

# Advanced Material Rendering in Blender



Martin Hatka and Michal Haindl

Institute of Information Theory and Automation of the ASCR, Prague, Czech Republic

**Abstract**—Physically correct and realistic visual appearance rendering or analysis of material surface visual properties require complex descriptive models capable of modelling material dependence on variable illumination and viewing conditions. The recent most advanced representation of visual properties of surface materials is a Bidirectional texture function (BTF). BTF is 7D function of planar coordinates, spectral coordinate, and viewing and illumination angles, respectively. 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 modelling but also makes its rendering noticeably more demanding. BTF textures are acquired by costly measurements of real materials and their subsequent nontrivial processing. While several techniques for measurement or processing of BTF textures have been described already, there is no environment allowing 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 the integral part of its implementation. The presented plugin allows creating realistic computer animations with additional BTF textures of any required size mapped onto an object surfaces while the other functionality of Blender retains.

**Index Terms**—realistic material rendering, bidirectional texture function, Blender

## I. INTRODUCTION

There is neither professional nor any open source 3D graphics application currently available which enables realistic surface materials rendering in its most advanced BTF representation form [3], [5]. 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 available 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 complicated and resources demanding and 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 (<http://www.blender.org>, see Fig. 1). The Blender is 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 for further elaboration.

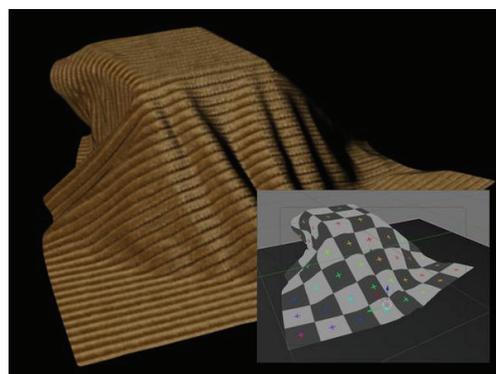


Fig. 1. A 3D drapery model created and textured using UV-mapping in Blender can be easily coated with a BTF material measurements thanks to our BTF texture plugin.

The rest of the paper is organized as follows. Section 2 describes the reflectance function visualization problems, while section 3 proposes a modification of the Blender's renderer to be able to calculate viewing and illumination angles for texturing purposes, and extension of Blender's texture plugin interface. Section 4 provides an insight into the BTF texture plugin architecture. Section 5 details the BTF roller underlying algorithm. Section 6 illustrates some visualization results achieved with the implemented rendered and compares them with the classic inadequate approach where the smooth planar textures were used instead. Finally, the concluding section 7 sums up our achievement and suggest some future development of the renderer.

## II. REFLECTANCE REPRESENTATION AND VISUALIZATION

Real surface material reflectance is very complex phenomenon which depends on too many (sixteen) physical variables to be able to measure or model all of them and all their possible interactions accurately using available tools. Even the recent most advanced approximation of the general reflectance function – the bidirectional texture function (BTF) requires the state-of-the-art [5] nontrivial mathematical models, measurement systems and computer technology both for

processing as well as data storage and visualization. Multispectral BTF neglects light transport time, variable radiance along light rays, and light transmittance among others. BTF is then a seven-dimensional function, which considers measurement dependency on colour 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  $\phi, \theta$  are elevation and azimuthal angles of the illumination and view direction vector (see Fig. 2).

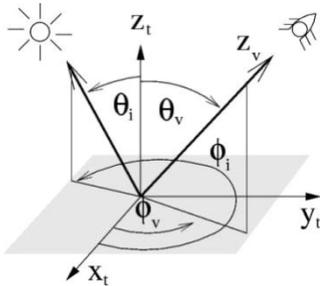


Fig. 2. Relationship between illumination and viewing angles within the sample coordinate system.

The BTF measurements comprise a whole hemisphere of light and camera positions in the observed material sample coordinates according to the selected quantization steps (see Fig. 3).

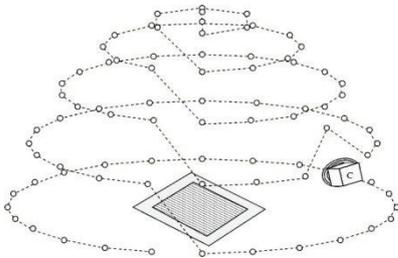


Fig. 3. An example of light trajectory above the sample during measurement while camera is fixed.

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 (modelling) method of this otherwise huge BTF data is inevitable and also constitutes an integral part of our BTF plugin.

