of the geometry.

Now we need to collect the radiance data for each sampling point. We obtain the 2D sampling locations by first defining a set of evenly spaced sampling points on the top face of the model's bounding box, as can be seen on the left in Figure 2. Then we project these points according to the current viewing direction (see Figure 2 in the middle) and collect the radiance samples from the surface visible through these 2D projections (see Figure 2 right), similarly to obtaining a light field.

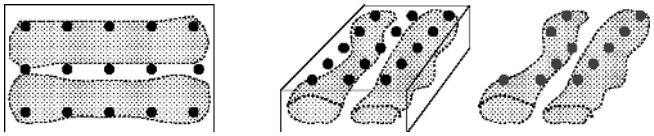

Fig. 2. Computing the sampling locations for the radiance values. Left: top view, middle: projection, right: resulting sampling locations, discarding samples at holes.

Note that, due to parallax effects, for each entry we combine radiance samples from a number of different points on the actual geometry. Like in [17], we will use this information from different surface points to fit a BRDF for the given sampling location.

```
for each v⃗ {
  ComputeSamplingPoints();
  RepeatScene(vertex color=normals);
  StoreNormals();
  StoreAlpha();
  for each l⃗ {
    ComputeLighting();
    RepeatScene(vertex color=lighting);
    StoreRadiance();
  }
}
AverageNormals();
```

Fig. 3. Pseudo code for the acquisition procedure.

As the stitch geometry can have holes, there might be no surface visible at a sampling point for a certain viewing direction. We store this information as a boolean transparency in the alpha channel for that sample. Multiple levels of transparency values can be obtained by super-sampling, i.e. considering the neighboring pixels.

In order to compute the normals, we display the scene once for each viewing direction with the normals coded as color values. An average normal is computed by adding the normals separately for each sampling point and averaging them at the end. We can construct a tangent from the normal and the bi-normal, which in turn we define as the vector perpendicular to both the normal and the x-axis. Figure 3 shows how the steps are put together in the acquisition algorithm.

5 Fitting Process

Once we have acquired all the necessary data, we use it to find an optimal set of parameters for the Lafortune model for each entry in the array of BRDFs. This fitting procedure can be divided into two major steps which are applied alternately. At first, the parameters of the lobes are fit. Then, in the second step, the entries of the lookup table are updated. Now the lobes are fit again and so on.

Given a set of all radiance samples and the corresponding viewing and light directions acquired for one sampling point, the fitting of the parameters of the Lafortune model f_l requires a non-linear optimization method. As proposed in [12], we applied the Levenberg-Marquardt algorithm [15] for this task.

The optimization is initiated with an average gray BRDF with a moderate specular highlight and slightly anisotropic lobes, e.g. $C_x = 1.22 * C_y$ for the first and $C_y = 1.22 * C_x$ for the second lobe if two lobes are fit. For the first fitting of the BRDF the lookup table $T(\vec{v})$ is ignored, i.e. all its entries are set to white.

After fitting the lobe parameters, we need to adapt the sampling point's lookup table $T(\vec{v})$. Each entry of the table is fit separately. This time only those radiance samples of the sampling point that correspond to the viewing direction of the current entry are considered. The optimal color for one entry minimizes the following set of equations:

$$\left(r(\vec{l_1},\vec{v}), r(\vec{l_2},\vec{v}), \ldots, r(\vec{l_R},\vec{v})\right)^T = T(\vec{v}) \left(f_l(\vec{l_1},\vec{v}), f_l(\vec{l_2},\vec{v}), \ldots, f_l(\vec{l_R},\vec{v})\right)^T \quad (3)$$

where $r(\vec{l_1},\vec{v}), \ldots, r(\vec{l_R},\vec{v})$ are the radiance samples of the sampling point with the common viewing direction $\vec{v}$ and the distinct light directions $\vec{l_1}, \ldots, \vec{l_R}$. The currently estimated lobes are evaluated for every light direction yielding $f_l(\vec{l_i},\vec{v})$. Treating the color channels separately, Equation 3 can be rewritten by replacing the column vector on its left side by $\vec{r}(\vec{v})$, the vector on its right side by $\vec{f}(\vec{v})$, yielding $\vec{r}(\vec{v}) = T(\vec{v}) \cdot \vec{f}(\vec{v})$. The least squares solution to this equation is given by

$$T(\vec{v}) = \frac{\langle \vec{f}(\vec{v}) | \vec{r}(\vec{v}) \rangle}{\langle \vec{f}(\vec{v}) | \vec{f}(\vec{v}) \rangle} \quad (4)$$

where $\langle \cdot | \cdot \rangle$ denotes the dot product. This is done separately for every color channel and easily extends to additional spectral components.

