Accurate and Efficient Rendering of Detail using Directional Distance Maps

Ravish Mehra*
Department of Computer Science
Indian Institute of Technology, Delhi
ravish@cse.iitd.ac.in

Subodh Kumar
Department of Computer Science
Indian Institute of Technology, Delhi
subodh@cse.iitd.ac.in

ABSTRACT

We present an efficient GPU technique for rendering rich geometric detail (e.g., surface mesostructure) of complex surfaces. We use sphere-tracing aided by directional distance maps (DDMs) to quickly find ray-mesostructure intersection. High accuracy is achieved by analytically detecting such intersections, and space requirement is significantly reduced by distance map compression. Our technique can handle complex scenes containing both height-field and non height-field mesostructures in real time, with correct self-occlusion, self-shadow, interpenetrations and silhouettes. We demonstrate our algorithm on a variety of test scenarios and compare it with previous techniques.

Keywords

mesostructure, displacement mapping, GPU

1. INTRODUCTION

Surface details can be classified into microstructures, mesostructures, and macrostructures. Microstructures are minute details, and can be satisfactorily approximated by bump maps [3], or normal maps [4], BRDFs [2] or even standard texture maps. Displacement maps [5] are height-fields and are traditionally used for large-scale macrostructures. Mesostructures are fine-scale geometric details on the surface, and are larger than microstructures but smaller than macrostructures. They are often non height-field and can significantly alter shadows, occlusion, and silhouettes. One way to render such details is to tessellate the surface and modify the geometry to include these details. This tends to be rather inefficient for interactive display due to the large number of micro-polygons generated. It is worth noting that tessellation capability is available on recent GPUs [8]. However, the performance for high tessellation levels remains a concern. Moreover, tessellation may not be suitable for all workflows. Hence, *per-pixel* displacement mapping remains a useful alternative for some domains and provides the choice of a different performance-quality trade-off. Broadly, this technique generates fragments by rendering an extremely low-resolution mesh, sometimes a single quadrilateral, instead of the detailed geometry. The goal is to cover all fragments occupied by the geometry. For each resulting fragment, its color and depth is re-computed to match the appropriate point on the actual geometry. This is done by intersecting the ray through that fragment (starting from the eye) with the detailed geometry. The technique works whether the displacements are along the surface-normal (height-field) or not (non height-field).

Height-field mesostructure rendering is achieved by using height-maps stored as 2D texture-maps [22]. Since texture-maps only store samples of the displacements, aliasing is an important concern. At the same time, for far-away views, lower resolution textures may suffice [17, 29], affording more efficient display. Non height-field mesostructures, in the general case, require 3D textures [30, 6] to store, and can become memory limited. Still, the richness and generality of this representation makes it attractive in many cases. In this paper, we address the issues of memory-efficiency and rendering accuracy while maintaining a high performance level. We consider both height-field and non height-field mesostructures, and more generally, any model represented as a distance-field [9, 10].

Broadly, GPU based ray-object intersection algorithms employ two strategies. One attempts to bracket the intersection between two ray points and then searches for the root within this interval [22, 23, 28]. This process of bracketing intersection depends on non-robust and inefficient linear stepping, which introduces various artifacts. The other strategy is based on space skipping [6, 1, 21]. It uses pre-computed data structures to store *'safe regions'* of space, where no surface may lie. The search of the intersection then proceeds by marching along the pixel's ray, advancing past safe regions. In contrast with bracketing, the space skipping algorithms converge slowly in close proximity to the surface and may not find the exact intersection. We investigate the second strategy: space skipping. Sphere-tracing [6] is a space skipping technique to skip past areas of no intersection. It starts by computing a safe radius on a 3D regular grid. A sphere of this radius is guaranteed to not intersect the surface. The main drawbacks of this technique is slow convergence in regions close to the surface and high memory requirement for the 3D grid, combined with significant aliasing due to limited grid resolution (see Figure 2(a)).

In this paper, we address these drawbacks. In particular, we propose direction-specific distance maps taking inspiration from the work of Wang et al. [30], but devise a more hardware-friendly organization. This allows us to converge faster in close proximity to the surface. As demonstrated in our experiments, these directions need not be sampled densely in our scheme. We also propose a conservative sampling strategy for the 3D regular grid followed by C^0 surface construction for accurate ray intersection. This results in sig-

*Corresponding author

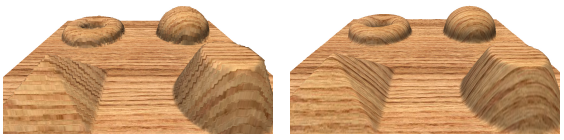
The Eighth Indian Conference on Vision, Graphics and Image Processing, Dec 2012, Mumbai, India.

Figure 1: (Left) Maze scene with a non height-field mesostructure (small golden ball in the upper corner) moving in a height map. This scene was rendered at 156 FPS with directional distance maps (DDMs) of resolution $128^2 \times 32$ along 8 direction cones and quadric approximation. (Right) A beaded teapot scene consisting 4900 golden beads (non height-field mesostructures) rendered with DDMs at resolution $16^2 \times 16$ along 4 direction cones and quadric approximation.

nificant quality improvement (see Figure 2(b)). We use a lossless distance map compression scheme that achieves high compression without impacting the rendering quality. Importantly, the resulting texture can further use default hardware-compression and exhibits high coherence. Finally, we devise an efficient multi-resolution representation for these directional distance maps, further reducing the memory foot-print. The main contributions of this paper are:

- Direction dependent sphere-tracing for fast convergence
- Efficient quadric surface approximation and intersection that improves the rendering quality
- Lossless distance map compression that reduces the memory footprint

Section 2 discusses related work and reviews the sphere tracing algorithm. In Section 3, we present our main technique. We discuss compression scheme in Section 4. We present implementation details and results in Section 5, conclude in Section 6, and give acknowledgements in Section 7.



