Fast Visualization of Complex 3D Models Using Displacement Mapping

The-Kiet Lu*
Nanyang Technological University

Kok-Lim Low†
National University of Singapore

Jianmin Zheng‡
Nanyang Technological University

ABSTRACT

We present a simple method to render complex 3D models at interactive rates using real-time displacement mapping. We use an octree to decompose the 3D model into a set of height fields and display the model by rendering the height fields using per-pixel displacement mapping. By simply rendering the faces of the octree voxels to produce fragments for ray-casting on the GPU, and with straightforward transformation of view rays to the displacement map's local space, our method is able to accurately render the object's silhouettes with very little special handling. The algorithm is especially suitable for fast visualization of high-detail point-based models, and models made up of unprocessed triangle meshes that come straight from range scanning. This is because our method requires much less preprocessing time compared to the traditional triangle-based rendering approach, which usually needs a large amount of computation to preprocess the input model into one that can be rendered more efficiently. Unlike the point-based rendering approach, the rendering efficiency of our method is not limited by the number of input points. Our method can achieve interactive rendering of models with more than 300 millions points on standard graphics hardware.

Index Terms: I.3.3 [COMPUTER GRAPHICS]: Picture/Image Generation—Viewing algorithms;

1 INTRODUCTION

Interactive visualization of high-detail 3D models has always been one of the main research focuses in computer graphics. Range scanning has become a common means to model real-world objects, and today's advancement in the range scanning technology has enabled us to even record fine brush strokes on paintings as well as chisel marks on sculptures [18]. This has produced very detailed and complex 3D models that can hardly be displayed at interactive frame rates. Earlier research has mostly focused on mesh simplification algorithms to reduce the number of vertices and triangles to be processed at the rendering stage [9]. However, as these methods require too much preprocessing time [30], more recently, researchers have become interested in using the input data directly for rendering by using points as rendering primitives. While these point-based rendering techniques can reach interactive frame rates for certain 3D models, the main bottleneck still lies at the vertex processing stage, such that this approach may not cope well with the rapid growth of data size in the near future.

In this paper, we propose an alternative solution that requires much less preprocessing time than the triangle mesh-based approach and its rendering speed is not limited by the number of input points. Our algorithm decomposes the input 3D model into a set of height fields and displays the model by rendering the height fields using per-pixel displacement mapping. Per-pixel displacement mapping performs image-space ray-casting on the 2D height fields, and thus allows us to avoid the vertex-processing bottleneck that plagues many point-based rendering techniques.

We assume the input model is a set of 3D points that sufficiently sample the surface of the object, and each 3D point has a surface normal vector. In practice, this type of input usually comes from range scanning of 3D objects. Traditionally, the detailed rendering of the acquired model can only take place after the overlapping range images have been aligned (registered), merged and simplified. The merging and simplification normally take many hours of computation and a huge amount of memory for very detailed scans. If the aligned range images were to be rendered as triangle meshes without merging and simplification, there would be too many triangles to render at interactive rates, and the overlapping regions would cause rendering artifacts, such as z-fighting.

Our algorithm is ideal for such application. It is able to take in all the points in the aligned range images, perform very simple local merging, and efficiently convert the entire model into multiple displacement maps. These displacement maps are then rendered at interactive rates, with accurate rendering of the object's silhouettes. The preservation of the surface details is only limited by the amount of texture memory on the graphics hardware. Our method is general enough that it can also take in those unprocessed triangle meshes as input and render them efficiently.

1.1 Contributions

The main contribution of our work lies in the simplicity of our method. In the preprocessing step, we use simple octree and PCA to convert the 3D model to a set of height fields. During rendering, we simply render the faces of each voxel to produce fragments for ray-casting on the GPU. Each ray is then transformed to the displacement map's local space. With this approach, we can accurately render the object's silhouettes even when using a simple 2D height field ray-casting algorithm, such as the iterative parallax mapping algorithm [2]. This is unlike existing displacement mapping techniques, which usually require complex decomposition of the input model into a set of base polygons or volumes, and use more complex algorithms to compute intersections between viewing rays and the displacement maps [4, 12, 22, 27]. More importantly, most of these methods do not correctly render the object's silhouettes [12].

The simplicity and generality of our decomposition and rendering algorithms have allowed our method to be used as a practical interactive rendering technique for high-detail 3D models. Our method is also a better alternative for large point-based models, in that it does not have the vertex processing bottleneck.

In the next section, we review existing work on real-time displacement mapping and on point-based rendering.We describe our algorithm in detail in Section 3. Our experiment results are presented in Section 4, and finally, we conclude the paper and discuss some possible future work in Section 5.

2 RELATED WORK

Image-space ray-casting was introduced by Roth in 1982 [29] for visualizing and modeling 3D solid objects. In general, image-space ray-casting offers more flexible visualization than the triangle-based rasterization approach. For example, Levoy introduced a 3D volume rendering algorithm for displaying medical images by compositing colored opacity along viewing rays [17]. Ray-casting is

*e-mail: TKLU@ntu.edu.sg
†e-mail: lowkl@comp.nus.edu.sg
‡e-mail: asjmzheng@ntu.edu.sg

also used for accurate rendering of parametric surfaces [14] [23]. Parametric surfaces such as Bezier or NURBs patches are defined only by sets of control points and cannot be rendered directly using triangle-based rasterization.

