International Journal of Image and Graphics
Vol. 4, No. 4 (2004) 1–29

World Scientific
www.worldscientific.com

DMESH: FAST DEPTH-IMAGE MESHING AND WARPING

RENATO PAJAROLA*, MIGUEL SAINZ† and YU MENG‡

School of Information and Computer Science,
University of California Irvine, CA 92697, USA
**pajarola@acm.org*
†*msainz@ics.uci.edu*
‡*ymeng@ics.uci.edu*

Received 15 February 2003
Revised 15 July 2003
Accepted 31 July 2003

In this paper we present a novel and efficient depth-image representation and warping technique called DMesh which is based on a piece-wise linear approximation of the depth-image as a textured and simplified triangle mesh. We describe the application of a hierarchical multiresolution triangulation method to generate adaptively triangulated depth-meshes efficiently from reference depth-images, discuss depth-mesh segmentation methods to avoid occlusion artifacts and propose a new hardware accelerated depth-image rendering technique that supports per-pixel weighted blending of multiple depth-images in real-time. Applications of our technique include image-based object representations and the use of depth-images in large scale walk-through visualization systems.

Keywords: Image based rendering; depth-image warping; multiresolution triangulation; level-of-detail; hardware accelerated blending.

1. Introduction

In recent years a new rendering paradigm called *Image Based Rendering* (IBR),[8] that is based on the reuse of image data rather than geometry to synthesize arbitrary views, has attracted growing interest. Since IBR works on sampled image data, and not on geometric scene descriptions, the rendering cost is largely independent of the geometric scene complexity, and depends mainly on the resolution of the sampled data. In fact, one of the goals of IBR is to de-couple 3D rendering cost from geometric scene complexity to achieve better display performance in terms of interactivity. Target applications include interactive rendering of highly complex scenes, display of captured natural environments, and rendering on time-budgets.

There exist many approaches to accelerate rendering for interactive navigation such as reducing the geometric scene complexity using different *levels-of-detail* (LODs) and exploiting frame-to-frame coherence. Although these approaches achieve significant improvements, they are still limited when the complexity of the

scene has increased far beyond the image-space resolution. In this case, IBR techniques offer advantages over traditional geometric rendering.

In this paper we expand on the technique of depth-image warping.[5,18] Images with depth per-pixel have been used to represent individual objects[17,20,31] or to approximate small parts of a large scene in interactive walk-through applications.[1,2,27,28,30,32] We present an alternative depth-image warping technique, called *DMesh*, based on adaptive triangulation, segmentation and simplification of the depth-buffer, and rendering this *depth-mesh* with the color texture of the input depth-image (see also Ref. 16). In addition to reducing the rendering cost from the geometric scene complexity to the resolution of the depth-images, adaptively triangulating the depth-map further reduces the rendering cost down to the complexity of the depth-variation within this image.

DMesh offers several improvements and alternatives compared to previous depth-image warping techniques. The main contributions include:

- A fast depth-buffer triangulation and simplification technique based on a hierarchical quadtree triangulation algorithm that performs adaptive (and view-dependent if desired) depth-meshing at interactive frame-rates for high-resolution depth-images (i.e. with 512^2 pixels or more).
- An efficient and simple multiresolution mesh segmentation method.
- Depth-image warping is efficiently performed by rendering a comparatively small and bounded set of textured triangle-strips instead of warping a large number of individual pixels.
- A novel algorithm for hardware accelerated per-pixel weighted blending of multiple reference depth-meshes in real-time.

Due to its efficiency, our approach is applicable in various rendering systems[1,2,28,30,32] which represent small parts of a large scene by image-based approximations and that dynamically update the image-based parts frequently at run-time.

The remainder of the paper is organized as follows. In Sec. 2 we briefly review related work in depth-image warping and Sec. 3 describes the basic quadtree triangulation approach used in DMesh. Section 4 describes our depth-meshing method, mesh segmentation is explained in Sec. 5, and Sec. 6 explains the rendering algorithm. In Sec. 7 we provide experimental results supporting our claims and Sec. 8 concludes the paper.

2. Related Work

The notion of *depth* per pixel has been introduced as disparity between images[5] and used for image synthesis by interpolation between pairs of input images. The depth information — distance from the center of projection along the view direction to the corresponding surface point — allows to re-project pixels from a depth-image to arbitrary new views. A unique evaluation order[18] guarantees correct back-to-front

drawing order when (forward) warping pixels from the input depth-image to the frame buffer of a new view.

A major problem of depth-image warping techniques are disocclusion artifacts in novel viewpoints that may occur due to regions being visible from the new viewpoint that were not visible from any reference viewpoint. This kind of exposure artifacts cannot be handled solely by algorithmic improvements in IBR methods but must be addressed by additional sample data. In general, if something is not visible in any of the input data sets — reference depth-images — it cannot be reconstructed. Storing multiple depth and color values per pixel has been proposed,[17] also in the layered depth-image (LDI) approach,[31] to cope with such disocclusion artifacts. Such LDIs are used together with an automatic preprocess image placement method to support interactive rendering at guaranteed frame rates,[2] and to represent objects using a bounding box of LDIs.[20] The idea of LDIs has further been extended to LDI-trees[3] for improved control over the sampling rate during image synthesis.

Exposure artifacts from reprojecting a depth-image to an output image with higher resolution, stretching the input samples over a wider output range, is another major problem of depth-image warping. This occurs when zooming in on the depth-image or viewing it from different angles. An efficient approach to cope with this type of exposure artifacts is to represent the depth-image as a triangulated surface.[16] The triangulated depth-buffer provides a connected surface approximating the 3D scene and supports automatic pixel interpolation for exposed or stretched regions due to parallax changes as well as hardware accelerated warping by rendering a textured triangle mesh.

Despite hardware accelerated rendering of textured depth-meshes, the performance of depth-image warping is still determined by the size of the reference images. Therefore, high-resolution depth-images of 512^2 pixels or more impose a strong limiting factor on the rendering speed. To reduce rendering cost, the segmented depth-buffer can be triangulated at a fixed minimum grid resolution.[33] Other approaches go a step further and propose an automatic simplification of the high-resolution depth-mesh.[6,7] Disocclusion artifacts can be addressed by the use of multiple layers of triangulated depth-images.[9,13]

In a similar way as our approach, the *view-based rendering* method[26] uses weighted depth-meshes as well as per-pixel quality factors and normalization to blend multiple reference views together to synthesize new images. In contrast to our DMesh approach, however, the generation of the depth-meshes is performed by manual selection of feature points and a Delaunay triangulation to approximate the depth field. Furthermore, their reference depth-mesh blending algorithm and the per-pixel normalization is performed in software.

As depth-images basically represent colored points in 3D they are related to point based approaches (i.e. Refs. 4, 11, 23–25, 29 and 35) which cope with exposure artifacts due to undersampling and zooming by rendering points as oriented disks, surface elements with non-zero extent. When rendered, the tightly packed surface elements appear to represent a smooth continuous surface.

3. Quadtree Triangulation

In DMesh we use the *restricted quadtree triangulation* (RQT) method[21,34] to generate a triangulated surface from the depth-buffer. Figure 1 shows the basic recursive quadtree subdivision and triangulation that introduces vertices from the grid in two steps.

To avoid cracks in the triangulated surface from unrestricted adaptive subdivision and triangulation of the quadtree as shown in Fig. 2, RQT subdivision is constraint such that the levels of adjacent quadtree nodes differ by at most one.

An efficient variation of this constraint is the dependency relation shown in Fig. 3.[14] This relation specifies for each vertex v on level l two other vertices v_a, v_b on level l as in Figs. 3(b) and (d), or on level $l-1$ as in Figs. 3(a) and (c), that

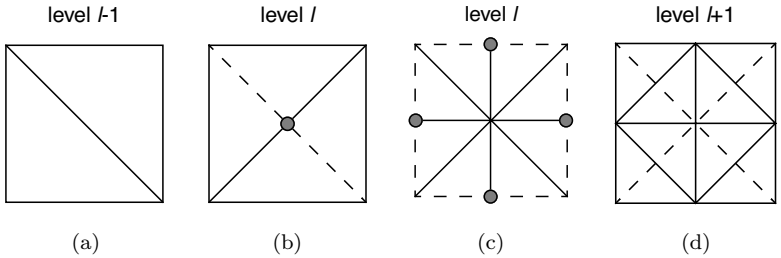


Fig. 1. Recursive quadtree subdivision and triangulation. Refinement points are shown as grey circles in (b) for a diagonal edge bisection and (c) for vertical and horizontal edge bisections. The bisected edges are denoted by dashed lines.

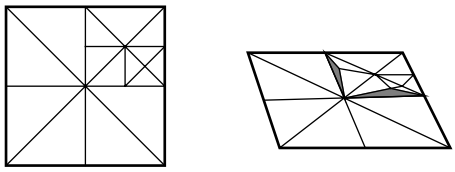


Fig. 2. Cracks (shaded in grey) resulting from an unrestricted quadtree subdivision.