To further improve the result we alternately repeat the steps of fitting the lobes and fitting the lookup table. The iteration stops as soon as the average difference of the previous lookup table's entries to the new lookup table's entries is below a certain threshold.

In addition to the color, each entry in the lookup table also contains an alpha value indicating the opacity of the sample point. This value is fixed for every viewing direction and is not affected by the fitting process. Instead it is determined through ray-casting during the data acquisition phase.

Currently, we also derive the normal and tangent at each sample point directly from the geometric model. However, the result of the fitting process could probably be further improved by also computing a new normal and tangent to best fit the input data.

5.1 Mip-Map Fitting

The same fitting we have done for every single sample point can also be performed for groups of sample points. Let a sample point be a texel in a texture. Collecting all

radiance samples for four neighboring sample points, averaging the normals, fitting the lobes and the entries of the lookup table then yields the BRDF corresponding to a texel on the next higher mip-map level.

By grouping even more sample points, further mip-map levels can be generated. The overall effort per level stays the same since the same number of radiance samples are involved at each level.

6 Rendering

After the fitting process has been completed for all sampling points we are ready to apply our representation of fabric patterns to a geometric model. We assume the given model has per vertex normals and valid texture coordinates $\in [0..t_N]^2$, where t_N is the number of times the pattern is to be repeated across the whole cloth geometry. Furthermore, we assume the fabric patterns are stored in a 2D array, the dimensions of which correspond to the pattern's spatial resolution $(\text{res}_x, \text{res}_y)$. Our rendering algorithm then consists of four steps:

1. Interpolate per pixel normals
2. Compute indices into the pattern array, yielding a BRDF f_r
3. Evaluate f_r with light and view mapped into geometry's local coordinate system
4. Write result to framebuffer

The goal of Step 1 is to estimate a normal for each visible point on the object. We do this by color coding the normals at the vertices and rendering the scene using Gouraud shading. Each framebuffer value with an alpha value $\neq 0$ now codes a normal.

The next step is to find out which BRDF we need to evaluate in order to obtain the color for each pixel. In order to do this we first generate a texture with the resolution $(\text{res}_x, \text{res}_y)$ in which the red and green channel of each pixel encode its position. Note that this texture has to be generated only once and can be reused for other views and light directions. Using hardware texture mapping with the above mentioned texture coordinates, the texture is replicated t_N times across the object. Now the red and green channel of each pixel in the framebuffer holds the correct indices into the 2D array of BRDFs for this specific fabric pattern.

Once we know which BRDF to evaluate, we map the light and viewing direction into the geometry's local coordinate system, using the normals obtained in Step 1 and a tangent constructed as described in Section 4. Note that two mappings need to take place: this one, which maps the world view and light to the cloth geometry's local coordinate system (yielding $\vec{l}$ and $\vec{v}$), and another when evaluating the BRDF, which transforms these values to the pattern's local coordinate system (yielding $\vec{l}'$, $\vec{v}'$).

The software evaluation of the BRDF model (see Section 3) returns three colors and an alpha value from the lookup table, which we then write to the framebuffer.

The presented rendering technique utilizes hardware as far as possible. However, the BRDF model is still evaluated in software, although mapping this onto hardware should be feasible with the next generation of graphics cards.

6.1 Mip-Mapping

As described in Section 5.1, we can generate several mip-map levels of BRDFs. We will now explain how to enhance the above algorithm to correctly use different mip-map levels, thereby exploiting OpenGL mip-mapping.

First we modify Step 2 and now generate one texture per mip-map level. Each texture's resolution corresponds to the BRDF's spatial resolution at this level. As before, the red and green channel code the pixel's location in the texture. Additionally, we now use each pixel's blue channel to code the mip-map level of the corresponding texture. For example, if we have 6 levels, all pixel's blue values are 0 in texture 0, 0.2 in texture 1, 0.4 in texture 2 and so on.