Last few years have seen a new interest in image-space ray-casting for real-time displacement mapping. The idea of adding fine local details on surfaces through 2D height maps was first proposed by Cook [5]. It allows complex geometric variations to be added onto much simpler geometry. Over the years, there have been several proposals for applying displacement mapping techniques on dedicated hardware. It began with vertex-based displacement mapping. For example, Bunnell presented a level-of-detail driven rasterization approach, implemented on GPU, that inserts new vertices into the base mesh [3]. Moule and McCool have improved the area coverage computation to detect change in the displacement map to drive a similar adaptive scheme [20]. However, if not done carefully, these methods can often result in an explosion of the number of vertices [11]. On the other hand, instead of adding new vertices, techniques using the image-space ray-casting approach for displacement mapping can improve the surface geometric details without adding extra vertices. This approach was initially proposed by Pharr and Hanrahan [24], and by Heidrich and Seidel [10]. However, the initial algorithms involve complex intersection calculations that do not map well to standard programmable graphics hardware.

Only in recent years do we see a new proliferation of real-time image-space displacement mapping techniques on GPU. Parallax mapping [2, 15] is one of the first methods for approximating the parallax seen on uneven surfaces. However, the technique uses single-step sampling that does not account for occlusion. The generalized displacement mapping method proposed by Wang et al. [33, 34], on the other hand, stores per-pixel displacement information from all viewing directions. The technique provides accurate and non-aliasing results by pre-computing the appropriate offset for each potential view direction, and thus avoiding ray-surface intersection computation during rendering. Similarly, per-pixel displacement mapping using distance functions [6] employs 3D texture, so called distance map, to store the distance from each voxel inside the displacement volume to the closest point on the height field. Large texture memory consumption is, however, an obvious problem for this algorithm, which makes it infeasible for large and high-resolution displacement maps. Later, McGuire and McGuire introduced the iterative steep parallax mapping [19] to improve on the previous parallax algorithm [15]. Steep parallax mapping uses a linear search to find ray-surface intersections along the viewing ray with regular intervals. The method provides better results but still has aliasing artifacts at grazing view angles [2]. However, an important advantage of steep parallax mapping [19] over the generalized displacement mapping method [33,34] is that it is much more memory efficient. Relief mapping [26] is another iterative ray-casting method that consists of a linear search, in the same fashion as steep parallax mapping, and a binary search to refine the intersection. To skip empty regions safely, Oh et al. used a hierarchical image pyramid of the displacement map to yield more accurate intersection results, even on steep-slope height fields [21].

In general, image-space per-pixel displacement mapping techniques add local distortion onto the surface efficiently by shifting texture coordinates according to the depth in the height map and the view direction. However, different from vertex-based displacement mapping techniques, most image-space displacement mapping techniques can only perform ray-casting on flat 2D planes. A view ray starts from the base polygon on top of the height map and marches down to search for new parallax texture coordinates at the intersection point. But in reality, a ray can pass over its current height map and continue to march into its neighboring height map as illustrated in Figure 3(b). This leads to incorrect rendering of the object's silhouettes often seen in many per-pixel displacement mapping methods [2, 7, 8, 25, 26]. The generalized displacement mapping technique of Wang et al. can produce accurate height field silhouettes [33], but the cached 5D data consumes too much memory for it to be used for rendering high-resolution 3D models. For an efficient and general displacement mapping method to render high-detail complex 3D models, we present a simple alternative, which we describe in detail in Section 3.

On the other hand, for quick visualization of range data, Rusinkiewicz and Levoy have proposed the use of points as rendering primitives [30] for displaying point-sampled surfaces directly without expensive preprocessing. However, since each point is "splatted" directly onto the framebuffer without neighboring interpolation, point-based rendering often results in aliasing artifacts. For high-quality rendering, low-pass filtering is employed and has been implemented on hardware [1, 16, 28, 31]. However, low-pass elliptical weighted average (EWA) filtering is an expensive operation as it requires multiple passes for rendering smooth surfaces on GPU. The performance, as a result, is thus sacrificed for visual quality. Most recently, Marroqium et al. [28] introduced a new point-based rendering algorithm that has good rendering performance and still maintains high-quality result. It uses pull-push interpolation in image space to reconstruct the surfaces from points. The method was claimed to have better performance than previous multi-pass methods because it is more GPU-friendly. It requires only single rendering pass for point projection, and is less limited by vertex processing bottleneck.

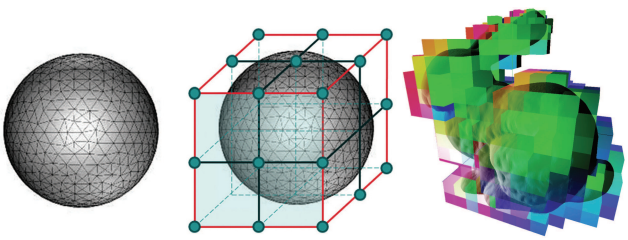
3 Our Algorithm

Our algorithm consists mainly of a preprocessing part and a rendering part. The following is a brief description of the two parts and the details are provided in the subsequent subsections.

