Eurographics Symposium on Rendering (2004)
A. Keller, H. W. Jensen (Editors)

Generalized Displacement Maps

Xi Wang† Xin Tong* Stephen Lin* Shimin Hu Baining Guo* Heung-Yeung Shum*

Tsinghua University * Microsoft Research Asia

Abstract

In this paper, we introduce a real-time algorithm to render the rich visual effects of general non-height-field geometric details, known as mesostructure. Our method is based on a five-dimensional generalized displacement map (GDM) that represents the distance of solid mesostructure along any ray cast from any point within a volumetric sample. With this GDM information, we propose a technique that computes mesostructure visibility jointly in object space and texture space which enables both control of texture distortion and efficient computation of texture coordinates and shadowing. GDM can be rendered with either local or global illumination as a per-pixel process in graphics hardware to achieve real-time rendering of general mesostructure.

1. Introduction

Fine scale surface geometry, known as mesostructure, is an integral component in the appearance of many real-world materials and objects. The rendering of mesostructure provides not only fine resolution form to a surface, but also rich visual details such as fine-scale shading, shadows, occlusions and silhouettes. To enhance the realism of synthesized images, much attention has been focused on efficient and comprehensive methods for rendering mesostructure and its detailed appearance features.

The two most common approaches to mesostructure rendering are by mapping images and by mapping geometry onto a surface. Bidirectional texture functions (BTFs) [DNvGK99] record images of a mesostructure sample under different lighting and viewing directions, but since this representation contains no 3D geometric information, silhouettes and the effects of surface curvature cannot be rendered. These problems can be handled by mapping mesostructure geometry onto a surface to form a detailed object model, as done with displacement maps [Coo84] and volumetric textures [KK89, Ney98]. Although the visual effects of mesostructure can be rendered from such a model, the large amount of detailed geometry mapped into the object space can lead to considerable processing.

The burdens of handling directly mapped geometry in object space can be avoided by precomputing the visibility of mesostructure points and storing it in texture space, as proposed in view-dependent displacement mapping (VDM) [WWT*03]. With this information, VDM achieves real-time rendering of mesostructure visual effects including silhouettes. This method, however, suffers from two significant drawbacks. One is that it can be applied only to height-field

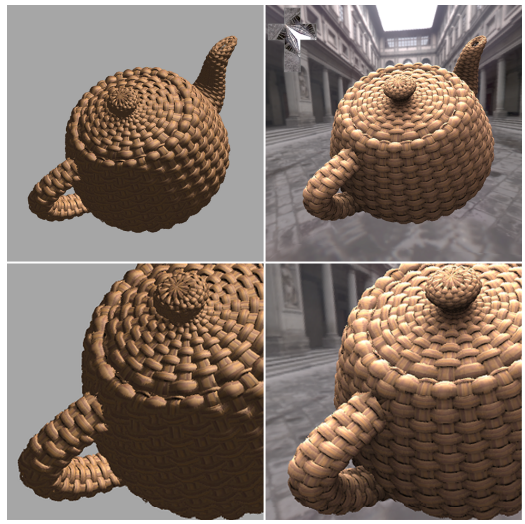

Figure 1: *Mesostructure rendering with GDM. Left column: under local illumination. Right column: under global illumination.*

† This work was done while Xi Wang was a visiting student at Microsoft Research Asia.

geometry on closed surfaces. The other is that VDM can be practically precomputed for only a limited class of surface shapes, e.g., with respect to a single surface curvature parameter as in [WWT*03]. As a result, the precomputed VDM lacks the specificity to accurately represent surfaces with curvature variations and texture warping, which can consequently lead to significant texture distortions.

