Eurographics Symposium on Rendering 2008
Steve Marschner and Michael Wimmer
(Guest Editors)

Volume 27 (2008), Number 4

Stylized Vector Art from 3D Models with Region Support

Elmar Eisemann[1,2], Holger Winnemöller[1], John C. Hart[1,3], and David Salesin[1,4]

[1]Adobe Systems, Inc. [2]ARTIS-INRIA / Grenoble University [3]University of Illinois Urbana-Champaign [4]University of Washington

Abstract

We describe a rendering system that converts a 3D meshed model into the stylized 2D filled-region vector-art commonly found in clip-art libraries. To properly define filled regions, we analyze and combine accurate but jagged face-normal contours with smooth but inaccurate interpolated vertex normal contours, and construct a new smooth shadow contour that properly surrounds the actual jagged shadow contour. We decompose region definition into geometric and topological components, using machine precision for geometry processing and raster-precision to accelerate topological queries. We extend programmable stylization to simplify, smooth and stylize filled regions. The result renders 10*K-face meshes into custom clip-art in seconds.*

1. Introduction

Compared to the memory-consumptive fixed resolution of raster graphics, vector graphics offer a compact, scalable, model-based alternative, well suited for output on heterogeneous resolution devices, such as printers, projectors, screens and mobile devices [HMK02, ICS05, QTG*06].

The strokes and fills of vector graphics more closely match the traditional tools of art and design, and vector graphics have become the preferred medium for the stylized illustrations that help visually communicate ideas and concepts in the form of "clip-art" libraries. But even though their digital representation (e.g. layered filled 2D polygons) is simple, constructing a vector illustration requires technical skills beyond those of the typical consumer, creating a demand for a rendering system that converts a 3D meshed polygonal surface model into a customized and stylized illustration with little to no user-input, as demonstrated in Fig. 1.

Two commercial applications, LiveArt (v. 1.2) and Swift3D (v. 4.5), provide tools for rendering meshed objects as filled-region vector art by vectorizing image regions (including cast shadows) from an intermediate rasterized image. This approach is common to cartoon-shaded NPR methods [LMHB00, SMC04, MF06, BTM06, TiABI07], but the fixed-resolution sampling of the region geometry can lead to aliasing artifacts, as shown in Fig. 2.

Filled-region vector art can be created from images

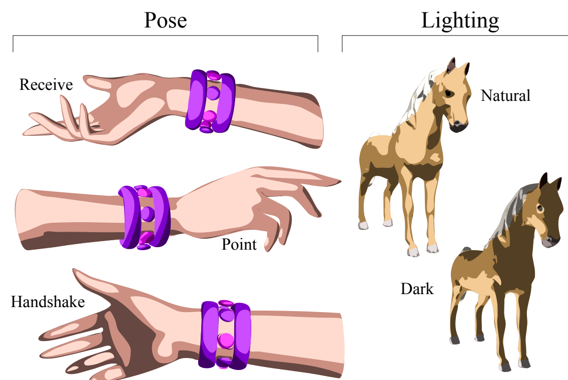

Figure 1: *Our vector-art renderer converts 3D meshed objects into custom 2D filled-region illustrations. 3D objects become re-usable illustration assets that can be styled, posed and lit for consistency and to better communicate a desired concept or mood.*

[KHCC05] and implicit surfaces [SEH08] but their application on meshed objects would require a similarly problematic intermediate representation. Contour-based NPR, e.g. pen-and-ink rendering [WS94, WS96], directly converts meshed objects into line drawings, but avoids defining and stylizing region fills in favor of stroke-based hatching or stippling that rely on simpler, parameterized and triangulated regions. Sec. 2 further compares prior approaches.

This paper describes a new method, diagramed in Fig. 3,

© 2008 The Author(s)
Journal compilation © 2008 The Eurographics Association and Blackwell Publishing Ltd.
Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

Figure 2: *Using rasterization to define filled regions can yield aliasing artifacts. as shown in Goofy's lower lip rendering with Swift3D (left) and the flamingo's beak/head intersection rendered by LiveArt.*

that directly renders a 3D meshed surface model into stylized filled-region vector art consisting of layered 2D polygons. The most difficult part of this process is the identification, processing and stylization of the regions defined by using the meshed object's contours as a boundary representation. This rendering approach requires several new contributions.

- Mesh contours defined by face normals are accurate but jagged, whereas those defined by interpolated vertex normals are smoother but do not accurately delineate regions. Sec. 3 defines smooth contours that properly surround regions with the novel combination of a jagged (but visually smooth) visual contour, smooth shading contours, cast jagged (but visually smooth) shadow volumes and a new adjusted smooth shadow contour that properly surrounds the accurate but jagged shadow contour.
- We analyze and decompose the vector-art rendering process into geometric and topological components. Sec. 4 shows how this novel decomposition enables the use of standard GPU techniques to accelerate only the discrete topological queries of contour visibility and region classification, whereas region shapes are defined geometrically at machine precision, to ensure the fidelity of the vector-art rendering process.
- We extend an existing programmable contour stylization method [GTDS04] with new features that capitalize on the region definitions, including styles that offset the region from the contour, small region elision (e.g. highlight specks), region-sensitive contour smoothing and fill processing that takes advantage of recent features available in the Scalable Vector Graphics format [SVG03], as described in Sec. 5.

Sec. 6 demonstrates a prototype filled-region vector-art rendering system based on these contributions that runs in seconds on 10K-face manifold models over several styles.

