Line Illustrations $\in$ Computer Graphics

Gershon Elber
Department of Computer Science
Technion, Israel Institute of Technology
Haifa 32000
Israel
Email: gershon@cs.technion.ac.il

March 23, 2004

Abstract

The revolution of the computer graphics field during the last two decades made it possible to create high quality synthetic images that even experts find it difficult to differentiate from real imagery.

In this paper, we explore a partially overlooked theme of computer graphics that aims at conveying simple information using simple line drawings and illustrations of polygonal as well as freeform objects.

Key Words: Sketches, Depth Cueing, Haloed Lines, Free Form Surfaces.

1 Introduction

Methods and techniques such as ray tracing and radiosity on one side and texture mapping and sophisticated shading models on the other made it possible to create synthetic images that appear photorealistic.

Lack of frame buffers combined with a limited computational power, during the early days of computer graphics two decades ago, forced the extensive research and

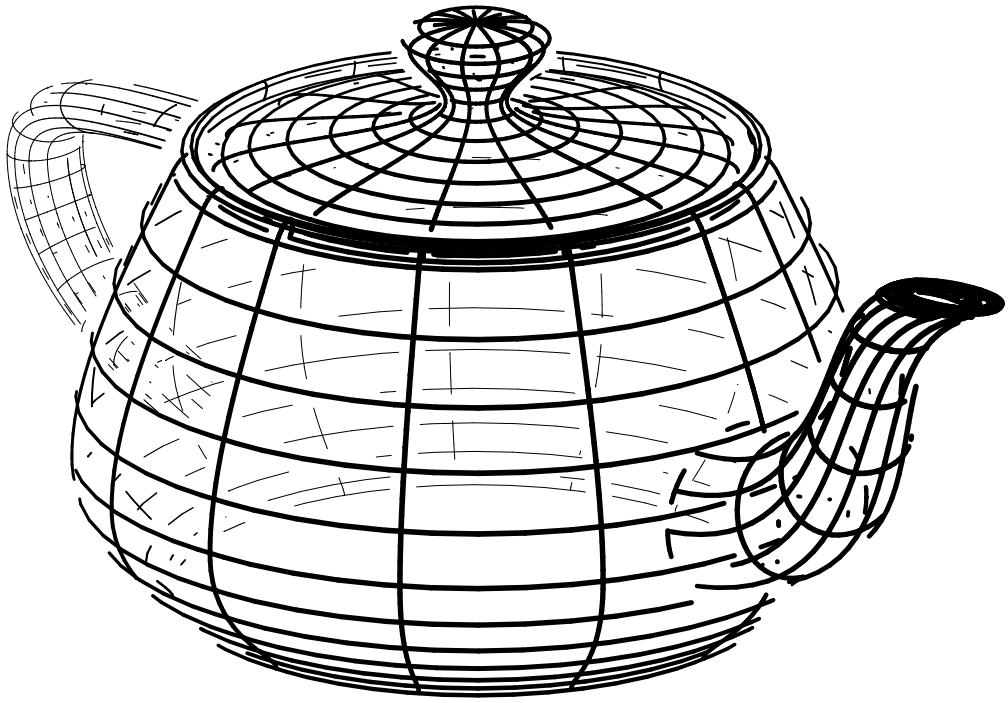

Figure 1: An example of line illustration of the Utah teapot. Depth cueing using line width is combined with edges trimmed in intersection locations to enhance the appearance of the figure.

use of line drawings and hidden line removal. Unfortunately, the automatic generation of line drawings and illustrations was partially overlooked in the last decade for the benefit of high quality rendering. Even today, photorealistic rendering is computationally expensive. However, in many instances a simple line drawing can be sufficient. Figure 1 shows a computer generated illustration of the Utah teapot. This illustration aesthetically conveys information about the object, displaying both the visible and the hidden portions of the model while allowing one to delineate between the two. Further, the illustration is less expensive computationally compared to a photorealistic rendering.

This paper has been inspired by the illustrations found in the excellent geome-

try book by Hilbert and Cohn-Vossen [Hilbert 1990]. We will try to show that the computer graphics field has the ability to contribute to the automation of line illustrations, such as the ones in [Hilbert 1990] for polygonal as well as freeform objects and models. Line drawings are often used in technical and scientific books and reports, probably more frequently than photorealistic rendered images. Yet, little can be found in the computer graphics literature on attempts to improve and automate the generation of line drawings and illustrations.

Automatic hidden line removal is one aspect of line drawings and illustrations that was thoroughly investigated in computer graphics. The notion of quantitative (in)visibility was introduced in [Appel 1967] to derive a general hidden line removal algorithm for polygonal objects. This algorithm served as a basis for numerous other hidden line algorithms for polygonal objects. In [Elber 1990, Hornung 1985], methods are investigated to extend the quantitative invisibility notion to freeform surfaces and create hidden curve removal algorithms. In [Dooley 1990a, Dooley 1990b, Saito 1990], line drawings, possibly intermixed with photorealistic rendering, are generated semi-automatically. In [Dooley 1990a, Dooley 1990b], the user is provided with an editing tool that allows one to specify different attributes to curves with different level of quantitative invisibility. The visibility of the curves is determined using an extension to the hidden curve removal algorithm described in [Elber 1990] that fully exploits the idea of the quantitative (in)visibility [Appel 1967]. In [Saito 1990], a similar combination of line and shaded drawing is presented. Unlike the methods in [Dooley 1990a, Dooley 1990b, Elber 1990], feature lines such as silhouettes are computed in [Saito 1990] by applying differential operators to the Z-depth image of the scene and by combining the image of the feature lines with the shaded image.

Illustrations are recognized as a very important tool to delineate ideas and portray complex mechanical structures. It is common for artists and illustrators to exploit techniques that create unrealistic drawings to better convey the intended ideas [Magnan 1970, Young 1985]. Use of the perspective transformation in computer graphics is one such example that is widely used. Depth cueing and widening lines that are closer to the viewer are other examples that are less frequent.

In this paper, we extensively exploit the capabilities of the PostScript page description language [Postscript 1985] to automate the process of illustrations of polygonal as well as freeform models. We explore several techniques that are commonly used in artistic line drawings. We will exploit the drawing of isoparametric curves to represent freeform parametric surfaces. In [Forrest 1979], it is recognized that isoparametric line drawings can be misleading because they also portray a non intrinsic surface property, the parametrization. Other curves such as contour and silhouette lines are suggested instead. Herein, and in a pursuit for simplicity, we limit our discussion and examples to drawings of isoparametric curves. Nonetheless, nothing in the illustrative techniques presented herein is limited to isoparametric curves.

