Proceedings of the 6th IASTED International Conference
COMPUTER GRAPHICS AND IMAGING
August 13-15, 2003, Honolulu, Hawaii, USA

SILHOUETTE-OPAQUE TRANSPARENCY RENDERING

Osman Sen
Department of Computer Science
University of California, Irvine
osen@ics.uci.edu

Chaitanya Chemudugunta
Department of Computer Science
University of California, Irvine
chandra@ics.uci.edu

M. Gopi
Department of Computer Science
University of California, Irvine
gopi@ics.uci.edu

ABSTRACT
Transparency in 3D graphics has traditionally been created by ordering the transparent objects from back-to-front with respect to the viewpoint, and rendering the opaque objects first and then the transparent objects in the prescribed order. This has three major disadvantages: need for splitting intersecting polygons, repeated ordering for varying view points, and finally, incorrect transparency at the regions near silhouettes. The first two problems are eliminated by order independent transparency rendering techniques. The goal of this paper is to eliminate the third disadvantage also. Silhouettes look more opaque than the rest of the regions of the model. We call this *silhouette-opaque transparency rendering*. We use the alpha value as a probabilistic measure, similar to other order independent methods. We differ from the traditional methods by using this probabilistic measure in object space rather than in image space to render the transparency in silhouettes correctly. We call our technique to achieve *silhouette-opacity* as *object-space screen-door transparency*.

KEY WORDS
Object-space screen-door transparency, silhouette-opacity, transparency rendering.

1 Introduction

Traditional transparency algorithms would separate opaque and transparent objects, and sort transparent objects back to front. The Z-buffer being enabled, the opaque objects are rendered first, and then the transparent objects are rendered back to front. The transparency of a fragment is denoted by its α value. The color of the fragment is linearly composited with the color already existing in the framebuffer.

Crow [3], and Kay and Greenberg [9] in their classic works noted that the transparency is dependent on the amount of material the light passes through. The amount of material depth is invariably more along the silhouettes, and hence silhouettes will look more opaque than other regions of the object. We call this as *silhouette-opacity*. Crow [3] proposed to change the α value towards silhouettes and interpolated the α value non-linearly based on the normal vector at the surface point and the viewing direction. This function is $\alpha = (\alpha_{max} - \alpha_{min})(1 - (1 - N_z)^p) + \alpha_{min}$. The quantities α_{max} and α_{min} are the maximum and minimum transparency of any point on the object. The N_z is the Z component of the unit normal to the surface and p is the cosine power factor. Kay and Greenberg [9] proposed solutions for refraction in the transparent medium by modeling the thickness of the material. This model also incorporated features to take care of *silhouette-opacity*. As far as we know, [3, 9] are the only works in the literature that talk about *silhouette-opacity* and propose solutions.

The advent of latest graphics hardware accelerators have not followed up on these proposals for various reasons. The model proposed by [9] has similar problems as ray-tracing algorithm has in terms of its amenability to hardware implementation. The solution proposed in [3] has the following problems. First, when all other more commonly used attributes like depth value, texture coordinates, and color are interpolated linearly (at least in homogeneous coordinates), interpolating the α value using a non-linear function is a significant overhead. Second, irrespective of the interpolation function, the ordering of triangles (or pixel fragments) from back-to-front has to be computed either by application or during rasterization.

Main Contributions: In this paper, we propose a method for *silhouette-opaque transparent rendering* that eliminates all the above mentioned disadvantages of the methods proposed earlier. First, we achieve order-independent transparency rendering by using a probabilistic method of sampling the surface.

This is similar to the screen-door transparency method but uses random alpha mask patterns in the object space, and super-sampling to generate high quality images. Second, we achieve *silhouette-opacity* by a simple object-space screen-door transparency. Our method is a single pass rendering method that also lends itself for easy hardware implementation.

Outline of the Paper: Next section analyzes the previous work in this area. Section 3 describes our method conceptually. Section 4 describes the implementation details of our algorithm and presents the results. Sections 5 and 6 list the limitations of our apprach and conclude this paper.