In this paper, we present a general mesostructure rendering technique based on a proposed *generalized displacement map* (GDM), which represents the distance of solid mesostructure along any ray cast from any point within a volumetric texture. Unlike previous displacement map methods which model surfaces using one offset value per texel, generalized displacement maps can handle arbitrary non-height-field mesostructures. Furthermore, the GDM approach overcomes both the texture distortion problems of VDM and the computational expense of directly mapped geometry by computing visibility jointly in texture space and object space. The precomputed GDM values facilitate texture coordinate computation of viewing rays whose piecewise linear trajectory in texture space is determined in object space according to the shape of the base mesh. By accounting for curvature variations and texture warping in object space, ray intersections in texture space are more accurately established, resulting in less texture distortion.

The GDM can similarly be used to quickly determine whether the intersection point is shadowed from local illumination, and can moreover be efficiently employed for scenes with global illumination. With a simple per-pixel implementation in graphics hardware, GDM can render complex volumetric textures including silhouettes in real time to produce detailed mesostructure appearance as shown in Fig. 1. The contributions of this work are summarized as follows:

- the generalized displacement map for rendering of general non-height-field mesostructure on both open and closed surfaces
- visibility computation in texture and object spaces to reduce texture distortion and increase efficiency
- a hardware implementation of GDM for real-time rendering with either local and global illumination

2. Related Work

Previous methods for mesostructure rendering can be categorized by the dimensionality of their geometric representations. Highly realistic mesostructure appearance can be modelled from collections of 2D images such as BTFs. Polynomial texture maps [MGW01] represent mesostructure appearance of each surface point under different lighting directions by fitting a biquadric polynomial. For non-diffuse surfaces, this approach models only a fixed viewpoint. Recently, [SLSS03] integrated the BTF with precomputed radiance transfer of macro-scale geometry, and then rendered the bi-scale lighting effects in real time. Although these image-based representations capture true appearances of a general mesostructure sample, the 2D geometric structure inherent in images precludes rendering of mesostructure silhouettes.

Most prior techniques assume that mesostructures have the form of height fields on a mesh, a 2-1/2 D representation. For rendering mesostructures that have height field geometry, bump mapping [Bli78] and its extensions [Max88, SC00, BM93, HDKS00] offer an efficient approach, but do not account for silhouettes. Silhouettes can be rendered by displacement maps [Coo84, CCC87], which explicitly model the geometric details of height fields. While techniques based on height fields benefit from relative ease in processing, they lack the generality to describe a range of surface geometries that includes weave patterns and slanted protrusions.

Volumetric textures provide a general 3D representation of mesostructure as volumetric data sampled on 3D regular grids [KK89, Ney98]. Traditionally, volumetric textures are rendered by tracing rays through a shell volume mapped on a surface, which is expensive for real time applications. Recently, a set of slice-based approaches have been developed for volumetric texture rendering. Meyer et al. [MN98] rendered the volume prism extruded from each triangle as a stack of axis-aligned textured slices. Lensch et al. [LDS02] used a set of slices parallel to the projection plane to render the shell volume. In [LPFH01], volumetric fur mapped on a surface is rendered as concentric layers from the skin outwards, and extruded fins from triangle edges near the silhouette are also rendered to reduce artifacts. Daubert et al. [DS02] applied a precomputed visibility map to render shadowing effects of thin knitwear. In these slice-based methods, the number of slices used for each triangle increases with volumetric resolution. This encumbers the rendering process with additional overhead that is magnified when sophisticated per-pixel shading is incorporated. To our knowledge, no previous work has been able to render volumetric textures with global illumination in real time.

Related methods for rapid software rendering include the work of Arvo and Kirk [AK87], which uses view-dependent distance data in volumes to accelerate ray tracing. Dischler [Dis98] used ray tracing in texture space to map mesostructures onto base geometry. He also proposed a special data structure to accelerate the software rendering. Daubert et al. [DKS*03] precomputed visibility information and then reused it for shadow computation and indirect illumination. In GDM, visibility information is used for rendering both silhouettes and shadows in real time.

3. GDM Modeling and Rendering

Our mesostructure rendering method takes mesostructure geometry as input and computes its GDM. After the GDM texture is mapped to a surface, rendering is performed by a per-pixel algorithm that can be accelerated in hardware. The details of GDM modeling and rendering are described in this section.