(a) Aliasing artifacts caused by discrete distance map (b) Sphere tracing + Quadric approximation

Figure 2: Sphere tracing on shapes a) with distance maps only b) with distance maps and quadric approximation. Note the absence of artifacts.

2. RELATED WORK

Recent years have witnessed significant research in mesostructure rendering and a number of techniques have been proposed to render both height-field and non height-field mesostructures. We first discuss the height-field case. Relief mapping [22] employs a two step intersection algorithm, which first performs a linear search to find a point inside the height-field, and then converges to the final intersection using a mid-point binary search. Other variants of this technique replace the binary search in the second step with iterations using tangents, secants [23], or piecewise linear curve approximation [28]. The overall accuracy and performance is sensitive to the step size chosen in the linear step. With smaller step size we get better rendering quality, but at the cost of reduced performance. With larger steps we miss features and compromise quality. Oh et al. [17] use maximum mipmapped height maps to move from cell to cell, skipping empty regions to reach the intersection point. Tevs et al. [29] further improve this approach by providing a distance-based LOD approach to truncate the height-field hierarchy. Tatarchuk [28] switches to a bump mapping if the distance between the intersection point and the viewer exceeds some threshold value. Cone-step mapping proposed by Dummer [7] uses a cone, as a measure of safe volume in order to find the ray height-field intersection. Policarpo et al. [21] combine cone stepping with relief mapping for accurately rendering height-fields. However, computing cone maps requires large pre-processing time and computing it for arbitrarily shaped mesostructure (especially non height-field) is hard. Kolb [14] uses erosion and dilation maps to bound the empty space for efficient intersection computation. Baboud et al. [1] compute a conservative 2D version of the safe radius for replacing the linear stepping of relief mapping. All these techniques are characterized by smaller pre-processing time and use of 2D textures, but are limited to only height-field rendering.

For non height-field case, Policarpo and Oliveira [20] extend traditional relief mapping by using multi-layered relief maps and perform the intersection computation on the GPU. However, aliasing artifact due to the linear search still persists. Generalized displacement map (GDM) [30] is a five dimensional displacement map that stores for each sample point (x, y, z), the minimum distance to the mesostructure in various sampled directions (θ, ϕ). This technique can be used to render a generic non height-field mesostructure at interactive rates. However, the costs include a large dataset (5D texture) that is compressed using a lossy compression scheme based on the singular value decomposition. The corresponding decom-

pression algorithm is not hardware friendly and suffers from incoherent memory accesses. Since this technique is based on linear interpolation of distances from sample points in 5D, it can cause severe artifacts in close-up views.

Ray-tracing approaches based on hierarchical data structures [15] can be used to perform ray-object intersection for mesostructure rendering. However, their computational complexity depends directly on the number of primitives (triangles, quads). The number of primitives required to faithfully capture all the mesostructure details is quite large, making the ray-tracing approaches prohibitive.

2.1 Sphere tracing

We now review the basic sphere tracing algorithm on which our technique is based. It was first proposed by Hart [11] for accelerating ray-surface intersection. The main idea is to advance along the ray past areas of non-intersection. Figure 3 explains the main steps of the algorithm.

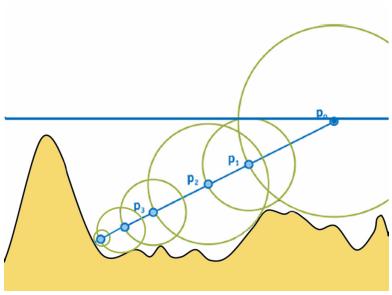


Figure 3: Sphere tracing algorithm steps. If any outside point $\mathbf{p}_i$ is at a distance d_i from the surface, then for any ray direction $\mathbf{r}$ we can proceed to $\mathbf{p}_i + d_i\mathbf{r}$ without missing any surface intersection. This process is repeated until we converge to the intersection point.

Donnelly [6] used this approach for rendering height-field mesostructures using a pre-processed data structure called distance map. It is a volumetric texture that for each 3D cell (also called voxel) stores distance to the closest surface-point from the center of the voxel. This distance function can also be used for non height-field mesostructures and more generally, any model represented as a distance-field [9, 10]. The ability to render all types of mesostructures, along with recent improvements in algorithms to efficiently compute distance-fields [27, 26] and increasing GPU memory, makes this method a promising rendering technique. But this technique has several drawbacks, specifically:

a) For points close to the surface, especially near silhouettes, the method progresses slowly due to small distance values (see Figure 4(a)).

b) The memory requirement for volumetric textures is high and sampling frequency must be carefully chosen to ensure high rendering quality.

c) Ray-marching is terminated once the distance (linearly interpolated from samples) reaches a small threshold. The resulting voxel, or the final intersection computed within this voxel can be quite inaccurate. This causes severe artifacts (see Figure 2(a)).

3. MAIN TECHNIQUE

We first address the issue of slow convergence and limited accuracy due to the sampling resolution. The convergence of standard sphere tracing algorithm is slow near the surface. An important reason for this is the maintenance of distance to a nearest surface-point even if the ray already passed by. A key observation here is that distances only in the direction of the ray are important or in other words *'look forward'*. This leads us to decompose the distance maps into direction specific maps called directional distance maps (**DDM**s). Each DDM limits consideration of the nearest surface-points to a direction cone (see Figure 4). As we will see later, a small number of directions suffice. Intuitively, we are mostly interested in distinguishing behind from in-front.

