Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes

Gernot Schaufler

GUP, Johannes Kepler Universität Linz
Altenbergerstr. 69, A - 4040 Linz, Austria / EUROPE
schaufler@gup.uni-linz.ac.at

Abstract. This paper proposes a simple augmentation to texture mapping hardware which produces the correct depth buffer content and hence correct visibility when replacing complex objects by partially transparent textured polygons. Rendering such polygons exploits frame-to-frame coherence in image sequences of dynamic scenes.
Correct depth values are obtained by keeping a small depth delta for every texel which represents the texel's deviation from the textured polygon. The polygon's depth values are modified at every pixel to match the depicted object's geometry.

1 Introduction and Motivation

Rendering acceleration in three-dimensional computer graphics is usually achieved with dedicated graphics hardware. Available accelerators generate up to sixty frames per second of moderately complex scenes. Their major limitation is that every frame is rendered from scratch even though smooth frame sequences contain a great amount of coherence. Nevertheless depth-buffered scan conversion of polygons enjoys great popularity because visibility is correctly resolved for any point of view and any rendering sequence of the primitives.

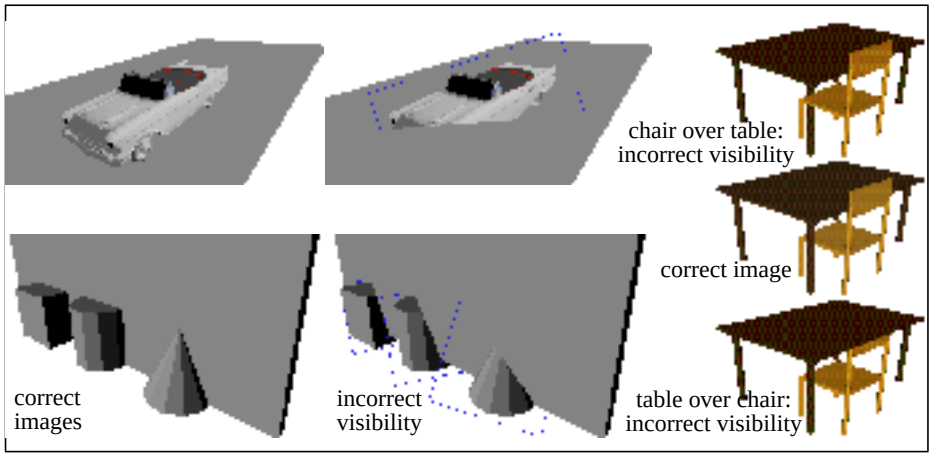

Fig. 1: Left and Middle: Incorrect visibility due to interpenetrating polygons when individual objects are replaced by partially-transparent textures.
Right: Incorrect visibility due to layering of object images.

In contrast image-based rendering uses pre-generated or recently generated images to obtain the final view. The rendering cost depends on the number of pixels in the final image and not on the scene complexity. In smooth animation sequences object images

are not rendered directly into the final image but into dedicated buffers. Texture mapping and image layering have been used to merge this image data into the final frames to be displayed. Image data can be re-used as long as the geometric and photometric errors remain below a given threshold. Approximations to these errors allow to quickly determine if image data can be reused or must be refreshed.

While visibility is correctly resolved on individual texture maps and within single image layers, assigning depth priorities to different layers does not correctly resolve visibility in general (see figure 1). Only one object can be in front of the other. Replacing complex objects by partially transparent textured polygons also produces visibility errors because polygons mutually intersect or intersect other geometry (see figure 1).

In static scenes grouping close or interpenetrating objects into one texture or image layer avoids visibility problems at the cost of losing the convenience to render objects in any sequence. In dynamic scenes such a static grouping is not possible as the spatial relations among objects change over time.

So far mainly independent polygons have proven general and efficient enough to generate smooth animations of dynamic scenes with considerable complexity in real-time. In order to also exploit coherence in dynamic scenes, a new more powerful rendering primitive instead of the textured polygon is needed. Such a primitive must fit into the well-established context of depth-buffered rendering so that highly dynamic objects exhibiting little or no coherence can still be rendered efficiently as polygonal geometry.

2 Related Work

Correct visibility of arbitrary surfaces on rastered images is often achieved with the depth buffer algorithm [3]. Efficient implementations work incrementally in screen space as perspective projections preserve lines and planes. However, this projection is not affine but requires the “perspective division” [6].

A number of such images can be composited with correct visibility by making use of each image’s depth buffer contents: the pixel visible in the final image is the one with the smallest depth value [5]. Regan et al. [11] have designed a custom video hardware which performs such depth composition of multiple images during video signal generation. After grouping the objects in the scene by their distance to the viewer, more distant objects are rendered less frequently as their images change slower. The incurred cost is the multiplication of frame and depth buffer memory requirements.

