Interactive Shading of 2.5D Models

João Paulo Gois* Bruno A. D. Marques† Harlen C. Batagelo‡

Universidade Federal do ABC, Brazil

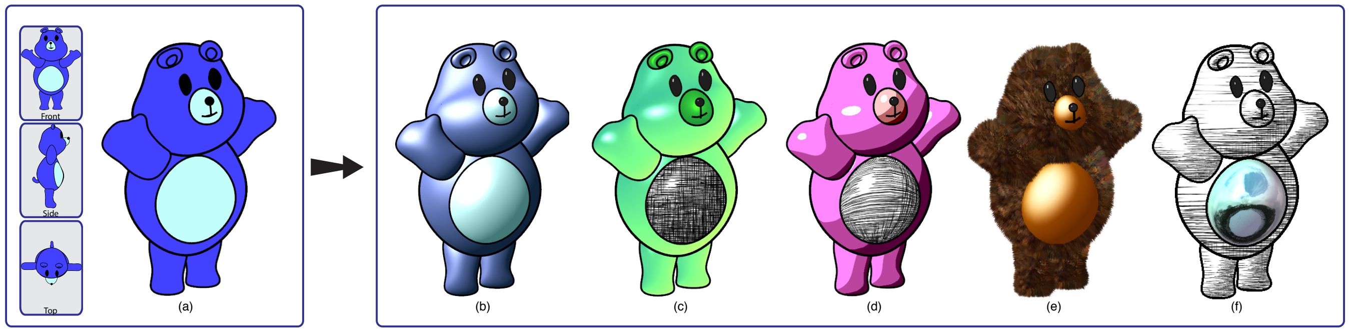


Figure 1: Interactive shading of 2.5D models: On the left, three 2D user-drawn views of a model (front, side and top) and (a) the resulting 2.5D model which simulates 3D rotation; on the right the shaded 2.5D model with (b) Phong shading, (c) Gooch shading and screen-space texture hatching, (d) cel shading and object-space texture hatching, (e) fur simulation and Phong shading, (f) screen-space texture hatching and environment mapping. The modeling and the shading are both generated in real-time.

Abstract

Advances in computer-assisted methods for designing and animating 2D artistic models have incorporated depth and orientation cues such as shading and lighting effects. These features improve the visual perception of the models while increasing the artists' flexibility to achieve distinctive design styles. An advance that has gained particular attention in the last years is the 2.5D modeling, which simulates 3D rotations from a set of 2D vector arts. This creates not only the perception of animated 3D orientation, but also automates the process of inbetweening. However, previous 2.5D modeling techniques do not allow the use of interactive shading effects. In this work, we tackle the problem of providing interactive 3D shading effects to 2.5D modeling. Our technique relies on the graphics pipeline to infer relief and to simulate the 3D rotation of the shading effects inside the 2D models in real-time. We demonstrate the application on Phong, Gooch and cel shadings, as well as environment mapping, fur simulation, animated texture mapping, and (object-space and screen-space) texture hatchings.

Keywords: 2.5D Modeling, Shading Effects, Real-time Rendering, Cartoon

Index Terms: I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1 Introduction

Animators have used computer-assisted techniques to automate or facilitate animating 2D drawings. Particularly, strategies which generate intermediate frames between two keyframes (inbetweening techniques) [5, 12] and the simulation of 3D effects on cartoons [16, 6, 24] have received much attention.

2.5D modeling techniques aim at simulating 3D rotations from a set of 2D drawings composed of vector-art shapes [5, 21]. Generally, front, top and side views of the 2D drawings suffice to mimic plausible 3D points of view.

Current 2D drawing tools employ shading techniques such as light mapping, gradient-based region filling, texturing and shadows. However, interactive 2.5D modeling techniques only support 2D shapes filled with solid colors. This restricts not only the perception of shading as the shapes are rotated but also the choice of materials to achieve different artistic effects.

