Beyond Pixels: Illustration with Vector Graphics

Tobias Isenberg* Angela Brennecke† Mario Costa Sousa* Sheelagh Carpendale*
*Department of Computer Science
University of Calgary, Canada
†Department of Simulation and Graphics
Otto-von-Guericke University Magdeburg, Germany

1 Introduction and Motivation

This report presents a novel vector rendering pipeline that allows us to easily break the pixel barrier and create high-quality illustrations. Recently, most graphic research has been directed towards rendering pixel images that appear realistic. In contrast, we investigate the generation of vector graphic illustrations using non-photorealistic techniques such as line rendering and Gooch shading. By combining vector output from both shading and line rendering of 3D models we create high-quality illustrations that can directly be used in print reproduction. Our approach uses a vector graphic pipeline that tracks multiple attributes of strokes and uses them for stylization. This allows to have multiple layers of line rendering such as different stroke types or visible and hidden parts of strokes, each treated differently according to specific stylization rules.

Using high quality vector graphics (as opposed to pixel renditions) for representing illustration is essential, in particular, in the print reproduction process. Foremost, vector graphics can be reproduced at any desired resolution; they do not suffer from the resolution dependence of pixel images. In addition, only vector graphics can capture fine details accurately while maintaining a reasonable file size. Finally, vector graphics do not need to be half-toned when printed as long as spot colors are used. Even if some layers of the image use, e. g., shading, only those parts of the vector graphic need to be half-toned that actually do not make use of the available spot colors. Thus, we can combine both shading and line layers without compromising print quality.

2 Approach/Method

Our approach for generating linear vector primitives is based on the so-called *G-strokes* concept that captures and processes additional stroke properties along with the stroke geometry and topology [Isenberg and Brennecke 2005]. G-strokes are inspired by G-buffers [Saito and Takahashi 1990] but store properties of strokes rather than pixels. Therefore, they have to meet other constraints such as adaption to geometry and topology changes during stroke processing. In addition, G-strokes maintain vector information throughout the entire pipeline in 3D world coordinates until the strokes are being rendered, thus, forming a vector rendering pipeline. This pipeline is different from typical 3D rendering pipelines in that it can have multiple stages that capture new properties as well as use these properties to manipulate the stroke. A typical vector rendering pipeline comprises at least stages to extract strokes from another data structure (e. g., a 3D model), determine their visibility, and to render them in a specific line style.

In order to treat parts of the stroke set differently, the stroke pipeline cannot be modeled as a linear structure but rather has to take hierarchical form where each subtree represents the generation of one layer. Thus, we use an OPEN INVENTOR scene graph approach to realize the vector pipeline. To separate the treatment of individual layers of stroke data from the main pipeline we implemented a filter mechanism that not only facilitates different stylization but also the treatment of only a subset of the whole stroke data (cf. Fig. 1).

*e-mail: {isenberg | mario | sheelagh}@cpsc.ucalgary.ca
†e-mail: abrennec@isg.cs.uni-magdeburg.de

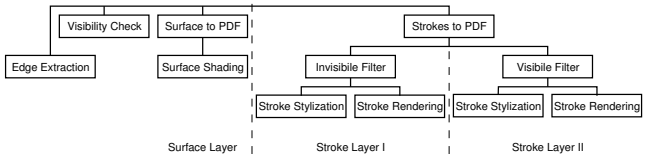


Figure 1: Example vector graphic pipeline modeled as a tree. More stroke layers can be used to visualize other aspects of the model.

For realizing vector output of the stroke pipeline, we render the stroke data using a dedicated render node based on the ClibPDF library. Doing so, we have complete control over the rendering process and we can produce high-quality strokes. For capturing other parts of the scene such as non-photorealistic shading (e. g., Gooch shading, cf. Fig. 2) we employ the GL2PS library which redirects OPENGL calls to a vector graphic format (PS, EPS, PDF). It outputs the rendition as a 2D mesh and allows to capture Gouraud shading. However, since GL2PS derives the visibility information using an object-space BSP tree approach this has limitations in terms of possible triangle counts (more than approx. 50,000 triangles after backface culling did not seem feasible in our tests). On the other hand, it is not necessary to render very dense meshes to achieve a very good NPR shading. Although redirecting OPENGL calls could also be used for outputting the stroke data we decided against it because the resulting file would be based on meshes rather than polygons which are more difficult to process further in common vector graphics suites such as Corel Draw® and Adobe Illustrator®. In addition, the separate treatment of shading and line output has the advantage that the hidden surface removal of GL2PS does not interfere with the desired layered rendering.

