# BTF Rendering in Blender

Martin Hatka* Michal Haindl†
Institute of Information Theory and Automation of the ASCR
Prague, Czech Republic

## Abstract

Bidirectional texture function (BTF) is 7D function of planar coordinates, spectral coordinate, and viewing and illumination angles, respectively. BTF is the recent most advanced representation of visual properties of surface materials. Unlike smooth textures, it specifies their altering appearance due to varying illumination and viewing conditions. This BTF visual appearance dependency on viewing and illumination conditions significantly complicates not only its acquisition, representation, and modeling but also makes its rendering noticeably more demanding. BTF textures are acquired by costly measurements of real materials and their subsequent non-trivial processing. While several techniques for measurement or processing of BTF textures have been described already, there is no environment allowing to support BTF texture rendering. This contribution describes novel Blender texture plugin for the purpose of BTF texture mapping and rendering. The plugin benefits from our previously developed BTF Roller texture enlargement method which is integral part of its implementation. The presented plugin allows to create realistic computer animations with additional BTF textures of any required size mapped onto an object surfaces while the other functionality of Blender retains.

**CR Categories:** I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Radiosity;

**Keywords:** rendering, bidirectional texture function, Blender

## 1 Introduction

There is neither professional nor open source 3D graphics application currently available which enables BTF texture [Dana et al. 1997; Filip and Haindl 2009] rendering. However an ever-growing number of real world computer vision applications require realistic rendering of genuine materials which cannot be achieved without this recently most advanced surface material representation. A simple alternative would be to write a proprietary BTF shader, however to develop essential 3D graphics environment for BTF rendering would be very difficult and resources demanding probably even all options offered by contemporary 3D graphics applications could not be achieved. Thus an existing graphical application suitable for BTF texture rendering enhancement is the appropriate solution. As it turned out the best choice is the 3D graphics application Blender (see Fig. 1). The Blenderis an open source software which is being actively developed under the supervision of the Blender Foundation and the source codes written in C++ are freely available.

*e-mail: hatka@utia.cas.cz
†e-mail: haindl@utia.cas.cz


VRCAI 2011, Hong Kong, China, December 11 – 12, 2011.
© 2011 ACM 978-1-4503-1060-4/11/0012 $10.00

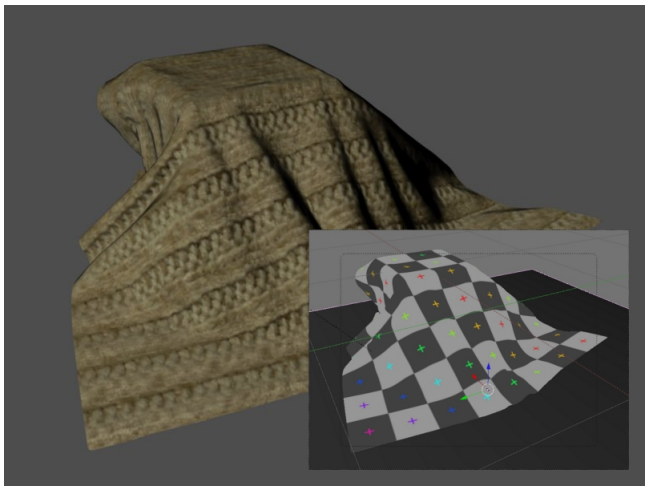

**Figure 1:** *Drapery 3D model created and textured using UV-mapping in Blender can be easily coated with BTF texture thanks to our BTF texture plugin.*

### 1.1 Bidirectional Texture Function

Multispectral BTF is a seven-dimensional function, which considers measurement dependency on color spectrum, planar material position, as well as its dependence on illumination and viewing angles:

$$BTF(r, \theta_i, \phi_i, \theta_v, \phi_v) \quad (1)$$

where the multiindex $r = [r_1, r_2, r_3]$ specifies planar horizontal and vertical position in material sample image, $r_3$ is the spectral index and $\theta, \phi$ are elevation and azimuthal angles of the illumination and view direction vector (see Fig. 2). The BTF measurements comprise a whole hemisphere of light and camera positions in observed material sample coordinates according to selected quantization steps (see Fig. 3). A fast BTF synthesis method with substantial compression is essential for applications requiring accurate realtime rendering of these data using graphics hardware. In addition, the original BTF measurements only cannot be used in any practical application due to missing necessary measurements from all arbitrary vantage points under arbitrary illumination and due to their small size. Thus, a seamless spatial enlargement (modeling) method of this otherwise huge BTF data is inevitable and also constitutes an integral part of our BTF plugin.