With texture mapping hardware (e.g. [1]) image data can be incorporated into object space and not only into image space. A number of interesting shading effects are possible requiring multi-pass rendering [15] and perspective-correct texture mapping [7]. In particular real-time shadows require that depth values are calculated both in eye and lightsource coordinates.

Maciel et al. [8] incorporate pre-rendered image data into the rendering process for walk-throughs of static environments. The object images are mapped onto transparent polygons which are positioned and oriented properly with respect to the point of view in order to closely mimic the appearance of the objects.

Schaufler introduced the concept of dynamically generated impostors [12]. Object images are not pre-generated but retained from previous frames and re-used as long as possible [13]. In some cases visibility errors occur because a flat polygon does not capture the actual geometry of an object. Rendering the polygon instead of the geometry does not produce the same depth values and visibility errors result, if objects are too close or interpenetrating (see figure 1).

Shade et al. [16] and Schaufler et al. [14] independently developed a solution to this visibility problem for static scenes. Scene sub-volumes are replaced by impostors instead of replacing individual objects. Correct visibility is guaranteed by using a spatial hierarchy to depth-sort impostors.

Image caching is simple enough to be efficiently implemented in low-cost hardware: the Talisman system [17] generates the final image from warped image layers (similar to impostors) during video signal generation. The authors state that "independent prioritized layers" replace "non-interpenetrating objects". No solution is given for the problem depicted in figure 1. Open questions are what is to be considered an independent object (what about very large objects such as terrain?) and when to change the priority of a layer.

The next section introduces a new trade-off between geometric modeling and image-based modeling. The proposed way of writing texture mapped polygons into the frame and depth buffers avoids the visibility problems of dynamically generated impostors. The major advantage is that impostors, hierarchical image caches and traditional geometry can be rendered in any desired sequence or mixture because the depth buffer values are correct at every pixel. This mixture allows image caching to be done for dynamic scenes as will be shown in section 7.

3 Nailboards

Nailboards are polygons onto which an RGBΔ texture is mapped to mimic the appearance of a complex object. For every pixel in the texture (texel) the additional component Δ measures how far the object's point depicted on this texel deviates from the polygon. The object's image and Δ values are generated on demand and are reused over several frames as long as the geometric and photometric errors remain below a certain threshold.

When the texture of the nailboard is generated from object geometry, the Δ values are taken from the depth buffer and are stored in the texture together with the RGB values of the image. When the nailboard is rendered, the Δ values are used to change the polygon's depth values into the depth values for the object depicted on the nailboard.
Figure 2 shows the four coordinate spaces involved in the correction of depth values: *texture space* (denoted by superscript ^{t}), *nailboard space* (^{n}), *eye space* (^{e}) and *screen space* (^{s}).
The viewing frustum used to render the object image is defined in *nailboard space*. It contains the nailboard polygon which is normal to the *z*-axis of nailboard space. The extent of the frustum is determined from a bounding volume (e.g. a bounding box) known to contain the object. The eye is at the origin.

After rendering the object to obtain RGB and Δ values the depth buffer contains depth values in *texture space* which is related to nailboard space by the perspective projection matrix followed by the perspective divide [10]:

$$\begin{bmatrix} x^t \\ y^t \\ z^t \\ w^t \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\left(\frac{f+n}{f-n}\right) & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x^n \\ y^n \\ z^n \\ 1 \end{bmatrix} \quad \text{and} \quad X^t = \begin{bmatrix} \frac{x^t}{w^t} \\ \frac{y^t}{w^t} \\ \frac{z^t}{w^t} \end{bmatrix} \tag{1}$$

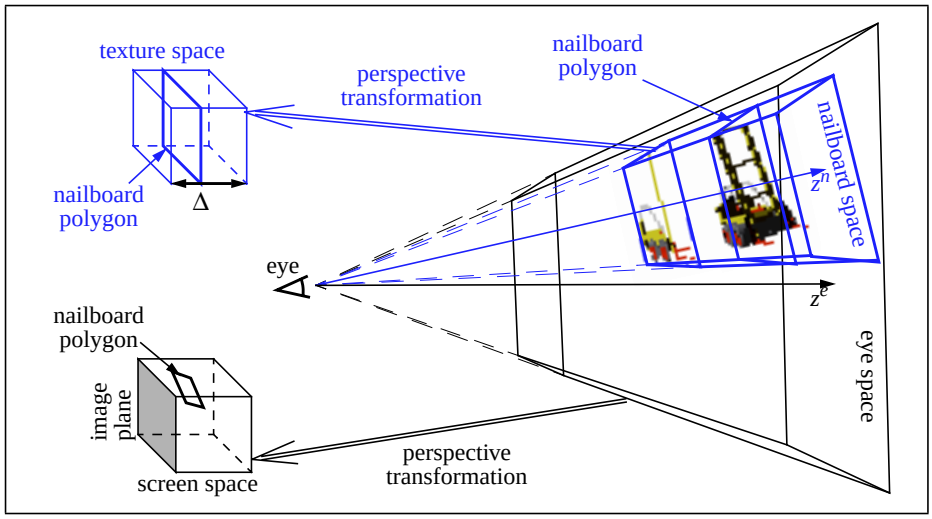

Fig. 2: Viewing the original object and it's nailboard:
nailboard space is embedded into eye space to obtain correct depth values.