2. Previous Work

Our renderer converts meshes into filled-region illustrations. Many NPR approaches focus only on extraction and stylization of contours and avoid the need to connect them to produce and stylize regions [MKG*97, Elb98, KDMF03, DFRS03, SH05, BKR*05, OZ06, JD07].

Winkenbach and Salesin constructed stroke-textured vector-art regions from meshed models for pen-and-ink illustration of architectural models [WS94] and parametric surfaces [WS96]. These systems used a BSP tree, exact rational arithmetic and shadow volumes to construct a precise planar map of clipped, self-shadowed visible regions. Their regions consisted of simple polygons (triangles) whose texture coordinates allowed them to be filled with stroke-based hatching and texture. Our rendering system similarly constructs a self-shadowed planar map, but we generate complex polygons to support the kind of shape-dependent stylization found in many clip-art styles and to reduce the complexity of the representation. Furthermore, architectural meshed models [WS94] contain simple contours and parametric surface tesselations [WS96] can be subdivided to yield precise contours whereas general meshes require the careful analysis of Sec. 3 to properly define regions.

Stroila *et al.* [SEH08] used particle systems constrained to implicit surfaces to create layered filled-region vector-art illustrations. This paper expands that work to include meshed objects as input, which require special attention to yield smooth appearing contours that bound regions. We also explore a wider variety of clip-art styles in Sec. 5.

Our region-based stylization approach is built on the programmable contour stylization approach of Grabli et al. [GTDS04], which describes an NPR-adapted shading language for stylizing outline renderings of meshed objects. Our programmable stylization relies on hooks in the intermediate representation to access each element's 2D image-plane context and 3D mesh neighbors, and similar information was provided by the BSP trees used by Winkenbach and Salesin [WS94, WS96].

Numerous stylization approaches exist for images [ST90], including watercoloring [CAS*97, LD04, BNTS07] and the conversion into photorealistic color-gradient vector art [PB06,SLWS07]. Kang *et al.* [KHCC05] outlined regions in an input image with multires B-splines that served as tangent vector fields for region stylization. Image segmentation often requires user input, so DeCarlo *et al.* [DS02] used eye tracking to control the level of detail of image region abstraction, and Kolliopoulos *et al.* [KWH06] presented an automatic segmentation and generalization of image segment stylization. Our mesh-based region stylization expands on the image-based region stylization of Wen *et al.* [WQL*06]. They focused on a single stylization that reduced foreground regions biased by illumination so the revealed whitespace served as a highlight, and shifted colors to better indicate foreground/background separation.

3. Region-Robust Contour Extraction

Our conversion of a 3D meshed surface into a 2D layered polygon depiction begins by extracting the contours whose image projection define the 2D polygonal layers. In our

© 2008 The Author(s)
Journal compilation © 2008 The Eurographics Association and Blackwell Publishing Ltd.

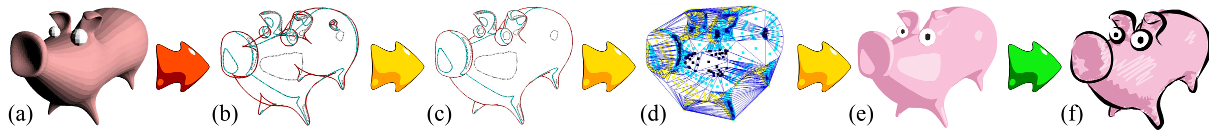

Figure 3: *We process a 3D meshed surface (a) into a 2D vector clip-art depiction (f). We first extract contours (b) and determine their visibility (c). We then construct a planar map and classify the regions enclosed by contours (d) to yield a base vector depiction (e) for stylization (f).*

prototype implementation, these contours include the visual contour ($N \cdot V = 0$), Lambertian isophotes ($N \cdot L = k$), the shadow boundary ($N \cdot L = 0$), reflection lines (e.g. $N \cdot H = k$), suggestive contours [DFRS03] and apparent ridges [JD07].

Our goal is to extract contours with enough precision to use as boundaries for the filled regions of layered vector art. Otherwise portions of the region (and other feature curves in these areas) can escape the contour that defines its boundary and will confuse region classification and stylization, as demonstrated in Figs. 4 and 5. The contours extracted from smooth surfaces are precise enough to use as a boundary representation for filled regions [WS96, SEH08], but the contours of typical meshes are not as cleanly defined [IFH*03].

The visual contour that lies along the edges between the front and back facing elements of a mesh can be notoriously jagged, including short switchback loops. Winkenbach and Salesin [WS96] extracted a precise visible contour from a tesselated smooth parametric surface by refining the tesselation near the contour. Hertzmann and Zorin [HZ00] extracted a less-jagged visible contour from a meshed surface using interpolated per-vertex normals. These interpolated contours are much smoother, but do not define geometrically accurate regions. Sec. 3.1 analyzes this issue for the visual contour, concluding that the face-normal contour should be used as the silhouette to obtain well-defined regions. Sec. 3.2 examines the shadow contour and devises an adjustment that yields a smooth shadow contour guaranteed to contain the actual (jagged) mesh shadow.

3.1. Smooth Visual Contours Considered Harmful

Interpolated vertex normals can generate a smooth visual contour, but the smoothed silhouette (the portion of the projected visual contour generator that separates the surface from the background) no longer contains the projected interior of the mesh. The interpolated-vertex-normal visual contour rarely passes through vertices, but the vertices form the outer bounds of the silhouette of a meshed surface. When plotted on a visible-surface mesh rendering, the vertex-interpolated visual contour slaloms around these extremal silhouette vertices, alternating between visible and occluded as it weaves its way around the perimeter. The upper inset of Fig. 4 illustrates an example where the smoothed visual contour (in blue) fails to contain the surface projection.

