Pixel Masks for Screen-Door Transparency

Conference Paper · January 1998

DOI: 10.1109/VISUAL.1998.745323 · Source: DBLP

3 authors, including:

Frans C. A. Groen
University of Amsterdam
277 PUBLICATIONS **4,520** CITATIONS
SEE PROFILE

Jarke J. van Wijk
Eindhoven University of Technology
83 PUBLICATIONS **4,315** CITATIONS
SEE PROFILE

Pixel Masks for Screen-Door Transparency

Jurriaan D. Mulder * Frans C.A. Groen † Jarke J. van Wijk ‡

Abstract

Rendering objects transparently gives additional insight in complex and overlapping structures. However, traditional techniques for the rendering of transparent objects such as alpha blending are not very well suited for the rendering of multiple transparent objects in dynamic scenes. Screen-door transparency is a technique to render transparent objects in a simple and efficient way: No sorting is required and intersecting polygons can be handled without further preprocessing. With this technique, polygons are rendered through a mask: Only where the mask is present, pixels are set. However, artifacts such as incorrect opacities and distracting patterns can easily occur if the masks are not carefully designed. In this paper, first the requirements on the masks are considered. Next, three algorithms are presented for the generation of pixel masks. One algorithm is designed for the creation of small (e.g. 4×4) masks. The other two algorithms can be used for the creation of larger masks (e.g. 32×32). For each of these algorithms results are presented and discussed.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation - Display Algorithms; I.3.6 [Computer Graphics]: Three-Dimensional Graphics and Realism - Visible Line/Surface Algorithms.

Additional Keywords: Screen-Door Transparency

1 Introduction

For many 3D scientific visualization applications it is essential that objects can be rendered transparently. This particularly holds in the area of volume rendering, where volumetric data is rendered by accumulating color and transparency [11]. But also for other applications transparency is a valuable feature, e.g. for the visualization of multiple iso-surfaces or overlapping and intersecting discrete objects.

Accurate simulation of realistic object transparency including refraction is a difficult and computationally expensive process in computer graphics rendering. Only with ray-tracing more or less realistic transparency rendering can be realized. Dynamic scenes require fast rendering. Therefore, realistic transparency simulation is not feasible in such interfaces. Instead, *nonrefractive transparency* is used, i.e. the refraction of light that passes through an object is not taken into account. Fortunately, the lack of realism does not inherently mean that nonrefractive transparency is less useful. In fact, for most scientific visualization applications it is even to be preferred over realistic transparency as it provides undistorted views of objects occluded by other structures.

In nonrefractive transparency the colors of two objects are combined when one object is in front of the other [7]. The intensity I of a pixel onto which two overlapping objects are projected (where object 1 is in front of object 2) follows from:

$$I = \alpha_1 I_1 + (1 - \alpha_1) I_2 \tag{1}$$

Coefficient α_1 is the opacity factor of object 1. If $\alpha_1 = 1$, object 1 is opaque and if $\alpha_1 = 0$ object 1 is completely transparent.

Most modern high-end raster graphics workstations provide hardware support for nonrefractive transparency. The process is often called *alpha blending* [8]. However, alpha blending has two major disadvantages: The transparent polygons have to rendered in a back to front order and the polygons are not allowed to intersect.

A technique that does not suffer from these drawbacks is so called *screen-door* transparency [2]. Screen-door transparency relies on the capability of the human eye to perform spatial integration on the final scene to simulate nonrefractive transparency. A polygon's opacity is now expressed with a 2D *pixel mask*. This pixel mask determines how many and which pixels of a projected polygon are actually rendered on the screen. The more pixels are rendered the less transparent the polygon will seem and vice versa. The main advantages of screen-door transparency over alpha blending are that it does not require the polygons to be rendered in a sorted order, and it allows the polygons to intersect.

The main disadvantages of screen-door transparency are the possible occurrence of distracting patterns and erroneous transparencies. Both drawbacks are directly related to the pixel masks used. Figure 1 shows the artifacts that can occur. Here, four balls have been rendered with different transparencies using pixel masks of poor quality. The green ball is the front most followed by the yellow, red, and blue balls. The occurrence of distracting patterns and erroneous transparencies is evident in the figures.

Only little attention has been given to screen-door transparency. One of the earlier references to screen-door transparency is in [3]. They describe several algorithms for fast rendering in pixel-planes. Although they do refer to the screen-door method to achieve transparency effects, the concept is only briefly addressed and not fully explored. A more recent example of the application of screen-door transparency is the *Vis-5D* system developed at the University of Wisconsin-Madison Space Science and Engineering Center [4]. The Vis-5D system supports both alpha blending and screen-door transparency for the rendering of iso-surfaces. However, both methods can cause artifacts in their implementation. The alpha blending technique gives artifacts as the polygons to render are not sorted. This results in incorrect transparencies. Their screen-door technique gives the artifacts of distracting patterns and incorrect transparency levels for self or piecewise overlapping iso-surfaces. To a great extent, these artifacts are due to the pixel masks used in the implementation. The problem of the screen-door method for multilayered transparency is also pointed out in [5]. Here it is used as an argument in favor of hardware accelerated z-ordered rendering to be used over single layer z-buffering.

Screen-door transparency can also be implemented on a *subpixel* level. Here, the scene is rendered at a higher resolution than the actual display screen. The final intensity of a screen pixel is computed by averaging the intensities of its subpixels. Hence, the main advantage of subpixel screen-door transparency is that one does not have to worry about the occurrence of distracting visible patterns. Subpixel screen-door functionality is for instance provided by the multisampling hardware in the Silicon Graphics RealityEngine [1].

*Center for Mathematics and Computer Science CWI, P.O. Box 94079, 1090 GB Amsterdam, the Netherlands. E-mail: mullie@cwi.nl

†Faculty of Mathematics, Computer Science, Physics, and Astronomy, University of Amsterdam, the Netherlands.

