Per-Object Image Warping with Layered Impostors

Gernot Schaufler, GUP, Altenbergerstr. 69, A-4040 Linz, AUSTRIA
gs@gup.uni-linz.ac.at

Abstract. Image warping is desirable in the context of image-based rendering because it increases the set of viewpoints for which a single image can be used. This paper proposes a method for image warping with adaptive accuracy compatible with current texture-mapping hardware. It is based on the observation that pixels at similar depth move in a similar way during warping. The method also generates approximate depth values at each pixel so that polygonal and image-based rendering can be applied in a mixed fashion.

1 Introduction and Motivation

In interactive computer graphics hardware-accelerated rendering of polygonal models is a well-established visualization technique. However, the ever growing complexity of polygonal models continues to outperform the advances made in hardware technology.

In recent years an alternative approach called image-based rendering (IBR) was proposed concerned with generating new images from existing ones. It decouples scene complexity from rendering complexity by employing work proportional to the number of pixels in the final image instead of work proportional to the number of modeling primitives in the scene.

Research on IBR has focused on how to best use the available image data and has progressed to warping images augmented with depth information for new viewpoints (view interpolation and extrapolation). Different approaches for this warp have been taken and will be reviewed in the related work section. In particular surfaces pose problems which were undersampled or occluded in the available images but are visible in the image to be generated causing objectionable holes and tears in the final image.

Pure image-based representations require overwhelming amounts of image data to represent a complex scene for every arbitrary point of view. Hybrid rendering systems apply image-based methods mostly for distant scene elements to exploit the coherence in the images of those scene parts. Image data generated for recent frames is reused in the next frames to avoid the expensive re-rendering of the complete scene.

The approach to be presented uses hardware-accelerated texture mapping to warp the image data for new viewpoints and is based on the observation that portions of an image at a certain distance from the viewer move in a uniform way: the warp is approximated by drawing several layers of texture-mapped quadrilaterals. This is similar in spirit to using several impostors per object like in the hierarchical image cache [20][22]. However, this is prohibitive both in texture memory consumption and texture generation cost. The presented method avoids these problems by caching only one image per object and warping it for new viewpoints. The warping accuracy is adaptive to the desired change to the image. Little changes to the image are computationally less expensive.

After an overview of the relevant work in this field the following section introduces layered impostors and describes the ideas behind them. Implementation details are given in section 4 followed by possible optimizations in section 5. Results are summarized in section 6 and conclusions drawn in section 7.

2 Previous Work

A useful classification of IBR methods is by complexity of the warping technique. It is most straightforward to re-use whole images or parts thereof to save rendering costs or to achieve greater temporal fidelity [7][8][15].

The next complex class of image reuse is as (possibly partially transparent) texture maps which are mapped onto simple geometry. This coarse type of reuse avoids tears in the images resulting from previously occluded surfaces. Maciel et al [11] pre-generate textures for objects and clusters of objects to achieve a uniform frame-rate in their walkthrough system. Schaufler et al [20] and Shade et al [22] dynamically update the textures to always match the object's or scene's appearance from the current point of view. Aliaga et al [1] warp the geometry to minimize the popping effect when switching between textures and geometry. The Talisman architecture [24] approximates the 3D warp with a hardware implementation of affine transformations. Schaufler [21] augments textures with depth to avoid visibility errors caused by the "flatness" of impostors.

In an attempt to better approximate the required changes to the reused images triangular meshes have been generated from the images and their depth map. While Mark et al [12] have found pixel-grained meshes to be expensive, meshes solve the problem of holes in the images. However, they introduce the problem of "rubber sheets" in place of the holes. Subsampling [23] and mesh-decimation [5] give meshes of manageable size. Pulli et al [17] use a "soft" z-buffering approach to blend together the textured meshes obtained from several range-images.

Instead of triangle meshes individual pixels have been warped as well. Both forward and backward mappings have been tried. With forward mapping [2][6][9][14][16] holes and tears in undersampled areas are avoided by splatting.

Backward mapping produces no holes but such a mapping is only simple to calculate if the current point of view is the same as the viewpoint from which the input images were taken [3][18]. Otherwise a search must be performed in the source image to identify the pixel which maps to the current output pixel [10].

Only one object-centered approach to image-based rendering has been published so far, also based on McMillan's warping algorithm, namely delta trees [4]. They are very efficient in storage requirements as redundancies among a number of input images are avoided.

In several publications hardware implementations of IBR-methods are being called for and the need for research on applicable hardware architectures for IBR has been pointed out [12][13][24].