### 2.1 BTF Visualization

Applying BTF textures to 3D models surfaces dramatically

enhances the visual appearance of the objects in a 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 the 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 position, the orientation and the scale of the texture on the surface of an object.

### 2.2 Blender

Blender 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 scanline rendering. Modelling techniques are primarily aimed at facet representation of the objects. Bezier curves or NURBS surfaces are supported as well. Animation capabilities incorporate keyframed 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 exists as a separate file on a computer. When called in it communicates with Blender through given interface to generate the texture.

Overall scheme of the proposed solution to BTF texture rendering in Blender is summarized in the Fig. 4.

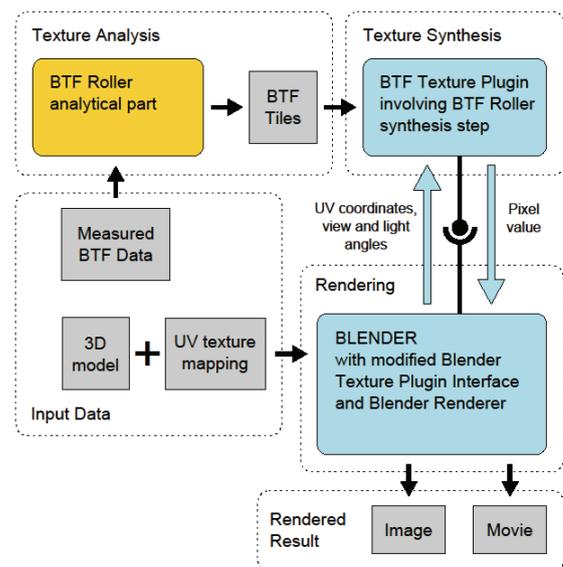


Fig. 4. BTF rendering using Blender, texture plugin, and the BTF Roller texture synthesis algorithm. Texture analysis using the BTF Roller (yellow box) can be done independently before the rendering. Texture synthesis implemented in the texture plugin is performed as inseparable part of the rendering process (red

box). BTF tiles are generated and stored and they are subsequently reused during the rendering.

### III. 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 direction and illumination direction, the texturing is performed earlier than the shading. The utilization of the shaders for the purpose of the BTF texturing has not any sense. This is the reason why the BTF texturing should be solved in different way. The way is to involve the capability to vary the texture appearance on the illumination and viewing conditions directly in the texturing process.

Because of the huge amount of BTF data, BTF support directly in Blender seems to be a very complex, difficult and ineffective task. On the other hand, the texture plugin seems to be an interesting and much simpler way to incorporate BTF textures to Blender. The 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. The second task is to modify the plugin interface to be able to pass these angles to the texture plugin.

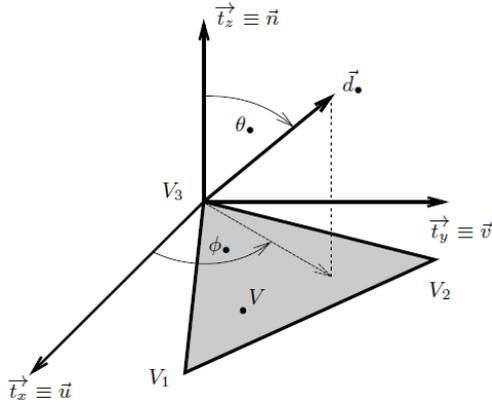


Fig. 5. Computation of the azimuthal and elevation angle for the view and illumination direction is based on a transform of view vector or illumination vector to the texture coordinate system.

#### 3.1 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. The principle of the modification is as follows.

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  $\Delta V_1 V_2 V_3$  (see Fig. 5). The spatial coordinates of the vertices  $V_1, V_2, V_3$  of the facet  $\Delta V_1 V_2 V_3$  in the orthonormal basis  $S=(\mathbf{x}, \mathbf{y}, \mathbf{z})$  are  $(V_1)_S = (v_1^x, v_1^y, v_1^z)$ ,  $(V_2)_S = (v_2^x, v_2^y, v_2^z)$ ,  $(V_3)_S = (v_3^x, v_3^y, v_3^z)$  respectively.