‡Faculty of Mathematics and Computer Science, Eindhoven University of Technology, the Netherlands

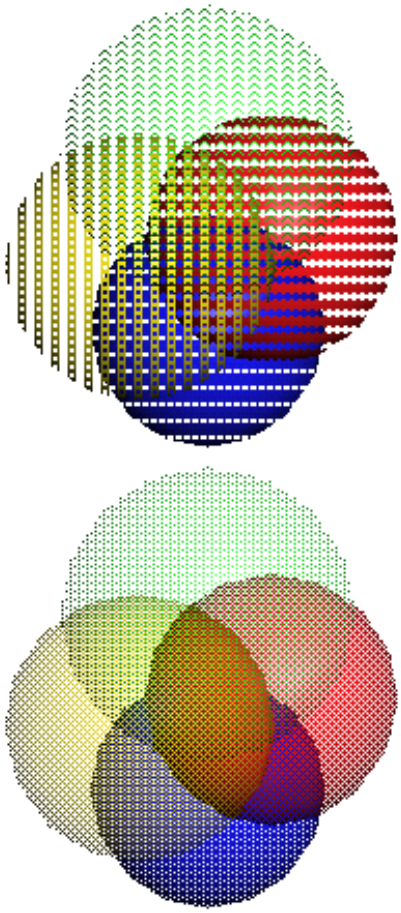

Figure 1: Common artifacts in screen-door transparency. Top: Good transparencies but distracting patterns. Bottom: Less distracting patterns but incorrect opacities.

A disadvantage of this particular realization however, is that the user does not have full control over the screen-door masks used. Furthermore, the final intensity of a pixel is not calculated by averaging all of its subpixels, but by averaging 4, 8, or 16 subpixel samples. One of the software packages that uses the multisampling hardware for screen-door transparency is IRIS Performer.

Despite the disadvantages of screen-door transparency, it can provide a good alternative to other methods as a fast and simple technique to render nonrefractive object transparency. Therefore, we have investigated techniques to create and select pixel masks that reduce forementioned problems. The attention was focused on masks for pixel-based screen-door transparency, but many aspects of these techniques can also be applied for subpixel screen-door transparency masks.

The next section describes the important characteristics of pixel masks used for screen-door transparency. In section 3 three methods developed for the generation and selection of pixel masks are described. Section 4 presents and discusses the results obtained with these methods. Finally, in section 5 some concluding remarks are given and areas for future research are indicated.

2 Pixel Masks

In screen-door transparency, a two dimensional binary pixel mask M filled with 0's and 1's is placed over the projection of a polygon. Only at those pixel locations where the mask has a value of 1 the polygon pixel is drawn. Therefore, the locations in pixel mask M with a value of 1 are called the opaque pixels of the mask, and the locations with a value of 0 the transparent pixels of the mask. The masks we used were confined to a fixed (rectangular) size n, they are statically aligned to the display screen, and repeated periodically along both screen dimensions to cover the entire screen, as this functionality is offered by graphics libraries such as OpenGL [6].

A pixel mask has two important characteristics: The mask's *opacity level* and the mask's *pattern quality*. The opacity characteristic of a mask concerns the perceived opacity of an object when rendered with the mask. The pattern characteristic of a mask concerns the possible occurrence of distracting patterns when the mask is used to render a transparent object.

Besides single layers, the results for multiple layers of masks have to be studied. The *first order* characteristics describe the mask properties when one transparent polygon is rendered over a solid background. The *higher order* characteristics describe the properties when multiple overlapping transparent polygons are rendered.

2.1 First Order Characteristics

Mask Opacity

The *coverage* C of mask M is defined as the ratio between the number of opaque pixels o and the total number n of pixels in the mask:

$$C(M) = \frac{o}{n}$$

When a polygon is drawn with mask M, $C(M)$ determines the perceived opacity of the polygon. The different opacity levels that can be rendered with a mask of size n are:

$$\frac{0}{n}, \frac{1}{n}, \ldots, \frac{n-1}{n}, \frac{n}{n}$$

Thus, the number of different opacity levels that can be rendered with a mask of size n is $n+1$. When a polygon p with opacity α_p is to be drawn on the screen, a mask M should be used such that the difference between the actual and perceived opacity is minimal, i.e. the *opacity error* $|C(M) - \alpha_p|$ should be minimal. As α_p is an arbitrary value in the range $[0, 1]$, the maximum opacity error that can occur is $\frac{1}{2n}$. So, in order to minimize the maximum error, the size of the masks to be used should be as large as possible. There are however two other aspects to consider: The number of pixels covered by the projection of the polygon itself, and the spatial integration capabilities of the human eye.

The relation between opacity levels and mask size also holds for opacity levels and the size of the projected polygon: If the projection of a polygon onto the screen only covers four pixels, the polygon can be drawn only at five opacity levels. So, using larger masks than 2×2 will not reduce the maximum opacity error.

The human eye has limited capabilities to perform the spatial integration needed to perceive polygons rendered with screen-door masks as transparent polygons. A polygon that covers only a few pixels that are at a large distance from each other will not be perceived as one single polygon. Furthermore, there is no need for very small steps between consecutive opacity levels: A human will not be able to perceive any difference in opacity when a polygon is drawn with 999 or 1000 pixels.

Mask Pattern Quality

When rendering a polygon with a screen-door mask, distracting pixel patterns can occur, such as lines, bands, or other pixel clusters. Such patterns are due to low spatial frequencies present in the mask. Although sometimes hard to detect, these patterns are present inside one single mask. As the masks are repeated to cover

the entire display screen they become more evident and more distracting.