Section 2 discusses several, simple and automatic techniques that can be used to enhance the understanding and clarity of line drawings. Techniques that are common in artistic line drawings such as the ones in [Hilbert 1990] are adopted and employed in computer graphics.

All the figures in this paper were created using an illustration tool developed as part of the IRIT [IRIT 1993] solid modeling system, developed at the Technion. The output of the illustration tool is a definition of a figure in the PostScript page description language.

2 Computer Generated Illustrations

It is quite surprising how simple techniques can enhance the appearance and appreciation of a line drawing. Line drawings are commonly used in computer graphics to provide fast or even real time display of geometric models.

Given a polygonal model, the polyline boundary of each polygon is usually extracted as a wireframe representation of the polygon. Given a free form model, two sets of isoparametric curves, in both parametric directions, are usually utilized. The resulting polylines can then be displayed. See Algorithm 1. Interestingly enough, every line in the scene is traversed twice in Algorithm 1. If the topology of the model is well defined and all polygons are properly oriented, every edge that is shared by two polygons will be traversed twice in the opposite directions. One can ensure that an edge, $\mathcal{E}$, is drawn only once by uniquely enumerating the vertices of the scene. Let P_1 and P_2 be the two end vertices of $\mathcal{E}$. $\mathcal{E}$ is drawn if $Enum(P_1) < Enum(P_2)$, where $Enum(P_i)$ enumerates vertex P_i. An simpler approach selects $Enum(p_i) = x_i$, for which the possibility of $Enum(P_1) = Enum(P_2)$ must be properly handled.

Varying the line width in proportion to the distance to the eye is a common technique in artistic line illustrations which is easily adapted into computer generated line drawings. Figure 2 (a) is a polygonal model drawn with line thickness that is directly proportional to the distance from the eye. Figure 2 (b) is the same object drawn with constant line thickness. In order for the thickness of the line to continuously vary along the line, each line is broken into small enough segments. Each segment is then drawn with slightly different width. Using the PostScript page description language, the drawing is exploiting the language's control over the line width.

Changing the line thickness as a function of the distance from the eye is a common

Algorithm 1 - `Wireframe representation of objects`

Input:
`` $\mathcal{P}$ and $\mathcal{S}$, set of polygons and surfaces; ``
Output:
`` $\mathcal{L}$, polyline representation of the geometry; ``
Algorithm:

```
L ⇐ ∅;
For each P in P do
    L ⇐ L ⋃ { polyline boundary of P };
end
For each S in S do
    L ⇐ L ⋃ { set of isoparametric curves
                              representing S as polylines };
end
```

feature in illustrations. However, using the computer graphics' depth cueing technique one can as easily set the color or the grey level of the line to be a function of the distance to the eye. This slight change enabled the generation of Figures 3, 4, and 5. However, not all the drawn segments are now painted with the same color or grey level. The broken segments must be sorted in depth to make sure the visibility is properly maintained, and only then drawn back to front. See Figure 3. The process is summarized in Algorithm 2. τ, the maximal length of a line segment, directly controls the gradual change of the line width or color and can be found empirically.

One can always find pathological cases [Foley 1990] for which the Z sorting of the small segments is not sufficient to solve the hidden line problem. Not being a total order, the hidden line removal problem cannot always be solved using Z sorting. Herein two small segments that are drawn in the wrong order due to the sorting failure must be adjacent in their Z depth and therefore will be drawn in a similar

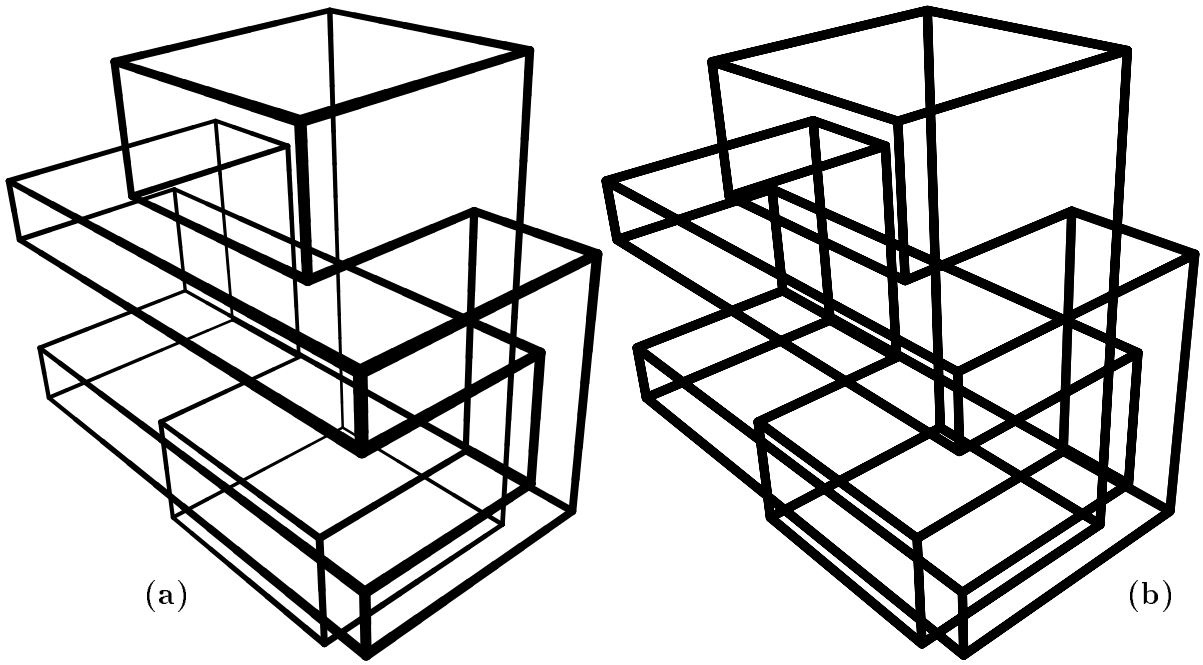

Figure 2: This simple polygonal object is drawn with line width that is proportional to the distance to the eye in (a) and with constant line width in (b).

grey level. Hence, simple Z sorting can provide the correct solution in the majority of the cases, as can be seen in Figures 3, 4, and 5.