Then the texture mapping function  $T_{UV}, T_{UV}(v_1^x, v_1^y, v_1^z) = (v_1^u, v_1^v)$ , assigns to the vertices  $V_1, V_2, V_3$ , the corresponding texture coordinates  $(V_1)_T = (v_1^u, v_1^v, 0)$ ,  $(V_2)_T = (v_2^u, v_2^v, 0)$ ,  $(V_3)_T = (v_3^u, v_3^v, 0)$ , where  $T=(\mathbf{u}, \mathbf{v}, \mathbf{w})$  is the orthonormal basis of the 3D texture coordinates. Note that the triangular facet  $\Delta V_1 V_2 V_3$  is coplanar with the plane given by the texture axes  $\mathbf{u}$  and  $\mathbf{v}$  and its normal  $\mathbf{n}$  is parallel with the axis  $\mathbf{w}$  of the texture coordinates space. 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.  $\mathbf{t}_x = (\mathbf{u})_S$ ,  $\mathbf{t}_y = (\mathbf{v})_S$  and  $\mathbf{t}_z = (\mathbf{w})_S$ .

As the next step, let's denote the view and illumination directions expressed in spatial coordinates as  $\mathbf{d}_v$  and  $\mathbf{d}_l$  respectively. Then the projection of a vector  $\mathbf{d}_\bullet$  in the directions of the axes  $\mathbf{t}_x, \mathbf{t}_y$  and  $\mathbf{t}_z$  of the texture coordinates expressed in the spatial coordinates yields in a vector  $\mathbf{s}_\bullet$ . Vector  $\mathbf{s}_\bullet$  corresponds to the vector  $\mathbf{d}_\bullet$  expressed in the texture coordinates, i.e.  $\mathbf{s}_\bullet = (v_1^u, v_1^v, v_1^w) = (\mathbf{d}_\bullet)_T$ . Finally, while the vector  $\mathbf{s}_\bullet = (\mathbf{d}_\bullet)_T$  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:

$$\cos \theta = s^w \quad (2)$$

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

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

#### 3.2 BTF Texture Plugin Interface

The texture plugin takes 3D texture coordinate vector  $(u, v, w)$  as the input and returns back the vector  $(y, Y_R, Y_G, Y_B, N_u, N_v, N_w)$  as the output, where  $y$  is the pixel intensity (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 2, BTF is the 7D function of planar coordinates, spectral coordinate, and viewing and illumination

angles,  $BTF(r, \theta_i, \phi_i, \theta_v, \phi_v)$ . The input interface was extended to pass the viewing and illumination angles hence the 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  $v$ , the UV-mapping technique, which is considered as the most accurate, was used.

#### IV. BTF TEXTURE PLUGIN

BTF texture measurement consists of thousands of colour images (section 2) and to incorporate them directly in Blender is difficult. For that reason it proved to be very effective to take advantage of texture plugin. The proposed BTF texture plugin communicates with Blender through the Blender texture plugin interface Figs.9-11 additionally extended of viewing and illumination azimuthal and elevation angles.

The main advantage of this approach is a possibility to implement various texture synthesis algorithms directly in the plugin, particularly the BTF Roller algorithm (sections 4.3 and 5). The plugin performance optimization is then straightforward.

##### 4.1 Barycentric Coordinates Interpolation

BTF textures are measured for a finite number of camera and light source positions, however in practice it is necessary to evaluate pixel values of unmeasured combinations of camera and light source positions. For such a combination interpolation, which is motivated by spherical barycentric coordinates [17], on a triangle with known vertex pixel values is performed. The 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 the required 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

$$Y_P = w_1 Y_{P_1} + w_2 Y_{P_2} + w_3 Y_{P_3},$$

where  $w_1, w_2$  and  $w_3$  are the weights of  $Y_{P_1}, Y_{P_2}$  and  $Y_{P_3}$ ,  $w_1 + w_2 + w_3 = 1$ .

The weights are defined in the following way:

$$w_1 = \frac{V_1}{V}, w_2 = \frac{V_2}{V}, w_3 = \frac{V_3}{V}, \quad V = V_1 + V_2 + V_3, \quad (5)$$

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

$$V_1 = \frac{1}{6} |\det(P, P_2, P_3)|, \quad (6)$$

$$V_2 = \frac{1}{6} |\det(P, P_3, P_1)|, \quad (7)$$

$$V_3 = \frac{1}{6} |\det(P, P_1, P_2)|, \quad (8)$$

##### 4.2 BTF Data Buffer

To store the whole BTF texture dataset in memory is ineffective. The best solution is to implement a buffer to store BTF texture data. The buffer size is the user specified parameter. An 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.