2 Previous Work

Rendering objects transparently can open the door for a multitude of graphics and 3D visualization applications. However, rendering of realistic object transparency simulation, including refraction, is a computationally expensive operation. Therefore, non-refractive transparency is used

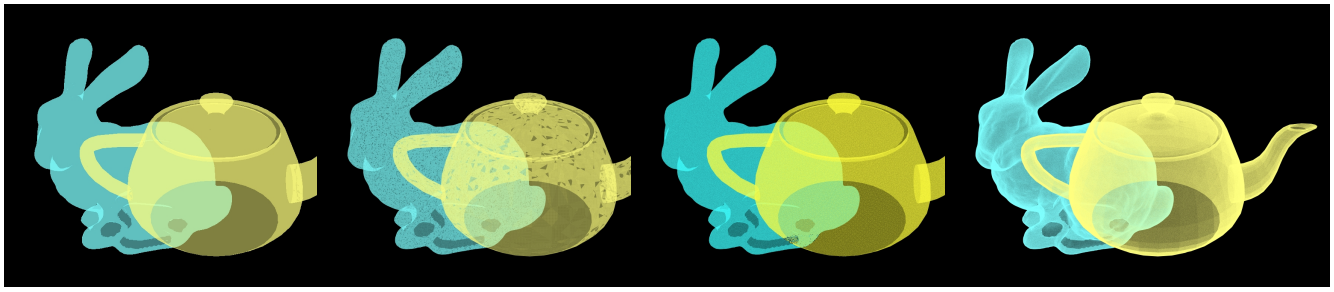

Figure 1. The imagea show the results of using conventional alpha blendin, using stipple buffer [14], our hardware assisted implementation and our software implementation of our object-space screen door transparency algorithm. Notice the silhouettes in our software implementation. The artifacts in the stipple image are due to overlapping polygons getting mapped on to similar stipple maps. Note that this is a filtered image with 16×16 kernel size.

for most of the applications requiring fast rendering. Exact simulation of non-refractive transparent surfaces with alpha blending requires ordering of polygons from back to front and subdividing intersecting polygons [2, 15]. The current day transparency algorithms can be classified as sorting based algorithms and order independent transparency algorithms.

Sorting based algorithms require the primitives to be sorted back to front with respect to the viewpoint. These algorithms can be further classified based on how the sorting is done: application sorting, hardware assisted application sorting, and hardware sorting. Examples of application sorting algorithms include [2, 15, 16, 11, 13, 9] where application takes the complete responsibility of sorting the primitives. Hardware assisted sorting based transparency algorithms include methods like layer depth sorting (depth peeling) using data structures for pixel depth information [5]. Hardware sorting are special purpose architectures for sorting rasterized fragments [18, 1, 13, 8, 12, 4, 10, 17, 11]. Most of these works are derivatives of A-buffer [1]. A few of these hardware-assisted or hardware algorithms use multi-pass rendering methods [5, 12, 4, 17, 11]. These multi-pass algorithms use data structures like linked list of pointers with special hardware and/or render a fixed number of transparent levels.

Order independent transparency algorithms, typically model the α value as a probability measure. Usually, polygons are rendered in the image space that is overlaid with masks. Random masks are used to choose or reject pixels to produce dithering-like effects in the image space [14, 7]. Methods like screen-door transparency using stipple and alpha buffers fall under this category. These techniques have several advantages like they are single pass methods, do not require sorting of primitives, and are able to handle intersecting polygons without further processing. They also suffer from artifacts such as incorrect opacities and distracting patterns due to dithering and masks. Further a few of these methods also have the disadvantage of storing many masks for each transparency value. Supersampling and filtering is a common technique that is used to remove image quality problems. But the correctness of the image generated, especially along the silhouettes cannot be improved by these image-space masking techniques.

