Partial Visibility for Stylized Lines

Forrester Cole
Princeton University

Adam Finkelstein
Princeton University

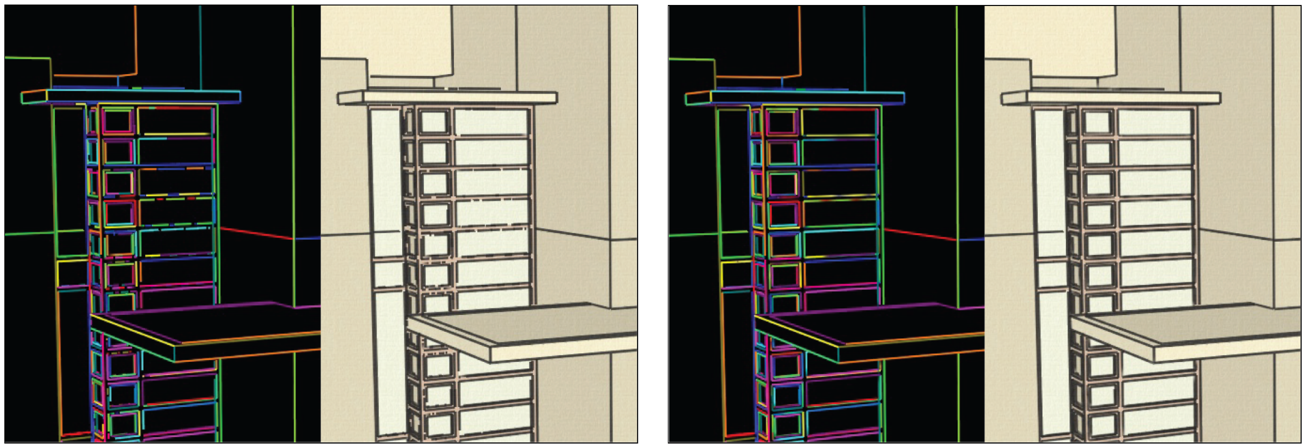

Figure 1: Anti-aliased line visibility. *Left pair:* Aliasing in the visibility test for lines (visualized over black) causes breaks and other artifacts in the rendered lines (drawn on beige model). *Right pair:* Supersampling and ID peeling removes these artifacts from the rendered lines.

Abstract

A variety of non-photorealistic rendering styles include lines extracted from 3D models. Conventional visibility algorithms make a binary decision for each line fragment, usually by a depth test against the polygons of the model. This binary visibility test produces aliasing where lines are partially obscured by polygons or other lines. Such aliasing artifacts are particularly objectionable in animations and where lines are drawn with texture and other stylization effects. We introduce a method for anti-aliasing the line visibility test by supersampling, analogous to anti-aliasing for polygon rendering. Our visibility test is inexpensive using current graphics hardware and produces partial visibility that largely ameliorates objectionable aliasing artifacts. In addition, we introduce a method analogous to depth peeling that further addresses artifacts where lines obscure other lines.

Keywords: NPR, Line Drawing, Visibility, Hidden Line Removal

1 Introduction

Common to many non-photorealistic rendering (NPR) techniques is the use some form of stylized lines. Such lines may include static 3D features such as creases and texture boundaries, as well as view-dependent features such as silhouettes, suggestive contours and suggestive highlights. Using the conventional graphics pipeline, such features may be drawn as solid, straight lines in 3D, and their visibilty can be resolved using the standard z-buffer algorithm (typically offset slightly relative to the polygons to disambiguate visibility where the lines and polygons are colocated). However, the z-buffer approach cannot be used for lines drawn with stylization effects such as varying thickness, over- or under-shoot, wavy path, and texture (e.g., Figure 6) because such effects cause the lines to be drawn in areas of the image near but not exactly identical to the location where visibility should be tested. Therefore, algorithms for drawing stylized lines from 3D models generally compute visibility for the lines prior to rendering them.