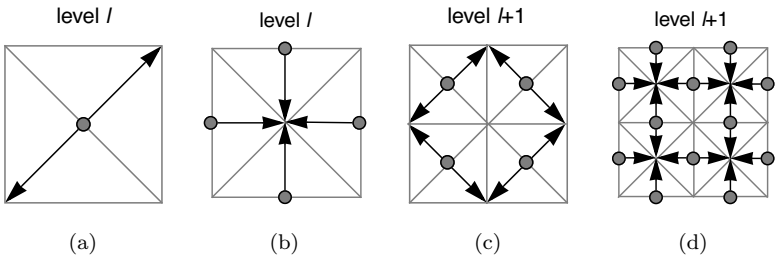


Fig. 3. Dependency relation of a RQT. The center vertex (a) depends on the inclusion of two corners of its quad region. The boundary edge midpoints (b) depend on the center vertex. Dependencies within and between the next higher resolution levels are shown in (c) and (d).

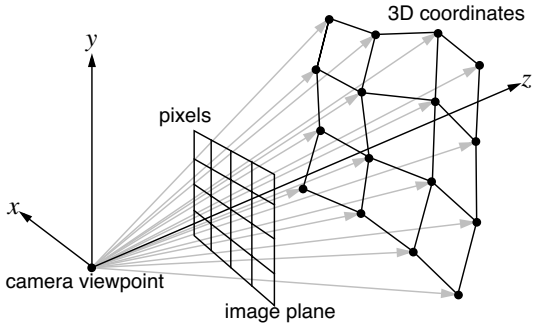


Fig. 4. Grid of depth-image pixels defines a projective height-field grid of 3D coordinates in the camera coordinate system.

must be included in the triangulation such that v itself can be selected without introducing a crack.

Note that this hierarchical triangulation allows the entire triangle mesh to be represented by one single generalized triangle strip[21] which we exploit for rendering efficiency.

4. Depth-Image Meshing

4.1. *Overview*

The depth values of a depth-image can be considered to be a 2.5-dimensional (projective) height-field with a regular grid structure similar to terrain elevation models. Given the depth values in the z-buffer for a particular reference image, we can calculate for each pixel (i, j) its corresponding 3D coordinate $P_{i,j} \in \mathbf{R}^3$ in the camera coordinate system as illustrated in Fig. 4. Using the coordinates of points $P_{i,j}$, a quadtree based multiresolution triangulation hierarchy as outlined in Sec. 3 can be constructed on the grid of pixels of the reference depth-image. We call this triangulation of a depth-image a *depth-mesh*, and the representation of an object by multiple depth-image triangulations a *depth-mesh object*. In this section we focus on how a single depth-mesh is initialized from a given depth-image, and how adaptively triangulated depth-meshes are generated efficiently.

Figure 5 illustrates the stages to generate the multiresolution depth-mesh data structures. After rendering a reference view, the depth-buffer has to be converted into 3D points given in the camera coordinate system. On the resulting grid of 3D points the quadtree hierarchy is initialized by computing an error-metric and quality measure for each point as described below. Furthermore, per-triangle segmentation is performed as outlined in Sec. 5.

4.2. *Initialization*

For a multiresolution triangulation using the method outlined in Sec. 3, an error-metric is required to determine the approximation error of a simplified triangle

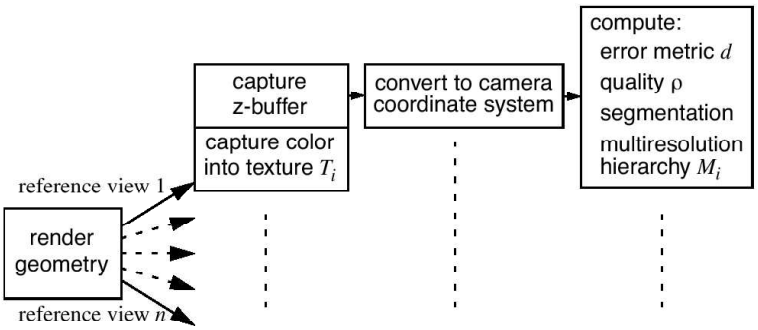


Fig. 5. Depth-mesh generation steps.

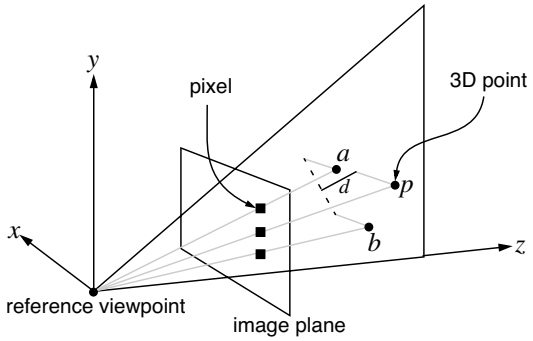


Fig. 6. Approximation error of vertical refinement point p is calculated as point-to-line distance in the projection on the y, z-plane.

mesh. As object-space geometric error metric we use a point-to-surface distance. The error of a refinement point is its distance along the elevation axis, the z-axis, to the refined edge, thus a point-to-line distance function. In general, for refinement points p bisecting an edge between points a and b the approximation error is calculated as the 3D point-to-line distance:

$$d = \frac{|(b-a) \times (b-p)|}{|b-a|} \,. \tag{1}$$

Equation (1) is used for refinement points bisecting a diagonal edge as in Fig. 1(b).

For performance reasons, vertical and horizontal refinement points as shown in Fig. 1(c) use an approximative 2D point-to-line distance function instead of Eq. (1). The approximation error d of a refinement point p, bisecting a vertical (or horizontal) edge between two points a and b is calculated as the 2D distance of p to the line $\overline{ab}$ in the orthogonal projection onto the y, z-plane (or x, z-plane). Figure 6 shows an example configuration of projecting points a, b and p, from vertically aligned pixels, onto the y, z-plane. Thus using the line equation $z = m \cdot y + b$ for $\overline{ab}$

projected onto the y, z-plane, with $m = (z_b - z_a)/(y_b - y_a)$, the approximation error for vertical refinement points (and analogously for horizontal refinement points) is given by:

$$d = \frac{m(y_p - y_a) - (z_p - z_a)}{\sqrt{1 + m^2}} \,. \tag{2}$$

For field-of-view angles of less than 50°, Eq. (2) introduces an error of less than $1 - \cos 25° = 9.4\%$.[a] Note that at most one third of the refinement points are center vertices bisecting a diagonal edge as shown in Fig. 1(b), all others are vertical or horizontal refinement points as in Fig. 1(c). Therefore, because less than 33% of points are diagonal refinement points, we achieve a significant speed-up using the approximate Eq. (2) instead of Eq. (1) for vertical and horizontal refinement points. Note also that working with squared distances d^2 eliminates the need to compute expensive square root operators in Eqs. (1) and (2).

To achieve a conservative monotonic error metric in the quadtree hierarchy, this distance error metric is maximized in such a way that center vertices (Fig. 1(b)) store the maximal error of all points within that subtree of the quadtree hierarchy.

Additionally for adaptive meshing and rendering purposes we compute and store the following information in the hierarchy. For each depth-mesh vertex p we store the surface normal n_p estimated from the depth-buffer, and we calculate a static per-vertex quality measure $\rho_p = |n_p \cdot v_p^{\mathrm{T}}|$ with the vector v_p being the direction from the camera to the point p in the local camera coordinate system. This quality metric ranks the fidelity of the sampling. Depth-mesh regions perpendicular to the view direction are of highest quality while areas parallel to the view direction are not well sampled. A per-vertex quality measure allows smooth interpolation and blending over depth-mesh triangles in contrast to a per-triangle quality measure.[7]

Note that the use of normals for each depth-mesh point allows interactive relighting of the displayed depth-meshes. Given that the reference views were rendered without illumination but only basic color texturing or per-vertex color not modified by any lighting calculations.

4.3. *Adaptive triangulation*

The multiresolution quadtree triangulation hierarchy imposed on the depth-buffer as explained above can be used in different ways to generate adaptively triangulated depth-meshes. The basic algorithm is to perform a recursive top-down traversal of the restricted quadtree triangulation hierarchy[21,34] and select points adaptively according to some error tolerance criterion. To guarantee a crack-free triangulation, for each selected point the dependent points are selected as well (see Fig. 3 and Refs. 14, 21). The so selected set of points define a conforming restricted quadtree

[a]Thus this could conservatively be taken into account and added to the result of Eq. (2) by multiplying with 1.1 if desired.

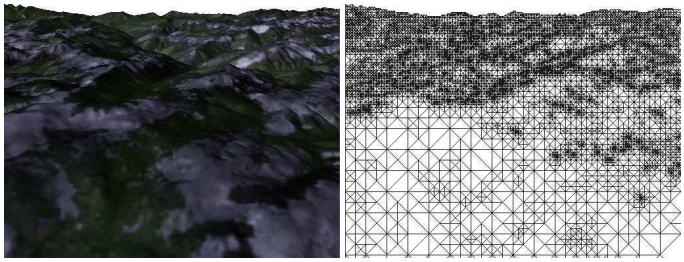


Fig. 7. Adaptively triangulated reference depth-image. Selected 32,961 triangles from the reference depth-image of $513 \times 513 = 263{,}169$ pixels.

triangulation which can be represented by a single triangle strip. This triangle strip can also be generated in a single top-down traversal.[21]

For rendering from novel viewpoints, we can distinguish between view-independent as well as view-dependent (Sec. 4.4) reference depth-mesh simplification. A view-independent depth-mesh simplification does not take the novel viewpoint into account for which depth-image warping is performed. We call such a simplification *static* since the reference depth-mesh has to be simplified only once and then can be warped to any novel view without re-creating it. In this case a reference depth-mesh is simplified according to an object-space geometric approximation error threshold ε. Hence given an error tolerance ε and since we initialized the error-metric as squared distances, a static simplified depth-mesh is extracted by selecting all points with

$$d_p^2 > \varepsilon^2 \,. \tag{3}$$

Figure 7 shows an example of a triangulated reference depth-image of size 513^2 pixels viewed from the actual reference viewpoint. While traditional depth-image warping methods have to warp 263,169 individual pixels, DMesh — using a small error tolerance — warps this reference depth-image to any new viewpoint by rendering 32,961 textured triangles (using hardware acceleration). This dramatically demonstrates the power of adaptive depth-mesh simplification.