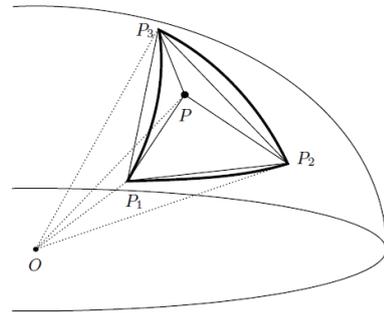


Fig. 6. Interpolation of the pixel  $P$  value is motivated by barycentric coordinates. The value of the  $P$  is a weighted sum of the  $P_1, P_2$  and  $P_3$  while the weight of each pixel corresponds to the volume of the opposite quadrilateral.

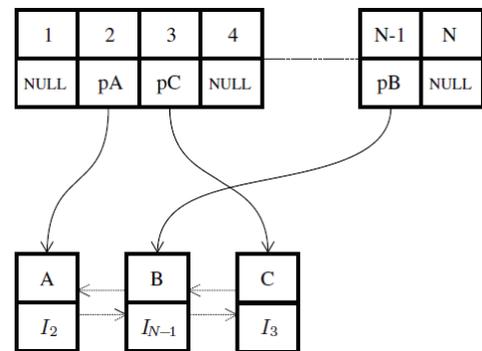


Fig. 7. Buffer with loaded texture data. The buffer consists of double linked list containing image data and an array with pointers to the list nodes. The array is used for fast search of the indexed images.

### 4.3 Texture Synthesis

BTF texture samples have only limited size, typically several hundreds or thousands of pixels. This limited size of texture measurements is the fundamental problem while an object of a 3D scene should be 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 resulting texture visual quality.

In order to generate top quality BTF texture without visible seams in the synthesis step, the texture enlargement sampling type of algorithm, which is called the BTF Roller [7], was incorporated in the plugin (see Fig. 8). The roller method was chosen from possible alternatives ([4], [19], [15] and many others) because it is the 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 efficient 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. The roller algorithm is described in detail in the following section 5.

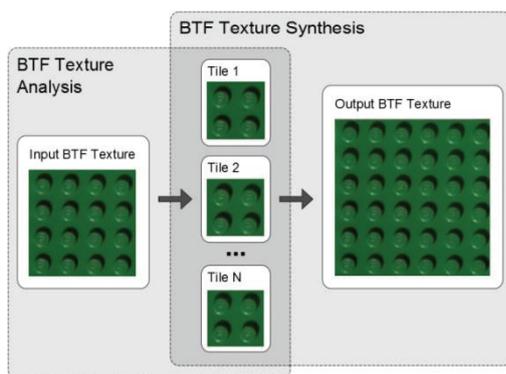


Fig. 8. The principle of the BTF Roller algorithm. During the analytical part, several mutually interchangeable tiles (middle) are extracted from the input texture (left). During the extremely fast synthesis step, synthetic texture (right) from the set of extracted tiles (middle) is generated by random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

Then the plugin input is the set of several rectangular and mutually interchangeable BTF texture tiles. These tiles are precomputed only once 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 Blender, it is inefficient to generate whole BTF slices as random tiling. More efficient approach is to generate aperiodic tiling [2]. In our BTF texture plugin we have used the iterative version [18] 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 [12], [0], [1], [10], [0], [11], [6] 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.

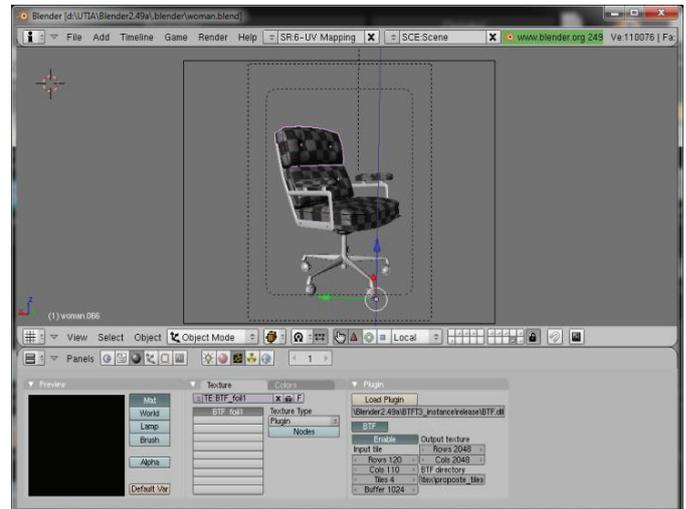


Fig. 9. Screenshot of Blender window with 3D scene containing chair model with UV texture coordinates defined (shown using a test textural pattern).