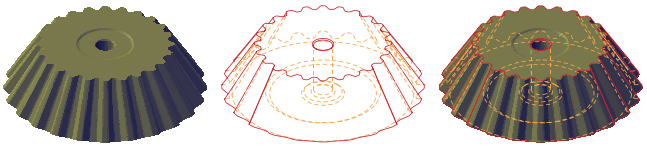


Figure 2: Gooch shading and lines in a vector graphic illustration.

We observed that the proposed pipeline provides great flexibility for combining shading methods with line drawings and to export the images into vector graphic formats. We are currently investigating labeling techniques to further extend the illustration's information/predication. Also, we would like to explore the potential of vector graphics for NPR on the reproduction of traditional illustration media and techniques. This would include, for instance, methods for clustering vector-based stroke sequences to cover scene/object regions, and techniques to reproduce small but important artifacts due to the interaction of the drawing/painting device and correspondent drawing/painting surface (i. e., engraving). We see great potential in our approach since publishing in science and engineering can greatly benefit from an easy generation of high-quality illustrations.

References

ISENBERG, T., AND BRENNECKE, A. 2005. G-Strokes: A Concept for Simplifying Line Stylization. Tech. Rep. 2005-780-11, Department of Computer Science, University of Calgary, Canada, Apr.

SAITO, T., AND TAKAHASHI, T. 1990. Comprehensible Rendering of 3-D Shapes. *ACM SIGGRAPH Computer Graphics 24*, 3 (Aug.), 197–206.

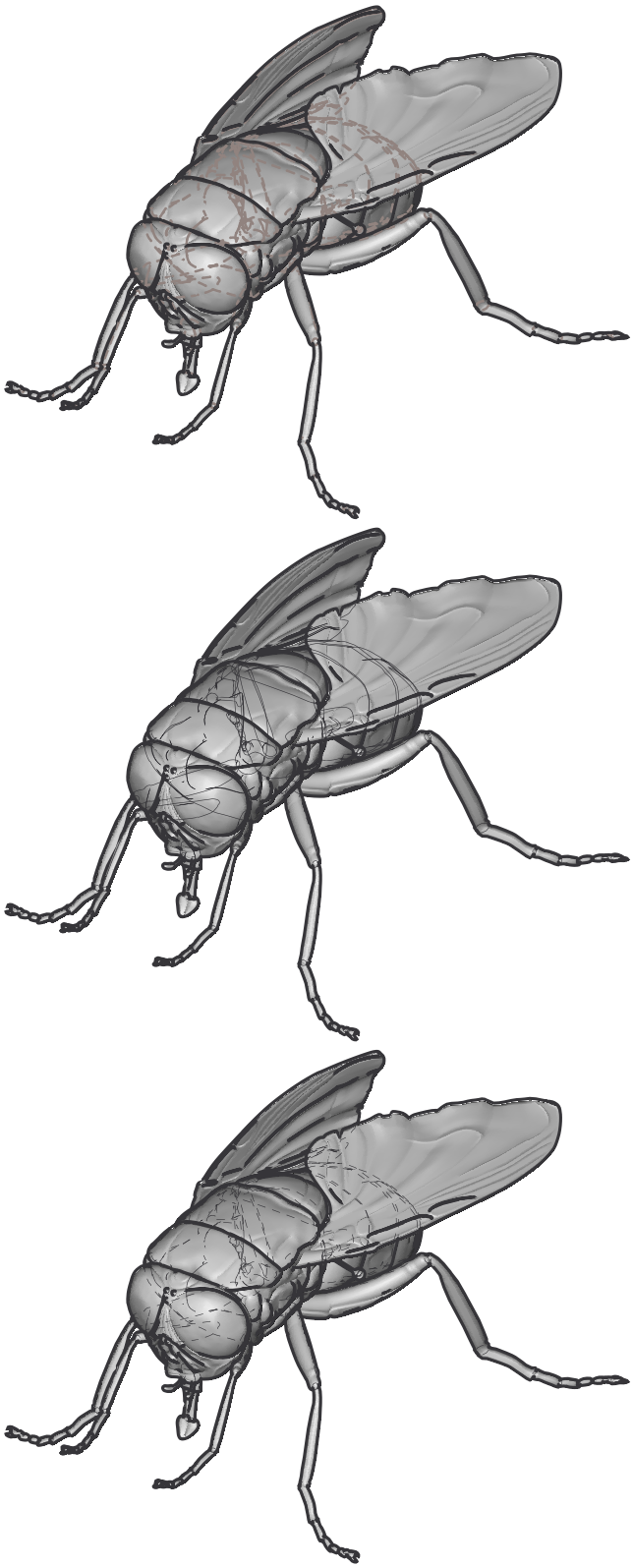


Figure 3: Example of biologic illustrations: a high-resolution model of a fly is visualized with the hidden lines rendered differently; Gouraud shading is used for the surface to show fine details.