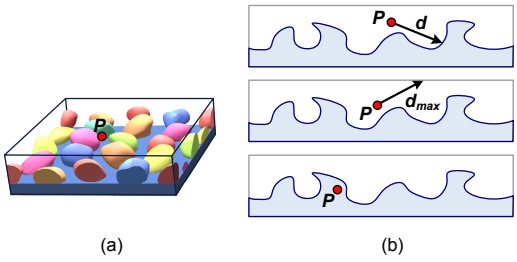

Figure 2: *GDM model. (a) The GDM for a ray cast from point P in direction V is defined as the distance of the nearest solid volume. (b) Three cases in GDM computation. From top to bottom: P outside solid volume with an intersection in direction V, P outside solid volume with no intersection in direction V, P lying within solid volume.*

3.1. Modeling

The GDM values for a given mesostructure are computed at points on a 3D regular grid according to existing sampling algorithms used for volumetric texture construction [KK89, Ney98]. With these sampling algorithms, artifacts may nevertheless arise due to high-frequency mesostructure details, but could be reduced with higher sampling rates or mesostructure pre-filtering. As illustrated in Fig. 2(a), from each point $p = (x,y,z)$ a ray is cast toward sampled viewing directions $\vec{V} = (\theta,\phi)$ expressed in spherical angles, where the polar angle θ ranges from 0 to π. The 5D GDM $d_{GDM}(x,y,z,\theta,\phi)$ records the distance along each viewing direction to the nearest mesostructure surface, according to

$$d_{GDM}(x,y,z,\theta,\phi) = \begin{cases} d & \text{if ray intersects mesostructure} \\ d_{max} & \text{if no intersection exists} \\ 0 & \text{if p inside solid volume} \end{cases}$$

The three different cases for this measurement are displayed in Fig. 2(b). For points in free space or lying on the mesostructure surface, if the ray from p in direction $\vec{V}$ intersects mesostructure, the displacement is the distance d between P and the closest intersection. If no intersection exists, then we record a special value d_{max} that indicates this case. For a point located inside the solid volume of the mesostructure sample, its displacement value is zero. To ensure that rays exit through the top or bottom plane of the texture and not through the texture boundaries, we surround the mesostructure volume by identical tiles in the x,y plane before computing the GDM.†

The GDM can then be mapped onto an arbitrary surface like a volumetric texture $V(x,y,z)$ with a 2D function of (θ,ϕ) stored in each voxel. Mapping a volume texture onto a triangle mesh extrudes each triangle vertex towards its normal direction to generate a prism volume for each mesh triangle, as exemplified in Fig. 3. Hence, the mesostructure of each prism in the object space is given by a corresponding volume in the texture space. The bilinear prism fins are each approximated by two mesh triangles. Each fin is shared by two prisms, and its normal directions are defined outwards with respect to the prism being processed.

† An arbitrary volume texture can be made tileable by using constrained texture synthesis [WL01].

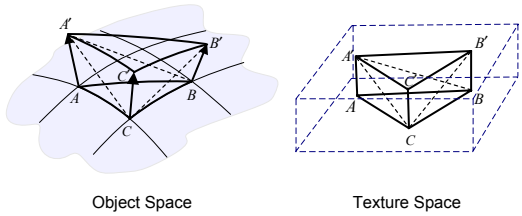

Figure 3: *An extruded prism in object space, and its corresponding volume in texture space.*

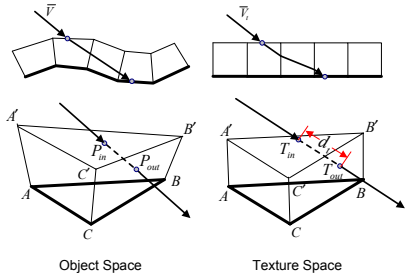

Figure 4: *Ray path in object space and its corresponding path in texture space.*