Typical line visibility algorithms generate a binary decision for every line fragment. Unfortunately such tests are subject to aliasing, leading to rendering artifacts such as those shown on the left in Figure 1. This paper describes an anti-aliasing approch to line visibility that results in partial visibility at every line fragment and, as shown on the right, largely ameliorates aliasing artifacts in the rendered lines. The specific contributions are:

- The notion of rendering lines with *partial visibility* as a mechanism for anti-aliasing.
- An algorithm for computing partial visibility of lines by super-sampling from an item buffer.
- The use of *ID peeling* in an item buffer to improve rendering quality where lines overlap in image space, and providing a quality-performance tradeoff when readback of the item buffer becomes expensive.

2 Background and Related Work

For an overview of line visibililty approaches, especially with regard to silhouettes, see the survey by Isenberg et al. [2003]. One general strategy combines visibility and rendering by simply causing the visible lines to appear in the image buffer, for example the techniques of Raskar and Cohen [1999] or more recently Lee et al. [2007], both of which worked at interactive frame rates by using hardware rendering. These approaches limit stylization because by the time visibility has been calculated, the lines are already drawn. On the other hand, *explicit* computation of line visibility has been the subject of research since the 1960's. For example, Appel [1967] introduced the notion of *quantitative invisibility* (QI), and computed it by finding changes in visibility at certain (typically rare) locations. This approach was further improved and adapted to NPR by Markosian et al. [1997] who showed it could be performed at interactive frame rates for models of moderate complexity.

Appel's algorithm and its variants can be difficult to implement and are somewhat brittle when the lines are not in general position. Thus, Northrup and Markosian [2000] adapted the use of an *item buffer* (which had previously been used to accelerate ray tracing [Weghorst et al. 1984]) for the purpose of line visibility, calling it an "ID reference image" in this context. Several subsequent NPR systems have adopted this approach, e.g. [Kalnins et al. 2002; Kalnins et al. 2003; Cole et al. 2006], and the algorithm described in this paper also builds on this strategy. Kaplan [2007] described a method for computing QI using an item buffer, and we believe we could compute "partial" QI by combining his method with ours.

Any binary visibility test, including the item buffer approach, will lead to aliasing artifacts, analogous to those that appear for polygons when sampled into a pixel grid. The classic polygon aliasing artifacts are the "jaggies" that appear along the boundaries of a polygon, where the polygon covers only a fraction of a pixel. Considerable effort has been devoted to antialiasing for polygons [Foley et al. 1990]. A common strategy for addressing such artifacts is to supersample, for example with the use of the A-buffer [Carpenter 1984]. This paper demonstrates that such methods, originally developed for polygons, can be adapted to anti-alias line visibility with a similar quality-performance tradeoff.

This paper also describes *ID peeling*, which is based on the *depth peeling* approach described by Everitt [2001]. Depth peeling was originally used to correctly render transparent objects without depth sorting. As described in Section 3.2, the ID peeling can be adapted to allow the item buffer to store more than one line per pixel.

Finally we note that partially visible lines have already been used for various stylistic effects in NPR. For example Winkenbach and Salesin [1994] and Hertzmann and Zorin [2000] controlled line density among hatching lines by varying line weight and opacity.

3 Algorithm

The basis of our line rendering pipeline is the item buffer method of Northrup and Markosian [2000]. An item buffer is an off-screen buffer that contains visibility information for a set of 3D lines. To create an item buffer, the polygonal model is first drawn into the depth buffer. Each individual line is then drawn into the color buffer with a unique color (as in Figure 1), while testing against the model's depth buffer. For a model M, a set of 3D lines L, and associated colors C, the item buffer is created as follows:

```
def drawItemBuffer(M, L, C):
    set color mask = false, depth mask = true
    draw M
    set color mask = true, depth mask = false
    draw each l in L, colored by C
```

Lines are drawn with depth mask (writing to the depth buffer) disabled to prevent z-fighting between lines. Drawing the item buffer without depth writing usually does not cause additional visual artifacts, because when all lines are drawn in a similar style, it is usually not possible to tell which line is in front and which is behind.

