Displacement Mapped Billboard Clouds

Stephan Mantler[1] and Stefan Jeschke[2] and Michael Wimmer[2]
[1]VRVis Research Center, Vienna, Austria
[2]Institute of Computer Graphics, Vienna University of Technology

Figure 1: Impostors of the dragon model. Left: traditional billboard cloud with 31 rectangles and 1.76 MB required texture memory. Middle: traditional billboard cloud with 186 rectangles and 11MB required texture memory. Right: displacement mapped billboard cloud with 31 boxes and 10.3 MB required textured memory.

Abstract

This paper introduces *displacement mapped billboard clouds* (DMBBC), a new image-based rendering primitive for the fast display of geometrically complex objects at medium to far distances. The representation is based on the well-known *billboard cloud* (BBC) technique, which represents an object as several textured rectangles in order to dramatically reduce its geometric complexity. Our new method uses boxes instead of rectangles, each box representing a volumetric part of the model. Rendering the contents of a box is done entirely on the GPU using ray casting. DMBBCs will often obviate the need to switch to full geometry for closer distances, which is especially helpful for scenes that are densely populated with complex objects, e.g. for vegetation scenes. We show several ways to store the volumetric information, with different tradeoffs between memory requirements and image quality. In addition we discuss techniques to accelerate the ray casting algorithm, and a way for smoothly switching between DMBBCs for medium distances and BBCs for far distances.

1 Introduction

Image-based surface representations are popular in real-time rendering due to their ability to provide rich visual surface detail at very low computational expense. The most widely used examples are texture and normal maps. More recently, so-called *displacement maps* introduce geometric detail using a height field that can be rendered by exploiting modern graphics hardware, while keeping memory and computational costs relatively low.

In addition to enhancing surface appearance, image-based representations have also been used to accelerate rendering by replacing complex parts of a scene with simple textured geometry that resemble the geometric models to certain degree. Such image-based representations are called *impostors*. A recently proposed impostor technique that received much attention are *billboard clouds* (BBC) by Decoret et al. [Decoret et al. 2003]. While most existing impostor primitives are restricted to a particular viewing region, BBCs represent a part of a scene as a collection of arbitrarily placed and oriented texture-mapped rectangles to which parts of the original geometry are projected (see Figure 1). Consequently, the represented scene parts can be observed from all sides. While the quality of BBCs is often sufficient for distant parts of a scene, the "flatness" of the individual rectangles becomes very noticeable from closer distances or at grazing angles. This cannot be easily avoided: switching to full geometry earlier defeats the purpose of BBCs as this leads to a much higher geometry load, while substituting a different BBC with a higher accuracy typically leads to noticeable popping artifacts. This restricts the usefulness of the BBC technique for scene parts closer to the viewer.

In this paper, we introduce a new representation called *displacement mapped billboard cloud* (DMBBC) that overcomes this problem. The main idea is to replace the *billboard rectangles* of a BBC with *billboard boxes*. Within every such box, fine scale geometric details are represented using a texture-based representation called *volumetric displacement function*, which can be a 2.5D height map (the basic displacement map) or even a full 3D texture (a "volumetric displacement map"). This leads to a much lower geometric and visual error compared to BBCs. Impostors based on DMBBCs allow for fast rendering of complex scenes entirely on the GPU with sufficiently high image quality even for closer objects. In particular, the artifacts typically associated with billboard clouds are significantly reduced through the additional parallax and visual depth, while the representation remains image-based, i.e. only negligible additional geometric complexity is introduced. The new representation allows for smooth blending with traditional billboard clouds

so that distant scene parts can be rendered even faster. We show the potential of the new representation for complex objects containing curved surfaces, where traditional billboard clouds typically fail to provide a sufficiently high image quality.

2 Related Work

For a comprehensive overview of impostor techniques (including simple *planar impostors*, *layered depth images* and *textured depth meshes*), we refer the reader to a recent state-of-the-art report by Jeschke et al. [Jeschke et al. 2005]. Since DMBBCs are based on the billboard cloud technique, we will discuss in Section 2.1 work closely related to this technique. Section 2.2 describes methods based on textures that are processed in the pixel shader, mostly for describing surfaces. Although the technique in this paper is not restricted to surfaces, the used methods are quite similar.

2.1 Billboard clouds

A billboard cloud [Decoret et al. 2003] is a powerful impostor primitive because it represents scene parts from all possible viewing directions. This allows for a straightforward use of shadow mapping as well as an easy integration into most existing rendering systems just like geometric levels of detail. Normal maps can be stored with every rectangle, allowing for dynamic lighting at runtime. Meseth and Klein [Meseth and Klein 2004] stored bidirectional texture functions for the BBC rectangles in order to preserve view and light dependent effects.