3.2. Rendering

In contrast to methods for volumetric texture rendering, the GDM rendering algorithm does not trace rays through mesostructures synthesized onto a surface, nor does it involve formation of numerous volume texture slices. Instead, it rapidly processes ray segments in each extruded prism as illustrated in Fig. 4. In object space, a ray may pass through multiple prisms with varying orientations according to the shape of the base mesh. Since mesostructure geometry is not explicitly represented in object space, each ray segment in a prism is mapped to a corresponding ray segment in texture space to determine intersections with geometry. For each prism, its ray in texture-space is approximated as a straight-line segment between the ray entrance and exit points of the prism, as described in [KK89]. From the resulting piecewise-linear ray path in the texture space, the ray intersection can be rapidly determined from the GDM, which contains the displacements along each of these line segments.

This joint use of object and texture spaces for determining ray intersections benefits from greater efficiency by the use of GDM in the texture space and from reduced texture distortion, since computing the ray path in object space accounts for variations in surface curvature and texture warp-

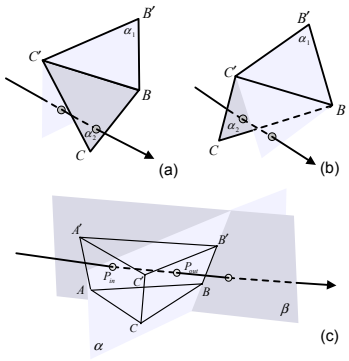

Figure 5: *Determination of ray exit point from a prism. (a) Concave fin. (b) Convex fin. (c) Intersections with prism face planes.*

ing. In principle, the texture distortion could be made arbitrarily small by more finely tessellating a surface to obtain a more precise piecewise-linear ray path in texture space.

In rendering, each prism is processed individually and can be handled in any order. For each prism, its forward-facing sides, which may include prism fins and triangles of the base surface, are rasterized. For each rasterized face, its pixels are then processed by our rendering algorithm, which is summarized in the following pseudocode.

```
PixelShading(P_in, T_in)
  compute exit point (P_out, T_out) of ray segment in prism
  compute texture space length d_t = |T_out − T_in| of ray segment
  compute view direction V_t in texture space
  query the GDM value d_GDM(T_in, V_t) of p
  If ( d_GDM < d_t )
    compute texture coordinate T of intersection point
    shading computation
  Else
    no intersection
```

In the pixel shader, from the object and texture space coordinates P_{in} and T_{in} of the ray entrance point into the prism, the prism exit point P_{out} is first computed to obtain the texture space ray segment $\overline{T_{in}T_{out}}$. From this ray segment, we determine whether a mesostructure intersection point exists in the prism using the GDM. If an intersection exists, its texture coordinate T is calculated with the GDM, and its shading under either local or global illumination is computed. The details of these steps are explained in the remainder of this section.

Intersection computation: To render each pixel, we must compute the mesostructure intersection point of its cast ray, as well as the texture value of that scene point. For the traced ray of a pixel, its direction V, prism entrance point P_{in} and corresponding texture coordinate T_{in} are known. To locate the other endpoint T_{out} of the ray segment in texture

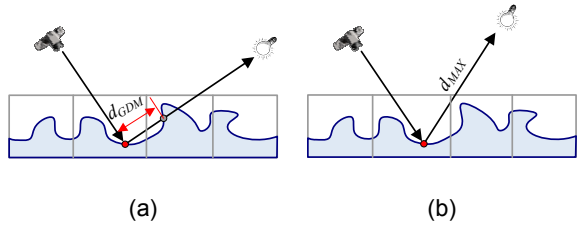

Figure 6: *Shadow determination for local illumination using GDM. (a) shadowed point; (b) directly illuminated point. Since textures mapped onto neighboring triangles come from a continuous region in texture space, GDM values account for shadows cast from neighboring prisms.*

space, we must compute the object space point P_{out} where the traced ray would exit the prism if it were unimpeded by mesostructure geometry, as illustrated in Fig. 4.

From P_{in} and V, the position of P_{out} can be calculated based on ray intersections with the planes defined by other faces of the prism as shown in Fig. 5(c). Our method computes ray-plane intersections rather than hardware-accelerated ray-triangle intersections [CHH02, PBMH02] because ray-triangle intersection computations are complex in comparison and are performed per pixel. Ray-plane intersections involve only per-vertex calculations.