If portions of the surface can project outside of the silhouette, then other feature curves defined on the surface can project outside of the silhouette too. The lower inset of Fig. 4 demonstrates an example where the red face-normal visual contour (a shadow contour) escapes the silhouette defined by the blue smoothed vertex-interpolated visual contour generator (the situation can similarly occur with a smoothed feature contour).

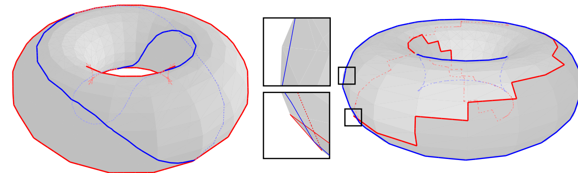

Figure 4: *Two views of the same torus. The visual contour of the left view (red) is computed with face normals and appears jagged in the right view. The visual contour of the right view (blue) is computed using interpolated vertex normals and appears smooth in the left view. Note that each visual contour appears smooth in its particular view. The insets (center) demonstrate that the blue vertex-interpolated visual contour does not properly surround the geometry, and contours (e.g. the red visual contour) can escape it.*

The intersection and trimming of feature curves that escape the silhouette would cause problems in region classification. Furthermore, the visibility near these area spills becomes ill-defined as the feature contour's mesh visibility would change at a different location than predicted by a smoothed vertex-interpolated visual contour generator.

Fig. 5 shows several other potential problems. Elements on back-faces can extend over the silhouette (eye feature line in (a)). The silhouette can show very complex behavior that cannot occur, as such, on real, smooth manifolds (cusp in (b)). This makes visibility computation particularly challenging. Inset (c) shows how deformed the contour becomes when triangles have strongly differing vertex normals. The curve is *dragged* off its original position and even behind the shadow regions. Inset (d) illustrates similar problems for color contours (black, stippled meaning invisible) that do not match up with any silhouettes (red outer, yellow inner). For example, in the center of the figure, the vertex silhouette only creates a small detached region on the back of the horn.

We thus rely on the visual contour generator defined by

© 2008 The Author(s)
Journal compilation © 2008 The Eurographics Association and Blackwell Publishing Ltd.

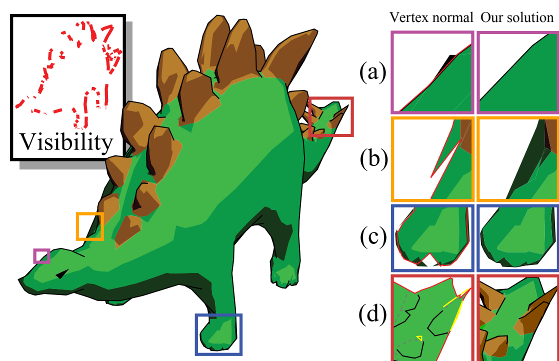

Figure 5: REGION ARTIFACTS *can result from using interpolated vertex normals for the visual contour. Determination of visibility is challenging (see inset, based on ray-tracing). Other problems include: (a) A feature element (the eye) on the back face extends over the silhouette. (b) A cusp escapes the view tangent plane. (c) Badly shaped triangles lead to visibility problems. (d) Incompatibility between the visual contour and color contours.*

the subset of mesh edges that lie between front-facing and back-facing mesh polygons. Its jagged artifacts and switchbacks occur in a view-tangential direction which hides them from the viewer, but can cause problems in stylization, fixed by the programmable chaining described in Sec. 5.

3.2. A Proper Smoothed Shadow Contour

Two kinds of contours are used to depict shadows. The *shadow boundary* is a visual contour generator from the view of a light source, whereas the *cast shadow contour* is a projection of the shadow boundary along the light direction onto further portions of the surface.

We extract cast shadow contours by constructing a shadow volume and intersecting it with the meshed surface. We first create a temporary copy of the mesh whose faces are then subdivided by the shadow volume. We output the cast shadow contour as a sequence of vertices using face indices and barycentric coordinates to indicate their position on the original mesh.

We extract a smooth shadow boundary using vertex interpolated normals. This smoothed shadow boundary can disagree with the shadow cast by the actual geometry, as illustrated in Fig. 6 (left), which can result in shading discontinuities along the shadow boundary. To overcome this issue, we adjust the vertex normals indirectly by manipulating the weighting of face normals. (Similar adjustments to the shading calculation have been proposed for more direct artistic control of rasterized cartoon highlights and shadows [iAWB06, TiABI07]). For the sake of computing a smooth shadow boundary, we set the vertex normal to the

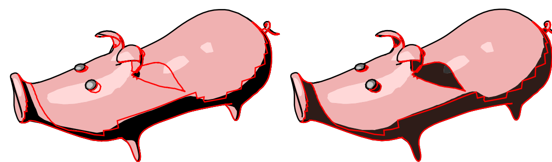

Figure 6: *Left: The shadow contour where interpolated vertex normals are perpendicular to a light direction (dark regions) can disagree with the actual shadow cast by the geometry (red contours). Right: We create a new smooth shadow contour regions using adjusted normal interpolation (dark) to ensure that it strictly contains the actual cast geometric shadows (red contours).*

value of its adjacent face normal that forms the greatest angle with the light vector (minimizes $N \cdot L$). This ensures that the shadow boundary passes through the bright-side faces along the edge-boundary between faces pointing toward the light and those pointing away.