Part 1: Preprocessing

1. We use an octree to decompose the input 3D model into a finite set of 2D height fields. Each height field has a domain plane such that no two points on the height field surface can be orthogonally projected onto the same point in the domain plane. The number of height fields depends on the model's geometric complexity rather than its vertex density. For example, a single sphere shown in Figure 1(a) can be decomposed into eight separate height fields regardless of how many points are actually used to approximate the curved surface.

2. Next, each height field is converted into a 2D height map (or displacement map) and packed into hardware memory in the



(a) HF decomposition of 3D Sphere (b) HF decompsition of Bunny

Figure 1: Decomposition of 3D models into 2D height field surfaces. Each height field surface is bounded by a bounding box.

form of a 2D texture atlas (see Figure 2).

3. Finally, the faces of the octree voxel that bounds the height field are appropriately assigned texture coordinates. These faces are the polygons that are being displacement-mapped, and will be rasterized during rendering to produce the necessary fragments. Each of these fragments represents a viewing ray, which is used to search the height map for intersection. Figure 1(b) shows a height-field decomposition of the Stanford Bunny model, together with the bounding boxes.

Part 2: Rendering

1. The bounding boxes of the height fields are projected and rasterized to produce fragments, where each fragment provides the initial coordinates from which the viewing ray starts marching into the height field domain.

2. For each fragment, the viewing ray is transformed from the world coordinate frame to the local coordinate frame of the corresponding 2D height map.2

3. Finally, we perform ray-casting algorithm to find the first intersection between the ray and the height field surface, as illustrated in Figure 3.

Our rendering algorithm performs image-space ray-casting on 2D height fields, and its performance thus has no relation to the original number of vertices or points. The time complexity of our rendering algorithm is $O(F+B)$ where F is the total number of fragments produced by the rasterization and B is the total number of bounding boxes. In our experiments, even for very huge models with hundred millions vertices, the value of B is typically less than 100 thousand. In modern graphics processors, this number of boxes can be handled comfortably by the vertex processing units at high frame rates. The performance of our rendering algorithm is thus very unlikely to be geometry-bound.

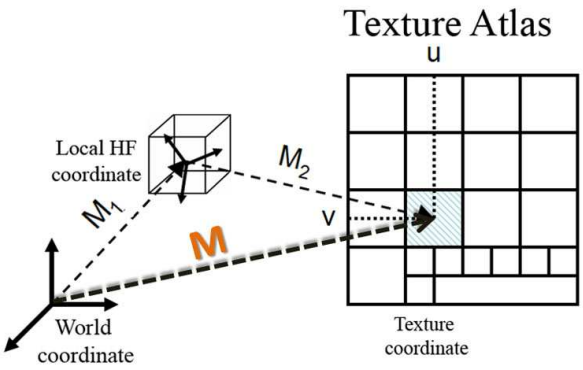
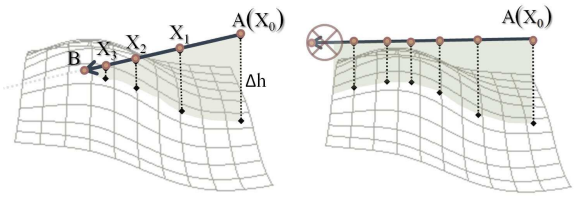


Figure 2: Transformation of coordinates. During ray-casting, each viewing ray is transformed from the world coordinate space into the texture coordinate space using the transformation matrix **M**.

3.1 Height Field Decomposition

Given an input 3D model, the main purpose of this process is to subdivide the original complex 3D surface into many simpler 2D height field surfaces. Height field surfaces allow simpler and more efficient computation of ray-surface intersections. Finding suitable height field domain planes such that the number of height fields is minimal is not a trivial problem. Here, we use an octree to partition the model surface and use the PCA transform to test and compute a suitable height field domain plane for each surface patch.



(a) Case 1 : Ray hits surface

(b) Case 2 : Ray passes over surface

Figure 3: Illustration of iterative parallax ray-casting method for 2D height field.