2.2 Higher Order Characteristics

Mask Opacities

If two overlapping polygons with opacities α_1 and α_2 are to be drawn over a solid background, they are rendered with masks M_1 and M_2 according to their opacity. If polygon 1 is in front of polygon 2, then polygon 2 can be regarded as being rendered with a mask $M_2 \wedge \neg M_1$. That is, mask M_2 out of which all opaque pixels also present in mask M_1 have been replaced with transparent pixels. For the best result, the coverage of this mask should be as close to the calculated opacities as possible, i.e. the opacity error $|C(M_2 \wedge \neg M_1) - (1-\alpha_1)\alpha_2|$ should be minimal. The worst case in this regard occurs when the two polygons have the same opacity and are rendered with the same mask. The maximum error here is $\frac{1}{4}$, which occurs when both polygons have an opacity of $\frac{1}{2}$ and are rendered with the same mask: No pixels remain of polygon 2.

This can be extended to the case where more overlapping polygons are to be drawn. In general, the mask used for polygon m, located behind polygons 1 through $m-1$ rendered with masks M_1 through M_{m-1}, is:

$$M = M_m \wedge \neg(M_1 \vee \cdots M_{m-1})$$

and the opacity error that occurs for polygon m is

$$|C(M) - (1-\alpha_1)\cdots(1-\alpha_{m-1})\alpha_m|$$

Mask Pattern Qualities

Rendering overlapping polygons with different masks can create distracting patterns, even if the used masks are free of distracting patterns themselves. Such patterns can occur especially when several of the overlapping polygons are of the same color or have the same color as the background.

Suppose two overlapping polygons are rendered with masks M_1 and M_2 where polygon 1 is closer to the viewpoint than polygon 2. If both polygons are of the same color, the overlapping region will look like a single polygon rendered with a mask defined by $M_1 \vee M_2$. If polygon 1 is of the same color as the background, then the overlapping region will look like a polygon that has been rendered with a mask defined by $M_2 \wedge \neg M_1$. Both these 'new' masks can show distracting patterns that are not present in masks M_1 and M_2.

In general, when rendering overlapping polygons 1 through m with masks M_1 through M_m, where polygon 1 is closest to the viewpoint and polygon m is furthest away, the set R of all masks that can show distracting patterns is defined as:

$$\begin{aligned} &R(M_1) = \{M_1\} \\ &R(M_1 M_2 \cdots M_{m-1} M_m) = \\ &\quad \cup\{M_x, M_m \vee M_x, M_m \wedge \neg M_x\}, \\ &\quad \forall M_x \in R(M_1 M_2 \cdots M_{m-1}) \end{aligned}$$

2.3 Mask Size Trade-Off

Summarizing, the quality of a pixel mask is determined by its opacity and pattern characteristics. For single layer transparency, a mask should have no or little opacity error and a good pattern characteristic, i.e. few or no strong low spatial frequency components. For multilayer transparency, combinations of masks determine the quality of the masks. For each mask used it is required that in combination with the other masks used, the resultant opacity error is minimal and no new distracting patterns are created. The size of the masks is an important factor when creating a set of masks. Small masks prohibit the existence of coarse grain patterns but only allow for a small number of opacity levels and only few masks per level to choose from. Large masks allow for many opacity levels and many masks per level to choose from, but have a high risk of showing coarse grain patterns.

3 Mask Generation

We have developed three methods for the generation of pixel masks for screen-door transparency. The first method, presented in section 3.1, is aimed at the creation of small masks, e.g. 4×4. In this method, all possible masks for the desired opacity levels are generated after which the desired number of masks is selected from this set according to the first and second order characteristics. This method is referred to as the Hadamard method as it makes use of a Hadamard transform to evaluate the pattern characteristics of the generated masks. The other two methods aim at the creation of large masks, and are described in sections 3.2 and 3.3. In the first of these two methods, a technique for digital halftoning is adapted to fill the masks according to a *blue noise* pattern. The other method also uses this technique for the distribution of the transparent and opaque pixels over the masks, but in addition a *binary pixel tree* is used to have more control over the higher order transparency characteristics.

3.1 Hadamard Masks

Small masks can be used if only a limited number of opacity levels and masks per level are required. An advantage of using small masks is that the risk of the occurrence of distracting patterns due to low frequency components in the masks is limited, as the lowest possible frequency component is limited to the mask size. Furthermore, when using small masks the total number of possible masks is limited. This allows for the generation of all possible masks and an exhaustive search to select the optimal mask set. The algorithm used is as follows:

1. Generate all possible masks for each desired opacity level;
2. Sort the masks per opacity level according to their first order pattern quality;
3. Select a number of the best masks for each transparency level as to discard the masks with untolerable first order pattern characteristics;
4. From this set, select the final subset of masks to be used according to the second order criteria.

In the last step, the heuristic used for the second order transparency of a mask M_i is simply the difference between the obtained transparency and the analytically calculated transparency for any of the other masks M_j in the final set, i.e. the opacity error $|C(M_j \wedge \neg M_i) - (1-\alpha_i)\alpha_j|$. The heuristic used for the higher order pattern quality of a mask M_i is the pattern quality of the new masks that can be created if mask M_i is combined with mask M_j, i.e. $M_j \vee M_i$ and $M_j \wedge \neg M_i$. To determine a mask's pattern quality, the low frequency components in the mask were considered with the use of an adapted Hadamard transformation function [9] and a weighing matrix to compute a normalized pattern quality.

There are many different configurations of this algorithm for the generation and selection of the pixel masks. Several of the control variables of the algorithm (such as the number of masks selected in steps 3 and 4, the ratio between the second order opacity and pattern quality criteria in step 4, etc.) were parametrized to experiment with different settings.