3 Layered Impostors

For generating an image of an object a possibly approximate representation of the object's shape is needed. A polygonal model is one such representation, usually considered view-independent (it can be used for any viewpoint, only in extreme close-ups the polygonal approximation becomes apparent). An image of an object mapped onto a polygon facing the viewer is another approximation of the object's shape, but a view-dependent one. Such a textured polygon (or impostor [11]) can only be used for points of view close to the point from where the image was taken.

Layered impostors are a generalization of dynamically generated impostors [19] in the following sense: an impostor replaces an object by one transparent polygon onto which the opaque image of the object is mapped. Thereby the depth of the object is discarded and the polygon's flatness becomes apparent when the viewpoint moves.

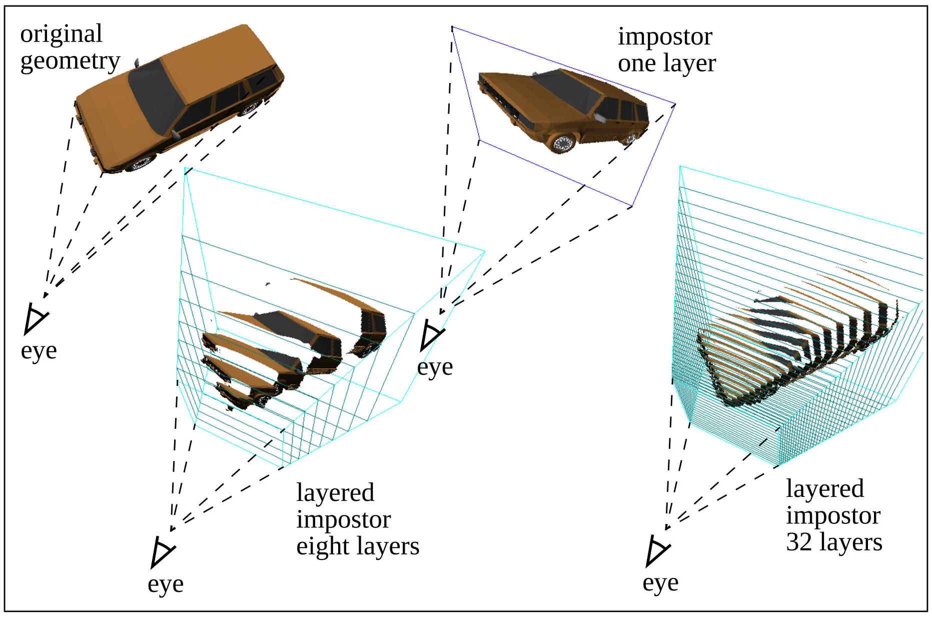

Fig. 1: Replacing an object (top left) with an impostor (top right) or a layered impostor (bottom)

A layered impostor consists of many such transparent polygons. On every polygon all drawn texels show those parts of the object's surface which are at a similar distance to the viewer as the polygon. In other words, every layer in the impostor depicts a slice of the object at some distance to the viewer (see figure 1). The more slices are used the better the approximation.

3.1 Texture generation and storage

The color information in the texture is generated in exactly the same way as for impostors: a tight viewing frustum is placed around the object and a view of the object is rendered from the current point of view. The object's surface can be textured as well. In this paper, however, object image and texture interchangeably refer to the image of the object which is mapped onto the layers. Texel refers to one pixel in such a texture. Final image refers to what is eventually displayed on the screen.

When the object's image is generated, instead of discarding the depth-buffer contents (the information describing the object's three-dimensional shape) the depth buffer contents are retained and stored together with the color information for each pixel. From this RGBz information a texture is defined in the format RGBα (with z in the α component) available on today's graphics workstations. (This is in contrast to McMillan's plenoptic modeling approach [16] for which disparity values must be derived from depth values).

3.2 Using the depth information in the texture

Instead of rendering one transparent polygon as with impostors several polygonal layers are rendered as a pyramidal stack having the point of view as it's apex (see figure 1). The object's image was generated from the current viewpoint and is mapped onto the layers as a partially transparent texture. In each layer only those areas of the image are drawn where the stored z-value closely matches the layer's distance from the viewpoint. A pixel test based on the pixel's alpha value (similar to the depth test) is used to select these areas (described in detail in the implementation section 4).

If disjoint depth intervals are drawn on each layer, any deviation of the point of view from the point of texture generation results in cracks being visible between the individual layers. These cracks are avoided by drawing slightly overlapping depth intervals onto every layer (see figure 2).