We first compute for each fin the intersection points on the two planes defined by its two triangles. As shown in Fig. 5(a), if two triangles α_1, α_2 form a concave side of the prism, then the intersection for the face is determined as the one with the larger distance from P_{in}. If the triangles form a convex side as in Fig. 5(b), then the intersection with the fin is taken as the point with the smaller distance. From the ray intersections on each face plane, P_{out} is determined as the point with the smallest distance from P_{in}. The texture coordinate of P_{out} is then assigned to T_{out}.

From the texture-space ray segment defined by T_{in} and T_{out}, we approximate the ray direction V_t in texture space as $(T_{out} - T_{in})/d_t$, where the ray segment length is $d_t = ||T_{out} - T_{in}||$. If $d_{GDM}(T_{in}, V_t) > d_t$, then the ray passes through the prism without intersection. Otherwise, the ray intersects mesostructure at a point P in the prism with texture coordinate

$$T = T_{in} + d_{GDM}(T_{in}, V_t) \cdot \vec{V_t},$$

which is used for rendering the pixel.

Local Illumination: If the object is illuminated by a local light source, the GDM can be used to determine whether P is shadowed by other parts of the mesostructure. This can be done by rendering the shadow map with the GDM and saving the distance of the ray intersection instead of the color. Note that this shadow map generation differs from the traditional implementation in that it is computed taking the mesostructure into consideration.

Instead of computing a shadow map, our implementation employs a more rapid solution by approximating the light trajectory L_t in the texture space as the object space light direction L. As illustrated in Fig. 6 if $d_{GDM}(T,L)$ is not equal to d_{max}, this indicates that P is occluded in the lighting direction. If P is in shadow, its appearance is computed with ambient illumination only. Otherwise, P is shaded according to a given reflectance model with parameters stored in an additional 3D texture map. We note that this technique accounts only for the fine-scale shadows of local mesostructure, and does not handle shadows from the base geometry or distant mesostructure. Also, it assumes the base geometry to be locally flat, since local shadow alias may occur in areas of high surface curvature. Even with this assumption, acceptable results can nevertheless be achieved, as in horizon maps [Max88] and VDM [WWT*03].

Global Illumination: To compute the shading of P under global illumination, we adapt the bi-scale radiance transfer framework proposed in [SLSS03], which precomputes the transfer of global illumination at each vertex to illuminance at the meso-scale, and then computes the meso-scale response to this illuminance for each pixel. The only difference of our rendering algorithm is that it defines the meso-scale response on mesostructure surface points, rather than on the reference plane [SLSS03]. This response, called the radiance transfer volume (RTV), accounts for the shadowing and interreflection of the meso-scale lighting within the mesostructure volume, and is expressed as a set of 5D functions that correspond to spherical harmonic coefficients $f^i_{VRF}(x,y,z,\theta,\phi)$, $i = 0...k$, where (θ,ϕ) represents the viewing direction in spherical angles. The number of spherical harmonic coefficients k is chosen by the user, and is set to 16 in our implementation. With this spherical harmonic representation, the shading of a voxel is calculated simply by taking the inner product of the RTV vector and the meso-scale illuminance vector determined by precomputed radiance transfer (PRT) [SLSS03]. The details of the hardware implementation are given in Sec. 4.1.

Compared to the radiance transfer texture (RTT) of [SLSS03] which is defined on the reference plane, the RTV provides several benefits for bi-scale radiance transfer. First, both mesostructure silhouettes and its shading effects can be rendered with the RTV/GDM, while mesostructure silhouettes are ignored in bi-scale radiance transfer. Second, by representing reflectance with respect to mesostructure position p, RTV obtains better compression because of greater data coherence in comparison to RTT, for which reflectance at a reference surface point corresponds to different mesostructure points depending on the viewing angle. As a result, little compression can be achieved in the view dimensions by RTT. Finally, for curved surfaces, large viewing angles from the surface normal direction can result in significant differences in both position and local coordinate frames of P and p. This difference consequently introduces error in computing the incident illumination at p. More accurate incident illumination can be obtained by our method because the prism fins partition the texture into smaller volumes.