3.2 Blue Noise Masks

When using large mask sizes the number of possible masks is enormous. As a result, it is no longer feasible to generate all possible masks out of which the final set of masks to be used can be selected. Therefore, a different method for the mask generation is needed. The method presented here is based on a technique that has been developed for *digital halftoning*.

Digital halftoning, often called *spatial dithering*, is the process of creating the illusion of continuous-tone images from the judicious arrangement of binary picture elements [10]. Much research has been done in this area, and good techniques have been developed for the rendering of fixed grey levels. The main difference between screen-door transparency and digital halftoning of fixed gray levels is that for the latter only the first order requirements apply. There is no stacking of masks.

In [10] it is stated that fixed grey levels can best be rendered using *blue noise* on a rectangular grid using a serpentine raster. Blue noise is a form of noise which power spectrum mainly consists of high frequency components, as opposed to for instance *white* or *pink noise*, which power spectra are respectively flat across all frequencies and dominated by low frequency components. In [6] it is suggested to use random mask generation for screen-door transparency, i.e. masks filled with white noise. As will be shown in section 4, large randomly filled masks tend to show distracting coarse patterns, due to the presence of the low frequencies in white noise. We adapted a technique described in [10] called *error diffusion with perturbation* to create large screen-door masks filled with blue noise.

3.3 Pixel Tree Masks

The method described in the previous section was aimed at creating masks with good pattern characteristics. However, no direct control over the higher order opacity characteristics is available. Therefore, a third method was developed that does allow more control over the higher order opacities. This method still makes use of blue noise for the *spatial* distribution of the pixels over the masks, but in addition, a binary pixel tree is used to determine the *logical* distribution of the transparent and opaque pixels over the masks.

A mask can be considered as a set of pixels. Each pixel in this set has two properties: Its value, i.e. either 1 (opaque) or 0 (transparent) and its position in the mask. This method for the generation of pixel masks is based on two steps: First, the transparent and opaque pixels of the masks to be created are distributed logically over the masks to obtain good first and higher order opacity characteristics, i.e the values for the pixels in each mask is deterimined. In the second step, the pixels are distributed spatially over the mask's raster according to a blue noise pattern, i.e. the position of the pixels is determined.

3.3.1 Mask Opacities

Let S_0^0 be the set of pixels upon which the masks M_1 through M_m with opacities α_1 through α_m have to be created. Then, for each mask M_i, it has to be determined which pixels in S_0^0 will have to be transparent, and which pixels have to be opaque. When performing this logical distribution of the transparent and opaque pixels of the masks over set S_0^0, it has to be ensured that the first and higher order opacities of the masks are correct. This is accomplished by the construction of a binary pixel distribution tree. For the first mask M_1 to be created, set S_0^0 is split into two subsets S_0^1 and S_1^1 according to the opacity of the mask α_1: Subset S_0^1 contains the transparent pixels for mask M_1, and subset S_1^1 contains the opaque pixels for mask M_1. For the second mask M_2 to be created, this procedure is repeated but now the subsets obtained for mask M_1 are split: Subset S_0^1 is split into subsets S_0^2 and S_1^2, and subset S_1^1 is split into subsets S_2^2 and S_3^2. Subsets S_0^2 and S_2^2 contain the transparent pixels for mask M_2 and subsets S_1^2 and S_3^2 contain the opaque pixels for mask M_2. This procedure is repeated for each additional mask to be created: For each new mask all subsets of the previously added mask are split to obtain the logical pixel distribution for the new mask. In pseudo code, the algorithm is described as:

```
1.  for (i = 1; i <= nr_mask; i++)
2.      for (j = 0; j < 2^(i-1); j++)
3.          split set S_j^(i-1) into subsets S_2j^i and S_2j+1^i
                    such that |S_2j+1^i| = α_i |S_j^(i-1)|
                    and       |S_2j^i| = |S_j^(i-1)| - |S_2j+1^i|
```

The result of this algorithm can be regarded as a binary tree of pixel sets S_j^i, where each level $i > 0$ in the tree represents the logical pixel distribution of the transparent and opaque pixels of mask M_i, i.e.

$$M_i = \bigcup_{j=0}^{2^i-1} S_j^i$$

where the pixels in the set S_j^i are transparent if j is even, and opaque if j is odd.

It can be proven that for an infinitely large set S_0^0 the obtained pixel distributions are correct for the first and higher order opacity characteristics of all masks in the tree. Practical screen-door implementations however, cannot use infinitely large masks. Because of the finite number of pixels in a set, an opacity error might be introduced when this set is split into two subsets. The less pixels there are in a set, the more severe this error can be. An example is shown in figure 2. Here, a pixel tree was generated for $M_1, M_2,$ and M_3 with opacities of respectively $\frac{1}{2}$, $\frac{1}{2}$, and $\frac{1}{4}$. The pixel set S_0^0 consisted of 8 pixels. The obtained first order opacity for mask M_3 is $\frac{1}{2}$ instead of the desired $\frac{1}{4}$.

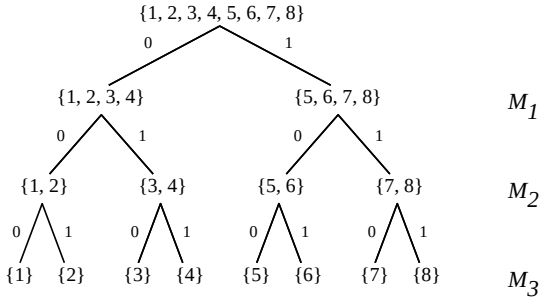

Figure 2: Obtaining incorrect opacity for mask M_3 by independent subset splitting.