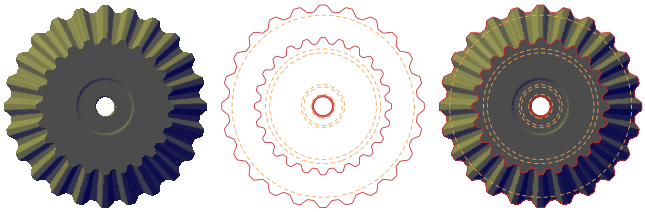


Figure 4: Different view of Figure 2.

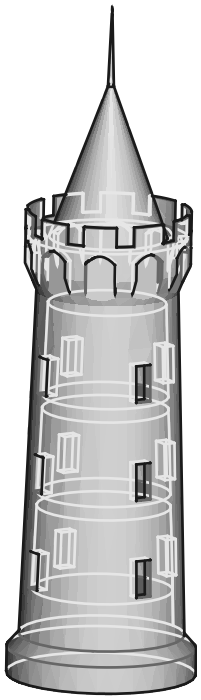
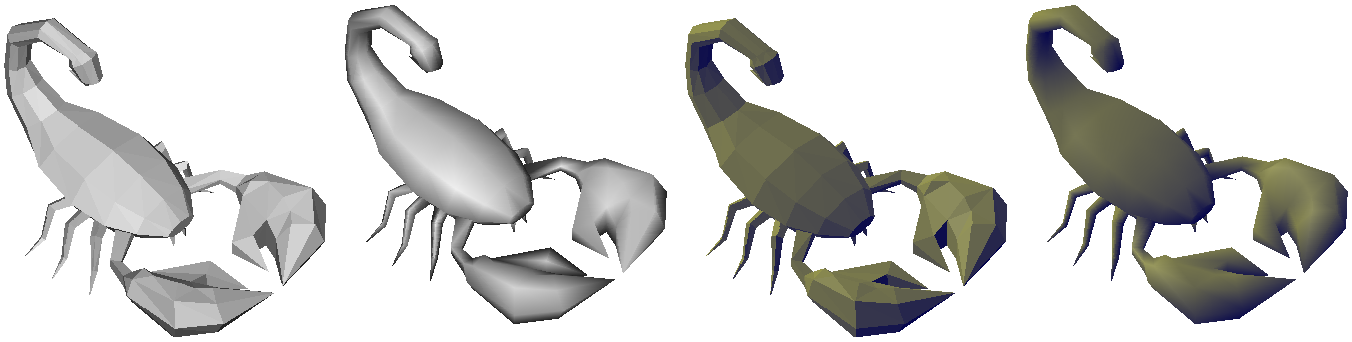
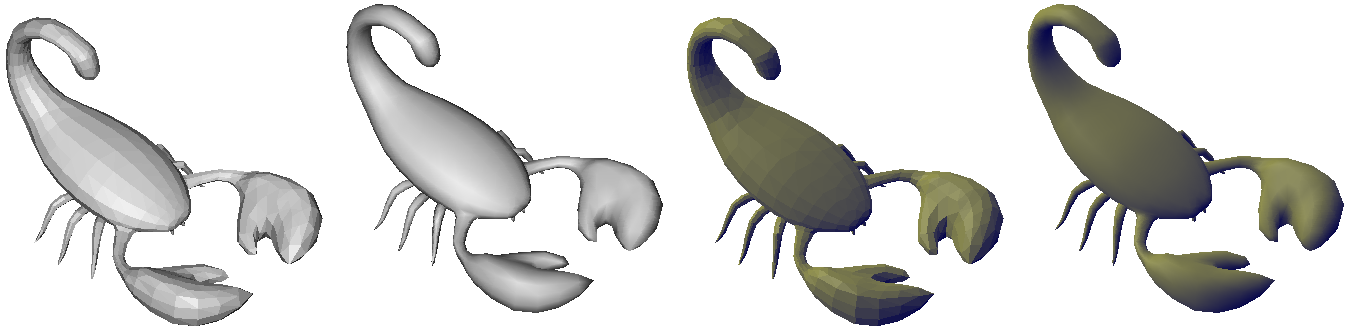