In the present work, we propose a technique to incorporate shading effects to 2.5D modeling. Fig. 1 presents examples of shaded 2.5D models generated by our approach. Specifically, we demonstrate the technique for a variety of shading effects originally proposed for 3D real-time rendering, such as Phong, Gooch and cel shadings, environment mapping, static and animated texture mappings, texture hatchings and fur simulation. These effects depend on geometric properties of 3D models such as surface normals and surface parametrization, which are not available in previous 2.5D modeling approaches.

Our method begins in the CPU with the 2.5D modeling (Sec. 3). After this initial 2.5D modeling, we delegate to the GPU the 2.5D shading simulation. Specifically, we estimate on the GPU the 3D properties in the interior of the 2D drawings and use shaded 3D reference models to guide the shading of the 2D shapes and to simulate 3D rotations (Sec. 4). As a byproduct of our approach, we present an interactive 2.5D modeling tool capable of simulating different 3D shading effects in real-time (Sec. 5).

2 Related Work

In the last few years, many approaches have been proposed to enrich the visual appearance of 2D drawings. Di Fiore *et al.* [5] compute inbetweening of 2D drawings by simulating 3D rotations using 2.5D modeling. The artist provides a set of 2D point-of-views of a 2D drawing and defines the depth order of each curve of each drawing. The technique then generates new point-of-views for any given 3D orientations while simultaneously interpolating the input drawings and resolving the depth order of the curves. Rivers *et al.*

*e-mail:joao.gois@ufabc.edu.br
†e-mail:bruno.marques@ufabc.edu.br
‡e-mail:harlen.batagelo@ufabc.edu.br

Copyright held by authors. Permission granted to CHCCS/SCDHM to publish in print and digital form, and ACM to publish electronically.

[21] also present a technique for simulating 3D rotations from 2D drawings, titled 2.5D Cartoon Models. Unlike the previous work, they determine the depth order automatically and use a pitch-yaw parametrized orientation space to interpolate the 2D drawings.

A technique for automatically converting 3D objects into 2.5D Cartoon Models was proposed by An and colleagues [1, 2]. Instead of using 2D graphics in different views, the method begins with a 3D object that is segmented to generate the depth-ordered 2D curves used by the 2.5D Cartoon Models.

Di Fiore and Van Reeth [6] proposed an approach that assists the artist in drawing new spatial point-of-views from a single 2D drawing. 3D models are generated from rounded and circular shapes of the initial 2D drawing. The outlines of these 3D models can be drawn in new 3D orientations, thus guiding the artist in the creation of new 2D views.

Yeh *et al.* [28] presented another use for the name 2.5D, called *double-sided 2.5D graphics*, to simulate effects that use the front and back sides of 2D drawings. Particularly, they attach textures to both sides of the 2D shape and simulate geometrical effects such as rolling, twisting and folding. Contrasting with previous 2.5D modeling techniques [21, 5], the double-sided 2.5D graphics does not simulate rigid 3D rotations.

Sýkora and colleagues [25] presented an easy-to-use interface to incorporate smooth depth information in 2D cartoons. The method employs a set of (in)equalities that avoid the need of absolute depth levels. This is formulated as an optimization problem that can be solved by quadratic programming. The authors noticed that this approach is time-consuming for interactive rates and proposed an approximate solution which depends on solving a Laplace equation. This formulation leads to a very sparse linear system that the authors suggest solving on the GPU.

Literature provides two main strategies for simulating relief from 2D drawings. The first is based on reconstructing 3D geometry, *e.g.*, mesh-based [13, 10, 17, 18], point-set-based [4] or curve-based [27] approaches. The second is based on producing visual effects from estimated 3D normals inside the 2D curves [11, 26, 16, 9, 22]. The reason for using 3D normals to simulate the effect of relief is the same employed by the normal mapping technique [14]. The interaction of the light vector with the surface normals simulates the effect of relief on a smooth surface as described in the Phong reflection model. In the remaining of this section, we focus on this strategy as our approach belongs to it.