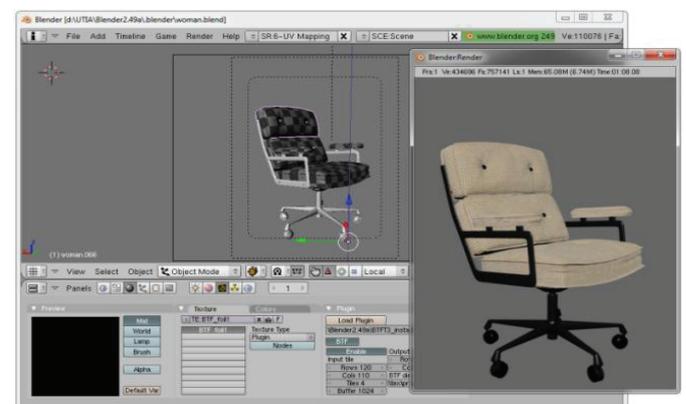


Fig. 10. Screenshot of Blender window with rendered result – chair covered by the realistic textile BTF material.

### 4.4 Plugin Parameters

Blender texture plugin interface provides a capability to define the plugin control panel. BTF texture plugin control panel (see Fig. 11) consists of several control elements. The user must specify a path to the BTF texture data, the number of BTF texture tiles used by the synthesis algorithm, and their size. Also the size of the BTF Data Buffer can be set by the user. Finally the size of the resulting texture has to be set by the user. The user can disable data loading for preview purposes and thus speed up the rendering process. Then the BTF texture is not applied to the object.

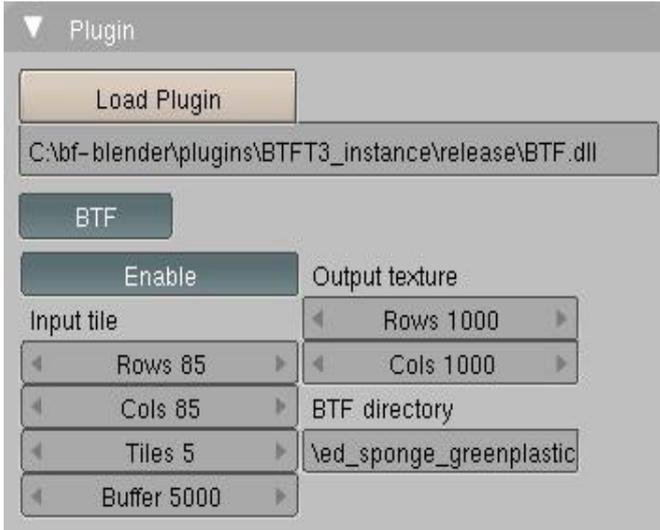


Fig. 11. BTF plugin control panel in Blender. User can specify number of input tiles, size of the buffer and the size of the output to control the quality of the resulting texture and performance of the plugin. The panel is defined inside the plugin but when the plugin is loaded into the Blender then it can be accessed through the Blender

## V. BTF ROLLER ALGORITHM

We assume mutually well registered BTF data of the size  $N$  by  $M$  for changing viewing angle and changing illumination angle. Then the algorithm produces simultaneously identical tiles for all viewing and illumination angles.

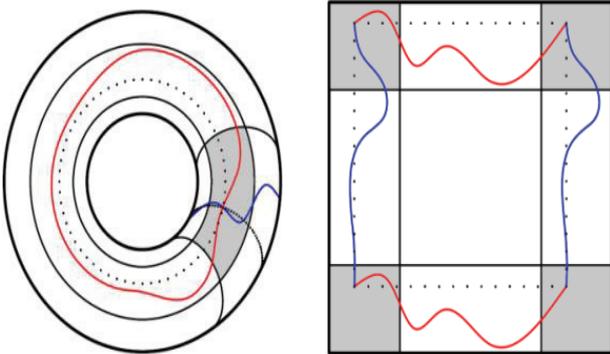


Fig. 12. Idea of double toroidal tile creation is to expand the torus covered with overlapped texture sample to a plane.

### 5.1 Minimal tile size

The minimal rectangle to which the tile is inscribed is limited by the size of BTF measurements, the number  $n$  of toroidal tiles we are looking for, and the sample spatial frequency content. From the Fourier transformation of a single monospectral perpendicularly illuminated texture component we detect the dominant low frequency  $f_r$  we want to preserve. The multiindex  $r$  has two components  $r = [r_1, r_2]$ , the first

component is row and the second one is the column index, respectively. The rectangle vertical size is chosen to be

$$n_{row} \in \left[ \frac{N}{f_{r_1} + 0.5}, \frac{N}{f_{r_1} - 0.5} \right], \quad (9)$$

and if we require  $n > 1$  number of multiple tiles we add additional condition  $n_{row} n_{col} \leq NM/n$ . The horizontal tile size is found similarly.