Recall that standard sphere-tracing only samples distances at the center of the voxel and terminates when the linearly interpolated value is *'close enough'* to the surface. This, in certain cases, causes convergence to the wrong voxel . Even if the chosen voxel is correct, the point of intersection can be erroneous. Note that, even if the surface varies bi-linearly, the distance-field need not. We use a *conservative* approach in which we store the distance to the nearest surface point from *anywhere* within a voxel. This eliminates the interpolation errors. A voxel containing the surface now stores a value of 0. We further improve the accuracy by storing a C^0 quadric approximation of the actual surface passing through such voxels.

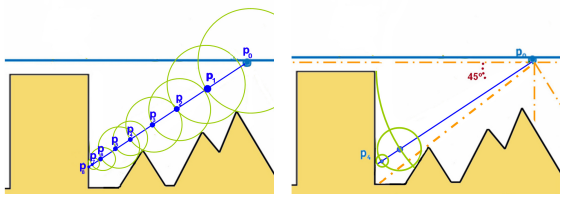
The online algorithm performs *directional* sphere tracing until directional distance values becomes zero (*indicating possible intersection*). The actual intersection is computed by analytically performing ray-quadric intersection. If the ray hits the quadric, we have found the point of intersection. Otherwise, we advance to the boundary of the voxel and continue marching from there. Finally the fragment is shaded by the color of the point of intersection. Shadows can be further computed by tracing a ray from the point of intersection to the light. We next describe each step in detail.

3.1 Pre-processing

We first describe how to construct the data structure that helps us determine the safe radius at any point in 3D space. Conservative DDMs are created in the pre-processing phase.

Directional Distance Map.

A Directional Distance Map (DDM) is computed on a voxel grid in texture space. Each voxel V stores the minimum distance from V to the nearest point of the mesostructure in a given viewing direction-cone. It is the similar to the original distance map defined by Donnelly [6] but computed directionally and conservatively.



(a) Distance map (b) DDM

Figure 4: Comparison between sphere tracing using distance map and using DDMs. Note faster convergence near mesostructure boundary for DDMs.

We divide the spherical angles (θ, ϕ) covering the entire space into intervals forming directional cones. θ varies horizontally from 0 to 360 degrees and ϕ varies vertically from -90 to $+90$ degrees. Viewing rays require ϕ lying between 0 to -90 whereas shadow rays (which travel upwards) require 0 to $+90$. The distance map is

calculated along each of these directional cones separately and conservatively. For mesostructure M, voxel V in the texture space, and directional cone interval $[\theta_1, \theta_2] \times [\phi_1, \phi_2]$, the DDM is formally defined as:

$$\text{DDM}(V, [\theta_1, \theta_2] \times [\phi_1, \phi_2]) = \min\{dist(\mathbf{m}, \mathbf{v}) \quad \forall \mathbf{m} \in \mathbf{M}, \text{ any } \mathbf{v} \in \mathbf{V} \text{ st. } (\mathbf{m} - \mathbf{v}) \text{ lies inside}([\theta_1, \theta_2] \times [\phi_1, \phi_2])\}.$$

The notation $dist(\mathbf{m}, \mathbf{v})$ is the distance between a point $\mathbf{m}$ on mesostructure M and a point $\mathbf{v}$ in the voxel V. Note that if any point of the mesostructure lies inside V, the DDM value becomes zero. The distance computation is only a pre-processing step and there exists algorithms to perform this computation at near-interactive rates [26]. DDMs are stored as 3D textures and exhibit high coherence (neighboring pixels tend to be in the same direction cone). The texture corresponding to each direction is compressed as described in Section 4. In all the tables, notation DDM $m \times n$ denotes that θ is divided into m intervals and ϕ into n intervals. Therefore, there are mn direction cones and mn DDMs.

Quadric Surface.

Piecewise C^0 approximation for height-field is straightforward. Take four corners of the height-field on a grid and construct a bilinear patch passing through them. But non height-field mesostructure could be any general 3D shape. Piecewise quadratic approximation of point cloud or 3D mesh has been researched extensively in different contexts [19, 18, 25]. Note that we need the quadric construction only for voxels that have the mesostructure passing through them, i.e., the DDM value is zero. We have devised a simple strategy for this by assuming a simple bilinear shape within each voxel which satisfies C^0 quadric approximation.

a) For a voxel V that has mesostructure passing through it (DDM value 0), we find parts of the mesostructure that lie inside or in the neighborhood of V. We sample it densely and then fit the quadric patch Q_V to these sampled points $\{(x_i, y_i, z_i)\}_{i=1}^n$.

$$Q_V \begin{cases} Ax^2 + By^2 + Cz^2 + Dxy + Eyz \\ \qquad +Fzx + Gx + Hy + Iz + J = 0 \end{cases} \tag{1}$$

Least square fitting method [16] is used to solve for the parameters of the quadric by solving the following linear system:

$$\begin{bmatrix} x_1^2 & y_1^2 & z_1^2 & \dots & 1 \\ x_2^2 & y_2^2 & z_2^2 & \dots & 1 \\ . & \dots & . & & \\ . & \dots & . & & \\ x_n^2 & y_n^2 & z_n^2 & \dots & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ . \\ . \\ J \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ 0 \end{bmatrix} \tag{2}$$