4. Fast Topological Region Processing

Once all of the contours have been extracted, we compute their mutual intersection to divide them into (polyline) contour segments. These contour segments collectively form well-defined region boundaries in the projection, without dangling segments. We compute both object space intersections (between shading and shadow contours), as well as image-space visibility intersections (between those contours and the visual contour).

We then find the visible subset of these contour segments, and use them as a boundary representation to construct a planar map of visible regions which must then be classified according to the kind of shading they represent. Both visibility and classification are topological operations based on an ordering or labeling, which is discrete and well-defined for positions well inside the interior of elements. This realization permits the use of GPU techniques to accelerate the topological queries of contour visibility and region classification for our vector art. For these queries, we use an efficient fixed-precision rasterization as a conservative test, and revert to slow, robust ray-casting for ambiguous cases. This approach differs from previous, image-based approaches (e.g. index [NM00], z- or normal-buffers [Her99]) that process both geometry and topology at image resolution.

4.1. Contour Visibility via Occlusion Query

Since we mutually intersect all contours, visibility does not change along any contour segment. The visible subset of these define region boundaries. Ray casting can determine contour visibility [SEH08], but would be too slow for large meshes. A voting estimate is faster [GTDS04], but its inaccuracy, especially near cusps, would interfere with dependably forming closed region boundaries.

© 2008 The Author(s)
Journal compilation © 2008 The Eurographics Association and Blackwell Publishing Ltd.

We instead use rasterization to answer this topological query more rapidly while maintaining accuracy. We first cull any contour segments lying on back-facing polygons. We also cull any visual contour segments lying on crease edges if the mesh is a closed, orientable, and watertight manifold. We then rasterize the polylines of the remaining contour segments along with the mesh faces. OpenGL's occlusion query determines which contour segments passed the depth test.

We use OpenGL's polygon offset to resolve polyline/polygon depth buffer interference, but occluded contour segments near the visual contour can still poke through the mesh. We thus only rasterize the centers of contour segments, avoiding their error-prone ends. When a contour segment is too short, we resort to precise but costly ray casting. We ensure watertight regions with a second test using the previously found visible segments. Because the rasterization query can fail where several visible silhouettes project to the same pixel, we mark a small pixel neighborhood around the previously determined visible silhouettes and any visual contour intersecting these potentially ambiguous pixels (detected efficiently via stencil tests) is handled with ray casting. On a model with tens of thousands of faces, contour visibility takes a couple of seconds (see Table 1) instead of a full minute needed for ray casting.

4.2. Region Classification via a Fragment Shader

We use the visible contour segments as a boundary to define regions of homogeneous shading. These regions are organized by a winged-edge structure called a *planar map*, constructed in our prototype by the CGAL Arrangement Package [FHHN99]. The specific kind/combination of shading represented by each region must next be established by a classification step.

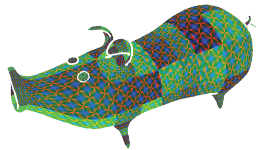

Figure 7: *Rasterization accelerates region classification. Green = triangle ID. Red/blue = barycentric coordinates.*

We accelerate the topological query of region classification by rasterizing the surface mesh using a fragment shader that encodes the face index and barycentric coordinates of the fragment's surface point, as shown in Fig. 7. We classify each planar map region by picking one of its interior points, using the rasterization to find its corresponding object-space coordinates, and computing that mesh point's illumination. We similarly classify regions as shadowed/unshadowed with a second rendering using shadow volumes.

To avoid misclassification errors due to numerical imprecision, we remove pixels that touch the region's boundary from the query set, similar to Sec. 4.1. If a region is too small, it may not correspond to any pixels in the region rasterization, in which case we must cast a ray through it toward the mesh to determine its shading classification.

Our prototype implementation finds for each region a sequence of interior points by tesselating the region, as shown in Fig. 3 (d), and returning the triangle centroids in order of decreasing area.

5. Stylization

The contour extraction and region processing results in a planar map that can be considered a bare-bones illustration of the 3D meshed model, e.g. Fig. 3 (e).The stylization of a rendering defines its *look* and achieves a variety of visual communication objectives, e.g. to elicit an emotional response, to relate otherwise disparate objects, or to generalize a specific instance through iconography [GG01, SS02].

To provide such flexibility in our system, we strive to separate style from content via the definition and use of style *templates* that encapsulate an artist's design for use by novices. We expand programmable contour stylization [GTDS04] to programmable vector-based region stylization. Fig. 11 demonstrates this approach by applying several style templates to 3D models of varying complexity.

Unlike line stylization (e.g. [MKG*97, Elb98, IHS02, HOCS02, KDMF03, DFRS03, GTDS04, SH05, BKR*05, OZ06, JD07, GVH07]) and raster-based region stylization (e.g. [ST90, Mei96, CAS*97, LMHB00, KMM*02, LD04, BNTS07]), there exist relatively little previous work on stylization of vector regions [GG01, SS02]. Vector graphic fills have traditionally been limited to solid fills or simple gradients, and the stylization of vector graphics has focused on either contour-stylization of their outlines [IHS02, HOCS02, GTDS04, GVH07] or particle-sampled stippling, hatching, and half-toning of their interiors [WS96, HZ00, PHWF01, Sec02, FMS02, WB03, PFS03].