### 1.2 BTF Visualization

Applying BTF textures to a 3D model surfaces dramatically enhances the visual appearance of the objects in rendered scene. Such texturing is the best and physically correct way to achieve photo-realistic results. Accurate texture mapping is essential to get a high quality visualization. Suitable for BTF texturing is UV-mapping technique which projects a texture map onto a 3D object while the texture map is handled manually. If the accurate UV-mapping of the texture is done, the BTF application to the surface is straightforward. UV texture coordinates unambiguously define the posi-

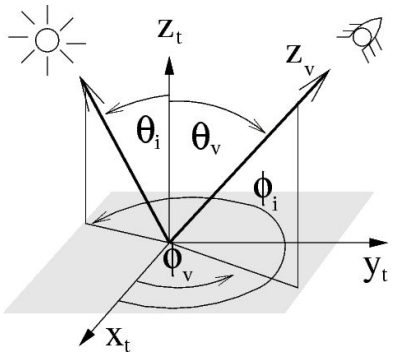


**Figure 2:** *Relationship between illumination and viewing angles within sample coordinate system.*

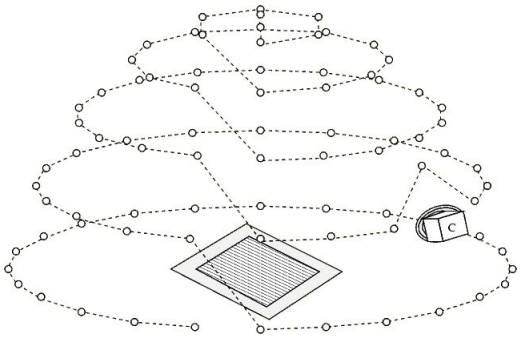

**Figure 3:** *An example of light trajectory above the sample during measurement while camera is fixed.*

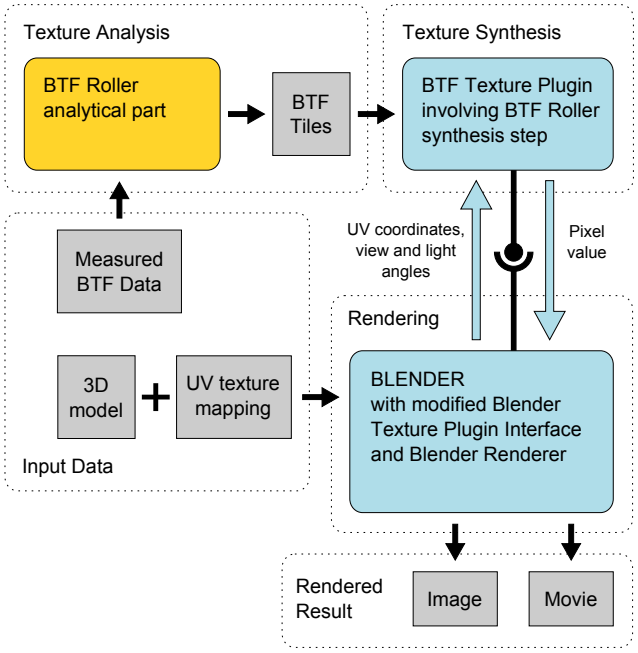


**Figure 4:** *BTF rendering using Blender, texture plugin, and the BTF Roller texture synthesis algorithm. Texture analysis using the BTF Roller (yellow box) can be done independently before the rendering. Texture synthesis implemented in the texture plugin is performed as inseparable part of the rendering process (red box). BTF tiles are generated and stored and they are subsequently reused during the rendering.*

tion, the orientation, and the scale of the texture on the surface of an object.

### 1.3 Blender

Blender[1] is the free open source 3D graphics application for creation 3D models, visualizations and animations. Blender is available for all major operating system under the GNU General Public License and it is being actively developed. The purpose of Blender is to model and render 3D computer graphics and animations using various techniques such as raytracing, radiosity, ambient occlusion or scan-line rendering. Modeling techniques are primarily aimed at facet representation of the objects. However, Bezier curves or NURBS surfaces are supported as well. Animation capabilities incorporate key-framed animation tools including inverse kinematics, armature, curve and lattice-based deformations, fluid dynamics, and a particle system with collision detection.

Blender, as is, does not handle the dependency of the texture appearance on the lighting conditions. On the other hand, Blender provides an interface for texture plugins. Texture plugin is a dynamically loaded library that exist as a separate file on a computer. When called in it communicates with Blender through given interface to generate the texture.

The rest of the paper is organized as follows. Section 2 describes a modification of the Blender's renderer to be able to calculate viewing and illumination angles for texturing purposes and the extension of Blender's texture plugin interface. Section 3 provides an insight into BTF texture plugin architecture. Section 4 contains results where the BTF measurements were used in comparison with the alternative established approach which uses only smooth planar textures. Last section 5 concludes the paper. The overall scheme of our proposed BTF texture rendering solution using Blender is provided in the Fig. 4.