The method we present in this paper is an order-independent transparency algorithm. Hence our method is a single pass method and use alpha masks as other methods. Unlike previous techniques, we use these masks in the object space rather than in the image space. This solves the problem of incorrect opacities and also produces silhouette-opaque rendering. We also use supersampling to eliminate the dithering artifacts introduced by masking. Finally, we suggest the use of certain hardware features to accelerate our implementation. In the next section, we formally introduce the underlying concepts behind our method, and our method itself.

3 Object Space Screen-Door Transparency

Let us define the transparency factor α. Consider two vector fields in 3D space. One field contains the outgoing light in all directions from any 3D point P and the other consists of the incoming light from all directions to P. The length of a vector $V_o(P)$ (or $V_i(P)$) at a 3D point P denotes the amount of the outgoing (or incoming) light in the direction of $V_o(P)$ (or $V_i(P)$). The relationship between every $V_i(P)$ and $V_o(P)$ is the radiance transfer equation. But we are going to restrict this relationship between those two rays that are the same. That is, $V_i(P) = V_o(P) = V$.

Given a direction V and a point P, let $\beta_V(P) = |V_o(P)|/|V_i(P)|$, where $V_i(P)$ and $V_o(P)$ are vectors from the two fields in the same direction as V. The transparency factor $\alpha_V(P) = 1 - \beta_V(P)$. (Actually, α should be called the *opacity factor*. We call it the transparency factor to be consistent with the convention.) If P is vacuum, then $\alpha_V(P) = 0$, for any V.

Notice the dependence of α on the direction V, and its independence on the incoming light from directions other than V. In other words, this definition models the anisotropic transparency effects of object points, and ignores the refractive properties. Similar definitions can be arrived at from the above model for refraction, reflection, and other radiance properties of the object.

Silhouette-Opacity: The silhouette-opacity property, where the silhouettes are more opaque than the other parts of the object, can be modeled as an anisotropic property of the object regions. Basically, $|V_o(P)|$ the outgoing vector towards the viewpoint is less if P is near silhouette than if P is in other regions of the object. Since $|V_o(P)| = (1 - \alpha_V)|V_i(P)|$, this attenuation near the silhouettes can be modeled either by reducing the incoming light, or by increasing α_V.

The approach taken by [3] to model silhouette-opacity, increases the value of α_V as a non-linear function of the viewing direction and the normal vector at that point. In our method, we take the alternate approach of reducing the length of the incoming light vector to model silhouette-opacity, and assuming that α_V is same in all directions (that is, isotropic and independent of V).

We model the object using uniform sized primitives and the union of these primitives covers the object. Since the density of primitives in the projection plane is naturally more along the silhouettes, the light reaching the viewpoint from these regions will be attenuated more that the light from other regions of the model. Note that the "primitive" is a generic term. A primitive can be a point (disk) or a triangle or any other geometric feature that is small enough and reliably covers the surface.

Order Independent Transparency: The above process takes care of the silhouette-opacity. We still have to solve the problem of order-independency. In the above sampling of the surface, every primitive is assumed to be of the same size and have the same α value. The amount of outgoing light is $|V_o(P)| = (1 - \alpha)|V_i(P)|$. Consider a set of n primitives in a small region. The amount of light let through by this set of n regions is $(1 - \alpha)\Sigma_{j=1}^{n}|V_i(P_j)|$ corresponding to all primitives in the set. Assuming that the same amount of light is incident on each primitive, due to their spatial proximity, the amount of light let through by the set of n primitives is $|V_o| = n(1 - \alpha)|V_i|$. Let us consider a case in which $|V_o|$ is same as above, but under the condition that every primitive can have either $\alpha = 0$ or 1. Assuming same amount of input light $|V_i|$, the number of primitivies m (out of the above n primitives), that can have $\alpha = 1$ is $m = n\alpha$. If $\alpha = 1$, then that primitive is rendered opaque; otherwise it is not rendered. Rendering $m(< n)$ opaque primitives, instead of n semi-transparent primitives, would *on an average* give the same effect. Choosing m out of n primitives can be achieved using a mask. If these primitives are pixels, and the mask is applied on the image plane, then this method is called *screen-door transparency*[14].