Johnston [11] proposed the Lumo framework for computing normal mappings from 2D curves. It assumes that each drawing curve belongs to the silhouette of a 3D object which implies that the normals on the curve are not only orthogonal to the curve but also is in the screen plane. From normals sampled along each curve, Lumo fills the region inside the curve with a 3D normal field that is propagated by an iterative dampened-spring diffuser method.

Nascimento *et al.* [16] proposed an explicit method to accurately compute the 3D normal fields inside the curves. The authors claim that this method guarantees smoothness for a coherent illumination. An advantage of this approach is that it is easily parallelizable on the GPU.

The technique developed by Sýkora *et al.* [25], which propagates depth estimates inside drawing curves, can be used to interpolate normals with respect to depth discontinuities. This technique was employed in later works for improving shading effects. Particularly, this is used in Textoons, a method for coherent texture mapping for hand-drawn cartoon animation [23], and Ink-and-Ray, a method for global illumination effects [24]. The shading effects produced by these methods are not generated in real-time and do not consider the simulation of 3D rotations.

3 2.5D MODELING

The effectiveness of a 2.5D modeling relies on its ability to interpolate 2D drawings considering their designated space orientation, and to determine the depth order of the shapes that compose the resulting 2.5D model.

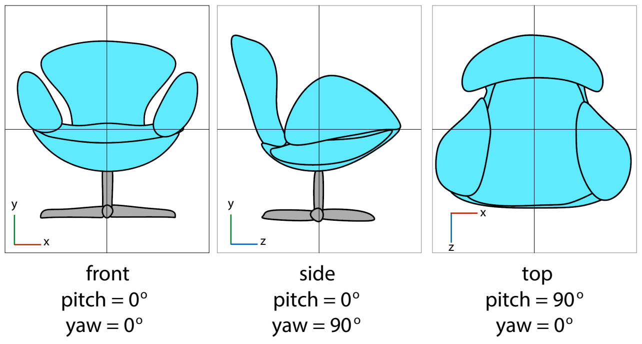


Figure 2: Set of 2D inputs and their respective positions in the pitch-yaw orientation space.

In our approach, the user defines a set of 2D views (*e.g.* front, side and top) of the same object. Figure 2 depicts an example of a chair. The drawing in each view is composed of a set of filled shapes. Each 2D view is assigned to a $(pitch, yaw)$ parameter that corresponds to a position in the orientation space with $-\frac{\pi}{2} \leq pitch \leq \frac{\pi}{2}$ and $-\pi \leq yaw < \pi$, similar to Rivers *et al.* [21]. Notice that, although Figure 2 presents only three distinct views with $(pitch, yaw) = (0,0), \left(0, \frac{\pi}{2}\right), \left(\frac{\pi}{2}, 0\right)$, any orientation and any number of views are allowed. We also assume that the 2D drawings for the parameters $(pitch, yaw) = (0,0)$ (front view) and $(pitch, yaw) = (0, \pi)$ (back view) correspond to orthographic projections onto the xy-plane. Any 2D drawings defined in different pitch-yaw parameters, for instance, side and top views, will correspond to a projection in a plane nonparallel to the xy-plane, therefore with a uniquely defined z component that can be used to determine the depth order. Thus, we can define a (x, y, z) reference position for each drawing shape. Specifically, the (x, y) components correspond to the center of the bounding box of the shape in the front view. The z component could be the z value of any non-front and non-back drawings. However, since the z values of the same shape in different views may vary, we take the average of the z values over all views that contain this depth information, *i.e.*, all drawings except the front and back one. An alternative solution for automatic depth order estimation was proposed by Rivers *et al.* [21], which computes the (x, y, z) reference position iteratively.

An effective approach to interpolate among the 2D drawings was proposed by Rivers *et al.* [21]. In that approach, the pitch-yaw orientation space is first mapped to a 2D plane. Each 2D user drawing is associated with a 2D position on the plane. In addition, a set of other 2D drawings is automatically generated by reflections and rotations of the user drawings. These automatically generated drawings are also associated to 2D positions on the plane. A Delaunay triangulation is then computed considering the 2D positions of the drawings on the plane as vertices of the triangulation. New drawings are determined by barycentric interpolation inside the Delaunay triangulation. We use this method in our approach.