[1] http://www.blender.org/

## 2 Blender Modification

In the Blender's rendering pipeline there is no dependency of the texture appearance on the viewing and illumination conditions considered. Although several types of diffuse and specular shaders are implemented and these shaders use the surface normal, viewing and illumination directions, the texturing is performed earlier than the shading. Therefore these shaders cannot be used for the BTF texturing. This is the reason why the BTF texturing should by solved in a different way. The solution is to involve the capability to vary the texture appearance on the illumination and viewing conditions directly in the texturing process.

Due to the huge amount of typical BTF data, a direct BTF support in Blender seems to be a very complex, difficult and ineffective task. On the other hand, a texture plugin is an interesting and much simpler way to incorporate BTF textures to Blender. Texture plugin communicates with Blender through the standardized interface and texture data can be treated outside Blender. The plugin interface has to be able to handle the dependency of the texture on the viewing and illumination conditions.

The first task is to extend the Blender's renderer to compute the viewing and illumination azimuthal and elevation angles and the subsequent task is to modify the plugin interface to be able to pass these angles to the texture plugin.

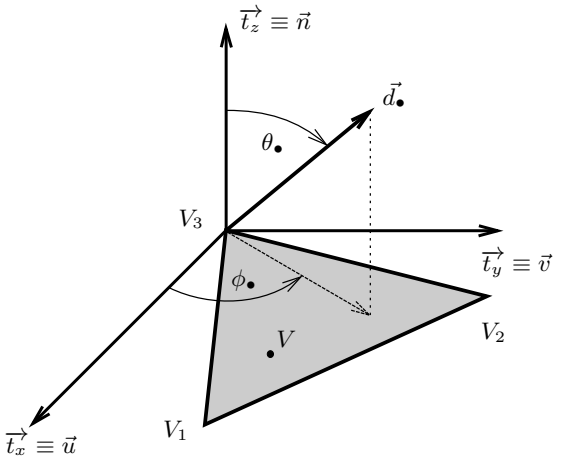


**Figure 5:** *Computation of the azimuthal and elevation angle for the view and illumination direction is based on a transform of view vector or illumination vector to the texture coordinate system.*

### 2.1 Rendering Pipeline Viewing and Illumination Angle Computation

The only necessary modification of the renderer was to incorporate an evaluation of elevation and azimuthal angles of the illumination and view direction. Principle of the modification is the following.

Each rendered pixel belongs to a triangular facet. In view of the fact that the spatial coordinates and UV texture coordinates of the triangle vertices are known, the UV texture coordinates of the rendered pixel can be evaluated. Moreover, view and illumination vectors are known. Projection of the view and illumination vector on the axes of the texture space provides the coordinates of the view and illumination vector in the texture space. Finally, the elevation and azimuthal angles of the view and illumination vector are calculated.

Let's consider any rendered 3D object which consists of an arbitrary number of triangular facets. The scene is rendered in pixel by pixel order and each rendered pixel belongs to a certain facet of the rendered object. Further, let's consider the rendered pixel $V$ which belongs to the facet $\triangle V_1V_2V_3$ (see Fig. 5). The spatial coordinates of the vertices $V_1$, $V_2$, $V_3$ of the facet $\triangle V_1V_2V_3$ in the orthonormal basis $\mathcal{S} = (\vec{x}, \vec{y}, \vec{z})$ are $(V_1)_\mathcal{S} = (v_1^x, v_1^y, v_1^z)$, $(V_2)_\mathcal{S} = (v_2^x, v_2^y, v_2^z)$, $(V_3)_\mathcal{S} = (v_3^x, v_3^y, v_3^z)$, respectively.

Then the texture mapping function $T_{UV}$, $T_{UV}(v_\bullet^x, v_\bullet^y, v_\bullet^z) = (v_\bullet^u, v_\bullet^v)$, assigns to the vertices $V_1$, $V_2$, $V_3$ corresponding texture coordinates $(V_1)_\mathcal{T} = (v_1^u, v_1^v, 0)$, $(V_2)_\mathcal{T} = (v_2^u, v_2^v, 0)$, $(V_3)_\mathcal{T} = (v_3^u, v_3^v, 0)$, where $\mathcal{T} = (\vec{u}, \vec{v}, \vec{w})$ is the orthonormal basis of the 3D texture coordinates. Note that the triangular facet $\triangle V_1V_2V_3$ is coplanar with the plane given by the texture space axes $\vec{u}$ and $\vec{v}$ and its normal $\vec{n}$ is parallel with the axis $\vec{w}$ of the space of the texture coordinates. Because the relationship between the spatial coordinates and the texture coordinates is known (the UV-mapping is done in a 3D model), the axes of the texture space can be expressed in the spatial coordinates, i.e. $\vec{t_x} = (\vec{u})_\mathcal{S}$, $\vec{t_y} = (\vec{v})_\mathcal{S}$ and $\vec{t_z} = (\vec{w})_\mathcal{S}$.