where $(x^n,y^n,z^n,1)^T$ is a point in nailboard space, $X^t=(x^t/w^t,y^t/w^t,z^t/w^t)^T$ is the same point in texture space, n is the distance to the near clipping plane (which is also the image plane for generating the texture), f is the distance to the far clipping plane, l and r delimit the horizontal extent of the image, t and b delimit its vertical extent as shown in figure 3 (looking down the negative z-axis in a right-handed coordinate system):

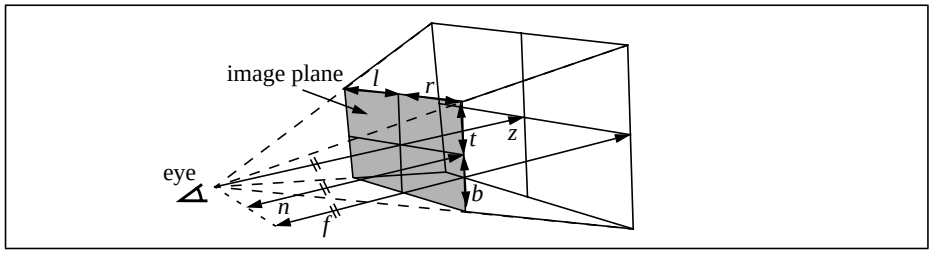

Fig. 3: Parameters of the viewing frustum.

In equation (1) division by zero only occurs for degenerate image extents as l and b are signed values and usually negative. Equation (1) maps the viewing frustum onto a cube centered around the origin. The general form of the function relating z^n in nailboard space to z^t/w^t in texture space is

$$\frac{z^t}{w^t} = \lambda + \frac{\mu}{z^n} \quad \text{with} \quad \lambda = \frac{f+n}{f-n} \quad \text{and} \quad \mu = \frac{2fn}{f-n} \tag{2}$$

Please note that this function only depends on the near and far distances: all points on planes parallel to the image plane have the same depth value. In texture space the nailboard polygon corresponds to one such plane parallel to the near plane.

The third space shown in figure 2 is *eye space*. In both eye space and nailboard

space the eye is at the origin of the coordinate system. When rendering the final image, points $(x^e,y^e,z^e,1)^T$ in eye space are mapped to $(x^s/w^s,y^s/w^s,z^s/w^s)^T$ in *screen space* by the same matrix equation as above (equation (1) for a different frustum) followed again by the perspective divide. After resolving visibility, z^s/w^s is discarded to obtain coordinates $(x^s/w^s,y^s/w^s)^T$ on screen (in the final image).

The depth buffer of the final image contains depth values z^s/w^s in screen space. Therefore, using depth values from one perspective projection (texture space) to correct depth values in another space (screen space) requires reversing the perspective divide, transforming to the second space and performing the perspective division for this space.

With this mathematical background a nailboard implementation is possible (actually a first reference and prototype implementation worked in this way). The nailboard is rendered into the color buffer as the usual textured polygon. However, the depth values of the nailboard polygon are ignored because they do not match the geometry of the object depicted on the nailboard. As the object's depth values were obtained in texture space they must be taken to screen space via nailboard space and eye space. This requires inverting the perspective projection, transforming to eye space and performing the perspective projection to obtain the object's depth values in screen space. These numerical operations include two divisions and a full matrix multiplication and are required at every pixel. Keeping depth values with 32 bit resolution roughly doubles the memory requirements compared to RGB textures. All this should be avoided.

4 Efficiency Improvements

The proposed efficient implementation of nailboards is based on the fact that the depth values of the textured polygon are computed during span interpolation on graphics accelerators. These depth values provide a reference in screen space which is then corrected by a small depth Δ to arrive at the depth values matching the depicted object.

The nailboard polygon's depth values have a number of desirable properties: First, the object's depth deviations from the polygon are small compared to the depth extent of the whole viewing frustum. Second, the position and orientation of the polygon in eye space is determined by the transformation relating nailboard space and eye space. Applying the depth Δ relative to the depth of the polygon accounts for most of the effects of the transformation from nailboard space to eye space (one additional term must be considered, as will be explained in section 4.2) as depth correction only works along one line of sight changing the pixel's depth value and not it's position on screen.