In order to compute the interpolation among the 2D drawings, we firstly discretise the contours by polylines with uniformly spaced vertices. For computational simplicity, we assume that for each contour the same number of vertices is used in all of its corresponding contours in the other 2D views. For instance, considering the chair in Fig. 2, its left armrest has the same number of vertices in all 2D views. Once the contours are consistently discretised, the interpolation is then performed among these vertices.

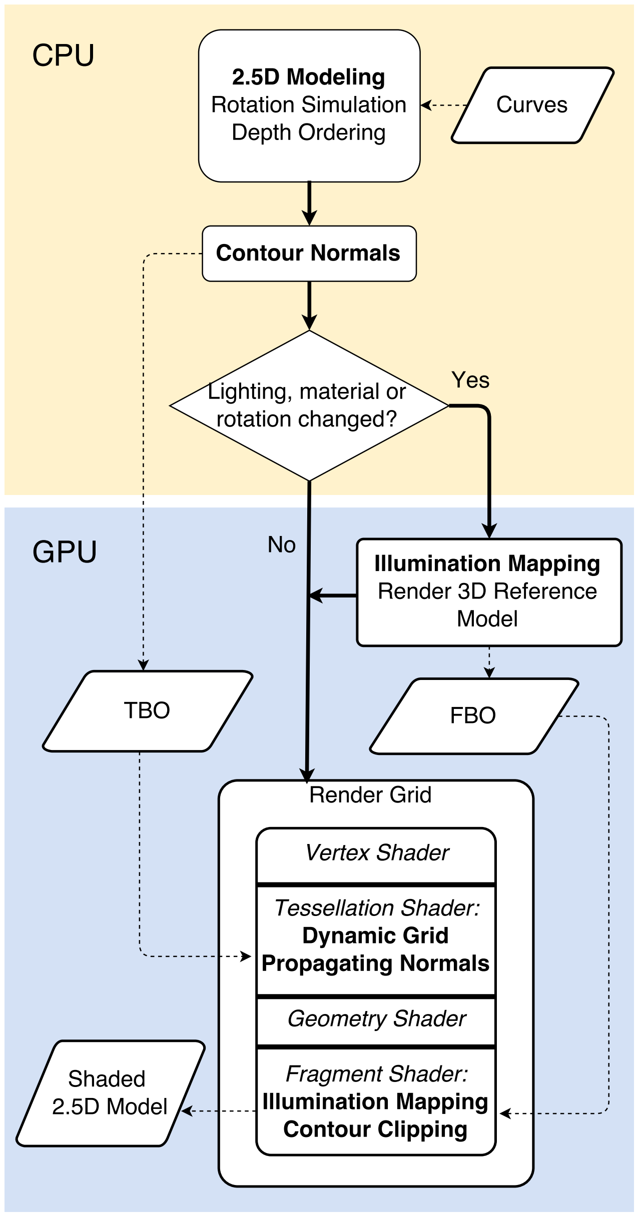


Figure 3: Overall control and data flow from the CPU to the GPU for shading 2.5D models.

Since the depth ordering and the interpolation involve sequential operations which are not computationally intensive, we implement them on the CPU.

4 Shading 2.5D Models

Our goal is to interactively simulate 3D shading effects in the interior of the shapes of the 2.5D models. Figure 3 presents a general diagram of our technique, from 2.5D modeling on the CPU to the final shaded model on the GPU. Bold arrows indicate processing flow and dashed arrows show data communication. In this section we detail this workflow.

4.1 Contour Normals

Our technique uses 2D cubic splines as input curves. Thus, we can assume that the contour of any 2D shape of a 2.5D model corresponds to the silhouette of a smooth surface. The 3D normals of such surface silhouette have components $(x,y,0)$ where (x,y) are the normals along the contours of the 2D shapes [11, 16], as illustrated in Fig. 4-(a)-(b).