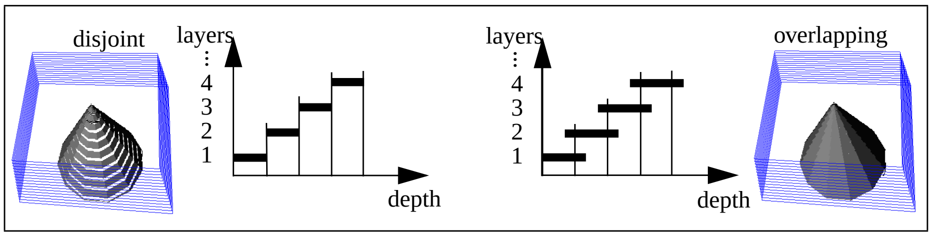

Fig. 2: Layered impostor for an approximated cone: overlaying depth intervals are assigned to each layer to avoid cracks to appear between layers when moving the viewpoint.

3.3 Texture lifetime and texture regeneration

With an increasing number of layers in the impostor the drawn pixels match the spatial location of the object's surface closer than the pixels on a single impostor polygon. As a result, in response to changes of the point of view the drawn pixels move in much the same way as the points on the surface of the object. Consequently, the layered impostor approximates the object's appearance over a much larger set of viewpoints than a single polygon. Compared to impostors the incurred error due to a mismatch of the spatial location of texels and object's surface points is halved by doubling the layers in the impostor.

The error introduced by impostors can be controlled by observing error angles [19]. Their derivation is repeated here briefly for comparison. When an impostor is used instead of an object, points on the object are drawn into the final image as texels on a textured polygon. When viewing the impostor from a point of view different from the one for which the texture was generated, a discrepancy between the point on the object's surface and the point on the texture will appear. The angle under which these two points are observed by the viewer can be calculated and used to limit the amount of error introduced. Whenever a given error threshold is exceeded (say the angle under which a pixel is observed on screen) the impostor's texture is regenerated.

A conservative estimation for the maximum error angle for all points on the object's surface can be calculated from a bounding volume. In figure 3 the error angles for extreme point's on the object's bounding box under orthogonal movements of the viewpoint are shaded in grey. These error angles are the result of the maximal distance of the object's points from the impostor. With layered impostors this distance is reduced with each additional layer. Error angles diminish accordingly as is evident when comparing figure 4 to figure 3.

3.4 Warping with variable accuracy

When drawing a layered impostor into the final image, the number of used layers can be selected. By not considering all the bits of the depth component in the texture, less

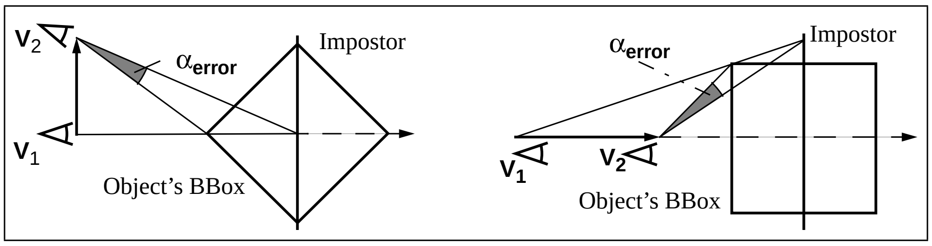

Fig. 3: Impostors: error angles on extreme points of the bbox for orthogonal motion cases.

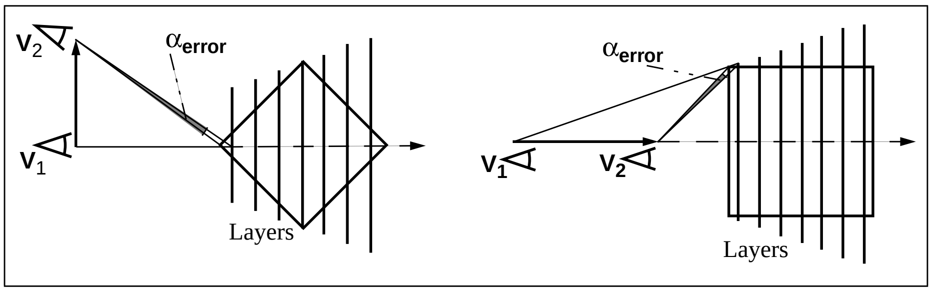

Fig. 4: Layered impostors: error angles on extreme points of the bbox for motion cases as above.

layers are obtained. As a result, the image warp is performed with less and less accuracy because texels from larger and larger depth intervals are warped in a uniform way (see figure 1).

Adapting the accuracy of the image warp is desirable because the warping accuracy required depends on the amount by which the viewpoint has moved from the point of view $\mathbf{V}_1$ for which the texture was generated. As long as the viewpoint is coincident with the point of texture generation, one layer in the impostor is a sufficient approximation to the object's appearance.

The more the point of view moves away from the point of texture generation, the worse the approximation of the object's appearance by a single textured polygon. A warp of the object's image is required. However, for small deviations, a crude approximation to the image warp is sufficient. Layered impostors support different approximation accuracies by varying the number of layers used to draw the impostor into the final image.