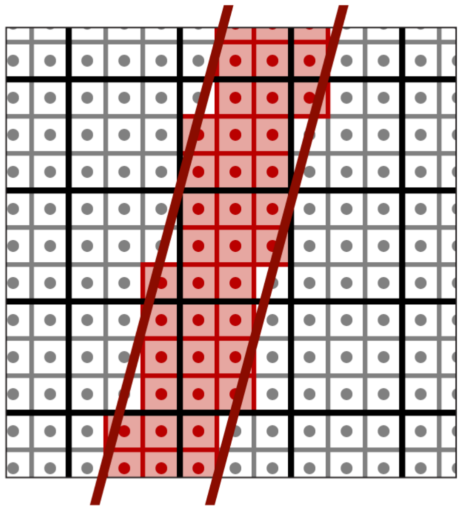

Figure 3: 9× *Supersampling.* Black lines are pixel boundaries, gray lines are subpixel sample boundaries. The edges of a fully visible line are dark red. Red subpixel samples fall within this line. Even though the line is fully visible, no single pixel (black box) contains nine red samples.

Each pixel of the item buffer contains the unique color of a single visible line fragment at that pixel. While efficient and fairly accurate, this approach suffers from two major flaws, one general and one peculiar to the item buffer. The general flaw is that an item buffer has limited resolution, and cannot be trivially anti-aliased due to the special meaning of the line colors. Any visibility algorithm (e.g. raytracing) suffers from this restriction, and our supersampling implementation also generalizes to visibility approaches besides the item buffer. The particular flaw is that only a single line color can exist as a given pixel, even if more than one line fragment is visible at that pixel. This restriction is due to the limited size of the graphics card's framebuffer, and ID peeling is a way to circumvent this hardware restriction.

3.1 Supersampling

The conventional approach to anti-aliasing for rendering is to take several sub-pixel samples and average their colors to obtain the final pixel color. This approach fails in the case of the item buffer, however, because color is used to encode the line indices. Averaging two colors results in a spurious line index. In order to supersample the item buffer, we need an aggregating operation that preserves the proper line indices.

Our approach is first to render the item buffer at high resolution – between two and six times the full-screen resolution. This can be done by either increasing the screen resolution and drawing the lines with width equal to the supersampling factor (width 2 for 2×2 supersampling), or drawing multiple copies of the item buffer with subpixel offsets. We chose the latter, because although drawing the geometry multiple times can degrade performance, we have noticed that thick lines in OpenGL behave unpredictably and may vary from platform to platform. Subpixel offsets also allow the possibility of jittered supersampling with random offsets, though we have not implemented such a scheme.

A fully visible line fragment may lie across the subpixel samples of multiple adjacent pixels (Figure 3). If each pixel contains n samples, we label each sample from $1..n$. A fragment's visibility is

