LAYERED RELIEF TEXTURES

Sergey Parilov, Wolfgang Stuerzlinger

Department of Computer Science
York University, 4700 Keele Street
M3J 1P3, Toronto, Ontario
Canada
{parilov|wolfgang}@cs.yorku.ca www.cs.yorku.ca/~wolfgang

ABSTRACT

In this paper we present an Image-Based Rendering method for post-warping LDI's in real-time on existing systems. The algorithm performs accurate splatting at low computational costs, reduces memory-access bottlenecks, enables us to trade-off the quality for the speed, and is simple to implement.

Keywords: image-based rendering, image warping, real time rendering, relief textures, layered depth images, splatting.

1. INTRODUCTION

Over the last few years, the growing needs of the graphics community and the recent advances in technology have resulted in the considerable increase in the rendering speed of polygon-based graphics hardware. Although it is possible to render up to 300 thousand triangles per frame in practice, the hardware still cannot create photo-realistic images in real-time. Among the reasons are the complexity of creating realistic polygonal models in general, and the inability of rendering the complex ones at high frame rates. In special cases the quality of the images can be improved with bump-maps, light maps and similar advanced rendering techniques. This still does not solve the problem for acquired imagery and point sampled models.

Image-Based Rendering by Warping (IBRW) was proposed as a method to overcome these difficulties by using images as the model for representing the source data. Current IBRW techniques can create photo-realistic renderings at the cost of the low speed that prohibits the use of IBRW in real-time interactive applications. Despite several research projects aimed at creating IBRW-capable hardware, even the simplest IBRW methods have not been demonstrated until now.

We present a low computational cost IBRW method targeted to support insertion of realistic objects into polygonal scenes. We employ the texture mapping capabilities of existing graphics hardware, and perform all the other operations in software. In this paper, we

- show how the LRT (Layered Relief Textures) method can achieve a high-quality reconstruction image with small computational cost; and
- demonstrate IBRW at real-time rates for medium-complexity models.

The paper is organized as follows. In the following section we present a brief overview of the existing methods and describe the basic warping algorithms on which we build our method. Section 3 describes a way to mitigate the inability of one of the previous IBRW methods (which this work builds onto) to render objects from arbitrary views. Section 4 discusses the problem of efficient image reconstruction from point-samples. We compare the quality of our method with that of the existing techniques. In section 5, we propose the method to use the cache memory in a more efficient way. Section 6 discusses our implementation and the results.

2. PREVIOUS WORK

Several different image-based rendering methods based on different data structures have been presented over the last few years. In this work, we limit the discussion only to images augmented with per-pixel disparity[1], as defined in [Mcmil97].

[1] the quantity inversely proportional to depth

For Image-Based Rendering in general, the quality of the final result depends primarily on the quality of the source images. Using photographs or images synthesized with global illumination techniques IBR methods can generate highly realistic results [Mcmil97, Debev96, Debev98]. Acquiring models for IBR implies computation of camera parameters for source images and obtaining depth values for each source image pixel. For real images, both these tasks can be addressed with computer vision methods. For synthesized images this information is easy to store during image generation.

IBRW algorithms essentially solve two problems - geometrical warp problem defined as the map from the source to the destination image according to the camera configuration, and the signal reconstruction problem to reconstruct the intensities defined by the samples of the source images [Mcmil97, Shade98, Olive00, Popes98]. The (planar) forward warp derived in [Mcmil97]

$$\bar{x}_2 \doteq \ddot{a}(\bar{x}_1)\mathbf{P}_2^{-1}(C_1 - C_2) + \mathbf{P}_2^{-1}\mathbf{P}_1\bar{x}_1$$

computes the location of the point x_2 in the destination image corresponding to a point x_1 in the source image, where $\ddot{a}()$ is the disparity of a point, and P_1, C_1, and P_2, C_2 define camera configuration of the source and the destination images respectively. The occlusion compatible order guarantees correct visibility between samples with a simple painter's algorithm. In [Mcmil97], it is defined as a source image plane traversal such that along every line in the source image passing through the epipole[2] the points farther away from the epipole are visited first. The cost of the computation is a matrix multiplication per source image pixel.