The contour normals are computed in the vertices of the polylines that approximate the contours. This process is performed on the CPU and the resulting normals are sent to the GPU as a texture buffer object (TBO), shown in Fig. 3. Once the contour normals are in the GPU, they will be propagated to the interior of the shape as 3D normals.

4.2 Propagating Normals

To compute a normal n in a point p inside the shape, we consider all the μ_i $(i = 1,\dots,N)$ normals at the p_i points along the contour of the shape. The x and y components of the normal n are then given by

$$n_{\{x,y\}} = \frac{\sum_{i=1}^{N} \frac{\mu_{i_{\{x,y\}}}}{\|p-p_i\|^2}}{\omega} \tag{1}$$

where

$$\omega = \sum_{i=1}^{N} \frac{1}{\|p-p_i\|^2}. \tag{2}$$

The normalization of the normal vector in p is ensured by imposing

$$n_z = \sqrt{1-n_x^2-n_y^2}. \tag{3}$$

There are different approaches in propagating normals. Johnston [11] computes the normals inside the shape by iteratively propagating the contour normals using a dampened-spring diffuser method. More recently, Nascimento *et al.* [16] proposed an explicit formulation that employs line integrals. Our approach can be seen as an approximation of this last one. It is found to be more suitable for the SIMD architecture of the GPU since it is easily parallelizable. Particularly, the normal of each point is independently computed in a single shader pass.

It is worth mentioning that the effectiveness of the method relies on averages of isotropically distributed data, *i.e.*, for reasonable results, it is expected that the data are evenly placed. Our regular distribution of points along the curve and the use of a dynamic regular grid, presented in Sec. 4.3, follow this assumption.

Besides the normals in the contour of the shape, we can also consider normal constraints inside the shape for producing effects such as ridges and creases.

4.3 Dynamic Grid

In order to avoid a computationally expensive per-pixel propagation of 3D normals inside the contours, we compute the 3D normals only at the vertices of the regular grids that fit the bounding rectangle of the shapes (Fig. 4-(c)-(d)). Once these normals are computed, they are interpolated for each fragment using linear interpolation in the rasterizer (Fig. 4-(e)). The resolution of these grids is dynamically adjusted according to the bounding rectangle of each shape.

In order to create the regular grid on-the-fly, we use the tessellation control and tessellation evaluation shader stages. We start with a single quad that is suitably scaled to fit the bounding rectangle of the shape that will be shaded. The inner and outer tessellation levels are adjusted to subdivide this quad in proportion to changes in the horizontal and vertical scaling factors of the bounding rectangle. In our experiments, the grid resolution ranges from 4×4 to 64×64 for a corresponding bounding rectangle of 512×512 pixels. This suffices for obtaining smooth normal fields at interactive frame rates. The Phong-shaded Turtle Model (Fig. 11) achieves 103 fps with this technique, while a per-pixel computation results in 50 fps with

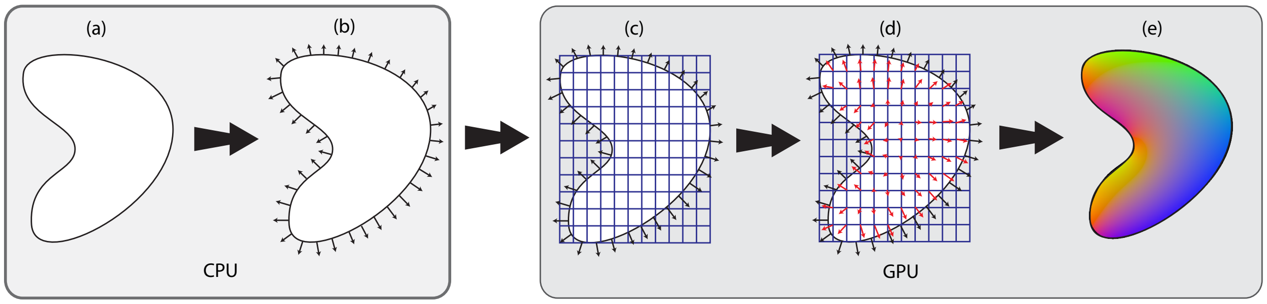


Figure 4: Pipeline for computing 3D normals inside a shape of a 2D drawing: (a) shape interpolated by the 2.5D modeling; (b) 2D normals computed along the contour of the shape; (c) tessellation of the bounding rectangle of the shape (d) 3D normals estimated in the vertices of the tessellated bounding rectangle; (e) normals interpolated for each fragment, encoded in RGB color.