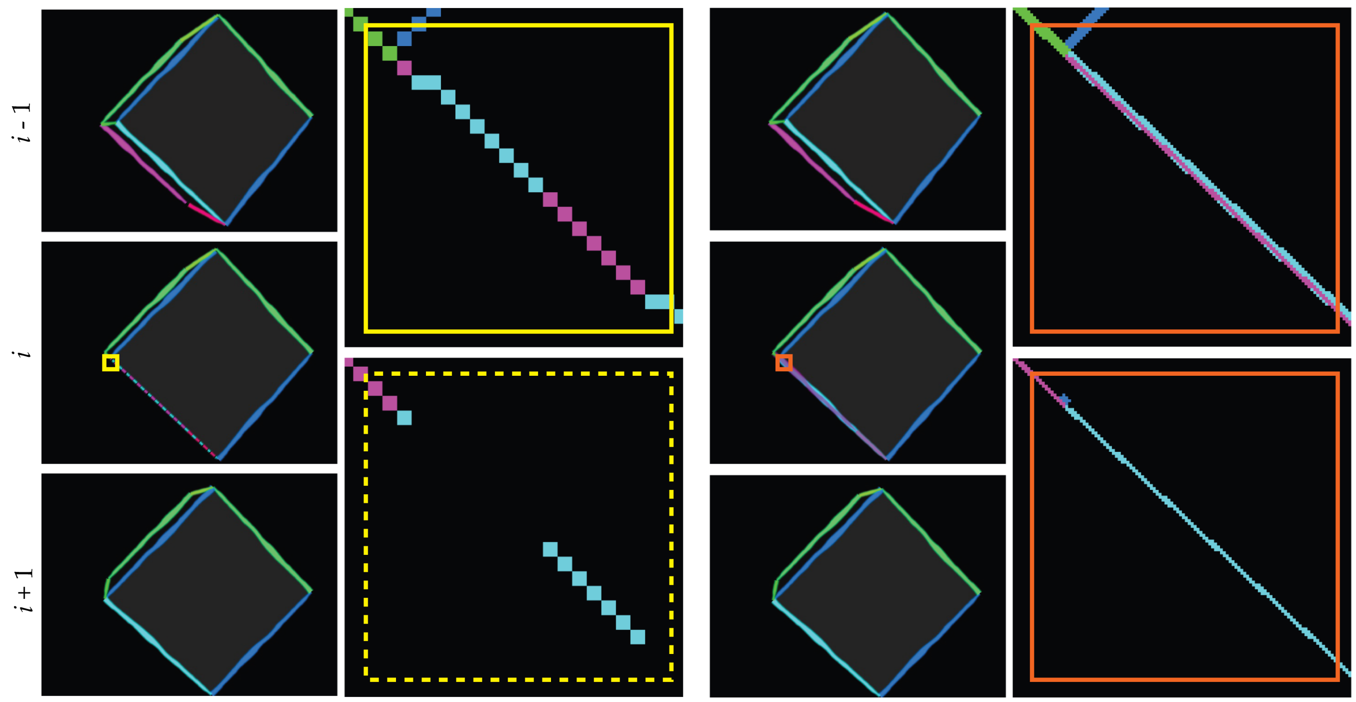

Figure 2: *Aliasing at visibility changes.* Item buffer aliasing and overwriting at changes in visibility can cause obvious artifacts in stylized lines (left). For a single item buffer (enlarged solid yellow), the pink edge interferes with the aqua edge as it transitions from visible to invisible. A hypothetical second item buffer layer (enlarged dotted yellow) could recover the pixels overwritten by other lines. With both supersampling and ID peeling the aqua line is intact and the pink line is partially visible on top of it (right: two layers of item buffer enlarged in orange).

determined by the number of sample labels covered in a 3x3 pixel neighborhood around the fragment. For example, if $n = 4$ and the line covers samples $1, 3$ in one pixel and samples $2, 4$ in an adjacent pixel, the fragment has full visibility. Because of the neighborhood check, this test can overestimate the visibility of a line fragment by up to one pixel.

3.2 ID Peeling

A conventional item buffer holds only a single ID per pixel. In even simple models, however, multiple lines will often project to the same item buffer pixel (Figure 2). This problem becomes worse as the model becomes more complex. For a large number of cases, however, only a small number of lines will fall on any single pixel (Table 2). We can exploit this property by adapting the technique of depth peeling [Everitt 2001].

In depth peeling, multiple layers of depth information are obtained by rendering the scene multiple times, each time allowing a fragment to pass the depth test only if it is farther from the camera than the closest fragment at the same position in the previous layer. Our version is similar, except that instead of using a second depth test, we allow a line fragment to pass only if its *index* is lower than the highest index at the corresponding pixel in the previous layer (assuming lines are drawn in ascending order). If the maximum number of lines overlapping a single pixel is n, we can recover the full visibility information in n passes.

The result is a set of n item buffers that when taken together, provide complete visibility information for each line (Figure 2).

4 Results

Supersampling and ID peeling together repair most of the visual artifacts associated with visibility testing using an item buffer (e.g., Figures 1 and 2). However, both methods impose a performance cost. Table 1 shows the effect of supersampling and ID peeling on framerates for the Falling Water model. The Falling Water model is a relatively complex model with many parallel and overlapping lines, so it provides a good "stress test" for our algorithm. Both supersampling and ID peeling impose a sub-linear performance cost, though ID peeling dominates. Tripling the number of layers (from 1 to 3) roughly halves frame rate, while for the same performance hit the number of samples may be increased from 1 to 9.

We have found that for almost all models and views, nearly full visibility information can be recovered with three or four item buffer layers, though to remove all artifacts under animation more layers may be required. More layers are also required at high supersampling levels, as supersampling tends to increase the item buffer depth complexity.