Let us first prove that screen-door transparency produces correct results under restricted conditions. Since binary-alpha transparency rendering is an "averaging" process, it is inherently probabilistic. So the image is super-sampled such that n pixels are averaged to get one pixel in the final image. Assume that there is one transparent object with $\alpha = \alpha_1$ and hence the number of primitives to have $\alpha = 1$ is $m = n\alpha_1$. Let the color of the object be C_1 and that of the background color be C_2. The final color of the pixel after filtering is $C = (mC_1 + (n - m)C_2)/n = \alpha_1 C_1 + (1 - \alpha_1)C_2$, which is the equation for traditional transparency. Assuming that the Z-buffer is enabled, this concept can be proved to be correct for multiple objects also, and does not dependent on the order of rendering of the objects.

Transparency is inherently an object property. Obviously, the above "image-space" argument is correct only if n primitives on the object surface covers n pixels in the image space. But, this is not true for the primitives in the silhouette of the model. Hence, the "image-space screen-door transparency" cannot be used to produce silhouette-opacity.

Silhouette-Opaque Order-Independent Transparency: To achieve order-independent transparent rendering, we choose m out of n primitives from the *object*, rather than from the *image*. We call this method, *object-space screen-door transparency*. To achieve silhouette-opacity, we use *uniform object space sampling*. By combining these two techniques, we achieve *order-independent silhouette-opacity transparency rendering*.

3.1 Decision Parameters

We are now ready to analyze various parameters that would influence the quality of images generated by a method using *object-space screen-door transparency*.

Size and Number of the Point Samples: The number of primitives to be generated depends on α and the surface area. Since these two quantities do not change, the primitives can be generated as a pre-process. All the chosen primitives are opaque ($\alpha = 1$) and the rest of the area is totally transparent($\alpha = 0$). This static sampling has a disadvantage. In the worst case, the viewpoint can be such that one singe (opaque) primitive covers the whole image plane generating incorrect images. Hence the primitives should be sampled during run-time, and the size of the primitives should be dependent on the distance of the triangle(object) from the viewpoint. The optimum size of the primitive is when it covers just one pixel on the screen. The number of generated primitives is dictated by this chosen size of a primitive, the value of α, and the actual area of the polygon(object).

Randomness of Primitive Selection: The choice of primitives on the surface should be uniformly random for our theory to work correctly in practice. We choose primitives in two different ways: by software and by using alpha masks on the triangle in the object space. While choosing primitives by software, we use pseudo-random number generator. In our hardware-assisted implementation, we generate the alpha-mask using a (pseudo-) random pattern of ones and zeros and thus randomize the primitive selection from the object.

Image-space screen-door transparency runs the risk of rejecting all primitives falling on a pixel, if the same image-space mask is used for all objects (with same α value). Such methods avoid this problem by randomly choosing

mask from a set of pre-generated masks for each α value. In our method we have only one mask for each α value. In our method, we do not face this problem as the mask and selection of primitives are in object space rather than in screen space.

Size of Supersampling Kernel: The size supersampling kernel should be large enough to accommodate the required number of pixels generated by all the transparent objects that fall in that region. This size is based on the α values of all these objects and the transparent depth complexity (that is, the depth complexity till the first opaque object from the viewpoint). If we assume an 8-bit representation of α, then we found that $n = 256 = 16 \times 16$ would be more than sufficient for most practical purposes. All results shown in this paper, including the implementation of other algorithms, are generated with this kernel size.

4 Implementation Details

In this section, we describe two algorithms to demonstrate the concept of *object-space screen door transparency*. One is a software implementation, and the other a hardware assisted method. In both these implementations, we will discuss how the vital decision parameters explained in the previous section are taken into account.

4.1 Software Implementation

In this method, we discretize the triangles using point primitives. The number of samples, the choice of samples, and the generation of these samples form the core of our method. In the final step, the high-resultion image we generate is filtered to an image of required size.