As the occlusion compatible warping order is only valid for a pair of source and destination image, the regions in the destination image which should be depicting parts of the scene not visible from the source image camera appear empty. Layered Depth Images (LDI), defined as a "view of the scene from a single input camera view, but with multiple pixels along each line of sight" [Shade98], solve the problem. Furthermore, LDI's support occlusion compatible traversals. While the LDI's rely on the McMillan's warping equation, the incremental warping scheme is presented in [Shade98].

Relief textures [Olive00] present an alternative that requires slightly less arithmetic operations per source sample and exploits standard graphics hardware to perform part of the computations. The key innovation of this approach is to decompose the warping process into two separate passes - the pre-warp and the texture mapping. The pre-warping stage handles only the parallax effects by pre-warping the source image into an intermediate image. On the second stage, the intermediate image is used to texture map the polygon in 3D space corresponding to the source image's viewplane. This accounts for the perspective distortion, zoom, and rotation due to the current viewpoint and produces the same result as a conventional warp. The pre-warping equations are defined as

$$\boldsymbol{u} = \frac{\boldsymbol{u_s} + \boldsymbol{k}_1\boldsymbol{d}}{1 + \boldsymbol{k}_3\boldsymbol{d}}, \qquad \boldsymbol{v} = \frac{\boldsymbol{v_s} + \boldsymbol{k}_2\boldsymbol{d}}{1 + \boldsymbol{k}_3\boldsymbol{d}}$$

and determine the position *(u,v)* of a source image point (u_s,v_s,d) in the intermediate image, where d is the depth (or displacement), and k_1,k_2,k_3 are the coefficients that depend only on the camera configuration and are independent of the source point co-ordinates [Olive00].

The main shortcoming of relief texturing is that it does not support multiple pixels along one line of sight, thus being unable to handle objects that feature occlusions inside the volume represented by the relief texture. In other words, only a subset of objects can be handled correctly and artefacts appear if the fundamental assumptions are violated (e.g. the areas behind the chimneys in [Olive00]).

On modern graphics hardware, rendering an rectangular array of pixels after a conventional warp has run-time cost comparable to that of rendering a texture mapped triangle. Consequently, an additional texture mapping operation is practically free.

Because sampling rates of the source images rarely match the desired output resolution interpolation between samples is a major issue in IBRW methods. Many IBRW techniques use different kinds of splatting of varying quality of reconstruction and speed. The reconstruction errors introduced by splatting are analyzed in [Mark97].

In the original warping approach the splat for each sample was computed on the fly. As the computation of exact splat footprints is extremely computationally intensive different approximations are used in practice, thus degrading the quality. Fixed sized splats have been used previously to achieve high warping speed [Aliag99]. In [Shade98], the authors use a small pre-computed fixed set of Gaussian splats to achieve somewhat better quality. In [Pfist00], a splatting method that allows high-quality reconstructions with high computation costs is presented. In [Rusin00], the authors present a

[2] the projection of the center of projection of one camera on the viewplane of the other

hierarchical technique to visualize extremely large models.

Relief textures [Olive00] present an alternative that requires slightly less arithmetic operations per source sample and exploits standard graphics hardware to perform part of the computations. The key innovation of this approach is to decompose the warping process into two separate passes – pre-warp and texture mapping. The pre-warping pass warps the source image to take parallax into account. The second pass maps the output of the first pass as a texture onto a 3D rectangle in space. This accounts for perspective distortion, zoom and rotation due to the current viewpoint and produces the same result as a standard warp. Note that the result of a traditional warp still needs to be rendered to the screen after the warp. Rendering the resulting rectangular array of pixels has run-time cost that is comparable to rendering one texture-mapped triangle. Therefore, Oliveira's approach is slightly less costly than the McMillan's approach or the LDI splatting approach. The main shortcoming of relief texturing is that it is unable to handle objects that feature occlusions inside the volume represented by the relief texture.

That IBRW can achieve better speeds than polygonal computer graphics still needs to be demonstrated in the general case. IBR methods such as QuickTime VR can achieve the speed at the expense of limiting the viewer motion to a location and allowing only rotations. Hierarchical image caches or impostor hierarchies can be rendered in real-time, but rely on the existence of a geometric description [Schaufler96, Shade96]. Therefore, they cannot be applied to acquired imagery. More general IBR methods based on McMillan's warping method usually achieve a maximum speed of up to 10 frames per second [Shade98, Popes98, Chang99].