3.5 Occlusion errors

When warping an image depicting a three-dimensional scene, occlusion errors can result. Consider the simple scene depicted on the top of figure 5. The cone, cube and cylinder hide portions of the wall behind them. When moving away from this point of view, the previously hidden areas of the wall become visible. When warping the image obtained from the initial point of view these areas are missing and holes in the final image will result (see figure 5 bottom). These holes are the bigger, the further the objects are separated in depth. Errors caused by objects occluding each other are called inter-object occlusion errors in this paper.

As the distance between two objects in a scene is potentially unbounded, inter-object occlusion errors are unbounded when warping a single image per scene and cause objectionable warping artifacts in the final image. The objects mutually occlude

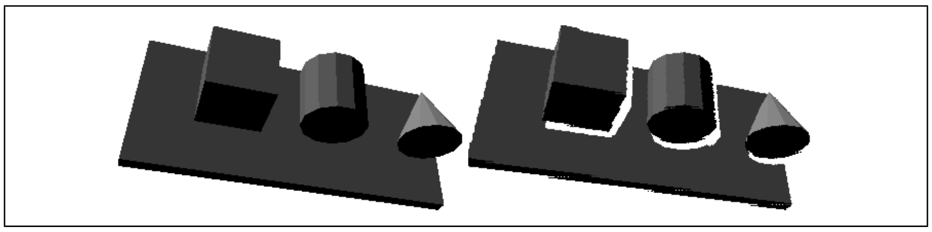

Fig. 5: Inter-object occlusion errors: surface parts previously hidden by another object become visible. In one image for all the objects information on these surface parts is missing.

each other and large holes appear in the final image due to the motion of objects in the image as caused by movements of the viewpoint. As an anticipation of the results section 6 consider the additional images of the scenes in figure 8. Each object occludes a large portion of the floor. When warping a single image of the whole scene large holes would appear around the objects in the floor as the camera orbits around the scene.

By using a separate image for each object and warping these images individually, inter-object occlusion errors are avoided. This is the approach taken by layered impostors: there is a layered impostor with a texture for each object in the scene. Object images are warped individually and are z-buffered into the final image. As a result, inter-object occlusion errors are avoided.

Another class of errors remains, namely intra-object occlusion errors (see figure 6). Concave objects possibly occlude portions of their own surface, resulting in similar holes to appear in individually warped object images. In contrast to inter-object occlusion errors, the size of an object is bounded and so are the artifacts introduced by intra-object occlusion errors. The error is proportional to the maximum depth discontinuity in the image which - unfortunately - is not reported by the graphics hardware. Having it allows to bound both inter- and intra-object occlusion errors.

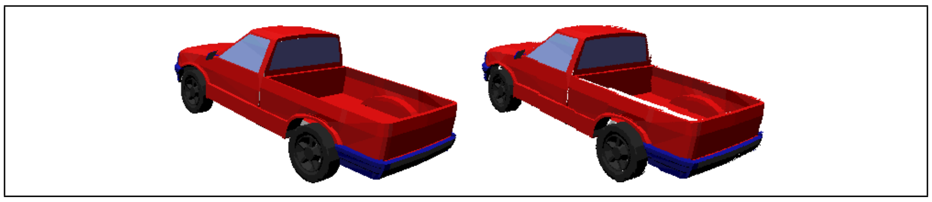

Fig. 6: Intra-object occlusion errors: a concave object hides parts of its surface. Even if one image is used per object, information on these surface parts is not available for warping.

4 Implementation

An implementation of layered impostors was done using the OpenGL graphics library which runs on a variety of platforms. As a prerequisite blending of RGB colors based on alpha values is disabled, so that every drawn pixel is always opaque.

The OpenGL standard provides an alpha test function to restrict the drawing of pixels to certain areas of a textured polygon. The alpha component stored in the texture is compared to a reference alpha value and pixels can selectively be drawn based on the outcome of the comparison. Drawing pixels with equal, smaller or greater alpha values than the reference value is possible.

For layered impostors depth values are stored in the alpha component of the texture. Ideally one would like to select pixels for a certain interval of depth values, namely the interval around the depth of each layer. As a test for depth intervals is not available in OpenGL, the implementation tests for equality of depth values and relies on the finite accuracy of depth values to actually result in an interval of depth values to be selected.