	Base	4×	9×	16×	25×
1 Layer	17.6	13.6	9.7	7.1	5.3
2 Layers	12.5	9.0	6.1	4.2	3.0
3 Layers	9.8	6.9	4.5	3.1	2.2

Table 1: *Frames per second for supersampling and ID peeling.* Timings are from a rotating a full view of the house model from Figure 1 at 800x600 window resolution. The model has approximately 10,000 line paths. Tested system had a 2.3GHz Athlon 64 CPU and an NVIDIA 8800GTS GPU.

Model \ Layers:	1	2	3	4	5	6
Box	0.89	0.11	0	0	0	0
House view 1	0.78	0.16	0.05	0.01	0	0
House view 2	0.57	0.24	0.09	0.06	0.02	0.01

Table 2: *Fraction of item buffer pixels that overlap multiple lines.* Of the item buffer pixels that touch any line, the vast majority touch four or fewer lines. The box view is Figure 2b, House view 1 is Figure 1, and House view 2 is the same view, but zoomed out until the entire model is visible (see Figure 5). No supersampling was used for this experiment.

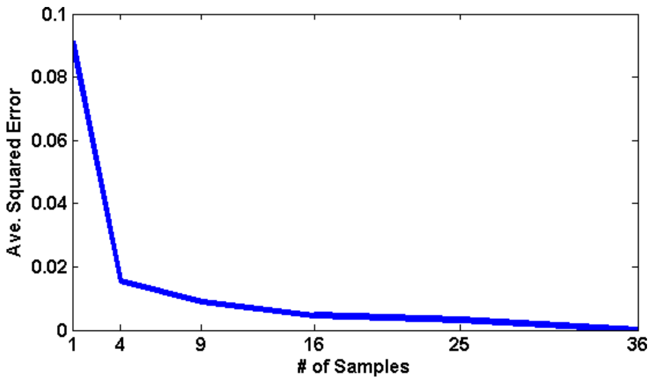

Figure 4: *Supersampling Error.* The average squared error in visibility against the number of samples taken, for the view in Figure 1. Error is computed against supersampling at 36x, which is considered zero error. Eight item buffer layers are used. The knee in the curve appears at 4× supersampling, and there is little gain after 16× supersampling.

Table 2 shows the percentage of item buffer pixels that touch one or more lines. For simple models such as the box, there are no pixels that contain more than two lines. For complex models such as the Falling Water, some pixels can overlap many lines, though for the view shown in Figure 1, 99% of all pixels that overlap any line overlap three or fewer lines. For a more difficult view, where the model is zoomed out until it fits entirely on the screen (Figure 5), 91% of all line pixels overlap three or fewer lines. Exceptional cases exist: for example, one could imagine zooming out until the entire model fit under a single pixel. In such pathological cases, however, perfect visibility information is usually not important.

The gain from supersampling generally falls off rapidly after only four samples, as measured by the average squared difference in visibility from an image created with 36 samples (Figure 4). Qualitatively, we find there are usually small visual differences that can be detected up to 16 samples, but after that point the visual impact of additional samples is minimal.

Finally, the accompanying video demonstrates the impact of these enhancements under animation. An animated cube shows aliasing artifacts that are addressed first by ID peeling and then by supersampling. The next example shows these effects for the more challenging Falling Water model, which includes many tiny overlapping lines. We note that in this example, 9× supersampling is used together with 8 ID layers, resulting in reduced aliasing. However, we believe that in this case, more sampling would further improve the result, but would exceed the memory capacity of our graphics card.