A solution to this problem is to split the subsets for a mask sequentially and incorporating the opacity error that has occurred when the previous subsets were split. Step 3 in the tree generation algorithm is then changed to:

```
3.  split set S_j^(i-1) into subsets S_2j^i and S_2j+1^i
            such that |S_2j+1^i| =
                      α_i Σ_(k=0)^j |S_k^(i-1)| - Σ_(l=0)^(j-1) |S_2l+1^i|
            and       |S_2j^i| = |S_j^(i-1)| - |S_2j+1^i|
```

The result of this new algorithm step, applied to the example, is shown in figure 3. The first order opacity error of mask M_3 is now minimized. However, the second order of mask M_3 is incorrect if M_3 is placed under mask M_2. Instead of the desired second

order opacity of $\frac{1}{8}$ (i.e. $\frac{1}{2} \times \frac{1}{4}$), a second order opacity of 0 is obtained. This is caused by the *order* in which the subsets have been split. In figure 3, the subsets have been split in a left to right order; $S_0^2, S_1^2, S_2^2,$ and S_3^2. Had the subsets been split in the order of $S_0^2, S_2^2, S_3^2,$ and S_1^2, then the correct second order opacity would have been obtained, as shown in figure 4.

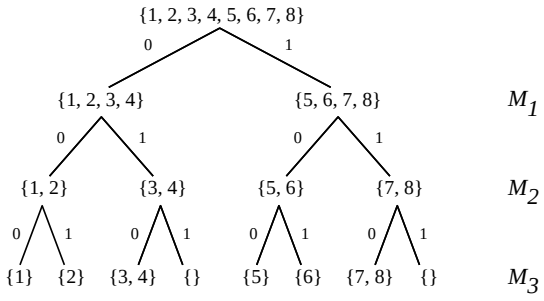

Figure 3: Obtaining correct opacity for mask M_3 by dependent (left to right ordered) subset splitting.

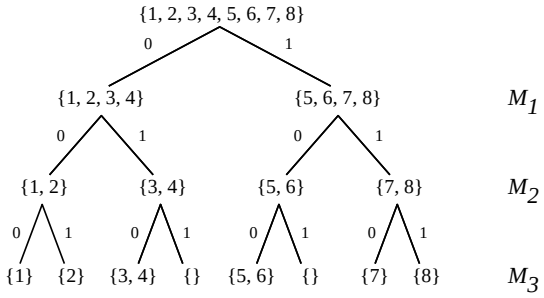

Figure 4: Correct first and higher order opacity for mask M_3.

The problem is in the oscillation of the first order opacity error during the sequential subset splitting. In general, the opacity error tends to oscillate around 0. If this oscillation coincides with an oscillation in the selection of the opaque and transparent pixel sets of any of the previously created masks, higher order opacity errors occur. Therefore, the order in which the subsets are split should be random to avoid such coincident oscillations.

3.3.2 Spatial Pixel Distribution

The result of the first step of the mask generation process as described in the previous section is a pixel tree with the logical distribution of the opaque and transparent pixels of all masks over a pixel set. The next step is to spatially distribute the pixels in the set over a raster to obtain the final layout of the masks. This distribution is achieved with the aid of blue noise generators as described in section 3.2. As a result, the spatial distribution of the pixels over the masks will be according to a blue noise pattern.

For each level in the pixel tree, i.e. for each mask that is to be created, a blue noise generator is started to fill a raster with a blue noise pattern according to the opacity value of the mask of that level. These blue noise generators are now used to select pixels from the leaf sets of the pixel tree to be placed in the raster. The tree is traversed from the root down according to the outcome of the blue noise generators. At each tree level, the outcome of the corresponding blue noise generator determines whether the pixel should be taken from the subset of transparent (0) pixels or from the subset of opaque (1) pixels. When a leaf set is reached, a pixel is picked from that set and placed in the raster. The pixel that is placed is then marked as being used (e.g. removed from the pixel sets so it cannot be used again). The algorithm can be described in pseudo-code as:

```
1.  for (p = 0; p < raster size; p++)
    {
2.      j = 0;
3.      for (i = 1; i <= number of masks; i++)
        {
4.          j *= 2;
5.          j += blue_noise(i, p);
        }
6.      get a pixel number from S^i_j;
7.      put the pixel in the raster at position p;
8.      mark the pixel as used;
    }
```

where `blue_noise(i, p)` is the blue noise generator that returns the value of the pattern for level `i` at raster position `p`.

As an example, suppose eight pixels used to create the pixel tree depicted in figure 4 are to be distributed over a 2×4 raster to create the three opacity masks $M_1, M_2,$ and M_3. For each level in the tree (i.e. for each mask) a generator is started to create blue noise patterns according to the opacities $\frac{1}{2}$, $\frac{1}{2}$, and $\frac{1}{4}$. The output of these generators is used to select the leaf pixel set from which a pixel is to be picked and placed in the raster. Figure 5 shows the first step of this process. The blue noise generators had an output of respectively 1, 1, and 0. This means that the pixel to be placed had to be picked from leaf set S_6^3 depicted in figure 4. There is only one pixel in this set, so pixel 7 is placed at the top left position in the raster (had there been multiple pixels in the set, one of them would have been picked at random). The top left positions of the masks to be created therefore are filled with the value pixel 7 has in the corresponding levels of the tree.

Figure 6 shows the next step. Here, the generators produced 0, 1, and 0. Pixel 3 is therefore placed at the second position in the raster.

Obviously, as this process progresses and the number of pixels still to be placed decreases, it can occur that no pixel can be selected according to the blue noise selection, i.e. the leaf pixel set contains no more unused pixels (or did not contain any pixels from the start). Then, a pixel from another leaf set will have to be selected. In such a case, a pixel from a 'next best' leaf set will have to be placed. This is achieved by 'taking a wrong turn' somewhere in the tree traversal path, i.e. selecting a set of transparent pixels instead of a set of opaque pixels or vice versa, at a particular level. A 'next best set' is a set that can be reached with as few wrong turns as possible.