Scaling depth values by a factor smaller than one maps several depth values onto one (e.g. scaling by 0.5 removes one bit of accuracy and two values are mapped onto one). Assume four bits of accuracy in the following example. When the most significant bit of the depth value is set[1] and the scaling factor is increased slightly to the next representable number larger than 0.5 (with 4 bits 0.5 is represented as 1000_2, the next larger representable value is 1001_2 or 0.5625) the behavior shown in table 1 is achieved. Requiring the most significant bit to equal one insures that incrementing the scale factor actually influences the multiplication result.

b	b * 1000_2	b * 1001_2
1000	0100	0100
1001	0100	0101*
1010	0101*	0101*
1011	0101*	0110
1100	0110	0110
1101	0110	0111*
1110	0111*	0111*
1111	0111*	1000

Table 1: Fixed point value multiplication using 4 bits of accuracy. Values from 0.5 (1000_2) to 1.0 (1111_2) are scaled by 0.5 (middle column) and the next larger value 0.5625 (1001_2, right).

The regular pattern (highlighted by * in the above table) is used to select overlapping depth intervals. For successive layers depth values are multiplied alternately by 1000_2 or 1001_2 and the desired interval is selected by testing for equality with the multiplication result given in the table. This depth interval selection is depicted in figure 7. The same behavior is also obtained for more bits of accuracy and can be used to distinguish a larger number of overlapping depth intervals. This implementation has been tested on SGI Reality Engine, SGI O2 and SGI INDIGO graphics.

5 Optimizations

The layered impostor method provides a number of tunable parameters which can be used to optimize the drawing performance. As was already mentioned, the image warping accuracy can be adapted by varying the number of layers drawn per impostor. In the current implementation the number of layers is dynamically adapted to the error which must be compensated for. To achieve this, the scale factor is changed accordingly.

Another improvement of the basic algorithm is to provide a more uniform frame rate. As the number of textures to be regenerated varies from frame to frame, the frame rate varies as well. For walkthrough applications it has proven sufficient to update no more than one texture per frame. With a high frame rate this results in several texture

1.Requiring the highest bit of the depth values to be set means that the frustum surrounding the object must be made twice as deep with only the back half being occupied by the object.

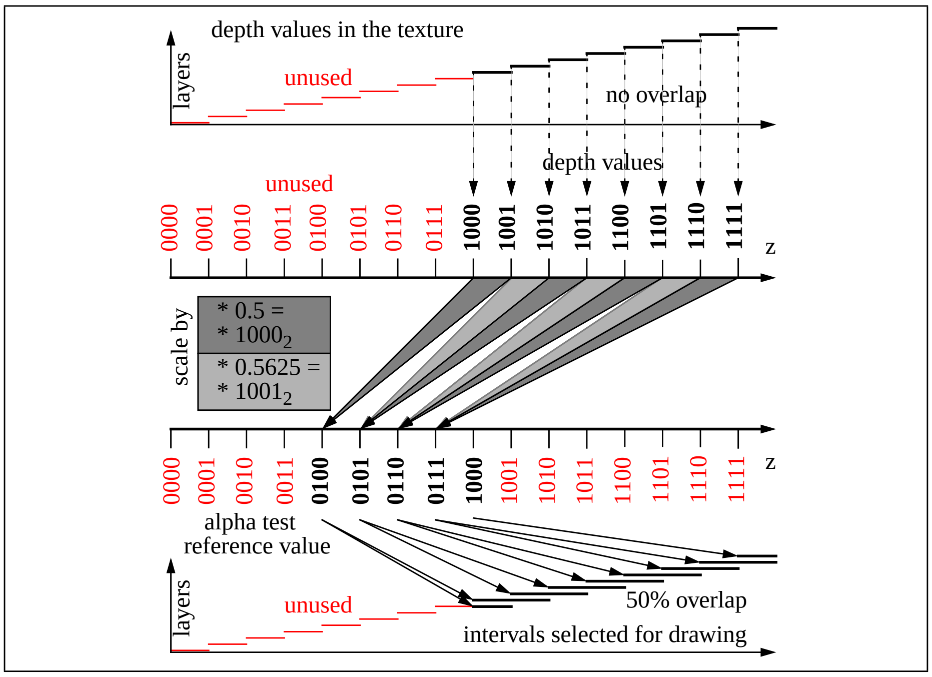

Fig. 7: Remapping depth values to obtain overlapping intervals.

updates per second. The warping accuracy of layered impostors is sufficient to hide the sparse updates of each texture even under object rotations.

While updating only one object image per frame guarantees that the frame-rate does not drop below a certain threshold, variations remain because in some frames no updates are necessary. These variations can be avoided with the saved time put to good use by updating exactly one object image per frame. This strategy has the advantage that any warping artifacts are kept to a minimum during slow movement of the point of view. When no object image would need to be regenerated under given error tolerance settings, the one with the highest error is updated. In particular intra-object occlusion errors are removed whenever possible.

6 Results and Discussion

The rendering performance of the layered impostor approach was tested on two scenes with different characteristics. The first one depicts a collection of cars where the complexity of the cars and other objects in the scene varies between 2816 and 21492 polygons. The second scene is made up from eighteen copies of one car model (the one with 21492 polygons).