Because image warping implies many memory accesses, both reads and writes, memory bandwidth becomes a significant issue. In [Mark97], the author has addressed the issue with algorithms that provide more efficient use of cache memory. Another solution is to limit the amount of data being considered by clipping, although little work has been done in this area [Shade98, Popescu98].

The approach presented in this paper addresses the issues mentioned by a combination of techniques. We combine layered depth images and relief textures and call the resulting data structure Layered Relief Texture (LRT). Similar to LDIs, we use multiple samples along the line of sight, and, similar to relief textures, we use simplified equations to pre-warp the samples and render them on the screen with the help of texture mapping hardware. This extends the relief textures approach to deal with arbitrary objects. Furthermore, the nature of pre-warping equation enables us to use a different solution to the splat size computation problem to achieve good image quality.

We use tiling of both the source and the pre-warped images such that every pre-warped image tile fits into L2 cache to address memory bottlenecks. The tiles of the pre-warped image can be rendered separately and selectively. This allows us to easily clip the image and to re-use the results of the warp of the previous frame if the estimated error is below some threshold or if the time allowed for the frame generation is over (similar to [Lengy97, Schau96]).

3. LAYERED RELIEF TEXTURES

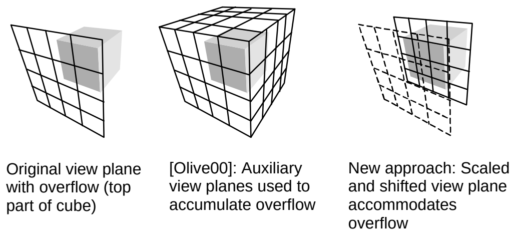
Layered Relief Textures (LRTs) combine the advantage of relief textures with layered depth images (LDIs). We base our approach on the optimised warping method as described in [Olive00]. This relief textures approach offloads part of the computational effort from the CPU to the graphics hardware. First the image is pre-warped and then texture mapping is used to complete the operation. For correct image generation, the object rendered should be complete, i.e. with no visible gaps and holes due to absence of some parts in source data set being used. In [Olive00], the authors propose to render several relief textures of the object per frame. Consequently, the same point of the object can be warped several times from different relief textures. Obviously, this degrades performance.

3.1 OVERFLOW HANDLING

If a 3D-point is far from the source viewplane of a relief texture, its projection in the destination image can fall outside the projection of the polygon being textured and such point cannot be drawn. In [Olive00], authors call this effect an overflow. To deal with this difficulty, the overflows can be accumulated on two auxiliary polygons, perpendicular to the source view plane [Olive00]. Everything that projects outside of the polygon corresponding to the source viewplane is drawn on these auxiliary polygons. While this solves the problem, it requires rendering several polygons for each relief texture and results in complicated occlusion-compatible orders. We have chosen to address this problem by dynamically resizing the texture.

We associate a bounding polyhedron with the entire relief texture, with its vertices stored as coordinates on the source image plane with displacements. A simplified convex hull (with a limited number of vertices) or a bounding box can be used. Before pre-warping the texture itself, we pre-warp the vertices of the bounding polyhedron and compute its bounding box. We are guaranteed that all source image pixels will project to within the projection of this box.

By computing the extent of the projected bounding box we can infer how much the source viewplane needs to be rescaled (shrunk or stretched) and shifted to fit all pre-warped points. We then re-scale the texture correspondingly and render it. Figure 1 illustrates the problem, how the relief textures approach handles the problem and how do we address it.


Original view plane with overflow (top part of cube)
[Olive00]: Auxiliary view planes used to accumulate overflow
New approach: Scaled and shifted view plane accommodates overflow