On the other hand, generating a BBC is not trivial. The rectangles have to approximate the surface of a scene part with respect to a user-defined geometric error. The problem is the choice of good rectangles that fulfill two contradicting demands: on the one hand, a small number of rectangles is desired, but on the other hand, every rectangle should contain as few "empty texels" (texels that do not represent anything at all) as possible in order to save memory. The original method by Decoret et al. implemented a greedy search based on rectangle evaluation in Hough space. An algorithm that automatically finds one rectangle covering the largest portion of a (water-tight) model was presented by Andujar et al. [Andujar et al. 2004]. Unfortunately, any greedy optimization that chooses rectangles consecutively cannot give a guarantee that the resulting *set* of rectangles is good. Meseth and Klein [Meseth and Klein 2004] presented two additional algorithms for finding a good set of rectangles based on mesh simplification and hierarchical face clustering. Wahl et al.[Wahl et al. 2005] used a RANSACK algorithm for converting sampled points to BBC. Fuhrmann et al. [Fuhrmann et al. 2005] as well as Behrendt et al. [Behrendt et al. 2005] improve on the original algorithm to create BBCs with more desirable properties for forest rendering.

However, the most problematic issues with BBCs are artifacts that occur when a rectangle is observed from a grazing angle, so that the rectangle becomes apparent as such or it even completely disappears. Curved surfaces are especially challenging in this respect. Although some of the work mentioned above shows improvements for special objects like trees, there is no automatic method that can *guarantee* the absence of cracks and wrong silhouettes. The displacement mapped billboard cloud technique proposed in this paper overcomes this drawback.

2.2 Texture-based surface descriptions

As an early approach, *bump mapping* [Blinn 1978] solely relies on shading, providing no correct silhouettes nor parallax effects. Several more recent methods enrich surfaces with geometric details that are stored in texture maps. For instance, *parallax mapping* by Kaneko [T. Kaneko and Tachi] (with its various extensions [Welsh 2004; McGuire and McGuire ; Tatarchuk 2006]) simulate the appearance of a height field by shifting the texture coordinates within a texture depending on the viewing angle. Note that these techniques are hardly applicable for our purpose because a billboard cloud typically contains empty areas in the texture which cannot be represented in this way.

Surfaces with a certain depth can also be represented using ray casting on the GPU. Hirche et al. [Hirche et al. 2004] were the first who defined a volume on a surface. They extruded the triangles along the vertex normals which results in prisms. Every prism is decomposed into three tetrahedrons. The ray casting is then applied to every tetrahedron by calculating the entrance and exit point and linearly sampling and evaluating a heightfield between them. The first hit point with the heightfield determines the position, and the color is read from a color texture. This technique supports correct object silhouettes, self-occlusions, interpenetrations, and even shadowing. Later Dufort et al. [Dufort et al. 2005] extended the work to semi-transparent data and Porumbescu et al. [Porumbescu et al. 2005] discussed the mapping in a broader way. Note that the idea of Hirche et al. is most closely related to the technique used in this paper: the ray casting step is implemented entirely in graphics hardware and allows for interactive frame rates, which makes the basic idea perfectly suitable for our DMBBCs. The main conceptual difference is that instead of prisms, DMBBC primitives are boxes which we can directly use for ray casting and we will allow the ray casting step to discover holes in the representation. We also accelerated the ray sampling process by using *sphere tracing* [Donnelly]. *Cone step mapping* [Paglieroni and Petersen 1994] or an enhanced version based on *safety zones* by Kolb et al. [Kolb and Rezk-Salama 2005] are further alternatives.

More recently, Policarpo et al. [Policarpo et al. 2005] applied ray casting to the original polygons of a mesh and implicitly defined a thick surface *inside* the object. While this works considerably faster than Hirche's method, silhouettes are still defined by the mesh geometry. They extended their work to better discover silhouettes [OLIVEIRA and POLICARPO 2005] and to support multiple height values per texel position [Policarpo 2006]. However, their algorithm is based on the assumption of a continuous surface with well defined "inside" and "outside" values, which do not exist in the context of DMBBCs. The same applies to the work of Wang et al. [Wang et al. 2003; Wang et al. 2004] who use massive preprocessing in order to reduce the cost for intersection searching at runtime. Please note that most of the presented techniques are explained in more detail in a book of Watt and Policarpo [Watt and Policarpo 2005].

3 Displacement Mapped Billboard Clouds

After a description of the basic method in Sections 3.1 and 3.2, Section 3.3 describes in more detail the construction of DMBBCs, showing different ways how to store the acquired data for balancing memory consumption and image quality. Section 3.4 presents a suitable acceleration technique for the ray casting algorithm as well as blending between different representations.