Efficient correction of depth values must work in screen space. It must use depth values from texture space as these are available in the depth buffer. Moreover, correction without divisions is desirable for a hardware implementation.

This section introduces two simplifications to allow efficient implementation of nailboards: one determines the number of bits actually required to store the Δ component of the texture; the other removes the two divisions per pixel and approximates the transformation from texture space to screen space.

4.1 Cutting Down Memory Requirements

Accurate depth buffering requires between twenty and thirty-two bits per pixel because after the perspective division of equation (2) the relationship between depth values and the distance from the viewer is no longer a linear but a rational function [6]. Two plots of this relationship are shown in figure 4 for different ratios of near to far plane distances (please note the different scales on the axes). The mapping of the rational func-

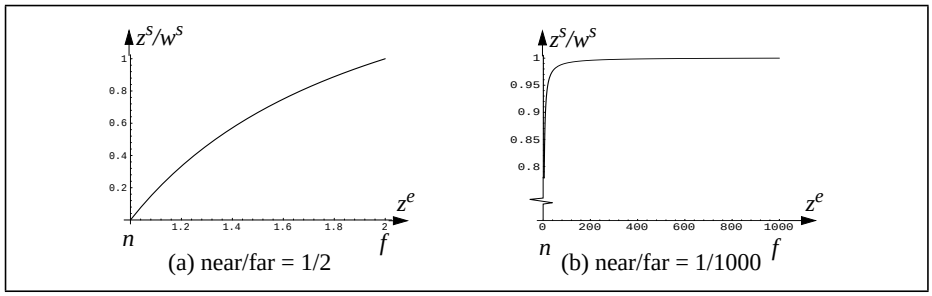

Fig. 4: Depth value as a function of distance to the viewer for different near/far ratios (near=1).

tion wastes a lot of accuracy in the back of the frustum. For example at a near/far ratio of 1/1000 all distances exceeding 200 are mapped onto values between 0.99 and 1 in the depth buffer (see figure 4 (b)).

This is not the case when a tight frustum is used around a single object. Then the ratio of near/far approaches one and the mapping of distance to depth values becomes more and more linear (see figure 4 (a)). So significantly less bits suffice to represent the different distances between the near and the far plane.

The implementation described in the next section uses only eight bits to represent Δ. A visual comparison of the results[1] obtained with two, four, six, eight and sixteen bits is shown in figure 5.

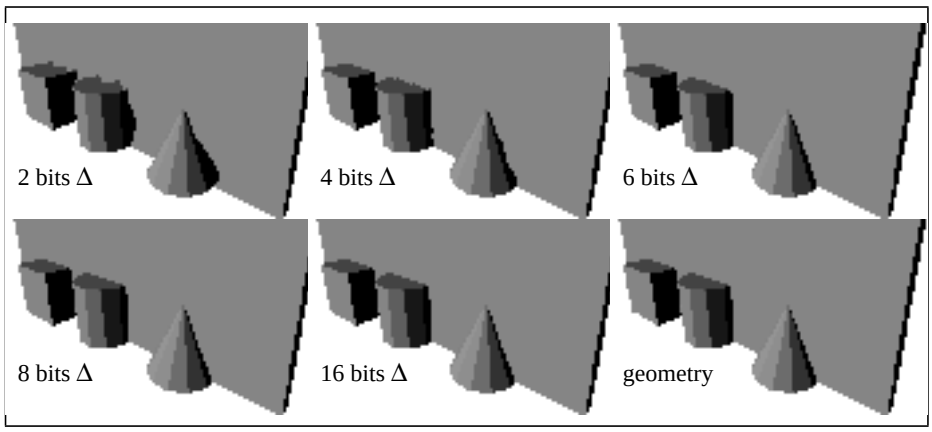

Fig. 5: Comparison of depth accuracy with different numbers of bits used to store Δ values.

The zigzag lines between the wall and the other solids in the first three images result from the intersection of the wall with the parallel planes defined by discrete Δ values. Only minor artifacts are visible with six bits Δ and artifacts are unnoticeable with eight bits. Both the images generated with eight and sixteen bits cannot be distinguished from the image rendered from original geometry with 32 bit z-buffering.

On a single nailboard aliasing artifacts of this kind do not occur because for generating the texture the full depth buffer resolution is used. However, only the most signif-

1. These images have been rendered using also the approximations described in the next section 4.2.

icant eight bits of the usual thirty-two bits are stored in the RGBΔ texture.
Impostors described previously [12][14][16] use the α channel for the transparent parts of the texture. With nailboards α bits are avoided by reserving one depth value for the transparent portions of the texture (e.g. the initial far distance value).

