Technical Report no. 2006-10

Image-Space Dynamic Transparency for Improved Object Discovery in 3D Environments

Ulf Assarsson *Niklas Elmqvist* *Philippas Tsigas*

CHALMERS | GÖTEBORG UNIVERSITY

Department of Computer Science & Engineering
Chalmers University of Technology and Göteborg University
412 96 Göteborg, Sweden

Göteborg, 2006

Abstract

We present an image-space algorithm for dynamic transparency with the purpose of supporting object discovery and access in information-rich 3D visualization environments. The algorithm is based on multiple rendering passes and detects instances of object occlusion in the image-space using the fragment shader capabilities of modern programmable graphics hardware, creating *alpha maps* of opacity gradients around the occluded objects. In essence, the effect is somewhat similar to "X-ray vision" of a superhero. We have implemented a prototype version of our algorithm with real-time rendering performance using a number of optimizations and speedups on current graphics hardware. To evaluate its use, we have also implemented three different example applications portraying different scenarios, including abstract visualization, virtual walkthrough, and gaming. Preliminary results from an empirical user study comparing our technique to standard viewpoint controls indicate that our technique is superior in regards to object discovery efficiency. These results hold over both completion times as well as correctness of a visual search task.

Keywords: dynamic transparency, occlusion reduction, alpha map, image-space techniques, fragment shaders

1 Introduction

The ability to utilize the full 3D space as a canvas for information-rich visualization applications is a mixed blessing–while 3D space on the one hand supports an order of magnitude of more layout opportunities for visual elements than 2D space, visualization designers are on the other hand faced with a number of new challenges arising from the nature of 3D space which do not occur in 2D. More specifically, designers must consider the *visibility* of objects when users wish to discover relevant objects, as well as their *legibility* when the user wants to access information encoded in a particular object. For instance, whereas objects that do not intersect can never occlude each other in 2D space, this can very well happen in 3D space depending on the viewpoint and the spatial relation between the objects. In the real world, this phenomenon is especially evident for a guest entering a crowded cocktail party–the throng of people occludes the individual guests, forcing the newcomer to potentially walk around the whole room to discover his acquintances and as well as to attract their attention. Even if special care is taken in visualization techniques and presentations to avoid these situations, this is very difficult to achieve properly, and is unavoidable in the general case.

Figure 1: Simple 3D scene showing dynamic transparency alpha maps uncovering an F-15 fighter hidden inside a building.

In this paper we describe an algorithm for image-based dynamic transparency designed to help reduce the impact of inter-object occlusion in 3D visualizations, primarily by improving the visibility of important objects to promote discovery, but also by improving legibility to facilitate information access of individual objects. The algorithm makes use of fragment shaders for the new generation of programmable graphics hardware to perform occlusion detection in the image space and automatically create *alpha maps* of opacity gradients to allow the user to see through intervening surfaces and uncover important objects, somewhat akin to the "X-ray vision" of a superhero. Alpha maps are constructed using multiple rendering passes in texture buffers by interpolating alpha buffer values to achieve the desired result. The alpha maps are then blended with the framebuffer, acting as a cumulative alpha map. Objects can be flagged as being *targets* or not; targets are automatically uncovered through intervening surfaces, whereas unimportant objects (distractors) are not. Furthermore, surfaces can also be made *impenetrable* so that they cannot be seen through. Finally, we distinguish between *active* and *passive* dynamic transparency, the former providing a user-controlled "flashlight" of attention for active search, whereas the latter simply detects all instances of occluded important objects and makes them visible.

We have implemented a version of the dynamic transparency algorithm in a C++ application using multiple rendering passes on off-screen texture buffers. We also present a number of speedups and optimizations to achieve a good implementation of the technique. Our test results indicate real-time rendering performance on the current generation of programmable graphics hardware on commodity PCs.

In order to evaluate the usefulness of our technique, we have implemented three different application examples depicting common scenarios within our problem domain: an abstract 3D environment of simple geometric primitives similar to information visualiza-

tion applications, a 3D virtual walkthrough application for a complex building environment, and a game-like scenario demonstrating the use of dynamic transparency in a 3D strategy game. Performance measurements on these different applications show interactive framerates on standard computer hardware. We are currently performing a formal user study comparing the time and correctness performance of human subjects using our technique as opposed to using standard viewpoint control. Preliminary results from this study indicates that dynamic transparency technique allows for significantly improved object discovery efficiency compared to standard methods, both in terms of completion times as well as correctness. In general, dynamic transparency technique seems to be superior for understanding information-rich 3D visualizations, although realism, visual quality and some rendering performance must be sacrificed for this.

The contributions of this paper are the following: (i) a consistent model for dynamic transparency that captures a natural way of achieving high object discovery efficiency; (ii) an efficient image-space algorithm for dynamic transparency using the new generation of programmable graphics hardware; and (iii) implementations of three typical scenarios where dynamic transparency can be useful.

This paper is organized as follows: We first discuss the related work in the field, including general occlusion reduction techniques, as well as papers related to employing transparency for object discovery in particular. We then present our theoretical model for dynamic transparency and the occlusion problem in Section 3. In Section 4, we give our algorithm that realizes the requirements put down in the previous section, and in Section 7 we present our results on implementing the algorithm and the performance results. We discuss these results in section 8. We end the paper with conclusions and a few words on future work.