3.1.1 Octree Decomposition

In the preprocessing stage, our algorithm subdivides the bounding cube of the 3D model into smaller cells such that each cell must contain either a 2D function surface (a height field surface) or just empty space. If the surface within a cell is not a height field surface, it is further subdivided into smaller cells. Besides its simplicity, octrees give us the following advantage. The octree partitioning scheme subdivides volume into eight orthogonal and non-overlapping bounding cubes. This allows simple and unique mapping of rays in the bounding boxes to 2D depth map coordinates, which enables our ray-casting algorithm to perform efficiently and correctly. This is unlike other iterative displacement mapping techniques that uses triangular prisms as bounding volumes. These prisms are formed by extruding the base triangle mesh in the direction of the vertex normals. As of now, there is still no accurate and efficient algorithm that can correctly render displacement maps using a base triangle mesh. Porumbescu et al. [27] have proposed subdividing the extrusion prisms into tetrahedra to provide a piece-wise linear mapping from "shell space" to height field "texture space". However, these mappings lead to objectionable artifacts at the subdivision boundary, similar to the artifacts induced by non-perspective-correct texture mapping [12]. The reason is that the straight ray path in the prism space becomes a nonlinear curve in the height field texture space [12]. Jeschke et al. [12] proposed curved shell mapping that is accurate and tangent-continuous at prism borders. However, curved shell mapping is slow due to its complexity and is not suitable for real-time visualization. Chen and Chang [4] attempted to address the tangent discontinuities along the viewing ray by employing an extra tangent-space map to bend the viewing ray in the texture space. However, the result is not accurate with noticeable inconsistent displacement from different viewing angles (these artifacts are visible in their demo video, and have been acknowledged by one of its authors). The generalized displacement mapping method [33] is able to give correct intersection between ray and height field surface in general case because ray marching distances have been precomputed and are retrieved during rendering. However, large memory requirement is a problem for this method.

3.1.2 Height Field Test

For each octree cell, we need to determine whether the enclosed surface is a height field surface. If it is not, then the octree cell is subdivided, otherwise a height field domain plane has to be computed. Here, we employ the PCA transform [13] for the height field condition test. First, we apply the PCA to all the points or vertices in the octree cell to find the best-fit plane. Next, using the plane's normal vector as the projection direction, we determine whether any of the points or vertices in the cell has its normal vector back-

facing the projection plane (assuming all the points are on one side of the projection plane). If there is, then this is not yet a height field surface and the cell needs to be subdivided. Otherwise, the best-fit plane is used as the height field domain plane for the cell. In our experiments, the octree height varies between 6 and 10.

The PCA height field condition test may fail even when the surface is already a height field surface. This can happen when many points are distributed on a few "tall mountains" on the surface. For this case, it may actually be better to let the test fail and subdivide the surface further. This is because "tall-mountain" surfaces result in abrupt changes in the height maps, which are more expensive to intersect accurately during image-space ray-casting.

3.2 Setting Up Displacement Mapping

In the preprocessing stage, after the height field decomposition, we need to generate the height maps (displacement maps), and find the transformation from the world space to the height field texture space, so that we can set up the texture coordinates of the bounding boxes, and transform the viewing ray from the world space to the texture space.

3.2.1 Height Map Generation

A height map is a discrete 2D image that records the scalar distance between each point on the height field surface and the domain plane. For each octree cell, if the input model is made up of triangle meshes, we simply orthographically project the meshes onto the framebuffer along the domain plane normal vector direction, with the camera placed at the cell center. The content in the depth buffer is used as the height map for the cell. If the input model is made up of points, then we employ the efficient pull-push interpolation method [28] to reconstruct the height map.

Each height map is a square image whose side is $\sqrt{3}$ times the length of the corresponding bounding box, so that any point in the cell can always be projected onto the height map regardless of the orientation of the domain plane relative to the bounding box. The height maps are then tightly packed into one or more texture atlases (see Figure 2). The image resolution allocated to each height map is half (in both dimensions) of that in its parent's level. Therefore, by counting the number of octree leaf nodes at different levels and with the given total available texture image size, we can compute the image resolution that can be allocated to each height map. We leave pixel-wide gaps between height maps in the texture atlas to take care of the hardware texture filtering.

3.2.2 Coordinates Transformation

To enable the rendering stage to perform ray-casting of a height map, we need to transform each viewing ray into texture coordinates in the texture atlas. The transformation of a viewing ray from the world coordinate space to the texture coordinate space can be decomposed into two successive transformations. The first one is a transformation from the world coordinate space to the height field coordinate space (the height field coordinate space is a by-product of the PCA height field test). The second transformation is one from the height field coordinate space to the texture coordinate space. Figure 2 illustrates these transformations. Let $\mathbf{M}$ be the combined transformation, and in matrix form,

$$\mathbf{M} = \mathbf{M}_1 \cdot \mathbf{M}_2$$

where

$$\mathbf{M}_1 = \begin{pmatrix} u_1 & u_2 & u_3 & -\mathbf{U}\cdot\mathbf{O} \\ v_1 & v_2 & v_3 & -\mathbf{V}\cdot\mathbf{O} \\ w_1 & w_2 & w_3 & -\mathbf{W}\cdot\mathbf{O} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{M}_2 = \begin{pmatrix} s_1 & 0 & 0 & t_1 \\ 0 & s_2 & 0 & t_2 \\ 0 & 0 & s_3 & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In $\mathbf{M}_1$, the vectors $\mathbf{U} = [u_1, u_2, u_3]^T$, $\mathbf{V} = [v_1, v_2, v_3]^T$, and $\mathbf{W} = [w_1, w_2, w_3]^T$ are the orthogonal axes of the height field coordinate frame, and $\mathbf{O} = [o_1, o_2, o_3]^T$ is the coordinates of its origin. The matrix $\mathbf{M}_2$ maps the height map into its actual size and location in the texture coordinate space of the texture atlas. The height values in the height map are also mapped by $\mathbf{M}_2$ to the range [0, 1].