4.2 Approximating the Divisions and the Transformation

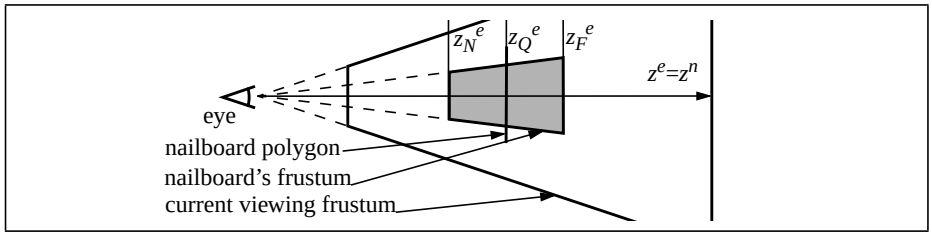

Fig. 6: Simplest case setup between current viewing frustum and nailboard's frustum: same z-axes and identical point of view.

Figure 6 shows a special case setup of the nailboard frustum with respect to the current viewing frustum in eye space (both texture space and screen space are unit cubes and would coincide). The eye point is at the origin for both frustums and the *z*-axes coincide. This situation occurs for a nailboard directly in front of the viewer (in the center of the final image) when the eye has not moved yet from the point for which the nailboard was generated. Nailboard space and eye space are then the same, only the near and far distances of the two frustums differ. (Typically the nailboard frustum is much smaller than the viewing frustum).

In this case depth values z^t/w^t from texture space are linearly related to z^s/w^s in screen space because they have both been obtained by the same type of rational formula (recall equation (2)):

$$\frac{z^t}{w^t} = \lambda + \frac{\mu}{z}, \qquad \frac{z^s}{w^s} = \lambda' + \frac{\mu'}{z} \qquad \text{and therefore}$$

$$\frac{z^s}{w^s} = \kappa\frac{z^t}{w^t} + \delta \quad \text{with} \quad \kappa = \frac{\mu'}{\mu} \quad \text{and} \quad \delta = \lambda' - \frac{\mu'}{\mu}a \tag{3}$$

where *z* is the point's depth value in the identical nailboard and eye spaces, κ and δ are the parameters of the linear transformation from depth values in texture space to depth values in screen space.

When nailboards are reused over several frames, the position of nailboard space relative to eye space can be more general in two ways: first a translation of either the eye or the object translates nailboard space along the *z*-axis of eye space. As rational functions are involved compensating for this translation by a linear transformation as above introduces an error. Second, when looking around, the direction of view rotates away from the center of the nailboard. In this case the parallel planes defined by discrete depth values in nailboard space become tilted with respect to the image plane.

Figure 7 shows this general setup of nailboard space with respect to eye space. Please note that only a two-dimensional cross-section is shown while the rotation of nailboard space is arbitrary around the eye. A possible small translation of the eye has been neglected in the figure for clarity.

Depth is only a one-dimensional value. Along individual lines of view the depth interval spanned by the nailboard's frustum is scaled and moved along the *z*-axis when

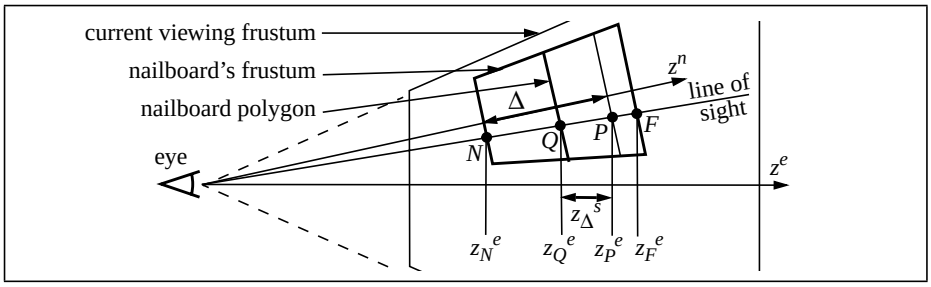

Fig. 7: Relationship between the depth values stored in the RGBΔ texture and the term z_Δ required to obtain correct depth values at *P*.

going from nailboard space to eye space. This linear relationship is exact, as nailboard space and eye space are related by the matrix equation (1).

The relationship between texture space and screen space can only be approximated by a linear relationship because of the perspective divide. For one line of sight let the interval of the nailboard's depth values be delimited by z_N^s and z_F^s in screen space (figure 7 only shows nailboard space and eye space). These are the depth values of the points *N* and *F* on the near and far planes of the nailboard's frustum for the current line of sight. The linear formula

$$z_\Delta^s = (z_F^s - z_N^s)\Delta + (z_N^s - z_Q^s) \tag{4}$$

takes a depth Δ stored in the texture. It moves and scales this Δ along the *z*-axis of screen space to obtain z_Δ^s. z_Δ^s approximates the depth correction for the visible point *P* on the object along the current line of sight. By adding z_Δ^s to the depth value z_Q^s of the point *Q* (the point on the nailboard's polygon) the depth value for this pixel is obtained. It reflects the distance of the visible point *P* on the object from the eye.