Handling of Relief Texture Overflow
Figure 1

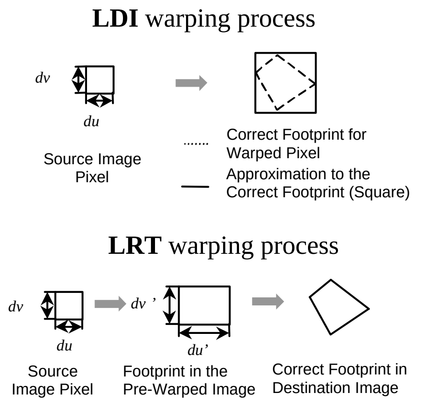
Changes in the size of the polygon being textured are equivalent to inverse changes in the size of the resolution in comparison with the original texture. When the angle between the observation direction and the source viewplane becomes too sharp the viewplane stretches by a large factor, thus lowering the effective resolution and decreasing the quality. We avoid this by storing multiple LRTs with different viewing directions and switching to another LRT with a less severe viewing angle. If all the used LRTs are consistent with each other, this switch is practically invisible. The only potential source of inaccuracies is a slight misalignment due to the discrete nature of textures and the polygon rendering hardware, but we find in practice that these effects are usually less than 1 pixel. An additional advantage is that we can dynamically adjust the resolution of the desired image if the quality can be sacrificed for the speed of rendering.

4. SPLATTING

A drawback of the presented combination of LDIs and relief textures is that we cannot use the interpolation method presented in [Olive00] due to intra-object visibility. Instead, we have to resort to splatting. This however turns out to be an advantage, as we can create better approximations to the footprint of the splats in the image plane than previous approaches. For efficiency however, the splat shape approximation must be fast to compute. The method we propose does not make use of normal vectors and other auxiliary information, but relies only on the displacement values of the samples. We approximate splats by perspectively distorted axis-aligned rectangles.

4.1 SPLAT SHAPE AND SIZE COMPUTATION

Assume that neighbouring source image pixels are at a distance *du* from each other horizontally, and *dv* vertically. Then, a pixel at *(u,v)* can be modelled as a rectangle with its corners having coordinates $(u \pm du, v \pm dv)$.


LDI warping process
dv
du
Source Image Pixel
Correct Footprint for Warped Pixel
Approximation to the Correct Footprint (Square)
LRT warping process
dv
du
dv'
du'
Source Image Pixel
Footprint in the Pre-Warped Image
Correct Footprint in Destination Image

Approximation of Splat Shape
Figure 2

Having only points with depth and without any further information, such as the presence of normals to the surface or connectivity between samples, a point is simply an evidence of the presence of a surface. Note that this assumption is true for most LDI data sets, due to the existence of multiple layers. Consequently, we can only assume that a sample is a flat, axis-aligned region of some surface facing directly towards the camera. According to the pre-warping equations a pre-warp of such an axis-aligned rectangular sample will result in an axis-aligned rectangular sample in the pre-warped image. Later, the texture-mapping pass will render the splat correctly on the screen. To pre-warp warp a source image pixel, we warp its center to find the center of the splat in the pre-warped image. The amount of the horizontal and vertical extent of the rectangular splat with respect to the original sample size depends only on the depth, and does not depend on the co-ordinates in the image plane. That is, the size of the splat can be found as

$$\boldsymbol{dv}' = \frac{\boldsymbol{dv} + \boldsymbol{k}_1 \boldsymbol{d}}{1 + \boldsymbol{k}_3 \boldsymbol{d}}, \quad \boldsymbol{du}' = \frac{\boldsymbol{du} + \boldsymbol{k}_2 \boldsymbol{d}}{1 + \boldsymbol{k}_3 \boldsymbol{d}},$$

where *du*, *dv* are horizontal and vertical size of the source image pixel, and *d* is the depth. Figure 2 illustrates the approach and compares it with the splatting directly to the screen.

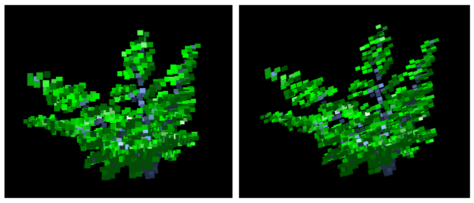
4.2 COMPARISON WITH OTHER METHODS

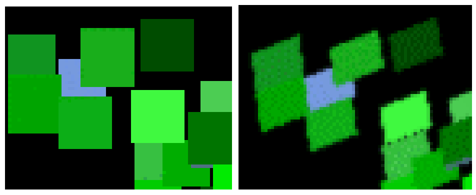
We found this splatting method to be simple yet giving good visual quality. One of the reasons is that the final result is a perspectively distorted splat, which is not orthogonal with respect to the axes of the desired view in general. This will generate a better image compared to other approaches that splat directly into screen space. Furthermore, as it has been shown in [Rusin00], simple rectangular splats are much faster to render than Gaussians, and given the time allotted for the frame generation they tend to produce better results.