4.1.1 Size and Number of Point Primitives

Since we use object-space sampling, the sampling should be independent of the orientation of the triangle. But as we discussed earlier, it should be dependent on the distance of the triangle from the viewpoint. Further, the size of the point should be approximately equal to the pixel size in the super-sampled image.

(Figure 2)Given three vertices of the triangle in the object space, we find the closest distance of the triangle from the viewpoint. We calculate the screen space area occupied by that triangle at that distance, by rotating the triangle to be parallel to the image plane. The number of pixels in this projected area is the number of samples in the triangle. This is calculated as follows:

$$n = \left(\frac{Nh_{im}}{dh}\right)^2 A$$

where N is the distance of the near plane from the viewpoint, d is the distance to the centroid of the triangle from the viewpoint, h_{im} is the height of image in number of pixels, h is the height of image in object space units, and A is the area of the triangle in object space units. Clearly, this method of calculating the number of samples is independent of the orientation of the triangle, and is dependent on its distance from the viewpoint. Hence, this sampling is truly an object-space sampling. We use *uv* parameterization of the triangle to equally distribute these primitives-to-be-generated on the object-space triangle. This achieves uniform object space sampling. Next step is to choose a subset of these uniform samples based on α.

4.1.2 Randomness

The number of chosen primitives is proportional to α of the triangle. For each *uv* parameterized coordinate of a triangle, a psuedo-uniform-random number is generated in the range of the transparency values. This random number is compared with the transparency value of the triangle to decide whether the *uv* point under consideration should be opaque or transparent. This approach is equivalent to that of using a dynamic random alpha mask mapped on the *uv* parameterization of a triangle.

4.1.3 Final Image Generation

As mentioned in Section 3, the generated points are rendered onto the high-resolution image using conventional techniques with Z-buffer being enabled. Then this image is repeatedly filtered down to one-fourth its size using box-filtering. This technique is same as the one used for texture mip-map generation. Results on comparison of our technique with other techniques are shown in Figures 1. Note the silhouette-opacity effects in these images. Further, note the artifacts in Figure 1 for rendering using stipple buffer. Even though this is a filtered image, these artifacts are due to the mapping of many overlapping polygons to the same stipple map.

4.2 Hardware Accelerated Implementation

The basic idea of the algorithm is to take the help of texture-mapping hardware support for the *uv* parameterization. This is achieved as follows. As a preprocess, the transparency values are discretized and for each transparency level an appropriate alpha mask (with binary alpha values for each texel) is generated. Each triangle is mapped on with an alpha mask corresponding to its α value. The transparent texels of the mask will produce the required number of points on the triangle automatically. The rest of the rendering proceeds as in our software implementation (Section 4.1). Note that, unlike the image-space screen-door transparency, generation of a single mask for each transparency level is sufficient for the object space. Further, the size of the mask when compared to the size of the triangle would dictate the size and number of pixels generated on the triangle. Next section elaborates the details of finding this scale to generate appropriate number of samples.