By observing that rotations move the interval $[z_N^s; z_F^s]$ more than the maximum allowable translation of the eye which does not trigger a re-generation of the texture it follows that this approximation introduces a relative error which is largest when the nailboard is maximally tilted with respect to the image plane and also directly behind the image plane (where depth values are small and depth buffer accuracy is highest). With a field of view of sixty degrees the maximum tilt is thirty degrees. It translates and scales the interval from $[z_N^s; z_F^s]$ to $[z_N^s \cos 30; z_F^s \cos 30]$.

Figure 8 shows the plots of the relative error occurring throughout this interval immediately behind the image plane (a) and at a moderate distance d=150 (b). The error is calculated as the difference between accurate depth values obtained as described in section 3 and depth values obtained by the approximating equation (4) in percent of the accurate value. The near to far ratio used is 1/100 (n=10, f=1000). The shown relative error is insensitive to the far distance which only changes the magnitude of the involved values but the relative error remains the same. In both cases the extent of the nailboard frustum (z_F^e-z_N^e) was assumed not to exceed the object's distance from the eye. Typically the extent of the nailboard frustum is only a fraction of the eye-object distance because only then image coherence is high for the object.

The further the nailboard is from the viewer, the smaller the relative error, because depth accuracy diminishes rapidly with distance from the viewer (see figure 4). Note how an error of 0.6% is never exceeded even directly behind the image plane.

To obtain (z_F^s-z_N^s) and (z_N^s-z_Q^s) for every pixel covered by the nailboard, the two

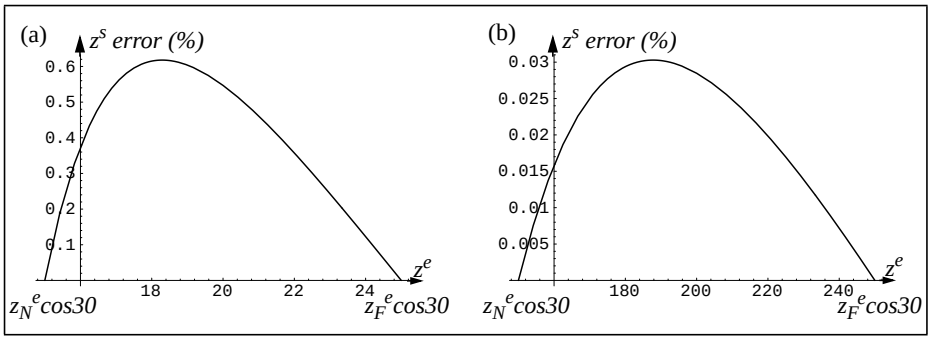

Fig. 8: Error introduced by a linear approximation to the accurate depth correction after rotating the viewing direction by 30 degrees: (a) nailboard directly behind the image plane (n=10, f=1000), (b) nailboard at moderate distance (n=10, f=1000)

values are calculated for the vertices of the nailboard's polygon. (z_N^s-z_Q^s) is immediately added to the depth values of the polygon vertices so that the term is already included in the depth value interpolated across scanlines. The scaling factor (z_F^s-z_N^s) must be added to the interpolated parameter list for the polygon. It gives the depth extent of the nailboard's frustum at every pixel. Both interpolations are exact in screen space.

In summary, both the depth value of the near plane and the depth extent of the nailboard's frustum are interpolated across scanlines. For every pixel the depth Δ is fetched from the texture, scaled by (z_N^s-z_F^s) and added to the depth of the nailboard's near plane. This gives the depth values of the depicted object with a maximum error of 0.6% as shown in figure 8 (a). The closer to the center of the screen and the further away from the viewer the nailboard is, the smaller the error. This behavior is desirable as distant objects exhibit more coherence and are preferably replaced by nailboards. Usually the user is considered to look at the center of the screen (requiring highest accuracy).

5 A Software Implementation

Although a hardware implementation is highly desirable, so far only a software implementation was done on top of the OpenGL graphics library as a proof of concept. OpenGL [10] does support rendering texture-mapped polygons but does not produce correct depth values, if objects are replaced by textured polygons.