### 5.2 Overlapping and optimal cut

The double toroidal tile (see Fig. 12) is limited by the selected minimal rectangle to be inscribed in from the original texture measurement. The texture tile is assumed to be indexed on the regular two-dimensional toroidal lattice. The optimal lattice searched by the algorithm allows for seamless repetition in both horizontal and vertical directions, respectively.

Let us define the overlap error for a pixel  $r$  as follows:

$$\psi_r^h = (Y_r - Y_{r+[N-h,0]})^2, \quad \forall r \in I_h, \quad (10)$$

$$\psi_r^v = (Y_r - Y_{r+[0,M-v]})^2, \quad \forall r \in I_v, \quad (11)$$

where  $Y_r$  denotes a multispectral pixel indexed on the  $N$  by  $M$  underlying lattice. The index sets  $I_h, I_v$  are defined  $I_h = (1, \dots, h) \times (1, \dots, M)$ ,  $I_v = (1, \dots, N) \times (1, \dots, v)$ . The horizontal and vertical overlaps are found from the following relations:

$$h = \frac{N}{2f_{r_1}}, \quad v = \frac{M}{2f_{r_2}}. \quad (12)$$

The optimal cuts for both the horizontal and vertical edge are searched in the corresponding overlaps using the dynamic programming method. Alternatively some other suboptimal search such as the  $A^*$  algorithm can be used if necessary to speed up also the analytical part of the method. The combination of both optimal vertical and horizontal cuts creates the toroidal tile as is demonstrated on Fig. 12.

For more efficient storage and manipulation the resulting tile is converted to the rectangular shape.

### 5.3 Multiple tiles

Some textures with dominant irregular structures cannot be modelled by simple singular tile repetition without clearly visible and visually disturbing regularly repeated effects. These textures are modelled by random ordering of several tiles which have identical tile borders but different internal content. Using analogous approach to the optimal tile border selection we find optimal cut between the optimal tile and new tile filling from another part of the input texture. All multiple tiles have the same size.

### 5.4 Enlargement

The synthesis of any required BTF texture size for a single tile case is simple repetition of the created double toroidal tile in both directions until the required textured area is generated. There is no computation involved in this step hence it can be easily implemented in real time. In the case of several mutually interchangeable tiles uniform random generator is needed to decide which tile will follow. However, this additional computation has negligible overhead.

Within the BTF texture plugin for Blender, the random generator was replaced by an aperiodic tiling generator.

## VI. RESULTS

The texture plugin together with the Blender's renderer core modification allow using BTF textures directly in Blender. Especially, UV-mapping of BTF textures and their subsequent rendering can lead to very realistic appearance of any 3D models. We have tested the plugin with BTF measurements either from the University of Bonn [16] or from the Yale University [14].

Figs. 13, 14 demonstrate the application of the BTF texture plugin involving BTF texture synthesis during the visualization of the Ferrari car model interior.



Fig. 13. 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/>).

The advantage of BTF rendering is also demonstrated in Fig. 15. 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 appearance realism is evident.

At first sight there is noticeable that the restriction to smooth textures, contrary to BTF textures, does not allow realistic visual effects like reflections or colour changes due to varying illumination and viewing conditions.



Fig. 14. Ferrari 360 Spider car model rendered using BTF textures which allow realistic visualization of 3D scene.

The example of utilization of the BTF texture measurements in the interior design is shown in the Figs. 16, 17.

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 a desired combination of illumination and viewing direction. The optimized version, as described in section 4.2, reduces this rendering time to 10%.

## VII. CONCLUSION

We presented the novel Blender plugin and the corresponding minor Blender modification 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 simultaneously allow also rendering of BTF edited measurements. Finally the future plugin release will benefit from the multithread functionality.