Another common technique in illustrations is to slightly trim the hidden edge at an intersecting point with another edge, a method used in [Appel 1979, Franklin 1987], and also known as *haloed lines*. Using a plane sweep algorithm [Preparata 1985], one can find all the k intersections of all the n line segments in the projection plane in an order of $O((n + k)log(n))$ operations. We consider only *valid intersections*, intersections that occur in the interior of the edge and that the two intersecting edges assume sufficiently different Z or depth. Once a valid intersection point is detected, the edge farther from the viewer is trimmed by a prescribed amount, δ, at the intersection. In [Appel 1979], an enhancement that changes the gap size in proportion to the difference in depth of the lines was found to be confusing. Herein,

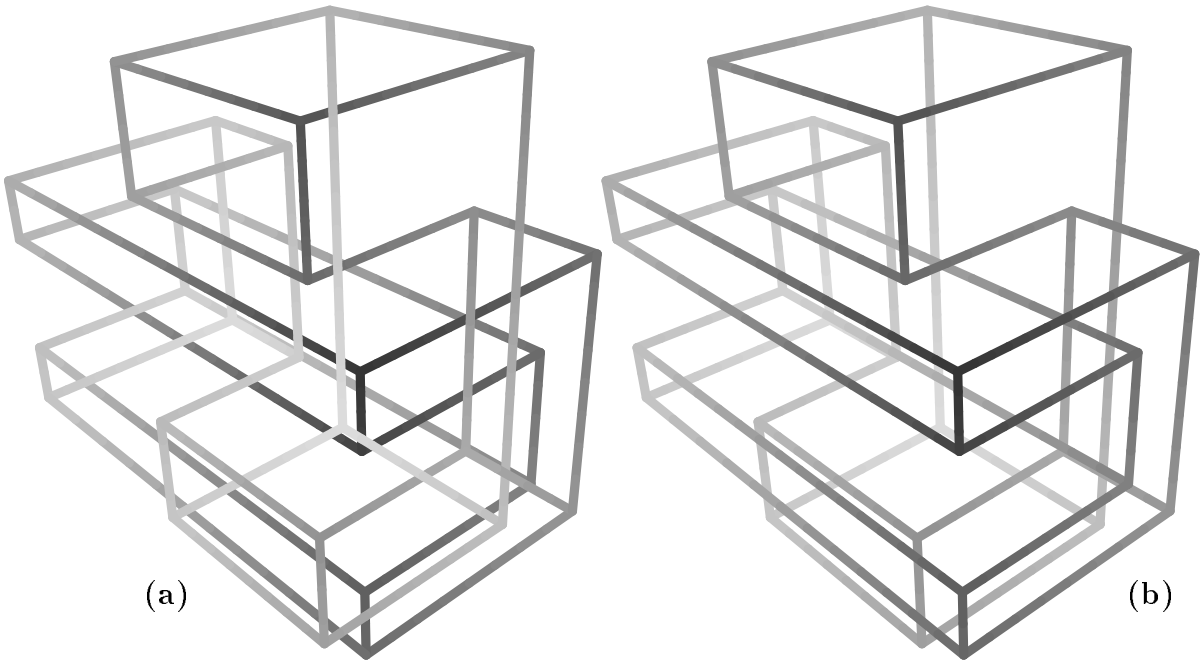

Figure 3: Depth cueing effects can be achieved by changing grey levels with according to the depth. In (a), we see the result with no Z sorting, by directly drawing the lines in the order that they are given. In (b), the segments are first sorted in Z and then drawn back to front.

we found it more appealing to trim the edge in proportion to the angle formed with the other edge at the intersection, θ. A trimming amount equal to $\delta/sin(\theta)$ up to a limiting upper bound when the angle is acute, was used. Algorithm 3 summarizes the approach. Algorithm 3 should be applied *before* Algorithm 2 so that trimming information could be propagated from one line segment to the next, along the polyline. Figures 1, 6, 7 and 8 show few examples. A side effect of this trimming algorithm is that edges farther from the viewer can be totally trimmed away, converging to a drawing with no hidden lines. Figure 6 show several such cases. Visible edges are slightly trimmed along the silhouettes while small hidden segments are also drawn, giving the line drawing a less "perfect" and more of an "illustration" appearance.

Algorithm 2 - `Depth cueing of wireframe representation`

Input:
$\mathcal{L}$, Set of polylines, result of Algorithm 1;
τ, maximal length of a line segment;
Output:
$\mathcal{R}$, Ordered set of line segments from $\mathcal{L}$, each shorter than τ, sorted according to Z;
Algorithm:

```
R ⇐ ∅
For each polyline P in L do
    For each line segment L in P do
        If ( Length(L) > τ ) do
            R ⇐ R ⋃ { L subdivided into
                                    Length(L) / τ segments };
        end
        else do
            R ⇐ R ⋃ {L};
        end
    end
end
Sort segments in R according to Z value of middle of
                                                    segments;
```

A technique that is commonly used by illustrators is to thicken a little the trimmed edge, at the trimmed regions. Figure 9 shows simple examples. Interestingly enough, we can intensify the color or grey level of the edge instead of making it wider, at trimmed intersection locations. See, for example, Figure 9.

Illustrations can also exploit traditional hidden line removal algorithms. One can totally remove the hidden portion or draw it in different attributes such as thinner lines. See Figures 10 and 11.

