Employing Dynamic Transparency for 3D Occlusion Management: Design Issues and Evaluation

Abstract. Recent developments in occlusion management for 3D environments often involve the use of dynamic transparency, or virtual "X-ray vision", to promote target discovery and access in complex 3D worlds. However, there are many different approaches to achieving this effect and their actual utility for the user has yet to be evaluated. Furthermore, the introduction of semi-transparent surfaces adds additional visual complexity that may actually have a negative impact on task performance. In this paper, we report on an empirical user study comparing dynamic transparency to standard viewpoint controls. Our implementation of the technique is an image-space algorithm built using modern programmable shaders to achieve real-time performance and visually pleasing results. Results from the user study indicate that dynamic transparency is superior for perceptual tasks in terms of both efficiency and correctness.

1 Introduction

The ability to utilize the full 3D space as a canvas for information-rich [1] 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 interaction between the objects.

A number of recent solutions to this problem involve the use of *dynamic transparency*, also known as virtual X-Ray [2], to make targets visible by turning intervening surfaces semi-transpareny on-demand as the user moves through the 3D world (see Figure 1 for an example). However, this approach may instead introduce additional visual complexity and reduce the user's depth perception. Furthermore, the actual utility of these techniques remains unknown.

In this paper, we evaluate the usefulness of dynamic transparency for solving visual tasks in both abstract and realistic environments. Note that dynamic transparency cannot be realized using the standard model for transparency, and no real-time performance algorithm exists in the literature that fulfills our requirements. Therefore, we also present an image-space algorithm for dynamic

Fig. 1. Dynamic transparency uncovering an engine inside a jeep.

transparency that makes use of fragment shaders for the new generation of programmable graphics hardware to perform occlusion detection in the image space and with real-time rendering performance. The effect is somewhat akin to the "X-ray vision" of a superhero.

We performed the evaluation by constructing two different application examples depicting common scenarios within our problem domain: an abstract 3D environment of simple geometric primitives similar to information visualization applications, and a 3D virtual walkthrough application for a complex building environment. We then conducted a quantitative user study in these two scenarios where we compared the time and correctness performance of human subjects using our technique as opposed to using standard 3D navigation controls. Results from this study show that dynamic transparency allows for significantly improved efficiency for perception tasks compared to standard methods, both in terms of completion times as well as correctness. In general, the approach seems to be superior for understanding 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 model for dynamic transparency that captures a natural way of achieving high efficiency for perceptual tasks; (ii) an efficient image-space algorithm for dynamic transparency using the new generation of programmable graphics hardware; and (iii) results from a formal user evaluation showing that dynamic transparency significantly improves both time performance and correctness for visual tasks involving discovery, access, and spatial relation of objects in 3D environments.

This paper is organized as follows: We first discuss the related work in the field and then present a model for dynamic transparency. In Section 4, we give our algorithm that realizes the requirements put down in the previous section. Section 5 and 6 present the user study and our results. We end the paper with some discussion of the results and conclusions.

2 Related Work

The general dynamic transparency approach makes heavy use of semi-transparent surfaces to reduce the impact of occlusion as well as to avoid the loss of 3D depth cues completely. In order to achieve correct results, transparent surfaces must be rendered in depth order. Everitt [3] discusses the *depth peeling* image-space algorithm for achieving this on modern graphics hardware based on the virtual pixel map concepts introduced by Mammen [4] and the dual depth buffers by Diefenbach [5]. The blueprints [6] technique uses depth peeling to outline perceptually important geometrical features of complex models using transparency and edge detection. However, depth peeling is a computationally demanding method and interactive frame rates can only be achieved for relatively low depth complexity.

Dynamic transparency is also commonly used in 3D games and virtual environments to allow users to see through occluding surfaces; Chittaro and Scagnetto [7] investigate this practice and conclude that see-through surfaces are more efficient than normal 3D navigation, although not as efficient as bird's-eye views.