The mask corresponding to the level where a wrong turn is taken will violate the blue noise generator pattern. Therefore, taking wrong turns when selecting next best leaf sets should be spread over the different levels as much as possible to distribute the blue noise errors over the different masks. In addition, the blue noise generators can use the thus far created masks to generate blue noise (according to the error diffusion with perturbation algorithm), to accommodate to the erroneous mask patterns.

Figure 7 shows the third step in the process of distributing the pixels of the tree in figure 4. The blue noise generators produced outputs of respectively 0, 1, and 1. Therefore, a pixel should be taken from leaf set S_3^3. This set however, does not contain any pixels. Therefore, a wrong turn in the tree path is taken (Left, Left, Right, instead of Left, Right, Right) and pixel 2 is placed in the third position in the raster.

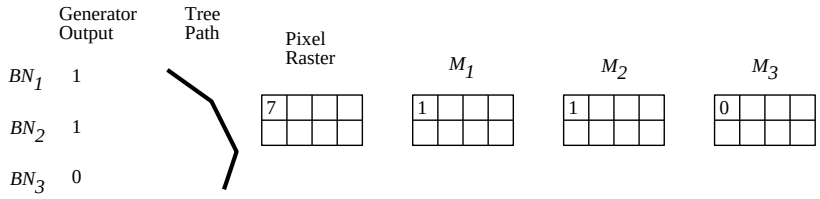

Figure 5: The first step in distributing the pixels over a 2×4 raster. The output of the blue noise generators determined the tree path. Pixel 7 is placed at the top left position in the raster and the masks are filled with the value of pixel 7 in the corresponding levels.

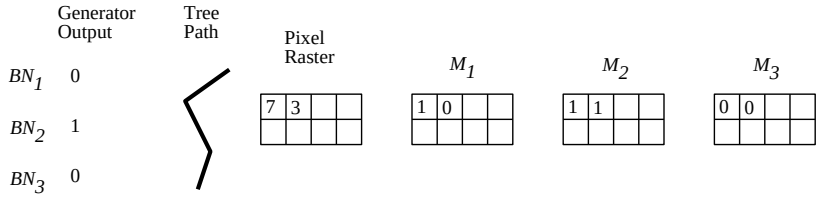

Figure 6: The second step in distributing the pixels. Pixel 3 is placed at the second position in the raster and the masks are filled with the value of pixel 3 in the corresponding levels.

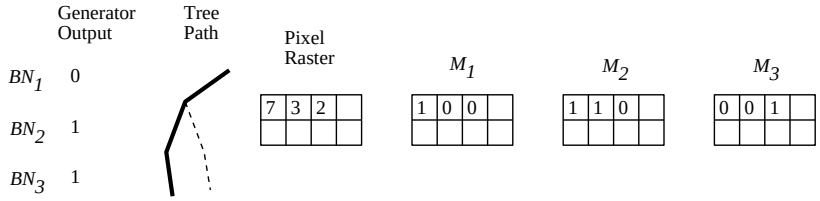

Figure 7: The third step in the pixel distribution process. A wrong turn is taken to obtain a pixel.

4 Results

Figure 8 shows the results of the Hadamard mask generation and selection method for small masks. Here, 4×4 masks were used to create six opacity levels with two masks for each level. The figure depicts a kind of Cartesian product matrix of the masks. Each mask is drawn over all masks in the set. The figure shows that the generated masks are quite acceptable with respect to the first order requirements, i.e. the masks for single layered transparency do not reveal coarse grain patterns. Combinations of masks are less optimal however, and give rise to errors in opacity and some slightly distracting patterns. Furthermore, there is a high diversity among the different combinations of masks, even among the combinations within two opacity levels. These results could not be improved manually by tweaking the algorithm control parameters described in section 3.1, and it is expected that for such small masks simply no better results can be achieved.

Figure 9 shows the results of the blue noise mask generation method. Here, 32×32 sized masks were used to create six opacity levels with two masks per level. The results are much better than the 4×4 sized mask. Both first and second order opacity are good and the occurrence of distracting patterns is reduced. The presence of more frequency components that contribute to the noisiness of the masks has a positive influence on the pattern qualities of the masks, despite the fact that the extra frequency components are low frequency components. Furthermore, there is less diversity in characteristics among the different combinations of masks.

Figure 10 shows the results of white noise mask generation as proposed in [6]. Again, 32×32 sized masks were used to create six opacity levels with two masks per level. Compared to the blue noise masks, this figure clearly shows more coarse grain patterns in both first and second order. The presence of stronger low frequency

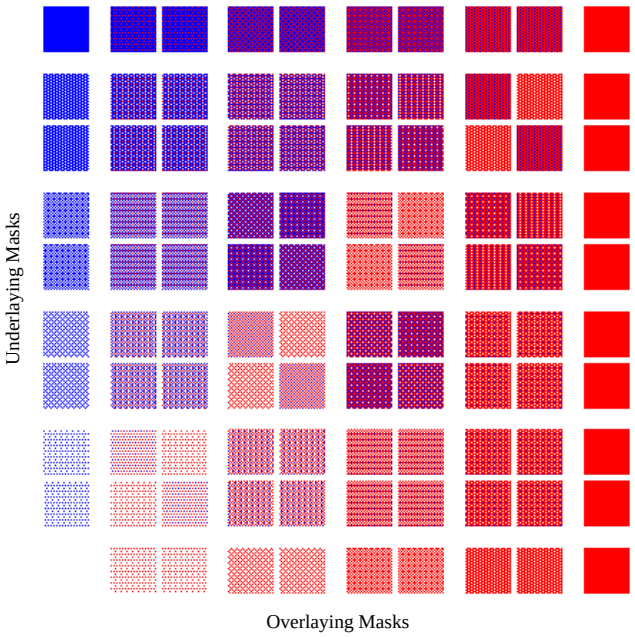

Figure 8: Matrix of the 4×4 masks generated with the Hadamard algorithm. Six incremental opacity levels were created with two masks per level (except for levels 0 and 1).