One can enhance the appearance of vertices of a polygonal model by drawing

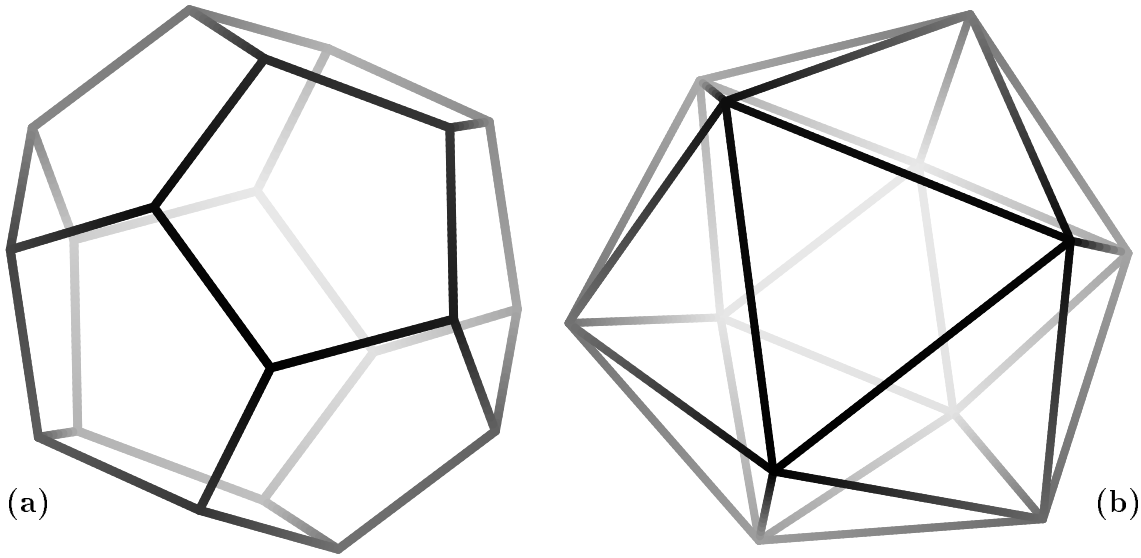

Figure 4: A dodecahedron (a) and an icosahedron (b) drawn using depth cueing of grey levels.

all the vertices in the scene as circular points. The size of the points can be scaled according to the depth or distance from the eye, in a similar way to the scaling or width of the line segments in proportion to their depth. Figure 12 shows two examples. For the hollowed points (Figure 12 (b)) to show up properly, the points should be depth sorted together with the linear segments so that they will be drawn in the appropriate order.

So far we have discussed techniques to enhance the appearance of a model. Frequently, one can find in hand drawn illustrations objects that are introduced to augment the scene. Two examples are shown in Figure 13. Added objects in Figure 13 (a) represent the *heat* wave emanating from a dish, while the introduced straight lines in Figure 13 (b) yield the *speed* sensation of the plane.

Such properties can be attached into a model and processed by the presentation tool. Algorithm 4 shows how the speed sensation can be added. The heat wave is created in a similar way, using a template of a helical curve.

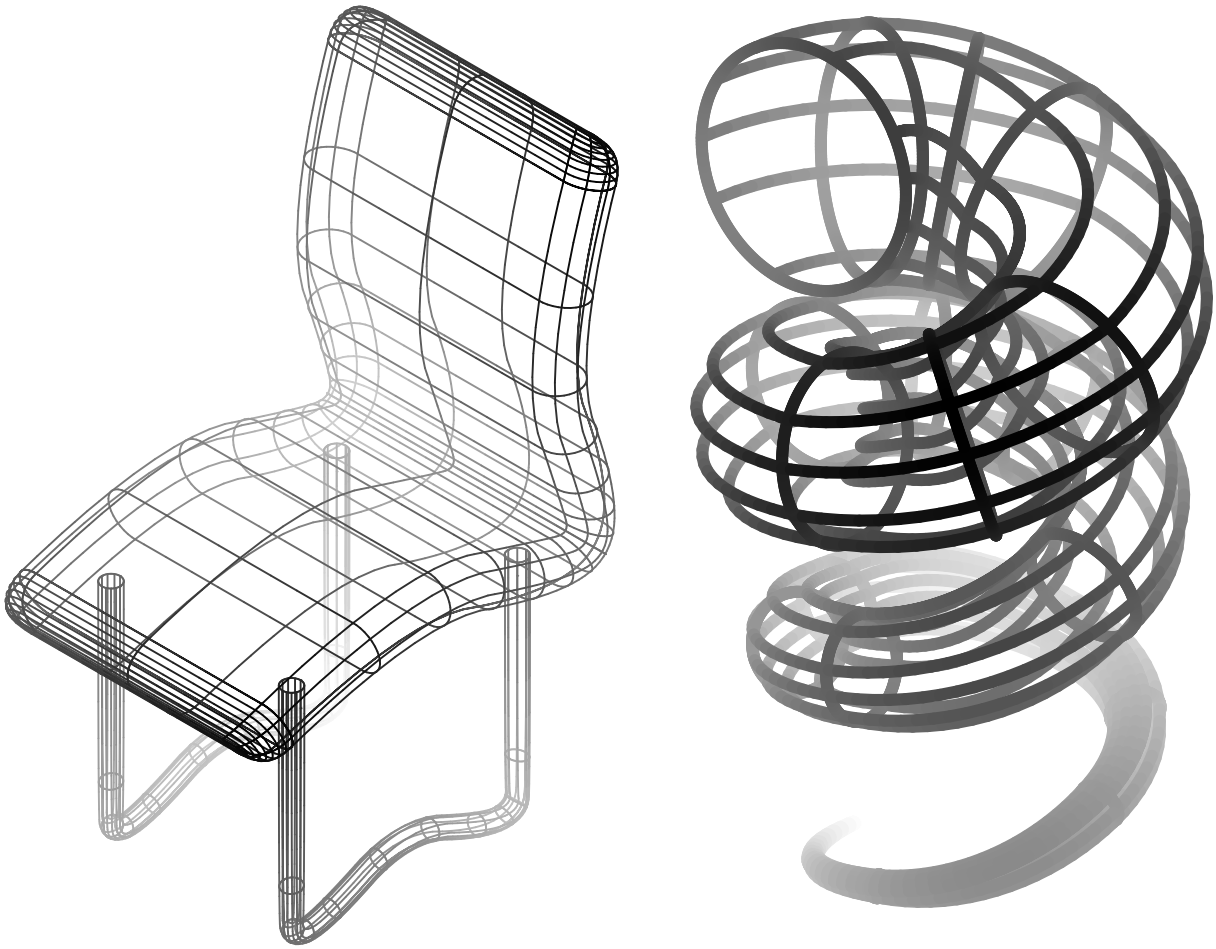

Figure 5: Free form NURB surfaces, drawn using depth cueing of grey levels.

3 Conclusions

This paper uses a set of simple computer graphics techniques to enhance the quality of line drawings and illustrations. Automatic yet fast and simple algorithms were shown to be useful tools to not only improve the quality, clarity and aesthetic display of line drawings but also to give them more of an illustration or a sketch appearance.