Whereas previous region shading approaches were limited to raster-based NPR (e.g. toon shaders), or very simple vector gradient fills, recent vector standards such as SVG or Flash offer a much richer palette of shading effects. The SVG specification [SVG03] contains a host of functionality to enable stylization of strokes, fills, and gradients. Filters, ranging from Gaussian blurs to noise functions, can be applied to modify the appearance of vector elements. Complex layering, masking & clipping, and compositing modes are analogous to raster-based multi-pass techniques, stenciling, and blending (respectively). In addition, SVG is fully scriptable, allowing some appearance changes to be packaged within the final output. This warrants the investigation of new region shading approaches to vector art. Rendering to vector art using complex gradient meshes [PB06, SLWS07] further exemplifies this possibility.

5.1. Region-Sensitive Contour Stylization

Once contours and regions are computed and organized into the planar map representation, they can be queried in both

© 2008 The Author(s)
Journal compilation © 2008 The Eurographics Association and Blackwell Publishing Ltd.

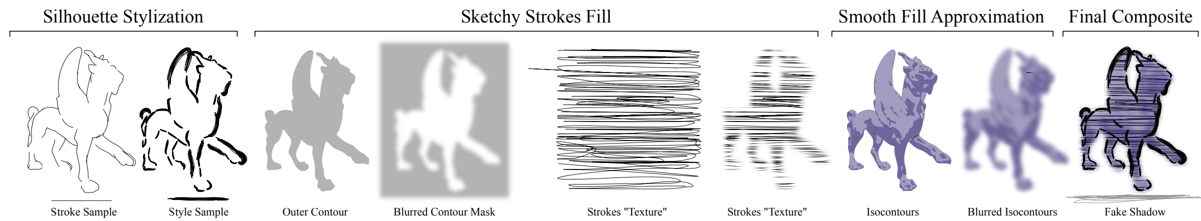

Figure 8: ANATOMY OF A STYLE - *we create a complex vector style by combining and layering various feature curves: (**Silhouette Stylization**) We map a user-defined scribble stroke onto the chained silhouette after removing some short segments. (**Sketchy Stroke Fill**) The outer contour of the object is blurred and used to mask a random scribble "texture", to achieve the effect of loose strokes with fading ends. (**Smooth Fill Approximation**) We only extract two light iso-contours, but increase the apparent complexity by blurring the result with an SVG filter. (**Final Composite**) All style elements are layered and a fake shadow is added, consisting of a fixed scribble, scaled to the width of the contour.*

the 2D projected image space and the 3D object space, via back-references into the original meshed model. This allows styles to refer to geometric data from the original mesh and can warp shapes directly in 3D to take perspective into account.

Chaining is the procedural processing of contour segments in preparation for stylization [GTDS04]. Chaining is especially vital for processing the visual contour, as its planar map projection frequently consists of segments that are disjoint in the original 3D mesh. Additionally, varying segment sizes, cusps, and loops can confound conversion of the visual contour into stylizable outlines [Her99, NM00, IFH*03]. We augment a previous programmable chaining system [GTDS04] to include queries and operations on regions, e.g. to allow for region size to control region contour thickness, or to remove small highlight regions.

Once contours are properly chained, we can simplify and smooth them with an angle-thresholded Bézier curve-fitting algorithm based on a Newton-Rhapson error minimization. We leave it to the style programmer to share contour fitted curves between adjacent regions to avoid accidental misalignment, but we do not force contour sharing, as some clip-art styles intentionally misalign regions, such as Fig. 11 (3rd row). Our styles further embellish these fitted Bézier segments by adapting a skeletal strokes technique [HLW93] to iteratively map strokes onto strokes [HOCS02], as illustrated in Fig. 9 and demonstrated in Figs. 8 and 11.

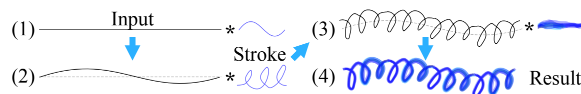

Figure 9: *Iterative mapping of user-strokes to achieve geometric stylization.*

5.2. Region Stylization Development

Stylization within our system relies on two distinct steps. First, a style template has to be programmed manually by an advanced user (described below). Second, a novice can apply a style template either fully automatically, or while adjusting some high-level, style-specific parameters exposed by the style developer, such as maximum line thickness in Fig. 11 (2nd row) or ambient color (4th row). Fig. 11 was created in a batch process with constant parameters after setting pose and lighting for each column.

To develop a style, our system offers a stylization framework that is embedded in several C++ classes with many default (e.g. chaining) and convenience methods (e.g. collators and iterators), allowing for quick prototyping, yet providing enough low-level control to implement arbitrarily complex styles. The styles presented in this paper take as input a planar map arrangement along with material definitions from the 3D object file, and stylization parameters including: fitting errors, pen strokes, and procedural chaining and region stylization functions. Typically, a style loops over all faces of a given face-type (*contour, albedo, specular, ...*) stored in the arrangement, and collates their half-edges into complex curves (contours + holes). More advanced styles can create custom regions by defining chaining rules within our chaining system, which is similar to Grabli *et al.* [GTDS04]. In most cases, different stylistic effects are applied to each face type. For example, Fig. 11 (3rd row) converts all regions into strokes by looping a single stroke at a user-defined angle, given some constraints on randomness, *loopiness*, and coverage. The base regions use a different angle than the shading regions (see pear). Note, that the base regions correctly fill the entire shape without leaving holes underneath the shading regions, allowing for diverse layering and compositing effects. In Fig. 11 (4th row) base regions are ignored, and small shading regions are eliminated. The remaining regions are converted from polygonal form to Bézier segments with a large, region-size dependent fitting error. This produces very vague shading regions, while keeping smaller regions well-behaved. Silhouette contours are chained and fitted with pen or brush strokes, as in previous works. The geometric primitives, along with attributes such as color, thickness, blur, etc. are then emitted by an SVG output class.