4. Hardware Implementation

4.1. Data Compression

Although the 5D GDM/RTV data could be reorganized into a 3D texture and directly loaded to hardware for rendering, it is still large with regard to the limited graphics memory. Moreover, the lower coherence among the reorganized data decreases the performance of the texture access.

For better performance and conservation of hardware memory, our method decomposes a GDM into several low dimensional maps. We achieve this through singular value decomposition (SVD), a method that has been used for compressing high dimensional data [KM99]. The high-dimensional GDM data is reorganized into a 2D matrix A where the rows are indexed by x,y,ϕ and the columns are indexed by θ,z. Although these maps could be differently organized for decomposition, experiments have shown that this solution provides the best compromise between accuracy and storage for all data used in this paper. Applying SVD to A produces a set of 2D eigen-maps $E_i(\theta,z)$ and corresponding 3D weight maps $W_i(x,y,\phi)$. The GDM data can be well reconstructed from a small number of these eigen-maps as

$$d_{GDM} = \sum_i W_i(x,y,\phi)E_i(\theta,z).$$

In practice, the special value d_{max} in the GDM data result in high frequency variations in GDM data, which requires more eigen-maps for good reconstruction. To address this problem, we replace d_{max} with the distance between P and the ray intersection point at the top or bottom boundary of the volume, as shown in the middle row of Fig.2(b). In rendering, we need to query the properties of the intersection point to determine whether the intersection actually exists inside the prism.

For compression, the RTV data is reorganized into matrices where the rows are indexed by x,y,z and columns are indexed by θ,ϕ. The RTV data is also compressed by SVD, and four eigenfunctions are used for representation.

4.2. Rendering Passes

GDM rendering is performed in multiple passes on graphics hardware. In the first pass, GDM computation is performed to obtain the volumetric texture coordinates of the mesostructure point for each pixel. The exit point P_{out} for each pixel is computed in the pixel shader based on interpolated distance values stored in the prism vertices. For each prism vertex, we project it in direction V onto the plane of each back-facing prism triangle and compute its distance. These distances and their texture coordinates are then sent

into the graphics pipeline as vertex attributes. After rasterization, the distances of the pixel p to each plane are interpolated from the vertex attributes and then input into the pixel shader to calculate d_t and $\vec{V}_t$.

Then the shading computation is done in the subsequent passes. For local illumination, the shadow test and shading computation can be finished by the pixel shader in one pass. For global illumination, we first compute the per-vertex matrix/vector multiplication on the CPU to obtain the incoming meso-scale radiance. Then the RTV vector for each pixel is reconstructed and dotted with the interpolated incoming radiance vector to obtain the final shading result in the pixel shader. Depending on hardware capability, the dot product of two vectors is finished in one or more rendering passes.

5. Results

We tested our algorithm on a 2.8GHZ Pentium IV PC with an ATI Radeon 9800XT 256MB graphics card. Hardware-accelerated GDM rendering is implemented using the ARB vertex and fragment program with OpenGL.

Differences between GDM and VDM rendering are exemplified for an open surface in Fig. 8(a-b). Noticeable for VDM is the lack of mesostructure geometry at the surface edges, which are correctly rendered by GDM. Fig. 8(c-d) displays differences in texture distortion between GDM and VDM, which are visually apparent at the curvature variations on the neck. For the undulating surface, GDM renders at 123.2 FPS, and VDM performs at 179.5 FPS. For the vase, the rendering rates are 69.6 FPS for GDM, and 110.4 FPS for VDM. The difference in speed can mainly be attributed to the larger number of geometry computations in GDM, including the processing of prism fins in addition to the top face.