We next find the intersection of the quadric patch Q_V with the edges of voxel V. For an edge e, neighboring voxels sharing e may hit it at slightly different positions. We store these positions in a list $L(e)$ for edge e.

b) For each edge e_i with intersection point list $L(e_i)$, we compute the average of the points in $L(e_i)$. We then map it back to the voxels sharing edge e_i and use it as the representative intersection point.

c) Each voxel V now contains a set of representative intersection points P_V (ordered by traversing its faces). If size$(P_V) = 3$, we fit a triangular patch passing through these points. If size$(P_V) = 4$, we form a bilinear patch passing through four points. In both the cases, the intersection of surface with each face is the straight line formed by two representative points. Since the same representative points are used on the other side of the boundary, the surface is C^0. If the voxel resolution is low, it is possible that size$(P_V) > 4$ for many voxels, meaning the surface inside the voxel is not bilinear. If the number of such voxels is high, we compute the DDM at higher voxel resolution. If not, we approximate the surface with more than one bilinear (or triangular) patches. There is an interesting trade-off between using a higher resolution DDM versus allowing the more complex patches (requiring longer shaders).

Both triangular and bilinear patches are converted in quadrics as described in Appendix B and their coefficients are stored in textures (see section 4). This bilinear patch can also be converted in quadric as shown in Stoll et al. [25].

3.2 Runtime

We start by first rendering a coarse resolution mesh to cover the fragments of the detailed mesostructure. For each fragment f with a texture coordinate $t = (t_x, t_y, t_z)$, we trace a ray in the viewing direction $\mathbf{r} = (\mathbf{r_x}, \mathbf{r_y}, \mathbf{r_z})$ as follows:

Algorithm 3.1 Sphere Tracing using DDM + Quadric Intersection

```
while ray_outside_surface do
    d = ACCESS_DDM(dir_cone(r, t))
    if d = 0 then
        if QUADRIC_INTERSECTION(r, t) = TRUE then
            exit
        else
            VOXEL_WALL_INTERSECTION(r, t)
    else
        t = t + dr
Use t for accessing the color texture, normal texture etc
```

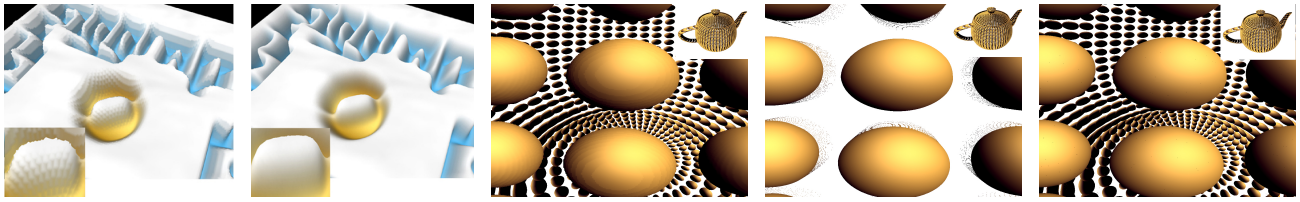
Algorithm 3.2 *QUADRIC_INTERSECTION(*$\mathbf{r}$*, t)*

We substitute the ray equation $t + \alpha\mathbf{r}$ in the quadric equation Q_V and solve the roots of the equation for the smallest positive value, α_+. If α_+ exists we advance the ray to $t + \alpha_+\mathbf{r}$ and return TRUE. Else we return FALSE.

Algorithm 3.3 *VOXEL_WALL_INTERSECTION(*$\mathbf{r}$*, t)*: Ray-cell intersection routine for height-fields given in Ki et al. [13] was extended to 3D for DDM of resolution (L_x x L_y x L_z). Here ϵ is a small constant.

$s = floor(t_x L_x,\ t_y L_y,\ t_z L_z)$
$u = (sign(\mathbf{r}) + 1.0)/2$
$w = ((s_x + u_x)/L_x,\ (s_y + u_y)/L_y,\ (s_z + u_z)/L_z)$
$d = min((w_x - t_x)/r_x,\ (w_y - t_y)/r_y,\ (w_z - t_z)/r_z)$
$t = t + (d + \epsilon)\mathbf{r}$

In the sphere tracing step, we determine the direction cone in which the viewing ray direction $\mathbf{r}$ belongs. We access the corresponding DDM to find the safe radius d in the viewing direction cone. If this safe radius is non-zero, we advance by d in the direction $\mathbf{r}$. Otherwise, we perform the analytical ray-quadric intersection. If an intersection is found, we use this intersection point for color shading. Otherwise, we must continue marching. We re-start ray-marching starting at the point the ray exits the voxel. Note that



(a) Sphere tracing (b) Quadric approx. (c) Sphere tracing (d) Relief mapping (e) Quadric approx.

Figure 5: (Left) Comparison between quality achieved by sphere tracing with distance maps ($128^2 \times 32$) (Donnelly [6]) and our quadric approximation ($128^2 \times 1$) for the maze scene. (Right) Comparison between quality achieved by sphere tracing with distance map ($128^2 \times 32$) (Donnelly [6]), relief mapping (256×256) (Policarpo [20]) and our quadric approximation ($16^2 \times 16$) for the teapot scene. Note the artifacts due to linear stepping in relief mapping and limited grid resolution in sphere tracing. Closeups shown.

the original sphere tracing approach using distance maps [6] would exit once $d = 0$ (or a small constant), making the accuracy of the result strongly dependent upon the resolution of the distance map texture.