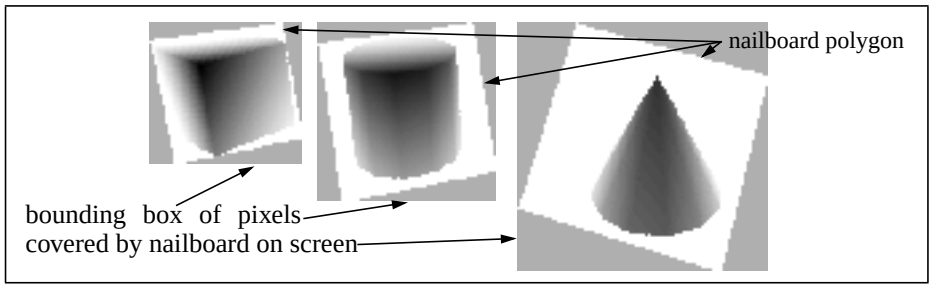

Fig. 9: Re-sampled depth Δ generated by the OpenGL implementation for figure 1.

```
DrawNailboard()
   define a texture with depth Δ in red channel
   draw nailboard polygon with this texture
   copy both depth and Δ values into memory
   correct depth values using equation (4)
   write depth values into depth buffer
   keep result of depth test in stencil plane
   enable stencil test and disable depth test
   draw RGB textured nailboard polygon
```

Fig. 10: Pseudocode for software implementation of nailboards.

Correct depth values are obtained by a method similar to z-composition of multiple RGBz images [2]. First both the depth values of the polygon and the depth Δ at each pixel are needed. Both data is obtained with one rendering pass: the polygon is drawn with a texture containing the Δ values in one of the color channels. Both the depth buffer and this color channel are read into main memory and the correct depth values are calculated for every pixel by applying equation (4).

These depth values are written into the depth buffer with the outcome of the depth comparison recorded in the stencil planes. Now the correct depth values are in the depth buffer. Finally the depth test is disabled and the textured polygon is drawn into all the pixels for which the depth comparison succeeded using the stencil test. This establishes the correct visibility of the nailboard in the final image.

Unfortunately using the stencil buffer and copying pixel from and into the frame buffer does not yield truly interactive frame rates for complex scenes, but the correctness of visibility even for complex scenes is demonstrated. Figure 9 shows the depth Δ values rendered as a texture in the first rendering pass. Figure 10 summarizes the algorithm as pseudocode.

6 A Hardware Implementation

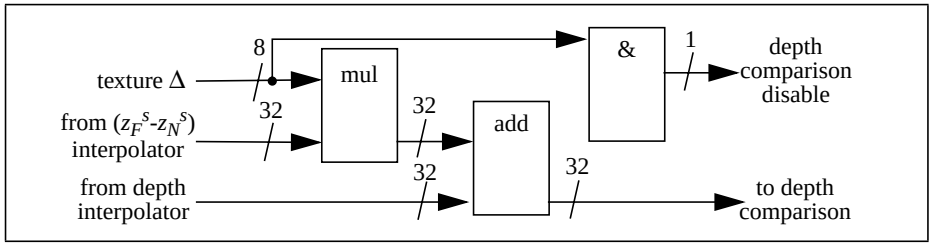

Fig. 11: Components to implement the additional functionality of RGBΔ textures in hardware.

Traditional graphics hardware architectures are easily adapted to support nailboards. (z_F^s-z_N^s) must be added to the interpolated parameter list (which already includes color and/or normal components, depth and texture parameters for textured polygons). Figure 11 shows the additional components required to add the RGBΔ texture functionality to the pixel pipeline. Before the polygon's depth values are fed into the depth comparison the properly scaled depth Δ from the texture is added to them. For the transparent portion of the texture (Δ=0xFF) the depth comparison is disabled so that

the respective pixels are left unchanged:

Even the new hardware architecture called Talisman [17] would benefit from available support of RGBΔ textures. In the description of the architecture the authors state that “independent image layers will generally be used for each non-interpenetrating object in the scene”.

7 Applications

The new graphics primitive improves a number of rendering algorithms: nailboards can replace impostors to avoid the artifacts from interpenetrating impostors shown in figure 1. Objects too close to each other or interpenetrating objects can now be safely replaced by individual nailboards without visibility problems. With impostors close objects must be grouped before being replaced by impostors. Whenever one object in the group needs re-rendering the whole group must be re-rendered. Therefore, coherence is not fully exploited. With nailboards no such restrictions apply.

Nailboards are also advantageous in the context of hierarchical image caching for dynamic scenes. Image caching as described by Schaufler et al. [14] and Shade et al. [16] only works for walk-throughs of static scenes because a spatial hierarchy is built on top of the scene during initialization. Updating such a hierarchy for dynamic scenes is not feasible because of the high computational overhead. One dynamic object destroys all the coherence for the scene parts the object moves through.