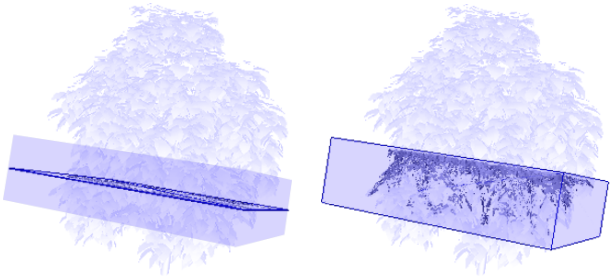


Figure 2: Left: billboard rectangle with its according validity region. Right: billboard box. Darker shaded regions show the rendered primitives together with the displayed contents. Note the difference between the flat rectangle and the volumetric contents of the box.

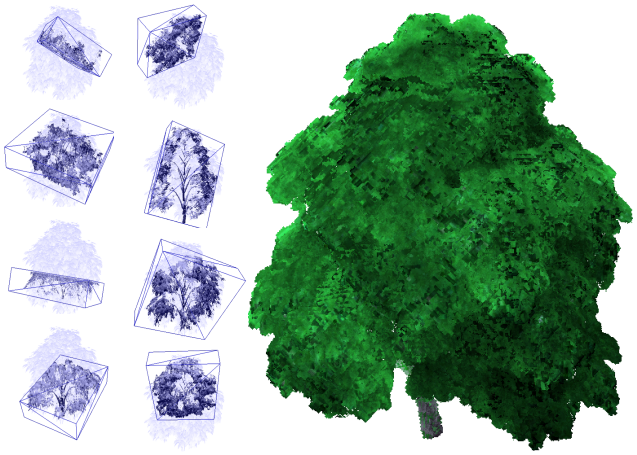


Figure 3: DMBBC representation of a chestnut tree consisting of 8 billboard boxes.

3.1 DMBBC definition

As mentioned earlier, a *billboard rectangle* in a BBC is used to represent all model parts that are closer to the rectangle than a user-defined *validity threshold* $\pm\varepsilon$. This validity threshold effectively represents a cuboid volume around the billboard rectangle, the *billboard box* (see Figure 2). The billboard box exactly defines the convex hull of the volume representing a DMBBC.

For a DMBBC, instead of simply projecting the model parts onto the rectangle and storing one color value, we store volumetric information, the so-called *volumetric displacement function*. At each texel (u,v), $f_{uv}(w)$ gives an opacity value for the height w along the rectangle normal, and an optional color value. For now, we assume that f is stored as a 3D texture with equidistant samples in w. Figure 3 shows the billboard boxes of a tree model.

3.2 Basic rendering algorithm

For displaying contents of a billboard box, we use a GPU ray-casting algorithm which determines the intersection of the viewing ray with the contained volumetric displacement function. The entry points of the viewing rays are interpolated from the corners of the billboard box by rendering the faces of the billboard box with interpolated (u,v,w) coordinates. The direction of the rays in texture space can simply be calculated with a world-to-texture space-matrix (provided as vertex attributes). The calculation of this matrix is very straightforward so we omit the derivation here.

The ray casting itself samples linearly along the ray in texture space (using a user-defined sampling rate) until an intersection is found, just like many previous methods described in Section 2.2. The search stops if either the ray leaves the box (which can be easily determined using the texture coordinates) so that the fragment can be discarded, or it intersects an opaque texel in the volumetric displacement function. In the latter case, the color value at the current position defines the output color. To optionally shade the current pixel, a normal can be read from an according map and used to apply an illumination model, as was already shown for BBCs [Decoret et al. 2003]. Note that the shader needs to output the correct depth value of the intersection point in order to correctly solve visibility between potentially overlapping billboard boxes. This is also straightforward and details are omitted here.

3.3 DMBBC generation

For generating the DMBBCs, we assume that the billboard rectangles together with the error value ε are already given. The rectangles can be obtained using any of the algorithms mentioned in Section 2.1. The extents of a billboard box are defined by a rectangle together with ε. The (u,v) texture resolution of the rectangle (defined by the closest distance the impostor should be valid for, see [Decoret et al. 2003]) is also valid for the box. Now given the original geometric model, we need to calculate and store the volumetric displacement function $f_{uv}(w)$. There are several ways how to represent this function. One obvious choice is as a 3D texture (see Section 3.3.1). However, if the high memory consumption of this approach is not acceptable, Section 3.3.2 and Section 3.3.3 present two other methods that significantly reduce the memory requirements while often providing a sufficiently high image quality.