2 Related Work

2.1 Navigation and Orientation

No paper on improving the perception of 3D environments is complete without references to the substantial body of work available on 3D navigation and orientation. However, 3D navigation addresses a higher-level issue which is caused by a number of different factors related to 3D environments, occlusion being just one. Nevertheless, relevant work here includes [6, 13, 22] and many more. Bowman *et al.* define the concept of *information-rich virtual environments* (IRVEs) [3], a combination of information visualizations within the framework of virtual environments, yet many of the challenges [25] presented in their work are directly applicable to any kind of 3D visualization application. The theoretical model for our technique is partly inspired by this work.

2.2 Non-Photorealistic Rendering

The approach presented in this paper can be applied to both rendering of abstract 3D information visualizations as well as general 3D models, but is an inherently non-photorealistic rendering (NPR) technique. The emphasis lies on conveying important structural or semantic information about a 3D scene or visualization, not necessarily a high-quality visual appearance.

NPR has in recent years become a popular research area in computer graphics; examples include painterly rendering [18], hatching [26], and edge and silouette extraction [19]. Although typically not employed directly for visualization, the approach has found a use in computer-generated technical illustrations. Gooch *et al.* [14, 15] describe the use of NPR-based silouette extraction and tone shading techniques for automatic and interactive technical illustrations. Nienhaus and Döllner [24] presents the blueprints method, which

employs edge extraction and depth layering to outline and enhance both visible and occluded features of 3D models. Freudenberg *et al.* [12] introduce another tone-based NPR primitive that may be useful in this context. Our method can similarly be employed for technical illustration, especially with the layer control mechanism discussed in Section 3.5, but this is not the primary purpose of our work.

2.3 Transparency

The algorithm presented in this work makes heavy use of semi-transparent surfaces to reduce the impact of occlusion in an information-rich 3D environment as well as to completely avoid the loss of 3D depth cues. The general linear model for transparency in computer graphics was introduced by Kay and Greenberg [21]. In order to achieve correct results, transparent surfaces must be rendered in depth order. Everitt [11] discusses the *depth peeling* image-space algorithm for achieving this on modern graphics hardware based on the virtual pixel map concepts introduced by Mammen [23] and the dual depth buffers by Diefenbach [7]. The blueprints [24] technique mentioned earlier uses depth peeling to outline perceptually important geometrical features of complex models using transparency and edge detection. Diepstraten *et al.* introduce view-dependent transparency [8] where NPR transparency techniques are employed for interactive technical illustrations. While closely related to our work in regards to the general method, Diepstraten employs a fixed two-pass depth peeling step to uncover the two foremost layers of transparent surfaces, whereas our method is based on iterative back-to-front rendering and blending using alpha maps, and is thus not limited to a specific depth. Furthermore, where the purpose of our work is to employ dynamic transparency for improved object discovery, view-dependent transparency is primarily aimed at computer-generated technical illustrations.

The use of alpha blending for exposing hidden content in windowing systems is well-known (e.g. [17]) but may result in loss of depth cues and legibility. Gutwin *et al.* [16] explore a dynamically adapting transparency mechanism based on the distance to the mouse cursor to avoid this. Multiblending [1] is a more advanced blending approach where many different image processing techniques are applied separately to different classes of graphical components. Ishak and Feiner [20] takes this a step further by introducing a content-aware transparency mechanism that dynamically adapts opacity depending on the importance of various parts of a window. Just like in our work, they employ smooth gradients to emphasize the continuity of the transparent objects. In addition, their system supports a magic lens-like [2] focus filter, similar to the active transparency searchlight in our work (although our version is a three-dimensional magic lens [27]).

Semi-transparency is also commonly used in 3D games and virtual environments to allow users to see through occluding surfaces; Chittaro and Scagnetto [5] investigate the merits of this practice and conclude that see-through surfaces seem to be more efficient than normal 3D navigation, although not as efficient as bird-eye views.

2.4 Cut-Away and Break-Away Views

One popular technique for traditional paper-based technical illustrations is called *cut-away* views, where parts of the depicted object is cut away to reveal interior objects that would otherwise be hidden from view. Diepstraten *et al.* present their work on computer-based cut-away illustrations [9], where a small set of rules are presented to generate an effective model for interactive technical visualization. Cut-away views are not view-dependent, however, and thus bear only superficial resemblance to our work.

In the same paper, however, the authors also present *break-away* views, where interior objects are made visible through the surface of containing objects through image-space holes. While similar to our work in method, Diepstraten's technique is simplified by semantic knowledge of inside and exterior objects, and the fact that the break-away view is

realized by a single hole. To this end, their approach is to compute the convex hull of interior objects and using it as a clipping volume when rendering. Our approach, on the other hand, derives spatial information through sorting and rendering the scene back-to-front, smoothly blending the gradient outline of targets to the scene buffer in an iterative fashion.