Although they can be combined with, none of the algorithms presented herein must be coupled with hidden line removal algorithms. Furthermore, the techniques presented can achieve similar or even better clarity than that presented by line drawings with hidden lines removed. An illustration can also present the invisible information

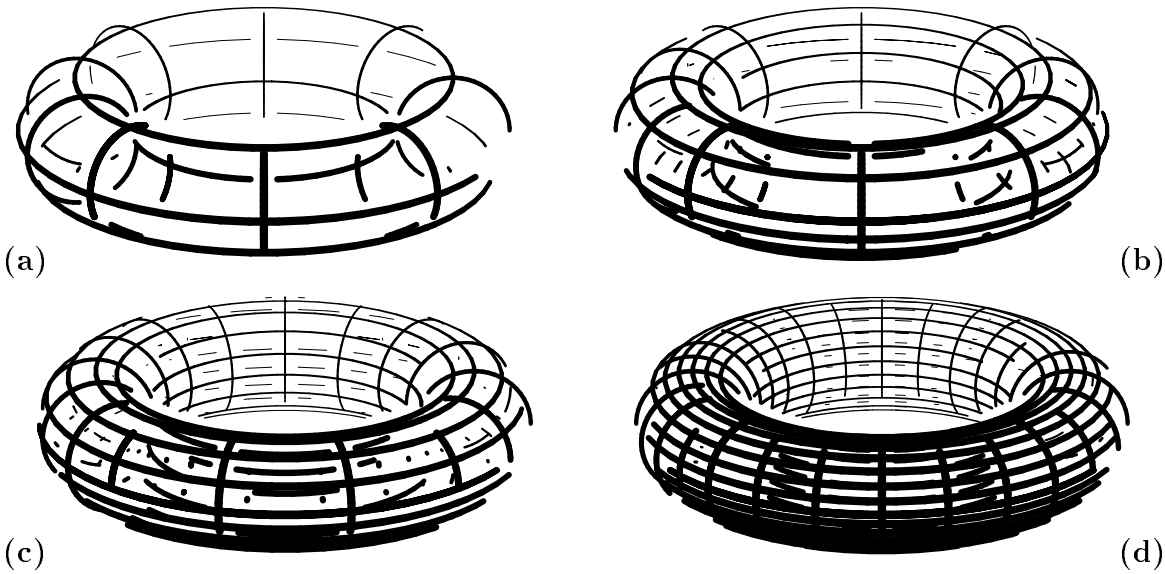

Figure 6: A torus with its edges trimmed at intersection locations using Algorithm 3. Algorithm 3 can be used as an approximation for hidden line removal. Shown are four stages, each with about twice as many isoparametric curves. The hidden curves are incrementally removed as more isoparametric curves are introduced.

of a model while providing the ability to delineate the visible part from the hidden.

We hope that this paper will refocus the interest in techniques to automate the generation of appealing line drawings and aesthetic illustrations.

References

[Appel 1967] **A. Appel.** The Notion of Quantitative Invisibility and the Macine Rendering of Solids. *Proceedings of ACM National Conference*, pp 387-393, 1967.

[Appel 1979] **A. Appel, F. J. Rohlf, and A. J. Stein.** The Haloed Line Effect for Hidden Line Elimination. *ACM Computer Graphics*, Vol. 13, No. 2, pp. 151-157, Siggraph 1979.

Algorithm 3 - `Haloed lines of wireframe representations`

Input:
`` $\mathcal{L}$, Set of polylines, result of Algorithm 1; ``
`` $\delta$, trimming amount; ``
Output:
`` $\mathcal{T}$, Set of trimmed polylines; ``
Algorithm:
```
    I ⇐ Set of all intersections in L;
    For each valid intersection I in I do
        θ ⇐ Angle between intersecting edges;
        Trim polyline further from the viewer, an amount
               equal to δ/sin(θ), from both sides of intersection;
    end
    T ⇒ Resulting, possibly trimmed, polylines;
```

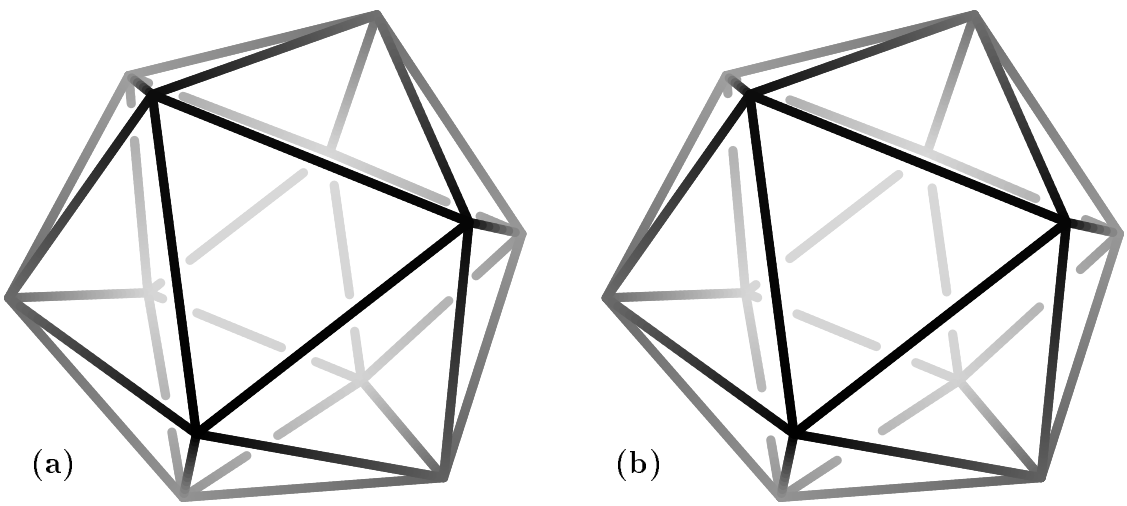

Figure 7: An icosahedron drawn using depth cueing of grey levels and trimmed edges at intersection points. In (a), the icosahedron is drawn with constant trimming length, while in (b), the icosahedron is drawn with trimming length which is a function of the angle between the two intersecting lines.

[Dooley 1990a] **D. L. Dooley and M. F. Cohen.** Automatic Illustration of Three Dimensional Models: Lines. *ACM Computer Graphics*, Vol. 24, No. 2, pp. 77-82, Symposium of Interactive 3D Graphics, 1990.