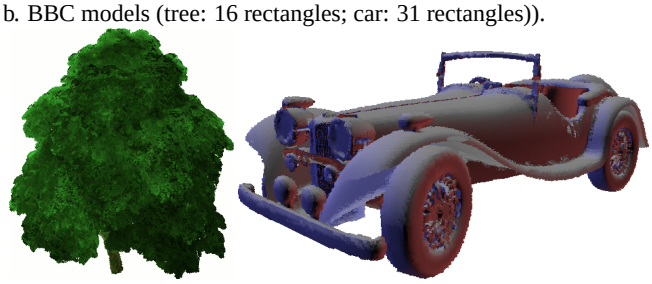
3.3.1 Volumetric Representation

The highest DMBBC quality can be achieved by sampling the volumetric displacement function in a 3D texture. The resolution (sampling interval) of the texture in w-direction should be chosen so that the voxels are approximately cubic, which effectively generalizes the concept of texture resolution to 3D in the sense that a texel defines the smallest representable feature. The 3D texture is then filled by rasterizing the triangles of the original model into these voxels, and collecting for each voxel its normal vector and color information. If multiple polygons occupy the same voxel, this data can be either averaged with equal weights (this was used to generate Figure 4, c), or, if inside/outside information is available, the outside sample can be preferred. We currently do this step in software, but a GPU-based implementation (eg. by rendering the model into each of the volume slices) would be straightforward if preprocessing time becomes an issue. The software approach requires between several seconds and a few minutes per billboard box, depending on the size of the original mesh. At runtime, the ray casting algorithm has to test whether the 3D texture contains an object part at the current sampling position or not, indicated by either a special color or by using the alpha channel.

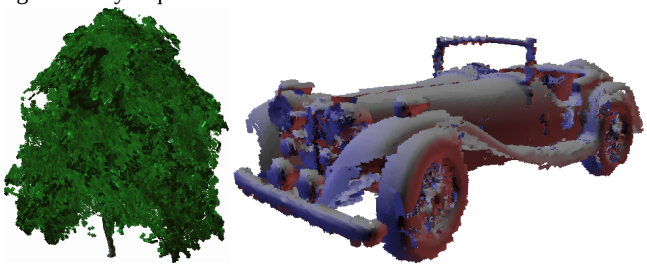
Although the obtained image quality is relatively high as also shown in Figure 4, c, the memory consumption is also high due to the typically large amount of empty texels. If the model should be dynamically illuminated at runtime, normal vectors can also be stored for every texel, again by averaging the normals of all surfaces that fall into a voxel or by selecting any of them. However, note that this further increases the required memory even though normals can

a. Original meshes.

b. BBC models (tree: 16 rectangles; car: 31 rectangles)).



c. Volumetric DMBBCs: the volumetric data is stored as 3D texture, resulting in good visual quality at the expense of relatively high memory requirements.



d. Shell DMBBCs: only one value per (u,v) position is stored. The tree still looks good except for the trunk, but the car shows many cracks and misses details at the front.

e. Thick shell DMBBCs: two values per (u,v) define an interval. The tree is distorted by many incorrect "slabs", but the jaguar model looks better than with a shell DMBBC.

Figure 4: Original, BBC, and DMBBC renderings of two models.

be stored with lower texture resolution and/or compressed to only 2 bytes per normal at the expense of slightly reduced output image quality. However, if certain assumptions about the input model can be made, the following two sections present ways how memory can be saved with little impact on the quality.

3.3.2 Shell Representation

If we assume that the volumetric displacement function contains only one occupied region (voxel) for each (u,v) texel, then $f_{uv}(w)$ can be more efficiently represented by a single displacement value. This dramatically reduces memory requirements. We call this a *shell DMBBC*.

The displacement value defines at what distance to the billboard rectangle the sample was acquired. Here the same problem occurs as for the 3d texture case: if multiple model parts map to the same (u,v) position, one has to decide wether to average over all height values, color values and normals or to just select one of them. Note that this issue is quite apparent since we deal with the whole thickness ε and not only with the thickness of one 3D texel as in Section 3.3.1.

In contrast to pure displacement maps, which represent surfaces, the geometry contained in a billboard box is not guaranteed to be contiguous. Therefore, a ray needs to be able to trace *through* holes in the representation. On the other hand, continuous surfaces should not be pierced. We solve this by assigning a user-defined *thickness* (therefore "shell" representation) to the samples. If this thickness is chosen too small, holes will appear in continuous surfaces, if it's too large, the model will look very blocky. At runtime, the ray casting algorithm tests whether the current sampling position is within the thickness distance to the displaced sample so that it can be assumed to be hit by the ray. Currently we adapt the thickness manually using visual inspection. It might also be defined as the standard derivation over all acquired texel positions.