In the preprocessing stage, each vertex of a bounding box is assigned texture coordinates. The texture coordinates are computed by multiplying the matrix $\mathbf{M}_1$ to the world coordinates of the vertex. During the rendering stage, these texture coordinates will be interpolated at each fragment produced by the rasterization of the bounding box. The interpolated texture coordinates provide the initial position of the viewing ray in the height field coordinate frame. The viewing ray direction vector is transformed by the top-left 3×3 sub-matrix of $\mathbf{M}_1$. Each query of the height map is transformed into the texture space by $\mathbf{M}_2$.

3.3 Rendering

Our rendering algorithm has two steps. The first step starts with the rasterization of the bounding boxes. With backface culling, at most three faces of each bounding box need to be rasterized. Each vertex of the bounding box has been assigned 3D texture coordinates as described in Section 3.2.2. These texture coordinates will be interpolated at each fragment produced by the rasterization of the bounding box. The interpolated texture coordinates provide the initial position of the viewing ray in the height field coordinate frame (X_0 in Figure 3).

In the next step, for each fragment, we transform the viewing ray direction from the world space into the height field space using the precomputed matrix $\mathbf{M}_1$ in Section 3.2.2. From the initial position of the viewing ray, we iteratively search for the first intersection between the ray and the height map. This step is illustrated in Figure 3.

Basically, we compute the intersection by marching along the ray, from the initial location, to find the position that has the closest distance above the height map. In fact, many algorithms have been proposed for 2D height field ray-casting [6,19,21,26,34]. Szirmay-Kalos and Umenhoffer [32] have presented a detailed analysis of the different existing techniques. We have adapted the iterative parallax mapping algorithm [2] for two reasons. Firstly, unlike many other methods [6, 7, 34], the iterative parallax mapping algorithm does not need to pre-compute extra information. This makes it memory efficient. Secondly, iterative parallax mapping produces less aliasing artifacts compared to other methods such as the relief mapping [32].

3.3.1 Boundary Culling

After an intersection on the height map has been found, we perform a test, called the boundary test, to ensure that the ray indeed intersects the current height map and not a neighboring height map in the texture atlas. This may happen when the viewing ray is at a shallow angle and passes over the current height map (see Figure 3(b)). If the intersection is in the wrong height map, the corresponding fragment is discarded. The boundary test tells whether the ray intersects the height field surface within the bounding boxes. If it does not, the fragment is discarded so that it will not occlude the correct fragment value when the ray intersects height fields inside bounding boxes that are further away from the viewpoint. The boundary test can be easily implemented in our case since we already know the limits of each height map in the texture atlas.

Model	Vertices	Cells	Texels	Preprocessing Time (mins)	Frames Per Second (1)	(2)	(3)
St. Mathew	368M	57K	101M	20.7	147	18	12
David	28M	31K	96M	0.7	300	32	23
Lucy	14M	41K	91M	1.4	162	28	17
Thai Statue	5M	74K	64M	1.2	170	21	9

Table 1: Rendering results. The framerates are for (1) NVIDIA GeForce 8800 GT, (2) NVIDIA GeForce 8500 GT, and (3) NVIDIA GeForce Go 7900.

Note that each pixel of the image represents a viewing ray, and it may intersect multiple bounding boxes, and thus multiple height fields. We use z-buffering to keep only the frontmost height field fragment intersected by the ray.

4 Results and Discussion

We tested our algorithm on various models from the Stanford 3D Scanning Repository (www-graphics.stanford.edu/data/3Dscanrep). The most challenging test is the model of the St. Mathew statue, which has 368 million vertices. Our algorithm is able to render the model, to a 1024x1204 viewport, with great details at 12 FPS on a laptop computer equipped with a GeForce Go 7900 GPU connected via PCI Express 16x to an Intel P4, 2.4 GHz CPU. We used a total of 101 million texels to store the 57 thousand height maps for this model, so fine details are quite well preserved. The height field decomposition of the St. Mathew model is shown in Figure 4. Eight iterations were used for the parallax mapping ray-casting.