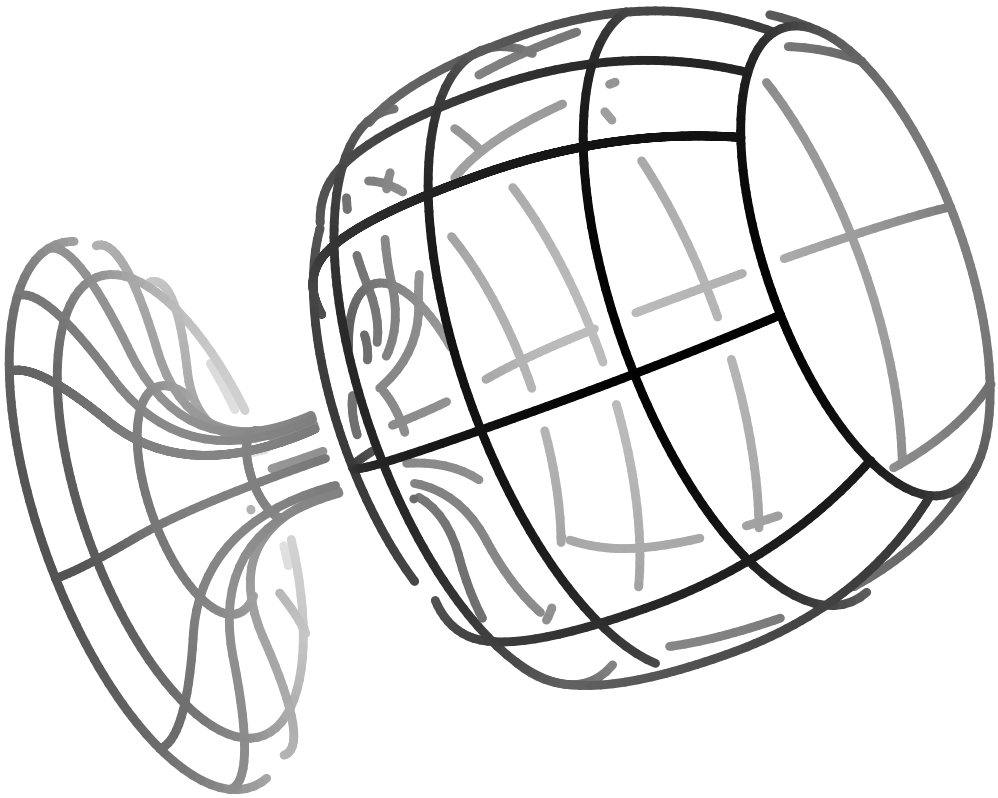

Figure 8: A freeform model of a glass drawn using depth cueing of grey levels and trimmed edges at intersection points.

[Dooley 1990b] **D. L. Dooley.** Computer Illustration of Three-Dimensional Sculptured Surfaces. *Master Thesis*, Computer Science Department, University of Utah, August 1990.

[Elber 1990] **G. Elber and E. Cohen.** Hidden Curve Removal for Free Form Surfaces. *ACM Computer Graphics*, Vol. 24, No. 4, pp. 95-104, Siggraph 1990.

[Foley 1990] **J. D. Foley, A. Van-Dam, S. K. Feiner and J. F. Hughes.** Computer Graphics, Principles and Practice, Second Edition. Addison-Wesley Systems Programming Series, Jul. 1990.

[Forrest 1979] **A. R. Forrest.** On the Rendering of Surfaces. *ACM Computer Graphics*, Vol. 13, No. 2, pp. 253-259, Siggraph 1979.

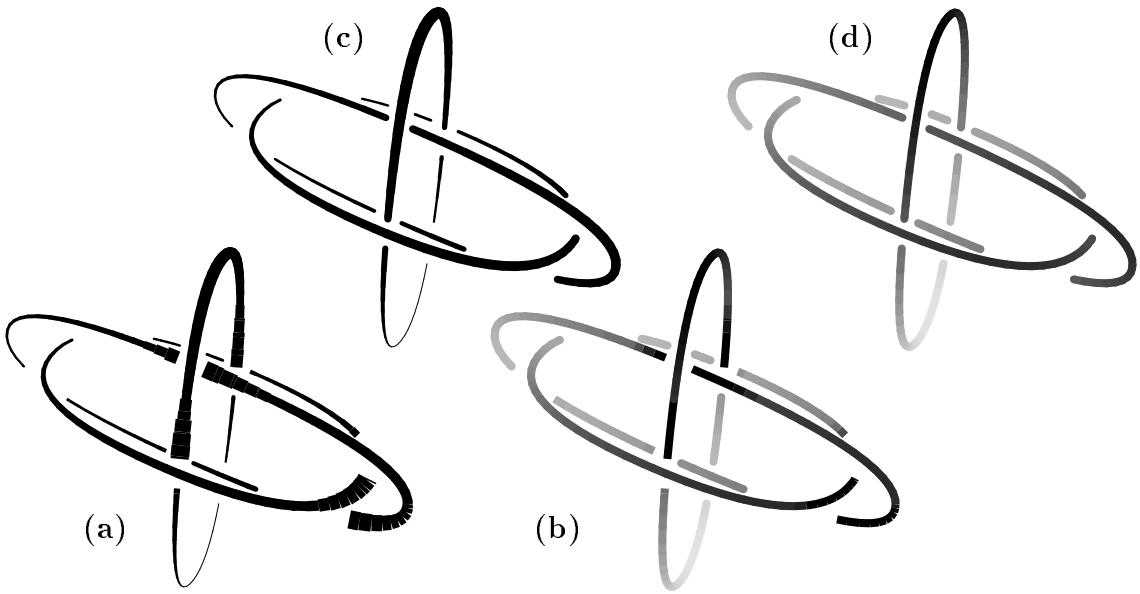

Figure 9: Borromean Rings. Common in illustrations, end of trimmed edges can be thickened by either wider lines (a) or darker color (b). (c) and (d) are the respective versions with no wider or darker ends of trimmed edges.

[Franklin 1987] **W. R. Franklin and V. Akman** A simple and Efficient Haloed Line Algorithm for Hidden Line Elimination. Computer Graphics Forum, Vol. 6, No. 2, pp. 103-110, 1987

[Hilbert 1990] **D. Hilbert and S. Cohn-Vossen.** Geometry and the Imagination. Chelsea Publishing Company, Second Edition, New York, 1990.