As the next step, let's denote the view and illumination directions expressed in spatial coordinates as $\vec{d_v}$ and $\vec{d_l}$, respectively. Then the projection of a vector $\vec{d_\bullet}$ in the directions of the axes $\vec{t_x}, \vec{t_y}$ and $\vec{t_z}$ of the texture coordinates expressed in the spatial coordinates yields in a vector $\vec{s_\bullet}$. Vector $\vec{s_\bullet}$ corresponds to the vector $\vec{d_\bullet}$ expressed in the texture coordinates, i.e. $\vec{s_\bullet} = (s_\bullet^u, s_\bullet^v, s_\bullet^w) = (\vec{d_\bullet})_\mathcal{T}$. Finally, while the vector $\vec{s_\bullet} = (\vec{d_\bullet})_\mathcal{T}$ expressed in the texture coordinates is known, required azimuthal angle $\phi_\bullet$ and elevation angle $\theta_\bullet$ can be easily calculated from the relationship between the cartesian and spherical coordinates:

$$\cos\theta = s^w \ , \tag{2}$$

$$\sin\phi = \frac{s^v}{\sqrt{(s^u)^2 + (s^v)^2}} \ , \tag{3}$$

$$\cos\phi = \frac{s^u}{\sqrt{(s^u)^2 + (s^v)^2}} \ . \tag{4}$$

### 2.2 BTF Texture Plugin Interface

Texture plugin takes 3D texture coordinate vector $(u, v, w)$ as an input and returns back the vector $(y, Y_R, Y_G, Y_B, N_u, N_v, N_w)$ as an output, where $y$ is the intensity of the pixel (in the case of monochromatic texture), $Y_R$, $Y_G$, $Y_B$ are the RGB components and $N_u$, $N_v$, $N_w$ are the normal vector components in texture coordinates.

As described in section 1.1, BTF is 7D function of planar coordinates, spectral coordinate, and viewing and illumination angles, $BTF(x, y, r, \theta_i, \phi_i, \theta_v, \phi_v)$. Input interface was extended to pass the viewing and illumination angles hence input vector $(u, v, w)$ was substituted by $(u, v, w, \theta_i, \phi_i, \theta_v, \phi_v)$, where $w$ is not used (only planar textures are considered). To set the texture coordinates $u$ and $v$, UV-mapping technique, which is considered as the most accurate, was used.

## 3 BTF Texture Plugin

BTF texture measurement consists of thousands of colour images (section 1.1) and to incorporate them directly in Blender seems difficult. For that reason it proved to be very effective to take advantage of texture plugin. Proposed BTF texture plugin communicates with Blender through Blender texture plugin interface additionally extended of viewing and illumination azimuthal and elevation angles.

Main advantage of this approach is a possibility to implement various texture synthesis algorithms directly in the plugin, particularly the BTF Roller algorithm (section 3.3). Plugin performance optimization is then straightforward.

### 3.1 Barycentric Coordinates Interpolation

BTF textures are measured for finite number of camera and light source positions, however in practice it is necessary to evaluate pixel values also for other unmeasured combinations of camera and light source positions. For such a combination interpolation, which is motivated by spherical barycentric coordinates [Polthier et al. 2006], on a triangle with known vertex pixel values is performed. Interpolation using spherical barycentric coordinates would be the most accurate but computationally complex. The following approximation seems to be sufficient.

Let's consider the hemisphere (see Fig. 6) with its center $O$ and the point $P$ on the hemisphere corresponding to desired azimuthal and elevation angle of the view or illumination direction. Let's $Y_P$ denotes the value of the desired pixel viewed or illuminated under the direction corresponding to the point $P$ on the hemisphere. Further, denote the three known measured directions, which are closest to the $P$, as $P_1$, $P_2$ and $P_3$ and the values of corresponding pixel as $Y_{P_1}$, $Y_{P_2}$ and $Y_{P_3}$. Then the value of the pixel $Y_P$ will be $Y_P = w_1Y_{P_1} + w_2Y_{P_2} + w_3Y_{P_3}$, where $w_1$, $w_2$ and $w_3$ are the weights of $Y_{P_1}$, $Y_{P_2}$, and $Y_{P_3}$, $w_1 + w_2 + w_3 = 1$.