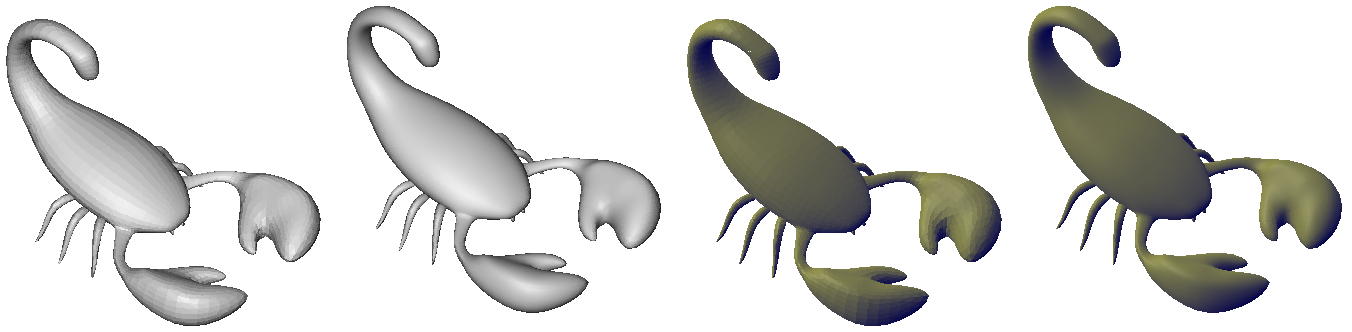
Figure 5: Illustration of an architectural object. Hidden lines, visible silhouettes, and visible feature lines are visualized using different shades of gray. In this example, different levels of subdivision were used for the line layer and the surface layer. Note that the inner structure can well be understood.



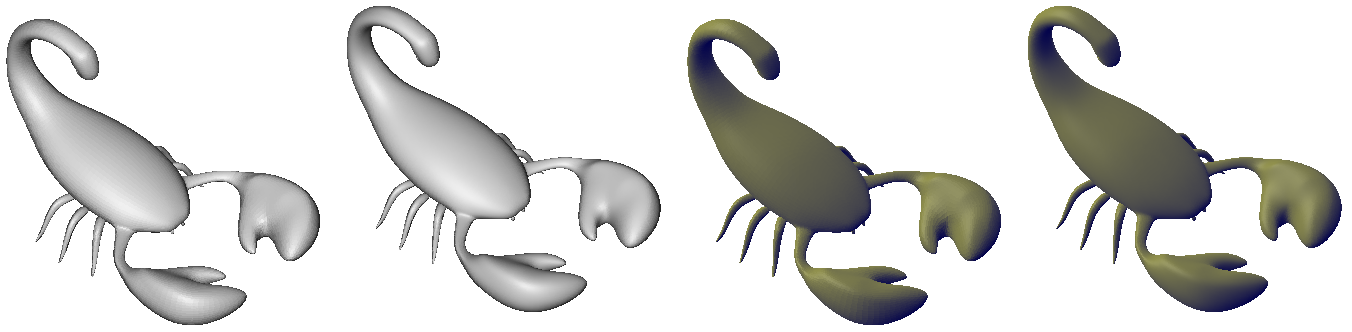
(a) 522 triangles, flat shading, 12 kB (compressed PDF).
(b) 522 triangles, Gouraud shading, 12 kB (compressed PDF).
(c) 522 triangles, flat shading, 12 kB (compressed PDF).
(d) 522 triangles, Gouraud shading, 12 kB (compressed PDF).



(e) 2,088 triangles, flat shading, 37 kB (compressed PDF).
(f) 2,088 triangles, Gouraud shading, 35 kB (compressed PDF).
(g) 2,088 triangles, flat shading, 37 kB (compressed PDF).
(h) 2,088 triangles, Gouraud shading, 35 kB (compressed PDF).



(i) 8,352 triangles, flat shading, 146 kB (compressed PDF).
(j) 8,352 triangles, Gouraud shading, 135 kB (compressed PDF).
(k) 8,352 triangles, flat shading, 144 kB (compressed PDF).
(l) 8,352 triangles, Gouraud shading, 134 kB (compressed PDF).



(m) 33,408 triangles, flat shading, 560 kB (compressed PDF).
(n) 33,408 triangles, Gouraud shading, 491 kB (compressed PDF).
(o) 33,408 triangles, flat shading, 551 kB (compressed PDF).
(p) 33,408 triangles, Gouraud shading, 485 kB (compressed PDF).

Figure 6: Effects of Gouraud shading vs. flat shading and the Gooch illumination model (right) vs. a gray shading (left) in the produced vector graphics shown for different levels of Loop subdivision. Note that for good results it is not necessary to have a high polygon count.