Fig. 9(a) shows under local illumination a rendered vase with a mesostructure interior that consists of empty space. Fig. 9(b) displays a torus decorated with chain links, a typical non-height field mesostructure, and Fig. 9(c) exhibits a bird with weave mesostructure. Visible are the complex shadowing effects under different lighting conditions and the well-defined silhouette under different viewing directions. Fig. 9(d) illustrates the rendering results of a dinosaur whose head and feet are rendered with conventional bump textures. The overlapped squamae on the dinosaur body cannot be accurately processed by conventional displacement mapping.

Fig. 7 displays the rendering performance of the examples shown in Fig. 9 and Fig. 1 for a 512x512 window. All GDM data listed here are sampled for 16×32 viewing directions at equal intervals in ϕ and θ, though uniform sampling over a sphere could potentially generate improved GDM data sets. Since SVD-compressed GDM and RTV data are directly used for rendering, the memory consumption for the data sets is manageable.

For comparison, we implemented a slice-based rendering algorithm on the same platform, in which the mesostructure is rendered under local illumination and without shadowing. It renders the bird model with a $64 \times 64 \times 16$ weave volume at 32fps. The rendering performance decreases to 15fps with an increased volume resolution ($64 \times 64 \times 32$) of the same data. In contrast, our method renders these two data sets with all visual effects at 60fps and 57fps respectively. Unlike conventional slice-based methods, the performance of GDM rendering depends primarily on output resolution and is not affected by the depth resolution of the volumetric textures.

Fig. 10 displays tree bark rendered with global illumination, in which the lighting effects of the macro geometry and the fine-scale geometric details are naturally integrated to generate a realistic appearance. The RTV and environment lighting are each represented by 16 spherical harmonic basis functions.

6. Conclusion

In this paper, we presented a technique for real-time rendering of non-height-field mesostructure and its detailed visual effects. With generalized displacement maps, we can render non-height-field mesostructure at a high frame rate with compelling visual effects. A noteworthy advantage of the GDM is that it is widely applicable, e.g., it handles open and closed surfaces equally well and is not sensitive to texture distortion and curvature variations. With the proposed approach of computing visibility jointly in texture and object spaces, both high efficiency and low texture distortion can be attained.

There exist a few directions for future work. One is to utilize texture synthesis rather than a tileable texture with GDM, to avoid pattern repetitions. A second direction is to develop a scheme for removing unneeded triangles from the mesh model to elevate computation speed. Third, an extension of our method to handle solid transparent mesostructure would provide greater generality.

Acknowledgements

We would like to thank the anonymous reviewers for their constructive critiques. Many thanks to Min Liu and Ying Song from Zhejiang University for their help in preparation of the input data and in implementation of the slice-based method, to Mingdong Xie for object mesh and geometry texture modeling. The authors from Tsinghua University were supported by Natural Science Foundation of China (Project Number: 60225016, 60321002) and the National Basic Research Project of China (Project Number 2002CB312100).

References

[AK87] ARVO J., KIRK D.: Fast ray tracing by ray classification. *Computer Graphics (SIGGRAPH '87 Proceedings) 21*, 4 (July 1987), 55–64.

Model	Triangle Number	Texture	Volumetric Resolution	Data Size		FPS
				GDM	RTV	
Teapot	4276	Weave	64× 64×32	2MB	8MB	56.0/16.3*
Vase	8640	Grid	32× 32×16	0.5MB	N/A	71.3
Torus	4880	Chain	64× 64×32	2MB	N/A	40.8
Bird	11603	Weave	64× 64×32	2MB	N/A	46.4
Dinosaur	2142	Squama	64× 64×32	2MB	N/A	80.9
Tree	19618	Bark	128×128×16	4MB	16MB	9.1*

Figure 7: *GDM rendering speed and memory consumption for different models. FPS values marked by a * indicate performance for global illumination.*

[Bli78] BLINN J. F.: Simulation of wrinkled surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings) 12*, 3 (1978), 286–292.

[BM93] BECKER B. G., MAX N. L.: Smooth transitions between bump rendering algorithms. *Computer Graphics (SIGGRAPH '93 Proceedings)* (1993), 183–190.