A shell DMBBC basically needs the same amount of memory as a BBC, except that along with the color information an additional displacement value per (u,v) position must be stored. On the other hand, the overall image quality is typically lower compared to the 3D texture approach. Figure 4 d shows an example for this where the introduced error is not too apparent for a largely unstructured model (here a tree) but becomes too obvious for an object with continuous surfaces and wrinkled details (here a car). The latter problem is caused by the global "thickness" value that on the one hand has to ensure that continuous surfaces appear closed but on the other hand should also preserve small details. The next section presents a method that overcomes this problem.

3.3.3 Thick Shell Representation

A more accurate representation of the volumetric displacement function is the *thick shell DMBBC* representation, which stores *two* height values per (u,v) position. The interval between the two values represents the part of the volume that is occupied by the model. We have simply stored the lowest and the highest displacement values for generating Figure 4 e, but other options might also be possible. Consequently, at runtime the ray caster tests if the current w value is within the stored interval at the current (u,v) position in order to determine if the model is hit.

There are many possibilities to balance image quality and memory requirements in the actual memory layout of a thick shell. It would for example be possible to store colors and normal vectors separately for the lower and higher interval boundaries and to linearly

interpolate between the resulting output color values at runtime depending on the actual height. However, in order to save memory, for generating Figure 4 e we simply stored one additional height value compared to the shell DMBBC approach. This works well for relatively thin objects such as the trunk of a tree where the color can typically be assumed to be identical for both sides. This also saves almost 50% memory. However, as Figure 4, c also shows, while the jaguar model looks considerably less cracky compared to shell DMBBCs (see Figure 4, d), the chestnut shows many incorrect "slabs" because neighboring leaves and branches are merged into one billboard box. This makes this technique much less useful for highly unstructured models like vegetation.

3.4 Rendering optimizations

While the basic rendering algorithm was already described in Section 3.2, this section describes how to accelerate the rendering process (see Section 3.4.1) using distance functions and how to smoothly blend between DMBBCs and BBCs (see Section 3.4.2).

3.4.1 Rendering Acceleration Using Distance Functions

GPU-based ray casting can become very costly since it is linear in the number of texels the ray projects to in each texture rectangle. In order to increase rendering speed without reducing the image quality we adapted an approach by Donnelly et al. [Donnelly], which is based on *distance functions*. Please note that it works similarly for all DMBBC variants presented in Section 3.3.

In the preprocessing step, a so-called 3D *distance texture* is created. The (u,v) dimensions of this texture are the same as for the billboard boxes. The w dimension can be chosen as a tradeoff between fast intersection calculation and required memory. For every 3D position in the distance texture, the closest Euclidean distance to the next non-empty element within the billboard box (defined by the 3D texture, the shell, or the thick shell) is stored. Texels that lie within a non-empty region have a distance of zero. Note that the distances define spherical regions around every texel.

At runtime, instead of using uniform sampling (as defined in Section 3.2), we read the distance texture for the current texel and either stop (if the value is 0) or move along the ray to the border of the distance region and repeat. Therefore, the distance function can be used to efficiently skip large empty parts in a billboard box without missing an intersection. Also, since the traversal automatically stops within occupied voxels, the main loop of the traversal is very efficient, only consisting of a texture lookup and two arithmetic operations. By using this technique we obtained speedup factors between 2 to 10 for typical objects.

3.4.2 Blending between Representations

One problem when rendering different representations for an object at different viewing distances is to stage a smooth transition between those representations, i.e. to avoid noticeable popping artifacts which inevitably draws the observer's attention to any visual differences. Due to the construction of the underlying billboard cloud, it is possible to calculate at which distance the maximum displacement ε encoded in a DMBBC projects to less than one display pixel. Behind this distance rendering can simply be switched to regular BBCs for faster performance, without being too much noticeable.

However, if the transition should occur at closer distances (typically for performance reasons) a blending can be applied between the DMBBC and its corresponding BBC in the following way: the displacement (i.e., the height of the billboard box) is linearly reduced to zero before switching, thus flattening the billboard box to the according billboard rectangle, similar to the geomorphing approach for terrain rendering [Hoppe 1998]. This provides a seamless transition between the two representations and requires only to dynamically adjust the size of the box and the according world-to-texture-space-matrix.

Note that we do not blend between the original geometric model and the DMBBC, as visual artifacts should be quite reduced due to the similarity of the DMBBC with the geometric model.

4 Results

We have implemented the variants of the DMBBC algorithm described in Section 3.3 as HLSL Shader Model 3.0 pixel shader and tested them with a number of objects. All of the following tests were performed on an Pentium D PC running at 3.2GHz with 1GB of RAM and an ATI Radeon X1900 XT graphics card, running Windows XP.