In the original LDI paper, the authors use a fixed set of 1x1, 3x3, 5x5, and 7x7 Gaussian splats, that are output directly to the destination image [Shade98]. In this case, the introduced reconstruction errors depend primarily on two factors - the error due to the approximation of a perspectively distorted splat by a square and a maximum of half a pixel error due to the final image sampling. In this work, we do not splat directly into the destination image, but into an intermediate image that will be used as a texture map. Since the pre-warp stage does not handle rotations or perspective distortion and both source and intermediate image are orthogonal, any axis-aligned rectangle with constant disparity corresponds to an axis-aligned rectangle in the intermediate image. Under the assumption of rectangular source samples, we get a splat that is optimal with this approach.

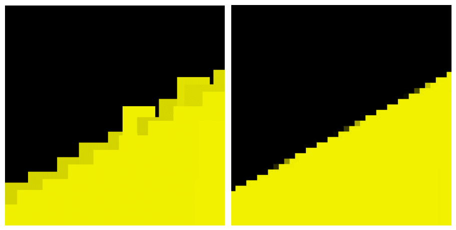
During the texture mapping stage the splat undergoes perspective distortion. When the texture mapping is done, each splat is rendered as a non-axis-aligned perspectively distorted quadrilateral, which matches the ideal splat shape much more closely than splats produced by many other approaches. Since we use arbitrary rectangles as splats, the error in the first pass is mainly due to the round-off errors in the positions of splats in the intermediate image, which are no more than half a pixel. During the second pass this error is reduced or magnified depending on the perspective distortion of the view. Furthermore, polygon rasterization and texture mapping artefacts introduce an additional error of less than half a pixel.

The error of our method is definitely less than the error of the LDI approach when the view is tilted or when the object is viewed at sharp angles as

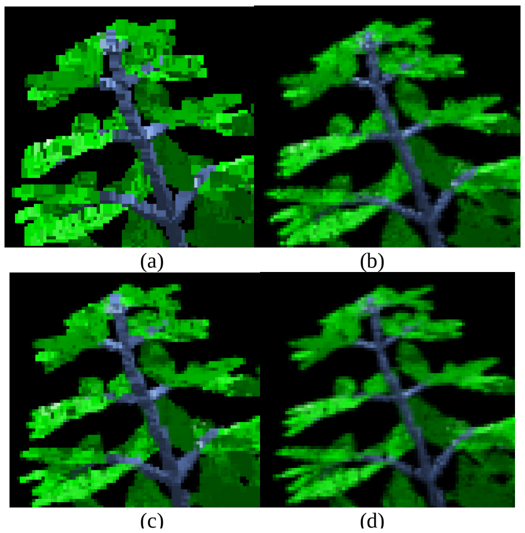

(a) (b)
Renderings of a fern model, (a) LDI; (b) LRT.
Figure 3


(a) (b)
Parts of the fern renderings zoomed in
Figure 4

the perspective distortion gets larger in these cases. Furthermore, our approach generates better results when the resulting image is smaller than the original image, i.e. when minification occurs, as we can benefit from mip-mapping. The LDI splats will introduce severe aliasing artifacts in this case. Only if there is very little perspective distortion and no viewer tilt may the LDI approach generate better images than the approach presented here. Also, our method can utilize bilinear interpolation of texture maps, which will further increase the quality of the result, even when the texture is magnified. Figures 3, 4 and 5, compare objects rendered with both LDIs and LRTs using rectangular splats.


(a) (b)
Zoomed-in edge of a cube rendered
with (a) LDIs (b) LRTs
Figure 5

The advantages of the LRT method become even more apparent, when one compares the results of Gaussian splatting with the LDI method with the results of the LRT method. Figure 6 shows all four possible combinations. Note that the result of the LRT approach with rectangular splats is almost equal to that of the LDI approach with Gaussian splats. Note however, that the LRT approach with perspectively distorted rectangular splats is much less costly than any approach using Gaussian splats.



Comparison of Splatting Methods.
(a) LDI rectangular, (b) LDI Gaussian,
(c) LRT rectangular, (d) LRT Gaussian
Figure 6

5. MEMORY BOTTLENECKS

For image warping, memory accesses are one of the most important factors limiting speed. To mitigate the problem, we tile the images to increase cache-efficiency. We use a sort-first method with static region assignment [Muell95, Muell97]. We partition both the source LRT and the pre-warped images into a set of regular, axis-aligned rectangular tiles. The selection of the optimal tile size is empirical and depends strongly on the characteristics of the system.