4.4. *View-dependent triangulation*

In addition to static depth-mesh simplification, the quadtree triangulation hierarchy also allows to simplify a reference depth-mesh specifically for any given novel viewpoint which the reference view has to be warped too. Such a view-dependent simplification can be achieved by changing the point selection criterion to be dependent not only on the distance error-metric d_p of a depth-mesh point p but also taking into account the novel viewpoint location e. We call such a reference depth-mesh simplification *dynamic* because it has to be performed for every new viewpoint. Thus using an image-space approximation error threshold τ and given the viewpoint e, a depth-mesh point is selected if $f(d_p, p, e) > \tau$. We choose f to

be the geometric error d_p projected onto the image plane of the new view. Thus at rendering time, for each frame an adaptively triangulated depth-mesh can be extracted, given the viewpoint e in the depth-mesh's local camera coordinate system and an image-space error tolerance τ, by selecting all points with

$$\left(\frac{d_p}{|p-e|}\right)^2 > \tau^2 \,. \tag{4}$$

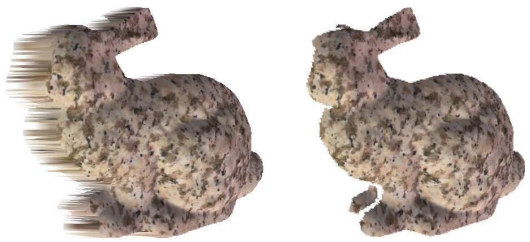
As outlined before, the vertex selection is performed recursively top-down in the quadtree hierarchy. In addition to the static simplification, the view-dependent simplification can also take view-frustum culling into account. Hence only points within the view-frustum and projected error exceeding the tolerance are selected. The vertex dependency rules (see Fig. 3 and Refs. 14, 21) are also adhered to and a triangle strip is implicitly generated using the selected vertices.[21]

5. Depth-Mesh Segmentation

5.1. *Overview*

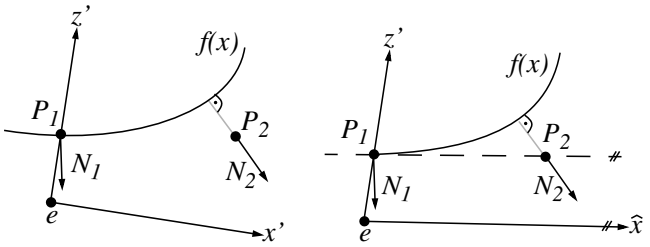
The triangulation of the z-buffer may introduce surface interpolations between different object surfaces and the background as shown in Fig. 8 that cause artificial occlusion or *rubber sheet*[16] artifacts when warped to new viewpoints. Therefore, it is important to determine whether triangles represent rubber sheets or not.

If not performed in a preprocess, this segmentation of the triangulated depth-mesh has to be done very quickly. Therefore, sophisticated range-image segmentation methods[15] cannot be applied since most of these approaches are far too slow for our purpose.[12] In DMesh, we perform an efficient per-triangle segmentation on the full resolution depth-mesh once during the depth-mesh initialization phase and propagate the results up the multiresolution hierarchy. We will briefly review fast segmentation alternatives such as *connectedness*[16] and *disparity*,[19] and then propose the simple *orthogonality* segmentation test. It is important to note here that any of these three methods can be used efficiently within our proposed multiresolution depth-meshing and warping approach.



(a) (b)

Fig. 8. Rubber sheet artifacts showing in (a) are removed in (b) by appropriate segmentation.



(a) (b)

Fig. 9. Basic quadratic curve fitting in (a), and modified sheared coordinate system in (b).

5.2. *Segmentation techniques*

The notion of *connectedness*[16] is defined to disambiguate the connected and disconnected mesh regions. Each triangle in the mesh is designated as either low-connectedness or high-connectedness, indicating whether or not the triangle is believed to represent part of a single actual surface. Given a surface normal for each pixel by interpolation, the connectedness between vertical or horizontal pixel neighbors is calculated by comparing the real z-difference with an estimated value from curve fitting.

Figure 9(a) illustrates the basic fitting of a quadratic curve $f(x) = A+Bx+Cx^2$ using the pixels $P_1 = (x_1, y_1, z_1)$ and $P_2 = (x_2, y_2, z_2)$ within a row of the depth-image, and given their corresponding normal vectors $N_1 = (nx_1, ny_1, nz_1)$ and $N_2 = (nx_2, ny_2, nz_2)$. Given the reference viewpoint e, the two points and normals are first transformed into the coordinate system with the z-axis aligned with the vector from e to P_1 and view-up vector being $(0, 1, 0)$. Using the normals to define the first derivatives $f'(x)$, the curve parameters can be evaluated to:

$$\begin{aligned} A &= f(0) = z_1\,, \\ B &= \partial f'(0) = \frac{nx_1}{-nz_1}\,, \\ C &= \frac{\partial f''(0)}{2} = \frac{\partial f'(x_2) - \partial f'(0)}{2x_2} = \frac{-nx_2/nz_2 - B}{2x_2}\,. \end{aligned} \tag{5}$$

We observed that endpoints of stretched edges have almost identical normals — steep rubber sheet triangles — if the normal is estimated from the z-buffer. This makes it easy to fit a quadratic curve according to Eq. (5) such that the points are on, or very close to the curve. To prevent the quadratic curve $f(x)$ from perfectly fitting vertices of rubber sheet triangles we used a sheared coordinate system as illustrated in Fig. 9(b) with the x-axis parallel to the edge from P_1 to P_2. Thus x_2 changes to $\hat{x_2} = |P_2 - P1|$.

The connectedness is then computed as the ratio of the absolute error in the range estimation to the parameter range $\hat{x_2}$:

$$r = \frac{|f(\hat{x_2}) - z_2|}{|\hat{x_2}|}\,. \tag{6}$$

If the ratio of Eq. (6) is greater than some threshold κ then the two pixels are considered to belong to different objects or surfaces. Triangles with such low connectedness $r > \kappa$ are removed from the depth-mesh. For each right-triangle in our depth-mesh we check the connectedness according to Eq. (6) for the endpoints of the vertical and horizontal edges and if one has low connectedness the triangle is removed. This connectedness is more computing intensive than other methods discussed below due to the fact that normals have to be computed for the points of the depth-image and the complexity of evaluating Eq. (6).

Also changes in *disparity*[19] values associated with pixels of the depth-image can be used to segment the reference image. Given the distance d of the image plane from the viewpoint (focal length), the disparity δ of a pixel with depth z is given by $\delta = d/z$. For three consecutive pixels (in a row or column of the depth-image) with $\delta_1, \delta_2, \delta_3$ and z_1, z_2, z_3 that are on a line or plane in 3D it holds that $\delta_2 - \delta_1 = \delta_3 - \delta_2$. As pointed out in Ref. 19 for curved surfaces, pixels form a piecewise bilinear approximation and are considered to represent a discontinuity if

$$|(\delta_2 - \delta_1) - (\delta_3 - \delta_2)| > \sigma \tag{7}$$

for a given disparity threshold σ. In that case the corresponding triangles are removed from the depth-mesh. Using $\delta_i - \delta_j = d \cdot (z_j - z_i)/(z_i z_j)$, Eq. (7) can be rewritten to:

$$\frac{d}{z_2} \cdot \left| \frac{z_1 - z_2}{z_1} - \frac{z_2 - z_3}{z_3} \right| > \sigma\,. \tag{8}$$

As pointed out in Ref. 19 Eq. (7) must vary the threshold parameter σ with the z-distance of the pixels (but is not further defined). As can be seen from Eq. (8), this adjustment is necessary because of the d/z_2 factor. In our implementation of this segmentation measure we drop this factor to get a projection-normalized decision inequality:

$$\left| \frac{z_1 - z_2}{z_1} - \frac{z_2 - z_3}{z_3} \right| > \sigma\,. \tag{9}$$

This modified disparity measure of Eq. (9) is an extremely simple measure, it only compares the difference in z-values between adjacent pixels. In our implementation of disparity, for each right-triangle in our depth-mesh we evaluate Eq. (9) for all three corners and if two or more exceed the disparity threshold σ we remove the corresponding triangle from the depth-mesh.