2.5 Importance-Driven Rendering

A generalization of cut-away views, importance-driven rendering assigns importance values to individual objects in a 3D scene and renders a final image that is a composite of not only the geometrical properties of the objects, but also their relative importance. This can be used to achieve various effects for expressing spatial and semantic information about the scene; Viola *et al.* employ it for importance-driven volume rendering [28] (IDVR) to actively reduce inter-object occlusion in the same way that we do in this work. While the IDVR technique described in the aforementioned paper uses a more general importance scale than the target vs. distractor dichotomy in our model, Viola's implementation (besides being aimed at volume rendering applications) does not provide interactive framerates, whereas our implementation makes use of modern graphics hardware to deliver real-time performance.

3 Model for Dynamic Transparency

In this section we present our model for the occlusion problem in 3D environments and describe the dynamic transparency approach to reducing it (see [10] for an extended description of this model).

3.1 Model

We represent the 3D world U by a Cartesian space $(x,y,z) \in \mathbb{R}^3$. Objects in the set O are volumes within U (i.e. subsets of U) represented by boundary surfaces (typically triangles). The user's viewpoint $v = (M,P)$ is represented by the view and projection matrices M and P.

A line segment r is *blocked* by an object o if it intersects any part of o. An object o is said to be *occluded* from a viewpoint v if there exists no line segment r between v and o such that r is not blocked. Analogously, an object o is said to be *visible* from a viewpoint v if there exists a line segment r between v and o such that r is not blocked. An object o is said to be *partially occluded* from viewpoint v if o is visible, but there exists a line segment r between v and o such that r is blocked.

An object can either be flagged as a *target*, an information-carrying entity, or a *distractor*, an object with no intrinsic information value. Importance flags can be dynamically changed. Occluded distractors pose no threat to any analysis tasks performed in the environment, whereas partially or fully occluded targets do, resulting in potentially decreased performance and correctness.

The surfaces defining an object volume have a transparency (alpha) function $\alpha(x) \in [0,1]$. A line segment r passing through a surface at point p is *not* blocked if $\alpha(p) < 1$ and the cumulative transparency value α_r of the line segment is less than 1. Passing through a surface increases the cumulative transparency of the line segment accordingly (multiplicatively or additively, depending on the transparency model).

3.2 Visual Tasks

The occlusion problem typically occurs in the following two *visual tasks*:

- *object discovery* – finding all objects $o \in O$ in the environment; and

- *object access* – retrieving graphically encoded information associated with each object.

More concretely, occlusion affects both the visibility and legibility of visual objects in an environment. Fully occluded objects are not visible and cannot be discovered (without manipulating the view or the environment), and consequently are illegible and cannot be accessed. Partially occluded objects are visible and can be discovered (although discovery efficiency is degraded), but their legibility may be low and thus access is difficult.

3.3 Dynamic Transparency

The general idea behind dynamic transparency is simple: we can reduce the impact of occlusion by dynamically changing the transparency (alpha) value of individual object surfaces occluding (either partially or fully) a target object. This results in fewer fully occluded objects in the environment and thus directly impacts the object discovery visual task.

The fact that the dynamic transparency mechanism operates on the transparency level of individual points of surfaces and not whole objects or even whole surfaces is vital; if whole surfaces or objects had been affected, important depth cues would have been lost. With the current approach, unoccluding parts of a surface will retain full opacity, providing important context to the transparent parts of the object. To give additional context, even occluding surface parts are not made fully transparent, but are set to a threshold alpha value α_T in order to still shine through slightly in the final image. There is a tradeoff here: the use of semi-transparent occluders will make object access difficult since intervening surfaces will distort targets behind them. However, it is a necessity in order to maintain the user's context of the environment.

We define our model for dynamic transparency through a number of discrete rules governing the appearance of objects in the world:

(R1) All targets in the world U should be visible from any given viewpoint v.

The first rule is the most basic description of dynamic transparency, and stipulates that no targets should be fully occluded from any viewpoint in the world. Note that a target may still be hidden from the user if it falls outside the current view.

(R2) An occluded object is made visible by changing the transparency level of points $p \in P$ of each occluding surface s from opaque ($\alpha_s(p) = 1$) to transparent ($\alpha_s(p) = \alpha_T$).

The second rule describes the actual mechanics of how to make targets visible through occluding objects. The selection of the set P is not fixed; depending on the application, this could either be a convex hull, circle, or ellipse that encloses the occluded object, or the occluded object's actual outline.

(R3) Surfaces can be made *impenetrable* and will never be made transparent.

The third rule provides a useful exception to the initial rule; in some cases, we may want to limit the extent of the dynamic transparency mechanism through the use of impenetrable surfaces (and objects).

(R4) Objects are allowed to self-occlude themselves.

The fourth and final rule provides another refinement of the previous rules; dynamic transparency is performed on object-level, even if transparency management is performed on individual surface points. This means that even if a part of a target is occluded by other parts of itself, none of its surfaces will be made transparent to show this.

3.4 Operation Modes

In addition to the basic operation outlined above, dynamic transparency can be used in either an active or a passive mode. *Passive* mode is the standard dynamic transparency performed on the whole view visible to the user; all occluded objects are revealed automatically without the user having to do anything. This may cause quite a severe impact on the visual quality of the scene, however, and make it difficult for the user to gain an understanding of the layout of the scene.