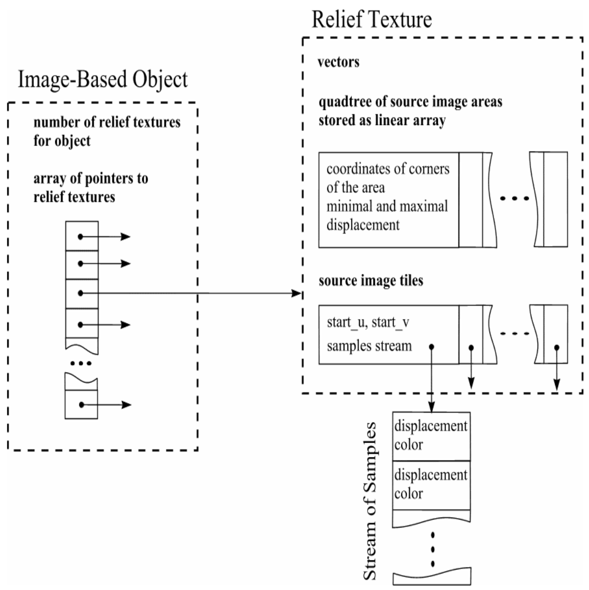
To correctly pre-warp into a given pre-warped image tile (destination tile), we select and pre-warp all LRT tiles (source tiles) that have a potential contribution. To determine this set we use a quad-tree. The root of the tree is the whole source image itself and the leaves of this tree are the source LRT tiles. At each node the information about the covered area for the node and the maximal and minimal displacements of the samples that belong to this area are stored.

We determine the set of the source tiles contributing to a given destination tile by descending recursively into the tree. At each node visited, we pre-warp the corners of the tile with the minimal and maximal displacement values and compute the bounding box of the resulting points. If the bounding box intersects the destination tile, we assume that some point of the source LRT area can contribute to it. To lower the overestimation of the affected area, the same could be done with bounding polyhedras.

For correct visibility we sort the set of the selected source tiles according to the occlusion compatible order between these tiles. With our tiling of the source LRT, the occlusion compatible ordering of tiles corresponds to the occlusion compatible ordering of the points farthest from the epipole for each tile. We then process source LRT tiles in this order, pre-warping points of each tile also in occlusion compatible order. Source LRT tiles that cross horizontal or vertical epipolar lines have to be split into sheets. Each of these sheets is processed in occlusion compatible order.

6. IMPLEMENTATION AND RESULS

We implemented our algorithm on an SGI Onyx2 with a 300 MHz R12000 CPU and 8 MB of level 2 cache. The rendering was done on InfiniteReality2 graphics hardware. The software was written in C.


Relief Texture
Image-Based Object
vectors
quadtree of source image areas stored as linear array
number of relief textures for object
array of pointers to relief textures
coordinates of corners of the area minimal and maximal displacement
source image tiles
start_u, start_v
samples stream
displacement
color
displacement
color
Stream of Samples

Run-time Data Layout
Figure 7

We store the quad-tree of source LRT tiles as a linear array in which the children of a node stored with the index n are stored with the indices n*4+1 through n*4+4. The samples of each LRT tile are stored as a stream of tuples, in the order they are considered according to occlusion-compatible ordering. As there are four possible traversal orders for the source image, we replicate each tile in memory four times with different sample orderings. We use unique markers to mark the advance to the next row or column of samples. The time overhead is still small since we need slightly more than one comparison per sample on average to detect the advance to the next column, row, or the end of the data stream. Depth information is stored for each pixel as a 32-bit floating-point value.

We used the model of a cube and a textured fish to test the quality of splatting in the case of sharp edges and fine-grained textures, and a model of a fern to test the intra-object visibility. The source images are 512x512 orthogonal projections synthesized with ray tracing. For the tests we rendered views to a 1024x1024 window. Although in all cases the texture is magnified at the rendering stage, we found the results to be remarkably free of visible artefacts. The results for the cube are shown in Figure 5, the fish appears in Figure 8 and the fern appears in Figure 3, 4, and 8.