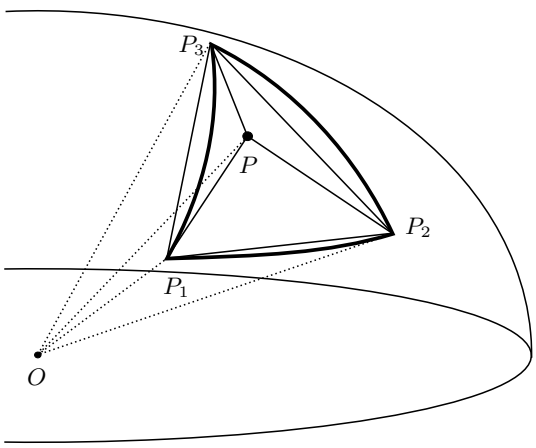


**Figure 6:** *Interpolation of the pixel $P$ value is motivated by barycentric coordinates. The value of the $P$ is a weighted sum of the $P_1$, $P_2$ and $P_3$ while the weight of each pixel corresponds to the volume of the opposite quadrilateral.*

The weights are defined in the following way:

$$w_1 = \frac{V_1}{V}, \quad w_2 = \frac{V_2}{V}, \quad w_3 = \frac{V_3}{V}, \quad V = V_1 + V_2 + V_3,$$

where $V_1$ denotes the volume of tetrahedron $PP_2P_3O$, $V_2$ the volume of $PP_3P_1O$ and $V_3$ the volume of $PP_1P_2O$. Moreover, if $O = (0, 0, 0)$, then

$$\begin{aligned} V_1 &= \frac{1}{6} \left|\det(P, P_2, P_3)\right| \ , \\ V_2 &= \frac{1}{6} \left|\det(P, P_3, P_1)\right| \ , \\ V_3 &= \frac{1}{6} \left|\det(P, P_1, P_2)\right| \ . \end{aligned}$$

### 3.2 BTF Data Buffer

To store whole BTF texture dataset in memory is ineffective. The best solution is to implement buffer to store BTF texture data. The size of the buffer is parameter specified by user. Appropriate software design of the buffer is essential to the plugin performance and the right optimization can dramatically speed up the BTF rendering process.

The buffer is designed as a double linked list (see Fig. 7). This list holds BTF texture image data, each node for the BTF slice corresponding to a particular texture measurement. Moreover, the list is extended by an array of pointers to the list nodes. This optimization allows efficient check if the BTF slice is loaded in the buffer and also fast search in the list. The BTF slices are kept ordered by the last access time and if the memory dedicated to the buffer is exhausted, the first accessed slice is removed from the list.

### 3.3 Texture Synthesis

BTF texture samples has only limited size, typically several hundreds or thousands of pixels. The limited size of the texture measurement is the fundamental problem while an object of a 3D scene should by covered with the BTF texture. The simplest way to overcome this problem is to tile the texture sample. On the other hand, the tiled texture has remarkable seams which dramatically decrease the quality of the resulting texture.

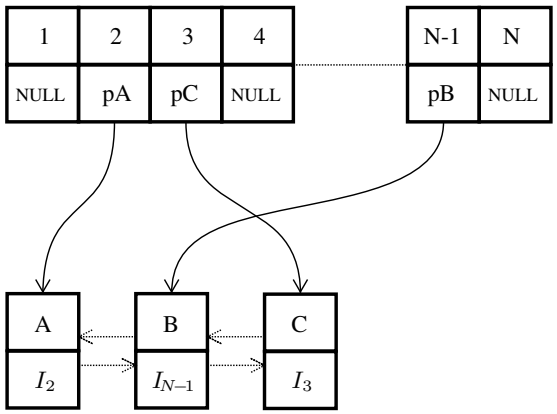


**Figure 7:** *Buffer with loaded texture data. The buffer consists of double linked list containing image data and an array with pointers to the list nodes. The array is used for fast search of the indexed images.*

In order to generate top quality BTF texture without visible seams the synthesis step of the texture enlargement sampling type of algorithm, which is called the BTF Roller [Haindl and Hatka 2005], was incorporated in the plugin (see Fig. 8). The roller method was chosen from possible alternatives ([Efros and Freeman 2001; Tong et al. 2002; Leung et al. 2007] and many others) because it is fully automatic and very fast method which produces high quality spatial data enlargement results.

The roller method is based on the overlapping tiling and subsequent minimum error boundary cut. One or several optimal double toroidal data patches are seamlessly repeated during the synthesis step. This fully automatic method starts with the minimal tile size detection which is limited by the size of control field, the number of toroidal tiles we are looking for and the sample spatial frequency content. Then the plugin input is the set of several rectangular and mutually interchangeable BTF texture tiles. These tiles are precomputed only one times during the analytical step of the BTF Roller algorithm. Because the resulting texture generated by the plugin has an arbitrary size and is randomly accessed by the Blender, it is inefficient to generate whole BTF slices as random tiling. More efficient approach is to generate aperiodic tiling [Cohen et al. 2003]. In our BTF texture plugin we have used the iterative version [Stam 1997] of the Wang Tiles. Then it could be simply decided which tile and its pixel will be used while the value of the particular pixel from the resulting texture is required by Blender. Alternatively the synthetic BTF textures can be generated from mathematical models [Haindl 1991; Haindl and Havlíček 1998; Bennett and Khotanzad 1998; Haindl and Havlíček 2000; Haindl and Havlíček 2002; Haindl et al. 2004; Haindl and Filip 2004] which are more flexible and extremely compressed, because only several parameters have to be stored. However, mathematical models can only approximate real BTF measurements, which results in visual quality compromise for some oversimplified methods.