In *active* mode, on the other hand, the user controls a searchlight (a 2D circle, typically) on the image plane of the scene specifying on which parts of the world dynamic transparency should be active. This is a less obtrusive mode of operation than passive mode and has less impact on the visual quality of the scene, but on the other hand requires direct manipulation by the user.

3.5 Layer Control

The standard dynamic transparency mechanism, as described above, will peel away all intervening surface layers to reveal occluded targets in a scene. However, in some cases, we may want to control the maximum number of layers to be peeled away by the mechanism. By introducing this capability to the specification of dynamic transparency, we allow for special classes of visualizations, such as the one-layer depth technical illustrations discussed in Diepstraten *et al.*[8, 9].

4 Image-Space Dynamic Transparency

An important observation that follows from our model of occlusion from the previous section is that occlusion can be detected in the image space by simply shooting a ray through the scene for every pixel that is rendered and checking the order it intersects objects in the scene. In modern graphics hardware, this essentially amounts to detecting whenever we are overwriting pixels in the color buffer or discarding pixels due to depth testing. In other words, programmable fragment shaders are perfectly suited for realizing dynamic transparency.

However, correct blending of transparency is order-dependent, and thus our algorithm, as well as most algorithms for transparent objects, needs the objects to be rendered in back-to-front order. This is a classical problem, since current graphics hardware cannot do the sorting for us, although suggestions for solutions exist [4]. Usually, depth sorting is performed on triangle-level. In our algorithm, for non-intersecting objects, it is sufficient to sort on object-level for normal objects that are opaque by default. For intersecting objects, sorting must be performed on a per-triangle-level. Intersecting objects are however rare and usually non-physical. As explained below, objects fully contained within other objects, like objects in a suitcase or nested Russian dolls, can be correctly treated by specifying a fixed sort order between a group of objects.

We divide the scene into groups. By default, a group contains one object. All groups are sorted with respect to their center point, which is precomputed once. The sorting metric is the signed distance to the group from the eye along the look vector. This is better than sorting by only the distance from the eye, because the former corresponds to how the z-buffer works. We use bubble sort, since frame coherency brings the resorting down to an average cost corresponding to $O(n)$.

If some objects are known never to have target objects behind them, like possibly floors, ceilings and outmost walls, those objects can safely be rendered to the frame buffer first.

In certain cases, like Russian dolls, the sort order between the dolls should be from the innermost to the outermost. A fixed rendering order between the dolls is then user-defined by putting them into the same group with a predefined rendering order, for instance by the

order of appearance in the group. E.g., the innermost doll should be rendered first, and the outermost doll last. This results in a correct transparency, since only the frontmost triangles of the dolls are visible (unlike for classic transparency). If objects are non-solid, like a suitcase or a building, and the inside of the non-solid object should be visible around a target object (see Figure 1), then the triangles of the non-solid object should preferably be individually sorted back-to-front. For the building in Figure 1 this means that the four walls are put into four separate groups. This behavior gives the user a tool to specify which objects that should be regarded as solids and not.

Here is an overview of our algorithm:

- The groups are rendered back-to-front.
- All objects are blended into the frame buffer using the value in the alpha-channel of the frame buffer, which default is 1 (non-transparent), as blending factor.
- Target objects also post-modify the values in the alpha-channel to a value < 1.

The algorithm needs to fulfill these three criteria:

1. Render all parts of objects (target or distractor) in front of a target object as transparent.
2. Render each object as a solid, i.e., only the front-most surfaces should be visible. Thus, the objects cannot be rendered as transparent in an ordinary sense. Back-facing triangles, or further front-facing triangles, should not be visible through transparent frontmost triangles.
3. Draw a gradual transition from no transparency to a predefined transparency in a *n*-pixel outline region around each target object (see Figure 2).

Algorithm 1 shows an outline of the main algorithm.

Algorithm 1: Main

Input: set of groups G.
Output: correctly rendered dynamic transparency scene.

```
1 BubbleSort(G), taking advantage of frame coherence.
2 for all groups g ∈ G do
3     for all objects o ∈ g do
4         if o is a target then
5             renderTargetObject()
6         else
7             renderDistractorObject()
```

Initial requirements for rendering both targets and distractors is that (i) the alpha buffer is initiated to 1 for each pixel at the start of each frame, (ii) rendering is done back-to-front on object level, and (iii) the alpha buffer contains the desired blending factor (transparency) at each pixel.

4.1 Rendering Distractor Objects

1. Render object to the z-buffer only (using `GL_LESS`), to mask out frontmost surfaces.
2. Blend object to color buffer (using `GL_EQUAL`).

The first step selects the frontmost surfaces of the object. The second blends these surfaces to the frame buffer, with blending using the alpha values stored in the frame buffer. These alpha values are 1 by default and less in front of, and in an n-pixel region region around, target objects.

4.2 Rendering Target Objects

1. Render step 1 and 2 as for distractor objects.
2. Render alpha mask, i.e. multiplicatively blend an alpha mask (see Figure 2) to the alpha channel of the frame buffer.

The final step ensures that the rendered target is visible by creating a mask that essentially protects the target from being fully overdrawn by following objects.

Figure 2: Alpha mask creation for an occluded target being made visible by dynamic transparency.