Diepstraten et al. introduce view-dependent transparency [8] for use in 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, and is thus not limited to a specific depth.

In another paper, Diepstraten et al. also present their work on computer-based *break-away* views [9], where interior objects are made visible through the surface of containing objects through image-space holes. While again similar to our work, 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 method is to compute the convex hull of interior objects in a pre-processing step and use it as a clipping volume. More importantly, their approach does not handle the case when several targets line up and occlude each other, a necessary requirement for dynamic visualizations with a high target density. Our method requires no off-line preprocessing and 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.

Looser et al. [10] describe a 3D magic lens implementation for Augmented Reality that supports information filtering of a 3D model using the stencil buffer, allowing the user to utilize a looking glass to see through the exterior of a house and into its interior, for instance. This approach relies on the 3D model having semantically differentiated parts, whereas our method requires no such extra information. Coffin and Höllerer [11] present a similar technique with active interaction where the user is controlling a CSG volume that is dynamically subtracted from the surrounding world geometry, again using the stencil buffer. This work does not rely on any semantic target information at all and facilitates exploratory interaction like active dynamic transparency. However, the depth of the volume cutout is limited and user-controlled, and no depth cues from the world geometry are retained other than the cutout border area. With dynamic

transparency, as described in this paper, we are guaranteed to always discover occluded objects regardless of depth, and some depth cues are retained using semi-transparency.

Finally, 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. Viola et al. employ it for volume rendering [12] (IDVR) to actively reduce inter-object occlusion in the same way that we do in this work. While clearly using a more powerful interest model than our work, 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.

Dynamic transparency can also be used in 2D windowing systems (see for example [13–15]) instead of 3D worlds, but this is beyond the scope of this paper.

3 Model for Dynamic Transparency

In this section, we present a model for the dynamic transparency approach. See [2] for a more in-depth treatment of general occlusion management.

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.

An object can be flagged either 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 one. Passing through a surface increases the cumulative transparency of the line segment accordingly (multiplicatively or additively, depending on the transparency model).

3.2 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 affects 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 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 the 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 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 using impenetrable surfaces (and objects).

(R4) Objects are allowed to self-occlude.

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.

4 Image-Space Dynamic Transparency

Since none of the previously presented methods fulfills our requirements, we here present a new algorithm for 3D dynamic transparency: 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, requires 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 [16]. 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 view 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 $O(n)$.

In certain cases, like for 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. In other words, the innermost doll should be rendered first and the outermost doll last. This results in correct transparency, since only the frontmost triangles of the dolls are visible (unlike for classic transparency). This mechanism gives the user a tool to specify which objects that should be regarded as solids and not.

Here is an overview of our algorithm:

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

The algorithm needs to fulfill these criteria:

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()
```

- Render all parts of objects (target or distractor) in front of a target object as transparent.
- 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 more distant front-facing triangles, should not be visible through transparent frontmost triangles.
- Draw a gradual transition from no transparency to a predefined transparency in an n-pixel outline region around each target object.

Algorithm 1 shows an outline of the main algorithm.

Initial requirements for rendering both targets and distractors are 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. Given these preconditions, we render distractor objects in the following way:

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

In contrast, target objects are rendered in the following way:

1) Render step 1 and 2 as for distractor objects.
2) Render alpha mask, i.e. multiplicatively blend an alpha mask 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 subsequently rendered objects.

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* $\{1 \dots n\}$ *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 Render the target region again to avoid jagginess at the border of the target object due to differences in resolution between the color and mask buffers.

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;

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 alpha mask is a little trickier.

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. To achieve this, we render to two external off-screen buffers alternately 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 and refer to Algorithm 3 for the fragment shader code.

4.1 Performance

Table 1 shows the performance of three example applications with and without dynamic transparency active (an abstract environment, an architectural walk-through, and the game-like example in Figure 1). 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.

Application	Triangles	Resolution	Inactive (FPS)	Active (FPS)
ABSTRACT	13,000	800×600	87	33
		1280×1024	87	33
WALKTHROUGH	464,220	800×600	40	11
		1280×1024	40	11
GAME	114,629	800×600	300	140
		1280×1024	188	90

Table 1. Performance for three example applications.

5 User Study

We hypothesize that users employing dynamic transparency for visual perception tasks 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.

5.1 Subjects

We recruited 16 subjects for this study, three of which were female. The subjects were drawn primarily from our university and were screened to have at least basic computer knowledge. Subject ages ranged from 20 to 35 years of age. All subjects had normal or corrected-to-normal vision, and no participants were color-blind. 12 out of 16 subjects had previous extensive 3D experience.

5.2 Equipment

The experiment was conducted on an Intel Centrino Duo laptop computer equipped with 2048 MB of memory running the Microsoft Windows XP operating system. The display was a 17-inch widescreen LCD display running at 1920×1200 resolution and powered by an NVidia Geforce 7800 GO graphics card.

5.3 Tasks and Scenarios

We designed the study to include two widely different scenarios, including an abstract 3D world and a virtual walkthrough in a 3D building, and four different tasks (two per environment). In this way, we aim to be able to measure not only basic target discovery, but also the more complex visual tasks of access and spatial relation.

5.4 Scenario: 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 2 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 and made visible using our dynamic transparency technique (for Task 2, distractors could be red as well). 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 frustum, and can be freely orbited around the focus point to afford view from all directions.

Task 1: count the number of targets (red objects) in the environment. (Purpose: *discovery*)
Task 2: identify the pattern formed by the targets (red cones) in the environment. (Purpose: *relation*)

The pattern is one of the five capital letters C, K, R, X, and Y, rasterized in a 5×7 horizontal grid of the same scale as the environment and rotated in an arbitrary fashion around the vertical axis. The subject is informed of the range of possible letters prior to performing the task, but not the exact rasterizations.

5.5 Scenario: Virtual Walkthrough

The second scenario (WALKTHROUGH) is a little more complex in nature and designed to mimic a real 3D walkthrough visualization application more closely. 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. A number of $n = 50$ objects are generated and placed in the environment, and all objects are made visible through the walls using dynamic transparency. The 3D objects chosen for this scenario were more complex 3D models, including pets, vehicles, and furniture, yet were easily distinguishable from each other. The user starts each instance in the center of the environment and navigates through it looking for the target using 3D game-like controls involving the mouse and keyboard (mouse to pan