As an alternative to the connectedness and the disparity measures discussed above we propose an *orthogonality* test on triangles that is almost as simple to compute as disparity but semantically more powerful. We can observe that the rubber sheet triangles introduced during the triangulation of the depth-image have the following property: the triangle normal is almost perpendicular to the vector

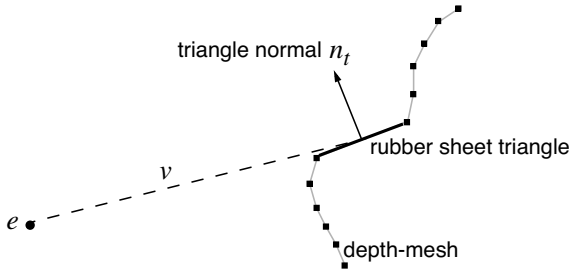


Fig. 10. Rubber sheet triangles in the depth-mesh.

from the viewpoint to the center of the triangle as shown in Fig. 10. Let v be the vector from viewpoint to the center of triangle t and n_t be the normal of t. The following inequality using an angular threshold ω can be used to determine if a triangle is a rubber sheet triangle:

$$\left| \frac{v}{|v|} \cdot n_t \right| < \cos(90^\circ - \omega)\,. \tag{10}$$

Compared to the disparity test of Eq. (9) which only compares z-ranges of two adjacent points, the orthogonality test of Eq. (10) is more powerful since it considers the orientation of a plane in 3D space with respect to the viewpoint.

Note that the proposed orthogonality segmentation criterion (and other segmentation methods as well) may also remove depth-mesh triangles that truly represent object surfaces nearly parallel to the view projection. Even though not intended, this is not considered to be a major drawback since these object surfaces are extremely undersampled in that case. Like other disocclusion artifacts, such strong undersampling is better solved by use of additional sample data, thus using more reference images.

In all segmentation methods, additional care should be taken not to remove very small triangles which represent rough surface features but do not constitute a discontinuity. Therefore, in addition to Eqs. (6), (9) and (10) we also consider the depth-range Δz of triangles at distance z in the camera coordinate system and we only remove triangles which span a depth-range larger than some threshold λ:

$$\frac{\Delta z}{z} > \lambda\,. \tag{11}$$

5.3. *Multiresolution segmentation*

The results of the segmentation process as described in the previous section, if a triangle is considered a rubber sheet, are computed at initialization-time of a reference depth-mesh at full resolution. These segmentation results at the leaf level of the quadtree triangulation hierarchy are then propagated upwards to the parent nodes. The segmentation result are stored as an 8-bit boolean flag S for the triangles a_1, a_2, b_1, b_2, c_1, c_2, d_1, d_2 of each quadtree node as shown in Fig. 11(a). The

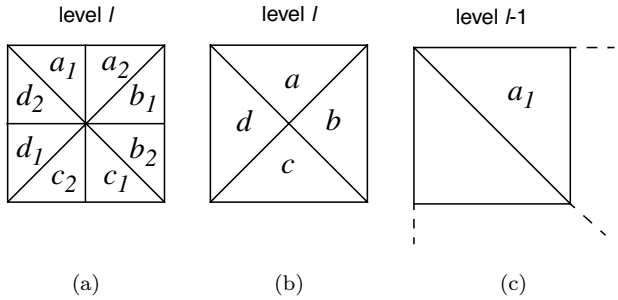


Fig. 11. (a) Segmentation flags for the full-resolution triangles of a quadtree node are stored as an 8-bit boolean field, and (b) for the simplified triangles are expressed as boolean or combinations. (c) Flags on level $l-1$ are recursively calculated from level l.

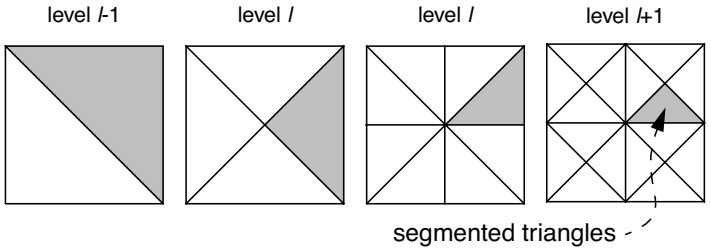


Fig. 12. Segmented triangle (on level $l+1$) causes all parent triangles (on levels l and $l+1$) to be segmented too.

same quadtree node, however, can also be triangulated with fewer triangles as in Fig. 11(b). For these aggregate triangles a, b, c or d the segmentation is derived from the eight sub-triangles. For an aggregate triangle $t = a$ (as well as b, c or d) the segmentation is given by $S(t) = S(t_1) \vee S(t_2)$. Similarly, the parent node on level $l-1$ determines the segmentation flags from the aggregate triangles of its child node on level l. For example, the segmentation flag of triangle a_1 on level $l-1$ in Fig. 11(c) is set to $S(a_1^{l-1}) = S(a^l) \vee S(b^l)$ according to the aggregate triangles a and b in the child node on level l.

Thus the segmentation of one triangle causes all parent triangles in the quadtree hierarchy to be segmented as well. Figure 12 illustrates this behavior of a segmented triangle that causes all parent triangles to be segmented too.

After the initialization of the segmentation at full resolution and the propagation to coarser levels in the multiresolution hierarchy, the segmentation of an adaptively simplified depth-mesh can then be determined extremely efficiently for each quadtree node by a simple boolean expression. This works effectively for both static as well as dynamic view-dependent depth-mesh simplification.

In DMesh, the removal of segmented triangles is directly integrated with the triangle strip generation. As mentioned before, given a set of selected points that specify a conforming restricted quadtree triangulation, a recursive top-down

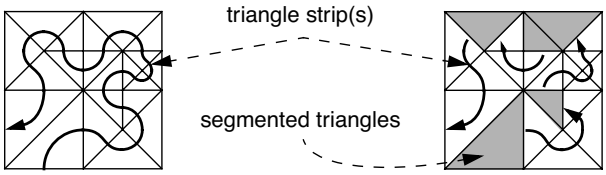


Fig. 13. (a) Triangle strip sequence of an adaptively triangulated quadtree, and (b) split into multiple shorter strips due to segmentation of triangles.

quadtree traversal[21] allows to incrementally generate a single generalized triangle strip,[b] see also Fig. 13(a). In DMesh this process is modified to take the segmentation flags of each quadtree node into account. While incrementally creating the triangle strip sequence the segmentation flags of each visited quadtree node are checked, and if necessary the strip is broken up into multiple smaller triangle strips as illustrated in Fig. 13(b). Using this procedure the initial single triangle strip is almost arbitrarily broken into smaller strips. However, as shown in Sec. 7 it still maintains a stripification that is superior to an indexed triangle list in terms of rendering performance.

6. Depth-Mesh Rendering

6.1. *Overview*

In DMesh, depth-image warping is efficiently performed by hardware supported rendering of textured polygons instead of projecting individual pixels from a reference depth-image to new views. The approximate depth-image consisting of a segmented triangulation of the depth-buffer, as outlined in the previous sections, is rendered using the color values of the reference frame-buffer as texture. Rendering a depth-image with a resolution of $2^k \times 2^k$ pixels involves warping of 2^{2k} pixels with traditional depth-image warping techniques (or rendering about $2 \cdot 2^{2k}$ triangles[16]). With the proposed technique, instead of warping the $2^{18} = 262{,}144$ pixels of a 512×512 reference image, a textured depth-mesh with only a few thousand textured triangles can be rendered using hardware accelerated 3D graphics at a fraction of the cost of per-pixel depth-image warping.

The depth-mesh initialization, segmentation and adaptive triangulation as outlined in Secs. 4 and 5 is performed in the reference view coordinate system. Whenever a depth-mesh has to be rendered, the coordinate system transformation of that reference view is used as local model-view transformation to place the depth-mesh correctly in the world coordinate system.

To support interactive relighting in depth-mesh rendering, the reference views should initially be captured without any illumination enabled. Thus only basic color

[b]Generalized triangle strips may include swap operations.

texturing or per-vertex colors should be enabled when rendering the reference views without applying any modulation from lighting calculations. This way, the color frame-buffer of a reference view contains the "flat" unlit colors. Using these flat colors as texture and using normals at each depth-mesh point, lighting is applied when rendering the depth-meshes for synthesis of new viewpoints.

To reduce disocclusion artifacts, most image warping techniques render multiple reference images that have to be merged to synthesize a new view. We present a novel and highly efficient blending algorithm that exploits graphics hardware acceleration and that supports per-pixel weighted blending of reference depth-images. Blending of n reference depth-meshes to synthesize a new view consists of the following basic steps:

(i) Select n reference depth-meshes M_i $(i = 1 \dots n)$ and textures T_i to be used for the current view, and calculate their positional blending weights w_i with respect to the current viewpoint e.
(ii) Adaptively triangulate the depth-meshes M_i for the current viewpoint e, and generate the segmented triangle strip representations S_i. (If static depth-meshes are used this triangulation is only performed once at depth-mesh initialization, see also Sec. 4.)
(iii) Render the triangle strips S_i without illumination and texturing to initialize the z-buffer Z_e for the current viewpoint e.
(iv) Render the triangle strips S_i again with their textures T_i and per-vertex quality ρ as alpha values enabled.[c] The result is rendered into separate color-with-alpha frame-buffers C_i. Depth-buffer evaluation using Z_e is set to read-only at this stage.
(v) Synthesize the new image I from buffers C_i using positional weights and alpha-blending:

$$I = \sum_{i=1\dots n} w_i \cdot C_i \,.$$

(vi) The image I contains the per-pixel weighted result. Note that the final alpha blending factor per pixel may be less than 1.0 at this stage and a normalization of the corresponding color yields the final image.

Figure 14 illustrates the data flow and rendering stages of our algorithm. The following sections explain the different stages in more detail and show how our algorithm exploits hardware acceleration.

Without restricting its generality, the proposed rendering algorithm is tested and explained in more detail below for an example *image-based object* (IBO) representation but is applicable to other rendering systems that make use of depth-image warping as well. An IBO[20] is constructed by generating six layered depth images

[c]The per-vertex quality measure ρ will be Gouraud interpolated across triangle faces and yield a per-pixel quality measure in the alpha-channel.