4. DISTANCE MAP COMPRESSION

Large memory requirement is a major shortcoming of non height-field mesostructure rendering techniques due to the requirement of 3D textures [30, 6]. To counter it, compression schemes have been proposed that are lossy, thus reducing quality, and induce incoherent memory accesses, thus reducing efficiency. We propose a new scheme for distance map compression addressing these drawbacks that performs lossless compression. Our compression scheme is conservative. Note that reducing the resolution of our DDM texture only means that each sample stores the nearest surface distance from a larger region of space. We reduce the resolution only where the variance of this distance across an entire region is low, generating in effect a multi-resolution texture.

Our compression scheme is illustrated in Figure 6. The 3D DDM is subdivided into blocks. The blocks that have a mesostructure surface passing through them (DDM value 0) are packed into a 3D texture atlas. The location of this block in the atlas is stored in a lookup grid at a coarse resolution. Both the atlas and the grid are in the same texture space as the distance map. For blocks of the DDM that do not have a mesostructure surface passing through them, we only store the minimum of distance values of the block into the lookup grid. This block itself is not stored in the atlas. One can extend this idea hierarchically to create multiple levels. In this paper, we limit the hierarchy to two levels.

It is possible to think of the quadric coefficient texture as another level in the DDM hierarchy, even though it stores coefficients and not distances. If a surface passes through a DDM block, a detailed distance sampling is available in the voxels of the atlas block as described before. If the ray reaches a voxel in the atlas with distance value of 0, a further look-up into a third quadric coefficient texture is performed to obtain the coefficients. This quadric coefficient texture can also be stored in an atlas in a similar manner as before. Note that quadric texture is not per-direction. The same coefficients are used for every direction.

Location of blocks in the texture atlas can be chosen by a variety of previously proposed techniques of packing texture atlases to achieve maximum compression [24, 19]. We follow the scheme of [19]. This algorithm, although designed for 2D texture atlases, extends easily to 3D. More details about the distance map compression and access can be found in Appendix A.

Recall that the final ray-mesostructure intersection is not affected by the grid-based compression. On the other hand, the lower resolution implies smaller stored distances and that can reduce the step size of sphere tracing. But only the voxels away from the mesostructure surface detail are compressed. Since most of the sphere-tracing time is spent near the surface, the convergence in this region is not slowed significantly by compression. Overall, the final speeds are comparable to the case without compression, and is never more than 5% slower. This is a good tradeoff for the high compression achieved. The results of the compression scheme are specified in Table 3. We have shown that our lossless 3D texture compression achieves a compression efficiency of 4 times to 256 times depending on the mesostructure. This is significant savings in runtime memory.

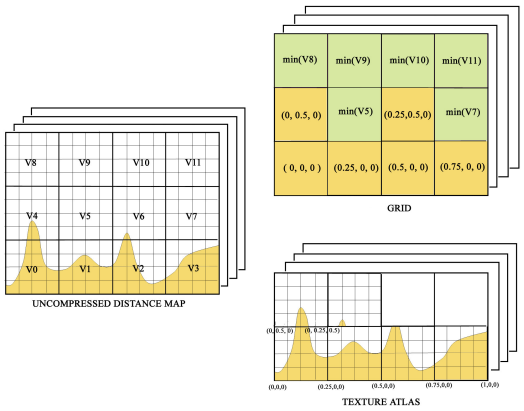
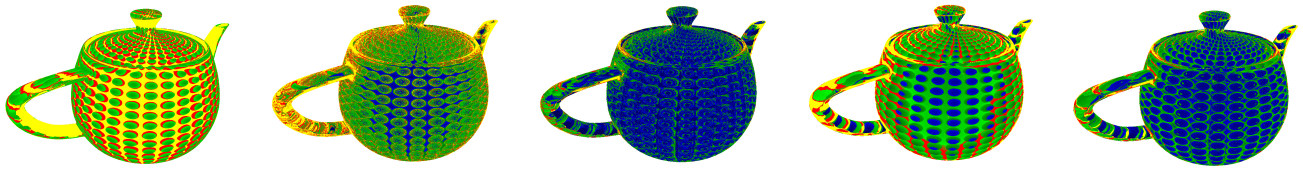


Figure 6: DDM compression using a lookup grid and texture atlas: For voxels V5, V7, V8, V9, V10, V11 of uncompressed DDM that do not contain mesostructure surface, we store the minimum value in the lookup grid $GRID$. For other voxels containing mesostructure surface (V0, V1, ...), we store the entire distance map block in the texture atlas $ATLAS$ and store its corresponding locations ($(0,0,0), (0.25,0,0), ...$) in the lookup grid $GRID$.

5. IMPLEMENTATION AND RESULTS

The technique proposed in the paper has been implemented as a Shader Model 4.0 fragment program. All the timings and comparisons given in the paper and the supplementary video were produced on Intel PentiumD 3.4GHz machine with 2 GB RAM and Geforce 8800 GTS GPU.

Our input to the DDM pre-computation pipeline is mesostructure M (one or many) in the form of generic triangular mesh. For any



(a) Relief mapping (b) Sphere tracing (c) Only DDM (d) Only quadric inter. (e) DDM+quadric

Figure 7: Comparing number of iterations. Color coding of iterations is: blue(0-10), green(10-34), red(35-59), and yellow(60-100).

other type of mesostructure, e.g., height-fields, we convert them into a triangular mesh format for ease of implementation. Our pipeline then computes a 3D directional distance field from this mesh. We extend HAVOC3D [12], incorporating the directionality and conservative constraints as follows:

1) Directionality constraint For each voxel V, we query triangles only if any of the direction vectors $(u_i - v_i)$ lies inside directional-cone interval $[\theta_1, \theta_2] \times [\phi_1, \phi_2]$ where u_1, u_2, u_3 are the vertices of the triangle and v_i are points in the voxel V. We provide these cone triangles to HAVOC3D for calculating distance map value at voxel V.