4.3 Rendering the Alpha Mask

As specified in Section 3.3, the alpha mask can be any type of shape exposing the underlying target, such as an ellipse or circle. We choose the expanded outline of the object with a transparency gradient as the alpha mask shape.

Multiplying a constant alpha value to the pixels covered by the target object is easily done by simply rendering the object to the alpha-channel only and using a color with the alpha-value set appropriately. Creating the n-pixel wide surrounding transition is a little bit more tricky.

We choose to alternately render to two external full-screen buffers to create a border around the target object with a smooth transition to full opacity. The resolution can be allowed to be quite low. We use a size of 128×128. See Algorithm 2 for pseudo code for the alpha mask algorithm. Refer to Algorithm 3 for the fragment shader pseudo code.

We found that it often looks better to have the transition from full opacity to a low start alpha value α_0 for the gradient outline, while keeping a higher threshold opacity α_T, for fragments directly in front of the target, maximizing both context and discovery.

Algorithm 2: RenderAlphaMask

Input: target object o, mask width n, two buffers B_1 and B_2.
Output: 128×128 alpha mask blended to the frame buffer.

1 Enable buffer B_1.
2 Render the target object o to the alpha channel only, setting the alpha values to α_T, the threshold transparency for objects in front of target objects.
3 Set buffer B_1 as texture.
4 Enable rendering to buffer B_2.
5 **for** *each layer* $\{1 \ldots n\}$ *of mask* **do**
6 Render buffer-sized quad with the fragment shader specified in Algorithm 3.
7 Set the rendered buffer as texture and enable rendering to the other buffer. Each iteration adds one pixel-wide layer of the transition.
8 Increase the border alpha value α_B in the shader incrementally starting from α_0 to 1.0.
9 Disable buffer and activate standard color buffer.
10 Multiplicatively blend the screen-size buffer texture to the color buffer (alpha values). Note that resolutions may differ, but linear filtering quite efficiently hides zooming artifacts.
11 To avoid uggly jagginess at the pixels along the border of the target object due to differences in resolution between the color and mask buffers, render the target region again (Line 2).

Algorithm 3: FragmentShader

Input: border alpha α_B, frame buffer F, screen position P.
Output: alpha value α_P for pixel at position P.

1 **bool** IsBorderPixel $\leftarrow$ **false**;
2 **for** *each neighbor* N *of position* P **do**
3 IsBorderPixel $\leftarrow F(N)$.`Alpha` != 1.0 **or** IsBorderPixel;
4 IsBorderPixel $\leftarrow (F(P)$.`Alpha` == 1.0) **and** IsBorderPixel;
5 **output** IsBorderPixel ? α_B : 1.0;

5 User Study

We hypothesize that users employing dynamic transparency for object discovery in 3D environments would be more efficient as well as more correct in performing their tasks than when not having access to the technique. In order to test these hypotheses, we designed a formal user study comparing the new technique to standard 3D camera navigation techniques. We also developed three separate application examples to use in this evaluation.

We are currently in the process of performing the user study, so only preliminary results are available at this time. However, analysis of the collected measurements indicates that both our hypotheses are correct; for all scenarios, subjects are significantly more efficient (i.e. use less time) and more correct when performing visual search tasks using dynamic transparency than without.

6 Application Examples

In this section we describe three of the application examples we have developed for evaluating the dynamic transparency technique.