© 2008 The Author(s)
Journal compilation © 2008 The Eurographics Association and Blackwell Publishing Ltd.

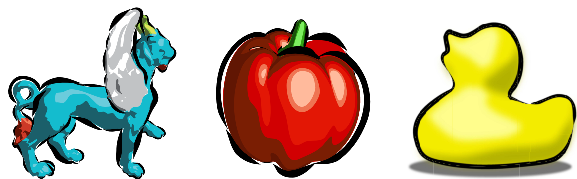

Figure 10: *(Left) The "feline" is rendered with regions defined by the visual contour, a shadow boundary and one Lambertian isophote contour. (Center) The pepper model is complex but the contours are simple. It uses two specular and diffuse contours for the pepper itself and one for the stem, respectively. (Right) The duck demonstrates advanced region shading by using blurred fills for its interior regions and shadow.*

Fig. 8 demonstrates a more complex, region-based style template, including SVG masking and blurring operations.

6. Results and Conclusion

Our system enables a typical consumer, unskilled in illustration, to take a meshed model (created or downloaded) and apply a style template to convert the model into stylized vector form, ready for insertion in a document or presentation, in a matter of seconds. As exhibited in Fig. 1, this system allows the user to re-pose and re-illuminate an object thereby improving its depiction as a vector illustration. Fig. 10 shows several examples of filled-region vector art rendered with our system, while Fig. 11 demonstrates the separation of style from content, by applying four different styles to three different meshed models.

Suggestive contours are also supported in our system as evident in the accompanying video. They are most effective only on highly tesselated objects, and when shading is unavailable (e.g. line-art only). This is somewhat reflected in the work of Lee et. al. [LMLH07] that derives contours based on shading.

The ability to manipulate light direction during the construction of clip-art enables greater control of its appearance, as evident in Figs. 1 and 12. Though clip-art libraries usually contain depictions of simple objects, our vector art renderer can be applied to more complex meshed models such as the pepper in Fig. 10 or the XYZ-dragon in Fig. 13.

Our prototype is designed as a proof-of-concept, to demonstrate that contour extraction can be made robust enough for filled-region definition and processing. Table 1 indicates the performance of this prototype, showing that vector rendering of meshes takes a matter of seconds. This prototype implementation contains many opportunities for efficiency improvements: the all-pairs contour intersection is probably too conservative, and a triangulation is not strictly necessary to find a region's interior point.

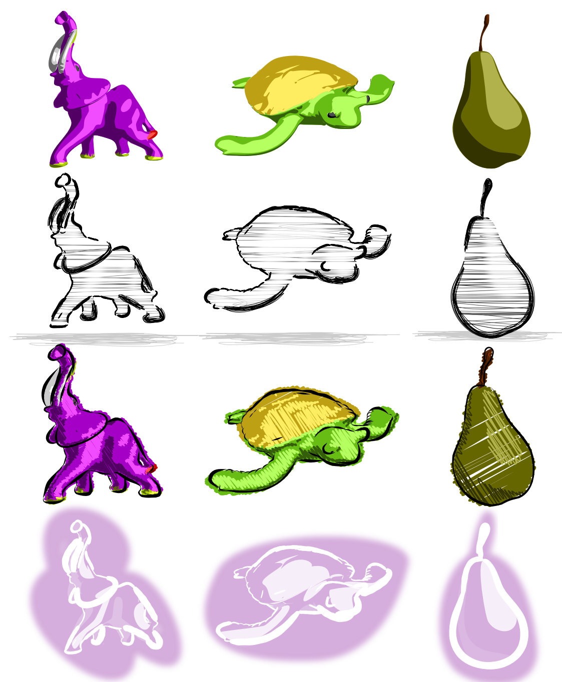

Figure 11: OBJECT/STYLE MATRIX – *Three models of varying complexity rendered in four different styles.*

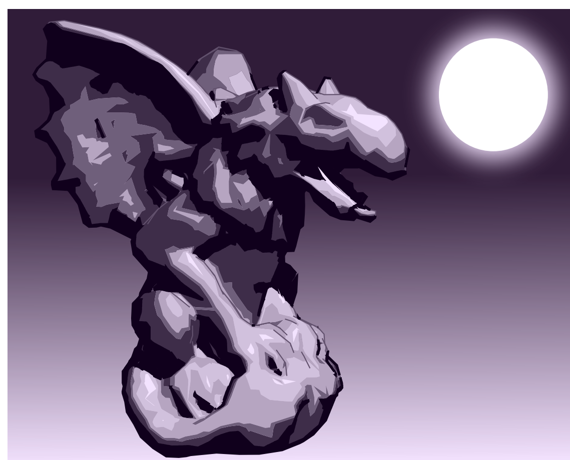

Figure 12: *Lighting and stylization create a dark mood.*

Currently, styles are not yet expressed in a scripting language, but compiled with C++ to allow a very efficient style application. In consequence, we can treat even complex maps rapidly. All styles are applied within the order of a second. Nevertheless, in the future we plan to rely on compiling scripts to share the benefits of both worlds.

Coupled with an intuitive interactive modeling system, we envision that this rendering engine would enable consumers

© 2008 The Author(s)
Journal compilation © 2008 The Eurographics Association and Blackwell Publishing Ltd.