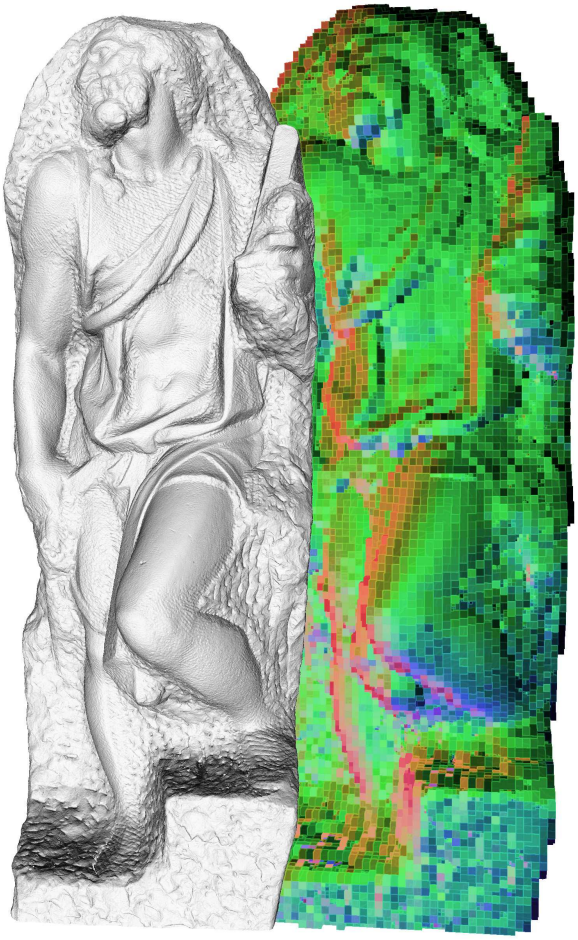


Figure 4: The St. Mathew model (368M) with its height field decomposition cells (57K). The different colors of the cells indicate the different orientations of the domain planes of the height fields.

Rendering times for several other models on different graphics hardware are summarized in Table 1. The rendered image is 1024x1024 in size. Sample images from the rendering of these models are shown in Figure 6, with close-up views for quality comparison. It is evident from the sample images that the object's silhouettes have been properly rendered. To achieve accurate results, using 12 iterations in the height field ray-casting has been sufficient to produce images with no noticeable errors. For comparison, Figure 5 shows sample results produced using different number of iterations in the ray-casting.

Even though the Thai Statue has only 5 million vertices, its geometric complexity is actually high, and this leads to high number of octree subdivisions. We use 64 million texels to represent the height maps, and this must have a lot of data redundancy since the original model has only 5 million vertices. We believe these height maps could be compressed very well through hardware texture map compression. Triangle-based rendering may be more suitable for this model since the vertices on the model are relatively sparse.

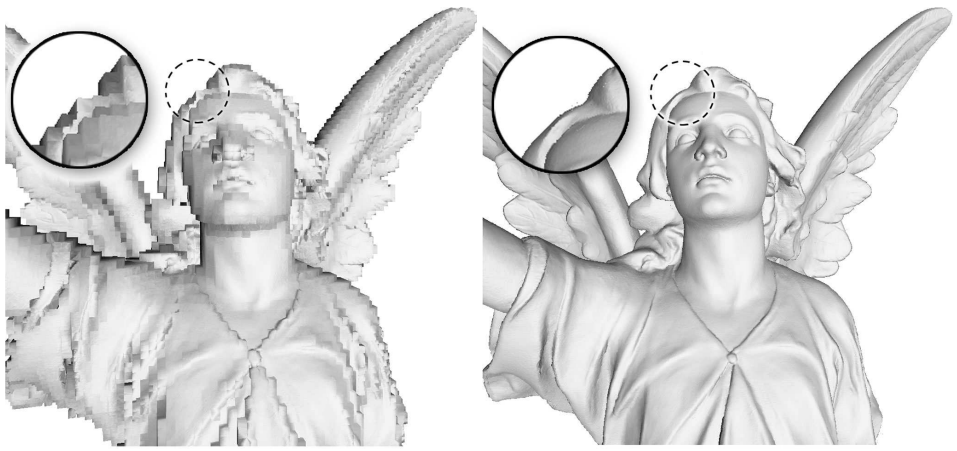
The St. Mathew model is an ideal case for our algorithm because the base surface is relatively smooth and simple, which is covered with a lot of fine surface details. This leads to fewer octree subdivisions because many large surface regions satisfy the height field condition. Smaller number of bounding boxes results in less pixel overdraw and thus speeds up the rendering.

Note that our height field data structure is not multiresolution because each surface of the model is only stored at a single resolution. However, the rendering is inherently and automatically multiresolution, in the sense that the projected image size of each bounding box determines how many ray-castings have to be performed for each height field.

5 Conclusions and Future Work

We have presented a new approach to decompose a complex 3D model into height fields and render the height fields using a image-space ray-casting algorithm. Compared to existing methods that use the point-based rendering approach [28], our algorithm offers significant performance improvement by avoiding the vertex-processing bottleneck. Moreover, the way we decompose the 3D model into height fields, together with our improved ray-casting algorithm, has resulted in a simple displacement mapping approach that can render object's silhouettes correctly and efficiently. The relatively short preprocessing time required by our algorithm has made it suitable for quick visualization of range scanning data on-site.

There are still some limitations of our algorithm that requires future research. For example, results have shown that our algorithm works well for surface with high-frequency features, but it is not efficient for flat and smooth surfaces. An interesting solution might be a combination of triangle-based rendering and per-pixel displacement mapping. The amount of details that we can preserve from the input model is limited by the amount of texture memory.



(a) 1 iteration

(b) 4 iterations

(c) 8 iterations

Figure 5: Iterative parallax mapping with different number of iterations in the ray-casting.

A limitation of our current height field decomposition method is that, although it is aware of the height field condition, it does not adaptively use the amount of surface details in a voxel to determine the height map size in the texture atlas.