[CCC87] COOK R. L., CARPENTER L., CATMULL E.: The REYES image rendering architecture. *Computer Graphics (SIGGRAPH '87 Proceedings)* (1987), 95–102.

[CHH02] CARR N. A., HALL J. D., HART J. C.: The Ray Engine. *Proc. Graphics Hardware 2002* (2002).

[Coo84] COOK R. L.: Shade trees. *Computer Graphics (SIGGRAPH '84 Proceedings) 18*, 3 (1984), 223–231.

[Dis98] DISCHLER J.-M.: Efficient rendering macro geometric surface structures with bi-directional texture functions. In *Rendering Techniques* (1998), pp. 169–180.

[DKS*03] DAUBERT K., KAUTZ J., SEIDEL H.-P., HEIDRICH W., DISCHLER J.-M.: Efficient light transport using precomputed visibility. *IEEE Computer Graphics and Applications 23*, 3 (May/June 2003), 28–37.

[DNvGK99] DANA K. J., NAYAR S. K., VAN GINNEKEN B., KOENDERINK J. J.: Reflectance and texture of real-world surfaces. *ACM TOG 18*, 1 (Jan. 1999), 1–34.

[DS02] DAUBERT K., SEIDEL H.-P.: Hardware-based volumetric knit-wear. *Computer Graphics Forum 21*, 3 (Sept. 2002).

[HDKS00] HEIDRICH W., DAUBERT K., KAUTZ J., SEIDEL H.-P.: Illuminating micro geometry based on precomputed visibility. *Computer Graphics (SIGGRAPH '00 Proceedings)* (2000), 455–464.

[KK89] KAJIYA J. T., KAY T. L.: Rendering Fur with Three Dimensional Textures. *Computer Graphics 23*, 3 (1989), 271–280.

[KM99] KAUTZ J., MCCOOL M. D.: Interactive rendering with arbitrary BRDFs using separable approximations. In *Rendering Techniques* (1999), pp. 247–260.

[LDS02] LENSCH H. P. A., DAUBERT K., SEIDEL H.-P.: Interactive semi-transparent volumetric textures. *Proc. Vision, Modeling and Visualization* (2002), 505–512.

[LPFH01] LENGYEL J., PRAUN E., FINKELSTEIN A., HOPPE H.: Real-Time Fur over Arbitrary Surfaces. *Symposium on Interactive 3D Graphics* (2001), 227–232.

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

[MGW01] MALZBENDER T., GELB D., WOLTERS H.: Polynomial texture maps. *Computer Graphics (SIGGRAPH '01 Proceedings)* (August 2001).

[MN98] MEYER A., NEYRET F.: Interactive volumetric textures. *Eurographics Workshop on Rendering* (1998), 157–168.

[Ney98] NEYRET F.: Modeling, animating, and rendering complex scenes using volumetric textures. *IEEE TVCG 4*, 1 (Jan. 1998), 55–69.

[PBMH02] PURCELL T. J., BUCK I., MARK W. R., HANRAHAN P.: Ray tracing on programmable graphics hardware. *ACM TOG (SIGGRAPH 2002) 21*, 3 (July 2002), 703–712.

[SC00] SLOAN P.-P., COHEN M. F.: Interactive horizon mapping. *Eurographics Workshop on Rendering* (June 2000), 281–286.

[SLSS03] SLOAN P.-P., LIU X., SHUM H.-Y., SNYDER J.: Bi-scale radiance transfer. *Proceedings of SIGGRAPH 2003* (August 2003), 370–381.

[WL01] WEI L.-Y., LEVOY M.: Texture synthesis over arbitrary manifold surfaces. *Proceedings of SIGGRAPH 2001* (August 2001), 355–360.

[WWT*03] WANG L., WANG X., TONG X., HU S., GUO B., SHUM H.-Y.: View-dependent displacement mapping. *Proceedings of SIGGRAPH 2003 22* (July 2003), 334–339.