The main advantage of the solution based on the texture plugin is the possibility to implement various BTF texture synthesis methods or various BTF texture models and verify them in commonly used 3D graphics application.

### 3.4 Plugin Parameters

Blender texture plugin interface provides a capability to define the plugin control panel. BTF texture plugin control panel (see Fig. 9) consists of several control elements. The user must specify a path

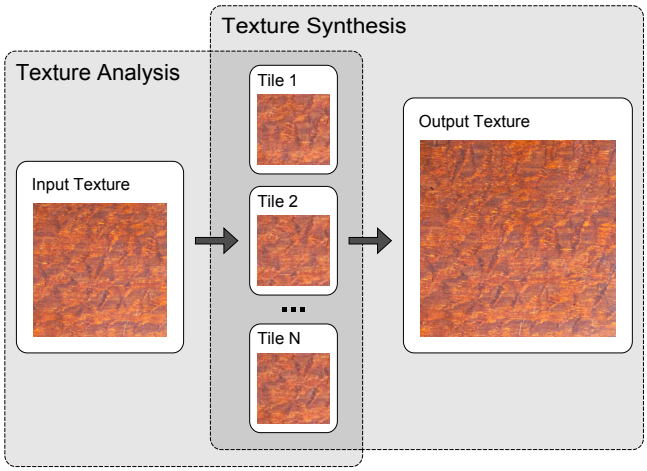


**Figure 8:** *The principle of the BTF Roller algorithm. During the analytical part, several mutually interchangeable tiles (middle) are extracted from the input texture (left). During the extremely fast synthesis step, synthetic texture (right) from the set of extracted tiles (middle) is generated by random or aperiodic tiling.*

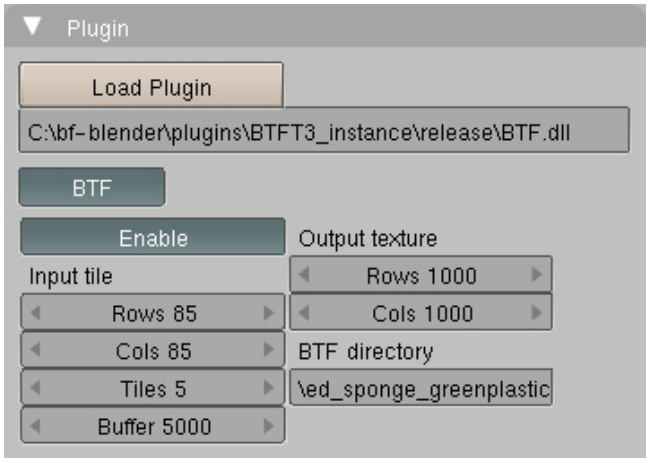


**Figure 9:** *BTF plugin control panel in Blender. User can specify number of input tiles, size of the buffer and the size of the output to control the quality of the resulting texture and performance of the plugin.*

to the BTF texture data. Then the user must specify number of BTF texture tiles used by the synthesis algorithm and their size. Also the size of the BTF data buffer can by set by the user. Finally the size of the resulting texture has to be set by the user. User has an opportunity to disable data loading for the preview purposes and speed up the rendering process. Then the BTF texture is not applied to the object.

## 4 Results

Texture plugin together with the modification of the Blender's renderer core allows to use BTF textures directly in the Blender. Especially, UV-mapping of the BTF textures and subsequent rendering can lead to realistic appearance of the 3D models. We have tested the plugin with BTF measurements either from the University of Bonn [Müller et al. 2004] or from the Yale University [Koudelka et al. 2003].

Fig. 10 demonstrates the application of the BTF texture plugin involving BTF texture synthesis during the visualization of the interior of the Ferrari car model. Fig. 11 demonstrates the same 3D scene without application of the BTF textures. At first sight there is noticeable that the application of smooth textures, contrary to BTF textures, does not allow to realistically visualize the effects like reflections or colour changes depending on illumination and viewing conditions. On the other hand, there are visible artificial reflections on the seat covered with textile material.

The advantage of BTF rendering is also demonstrated in the Fig. 12. Images in the top row were rendered using BTF texture measurements while the images in the bottom row were textured with the non-BTF version of the same materials. Considerable difference in the realistic appearance is evident. Similar comparison but in more complex 3D scene is provided in Fig. 13. The example of utilization of the BTF texture measurements in the interior design is shown in the Fig. 14.

To improve the performance of BTF texture manipulation, BTF Roller synthesis step has been implemented into texture plugin. This algorithm contemporary represents the fastest way to seamlessly generate BTF texture of an arbitrary size.