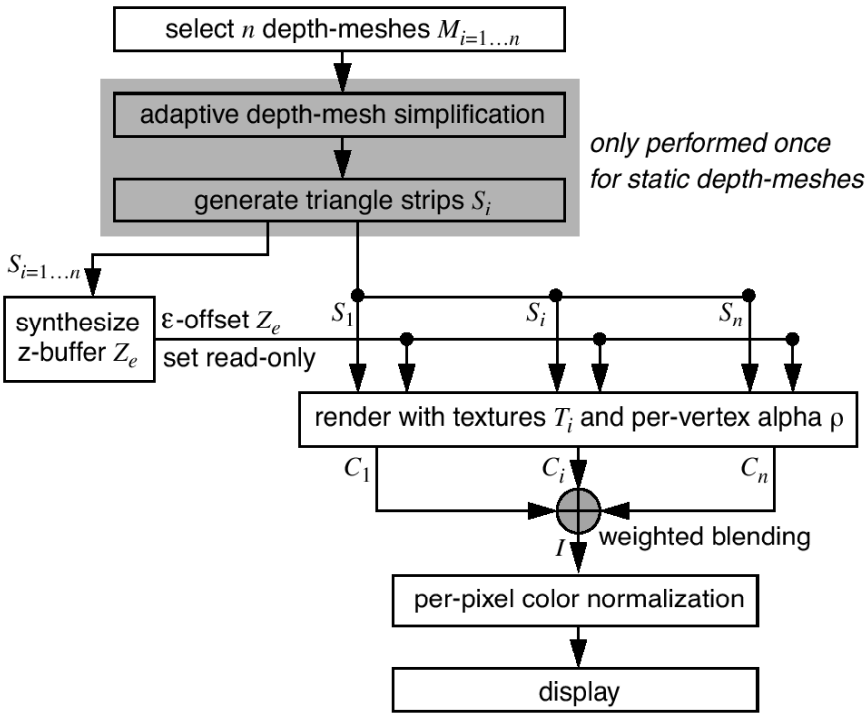


Fig. 14. Depth-mesh rendering and blending stages.

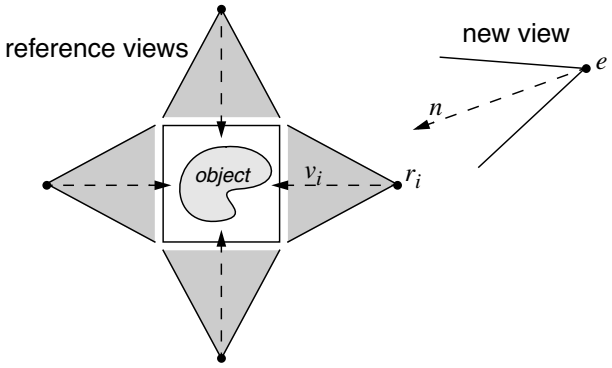


Fig. 15. Selection of reference views is dependent on geometric configuration of the new viewpoint e and its view direction n with respect to the reference view points r_i and directions v_i.

taken from the six sides of a cubic bounding box surrounding a single object. Similarly, we extract six textured depth-meshes around the axis-oriented bounding box of the object and store them as the reference views.

6.2. *Depth-mesh selection and triangulation*

During rendering, for each frame DMesh selects the most appropriate depth-meshes M_i to be used for the current view. For a depth-mesh object as explained above, we select up to five depth-meshes M_i out of the six reference views that may be visible from the novel rendering viewpoint e, see also Fig. 15. A reference view i is selected if the angle $\beta_{v_i,n}$ between the reference view direction v_i and the new view direction n is less than 135°. Based on this angle, an *angular weight*

$$w_i(\beta_{v_i,n}) = \begin{cases} 1.0 & \beta_{v_i,n} < 45° \\ \cos(\beta_{v_i,n} - 45°) & \beta_{v_i,n} \geq 45° \end{cases}$$

is assigned to the depth-mesh M_i. This angular blending weight avoids popping artifacts and guarantees smooth interpolation between contributing depth-meshes when rotating around the image-based object.

Furthermore, an additional *positional weight* $w_i(d_{r_i,e})$ based on the euclidean distance $d_{r_i,e} = |e - r_i|$ from the reference depth-mesh viewpoint r_i to the novel viewpoint e is associated with depth-mesh M_i. The final blending weights w_i are computed as $w_i = w_i(\beta_{v_i,n}) \cdot w_i(d_{r_i,e})$ and are then normalized such that $\sum_i w_i = 1.0$. Given a continuous motion of the viewpoint, the above weight factors also change smoothly and provide smooth blending changes during animation.

For static depth-mesh simplification, the depth-meshes M_i are adaptively triangulated only once at initialization time according to an object-space error threshold ε (see Sec. 4.3). Otherwise, each selected depth-mesh M_i is simplified view-dependently for the given new viewpoint e according to an image-space geometric error tolerance of τ pixels (see Sec. 4.4). The adaptively triangulated depth-meshes M_i are segmented and represented by a set of triangle strips S_i which contain vertices with texture coordinates into the reference view color texture T_i and with per vertex quality values ρ (in the RGBA color components).

6.3. *Rendering using an ε-z-buffer*

To achieve a smooth blending between overlapping depth-meshes M_i the rendering must allow some tolerance in the z-buffer visibility test. Multiple depth-meshes that for a pixel cover the same surface region within some tolerance ε should be blended together. Only if the z-buffer values are sufficiently different, larger than ε, the front-most depth-mesh determines the final color of that pixel, see also Fig. 16. The blending of overlapping depth-mesh parts does not constitute a simple blurring of available image information. In fact, in the overlap region the depth-meshes capture

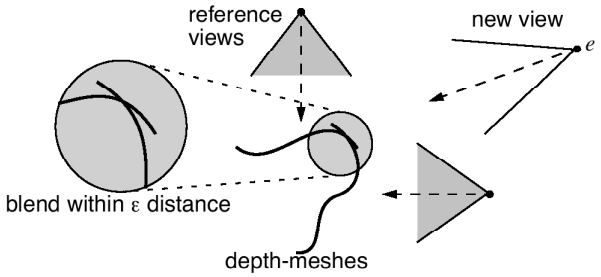


Fig. 16. ε-z-buffering of multiple reference depth-meshes.

the same object surface colors. Furthermore, a quality measure is assigned to each point of a depth-mesh as outlined in Sec. 4.2. A weighted blending that takes the relative positions of the reference views as well as the quality of the depth-image samples into account does not represent a blurring but favors the best possible information for image synthesis.

This ε-z-buffer rendering is achieved by first drawing the depth-mesh triangle strips S_i without any shading, illumination or texturing enabled to initialize the z-buffer to Z_e for the desired viewpoint e. Since the buffers require to be cleared during the following steps, the stencil buffer is set to store which areas of the frame buffer will be overwritten by the rendered depth-meshes. From here on we will assume that the stencil test is activated to block rendering in areas of the image where the depth-meshes do not cover any part of the new view.

In a second pass and using the previously computed z-buffer Z_e in read-only mode, the meshes S_i are rendered again into individual color buffers C_i with a small negative offset ε along the view-direction. To initialize the background in the stencil area of the depth-meshes for each buffer C_i, a background quad is rendered with its alpha channel set to the corresponding blending weight w_i. After this, illumination, shading, alpha-blending, texturing and per-vertex color components are enabled and the S_i are rendered a second time. Since each vertex RGBA color is set to its quality measure (ρ, ρ, ρ, ρ) illumination, Gouraud shading and texture modulation with the depth-meshs texture T_i will result in per-pixel weighted colors of the warped depth-mesh in C_i. (Technically, at this point the rendered color buffer C_i is copied into texture memory for further processing as outlined in the next section.) If not using any per-vertex quality measure, (ρ, ρ, ρ, ρ) can be set to a fixed constant which will result in a positional only weighting. Furthermore, uniform weighting can be achieved by setting all w_i to the same constant value.

At this point every pixel p in the frame buffer C_i with interpolated quality $\overline{\rho_p}$ from lighting, Gouraud shading and texture coordinates s, t will finally store the desired weighted and lit color $(w_i \cdot \overline{\rho_p} \cdot \mathrm{Red}(T_i(s,t)), w_i \cdot \overline{\rho_p} \cdot \mathrm{Green}(T_i(s,t)), w_i \cdot \overline{\rho_p} \cdot \mathrm{Blue}(T_i(s,t)), w_i \cdot \overline{\rho_p})$, or short $(w_i \cdot \overline{\rho_p} \cdot \mathrm{R}_p, w_i \cdot \overline{\rho_p} \cdot \mathrm{G}_p, w_i \cdot \overline{\rho_p} \cdot \mathrm{B}_p, w_i \cdot \overline{\rho_p})$.

Note that rendering the selected depth-meshes S_i twice is not very costly as demonstrated by our experiments in Sec. 7 because rendering of textured triangle strips is extremely efficient.

6.4. *Blending and normalization*

As outlined above, the images C_i now contain the quality and positional weighted contributions of the selected depth-meshes M_i. The final rendering stages must now perform the image composition and normalization of the color values. The two steps involved are the summation of the weighted colors by $I = \sum C_i$ followed by normalizing each color component. The image composition operation can be performed efficiently by alpha-blending n quadrilaterals using C_i as their texture maps.