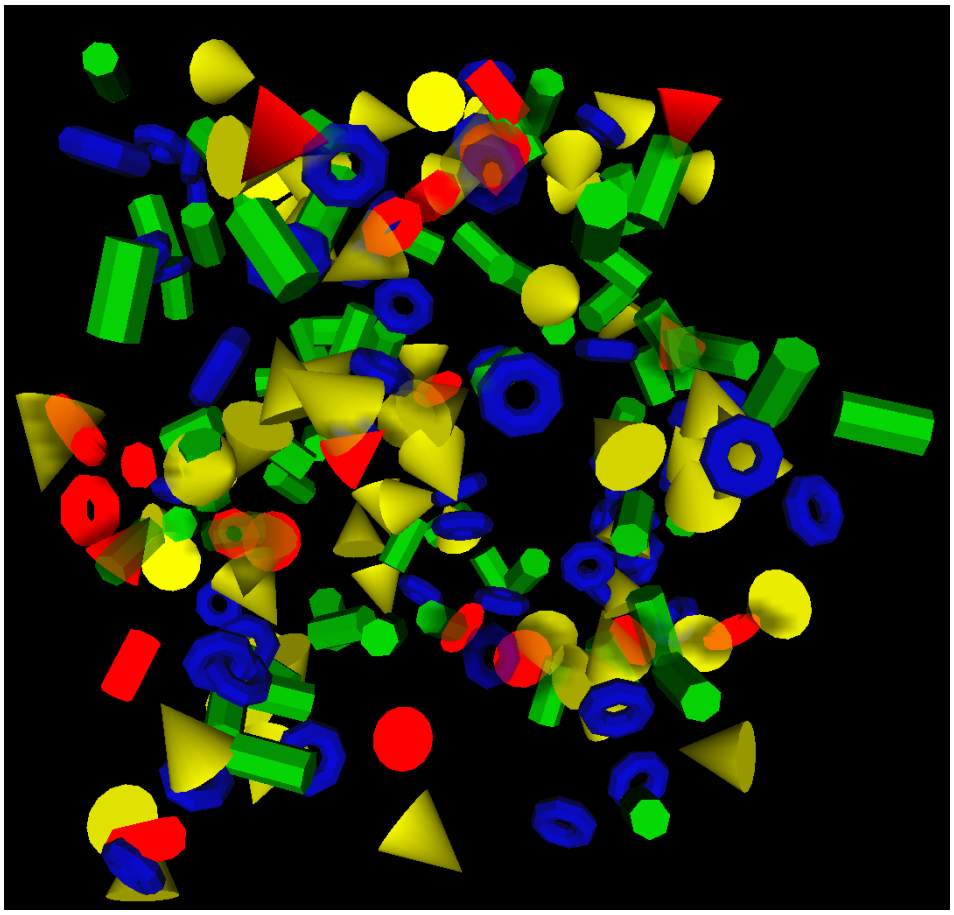

Figure 3: ABSTRACT application with dynamic transparency active.

6.1 Abstract 3D World

The first scenario (ABSTRACT) is intended to portray an abstract 3D visualization application and consists of a cubic 3D volume of size $100 \times 100 \times 100$ filled with $n = 200$ objects of randomized position and orientation (see Figure 3 for a screenshot). The objects are simple unit 3D primitives: spheres, cones, boxes, and torii. Objects are allowed to intersect but not full enclose each other. 10% to 20% of the objects are flagged as targets and the remainder as distractors. Distractor objects are randomly assigned green and blue color component values, while targets were set to a pure red color. The user view is fixed at a specific distance from the center of the environment cube so that no object can fall outside of the view frustrum, and can be freely orbited around the focus point to afford view from all directions. The task amounts to counting the number of targets in each randomized instance of the environment, with dynamic transparency either active or inactive.

6.2 Virtual Walkthrough

The second scenario (WALKTHROUGH) is a little more complex in nature and designed to more closely mimic a real 3D walkthrough visualization application. Here, a one-level floor plan is randomly generated from a simple 16×16 grid, creating walls, floors and ceiling as well as ensuring that all rooms were connected with all of its adjacent neighbors through doorways (see Figure 4 for an example). Again, a number of $n = 100$ objects

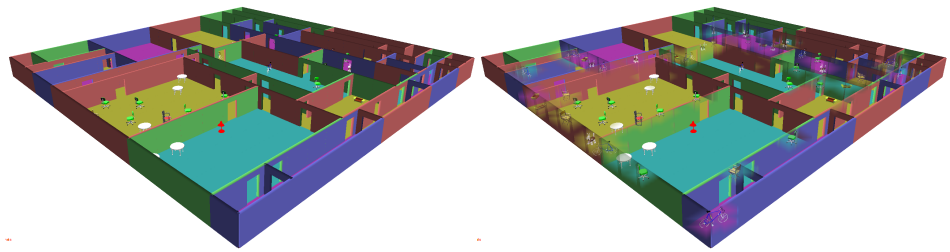

Figure 4: Example floorplan for the WALKTHROUGH application with dynamic transparency inactive and active.

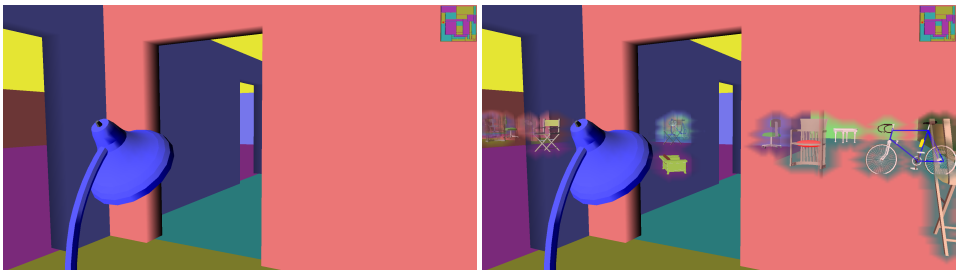

Figure 5: First-person view of the WALKTHROUGH application with dynamic transparency inactive and active.

are generated and placed in the environment, with one of them being the current target, the rest being distractors. All objects are made visible through the walls using dynamic transparency, however. The 3D objects chosen for this scenario were more complex 3D models, including pets, vehicles, and furniture. The user starts each instance in the center of the environment and navigates through it looking for the target using FPS[1]-like controls involving mouse and keyboard (mouse to pan the camera around, arrow keys to move). The view is constrained to floor level at all times. After finding the target, the user moves on to mark its estimated location on a 2D floorplan of the environment (seen from above). See Figure 5 for a screenshot.

6.3 3D Game

The concept of dynamic transparency has applicability beyond the theme of improving object discovery for visualization that this paper focuses on; we believe that it could also prove useful in computer games, where we often may want to temporarily suspend graphical realism by removing intervening objects in order to improve gameplay. Figure 6 shows an example from the third example application (GAME), a would-be 3D strategy game we have developed, rendered in real-time using our algorithm; the player-controlled tank hiding under the cover of the forest is made visible through the foliages in order to help the user see the friendly units. Also see Figure 1 for another example of a game-like scenario with an F-15 fighter aircraft hidden inside a hangar being exposed to the player using dynamic transparency.