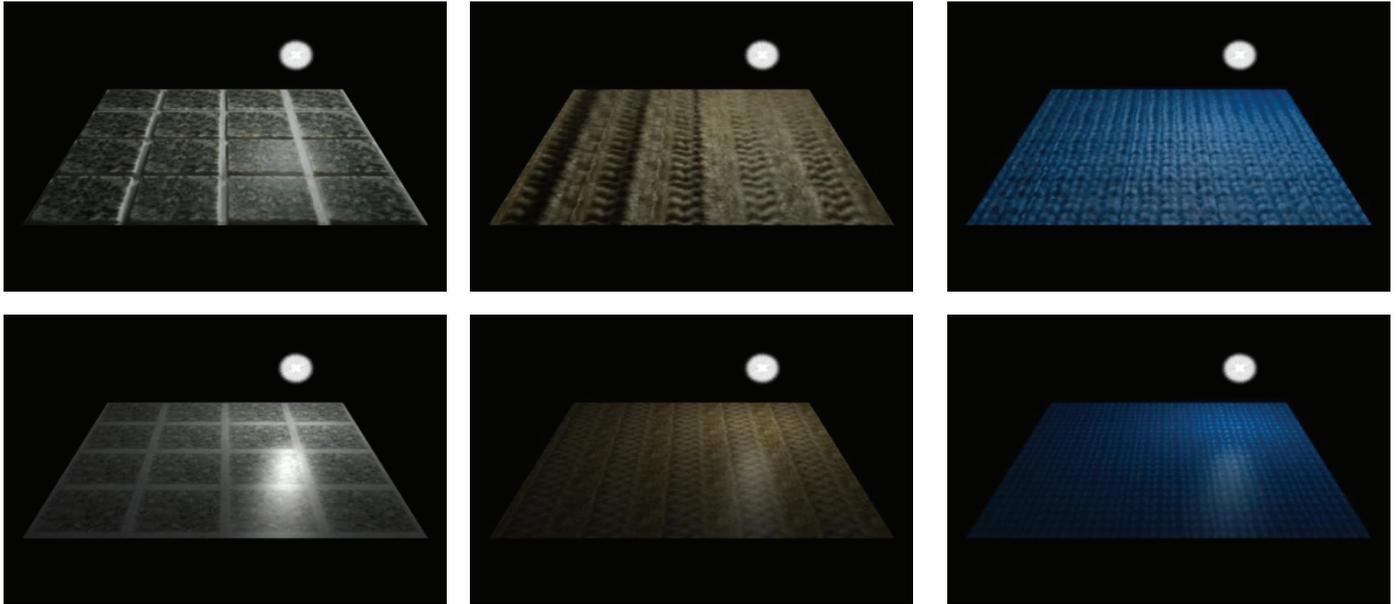


Fig. 15. Comparison of the appearance of the plane textured with BTFs of impala, pulli, and wool (top row) and corresponding smooth textures (bottom row).



Fig. 16. BTF corduroy, fabric, and proposte textures, respectively, applied to the chair model.

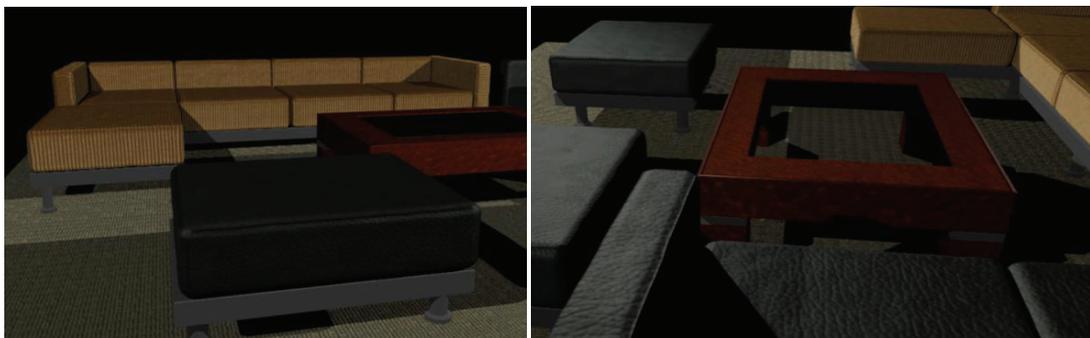


Fig. 17. BTF textures can be used in the interior design to create its realistic visualization. The sofas are textured with corduroy texture and with the dark leather texture (3D model courtesy of 3DModelFree.com, <http://www.3dmodelfree.com/>).

## ACKNOWLEDGEMENT

This research was supported by grant GAČR 102/08/0593 and partially by the projects GAČR 103/11/0335 and CESNET 387/2010, 409/2011.