The accumulation of images C_i yields an image I with intermediate pixel colors $c' = (\alpha \cdot R, \alpha \cdot G, \alpha \cdot B, \alpha)$, the α component contains the accumulated blending weight. These color values constitute the proportionally correctly blended values. However, the α values are not necessarily 1.0 as required. To get the final desired color $c = (R, G, B, 1.0)$ each color component of c' has to be multiplied by α^{-1}. This normalization is performed as an image post-process stage on the intermediate blending result I.

Without any hardware extensions to perform complex per-pixel manipulations this normalization step has to be performed in software. However, widely available graphics accelerators now offer per-pixel shading operators that can be used more efficiently. In our current implementation, we perform this normalization in hardware using nVIDIAs OpenGL *Texture Shader*[10] extension.

To compensate the color intensity deficiency we perform a remapping of the R, G and B values based on the value of α. During initialization time we construct a texture encoding in (s, t) of a look-up table of transparency and luminance values respectively, from 0 to 256 possible values. The pixels of this texture encode the new intensity for a given luminance t compensated with the α transparency s.

Based on this alpha-luminance map, we proceed to correct each of the R, G and B channels of every pixel. Using nVIDIAs texture shader extension operation GL_DEPENDENT_AR_TEXTURE_2D_NV the graphics hardware can remap the R and α by a texture lookup with coordinates $s = a$ and $t = R$ into our alpha-luminance map. At this point, rendering a quadrilateral with the intermediate image I as texture-one and the alpha-luminance map as texture-two, and setting the color mask to block the G, B and α channels will compensate the red channel by α^{-1} and store it in a new image I_R. Note that only the R and α channels are used by this dependent texture replace operation. Therefore, we need to remap the G and B channels to the R channel of two additional buffers I_G and I_B while copying the α channel as well. This is done by rendering two quads and using nVIDIAs register combiners. Then the dependent texture replace operation is also performed

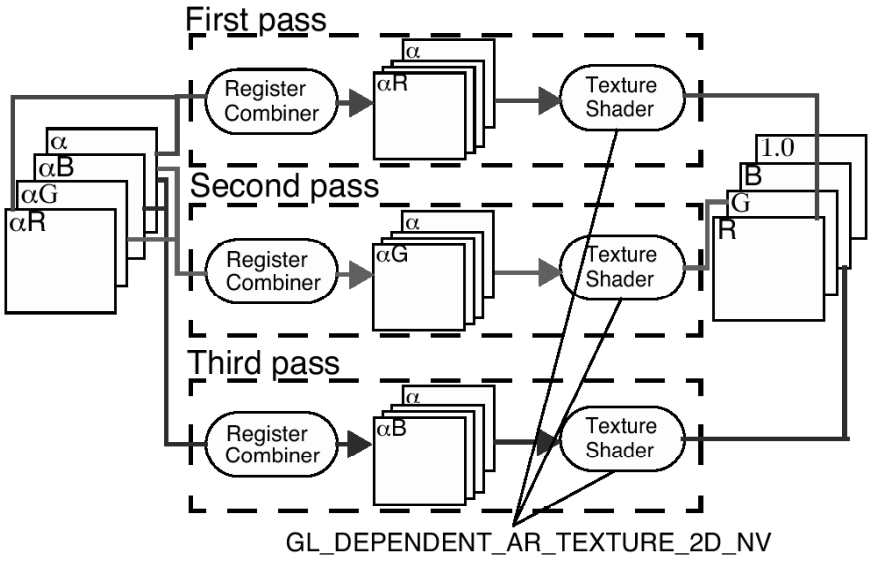


Fig. 17. Per-pixel normalization process.

on the images I_G and I_B. Thus by separating the RGB channels into three different images and using the αR-dependent texture replace operation we get the corrected RGB values (in the *red* channel of three new images I_R, I_G and I_B). Figure 17 illustrates this normalization process. Finally the three separated and corrected color channels are composited into the final image again using nVIDIAs register combiners by rendering three quads with textures I_R, I_G and I_B.

7. Experiments

In this section we report experimental results of our depth-meshing and warping algorithm. As an example to demonstrate the approach we use an image-based object (IBO) which consists of a bounding box of six depth-meshes around the object as outlined in Sec. 6. All experiments were performed on a a Dell 1.5GHz Pentium4 with nVIDIA GeForce4 4600Ti graphics card.

Using per-vertex depth-mesh normals and unlit captured reference images we perform full interactive relighting with diffuse as well as specular components when rendering the depth-meshes. This real-time lighting calculation is included in all frame rate timings reported in this section.

7.1. *Depth-mesh initialization*

Table 1 shows timing results for the generation of depth-meshes with our approach given a depth-image size of $513^2 = 263{,}169$ pixels. The table shows the size of the test models, the time it takes to render the geometry, the cost to capture

Table 1. Depth-mesh construction cost for different polygonal models. Given is the model size, geometric rendering cost, z-buffer capture and conversion, and multiresolution model construction.

Model	Triangles	Render	z-buffer	Quadtree
David	8,254,150	5862 ms	38 ms	641 ms
DHead	4,000,885	3731 ms	40 ms	660 ms
Dragon	871,414	634 ms	40 ms	650 ms
Female	605,086	688 ms	37 ms	654 ms

and convert the z-buffers into 3D points, and the cost to initialize the six depth-meshes M_i around the object. This initialization includes the calculation of the error metric, per-vertex quality measure, triangle segmentation flags, and other multiresolution parameters as described in Sec. 4. In the case of static depth-meshes the initialization time will also include the adaptive vertex selection and generation of triangle strips S_i for a given object-space error tolerance ε.

Not shown due to negligible cost is the capturing of the color textures T_i corresponding to particular depth-meshes M_i. The segmentation angle tolerance ω was set to 5° and the depth-range threshold λ to 5/100 of the objects maximal extent. As expected, one can observe that the depth-mesh initialization is constant for a given reference image resolution. The quadtree hierarchy construction grows with $O(n \cdot \log n)$ for a reference depth-image of n pixels.

As can be seen from Table 1, the initialization of a depth-mesh structure is very efficient. In this case the timing includes generating six depth-meshes around an object. The timings show that the proposed depth-meshes can indeed be generated quickly enough to be used in rendering systems[1,2,28,30,32] where they have to be updated dynamically at run-time. The major costs are rendering the actual geometry (not really part of the depth-mesh generation) and the construction of the depth-mesh quadtree triangulation data structure. For a single 513^2 depth-image the initialization takes about 150 ms.

We also performed several tests using the different segmentation methods outlined in Sec. 5.[22] In this paper we only want to summarize these results as the choice of segmentation method is independent of the algorithms and data structures of DMesh. The connectedness measure is by far the most expensive of the three approaches, while disparity and orthogonality are close in time cost with disparity being the fastest method of all. As can be seen in Fig. 18, despite its processing cost the connectedness based segmentation is not very good and the disparity or orthogonality tests provide much better segmentation at lower cost.

7.2. *View-independent depth-mesh rendering*

In Table 2 we report the rendering performance of DMesh using static, view-independently simplified depth-meshes. We used an object-space error tolerance ε of 1/1000 of the length of the bounding box diagonal. Selection cost and weight calculation of reference views was omitted due to negligible cost. Adaptive triangulation

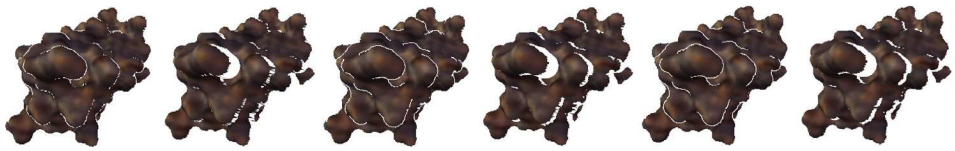


Fig. 18. Segmentation results using connectedness, disparity and orthogonality test (image pairs from left to right), displayed from the actual reference viewpoint as well as slightly rotated.

Table 2. Static DMesh rendering performance of image based objects. It shows the total number of triangles rendered for the selected depth-mesh triangle strips S_i and the timing of the different rendering stages.

Model	Geometry		DMesh				
	Time	FPS	Triangles in S_i	Synthesize z-buffer	Rendering of C_i	Blend and normalize	FPS
David	4247 ms	0.24	185,173	2.32 ms	3.35 ms	0.17 ms	157
DHead	5595 ms	0.18	382,356	4.50 ms	9.40 ms	0.16 ms	68
Dragon	633 ms	1.6	176,136	2.4 ms	3.4 ms	0.15 ms	165
Female	1029 ms	0.97	335,050	3.2 ms	6.4 ms	0.19 ms	98

of the reference depth-meshes (Step 2 from Sec. 6.1) has been performed once at initialization of the reference views and is not included in these timings. The cost of generating the z-buffer Z_e includes rendering the segmented triangle strips S_i of all selected depth-meshes once (Step 3 from Sec. 6.1). The rendering of images C_i corresponds to Step 4, and blending includes Steps 5 and 6 from Sec. 6.1.

As one can see from Table 2, the static DMesh depth-meshing and warping algorithm provides stable and very high frame rates for arbitrary complex objects, and that the rendering performance is completely independent from the size of the input polygonal model. The total depth-mesh rendering and blending cost is in the order of 5 to 15 ms. The weighted blending of the individual warped depth-images C_i is extremely efficient and scales to arbitrary number of input images C_i as long as the pixel fill rate of the graphics hardware is not exceeded. The average length of triangle strips for these depth-meshes range from 20 to 40 triangles per strip.