Counts/Times (ms)	Pig	Hand	Pepper	Feline
Faces	9,200	8,000	31,280	8,356
Segments	2,068	2,244	2,869	3,858
Crossings	439	945	677	2,672
Segments + Crossings	2,507	3,189	3,546	6,530
Visible	1,015	1,091	1,159	1,901
Regions	46	93	51	260
Contour Extraction	14	12	39	13
Contour Intersection	81	102	154	310
Visibility Culling	91	436	226	162
Planar Map Construction	219	300	252	590
Triangulation	585	951	589	1,935
Region Classification	911	1,049	1,224	3,157
Total Time	1,901	2,850	2,484	6,167

Table 1: *Structure (top) and time results (bottom) over several models. Times reported in milliseconds.*

to quickly construct custom illustrations from scratch that match the appearance of the professional artwork found in commercial clip-art collections, while at the same time enabling reuse of their digital assets.

Figure 13: *A detailed dragon model*

Acknowledgments. Elmar Eisemann was supported by the Exploradoc program of the region Rhône-Alpes. Many thanks for the models go to Stanford, Caltech and INRIA, IMATI (provided by the AIM@SHAPE repository).

References

[BKR*05] BURNS M., KLAWE J., RUSINKIEWICZ S., FINKELSTEIN A., DECARLO D.: Line drawings from volume data. In *Proc. SIGGRAPH, ACM TOG* (2005), pp. 512–518.

[BNTS07] BOUSSEAU A., NEYRET F., THOLLOT J., SALESIN D.: Video watercolorization using bidirectional texture advection. *Proc. SIGGRAPH, ACM TOG 26*, 3, Article 104 (2007).

[BTM06] BARLA P., THOLLOT J., MARKOSIAN L.: X-toon: An extended toon shader. In *Proc. NPAR* (2006), pp. 127–132.

[CAS*97] CURTIS C. J., ANDERSON S. E., SEIMS J. E., FLEISCHER K. W., SALESIN D. H.: Computer-generated watercolor. *Proc. SIGGRAPH, Computer Graphics 31* (1997), 421–430.

[DFRS03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A.: Suggestive contours for conveying shape. *Proc. SIGGRAPH, ACM TOG 22*, 3 (2003), 848–855.

[DS02] DECARLO D., SANTELLA A.: Stylization and abstraction of photographs. In *Proc. SIGGRAPH, ACM TOG* (2002), pp. 769–776.

[Elb98] ELBER G.: Line art illustrations of parametric and implicit forms. *IEEE TVCG 4*, 1 (1998), 71–81.

[FHHN99] FLATO E., HALPERIN D., HANNIEL I., NECHUSHTAN O.: The design and implementation of planar maps in CGAL. *Proc. Workshop on Algorithm Eng. 1668/1999* (1999), 154.

[FMS02] FREUDENBERG B., MASUCH M., STROTHOTTE T.: Real-time halftoning: a primitive for non-photorealistic shading. In *Proc. EGRW* (2002), pp. 227–232.

[GG01] GOOCH B., GOOCH A. A.: *Non-Photorealistic Rendering*. A. K. Peters, 2001.

[GTDS04] GRABLI S., TURQUIN E., DURAND F., SILLION F.: Programmable style for NPR line drawing. In *Proc. EGSR* (2004), pp. 33–44.

[GVH07] GOODWIN T., VOLLICK I., HERTZMANN A.: Isophote distance: a shading approach to artistic stroke thickness. In *Proc. NPAR* (2007), pp. 53–62.

[Her99] HERTZMANN A.: Introduction to 3D Non-Photorealistic Rendering: Silhouettes and Outlines. *SIGGRAPH Course Notes* (1999).

[HLW93] HSU S. C., LEE I. H. H., WISEMAN N. E.: Skeletal strokes. In *Proc. UIST* (1993), pp. 197–206.

[HMK02] HEKMATZADA D., MESETH J., KLEIN R.: Non-photorealistic rendering of complex 3D models on mobile devices. In *Proc. Conf. Intl. Assoc. Math. Geol.* (Sep. 2002), vol. 2, pp. 93–98.

[HOCS02] HERTZMANN A., OLIVER N., CURLESS B., SEITZ S. M.: Curve analogies. In *Proc. EGRW* (2002), pp. 233–246.

[HZ00] HERTZMANN A., ZORIN D.: Illustrating smooth surfaces. In *Proc. SIGGRAPH* (2000), pp. 517–526.

[iAWB06] ICHI ANJYO K., WEMLER S., BAXTER W.: Tweakable light and shade for cartoon animation. In *Proc. NPAR* (2006), pp. 133–139.

[ICS05] ISENBERG T., CARPENDALE M. S. T., SOUSA M. C.: Breaking the pixel barrier. In *Proc. Computational Aesthetics* (2005), pp. 41–48.

[IFH*03] ISENBERG T., FREUDENBERG B., HALPER

© 2008 The Author(s)
Journal compilation © 2008 The Eurographics Association and Blackwell Publishing Ltd.

N., SCHLECHTWEG S., STROTHOTTE T.: A Developer's Guide to Silhouette Algorithms for Polygonal Models. *IEEE CG&A 23*, 4 (2003), 28–37.

[IHS02] ISENBERG T., HALPER N., STROTHOTTE T.: Stylizing Silhouettes at Interactive Rates: From Silhouette Edges to Silhouette Strokes. *Proc. Eurographics, CGF 21*, 3 (2002), 249–258.

[JD07] JUDD T., DURAND F.: Apparent ridges for line drawings. *Proc. SIGGRAPH, ACM TOG 26*, 3, Article 19 (2007).