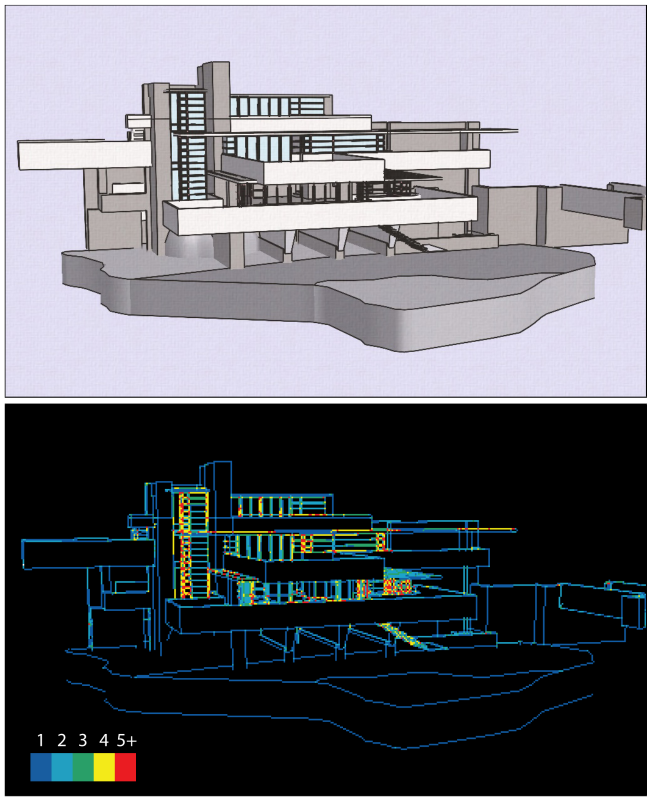

Figure 5: *ID Peeling.* Top: the “House 2” view. Bottom: visualization of item buffer layers. Colors indicate number of overlapping lines at each pixel.

5 Conclusion and Future Work

Supersampling and ID peeling together drastically reduce the number of objectionable visual artifacts when rendering stylized lines. For interactive applications, most of the benefit can be had by using only 4× supersampling and 3-4 item buffer layers. For offline animation, there is no reason not to use many samples and many item buffer layers to achieve the best possible quality.

There are several directions for future work in this area. First, our supersampling and ID peeling are currently too slow to achieve the best possible quality at interactive rates. While we were conscious of performance when creating our implementation, we left several possibilities for further optimization open.

For complex models and high supersampling rate, drawing the entire set of lines once per sample can become expensive. In these cases, it may be advantageous to draw a single, scaled-up item buffer and deal with the vagaries of thick line drawing.

The major and relatively fixed cost of the item buffer algorithm is the readback from the GPU and the processing on the CPU. Deeper layers of ID peeling tend to contain very few non-zero pixels. It may be possible to gain efficiency by using a hierarchical representation such as a quadtree where empty branches can be pruned.

In the longer run, our goal is to move the entire line visibility processing pipeline onto the GPU and avoid CPU-side processing as much as possible. Current graphics hardware should contain the functionality necessary to achieve this, but mapping the item buffer algorithm onto the GPU efficiently remains a challenge.