Nailboards allow to arbitrarily mix image caching, nailboards and polygonal rendering because they fully integrate into depth buffered rendering. As a result the following advantageous partitioning of a dynamic scene is possible:

For the static scene parts a spatial hierarchy is built and the part’s images are reused over several frames from a hierarchical image cache. Dynamic objects are individually replaced by nailboards or rendered as polygonal geometry depending on the complexity of the objects and the expected time of validity of their images. Without nailboards such a deliberate mixing of rendering methods leads to severe visibility errors. The depth ordering of dynamic objects changes constantly with respect to the textured polygons in the hierarchical image cache. Moreover polygonal geometry makes the flat nature of both the images on impostors and in the image cache apparent whenever they interpenetrate.

Figure 12 (see Appendix) shows a frame of an animation sequence of cars moving over a bridge: note how some of the cars disappear behind the bridge as impostors occlude each other (a). With nailboards these problems go away (b) and the image is indistinguishable from the one generated from original geometry (c). In (d) outlines around nailboards indicate which objects and parts of the static bridge have been replaced by nailboards. The outlines correctly disappear into the depicted geometry (e.g. the ground) which also demonstrates the correct combination of different drawing primitives in one image.

8 Conclusions and Future Work

This paper introduced nailboards for rendering arbitrarily complex objects. By storing every texel’s deviation from the nailboard polygon correct depth values can be calculated and hence visibility is resolved correctly even for touching, interpenetrating and moving objects. The additional computation necessary to arrive at correct depth values is simple enough to be implemented directly in graphics accelerators. For practical scenes eight bits are sufficient to store the depth offsets so that texture memory requirements are not bigger than for RGBα textures used for impostors so far.

In the future investigation of additional trade-offs between the validity duration of generated image data and the effort necessary to extract the final image will be pursued. Re-projecting pixels, view interpolation, rendering trees from pre-computed z-buffer views [9] and RGBΔ textures already demonstrate four such trade-offs but others may be possible.

The technique may be extendible to model objects using several nailboards and interpolate between them to more closely mimic the appearance of the object similar to view-dependent textures [4]. A planned cooperation with a computer graphics hardware manufacturer should result in a hardware implementation of nailboards to fully realize their potential.

References

[1] Akeley, Kurt, *"RealityEngine Graphics"*, Computer Graphics (SIGGRAPH '93) (August 1993) pp 109-116.

[2] Blythe, David and Tom McReynolds, *"Programming with OpenGL: Advanced Rendering"*, SIGGRAPH '96 Course.

[3] Catmull E., *"Computer Graphics Display of curved surfaces"*, IEEE Conference on Computer Graphics, Pattern Recognition and Data Structures, May 1975, reprinted in Tutorial and Selected Readings in Interactive Computer Graphics, H. Freeman (ed.), IEEE, 1980, pp. 309-315.

[4] 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.

[5] Duff, Tom, *"Compositing 3-D Rendered Images"*, Computer Graphics (SIGGRAPH '85) 19 3 (July 1985) pp 155-162.

[6] Foley, J.D., A. van Dam, S.K. Feiner and J.F. Hughes, *"Computer Graphics: Principles and Practice"*, Second Ed. Addison-Wesley, Reading, MA, 1990 pp 274-302, 554-556.

[7] Heckbert, Paul S. and Henry P. Moreton, *"Interpolation for Polygon Texture Mapping and Shading"*, State of the Art in Computer Graphics: Visualization and Modelling, David F. Rogers and Rae A. Earnshaw, eds., Springer Verlag, New York, 1991, pp. 101-111.

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

[9] Max, Nelson and Keiichi Ohsaki, *"Rendering Trees from Precomputed Z-Buffer Views"*, Proceedings of the 6th Eurographics Workshop on Rendering, (June 1995) pp 45-52.

[10] Neider, Jackie, Tom Davis and Mason Woo, *"Open GL Programming Guide"*, Addison-Wesley Publishing Company 1993, Silicon Graphics Inc., ISBN 0-201-63274-8.

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

[12] Schaufler, Gernot, *"Dynamically Generated Impostors"*, MVD'95 Workshop "Modeling - Virtual Worlds - Distributed Graphics" (Nov. 95) D. W. Fellner (ed.) Infix pp 129-136.

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

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

[15] Segal, Mark, Carl Korobkin, Rolf van Widenfelt, Jim Foran and Paul Haeberli, *"Fast Shadows and Lighting Effects Using Texture Mapping"*, Computer Graphics (SIGGRAPH '92), 26 (July 1992) pp 249-252.

[16] 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.

[17] Torborg, Jay and James T. Kajiya, *"Talisman: Commodity Real-time 3D Graphics for the PC"*, Computer Graphics (SIGGRAPH '96), (August 1996) pp 353-363.