Images of static DMesh rendering are shown in Fig. 19, comparing the geometric rendering (left) to a synthesized view from multiple reference depth-meshes (right). Each color in the occupancy images (bottom row) illustrates the use of different depth-meshes by varying colors and shows any combination of two or more overlapping depth-meshes by coloring this area orange-brown.

7.3. *View-dependent depth-mesh rendering*

Table 3 shows the rendering performance of view-dependent depth-mesh image based objects, and corresponding screenshots are given in Fig. 20. We used an image-space error tolerance τ of 1 pixel. The selection of the reference views has negligible costs and is omitted here. The synthesis cost of generating the z-buffer Z_e

Fig. 19. Static DMesh rendering of complex polygonal objects. Left image shows the actual polygonal object and the right image shows the weighted blending of multiple textured depth-meshes. Various blending combinations of different depth-meshes are shown on the far right by different shading (pseudo colors), darkest shade (orange-brown) denotes two or more overlapping depth-meshes.

Fig. 20. View-dependent DMesh rendering of complex polygonal objects. Left image shows the actual polygonal object and the right image shows the weighted blending of multiple textured depth-meshes. Various blending combinations of different depth-meshes are shown on the far right by different shading (pseudo colors), darkest shade (orange-brown) denotes two or more overlapping depth-meshes.

Table 3. View-dependent DMesh rendering performance of image based objects. It shows the total number of triangles rendered for the selected depth mesh triangle strips S_i and the timing of the different rendering stages.

Model	Geometry		DMesh					
	Time	EPS	Triangles in S_i	Synthesize z-buffer		Rendering of C_i	Blend and normalize	EPS
				Selection	Rendering			
David	4268 ms	0.23	61,190	41.0 ms	59.0 ms	9.55 ms	0.16 ms	9.7
DHead	5548 ms	0.18	151,961	90.4 ms	122 ms	20.8 ms	0.16 ms	4.6
Dragon	639 ms	1.5	73,456	56.6 ms	78.9 ms	13.8 ms	0.15 ms	6.9
Female	1028 ms	0.97	107,678	65.9 ms	88.7 ms	15.7 ms	0.16 ms	6.1

includes the view-dependent selection of vertices from the depth-meshes M_i, as well as rendering the segmented triangle strips S_i once (Steps 2 and 3 from Sec. 6.1). The rendering of images C_i corresponds to Step 4, and blending includes Steps 5 and 6 from Sec. 6.1.

It can be observed from Table 3 that the view-dependent DMesh rendering method also provides interactive frame rates for very complex objects. The total depth-mesh rendering and blending cost is in the order of 100 to 200 ms. The average length of triangle strips in S_i for these depth-meshes range from 15 to 30 triangles per strip. Compared to the static depth-meshes, the view-dependent ones provide similar quality renderings at lower triangle counts. However, since

the view-dependent depth-mesh simplification was performed for each frame in the experiments the overall rendering time is much slower than for static depth-meshes. This is expected since the view-dependent mesh simplification and generation of triangle strips is an expensive task.

Note however, that infrequent updates to the viewdependent depth-mesh simplification as performed in IBR-based rendering systems[1,2,28,30,32] may still be handled efficiently enough if not performed for each individual frame. Once a view-dependent depth-mesh is selected its rendering is about as efficient as static depth-mesh rendering.

Figure 20 shows images of view-dependent DMesh rendering, comparing the geometric rendering (left) to a synthesized rendering from multiple reference depth-meshes (right). Each color in the occupancy images (bottom row) illustrates the use of different depth-meshes by varying colors and shows any combination of two or more overlapping ones as orange-brown.

7.4. *Other experiments*

In Fig. 21 we compare static and view-dependent depth-mesh simplification for far, medium and near range rendering. The wire frame snapshots only show one out of the up to five rendered depth-meshes. Table 4 presents the experimental results, showing the varying number of rendered triangles for view-dependent (dynamic) depth-meshes. While the static depth-meshing approach is always faster

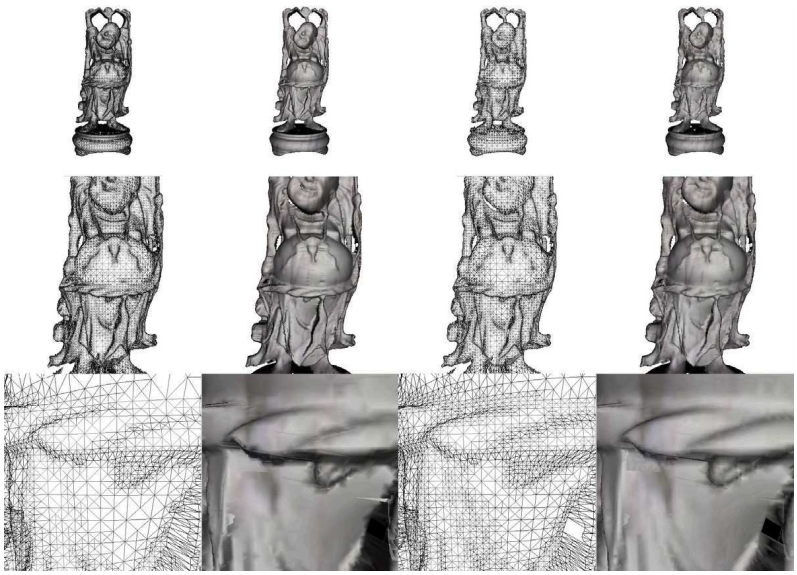


Fig. 21. Comparison between static (left two columns) and view-dependent (right two columns) depth-mesh simplification and rendering at different distances (far, medium, near) from the viewpoint.

Table 4. Comparison between static and dynamic (view-dependent) depth-mesh simplification and rendering.

Range	# Triangles		Rendering time		FPS	
	Static	Dynamic	Static	Dynamic	Static	Dynamic
Far	209,124	54,925	6.55 ms	114 ms	152	8.7
Medium	209,124	99,474	6.29 ms	157 ms	158	6.4
Near	209,124	75,258	6.23 ms	120 ms	160	8.3

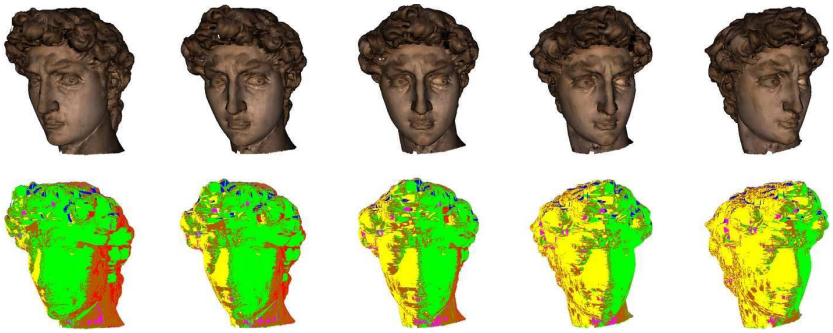


Fig. 22. Smooth rotation around depth-mesh object exhibiting strong changes in the weighted blending factors of the different depth-meshes.

in rendering performance, the view-dependent alternative may provide better quality depth-meshes for close-ups and may also exhibit less aliasing artifacts at far distances because the number of rendered triangles is adapted to the projection size on screen. One can also observe that depth-image warping speed using our DMesh technique is not restricted by pixel-fill rate limits of the video hardware. Independent of the screen-projection size the rendering performance is dominated by the complexity of the depth-meshes, and the adaptive triangulation in the view-dependent version.

Figure 22 illustrates the smoothness of our positional weighted blending approach. Despite strong changes in the weighted blending factors of the different contributing depth-meshes, the continuous rotation of the depth-mesh object does not exhibit any disturbing popping artifacts in the blending results.

8. Discussion

In this paper, we presented DMesh, an efficient depth-image meshing and segmentation method as well as a novel real-time depth-image blending algorithm that exploits hardware graphics acceleration. The proposed depth-buffer triangulation approach significantly reduces the complexity of depth-image warping by adaptive simplification of the triangulated depth-mesh, and by rendering textured triangle strips. Our novel blending technique provides real-time per-pixel weighted blending of multiple depth-images.

Our approach improves over previous depth-image warping methods[6,7,16,18,33] in particular in the following aspects: fast generation of depth-mesh based object representations, and per-pixel weighted blending of multiple reference views at interactive frame rates. The presented approach can be used as a rendering component in visualization systems such as large scale walk-through applications.[1,2,28,30,32]

The main limitation of the presented approach lies within the simple (but fast) segmentation of rubber-sheet triangles. The current approach is to provide a solution for use in interactive rendering systems where initialization and segmentation cost of depth-meshes is critical, but in specific circumstances may lead to over-conservative removal of triangles. At the expense of CPU cost, more sophisticated segmentation methods could be used (see for example Ref. 12) within the same framework.

Another conceived limitation of DMesh is its inability to cover disocclusion artifacts due to scene areas not visible in any reference depth-image. As already pointed out in Sec. 2, handling this type of exposure artifacts requires additional sample data and cannot be addressed algorithmically. DMesh does not introduce any restrictions handling the given input depth-images and hence *does not de facto* suffer from this limitation. For example, to alleviate disocclusion problems more sample data can be added by more depth-images. However, there are more intelligent approaches to add sample data such as layered approaches.[9,13,17,31] Thus extending DMesh to include multiple layers and sparsely populated depth-images is an important area of future work.