2) Conservative constraint In our current implementation, we use an ad-hoc method to compute conservative distance maps. We compute DDM at a very high resolution, with voxel-size comparable to the maximum curvature of the surface. To compute conservative DDM at the desired resolution, we simply apply the minimum filter. In other words, the sample at coarse level is the minimum of the samples at the finer level.

We now discuss some of the main advantages of our approach:
Quality improvement: Figure 5 compares the rendering quality of sphere tracing, relief mapping and quadric approximation. Quadric surface approximation results in a smooth appearance compared to linear stepping artifacts of relief mapping and limited resolution artifacts of sphere tracing with distance maps. This eliminates the need for approximate techniques like bilinear interpolation for magnification [17]. Note that the spherical ball used in various scenes is inherently a quadric polynomial which is accurately modeled with our technique.
Faster convergence and better silhouettes: Near the boundary of the mesostructures, DDM, by virtue of storing larger safe radii, results in faster convergence. Refer to Figure 4 and Table 1.
Only a few direction cones can produce sufficient improvement: It is worth noting that increasing the number of directions sampled have diminishing returns. Table 1 shows that only a few direction cones can increase the performance significantly.

	Distance map	DDM 2×1	DDM 4×1	DDM 8×1
Shape	5.5	4.2	4	3.9
Teapot	4.4	3.1	2.8	2.7
Golden ball	8.3	7.1	6.1	5.8

Table 1: Performance (in msec) of sphere tracing with distance map and DDM at varying number of directions for shape(figure 2(b)), golden ball of maze (figure 1 left) and teapot (figure 1 right). Distance map and DDM resolution is $128^2 \times 32$.

Model (No. of beads)	Relief map	ST Distance map	ST DDM 4×1	ST DDM + Quadric 4×1
Plane(400)	21.7	8.3	3.0	7.4
Cylinder(1600)	38.5	17.5	3.9	16.4
Teapot(4900)	23.8	25.0	10	50
Box(196)	27.7	12.7	6.3	30.3

Table 2: Performance (in msec) for relief mapping (256×256), sphere tracing (ST) with distance maps ($128^2 \times 32$), ST with DDM ($128^2 \times 32$ with 4×1 direction cones compressed to $\frac{1}{4}$th original size), and ST with compressed DDM with quadric approximation at $16^2 \times 16$. The rendering quality of these techniques vary as: ST with distance map < ST with DDM < relief mapping < ST with DDM + Quadric. These models are used in the Gallery scene (Figure 8(b)).

Coherent access: During directional sphere tracing, a ray only needs to access a single direction cone's DDM for the entire tracing algorithm. Moreover, neighboring pixels are likely to access the same texture.

In Figure 7, we provide a visual comparison between the number of iterations required to converge for relief mapping [20], sphere tracing with distance map [6], sphere tracing using DDM only, quadric intersections only and our complete approach (sphere tracing using DDM + quadric intersections). Only quadric intersection is performed by computing ray-quadric intersection for each voxel and moving from voxel to voxel until ray intersects the mesostructure. We also compare the performance achieved by these various techniques in Table 2. We want to emphasize that even with a higher resolution of relief texture (256×256), the result given by the quadric approximation ($16 \times 16 \times 16$) is better in quality than relief mapping. As mentioned, the quality changes as follows: sphere tracing with distance map < sphere tracing with DDM < relief mapping < sphere tracing with DDM + Quadric. The performance varies as: relief mapping < sphere tracing with distance map < sphere tracing with DDM. Quadric approximation improves the quality of the results significantly but incurs an additional cost. This causes the proposed technique to be slower in some scenarios.

Table 3 shows the compression efficiency achieved by our lossless DDM compression scheme. The compression becomes important for non height-field mesostructures since they require a 3D data structure to store the required spatial information. Note that compression efficiency of our scheme varies with the actual detail (rather free space) present in the mesostructure. If the mesostructure is locally present spanning a small region, compression efficiency is quite high (256 times for maze golden ball). Otherwise if the mesostructures spans the 3D space, the compression efficiency is less. The total texture memory used by our technique is small: between 250 and 400 KB for the examples used in this paper (not

	Uncompr.	Compressed								
		Beads			Kitten			Golden Ball		
		LOOKUP	ATLAS	Compr. Effic.	LOOKUP	ATLAS	Compr. Effic.	LOOKUP	ATLAS	Compr. Effic.
Distance map	$256^2 \times 64$	$128^2 \times 32$	$128^2 \times 32$	4	$64^2 \times 16$	$64^2 \times 64$	12.8	$32^2 \times 8$	$16^2 \times 32$	256
DDM 8 dirs	$(256^2 \times 64) \times 8$	$128^2 \times 256$	$128^2 \times 256$	4	$64^2 \times 128$	$64^2 \times 512$	12.8	$32^2 \times 64$	$16^2 \times 256$	256

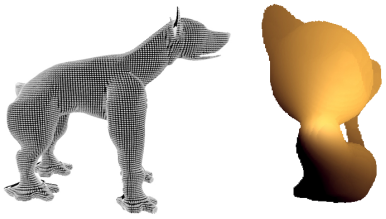
Table 3: Compression efficiency of distance map and DDM. Beads and kitten use DDMs of resolution $256^2 \times 64$ with 4×2 directional cones. Golden ball is calculated with directional cones 8×1. Compression efficiency (Compr. Effic.) is calculated as $\frac{\text{UNCOMPRESSED TEXTURE SIZE}}{\text{LOOKUP SIZE}+\text{ATLAS SIZE}}$. As Golden ball is much smaller in size, it has a much higher compression efficiency.

counting the ball, which is much smaller). In a large application, the saved memory could be used elsewhere. Also, in memory limited scenario, performance with compression will be much better. Therefore, compared to other displacement mapping techniques, our method has a better chance of handling large-scale and highly complex mesostructures.