Figure 1 and Figure 4 provide a visual comparison of the dragon, the jaguar and the chestnut tree model rendered as geometry, BBCs, and DMBBCs. Table 1 provides some details for these models. The dragon model was chosen because its billboard cloud representations are typically visually unpleasant due its curved surface, and the DMBBC representation offers a vastly improved representation. The chestnut tree is one of the target applications of the DMBBC algorithm; billboard cloud representations of trees have been used quite successfully, and we show that DMBBCs can provide a higher quality representation by eliminating the edge-on artifacts which are quite visible for trees (e.g., see Figure 4).

Figure 1 and Figure 4 demonstrate that the DMBBC algorithm is quite universally applicable. The comparison of various models rendered as BBC and DMBBC in Figure 5 illustrates how the DMBBC representation manages to preserve the general shape of the models more closely.

	faces	ε	(DM)BBC	generation time (s)	
model	original		rects/boxes	BBC	DMBBC
hippo	31583	10	10	130	303
moped	56882	5	22	834	1014
chestnut	159160	20	8	358	806
jaguar	188844	4	31	1129	2146
buddha	865792	10	8	1692	1794
dragon	871414	4	31	2799	2934

Table 1: Original and BBC/DMBBC figures for various models.

Concerning memory requirements, Table 2 shows statistics about the test models. While the volumetric DMBBC needs considerably more memory than the BBC, the shell and thick shell variants store the data more memory efficient. Also note that in Figure 1, a complex BBC still does not reach the visual quality of the according DMBBC, although it occupies even more memory. In general the tradeoff between a high image quality and low memory consumption must be made for a particular application.

The rendering speed of DMBBCs is fully determined by the pixel rendering power of the graphics hardware. DMBBCs therefore trade CPU and/or vertex processing power as well as CPU/GPU bandwidth for pixel processing speed. For Figure 6, we created

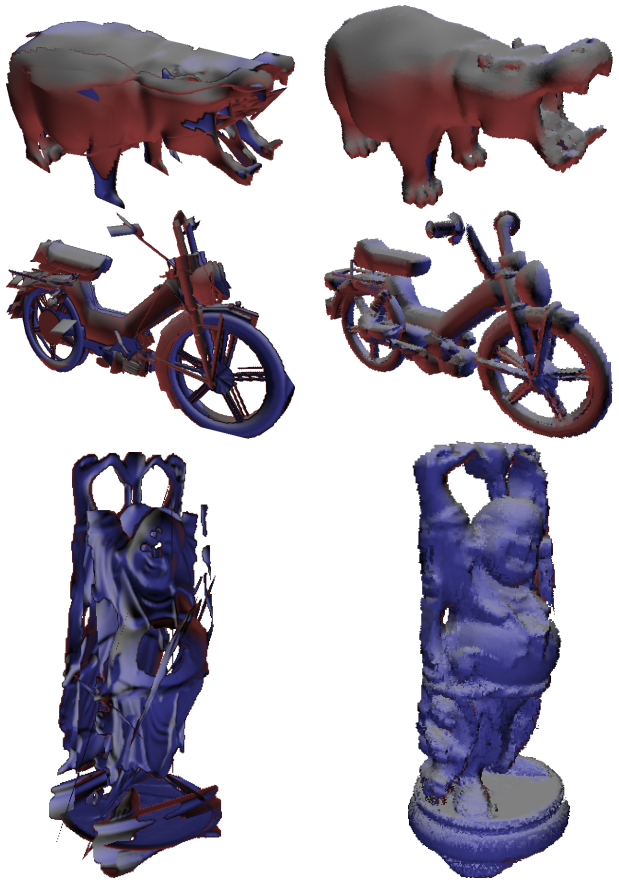


Figure 5: Left: BBCs of the hippo, moped and buddha. Right: the according DMBBCs.

	memory requirements (MB)			
		volumetric	shell	thick shell
model	BBC	DMBBC	DMBBC	DMBBC
hippo	0.37	12.2	0.76	0.93
moped	0.42	18.2	0.59	0.76
chestnut	1.85	17.25	2.67	3.1
jaguar	0.62	100	2.43	3.27
buddha	0.59	3.22	0.89	1.15
dragon	1.76	10.3	3.18	3.88

Table 2: Continuation of Table 1: memory requirements for the different representations, including normal data and distance textures for rendering acceleration.

BBC and volumetric DMBBC representations for the willow tree model with an error threshold of 20%. We then measured the average rendering time of a single such model for a number of viewpoints at distances from 500 to 5000 units in 20 unit increments. Furthermore, we performed the same test with the BBC representation of the same model, with the same error threshold as for the DMBBC, and the original geometry. Please note that the distance where the projected error threshold of the BBC becomes smaller than one pixel is at approximately 1300 units.