## REFERENCES

- [1] J. Bennett and A. Khotanzad, 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.
- [2] M. F. Cohen, J. Shade, S. Hiller and O. Deussen, 2003. Wang tiles for image and texture generation. *ACM Trans. Graph.* 22, 3 (July), 287–294.
- [3] K. J. Dana, S. K. Nayar, B. Van Ginneken and J. J. Koenderink, 1997. Reflectance and texture of real-world surfaces. In *CVPR, IEEE Computer Society*, 151–157.
- [4] A. A. Efros and W. T. Freeman, 2001. Image quilting for texture synthesis and transfer. In *ACM SIGGRAPH 2001, ACM Press*, E. Fiume, Ed., 341–346.
- [5] J. Filip and M. Haindl, 2009. Bidirectional texture function modeling: A state of the art survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31, 11, 1921–1940.
- [6] M. Haindl and J. Filip, 2004. A fast probabilistic bidirectional texture function model. *Lecture Notes in Computer Science*, 3212, 298 – 305.
- [7] M. Haindl and M. Hatka, 2005. BTF roller. In *Texture 2005: Proceedings of 4th International Workshop on Texture Analysis and Synthesis*, Heriot-Watt University, Edinburgh, M. Chantler and O. Drbohlav, Eds., 89–94.
- [8] M. Haindl and V. Havlíček, 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.
- [9] M. Haindl and V. Havlíček, 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.
- [10] M. Haindl and V. Havlíček, 2000. A multiresolution causal colour texture model. *Lecture Notes in Computer Science*, 1876 (August), 114–122.
- [11] M. Haindl, J. Filip and M. Arnold, 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.
- [12] M. Haindl, 1991. Texture synthesis. *CWI Quarterly* 4, 4 (December), 305–331.
- [13] M. Hatka, M. Haindl, 2011. BTF Rendering in Blender. In *Proceedings of VRCAI 2011: ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications to Industry*, ACM, Inc. 265-272.
- [14] M. L. Koudelka, S. Magda, P. N. Belhumeur, and D. J. Kriegman, 2003. Acquisition, compression, and synthesis of bidirectional texture functions. In *Texture 2003: Third International Workshop on Texture Analysis and Synthesis*, 59–64.
- [15] C.S. Leung, W.M. Pang, C.W. Fu, T.T. Wong and P.A. Heng, 2007. Tileable btf. *IEEE Transactions on Visualization and Computer Graphics*,
- [16] G. Müller, J. Meseth, M. Sattler, R. Sarlette and R. Klein, 2004. Acquisition, synthesis and rendering of bidirectional texture functions. In *Eurographics 2004, STAR – State of The Art Report*, Eurographics Association, Eurographics Association, 69–94.
- [17] K. Polthier, A. Belyaev, A. S.Editors, T. Langer, E. Belyaev, H. Peter Seidel and M. Informatik, 2006. Spherical barycentric coordinates.
- [18] J. Stam, 1997. Aperiodic texture mapping. Tech. rep., European Research Consortium for Informatics and Mathematics (ERCIM).
- [19] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo and H.Y. Shum, 2002. Synthesis of bidirectional texture functions on arbitrary surfaces. *ACM Transactions on Graphics (TOG)* 21, 3, 665–672.

**Martin Hatka** is currently a PhD student at the Department of Pattern Recognition at Institute of Information Theory and Automation of the ASCR, Prague, Czech Republic. He received his master's degree in Software Engineering from the Faculty of Nuclear Sciences and Physical Engineering of the Czech Technical University in Prague in 2006. His research interests are pattern recognition, image processing, texture modelling, 3D computer graphics and rendering



**Michal Haindl** (M'95–SM'04) graduated in control engineering from the Czech Technical University, Prague, in 1979. He received his PhD in technical cybernetics from the Czechoslovak Academy of Sciences (1983) and the ScD (DSc) degree from the Czech Technical University (2001). From 1983 to 1990, he was with the Institute of Information Theory and Automation of the Czechoslovak Academy of Sciences, Prague, working on different adaptive control, image processing, and pattern recognition problems. From 1990 to 1995, he was with the University of Newcastle, Newcastle; Rutherford Appleton Laboratory, Didcot; Centre for Mathematics and Computer Science, Amsterdam; and the Institute National de Recherche en Informatique et en Automatique, Rocquencourt, working on several image analysis and pattern recognition projects. In 1995, he rejoined the Institute of Information Theory and Automation where he is the Head of the Pattern Recognition Department. He is also the professor at the Czech Technical University, Prague. Prof. Haindl is an IAPR Fellow and a senior member of the IEEE. He is the author or co-authors of about 280 research papers published in books, journals, and conference proceedings. His current research interests are in pattern recognition applications of random fields, image processing, and automatic acquisition of virtual reality models.