If we set up OpenGL mip-mapping with these textures specified for the correct levels, the blue channel of each pixel will tell us which texture to use, while the red and green channel still code the indices into the array of BRDFs at this level.

Blending between two mip-map levels is also possible. As we do not want to blend the texture coordinates in the red and green channels, however, we need two passes to do so. The first pass is the same as before. However, in the second pass we setup the mip-map technique to linearly interpolate between two levels. We avoid overwriting the values in the red and green channels by using a color mask. Now the value of the blue channel v_b codes between which levels to blend (in the above example between levels $l_{\min} = \lfloor v_b/0.2 \rfloor$ and $l_{\max} = \lceil v_b/0.2 \rceil$) and also tells us the blending factor (here $(v_b - l_{\min} \cdot 0.2)/0.2$).

7 Results and Applications

We implemented our algorithms on a PC with an AMD Athlon 1GHz processor and a GeForce 2 GTS graphics card. To generate the images in this paper we applied the acquired fabric patterns to cloth models we modeled with the 3D Studio Max plug-ins Garment Maker and Stitch. Our geometric models for the knit or weave patterns consist of 1300–23000 vertices and 2400–31000 triangles. The computation times of the acquisition process depend on the number of triangles, as well as the sampling density for the viewing and light directions, but generally vary from 15 minutes to about 45 minutes. We typically used 32×32 or 64×64 viewing and light directions, uniformly distributed over the hemisphere, generating up to 4096 radiance samples per sampling point on the lowest level. We found a spatial resolution of 32×32 samples to be sufficient for our detail geometry, which results in 6 mip-map levels and 1365 BRDF entries. The parameter fitting of a BRDF array of this size takes about 2.5 hours. In our implementation each BRDF in the array (including all the mip-map levels) has the same number of lobes. We found out that generally one or two lobes are sufficient to yield visually pleasing results. The threshold mentioned in Section 5 was set to 0.1 and we noted that convergence was usually achieved after 2 iterations. Once all parameters have been fit we need only 4 MB to store the complete data structure for one type of fabric, including all mip-map levels and the lookup tables with 64 entries per point.

The rendering times e.g. for Figure 1 are about 1 frame per second for a resolution of 730×400 pixels. The bulk of this time is spent on reading back the framebuffer contents in order to evaluate the BRDF for every pixel. We therefore expect that with the advent of more flexible hardware, which will allow us to implement the rendering part of this paper without such a software component, the proposed method will become feasible for interactive applications.

The dress in Figure 4(a) displays a fabric pattern computed with our method. In Figure 4(b) we compare the results of a mip-mapped BRDF to a single level one. As expected the mip-mapping nicely gets rid of the severe aliasing clearly visible in the not mip-mapped left half of the table. Figure 5 illustrates how even complex BRDFs with color shifts can be captured using our model. Figure 1 and Figure 6 show different fabric patterns displayed on the same cloth geometry.

8 Conclusions

In this paper we have presented a memory-efficient representation for modeling and rendering fabrics that is based on replicating individual weaving or knitting patterns. We have demonstrated how our representation can be generated by fitting samples from a global illumination simulation to it. In a similar fashion it should be possible to acquire a fitted representation from measured image data. Our model is capable of capturing color variations due to self-shadowing and self-occlusion as well as transparency. In addition, it naturally lends itself to mip-mapping, thereby solving the filtering problem.

Furthermore we presented an efficient rendering algorithm which can be used to apply our model to any geometry, achieving near-interactive frame rates with a combination of hardware and software rendering. With the increasing flexibility of upcoming generations of graphics boards, we expect to be able to implement the rendering algorithm completely in hardware soon. This would make the approach suitable for fully interactive and even real time applications.