The LRT models of the fish and cube contain about 140 000 samples, and 250 000 samples for the model of the fern at the maximum level of detail. For a view, due to tiling of the source image, the number of samples actually warped is 20% greater on average than the number of samples drawn. During the test we rotated the model arbitrarily with a sequence of 300 steps. To ensure that we also tested cache miss performance the sequence included some large rotations. Although the implementation was not heavily optimized, the rendering frame rate including the graphics output with 140 000 samples is approximately 15 Hz, and with 250 000 is approximately 9 Hz. The warping itself runs at 20 Hz (= real-time !) in the case of the two medium-complexity models, and at 10 Hz in case of the fern. The graphics pipeline introduces some slowdowns, that is, the texture transfers take about 15-20% of the time of the rendering. This effect can be minimized by pipelining the warping and the rasterization steps (i.e., rendering the previous frame while warping the current one).

7. CONCLUSIONS

We presented a method for warping the LDIs at interactive frame rates. The method combines the layered depth images and the relief textures approaches and leverages the benefits from both. We have also discussed a precise and efficient method for splat size computation. Furthermore, we have demonstrated that the current hardware memory bandwidth is an important factor for image warping speed.

Future work includes parallelization of the LRT warping process, to increase the frame-rate even further. Initial results of a parallelized version on a machine with multiple CPU's indicate that this method can warp images of high-complexity models at real-time.

To increase the apparent frame rate even further one option is not to pre-warp the source image for every frame but only when the distortion perceived by the user exceeds some threshold, similar to [Lengy97, Schau96]. The incorrect parallax in such frames will be mitigated by the correct perspective distortion done on the texture mapping stage.

Another idea is to trade-off quality for speed, by changing the resolution of the rendered image and stretching the result to fit the desired area on the screen (similar to [Montr97]).

REFERENCES

[Aliag99] Aliaga,D., Cohen,J., Wilson,A., Baker,E., Zhang,H., Erikson,C., Hoff,K., Hudson,T., Stuerzlinger,W., Bastos,R., Whitton,M., Brooks,F., Manocha,D.: MMR: An Interactive Massive Model Rendering System Using Geometric and Image - Base Acceleration, *1999 Symposium on Interactive 3D Graphics*, pp. 199-206, 1999

[Chang99] Chang,C.-F., Bishop,G., Lastra,A.:LDI Tree: A Hierarchical Representation for Image-Based Rendering, *SIGGRAPH 1999*, pp. 291-298, 1999

[Debev96] Debevec,P.E.: Modeling and Rendering Architecture from Photographs, *Ph.D. thesis*, University of California at Berkeley, 1996

[Debev98] Debevec,P.E.: Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography, *SIGGRAPH 1998*, pp. 189-198, 1998

[Mark97] Mark,W.R., Bishop,G.: Memory Access Patterns of Occlusion-Compatible 3D Image Warping *1997 Siggraph/Eurographics Workshop on Graphics Hardware*, pp. 35-44, 1997

[Muell95] Mueller,C.: The Sort-First Rendering Architecture for High-Performance Graphics, *1995 Symposium on Interactive 3-D Graphics*, 1995

[Muell97] Mueller,C.: Hierarchical Graphics Databases in Sort-First, *IEEE Symposium on Parallel Rendering*, pp. 49-57, 1997

[Mcmil97] McMillan,L.: An Image-Based Approach to Three-Dimensional Computer Graphics, *Ph.D. Thesis*, University of North Carolina at Chapel Hill, 1997

[Lengy97] Lengyel,J., Snyder,J.: Rendering With Coherent Layers, *SIGGRAPH 1997*, pp. 233-242, 1997

[Olive00] Oliveira, M.M., Bishop,G., McAllister,D.: Relief Texture Mapping, *SIGGRAPH 2000*, pp. 359-368, 2000

[Pfist00] Pfister,H., Zwicker,M., van Baar,J., Gross,M.: Surfels: Surface Elements as Rendering Primitives, *SIGGRAPH 2000*, 2000