It can be observed that the frame rates for the full geometry and the BBC model are independent of the distance (and therefore independent of the screen size), leading to the conclusion that the GPU is certainly not fill rate limited for these models. The significantly lower performance of the visually equivalent BBC model is caused

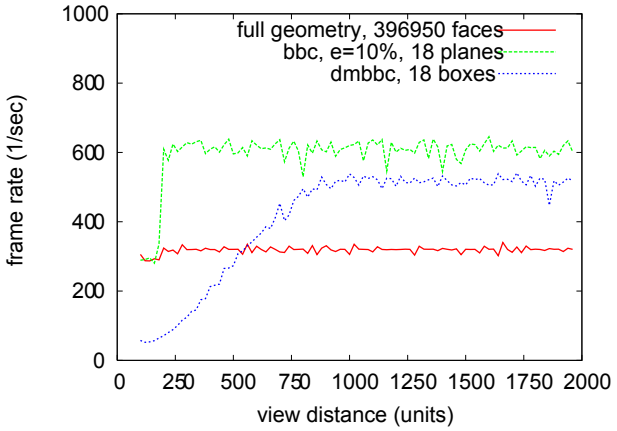


Figure 6: Frame rate vs. distance for full geometry, BBC and DMBBC representations of the willow tree model. Note that the top line (BBC) implies significant visual artifacts; rendering the BBC representation is therefore only useful once the visual error becomes sufficiently small (1 pixel at approximately 1300 units).

by the abundance of texture state changes required to render them—while the full geometry only uses two textures, the BBC representation requires a separate texture for each rectangle, causing a much larger overhead even if they are packed into larger texture atlases.

On the other hand, the rendering performance of a DMBBC model is highly dependent on the viewing distance. The lower performance of the more complex DMBBCs is mostly caused by pixel shader overdraw—since the renderer modifies depth information, no early culling may be performed and the full shader must be evaluated for each of the billboard boxes. It can be seen that although the initial performance is very low, the DMBBC renderer surpasses the performance of the original geometry rather quickly. Figure 6 illustrates that the DMBBC representation is ideally suited for accelerating rendering at moderate distances, and provides a good transition from full geometry to BBC representations. Also note that the rendering speed is practically equal for the three DMBBC variants described in Section 3.3 because the ray casting loop is almost identical.

Furthermore, since the performance of pixel processing within the GPU is increasing rapidly with each generation of GPUs (and GPU bandwidth less quickly by far), it can be expected that the slope of the DMBBC curves in Figure 6 will become larger, shifting the break-even point closer to the viewer.

5 Conclusions

We have introduced displacement mapped billboard clouds, a new image-based rendering primitive for complex objects. DMBBCs complement traditional billboard clouds for near to medium distances, where the visual shortcomings of BBCs are especially apparent and distracting. The main advantage of DMBBCs is that they allow an image-based representation to be used at closer distances than ordinary BBCs, leading to significantly higher rendering speeds than with the original geometry, while still providing good image quality. The improved image quality is due to the geometric detail added by image-based primitives, which also capture discontinuous geometry using so-called volumetric distance functions . The high rendering speed owes to the fact that a DMBBC

is displayed entirely by modern graphics hardware with a fast ray casting algorithm. Conventional shading and shadowing techniques like normal maps and shadow mapping for realistic illumination can still be used with the new technique. In addition, different objects can intersect each other, while the representations look still correct.

The scale of volumetric DMBBCs reaches from a purely *volumetric* representation (i.e., one billboard box for the whole model) to an almost purely *geometric* representation (large number of very thin billboard boxes). In other words, DMBBCs smoothly fill the gap between image-based and geometry-based representations.

In terms of future work, one interesting question is how to optimally simplify an object for displacement-mapped rendering. In our implementation we used the classical error metric of Decoret et al. [Decoret et al. 2003] based on the distance ε of a scene part to the rectangle it is mapped to. Although the displacement map practically eliminates the error measured by this metric, keeping the offsets small is still desirable in order to reduce overdraw in the output image. However, other metrics could adapt to the special characteristics of the DMBBC algorithm. For instance, while in shell DMBBCs, pixels can only adequately represent one surface piece, this does not hold true for the volumetric approach. New metrics can also be based on other optimization criteria, like using fewer boxes or making best use of memory by minimizing the number of empty pixels in every box.

References

ANDUJAR, C., BRUNET, P., CHICA, A., NAVAZO, I., ROSSIGNAC, J., AND VINACUA, A. 2004. Computing maximal tiles and application to impostor-based simplification. *Computer Graphics Forum 23*, 3, 401–410.