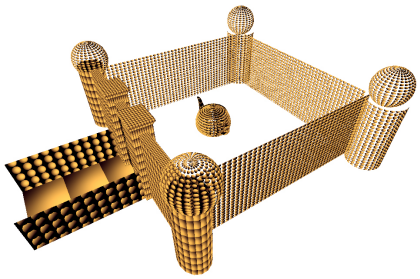
6. CONCLUSION

We present a memory efficient and accurate technique for rendering both height-field and non height-field mesostructures that can achieve higher quality results with less aliasing compared to other techniques. Our compression algorithm considerably reduces the overall memory requirement. Our technique has clear advantages over prior work and improves the state of the art in displacement mapping for non-height field mesostructures.

In future, we would like to extend our surface approximation algorithm for non-parametric mesostructure surfaces. We would also like to explore more efficient methods of arranging voxels in the texture atlas. Another possibility is to investigate the applicability of distance based LOD [29, 28] in our approach. A procedure of analyzing the mesostructure data and intelligently deciding which directions to choose for calculating the DDMs is an interesting problem. Finally, this technique can be gainfully combined with CUDA and DX11 tessellation to obtain efficient algorithms with consistently good quality.



(a) Dog and Kitten



(b) Gallery

Figure 8: Non height-field mesostructures: beads mesostructure applied on the dog and gallery scene and kitten mesostructure applied to a flat wall.

7. ACKNOWLEDGMENTS

This research is supported in part by grants from DST (Department of Science and Technology) and NRB (Naval Research Board). We thank Shaktisingh Shekhawat for helping with the textures, Pushkar Tripathi for proof-reading the paper, and Mr. K.R. Kaushik for his help with the infrastructure of the graphics lab. We thank the anonymous reviewers for their helpful suggestions.

8. REFERENCES

[1] L. Baboud and X. Décoret. Rendering geometry with relief textures. In *GI '06*, pages 195–201, Toronto, Ont., Canada, Canada, 2006.

[2] A. Ben-Artzi, K. Egan, F. Durand, and R. Ramamoorthi. A precomputed polynomial representation for interactive brdf editing with global illumination. *ACM Trans. Graph.*, 27(2):13:1–13:13, May 2008.

[3] J. F. Blinn. Simulation of wrinkled surfaces. *SIGGRAPH Comput. Graph.*, 12(3):286–292, 1978.

[4] P. Cignoni, C. Montani, R. Scopigno, and C. Rocchini. A general method for preserving attribute values on simplified meshes. In *VIS '98*, pages 59–66, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[5] R. L. Cook. Shade trees. *SIGGRAPH Comput. Graph.*, 18(3):223–231, 1984.

[6] W. Donnelly. Per-pixel displacement mapping with distance functions. *GPU Gems 2*, 22(3):123–136, 2005.

[7] Dummer. Cone step mapping: An iterative ray-heightfield intersection algorithm. available online at http://lonesock.net/files/conestepmapping.pdf. pages 55–62, New York, NY, USA, 2006. ACM.

[8] http://www.nvidia.com/object/tessellation.html.

[9] S. F. F. Gibson. Using distance maps for accurate surface representation in sampled volumes. In *VVS '98*, pages 23–30, New York, NY, USA, 1998. ACM.

[10] S. F. F. Gibson, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *SIGGRAPH*, pages 249–254, 2000.

[11] J. C. Hart. Sphere tracing: A geometric method for the

antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[12] I. Hoff, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized voronoi diagrams using graphics hardware. In *SIGGRAPH '99*, pages 277–286, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[13] H. Ki and K. Oh. Accurate per-pixel displacement mapping using a pyramid structure - http://ki-h.com/article/ipdm.html. pages 55–62, New York, NY, USA, 2007. ACM.

[14] A. Kolb and C. Rezk-Salama. Efficient empty space skipping for per-pixel displacement mapping. In *Proc. Vision, Modeling and Visualization.*, 2005.

[15] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast bvh construction on gpus. *Computer Graphics Forum*, 28(2):375–384, 2009.

[16] X. Li. Geometric property estimation from 3d range data points aided by local quadric surface fitting. In *CAD-CG '05*, pages 313–318, Washington, DC, USA, 2005. IEEE Computer Society.

[17] K. Oh, H. Ki, and C.-H. Lee. Pyramidal displacement mapping: a gpu based artifacts-free ray tracing through an image pyramid. In *VRST '06*, pages 75–82, 2006.

[18] Y. Ohtake, A. Belyaev, and M. Alexa. Sparse low-degree implicit surfaces with applications to high quality rendering, feature extraction, and smoothing. In *SGP '05*, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[19] S. Petitjean. A survey of methods for recovering quadrics in triangle meshes. *ACM Comput. Surv.*, 34(2):211–262, 2002.