Usage of BTF textures in Blender does not significantly slow down the rendering process. The most time consuming part is loading of BTF image data for desired combination of illumination and viewing direction. Optimized version, as described in section 3.2, reduces the rendering time to 10%.

## 5 Conclusion and Future Work

We presented the novel Blender plugin and the corresponding minor Blender modifications which together enable physically correct realistic rendering of surface materials represented in their most advanced form - the bidirectional texture function. This plugin produces from this open source graphical software system the only rendering system available which allows correct surface materials presentation.

Our current implementation works with sampling based method for BTF texture enlargement. However we plan to generalize the plugin also with some BTF mathematical models. This will not only substantially increase the BTF data compression rate but it will allow simultaneously also rendering of BTF edited measurements. Finally, the future plugin release will benefit from multithread functionality.

## Acknowledgements

This research was supported by grant GAČR 102/08/0593 and partially by the MŠMT grant 1M0572 DAR, GAČR 103/11/0335, CESNET 387/2010.

## References

BENNETT, J., AND KHOTANZAD, A. 1998. Multispectral random field models for synthesis and analysis of color images. *IEEE Trans. on Pattern Analysis and Machine Intelligence 20*, 3 (March), 327–332.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. *ACM Trans. Graph. 22*, 3 (July), 287–294.

DANA, K. J., NAYAR, S. K., VAN GINNEKEN, B., AND KOENDERINK, J. J. 1997. Reflectance and texture of real-world surfaces. In *CVPR*, IEEE Computer Society, 151–157.

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *ACM SIGGRAPH 2001*, ACM Press, E. Fiume, Ed., 341–346.

FILIP, J., AND HAINDL, M. 2009. Bidirectional texture function modeling: A state of the art survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence 31*, 11, 1921–1940.

HAINDL, M., AND FILIP, J. 2004. A fast probabilistic bidirectional texture function model. *Lecture Notes in Computer Science*, 3212, 298 – 305.

HAINDL, M., AND HATKA, M. 2005. BTF roller. In *Texture 2005: Proceedings of 4th Internatinal Workshop on Texture Analysis and Synthesis*, Heriot-Watt University, Edinburgh, M. Chantler and O. Drbohlav, Eds., 89–94.

HAINDL, M., AND HAVLÍČEK, V. 1998. Multiresolution colour texture synthesis. In *Proceedings of the 7th International Workshop on Robotics in Alpe-Adria-Danube Region*, ASCO Art, Bratislava, K. Dobrovodský, Ed., 297–302.

HAINDL, M., AND HAVLÍČEK, V. 2002. A multiscale colour texture model. In *Proceedings of the 16th International Conference on Pattern Recognition*, IEEE Computer Society, Los Alamitos, R. Kasturi, D. Laurendeau, and C. Suen, Eds., 255–258.

HAINDL, M., AND HAVLÍČEK, V. 2000. A multiresolution causal colour texture model. *Lecture Notes in Computer Science*, 1876 (August), 114–122.

HAINDL, M., FILIP, J., AND ARNOLD, M. 2004. BTF image space utmost compression and modelling method. In *Proceedings of the 17th IAPR International Conference on Pattern Recognition*, IEEE, Los Alamitos, J. Kittler, M. Petrou, and M. Nixon, Eds., vol. III, 194–197.

HAINDL, M. 1991. Texture synthesis. *CWI Quarterly 4*, 4 (December), 305–331.

KOUDELKA, M. L., MAGDA, S., BELHUMEUR, P. N., AND KRIEGMAN, D. J. 2003. Acquisition, compression, and synthesis of bidirectional texture functions. In *Texture 2003: Third International Workshop on Texture Analysis and Synthesis*, 59–64.

LEUNG, C.-S., PANG, W.-M., FU, C.-W., WONG, T.-T., AND HENG, P.-A. 2007. Tileable btf. *IEEE Transactions on Visualization and Computer Graphics*, –.

MÜLLER, G., MESETH, J., SATTLER, M., SARLETTE, R., AND KLEIN, R. 2004. Acquisition, synthesis and rendering of bidirectional texture functions. In *Eurographics 2004, STAR - State of The Art Report*, Eurographics Association, Eurographics Association, 69–94.

POLTHIER, K., BELYAEV, A., (EDITORS, A. S., LANGER, T., BELYAEV, E., PETER SEIDEL, H., AND INFORMATIK, M., 2006. Spherical barycentric coordinates.

STAM, J. 1997. Aperiodic texture mapping. Tech. rep., European Research Consortium for Informatics and Mathematics (ERCIM).

TONG, X., ZHANG, J., LIU, L., WANG, X., GUO, B., AND SHUM, H.-Y. 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics (TOG) 21*, 3, 665–672.