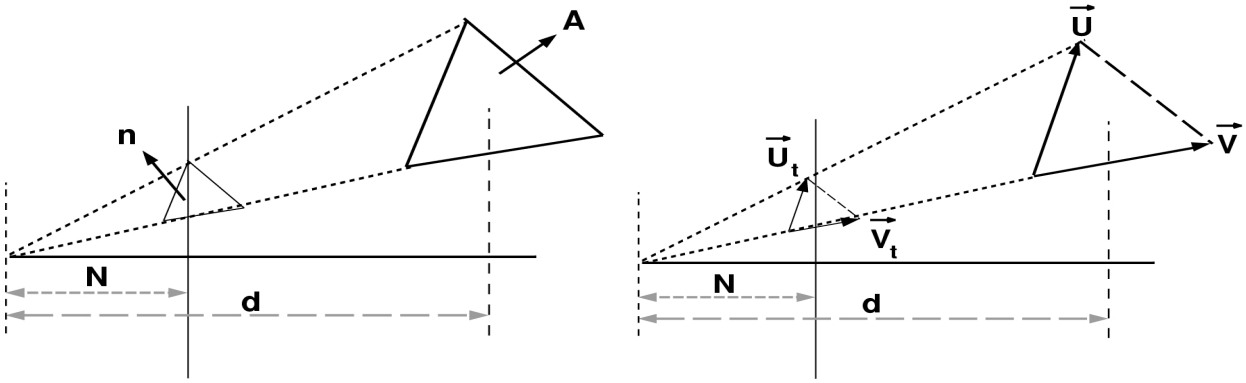

Figure 2. The left image shows diagram for computing the number of samples in a triangle in our software implementation. The right image shows the diagram used in our hardware assisted method.

4.2.1 Size and Number of Pixels

The scale of the texture with respect to the triangle can be controlled by changing the texture coordinates assigned to the triangle vertices. The texture coordinates of the vertices are computed such that each texel of the texture-mapped object-space triangle covers approximately one pixel in the image space when projected.

The computation of the texture coordinates for a mask is done as follows. (Figure 2)In the first step, the edge vectors of the triangle, $\vec{U}$ and $\vec{V}$, are scaled to their projected sizes after bringing the vectors parallel to the image plane, and they are transformed to image space units as,
$\vec{U}_{im} = \left(\frac{Nh_{im}}{dh}\right)\vec{U}$; $\vec{V}_{im} = \left(\frac{Nh_{im}}{dh}\right)\vec{V}$
where $\vec{U_{im}}$ and $\vec{V_{im}}$ are in the image plane and in image space units. The next step is to transform the dimensions to texture space units. As each texel corresponds to a pixel in the image, this transformation is performed simply by rescaling the vectors proportional to size of the mask in number of texels.
$\vec{U}_t = \left(\frac{1}{M}\right)\vec{U_{im}}$; $\vec{V}_t = \left(\frac{1}{M}\right)\vec{V_{im}}$
where M is the size of the mask in terms of number of texels.

Last step is computing the texture coordinates for the triangle points. The relationship between the st parameterization of the texture and the vertices of the triangle is given as follows. Let $k = \vec{U_t} \cdot \vec{V_t}/|\vec{U_t}|$.
$\vec{C_u} = \left(|\vec{U_t}|, 0\right)$; $\vec{C_v} = \left(k, \sqrt{\vec{V_t}^2 - k^2}\right)$
where $\vec{C_u}$ and $\vec{C_v}$ are the texture coordinates of the *vertices* U and V, respectively (refer to Figure 2).

4.2.2 Randomness

The randomness in the opaque texels in the alpha mask provides the randomness in the choice of object-space point samples. The generation of this random alpha mask is similar to that of the software implementation. For each texel in the alpha mask, a random number is produced and compared to the alpha level of the mask to decide whether the texel should be transparent or opaque. As a result the number of opaque texels is proportional to the alpha value and its distribution is (pseudo-)uniform. Example masks are shown in Figure 3.

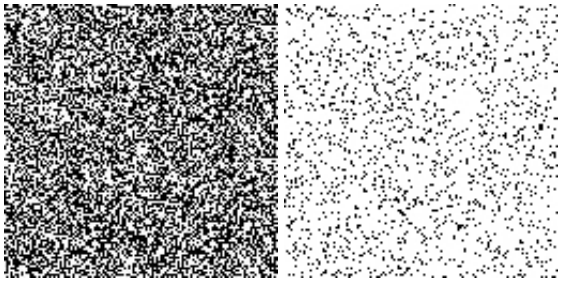

Figure 3. Two random alphs masks for $\alpha = 0.5$ (left) and $\alpha = 0.9$ (right), both of size 128×128.