[20] F. Policarpo and M. M. Oliveira. Relief mapping of non-height-field surface details. In *I3D '06*, pages 55–62, New York, NY, USA, 2006. ACM.

[21] F. Policarpo and M. M. Oliveira. Relaxed cone stepping for relief mapping. *GPU Gems 3. Addison-Wesley Professional,*, 18:409–428, 2007.

[22] F. Policarpo, M. M. Oliveira, and J. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *I3D '05*, pages 155–162, New York, NY, USA, 2005. ACM.

[23] E. A. Risser, M. A. Shah, and S. Pattanaik. Interval mapping. Technical report, University of Central Florida, 2005.

[24] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. Texture mapping progressive meshes. In *SIGGRAPH '01*, pages 409–416, New York, NY, USA, 2001. ACM.

[25] C. Stoll, S. Gumhold, and H.-P. Seidel. Incremental raycasting of piecewise quadratic surfaces on the gpu. *Interactive Ray Tracing 2006*, pages 141–150, Sept. 2006.

[26] A. Sud, N. Govindaraju, R. Gayle, and D. Manocha. Interactive 3d distance field computation using linear factorization. In *I3D '06*, pages 117–124, New York, NY, USA, 2006. ACM.

[27] A. Sud, M. A. Otaduy, and D. Manocha. Difi: Fast 3d distance field computation using graphics hardware. *Comput. Graph. Forum*, 23(3):557–566, 2004.

[28] N. Tatarchuk. Dynamic parallax occlusion mapping with approximate soft shadows. In *I3D '06*, pages 63–69, 2006.

[29] A. Tevs, I. Ihrke, and H.-P. Seidel. Maximum mipmaps for fast, accurate, and scalable dynamic height field rendering. In *I3D '08*, pages 183–190, 2008.

[30] X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum. Generalized displacement maps. *Computer Graphics Forum*, 22(3):334–339, 2004.

APPENDIX

A. DISTANCE MAP COMPRESSION

Algorithm A.1 *COMPRESSION*

Let D be the uncompressed distance map

for Voxel $V_i \in$ *GRID* **do**
  $D_i = \{\ d\ |d \in D$ and d lies inside $V_i\ \}$
  if *surface in*(V_i) **then**
    ATLAS$(x_i, y_i, z_i) = D_i$
    $V_i =< (x_i, y_i, z_i), TRUE >$
  else
    $V_i =< m, FALSE >$ where $m = \min(D_i)$

Algorithm A.2 *COMPRESSED_MAP_ACCESS*(t_x, t_y, t_z)

$< d, b, scale >=$ *GRID*(t_x, t_y, t_z)
if b $= FALSE$ **then**
  $d_{lookup} = d$
  $\mathbf{p} + d_{lookup}\mathbf{r}$
else
  d is a location triplet (o_x, o_y, o_z)
  $(x, y, z) = ((t_x, t_y, t_z) - corner(t_x, t_y, t_z)) * scale$
  $d_{atlas} =$ *ATLAS*$(o_x + x, o_y + y, o_z + z)$
  $\mathbf{p} + d_{atlas}\mathbf{r}$

We first access *GRID* at texture coordinate (t_x, t_y, t_z) to check whether the mesostructure surface passes through the voxel, stored as a $boolean$ flag. If the flag is $FALSE$, we perform sphere-tracing using the distance map value d_{lookup} given by *GRID*. Otherwise, we interpret the *GRID* value as the location in *ATLAS*. We now transform the texture coordinate with the accessed *scale* and its *offset* from the *GRID* voxel's corner (which is closest to the origin). This brings it into the atlas's space and now a simple offset by the accessed location provides the final *ATLAS* texture coordinate. This coordinate is used to access the distance d_{atlas} and perform sphere tracing. We store the location in the first three components and the scale in the last component of *GRID*. The boolean value is also encoded in the last component. Note that a scalar scale value implies a uniform scale in each dimension. If all blocks use the same resolution, scale can be a shader constant.

B. QUADRIC APPROXIMATION

Bilinear patch formed by four points $\mathbf{P_1}, \mathbf{P_2}, \mathbf{P_3}, \mathbf{P_4}$ taken in order is:

$$(1-u)(1-v)\mathbf{P_1} + u(1-v)\mathbf{P_2} + uv\mathbf{P_3} + (1-u)v\mathbf{P_4}$$

Substituting

$$\begin{aligned}(a_1, a_2, a_3) &= \mathbf{P_1} + \mathbf{P_3} - \mathbf{P_2} - \mathbf{P_4}, \quad (d_1, d_2, d_3) = \mathbf{P_1} \\ (b_1, b_2, b_3) &= \mathbf{P_2} - \mathbf{P_1}, \quad (c_1, c_2, c_3) = \mathbf{P_4} - \mathbf{P_1}\end{aligned}$$

and equating the x, y and z coordinates, we get

$$\begin{aligned}X &= uva_1 + ub_1 + vc_1 + d_1 \\ Y &= uva_2 + ub_2 + vc_2 + d_2 \\ Z &= uva_3 + ub_3 + vc_3 + d_3\end{aligned}$$

Eliminating u and v gives us the quadric equation and its coefficients. Similarly, we can solve for quadric coefficients of a triangular patch defined by three points taken in order $\mathbf{P_1}, \mathbf{P_2}, \mathbf{P_3}$:

$$u\mathbf{P_1} + v\mathbf{P_2} + (1-u-v)\mathbf{P_3}$$