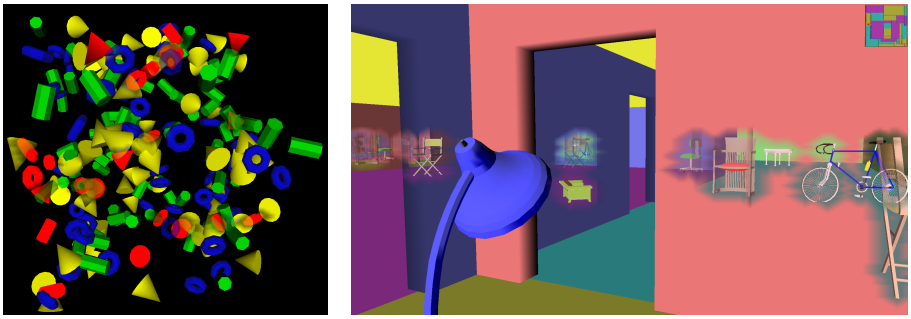

Fig. 2. The ABSTRACT and WALKTHROUGH applications with active dynamic transparency.

the camera around the vertical axis, arrow keys to move, no strafing allowed). The view is constrained to floor level and there is no collision detection with walls or objects.

Task 3: find the unique target in the environment. (Purpose: *discovery*)
Task 4: count the number of targets in the environment. (Purpose: *discovery, relation*)

For the first task, one of the objects in the environment is unique and the user is asked to find this target. The current target is shown in the upper left corner of the screen. After finding the target, the user moves on to mark its estimated location on a 2D floorplan of the environment on a separate screen.

For the counting task, a random number of the objects in the environment are of the same type and the user is asked to count the occurrences. The current object type is again shown in the upper left corner of the screen. After having estimated that all occurrences are found, the subject enters the amount into the application.

5.6 Design

The experiment was designed as a one-way ANOVA for each of the four tasks, with the independent variable DYNTRANS (two levels, "true" or "false", within-subjects). The dependent variables included completion times for all tasks, and the error for the counting tasks, error distance for the search task, and correctness for the pattern task. Subjects received both the tasks and dynamic transparency in counterbalanced order to manage systematic effects of practice.

Each task set consisted of three trials per condition. Completion times and user responses to the tasks were collected and silently recorded by the application. Every task set was preceded by a training session lasting up to five minutes where the subject was instructed in the current task and was allowed to explore the scenario as well as ask questions. During the execution of the actual task

#	Standard	DynTrans	Significance
1	56.26 (38.72)	40.44 (20.99)	$F(1,15) = 7.54, p = .015$
2	22.30 (16.20)	15.80 (10.21)	$F(1,15) = 5.28, p = .036$
3	62.78 (35.63)	23.21 (12.01)	$F(1,15) = 22.98, p < .01$
4	140.0 (61.75)	40.80 (24.16)	$F(1,15) = 48.61, p < .01$

Table 2. Average completion times for all four tasks (standard deviation).

set, only general questions were allowed. A full session lasted approximately 45 to 60 minutes.

6 Results

Analysis of the collected measurements indicates that both our hypotheses are correct; subjects are more efficient (i.e. use less time) and more correct when performing visual search tasks using dynamic transparency than without.

6.1 Time

Overall, the average completion time with inactive dynamic transparency was 65.17 (s.d. 27.75) seconds, compared to 28.69 (s.d. 11.02) with active dynamic transparency. This was also a significant difference ($F(1,15) = 49.54, p < .001$). Each of the individual tasks also showed significantly shorter average completion times for active dynamic transparency compared to inactive dynamic transparency down to $p < .05$. See Table 2 for a summary.

6.2 Correctness

For the counting tasks (task 1 and 4), we define correctness in terms of average relative error, i.e. the ratio between the absolute error and the total number of targets for all trials. The absolute error is the absolute difference between the sum of the targets and the sum of the subject answers for the trials. Overall, for task 1 and 4 combined, the average relative error was .100 (s.d. .141) when dynamic transparency was inactive compared to .027 (s.d. .045) when it was active. This is also a significant difference ($F(1,15) = 6.28, p = .024$).

Task 1 in particular showed average relative error of .042 (s.d. .046) for inactive dynamic transparency and .017 (s.d. .018) for active. This too was significant ($F(1,15) = 4.74, p = .046$). Task 4 showed .123 (s.d. .184) and .034 (s.d. .074) average relative error, respectively, not a significant difference ($F(1,15) = 4.12, p = .061$).

For task 2, we define correctness as whether or not the subject identified the pattern as the correct one. This figure was .963 (s.d. .109) for no dynamic transparency and .963 (s.d. .150) for active. This is obviously not a significant difference.

Finally, for task 3, we define correctness as the average Euclidean distance (in world units) from the real position of the target and the point marked on the map by the subject for each trial. With dynamic transparency inactive, this average distance was 16.99 (s.d. 14.44), as opposed to 16.21 (s.d. 8.88). This difference is not significant ($F(1, 15) = .068, p = .797$), and indicates that the spatial understanding of the subjects was not negatively affected by the use of dynamic transparency.