BEHRENDT, S., COLDITZ, C., FRANZKE, O., KOPF, J., AND DEUSSEN, O. 2005. Realistic real-time rendering of landscapes using billboard clouds. In *Computer Graphics Forum*, vol. 24, 507–516.

BLINN, J. F. 1978. Simulation of wrinkled surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 286–292.

DECORET, X., DURAND, F., AND SILLION, F. X. 2003. Billboard clouds. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, ACM Press, New York, NY, USA, 376–376.

DONNELLY, W. ch. Per-Pixel Displacement Mapping With Distance Functions, 123–136.

DUFORT, J.-F., LEBLANC, L., AND POULIN, P. 2005. Interactive rendering of meso-structure surface details using semi-transparent 3d textures. In *Proc. Vision, Modeling, and Visualization 2005*, 399–406.

FUHRMANN, A. L., UMLAUF, E., AND MANTLER, S. 2005. Extreme model simplification for forest rendering. In *Proceedings of the 2005 Eurographics Workshop on Natural Phenomena*, The Eurographics Association, E. Galin and P. Poulin, Eds.

HIRCHE, J., EHLERT, A., GUTHE, S., AND DOGGETT, M. 2004. Hardware accelerated per-pixel displacement mapping. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, Canadian Human-Computer Communications Society, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 153–158.

HOPPE, H. 1998. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *VIS '98: Proceedings of the conference on Visualization '98*, IEEE Computer Society Press, Los Alamitos, CA, USA, 35–42.

JESCHKE, S., WIMMER, M., AND PURGATHOFER, W. 2005. Image-based representations for accelerated rendering of complex scenes. In *EUROGRAPHICS 2005 State of the Art Reports*, The Eurographics Association and The Image Synthesis Group, Y. Chrysanthou and M.Magnor, Eds., EUROGRAPHICS, 1–20.

KOLB, A., AND REZK-SALAMA, C. 2005. Efficient empty space skipping for per-pixel displacement maps. In *Proceedings of the VMV 2005 Conference*.

MCGUIRE, M., AND MCGUIRE, M. Steep parallax mapping. *I3D 2005 Poster*.

MESETH, J., AND KLEIN, R. 2004. Memory efficient billboard clouds for btf textured objects. In *Vision, Modeling, and Visualization 2004*, Akademische Verlagsgesellschaft Aka GmbH, Berlin, B. Girod, M. Magnor, and H.-P. Seidel, Eds., 167–174.

OLIVEIRA, M. M., AND POLICARPO, F. 2005. An efficient representation for surface details. Tech. Rep. RP-351, Federal University of Rio Grande do Sul - UFRGS.

PAGLIERONI, D. W., AND PETERSEN, S. M. 1994. Height distributional distance transform methods for height field ray tracing. *ACM Trans. Graph. 13*, 4, 376–399.

POLICARPO, F., OLIVEIRA, M. M., AND COMBA, J. L. D. 2005. Real-time relief mapping on arbitrary polygonal surfaces. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 155–162.

POLICARPO, F. 2006. Relief mapping of non-height-field surface details. In *To appear in Proceedings of the 2006 Symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA.

PORUMBESCU, S. D., BUDGE, B., FENG, L., AND JOY, K. I. 2005. Shell maps. *ACM Trans. Graph. 24*, 3, 626–633.

T. KANEKO, M. INAMI, N. K. Y. Y. T. M. T. T., AND TACHI, S. Detailed shape representation with parallax mapping.

TATARCHUK, N. 2006. Dynamic parallax occlusion mapping with approximate soft shadows. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 63–69.

WAHL, R., GUTHE, M., AND KLEIN, R. 2005. Identifying planes in point-clouds for efficient hybrid rendering. In *The 13th Pacific Conference on Computer Graphics and Applications*.

WANG, L., WANG, X., TONG, X., LIN, S., HU, S., GUO, B., AND SHUM, H.-Y. 2003. View-dependent displacement mapping. *ACM Trans. Graph. 22*, 3, 334–339.

WANG, X., TONG, X., LIN, S., HU, S., GUO, B., AND SHUM, H.-Y. 2004. Generalized displacement maps. In *Proceedings of the Eurographics Symposium on Rendering 2004*, Springer-Verlag, 227–233.

WATT, A., AND POLICARPO, F. 2005. *Advanced Game Programming with Programmable Graphics Hardware*. A. K. Peters Limited. ISBN 156881240X.

WELSH, T. 2004. Parallax mapping with offset limiting: A per pixel approximation of uneven surfaces. Tech. rep., Infiscape Corporation.