[Hornung 1985] **C. Hornung, W. Lellek, P. Pehwald, and W. Strasser.** An Area-Oriented Analytical Visiblity Method for Displaying Parametrically Defined Tensor-Product Surfaces. *Computer Aided Geometric Design,* 2, pp. 197-205, 1985.

[IRIT 1993] **IRIT.** Irit 4.0 User's Manual, Technion, October 1993.

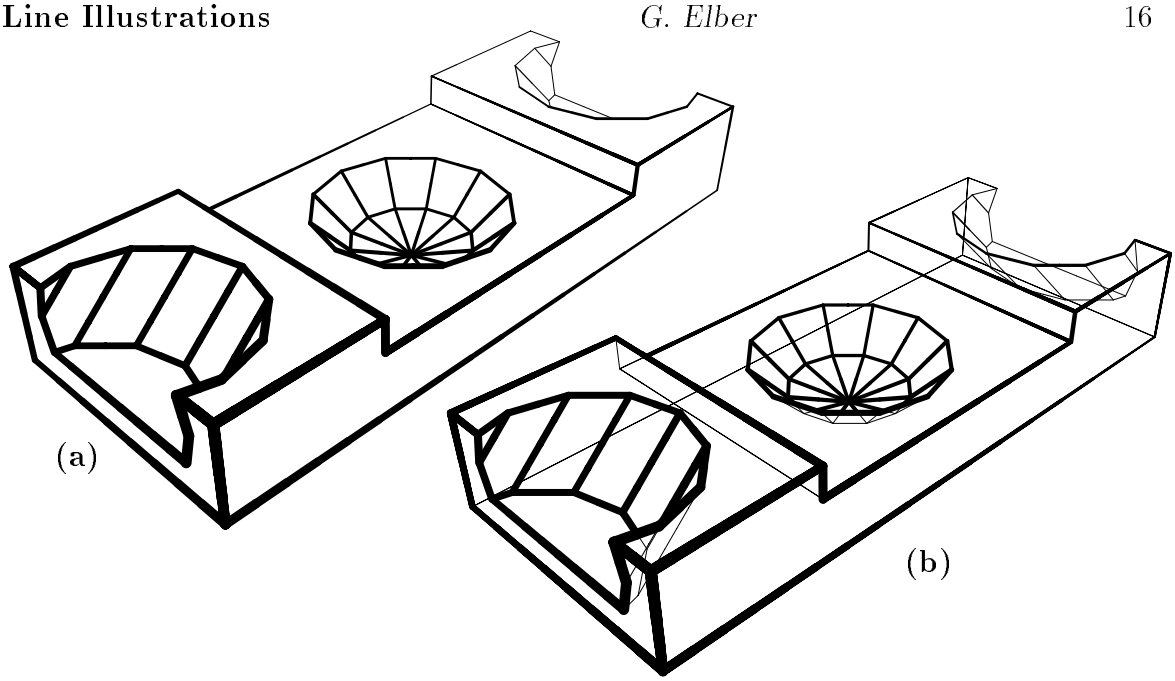

Figure 10: Depth cueing using line width of the visible set of the model (a). In (b), the hidden line set is also drawn, but with thin lines.

[Magnan 1970] **G. A. Magnan.** Using Technical Art: An Industry Guide. *John Wiley and Sons, Inc.*, 1970.

[Preparata 1985] **F. P. Preparata and M. I. Shamos.** Computational Geometry, an Introduction. *Springer-Verlag*, 1985.

[Postscript 1985] **PostScript** Language Reference Manual. *Addison Wesley Publishing Company.*, Reading Massachusetts, 1985.

[Saito 1990] **Saito, T. and Takahashi, T.** Comprehensible Rendering of 3-D shapes. *ACM Computer Graphics*, Vol. 24, No. 4, pp. 197-259, Siggraph 1990.

[Young 1985] **F. M. Young.** Visual Studies: A Foundation for Artists and Designers. *PrenticeHall, Inc.*, 1985.

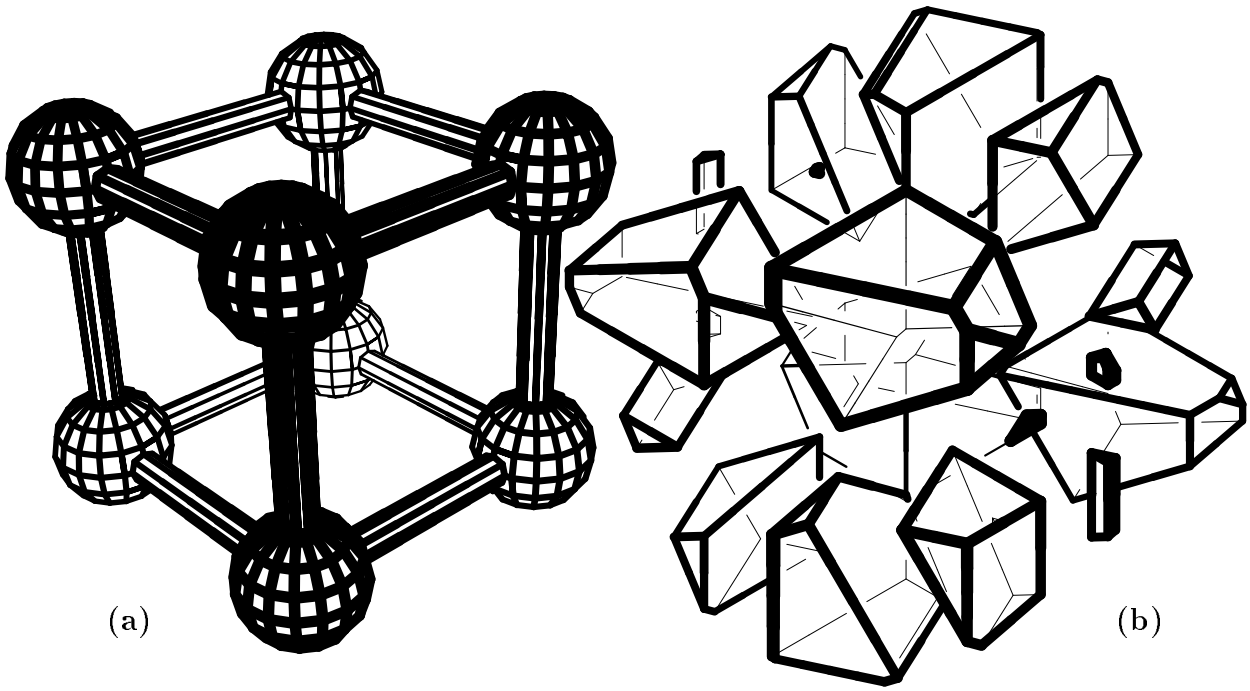

Figure 11: Depth cueing using line width for the visible set of a molecule (a). In (b), the hidden lines of a complex polyhedra are drawn trimmed at intersection points, using thin lines.