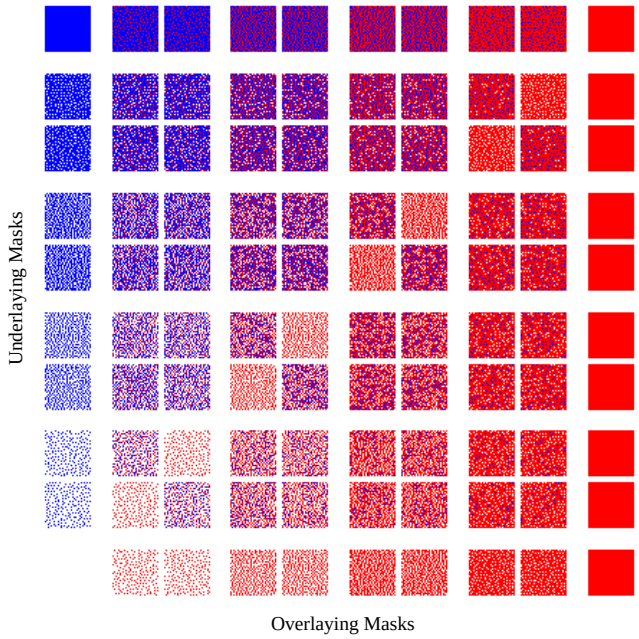

Figure 9: Matrix of the 32×32 masks generated with the blue noise algorithm. Six incremental opacity levels were created with two masks per level (except for levels 0 and 1).

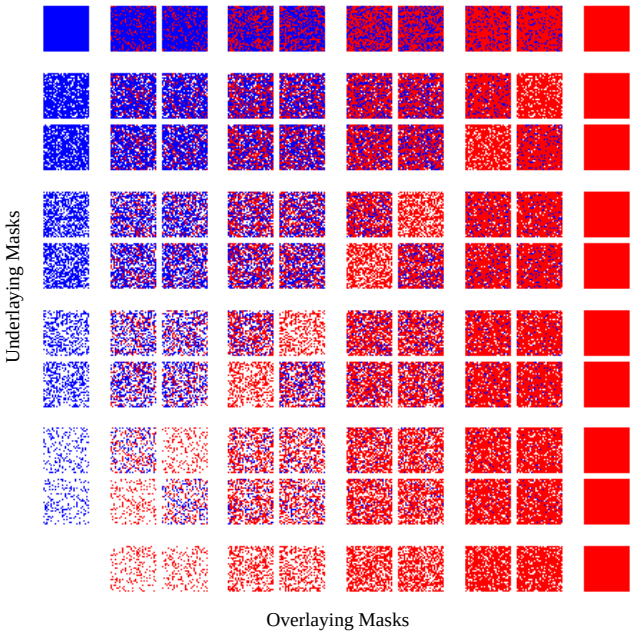

Figure 10: Matrix of the masks generated randomly, i.e. with white noise. Six incremental opacity levels were created with two masks per level (except for levels 0 and 1).

components in white noise compared to blue noise shows itself.

Figure 11 shows the results of the pixel tree mask generation algorithm using binary tree logical pixel distribution and blue noise spatial pixel distribution. Again, 32×32 sized masks were generated for six opacity levels with two masks per level.

The pixels in the masks shown in figure 11 were spatially distributed according to a blue noise pattern. However, due to the restrictions implied by the limited number of pixels available in the leaf sets of the pixel tree, the masks in some places violate the blue noise pattern (i.e. a wrong turn is taken in the spatial distribution process). In this particular case, each mask violates the original blue noise pattern at approximately thirty locations. This however, does not have a significant impact on the masks' pattern characteristics. Comparing the masks in figures 11 and 9 does not reveal any noticeable differences in the masks' visual appearance.

The first order opacities of the masks shown in figures 9, 10, and 11 are correct. For higher order opacities however, the tree generation method produces significantly better results than the other two. Figure 12 shows an analysis of the higher order opacity characteristics. Each mask was placed underneath one to up to seven of the other masks in the set, obtaining the second through eighth order opacities. The resultant opacity of the mask is then compared to the analytically determined opacity, i.e. the number of pixels of the mask that remained visible were compared to the number of pixels that should have been visible. Figure 12 shows the average, minimum, and maximum difference in the obtained and desired number of pixels for all the masks in the three mask sets. Clearly, the method based on the binary tree for the logical pixel distribution performs much better than the two methods that use blue and white noise for the logical pixel distribution.

Figure 13 depicts the same scene as was shown in figure 1, only now rendered with the mask sets of figure 11. The results are much better than that of figure 1; the pattern artifacts have disappeared and the obtained opacities are correct.

It should be noted that the mask generation algorithm using the binary tree for the logical pixel distribution is computationally rather expensive. The complexity of the algorithm is of order 2^m, where m is the number of masks that have to be created. This implies that the algorithm is not very well suited for run time, per frame mask generation if many masks are required. If the masks however, can be generated in advance, the algorithm produces good quality masks that can be used for the rendering of transparent objects without the need for sorting or additional tessellation of the object's polygons.

5 Conclusion

Transparency is a useful feature in scientific visualization. Screen-door transparency is a fast and simple method to render transparent polygons. The two major drawbacks of this technique are the occurrence of distracting patterns and inaccuracies in multilayered transparency. Although depth sorted techniques like alpha blending produce better results, careful generation and selection of screen-door masks tailored to the application to reduce these drawbacks make the screen-door technique a good alternative if high-speed rendering is required.