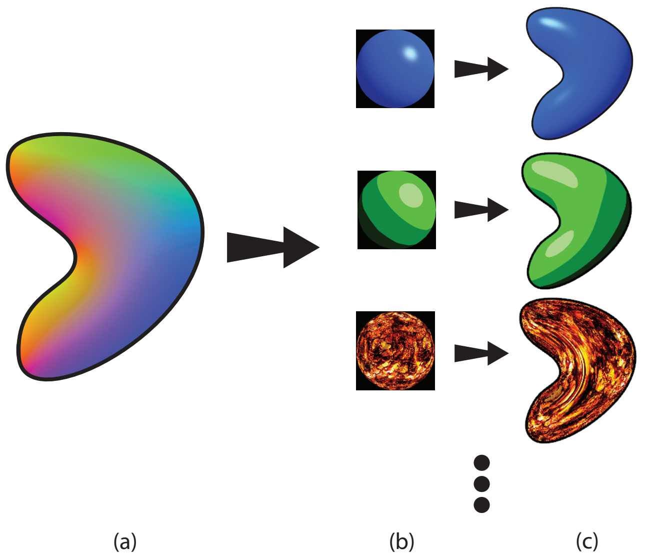


Figure 5: Using the estimated normals (a) and the rendered 3D reference model as an FBO (b) to shade the shapes (c). We use the reference model not only for speeding up lighting and texturing, but also for simulating the rotation of the shaded surface in the interior of the shape.

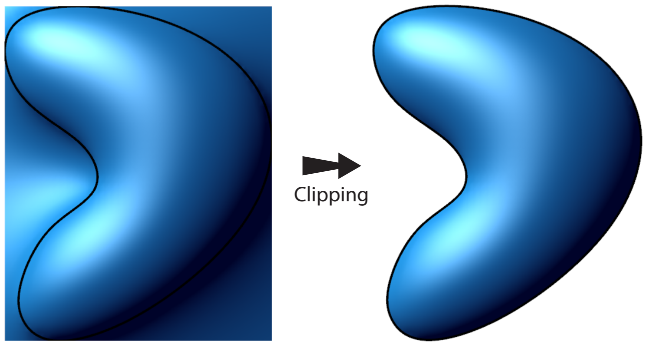


Figure 6: Example of the clipping of a rendered regular grid (left) to the interior of the shape (right).

equivalent visual quality. Generally, we experienced in our tests a speed up of at least 60% compared to a per-pixel evaluation.

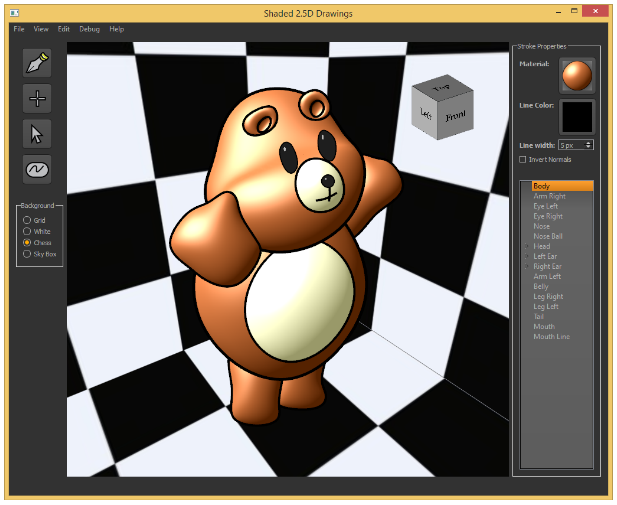
It is worth to mention that for bounding rectangles with aspect ratios different from one, the tessellation level is conservatively adjusted according to the dimension of longest length. This may still produce overly tessellated grids for highly stretched bounding rectangles, but guarantees that interpolation artifacts will not arise. We could have used an adaptive grid, instead. However, the regular grid is easily computed on the GPU and ensures smoothness to the normal field after linear interpolation on the rasterizer. It minimizes problems of scale dependency as new vertices are evenly created and scaled up or down proportionally to the contour size. In addition, the regular grid is necessary in our fur simulation approach to avoid discrepancies in the fur density.