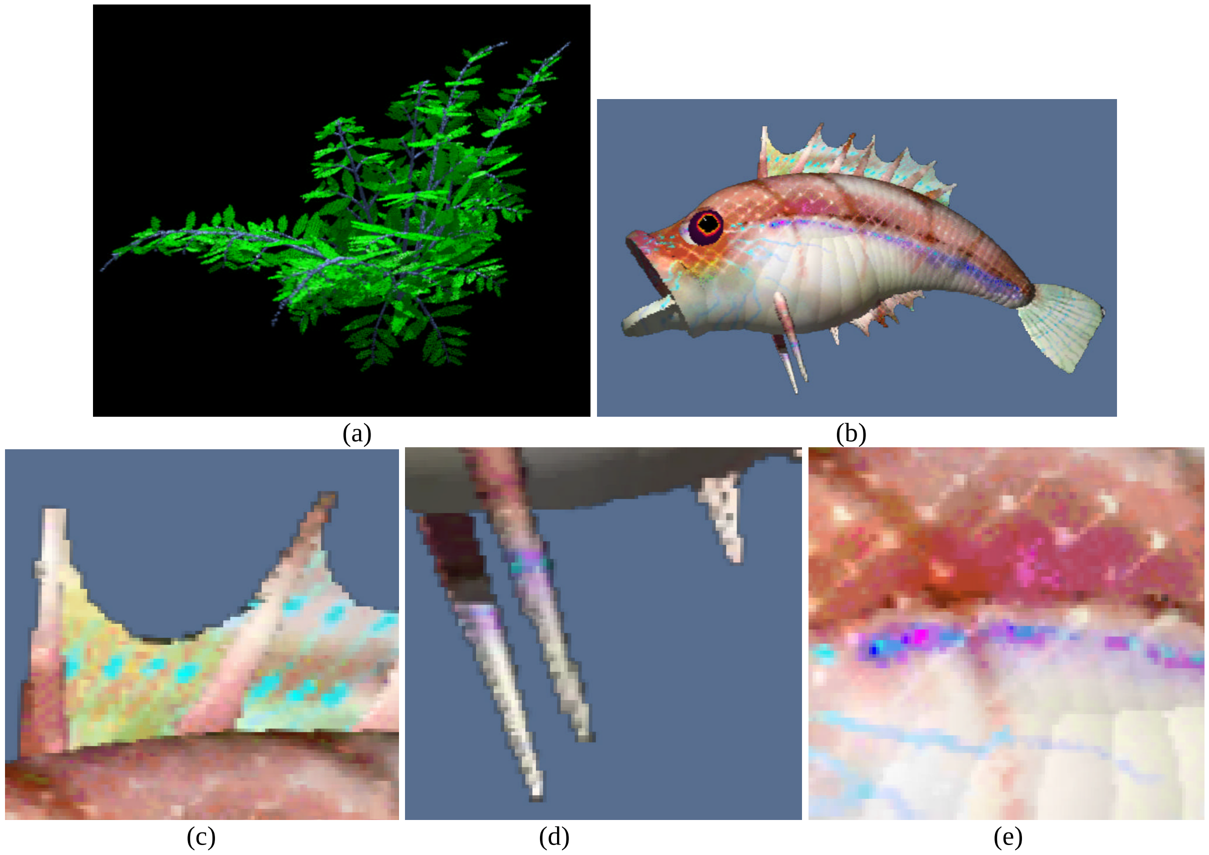
[Popes98] Popescu,V., Lastra,A., Aliaga,D., Oliveira,M.: Efficient Warping for Architectural Walkthroughs Using Layered Depth Images, *IEEE Visualization 98*, pages 211-215, 1998

[Ruzin00] Rusinkiewicz,S., Levoy,M.: QSplat: A Multiresolution Point Rendering System for Large Meshes, *SIGGRAPH 2000*, 2000

[Schauf96] Schaufler,G., Stuerzlinger,W.: Three Dimensional Image Cache for Virtual Reality, *EUROGRAPHICS 1996*, pp 227-236, 1996

[Shade96] Shade,J., Lischinski,D., Salesin,D., DeRose,T., Snyder,J.: Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments, *SIGGRAPH 1996*, pp.75-82, 1996

[Shade98] Shade,J., Gortler,S., He,L., Szeliski,R.: Layered Depth Images, *SIGGRAPH 1998*, pp. 231-242, July 1998



LRT rendering of (a) fern, (b) fish, and (c)-(e) close-ups of fish. Note the quality of the edges and reproduction of detail. The results were obtained with perspectively warped rectangular splats. For timings refer to the text.
Figure 8