Since we do not use mipmapping for our height maps, our method may produce aliasing artifacts at very small surface details. We cannot use straightforward mipmapping for our case as the texture atlas is made up of different height maps. We plan to get around this by pre-filtering each height map separately before putting them on texture atlases of different "mipmap levels."

Another possible extension to our algorithm is to make it suitable for dynamic or deformable surfaces. The success of that requires selective update of the height field decomposition of the affected surface regions. Many other preprocessing steps, such as height map generation, must be highly optimized to provide the much needed speed. Currently, our preprocessing algorithm is not out-of-core. An out-of-core algorithm is necessary when the input model is too large to fit into the system memory.

ACKNOWLEDGEMENTS

We thank Marc Levoy and the Digital Michelangelo Project team of Stanford University for providing, and giving us the permission to use, the beautiful 3D models. This project is partially supported by the National University of Singapore Academic Research Fund (WBS: R-252-050-241-112 & R-252-050-241-133).

REFERENCES

[1] M. Botsch and L. Kobbelt. High-quality point-based rendering on modern gpus. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on*, pages 335–343, 2003.

[2] Z. Brawley and N. Tatarchuk. Parallax occlusion mapping: Self-shadowing, perspective-correct bump mapping using reverse height map tracing. *ShaderX3: Advanced Rendering with DirectX and OpenGL*, pages 135–154, 2004.

[3] M. Bunnell. Adaptive tessellation of subdivision surfaces with displacement mapping. *GPU Gems*, 2:109–122, 2005.

[4] Y.-C. Chen and C.-F. Chang. A prism-free method for silhouette rendering in inverse displacement mapping. *Pacific Graphics*, 27, 2008.

[5] R. Cook. Shade trees. *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 223–231, 1984.

[6] W. Donnelly. Per-pixel displacement mapping with distance functions. *Addison Wesley*, 17:334–339, 2005.

[7] J. Dummer. Cone step mapping: An iterative ray-heightfield intersection algorithm, 2006.

[8] S. P. Eric A. Risser, Musawir A. Shah. Interval mapping. *Symposium on Interactive 3D Graphics and Games (I3D)*, 2006.

[9] P. Heckbert, M. Garland, and C.-M. U. P. P. S. O. C. SCIENCE. A survey of polygonal surface simplification algorithms, 1997.

[10] W. Heidrich and H. Seidel. Ray-tracing procedural displacement shaders. *Language*, 20(10):24, 1998.

[11] J. Hjelmervik and T. Hagen. Gpu-based screen space tessellation. *Mathematical Methods for Curves and Surfaces: Tromsø, 2004*, 2004.

[12] S. Jeschke, S. Mantler, and M. Wimmer. Interactive smooth and curved shell mapping. *Proceedings of EGSR'07*, 2007.

[13] I. Jolliffe. *Principal component analysis*. Springer New York, 2002.

[14] J. Kajiya. Ray tracing parametric patches. *Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pages 245–254, 1982.

[15] T. Kaneko, T. Takahei, M. Inami, N. Kawakami, Y. Yanagida, T. Maeda, and S. Tachi. Detailed shape representation with parallax mapping. *Proceedings of ICAT 2001*, pages 205–208, 2001.

[16] L. Kobbelt and M. Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28(6):801–814, 2004.

[17] M. Levoy. Display of surfaces from volume data. *Computer Graphics and Applications, IEEE*, 8(3):29–37, 1988.

[18] M. Levoy, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, J. Davis, J. Ginsberg, et al. The digital michelangelo project: 3d scanning of large statues. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 131–144, 2000.

[19] M. Mc Guire and M. Mc Guire. Steep parallax mapping. *I3D 2005 Poster*, 2005.

[20] K. Moule and M. McCool. Efficient bounded adaptive tessellation of displacement maps. *GRAPHICS INTERFACE*, pages 171–180, 2002.

[21] K. Oh, H. Ki, and C. Lee. Pyramidal displacement mapping: a gpu based artifacts-free ray tracing through an image pyramid. *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 75–82, 2006.

[22] M. Oliveira and F. Policarpo. An efficient representation for surface details. Technical report, UFRGS Technical Report RP-351, 2005.

[23] H. Pabst, J. Springer, A. Schollmeyer, R. Lenhardt, C. Lessig, and B. Froehlich. Ray casting of trimmed nurbs surfaces on the gpu. *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 151–160, 2006.

[24] M. Pharr and P. Hanrahan. Geometry caching for ray-tracing displacement maps. *Eurographics Workshop on Rendering*, pages 31–40, 1996.

[25] F. Policarpo and M. Oliveira. Relaxed cone stepping for relief mapping. *GPU Gems*, 3, 2007.

[26] F. Policarpo, M. Oliveira, and J. Comba. Real-time relief mapping on arbitrary polygonal surfaces. *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 155–162, 2005.

[27] S. Porumbescu, B. Budge, L. Feng, and K. Joy. Shell maps. *Proceedings of ACM SIGGRAPH 2005*, 24(3):626–633, 2005.