Further future work will include application of our blending algorithm to other IBR and point-rendering methods, and taking advantage of per-pixel and per-vertex programming features of modern graphics hardware accelerators.

Acknowledgments

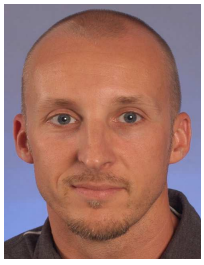
This work was partially supported by NSF grant CCR-0119053. We are very thankful to the various sources of geometric models, in particular we would like to thank the Stanford *3D Scanning Repository* and *Digital Michelangelo* projects as well as *Cyberware* for freely providing geometric models to the research community.

References

1. D. Aliaga, J. Cohen, A. Wilson, E. Baker, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, R. Bastos, M. Whitton, F. Brooks, and D. Manocha, "MMR: An interactive massive model rendering system using geometric and image-based acceleration," in *Proc. Symposium on Interactive 3D Graphics* (ACM SIGGRAPH, 1999), pp. 199–206.
2. D. Aliaga and A. Lastra, "Automatic image placement to provide a guaranteed frame rate," in *Proc. SIGGRAPH 99* (ACM SIGGRAPH, 1999), pp. 307–316.
3. C.-F. Chang, G. Bishop, and A. Lastra, "Ldi tree: A hierarchical representation for image-based rendering," in *Proc. SIGGRAPH 99* (ACM SIGGRAPH, 1999), pp. 291–298.

4. B. Chen and M. X. Nguyen, "POP: A hybrid point and polygon rendering system for large data," in *Proc. IEEE Visualization 2001*, pp. 45–52 (2001).
5. S. E. Chen and L. Williams, "View interpolation for image synthesis," in *Proc. SIGGRAPH 93* (ACM SIGGRAPH, 1993), pp. 279–288.
6. L. Darsa, B. C. Silva, and A. Varshney, "Navigating static environments using image-space simplification and morphing," in *Proc. Symposium on Interactive 3D Graphics* (ACM SIGGRAPH, 1997), pp. 25–34.
7. L. Darsa, B. C. Silva, and A. Varshney, "Walkthroughs of complex environments using image-based simplification," *Computers & Graphics* **22**(1), 25–34 (1998).
8. P. Debevec, C. Bregler, M. Cohen, L. McMillan, F. Sillion, and R. Szeliski, "Image-based modeling and rendering," SIGGRAPH 99 Course Notes 39 (1999).
9. X. Decoret, G. Schaufler, F. Sillion, and J. Dorsey, "Multi-layer impostors for accelerated rendering," in *Proc. EUROGRAPHICS 99*, pp. 61–72 (1999).
10. S. Dominé and J. Spitzer, "Texture shaders," *Developer Documentation* (2001).
11. J. P. Grossman and W. J. Dally, "Point sample rendering," in *Proc. Eurographics Rendering Workshop 98* (Eurographics, 1998), pp. 181–192.
12. A. Hoover, G. Jean-Baptiste, X. Jiang, P. J. Flynn, H. Bunke, D. B. Goldgof, K. Bowyer, D. W. Eggert, A. Fitzgibbon, and R. B. Fisher, "An experimental comparison of range image segmentation algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **18**(7), 673–689 (July 1996).
13. S. Jeschke and M. Wimmer, "Textured depth meshes for real-time rendering of arbitrary scenes," in *Proc. Eurographics Rendering Workshop*, pp. 181–190 (2002).
14. P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. A. Turner, "Real-time, continuous level of detail rendering of height fields," in *Proc. SIGGRAPH 96* (ACM SIGGRAPH, 1996), pp. 109–118.
15. A. Mangan and R. T. Whitaker, "Partitioning 3D surface meshes using watershed segmentation," *IEEE Transactions on Visualization and Computer Graphics* **5**(4), 308–321 (1999).
16. W. R. Mark, L. McMillan, and G. Bishop, "Post-rendering 3D warping," in *Proc. Symposium on Interactive 3D Graphics* (ACM SIGGRAPH, 1997), pp. 7–16.
17. N. Max, "Hierarchical rendering of trees from precomputed multi-layer z-buffers," in *Proc. Eurographics Rendering Workshop 96* (Eurographics, 1996), pp. 165–174.
18. L. McMillan, "A list-priority rendering algorithm for redisplaying projected surfaces," *Technical Report UNC-95-005*, University of North Carolina, 1995.
19. M. M. Oliveira and G. Bishop, "Dynamic shading in image-based rendering," *Technical Report UNC-98-023*, University of North Carolina, 1998.
20. M. M. Oliveira and G. Bishop, "Image-based objects," in *Proc. Symposium on Interactive 3D Graphics* (ACM SIGGRAPH, 1999), pp. 191–198.
21. R. Pajarola, *Access to Large Scale Terrain and Image Databases in Geoinformation Systems*, PhD thesis, Dept. of Computer Science, ETH Zürich, 1998.
22. R. Pajarola, Y. Meng, and M. Sainz, "Fast depth-image meshing and warping," *Technical Report UCI-ECE-02-02*, The Henry Samueli School of Engineering, University of California Irvine, 2002, submitted for publication.
23. R. Pajarola, M. Sainz, and P. Guidotti, "Object-space blending and splatting of points," *Technical Report UCI-ICS-03-01*, The School of Information and Computer Science, University of California Irvine, 2003, submitted for publication.
24. R. Pajarola, M. Sainz, and P. Guidotti, "Object-space point blending and splatting," in *ACM SIGGRAPH Sketches & Applications Catalogue* (2003).

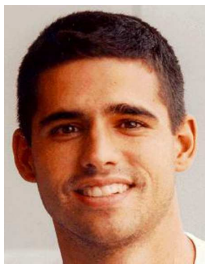
25. H. Pfister, M. Zwicker, J. van Baar, and M. Gross, "Surfels: Surface elements as rendering primitives," in *Proc. SIGGRAPH 2000* (ACM SIGGRAPH, 2000), pp. 335–342.
26. K. Pulli, M. F. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle, "View-based rendering: Visualizing real objects from scanned range and color data," in *Proc. Eurographics Rendering Workshop 97* (Eurographics, 1997), pp. 23–34.
27. H. Qu, M. Wan, J. Qin, and A. Kaufman, "Image based rendering with stable frame rates," in *Proc. IEEE Visualization 2000* (Computer Society Press, 2000), pp. 251–258.
28. M. Regan and R. Pose, "Priority rendering with a virtual reality address recalculation pipeline," in *Proc. SIGGRAPH 94* (ACM SIGGRAPH, 1994), pp. 155–162.
29. S. Rusinkiewicz and M. Levoy, "Qsplat: A multiresolution point rendering system for large meshes," in *Proc. SIGGRAPH 2000* (ACM SIGGRAPH, 2000), pp. 343–352.
30. G. Schaufler and W. Stürzlinger, "A 3D image cache for virtual reality," in *Proc. EUROGRAPHICS 96*, pp. 227–236 (1996).
31. J. Shade, S. Gortler, L.-W. He, and R. Szeliski, "Layered depth images," in *Proc. SIGGRAPH 98* (ACM SIGGRAPH, 1998), pp. 231–242.
32. J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder, "Hierarchical image caching for accelerated walkthroughs of complex environments," in *Proc. SIGGRAPH 96* (ACM SIGGRAPH, 1996), pp. 75–82.
33. F. Sillion, G. Drettakis, and B. Bodelet, "Efficient impostor manipulation for real-time visualization of urban scenery," in *Proceedings EUROGRAPHICS 97*, pp. 207–218 (1997). Also in Computer Graphics Forum 16(3).
34. R. Sivan and H. Samet, "Algorithms for constructing quadtree surface maps," in *Proc. 5th International Symposium on Spatial Data Handling*, pp. 361–370 (August 1992).
35. M. Zwicker, H. Pfister, J. van Baar, and M. Gross, "Surface splatting," in *Proc. SIGGRAPH 2001* (ACM SIGGRAPH, 2001), pp. 371–378.



Renato Pajarola received his Dr. Sc. Techn. in Computer Science in 1998 from the Swiss Federal Institute of Technology (ETH) Zurich.

Following his dissertation, he was a post-doctoral researcher and part-time faculty member at the Graphics, Visualization & Usability Center at Georgia Institute of Technology for a year. Since 1999 he has been an Assistant Professor in Information & Computer Science at the University of California, Irvine.

His current research interests include real-time 3D graphics, multiresolution modeling, image based rendering, image and geometry compression, interactive remote visualization, large scale terrain and volume visualization, as well as interactive 3D multimedia. He has published a wide range of technical papers in top journals and conferences. He served as reviewer and program or organization committee member for several international conferences and is a frequent referee for journal articles. He is a member of the IEEE Computer Society, ACM and the ACM Special Interest Group on Computer Graphics.



Miguel Sainz received his PhD degree in Electrical and Computer Engineering in 2003 from the University of California, Irvine.

He is currently a post-doc and part time lecturer at the School of Information and Computer Science at the University of California, Irvine. His current research interests include image based modeling and rendering, 3D model reconstruction from images, tracking and image processing and real-time 3D graphics. He is a member ot the IEEE Computer Society and the ACM.

Yu Meng is currently a PhD candidate in Information & Computer Science at the University of California, Irvine. He received his Master degree in Computer Science from the University of California, Irvine and his Bachelor degree from the Tsinghua University, Beijing, PRC. He is now conducting research on the area of graph visualization.