[1]First-person shooter.

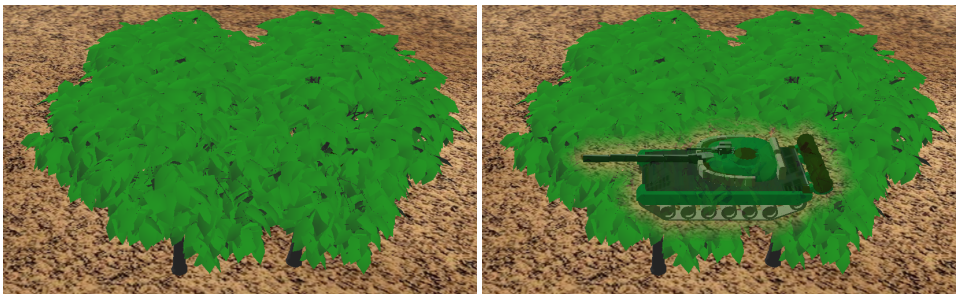

Figure 6: Applying the image-based dynamic transparency algorithm to units in a 3D real-time strategy game.

Application	Resolution	Dynamic Transparency	Framerate
ABSTRACT	800×600	no	87
		yes	33
	1280×1024	no	87
		yes	33
WALKTHROUGH	800×600	no	40
		yes	11
	1280×1024	no	40
		yes	11
GAME	800×600	no	300
		yes	140
	1280×1024	no	188
		yes	90

Table 1: Performance for three example applications with and without dynamic transparency active.

7 Results

Table 1 shows the performance of the three example applications with and without dynamic transparency active. The test was performed on an Intel Pentium 4 desktop computer with 1 GB of memory running Microsoft Windows XP and equipped with an NVidia Geforce 7800 GTX graphics adapter. As can be seen from the measurements, only the GAME application is fillrate-limited (the bottleneck seems to be buffer switching). For the WALKTHROUGH application, we are performing dynamic transparency on 50 complex objects, so 11 FPS is acceptable, if not quite interactive.

8 Discussion

It is important to remember that occlusion is an important depth cue that humans use to determine the spatial relation of objects in our environment; in esssence, the fact that nearby objects occlude more distant ones help us understand our surroundings. Introducing dynamic transparency may then adversely affect this mechanism, and can actually result in "reverse occlusion", i.e. the phenomenon that distant objects all of a sudden occlude nearby objects instead. In our approach, we partly avoid this problem by ensuring that intervening objects made transparent always retain at least some percentage of opacity in order to shine through on uncovered objects. This means that the user receives a visual indication of the existence of the transparent surfaces. Furthermore, in our implementation, we also support toggling dynamic transparency on and off; while turned off, the user is free to perceive the

3D environment in the normal way. Active "flashlight"-mode dynamic transparency can also help avoid this problem.

Another factor that is important for context is the shape of the alpha mask uncovering targets. In our image-space algorithm, we make use of an opacity gradient shaped as the outline of the object. Other alternatives would be to use a circle, ellipse, or a 2D convex hull of the object, potentially revealing more of the surrounding context of the target but removing more of the distractors. Our approach strikes a balance between these factors, but the appropriate method depends on the specific application.

One particular issue with the use of dynamic transparency in particular and transparency in general, is that it often causes a high degree of visual clutter in a scene that would otherwise be far less complex. Many users are simply unused to transparent surfaces and entities, and are easily confused by the side-effects of layering and transparency. While this may be a matter of training and habit, it is important to recognize this fact and provide means for the user to turn off the transparency if needed. Alternatively, active transparency, as mentioned above, may be another, less invasive, option.

9 Conclusions

We have presented a new model for dynamic transparency designed with the purpose of minimizing occlusion of important target objects in 3D visualization applications. This is achieved by dynamically adjusting the transparency of 3D surfaces occluding targets, resulting in "superhero-like" vision yet preserving the important context of the surrounding surfaces and objects. We have further devised an image-space algorithm and implementation realizing this model, utilizing modern programmable graphics hardware to create the desired effect. The algorithm uses the standard framebuffer as a cumulative alpha buffer, rendering the scene back-to-front and blending in alpha maps of target objects to allow for see-through surfaces. In order to verify the technique's usefulness for real visualization applications, we have developed three different example applications portraying typical uses of the technique. We are currently using these applications for formal user studies involving human subjects. Preliminary results show that dynamic transparency not only results in more efficient object discovery, but also that users are more correct with the technique than without.

10 Future Work

We envision improving our model for dynamic transparency with a more general interest-based importance scale, allowing users and applications to dynamically specify the relative importance of individual parts of 3D objects to a very high degree (possibly along the lines of the IDVR [28] importance model). We will continue working on more techniques for reducing the impact of occlusion in 3D environments, including the automatic generation of view-dependent animated exploding diagrams as well as the generation of occlusion-free grand tours of a 3D environment. In addition, we are interested in pursuing similar avenues for providing superhuman vision capabilities in information visualization applications.