Performance measurements were carried out on two machines, one from the low price range (an SGI O2 with 128MB memory and 175MHz R10000 CPU) and one high end graphics workstation (an SGI Onyx with RealityEngine2 graphics, 128MB memory and 195MHz R10000 CPU).

In the frame sequence measured the camera orbits in a circle around one of the scenes. This motion has been chosen because under rotation of the scene in front of the viewer the appearance of individual objects changes fastest. In all the previously pub-

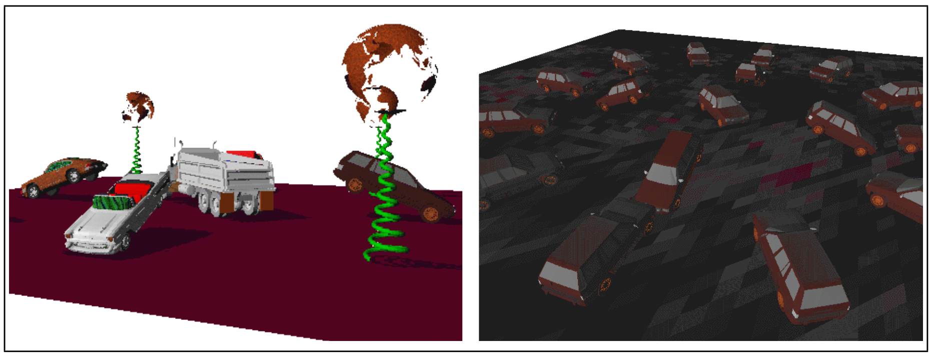

Fig. 8: Rendering with layered impostors: one texture update per frame removes most artifacts. **Left:** cars: 319847 vertices, 149488 polygons **Right:** jeeps: 1096096 vertices, 365368 polygons

lished image-based acceleration methods mentioned in the related work section this motion has been avoided. With layered impostors and their ability to warp object images this motion of the camera becomes feasible.

The window size was set to 768 by 576 pixels, the texture size was limited to 128 by 128 pixels which resulted in about one megabyte of texture memory being used in all cases. The maximal tolerable error angle was set to the angle under which one pixel is observed. As up to 32 layers are used per impostor this still results in an increase of the object image's lifetime by more than an order of magnitude in comparison to impostors.

For obtaining a more uniform frame rate with layered impostors the regeneration of exactly one object image per frame was forced by limiting the number of regenerations to one and using a very small error angle (much less than a pixel).

Average frame rates are given in table 2 for the two graphics platforms mentioned and the following three different rendering techniques:

- polygonal geometry only
- layered impostors with an error angle corresponding to viewing one pixel of the final image
- layered impostors with one texture update per frame (the one with the largest error angle)

rendering method	O2	RE2
	Car scene	
polygonal geometry	1.0470 Hz σ = 0.00413	3.23215 Hz σ = 0.0317553
layered impostors	8.4143 Hz σ = 1.69594	47.1101 Hz σ = 12.6673
layered impostors (1 update/frame)	4.3317 Hz σ = 0.995001	20.8030 Hz σ = 7.74516
	Jeep scene	
polygonal geometry	0.4853 Hz σ = 0.00606	1.42590 Hz σ = 0.0154931
layered impostors	3.9107 Hz σ = 0.961839	22.6625 Hz σ = 9.22151
layered impostors (1 update/frame)	2.4541 Hz σ = 0.0433292	11.2178 Hz σ = 0.470177

Table 2: Average frame rates and frame rate variation for the different rendering methods.

Rendering original geometry results in a very uniform frame rate because the same amount of geometry is visible in all frames. This rendering option is also slowest.

Layered impostors warp the object images and can be used for many frames even with the camera orbiting around the scene. As a result, most frames can be generated without regenerating textures and the frame rate is high. Whenever an object image must be regenerated the frame rate drops but usually no more than one image needs to be regenerated per frame. As described in the optimizations section a maximum of one update can be enforced without causing visible artifacts. It is a question of human factors whether it is a good thing to have such a high frame rate with occasional “dropouts”. If a uniform frame rate is desired, the fast frames can be delayed to match the frame rate of those frames, when a texture needs regeneration.

Some variations in the frame rate is caused by the dynamic adaptation of the warping accuracy even if no object images need to be regenerated. These variations are mainly due to the fact that the current implementation only supports powers of two as the number of layers in an impostor. Finer control over the alpha test function would be required to remove this restriction.

Instead of delaying the fast frames it is better to regenerate one texture every frame as described in the optimizations section. Now the remaining frame rate variations are due to the varying complexity of the objects which need regeneration. Texture regeneration time dominates the whole rendering process due to the overhead of texture definitions in OpenGL.