**Figure 10:** *Ferrari 360 Spider car model rendered using BTF textures which allows realistic visualization of 3D scene (3D model courtesy of DMI cars 3D models, http://www.dmi-3d.net/).*

**Figure 11:** *Ferrari 360 Spider car model rendered using smooth textures.*

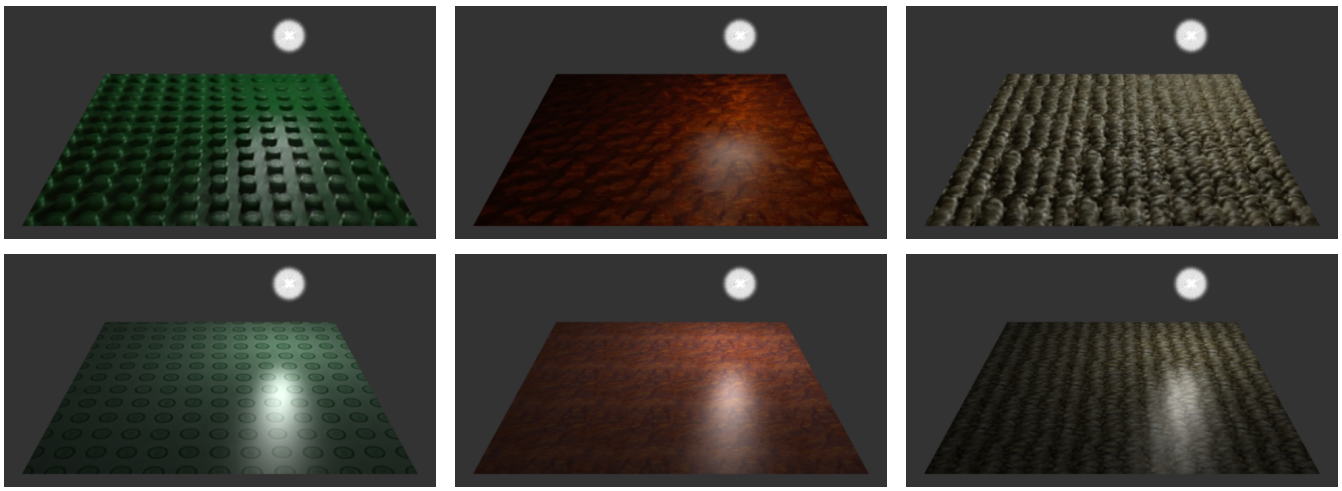

**Figure 12:** *Comparison of the appearance of the plane textured with BTF textures of lego tile, wood and carpet (top row) and corresponding smooth textures (bottom row).*

**Figure 13:** *BTF wood and corduroy textures applied to the chair model (left) and standard non-BTF rendering (right), respectively.*

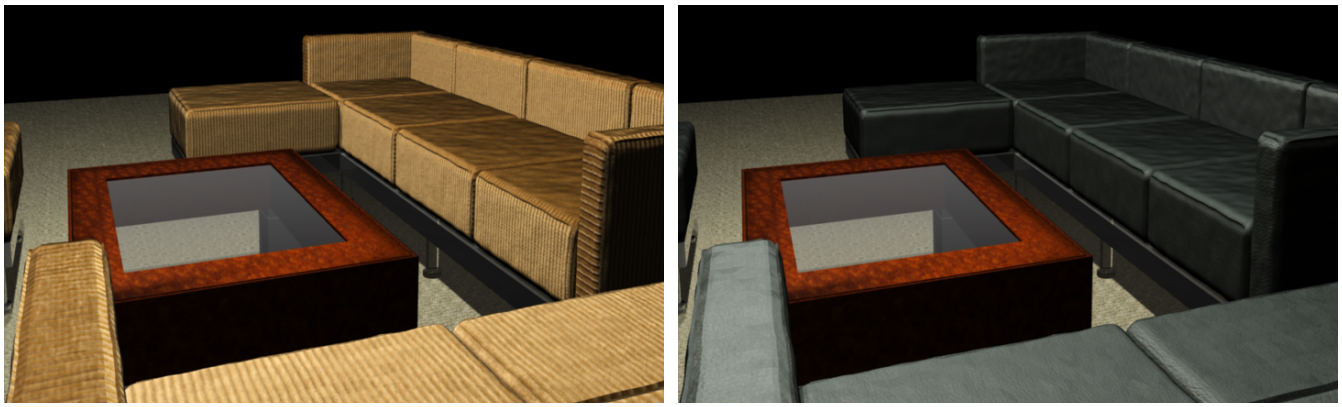

**Figure 14:** *BTF textures can be used in the interior design to create its realistic visualization. The sofa in the image on the left is textured with corduroy texture and in the image on the right with the dark skin texture (3D model courtesy of 3DModelFree.com, http://www.3dmodelfree.com/)*