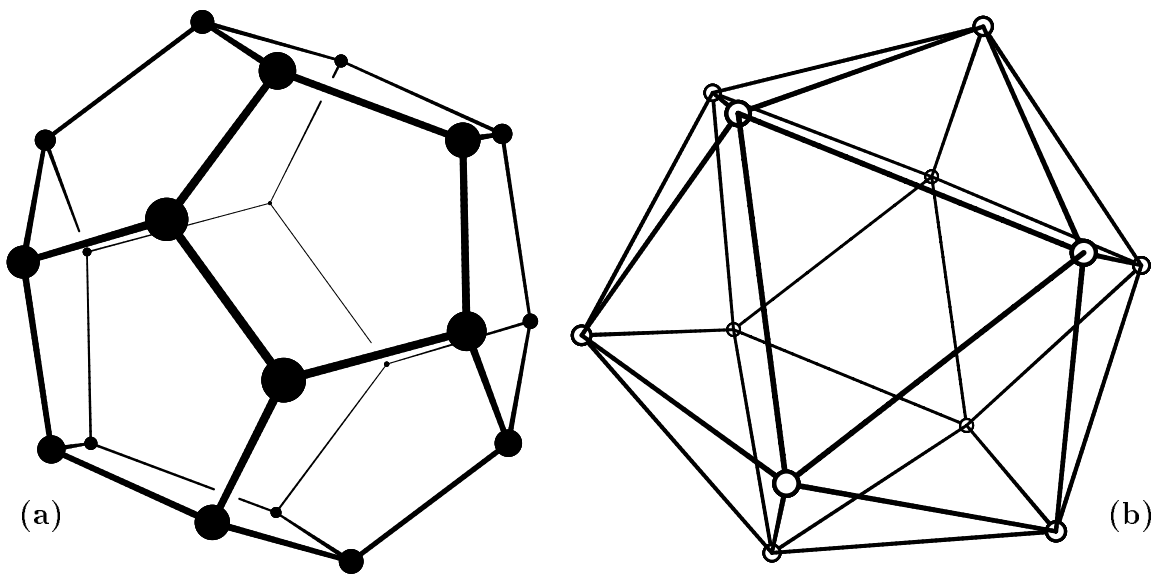

Figure 12: A dodecahedron (a) and an icosahedron (b) drawn using depth cueing and points at the vertices.

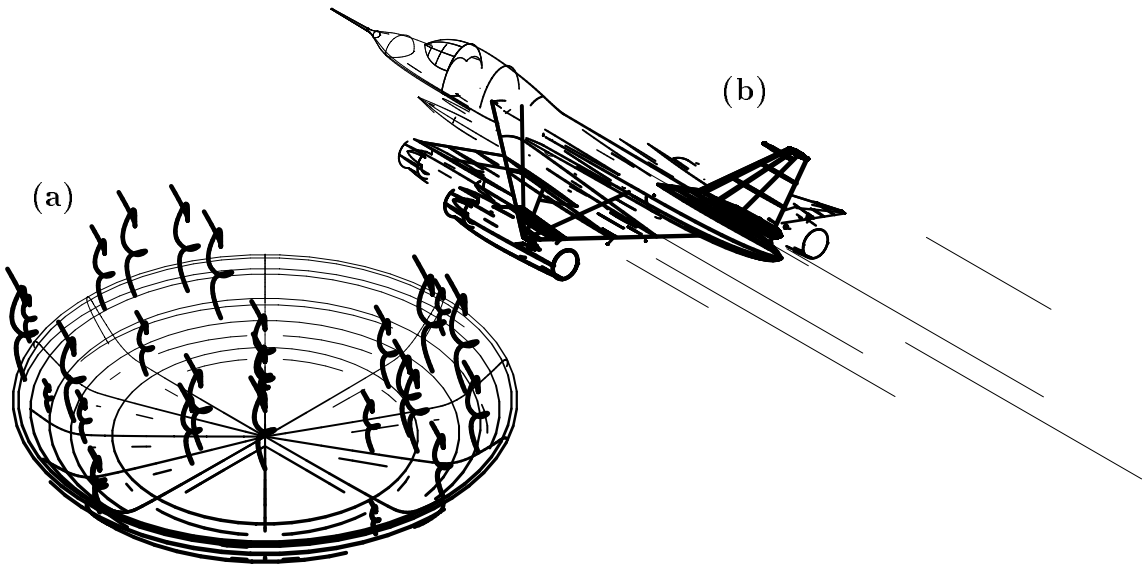

Figure 13: In (a), the sensation of a hot food in the dish is enhanced with the use of newly introduced helical curves. Similarly, in (b), a sense of speed is given to the plane using the introduced straight lines.

Algorithm 4 - `A speed sensation property attribute constructor`

Input:

$\mathcal{M}$, `A Model;`
$\mathcal{G}_i$, `Probability of Instantiation;`
$\vec{V}$, `unit vector in the direction of linear motion;`
L `and` D, `length and distance of generated lines;`
L_p `and` D_p, `length and distance perturbation of generated lines;`

Output:

$\mathcal{D}$, `An object to enhance to speed sensation of` $\mathcal{M}$;

Algorithm:

$\mathcal{D} \Leftarrow \emptyset$;
`For each point` P `in polygonal model` $\mathcal{M}$ `or`
`For each control point` P `in freeform surface model` $\mathcal{M}$
    `if (` $\mathcal{G}_i$ `>` $Random(0.0, 1.0)$ `) then`
    `begin`
        $P_1 \Leftarrow P + \vec{V}(D + D_p * Random(-1.0, 1.0))$;
        $P_2 \Leftarrow P_1 + \vec{V}(L + L_p * Random(-1.0, 1.0))$;
        $\mathcal{D} \Leftarrow \mathcal{D} \bigcup \left\{\overline{P_1 P_2}\right\}$;
    `end`
`end`