[28] P. R. C. Ricardo Marroquim, Martin Kraus. Interactive point based rendering using image reconstruction. *Symposium on Point Based Graphics*, pages 101–108, 2007.

[29] S. D. Roth. Ray casting for modeling solids. *j-CGIP*, 18:109–144, 1982.

[30] S. Rusinkiewicz and M. Levoy. Qsplat: a multiresolution point rendering system for large meshes. *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352, 2000.

[31] M. Sainz, R. Pajarola, and R. Lario. Points reloaded: Point-based rendering revisited. In *Proceedings Symposium on Point-Based Graphics*, pages 121–128, 2004.

[32] L. Szirmay-Kalos and T. Umenhoffer. Displacement mapping on the gpu-state of the art. *Computer Graphics Forum*, (0), 2007.

[33] X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H. Shum. Generalized displacement maps. pages 227–233, 2004.

[34] X. C. Wang, J. Maillot, E. Fiume, V. N. thow hing, A. Woo, and S. Bakshi. View-dependent displacement mapping. *ACM Transactions on Graphics*, 22(3):334–339, 2002.

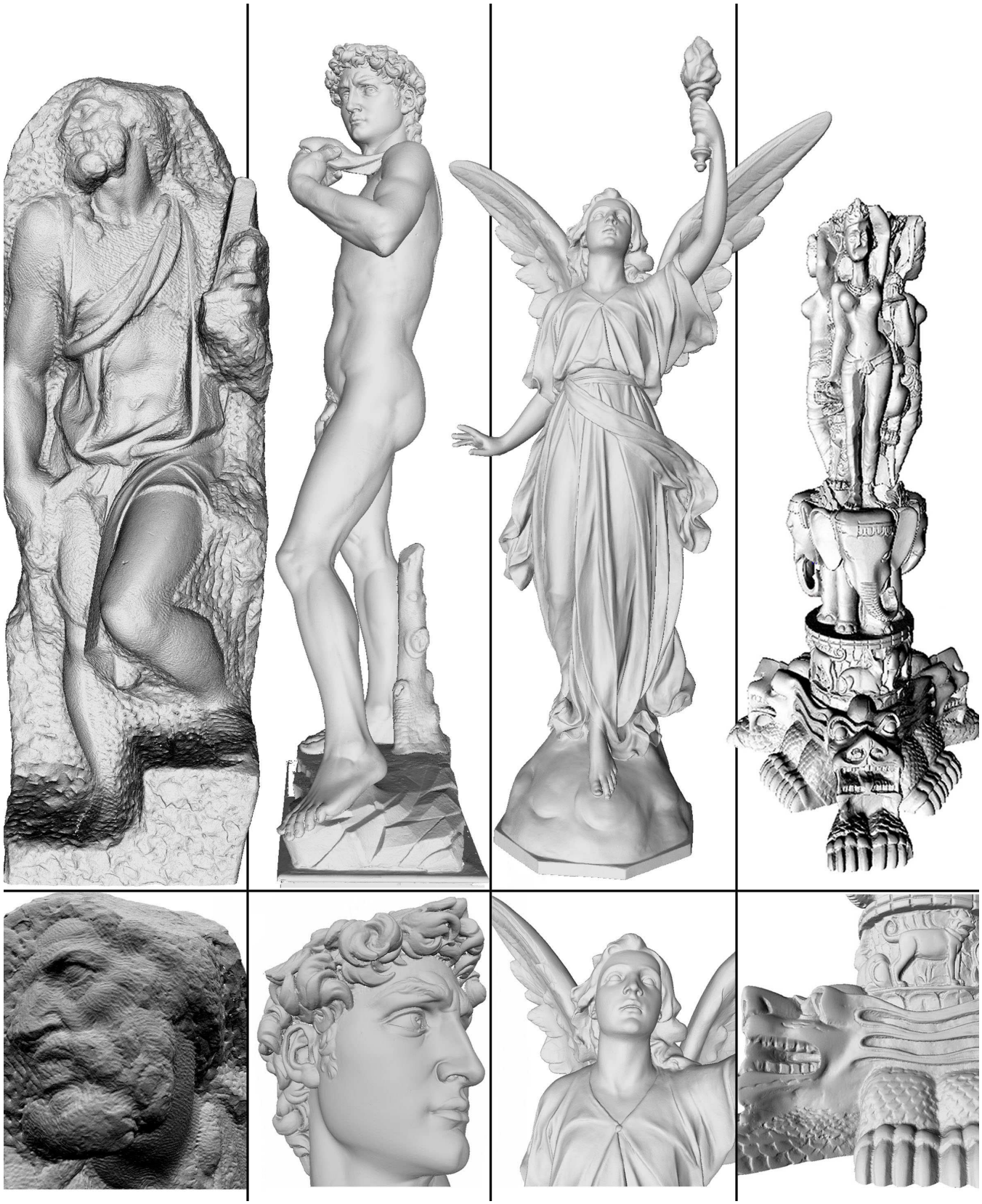


Figure 6: Sample images produced by our rendering algorithm. From left to right: St. Mathew (368M), David (28M), Lucy (14M), and Thai Statue (5M).