[KDMF03] KALNINS R. D., DAVIDSON P. L., MARKOSIAN L., FINKELSTEIN A.: Coherent stylized silhouettes. *Proc. SIGGRAPH, ACM TOG 22*, 3 (2003), 856–861.

[KHCC05] KANG H., HE W., CHUI C., CHAKRABORTY U.: Interactive sketch generation. In *Proc. Pacific Graphics* (2005), pp. 821–830.

[KMM*02] KALNINS R. D., MARKOSIAN L., MEIER B. J., KOWALSKI M. A., LEE J. C., DAVIDSON P. L., WEBB M., HUGHES J. F., FINKELSTEIN A.: WYSIWYG NPR: drawing strokes directly on 3D models. In *Proc. SIGGRAPH, ACM TOG* (2002), pp. 755–762.

[KWH06] KOLLIOPOULOS A., WANG J. M., HERTZMANN A.: Segmentation-based 3D artistic rendering. In *Proc. EGSR* (2006), pp. 361–370.

[LD04] LUFT T., DEUSSEN O.: Watercolor illustrations of plants using a blurred depth test. In *Proc. NPAR* (2004), pp. 11–20.

[LMHB00] LAKE A., MARSHALL C., HARRIS M., BLACKSTEIN M.: Stylized rendering techniques for scalable real-time 3D animation. In *Proc. NPAR* (2000), pp. 13–20.

[LMLH07] LEE Y., MARKOSIAN L., LEE S., HUGHES J. F.: Line drawings via abstracted shading. *Proc. SIGGRAPH, ACM TOG 26*, 3, Article 18 (2007).

[Mei96] MEIER B. J.: Painterly rendering for animation. In *Proc. SIGGRAPH, Computer Graphics* (1996), pp. 477–484.

[MF06] MCGUIRE M., FEIN A.: Real-time rendering of cartoon smoke and clouds. In *Proc. NPAR* (2006), pp. 21–26.

[MKG*97] MARKOSIAN L., KOWALSKI M. A., GOLDSTEIN D., TRYCHIN S. J., HUGHES J. F., BOURDEV L. D.: Real-time nonphotorealistic rendering. In *Proc. SIGGRAPH, ACM TOG* (1997), pp. 415–420.

[NM00] NORTHRUP J. D., MARKOSIAN L.: Artistic silhouettes: a hybrid approach. In *Proc. NPAR* (2000), pp. 31–37.

[OZ06] OLSON M., ZHANG H.: Silhouette Extraction in Hough Space. *Proc. Eurographics, CGF 25*, 3 (Sept. 2006), 273–282.

[PB06] PRICE B., BARRETT W.: Object-based vectorization for interactive image editing. *Vis. Comput. 22*, 9 (2006), 661–670.

[PFS03] PASTOR O. M., FREUDENBERG B., STROTHOTTE T.: Real-time animated stippling. *IEEE CG&A 23*, 4 (2003), 62–68.

[PHWF01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A.: Real-time hatching. *Proc. SIGGRAPH, ACM TOG* (2001), 581–586.

[QTG*06] QUILLET J.-C., THOMAS G., GRANIER X., GUITTON P., MARVIE J.-E.: Using expressive rendering for remote visualization of large city models. In *Proc. Web3D* (2006), pp. 27–35.

[Sec02] SECORD A.: Weighted Voronoi stippling. In *Proc. NPAR* (2002), pp. 37–43.

[SEH08] STROILA M., EISEMANN E., HART J. C.: Clip art rendering of smooth isosurfaces. *IEEE TVCG 14*, 1 (2008), 71–81.

[SH05] SU W. Y., HART J. C.: A programmable particle system framework for shape modeling. In *Proc. SMI* (2005).

[SLWS07] SUN J., LIANG L., WEN F., SHUM H.-Y.: Image vectorization using optimized gradient meshes. *Proc. SIGGRAPH, ACM TOG 26*, 3, Article 11 (2007).

[SMC04] SELLE A., MOHR A., CHENNEY S.: Cartoon rendering of smoke animations. In *Proc. NPAR* (2004), pp. 57–60.

[SS02] STROTHOTTE T., SCHLECHTWEG S.: *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann, 2002.

[ST90] SAITO T., TAKAHASHI T.: Comprehensible rendering of 3-D shapes. *Proc. SIGGRAPH, Computer Graphics 24*, 4 (1990), 197–206.

[SVG03] SVG WORKING GROUP: Scalable Vector Graphics (SVG) 1.1 Specification. Website, 2003.

[TiABI07] TODO H., ICHI ANJYO K., BAXTER W., IGARASHI T.: Locally controllable stylized shading. *Proc. SIGGRAPH, ACM TOG 26*, 3, Article 17 (2007).

[WB03] WINNEMÖLLER H., BANGAY S.: Rendering optimisations for stylised sketching. In *Proc. AFRIGRAPH* (2003), pp. 117–122.

[WQL*06] WEN F., Q.LUAN, LIANG L., XU Y.-Q., SHUM H.-Y.: Color sketch generation. In *Proc. NPAR* (2006), pp. 47–54.

[WS94] WINKENBACH G., SALESIN D. H.: Computer-generated pen-and-ink illustration. *Proc. SIGGRAPH, Computer Graphics 28* (1994), 91–100.

[WS96] WINKENBACH G., SALESIN D. H.: Rendering parametric surfaces in pen and ink. *Proc. SIGGRAPH, Computer Graphics 30* (1996), 469–476.

© 2008 The Author(s)
Journal compilation © 2008 The Eurographics Association and Blackwell Publishing Ltd.