5 Limitations

In both the algorithms, at the silhouettes, there are many sample-points that correspond to a single pixel in the image. However, in the hardware accelerated implementation, the silhouettes are not visible due to the limitations in the filters available for texture mapping. At present, there are two types of filters provided for texture mapping: *Linear* and *Nearest*. The *Nearest* filter is used in the current algorithm which results in images without silhouettes. However, this can be corrected if we have a *Max* filter that selects the maximum value among the available candidates.

The sampling of the object depends on the distance of the primitive from the viewpoint. This prohibits the use of these algorithms with large models.

6 Conclusion

In this paper we presented a new approach for order-independent transparency. The important problem that our work solves is the *silhouette-opacity*, and we achieve this using *object-space screen-door transparency*. Our method is a probabilitic, single-pass method and the results are significantly better than the previous approaches. We have implemented our method using a software approach for sampling and a hardware assisted approach for sampling points from the surface of the object. We believe that this work will renew the enthusiasm to improve the hardware capabilities for efficient and correct transparency rendering in the future.

References

[1] Loren Carpenter. The a -buffer, an antialiased hidden surface method. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 103–108, 1984.

[2] E. Catmull. A subdivision algorithm for computer display of curved surfaces. In *University of Utah, Salt Lake City*, 1974.

[3] Franklin C. Crow. Shaded computer graphics in the entertainment industry. In *Computer Magazine*, pages 10–22, 1978.

[4] S. J. Baker et al. Image generator for generating perspective views from data defining a model having opaque and translucent features. In *United States Patent Number 5,363,475*, Nov 8, 1994.

[5] Cass Everitt. Interactive order-independent transparency. In *Technical report, NVIDIA Corporation*. ACM Press, 2001.

[6] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics, Principles and Practice, Second Edition. Addison-Wesley, Reading, Masachusetts*. Addison-Wesley, 1990.

[7] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Jr. Frederick P. Brooks, John G. Eyles, and John Poulton. Fast spheres, shadows, textures, transparencies, and imgage enhancements in pixel-planes. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 111–120. ACM Press, 1985.

[8] Norman P. Jouppi and Chun-Fa Chang. Z3: an economical hardware technique for high-quality antialiasing and transparency. In *Proceedings of the 1999 Eurographics/SIGGRAPH workshop on Graphics hardware*, pages 85–93. ACM Press, 1999.

[9] Douglas Scott Kay and Donald Greenberg. Transparency for computer synthesized images. In *Proceedings of the 6th annual conference on Computer graphics and interactive techniques*, pages 158–164, 1979.

[10] Michael Kelley, Kirk Gould, Brent Pease, Stephanie Winner, and Alex Yen. Hardware accelerated rendering of csg and transparency. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 177–184. ACM Press, 1994.

[11] Kevin Kreeger and Arie Kaufman. Mixing translucent polygons with volumes. In *Proceedings of the conference on Visualization '99*, pages 191–198. IEEE Computer Society Press, 1999.

[12] Jin-Aeon Lee and Lee-Sup Kim. Single-pass full-screen hardware accelerated antialiasing. In *Proceedings 2000 SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 67–75. ACM Press, 2000.

[13] A. Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. In *IEEE Computer Graphics and Applications*, pages 9(4):43–55, 1989.

[14] Jurriaan D. Mulder, Frans C. A. Groen, and Jarke J. van Wijk. Pixel masks for screen-door transparency. In *Proceedings of the conference on Visualization '98*, pages 351–358. IEEE Computer Society Press, 1998.

[15] Thomas Porter and Tom Duff. Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 253–259, 1984.

[16] John Snyder and Jed Lengyel. Visibility sorting and compositing without splitting for image layer decompositions. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 219–230. ACM Press, 1998.

[17] Stephanie Winner, Mike Kelley, Brent Pease, Bill Rivard, and Alex Yen. Hardware accelerated rendering of antialiasing using a modified a-buffer algorithm. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 307–316. ACM Press/Addison-Wesley Publishing Co., 1997.

[18] Craig M. Wittenbrink. R-buffer: a pointerless a-buffer hardware architecture. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on on Graphics hardware*, pages 73–80. ACM Press, 2001.