Acknowledgments

The authors would like to thank Per A. Jonasson for valuable insights into superhero X-ray vision. Many thanks to John Stasko and the Information Interfaces Research Group at Georgia Tech for their comments and feedback on the intermediate stages of this work.

References

[1] Patrick Baudisch and Carl Gutwin. Multiblending: displaying overlapping windows simultaneously without the drawbacks of alpha blending. In *Proceedings of the ACM CHI 2004 Conference on Human Factors in Computing Systems*, pages 367–374, 2004.

[2] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony DeRose. Toolglass and Magic Lenses: The see-through interface. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 73–80, 1993.

[3] Doug A. Bowman, Chris North, Jian Chen, Nicholas F. Polys, Pardha S. Pyla, and Umur Yilmaz. Information-rich virtual environments: theory, tools, and research agenda. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 81–90, 2003.

[4] Loren Carpenter. The A-buffer, an antialiased hidden surface method. *Computer Graphics*, 18(3):103–108, July 1984.

[5] Luca Chittaro and Ivan Scagnetto. Is semitransparency useful for navigating virtual environments? In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 159–166, 2001.

[6] Rudolph P. Darken and John L. Sibert. Wayfinding strategies and behaviors in large virtual worlds. In *Proceedings of the ACM CHI 96 Conference on Human Factors in Computing Systems*, pages 142–149, 1996.

[7] Paul J. Diefenbach. *Pipeline Rendering: Interaction and Realism through Hardware-Based Multi-Pass Rendering*. Ph.D. thesis, 1996.

[8] J. Diepstraten, D. Weiskopf, and T. Ertl. Transparency in interactive technical illustrations. *Computer Graphics Forum*, 21(3):317–325, 2002.

[9] J. Diepstraten, D. Weiskopf, and T. Ertl. Interactive cutaway rendering. In *Proceedings of EUROGRAPHICS 2003*, pages 523–532, 2003.

[10] Niklas Elmqvist and Philippas Tsigas. View projection animation for occlusion reduction. In *Proceedings of the ACM Conference on Advanced Visual Interfaces*, 2006. to appear.

[11] Cass Everitt. Interactive order-independent transparency. NVIDIA Corporation, 2001. See *http://developer.nvidia.com*.

[12] Bert Freudenberg, Maic Masuch, and Thomas Strothotte. Real-time halftoning: A primitive for non-photorealistic shading. In *Proceedings of the 13th Eurographics Workshop on Rendering*, pages 227–232. Eurographics Association, 2002.

[13] Shinji Fukatsu, Yoshifumi Kitamura, Toshihiro Masaki, and Fumio Kishino. Intuitive control of "bird's eye" overview images for navigation in an enormous virtual environment. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 67–76, 1998.

[14] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of the ACM Conference on Computer Graphics (SIGGRAPH '98)*, pages 447–452, 1998.

[15] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard F. Riesenfeld. Interactive technical illustration. In *Proceedings of the ACM Symposium on Interactive 3D*, pages 31–38, 1999.

[16] Carl Gutwin, Jeff Dyck, and Chris Fedak. The effects of dynamic transparency on targeting performance. In *Proceedings of Graphics Interface 2003*, pages 105–112, 2003.

[17] Beverly L. Harrison, Gordon Kurtenbach, and Kim J. Vicente. An experimental evaluation of transparent user interface tools and information content. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 81–90, 1995.

[18] Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the ACM Conference on Computer Graphics (SIGGRAPH '98)*, pages 453–460, 1998.

[19] Tobias Isenberg, Bert Freudenberg, Nick Halper, Stefan Schlechtweg, and Thomas Strothotte. A developer's guide to silhouette algorithms for polygonal models. *IEEE Computer Graphics and Applications*, 23(4):28–37, 2003.

[20] Edward W. Ishak and Steven K. Feiner. Interacting with hidden content using content-aware free-space transparency. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 189–192, 2004.

[21] Douglas S. Kay and Donald P. Greenberg. Transparency for computer synthesized images. In *Computer Graphics (SIGGRAPH '79 Proceedings)*, pages 158–164, 1979.

[22] Jock D. Mackinlay, Stuart K. Card, and George Robertson. Rapid controlled movement through a virtual 3D workspace. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, pages 171–176, 1990.

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

[24] Marc Nienhaus and Jürgen Döllner. Blueprints: Illustrating architecture and technical parts using hardware-accelerated non-photorealistic rendering. In *Proceedings of Graphics Interface*, pages 49–56, 2004.

[25] Nicholas F. Polys and Doug A. Bowman. Design and display of enhancing information in desktop information-rich virtual environments: challenges and techniques. *Virtual Reality*, 8(1):41–54, 2004.

[26] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *Proceedings of the ACM Conference on Computer Graphics (SIGGRAPH 2001)*, pages 581–581, 2001.

[27] John Viega, Matthew J. Conway, George Williams, and Randy Pausch. 3D magic lenses. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 51–58, 1996.

[28] Ivan Viola, Armin Kanitsar, and Eduard Gröller. Importance-driven volume rendering. In *Proceedings of IEEE Visualization 2004*, pages 139–145, 2004.