7 Discussion

It is important to remember that occlusion is a vital depth cue that humans use to determine the spatial relation of objects in our environment. The introduction of 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 address 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. Self-reported ratings from the subjects themselves seem to indicate that depth perception is still acceptable with dynamic transparency active.

Fortunately, human perception relies on many more factors besides occlusion to disambiguate depth; examples include stereopsis, motion parallax, atmospheric perspective, texture gradient, etc. Even if we weaken the occlusion cue, other depth cues will help the viewer to perceive the 3D scene correctly.

Some subjects in our study had the interesting behavior of "respecting" the world more when dynamic transparency was inactive, using the doors in the virtual walkthrough rather than going through walls, whereas they would not hesitate to pass through walls when it was active. While this is an informal observation, this behavior might indicate that the impact that dynamic transparency has on visual realism causes the world to become more ethereal and less believable to the users, thus making them ignore the implicit rules of the environment.

8 Conclusions

We have presented an evaluation of the use of dynamic transparency for managing occlusion of important target objects in 3D visualization applications. In the absence of real-time algorithms for dynamic transparency that are suitable for interactive visualization, we have further devised an image-space algorithm and implementation realizing the model. The algorithm uses the standard framebuffer as a cumulative alpha buffer, rendering the scene back-to-front and blending in alpha masks of target objects to allow for see-through surfaces. Our evaluation consisted of a comparative user study evaluating efficiency and correctness gains from using the technique as opposed to standard 3D navigation

controls. Our results clearly show that dynamic transparency not only results in more efficient object discovery, but also that users are more correct with the technique than without.

References

1. Bowman, D.A., North, C., Chen, J., Polys, N.F., Pyla, P.S., Yilmaz, U.: Information-rich virtual environments: theory, tools, and research agenda. In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology 2003. (2003) 81–90
2. Elmqvist, N., Tsigas, P.: A taxonomy of 3D occlusion management techniques. In: Proceedings of the IEEE Conference on Virtual Reality 2007. (to appear)
3. Everitt, C.: Interactive order-independent transparency. NVIDIA Corporation (2001) See *http://developer.nvidia.com*.
4. Mammen, A.: Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. IEEE Computer Graphics and Applications **9**(4) (July 1989) 43–55
5. Diefenbach, P.J.: Pipeline Rendering: Interaction and Realism through Hardware-Based Multi-Pass Rendering. Ph.D. thesis, Computer Graphics, University of Pennsylvania (1996)
6. Nienhaus, M., Döllner, J.: Blueprints: Illustrating architecture and technical parts using hardware-accelerated non-photorealistic rendering. In: Proceedings of Graphics Interface 2004. (2004) 49–56
7. Chittaro, L., Scagnetto, I.: Is semitransparency useful for navigating virtual environments? In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology 2001. (2001) 159–166
8. Diepstraten, J., Weiskopf, D., Ertl, T.: Transparency in interactive technical illustrations. Computer Graphics Forum **21**(3) (2002) 317–325
9. Diepstraten, J., Weiskopf, D., Ertl, T.: Interactive cutaway rendering. In: Proceedings of Eurographics 2003. (2003) 523–532
10. Looser, J., Billinghurst, M., Cockburn, A.: Through the looking glass: the use of lenses as an interface tool for augmented reality interfaces. In: Proceedings of GRAPHITE 2004. (2004) 204–211
11. Coffin, C., Höllerer, T.: Interactive perspective cut-away views for general 3D scenes. In: Proceedings of the IEEE Symposium on 3D User Interfaces 2006. (2006) 25–28
12. Viola, I., Kanitsar, A., Gröller, E.: Importance-driven volume rendering. In: Proceedings of the IEEE Conference on Visualization 2004. (2004) 139–145
13. Gutwin, C., Dyck, J., Fedak, C.: The effects of dynamic transparency on targeting performance. In: Proceedings of Graphics Interface 2003. (2003) 105–112
14. Baudisch, P., Gutwin, C.: 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. (2004) 367–374
15. Ishak, E.W., Feiner, S.K.: Interacting with hidden content using content-aware free-space transparency. In: Proceedings of the ACM Symposium on User Interface Software and Technology 2004. (2004) 189–192
16. Carpenter, L.: The A-buffer, an antialiased hidden surface method. Computer Graphics **18**(3) (July 1984) 103–108