In a scene where objects of similar complexity are replaced with layered impostors, a uniform frame rate is achieved (as in the jeep scene). Objects of uniform complexity can either be created when modeling the scene or by automatically clustering or partitioning too simple or too complex objects. Partitioning can be done arbitrarily as in the image caching approaches [20][22] as visibility is correctly resolved from the approximated depth values. In the current implementation objects of equal complexity must be created during scene modeling. If off-screen frame buffers were available, texture regeneration for complex objects could be distributed over several frames.

In general, the amount of experimentation possible with this OpenGL implementation of layered impostors is limited by the inflexibility of the alpha test functionality. Further evaluation of finer choices on the number of layers would be useful and different amounts of overlap among layers should be tried.

In addition more flexible rendering architectures have already been proposed from which layered impostors would benefit. For example in the talisman architecture [24] image layers are composited into the final image during video signal generation. This has the advantage that object-image regeneration and final image composition do not share one frame time but are pipelined over two frames.

As with the talisman architecture layered impostors allow to adapt the resolution on individual object images to the importance of the object. For example, distant background objects can be rendered into textures containing less texels than the pixels the object covers on screen. The talisman architecture offers high-quality filtering to decrease the so caused “pixelization” of object images. With OpenGL this is not possible because filtering RGBα textures would result in an interpolation of depth values of layered impostors. This interpolation disturbs the selection of depth intervals by testing for equality of finite accuracy depth values.

7 Conclusions and Future Work

This paper presented layered impostors, a novel object centered approach to render objects from depth augmented textures. Layered impostors unify image generation and warping into a single real-time rendering framework. They warp the images using the images’ depth values to closely mimic the appearance of the depicted object from new points of view.

As with impostors only one texture is kept per object and reused as long as the errors introduced by the use of layered impostors remains below a given error threshold. The main advantage of layered impostors over dynamically generated impostors is the fact that they can be reused by an order of magnitude longer. Other advantages of layered impostors include:

- The drawing time for layered impostors is independent of the complexity of the represented object. Consequently, for sufficiently complex objects drawing time is saved by using layered impostors.
- As one layered impostor is used per object layered impostors work in dynamic scenes. Each layered impostor can be moved around individually.
- Layered impostors do not use more memory than impostors for storing the depth augmented texture. The usual eight bits of alpha are replaced by eight bits of depth.
- Approximate z-values are written into the depth buffer for correct visibility among layered impostors and objects rendered with polygonal primitives.
- The number of layers used in any layered impostor can be adapted to the required warping accuracy and the available fill-rate of the graphics hardware.
- Layered impostors cache the images of a scene on a per-object basis. Therefore, inter-object occlusion errors are avoided.
- Layered impostors are a hardware accelerated approach to 3D image warping.
- The warping quality is sufficient to accommodate object rotations.

Some disadvantages of layered impostors still need to be addressed:

- Large, highly overlapping polygons are used to draw the layers in the impostor. This results in high fill-rate requirements especially with many layers.
- Only one object image is warped. As a result, surface parts which were occluded when generating the image cause holes to appear in the warped image (intra-object occlusion errors). Always regenerating one texture per frame can partially compensate for this.

While the fillrate requirements may seem high polygonal scenes in real-time rendering typically cause a bottleneck in the vertex transformation stage of the rendering pipeline. Layered impostors provide a means to shift some of this work to the pixel processing stage and consequently better balance the work between the two stages.

One approach to deal with the fill rate requirements would be to detect and make use of the large amounts of transparent texels per layer. Intelligent pixel-fill hardware could detect these transparent areas and rapidly skip over them.

More flexible alpha test functions would allow further variations of the number of layers used and the amount of overlap between depth intervals.

Filling the holes in the warped image could be addressed by using more than one source image to calculate the warp. In particular, if a prediction of the user’s motion is available, the new images could always be generated along the user’s future trajectory. When the point of view for the image to be generated is within the convex hull of the viewpoints of available images, holes can be filled to a great extent.

References

[1] Aliaga, Daniel G., *“Visualization of Complex Models Using Dynamic Texture-based Simplification”*, IEEE Visualization ‘96, pp 101-106, Oct 28-Nov 1, 1996.

[2] Chen, Shenchang Eric and Lance Williams, *“View Interpolation for Image Synthesis”*, Computer Graphics (SIGGRAPH ‘93) 27 (August 1993) pp 279-288.

[3] Chen, Shenchang Eric, *“Quicktime VR - An Image-Based Approach to Virtual Environment Navigation”*, Computer Graphics (SIGGRAPH ‘95) (August 1995) pp 29-38.