References

1. B. Becker and N. Max. Smooth Transitions between Bump Rendering Algorithms. In *SIGGRAPH '93 Proceedings*, pages 183–190, August 1993.
2. K. Dana, B. van Ginneken, S. Nayar, and J. Koenderink. Reflectance and Texture of Real World Surfaces. *ACM Transactions on Graphics*, 18(1):1–34, January 1999.
3. K. Daubert, W. Heidrich, J. Kautz, J.-M. Dischler, and Hans-Peter Seidel. Efficient Light Transport Using Precomputed Visibility. Technical Report MPI-I-2001-4-003, Max-Planck-Institut für Informatik, 2001.
4. P. Debevec, T. Hawkins, C. Tchou, H.-P. Duiker, W. Sarokin, and M. Sagar. Acquiring the reflectance field of a human face. In *SIGGRAPH 2000 Proceedings*, pages 145–156, July 2000.
5. J.-M. Dischler. Efficiently Rendering Macro Geometric Surface Structures with Bi-Directional Texture Functions. In *Proc. of Eurographics Workshop on Rendering*, pages 169–180, June 1998.
6. S. Gortler, R. Grzeszczuk, R. Szelinski, and M. Cohen. The Lumigraph. In *SIGGRAPH '96 Proceedings*, pages 43–54, August 1996.
7. E. Gröller, R. Rau, and W. Straßer. Modeling textiles as three dimensional textures. In *Proc. of Eurographics Workshop on Rendering*, pages 205–214, June 1996.
8. W. Heidrich, K. Daubert, J. Kautz, and H.-P. Seidel. Illuminating Micro Geometry Based on Precomputed Visibility. In *SIGGRAPH '00 Proceedings*, pages 455–464, July 2000.
9. W. Heidrich and H.-P. Seidel. Realistic, Hardware-accelerated Shading and Lighting. In *SIGGRAPH '99 Proceedings*, August 1999.
10. J. Kautz and M. McCool. Interactive Rendering with Arbitrary BRDFs using Separable Approximations. In *Proc. of Eurographics Workshop on Rendering*, pages 247 – 260, June 1999.
11. J. Kautz and H.-P. Seidel. Towards interactive bump mapping with anisotropic shift-variant BRDFs. In *2000 Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 51–58, August 2000.
12. E. Lafortune, S. Foo, K. Torrance, and D. Greenberg. Non-Linear Approximation of Reflectance Functions. In *SIGGRAPH '97 Proceedings*, pages 117–126, August 1997.
13. M. Levoy and P. Hanrahan. Light Field Rendering. In *SIGGRAPH '96 Proceedings*, pages 31–42, August 1996.
14. F. Neyret. Modeling, Animating, and Rendering Complex Scenes Using Volumetric Textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1), January – March 1998.
15. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, 1992. ISBN 0-521-43108-5.
16. P. Sloan and M. Cohen. Hardware Accelerated Horizon Mapping. In *Proc. of Eurographics Workshop on Rendering*, pages 281–286, June 2000.
17. Tien-Tsin Wong, Pheng-Ann Heng, Siu-Hang Or, and Wai-Yin Ng. Image-based Rendering with Controllable Illumination. In *Proc. of Eurographics Workshop on Rendering*, pages 13–22, 1997.
18. Y.-Q. Xu, Y. Chen, S. Lin, H. Zhong, E. Wu, B. Guo, and H.-Y. Shum. Photo-realistic rendering of knitwear using the lumislice. In *Computer Graphics (SIGGRAPH '01 Proceedings)*, 2001. to be published.

(a) (b)

Fig. 4. (a) A dress rendered with BRDFs consisting of only one lobe. (b) Left: Aliasing artifacts are clearly visible if no mip-mapping is used. Right: using several mip-mapping layers.

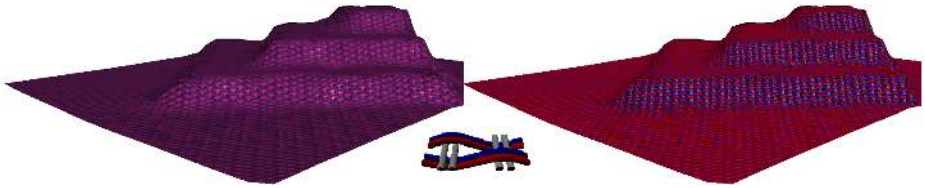

Fig. 5. The fabric patterns displayed on the models (left and right) were both computed from the micro geometry in the middle. In contrast to the right BRDF model, the left one does not include a lookup table. Clearly this BRDF is not able to capture the color shift to red for grazing angles, nicely displayed on the right.

Fig. 6. Different fabric patterns on the same model. Left: plain knit, middle: loops with different colors, right: perl loops.