It was shown that stochastic generation of large masks with the use of binary partitioning for the logical distribution and an error diffusion algorithm with perturbation for the spatial distribution of the pixels gives good results for both few and many opacity levels. The blue noise characteristic of the spatial distribution of the pixels over the masks ensures good pattern characteristics, while binary partitioning with the use of a pixel tree for the logical distribution of the pixels over the masks ensures optimal higher order opacity characteristics.

Much research can still be done in the area of mask generation and selection. For instance, an efficient technique for on-the-fly incremental mask generation would allow for the creation of masks according to the per frame changing requirements of dynamic scenes.

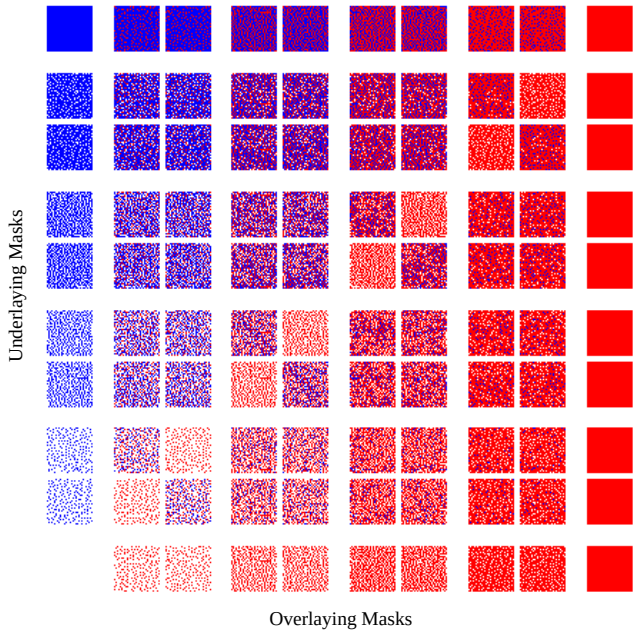

Figure 11: Matrix of the 32×32 masks generated with the pixel tree algorithm. Six incremental opacity levels were created with two masks per level (except for levels 0 and 1).

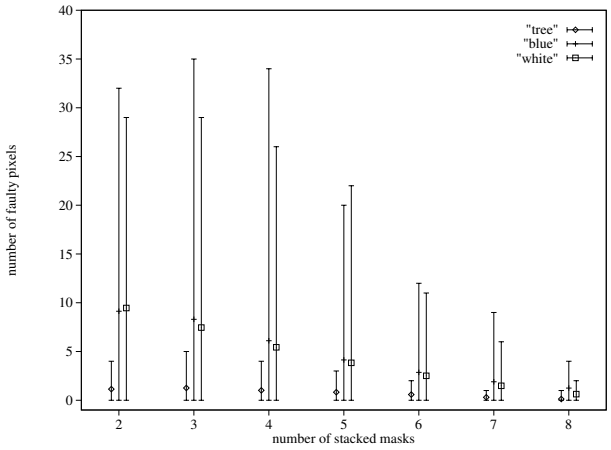

Figure 12: Higher order opacity analysis. For each algorithm, the average, minimum, and maximum number of faulty pixels are indicated when two to eight masks are stacked.

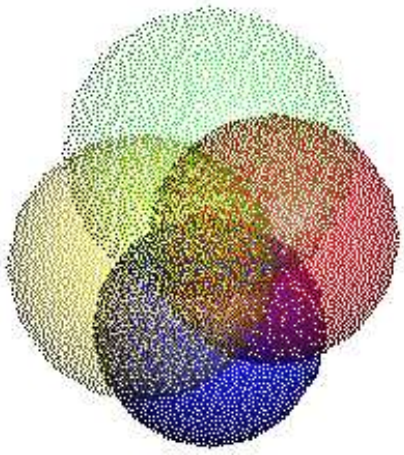

Figure 13: The scene of figure 1 now rendered with the masks shown in figure 11.

References

[1] K. Akeley. RealityEngine graphics. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 109–116, 1993.

[2] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, second edition, 1990.

[3] H. Fuchs, J. Goldfeather, J.P. Hultquist, S. Spach, J.D. Austin, F.P. Brooks, Jr., J.G. Eyles, and J. Poulton. Fast spheres, shadows, textures, transparencies, and image enhancements in Pixel-Planes. In B.A. Barsky, editor, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 111–120, 1985.

[4] B. Hibbard and D. Santek. The VIS-5D system for easy interactive visualization. In A. Kaufman, editor, *Visualization '90 (Proceedings of the IEEE 1990 Visualization Conference)*, pages 28–35, 1990.

[5] M. Kelley, K. Gould, B. Pease, S. Winner, and A. Yen. Hardware accelerated rendering of CSG and transparency. In *Computer Graphics (SIGGRAPH '94 Proceedings)*, volume 28, pages 177–184, 1994.

[6] J. Neider, T. Davis, and M. Woo. *OpenGL Programming Guide: The Official Guide to Learning OpenGL*. Addison-Wesley, Reading, Mass., first edition, 1993.

[7] M.E. Newell, R.G. Newell, and T.L. Sancha. A solution to the hidden surface problem. In *Proceedings of the ACM National Conference*, pages 443–450, 1972.

[8] T. Porter and T. Duff. Compositing digital images. In H. Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 253–259, 1984.

[9] W.K. Pratt. *Digital Image Processing*. Wiley, New York, 1978.

[10] R. Ulichney. *Digital Halftoning*. MIT Press, 1987.

[11] C. Upson. Volumetric visualization techniques. In D.F. Rogers and R.A. Earnshaw, editors, *State of the Art in Computer Graphics: Visualization and Modelling*, pages 313–358. Springer Verlag, 1991.