[4] Dally, William J., Leonard McMillan, Gary Bishop, and Henry Fuchs, *“The Delta Tree: An Object-Centered Approach to Image-Based Rendering”*, MIT AI Lab Technical Memo 1604, May 1996.

[5] Darsa, Lucia, Bruno Costa and Amitabh Varshney, *“Navigating Static Environments Using Image-Space Simplification and Morphing”*, Proceedings of the Symposium on 3D Interactive Graphics, April 27 - 30, 1997, Providence, RI, pp 25 - 34.

[6] Debevec, Paul E., Camillo J. Taylor and Jitendra Malik, *“Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach”*, Computer Graphics (SIGGRAPH ‘96) (August 1996) pp 11-20.

[7] Dorsey, Julie, Jim Arvo, and Donald Greenberg, *“Interactive Design of Complex Time-Dependent Lighting”*, IEEE Computer Graphics and Applications. 15(2) (1995), 26-36.

[8] Duff, Tom, *“Compositing 3-D Rendered Images”*, Computer Graphics (SIGGRAPH ‘85) 19 3 (July 1985) pp 155-162.

[9] Gortler, Steven J., Li-wei He and Michael F. Cohen, “Rendering Layered Depth Images”, Microsoft Technical Report MSTR-TR-97-09, March 1997.

[10] Laveau, Stéphane and Olivier Faugeras, *“3-D Scene Representation as a Collection of Images and Fundamental Matrices”*, INRIA Technical Report N˚2205, February 1994.

[11] Maciel, Paulo W. and Peter Shirley, *“Visual Navigation of Large Environments Using Textured Clusters”*, Symposium on Interactive 3D Graphics (April 1995) pp 95-102.

[12] Mark, William R., Leonard McMillan and Gary Bishop, *“Post-Rendering 3D Warping”*, Proceedings of the 1997 Symposium on Interactive 3D Graphics (Providence, RI), April 27-30, 1997, pp 7-16.

[13] Mark, William R., Gary Bishop, *“Memory Access Patterns of Occlusion-Compatible 3D Image Warping”*, Proceedings of the 1997 SIGGRAPH / Eurographics Workshop on Graphics Hardware (Los Angeles, California), August 3-4 1997, pp 35-44.

[14] Max, Nelson, *“Hierarchical Rendering of Trees from Precomputed Multi-Layer Z-Buffers”*, Proceedings of the 7th Eurographics Workshop on Rendering ‘96, Porto, Portugal, pp165-174.

[15] Mazuryk, Thomasz and Michael Gervautz, *“Two-Step Prediction and Image Deflection for Exact Head Tracking in Virtual Environments”*, Computer Graphics Forum (EUROGRAPHICS ‘95) 14 3 pp 29-41.

[16] McMillan, Leonard and Gary Bishop, *“Plenoptic Modelling: An Image-Based Rendering System”*, Computer Graphics (SIGGRAPH ‘95), (August 1995) pp 39-46.

[17] Pulli, Kari, Michael Cohen, Tom Duchamp, Hugues Hoppe, Linda Shapiro, and Werner Stuetzle, *“View-based Rendering: Visualizing Real Objects from Scanned Range and Color Data”*, Proceedings of 8th Eurographics Workshop on Rendering, St. Etienne, France, June 1997, pp 23-34.

[18] Regan, Matthew and Ronald Post, *“Priority Rendering with a Virtual Reality Address Recalculation Pipeline”*, Computer Graphics (SIGGRAPH ‘94) (July 1994) pp 155-162.

[19] Schaufler, Gernot, *“Exploiting Frame to Frame Coherence in a Virtual Reality System”*, VRAIS ‘96, Santa Cruz, California (April 1996) pp 95-102.

[20] Schaufler, Gernot and Wolfgang Stürzlinger, *“A Three-Dimensional Image Cache for Virtual Reality”*, EUROGRAPHICS ‘96, (August 1996), 15 3, pp 227-236.

[21] Schaufler, Gernot, *“Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes”*, Proceedings of the 8th Eurographics Workshop on Rendering ‘97, St. Etienne, France, June 16-18, 1997, pp 151-162.

[22] Shade, Jonathan, Dani Lischinski, David H. Salesin, Tony DeRose and John Snyder, *“Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments”*, Computer Graphics (SIGGRAPH ‘96), (August 1996) pp 75-82.

[23] Sillion, François X., George Drettakis and Benoit Bodelet, *“Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery”*, Proceedings of Eurographics’97, September 4-8, 1997, pp 207-218.

[24] Torborg, Jay and James T. Kajiya, *“Talisman: Commodity Real-time 3D Graphics for the PC”*, Computer Graphics (SIGGRAPH ‘96), (August 1996) pp 353-363.