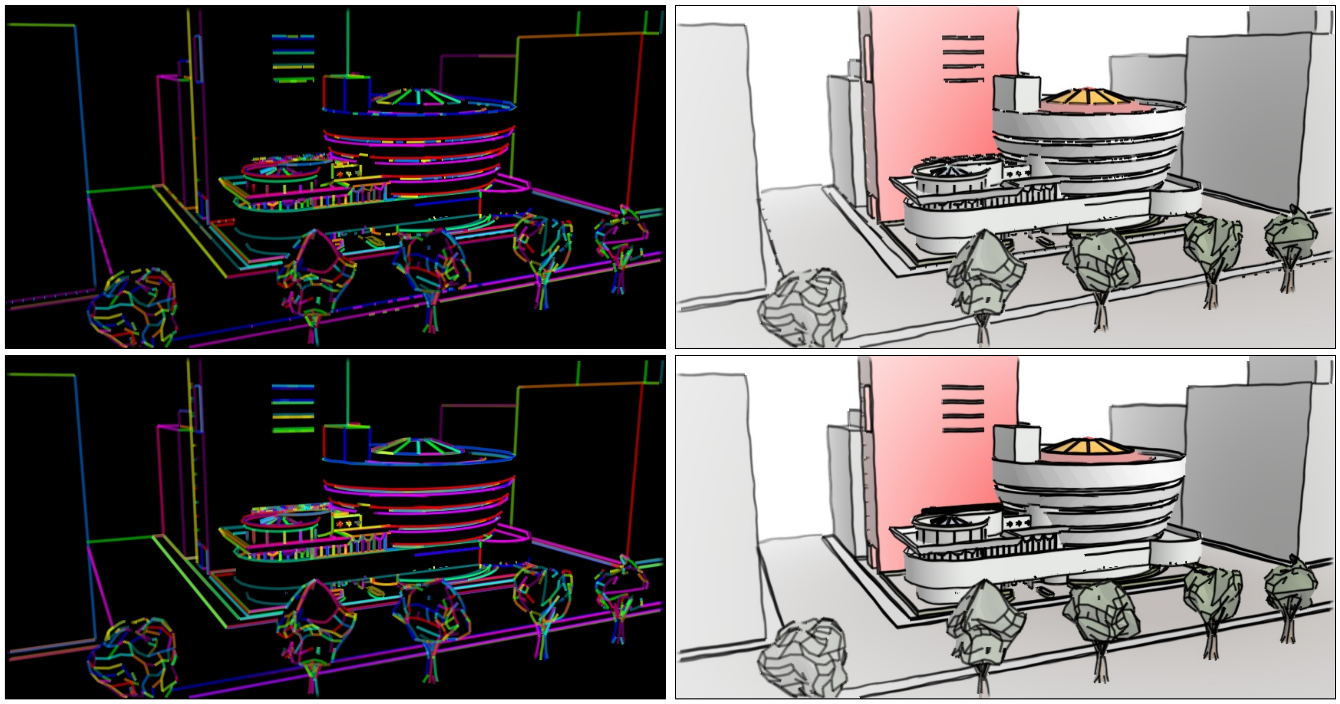

Figure 6: *Guggenheim Museum.* In typical views of complex models, there exist lines at the cusp of visibility (e.g., the rings of the tower). Top left: conventional item buffer visualization. Top right: resulting image. Bottom pair: 9× supersampling and 5 ID peeling layers.

References

APPEL, A. 1967. The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of the 22nd national conference of the ACM*, 387–393.

CARPENTER, L. 1984. The A-buffer, an antialiased hidden surface method. *SIGGRAPH Comput. Graph. 18*, 3, 103–108.

COLE, F., DECARLO, D., FINKELSTEIN, A., KIN, K., MORLEY, K., AND SANTELLA, A. 2006. Directing gaze in 3D models with stylized focus. *Eurographics Symposium on Rendering* (June), 377–387.

EVERITT, C., 2001. Interactive order-independent transparency. Technical report, NVIDIA Corporation, May 2001. Available at http://www.nvidia.com/.

FOLEY, J. D., VAN DAM, A., FEINER, S. K., AND HUGHES, J. F. 1990. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 517–526.

ISENBERG, T., FREUDENBERG, B., HALPER, N., SCHLECHTWEG, S., AND STROTHOTTE, T. 2003. A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE Computer Graphics and Applications 23*, 4 (July/Aug.), 28–37.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: drawing strokes directly on 3d models. In *Proceedings of SIGGRAPH 2002*, 755–762.

KALNINS, R. D., DAVIDSON, P. L., MARKOSIAN, L., AND FINKELSTEIN, A. 2003. Coherent stylized silhouettes. *ACM Transactions on Graphics 22*, 3 (July), 856–861.

KAPLAN, M. 2007. Hybrid quantitative invisibility. In *NPAR '07: Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, 51–52.

LEE, Y., MARKOSIAN, L., LEE, S., AND HUGHES, J. F. 2007. Line drawings via abstracted shading. *ACM Transactions on Graphics 26*, 3 (July), 18:1–18:5.

MARKOSIAN, L., KOWALSKI, M. A., GOLDSTEIN, D., TRYCHIN, S. J., HUGHES, J. F., AND BOURDEV, L. D. 1997. Real-time nonphotorealistic rendering. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 415–420.

NORTHRUP, J. D., AND MARKOSIAN, L. 2000. Artistic silhouettes: a hybrid approach. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, 31–37.

RASKAR, R., AND COHEN, M. 1999. Image precision silhouette edges. In *SI3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 135–140.

WEGHORST, H., HOOPER, G., AND GREENBERG, D. P. 1984. Improved computational methods for ray tracing. *ACM Transactions on Graphics 3*, 1 (Jan.), 52–69.

WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *Proceedings of SIGGRAPH 1994*, 91–100.