4.4 Illumination Mapping

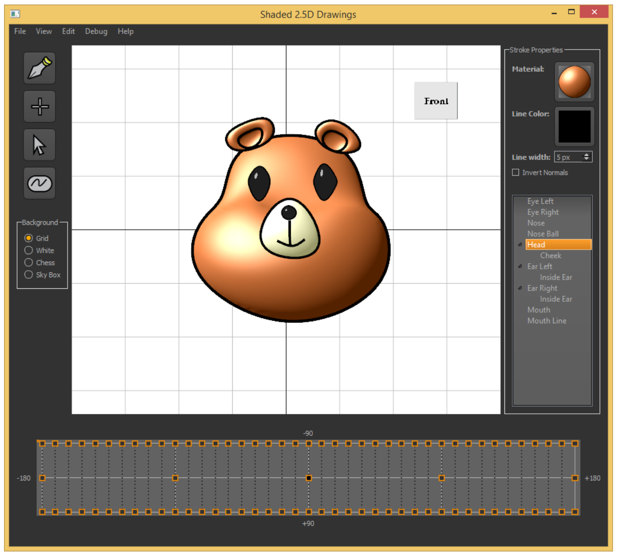
With the normals estimated for each fragment, we are now able to shade the interior of the shape. Instead of directly evaluating the lighting equation for each fragment inside a 2D shape, we first render to a texture a 3D reference model with the user-selected shader effect. This texture contains the shading result for every possible visible normal and will be mapped to the interior of the corresponding 2D shape according to the coordinates of the estimated normals (Fig. 5). Specifically, each reference model is a sphere whose rendering in a particular point of view is stored in a frame buffer object (FBO), as depicted in Fig. 5-(b). This FBO is used in a second shader pass as a look-up texture indexed by the x and y normal components. Since these components are in the range $[-1,1]$, we map them to texture coordinates in the range $[0,1]$ to sample the texture. This choice of using a proxy model for lighting is well known in the literature as a speed up approach for lighting [11] since the performance of the illumination does not depend on the complexity of the shapes. Another use of the 3D reference model is for texturing.

Albeit the 3D reference model is used for speeding up lighting and texturing, it has another important role in our 2.5D modeling approach. By performing a 3D rotation of this reference model using the same pitch and yaw angles used in the 2.5D interpolation process, we simulate the rotation of the shaded surface in the interior of the shape. Therefore, we must update the rendering of the 3D reference model whenever a rotation or change of the lighting parameters occur. Sometimes, the 3D reference model is not needed and the effect can be applied directly during the rendering of the shape. For instance, we show effects such as fur simulation and screen-space texture hatching where screen space suffices for surface parametrization.

Differently from the technique of Johnston [11], which uses a static texture of the rendered sphere, we must render this reference model whenever a change of illumination occurs. This is required due to the 3D rotations in the 2.5D Cartoon Models technique [21] which may change the relative position between the light sources and the viewer. This is also needed when the user changes the current effect or when animated shading is employed.



(a)



(b)

Figure 7: The modeling and shading tool: (a) main window with a Phong-shaded model in an oblique view; (b) main window presenting the advanced navigation interface, shown in the bottom part of the window. It is also possible to see the application of shading constraints in the ears and cheeks.

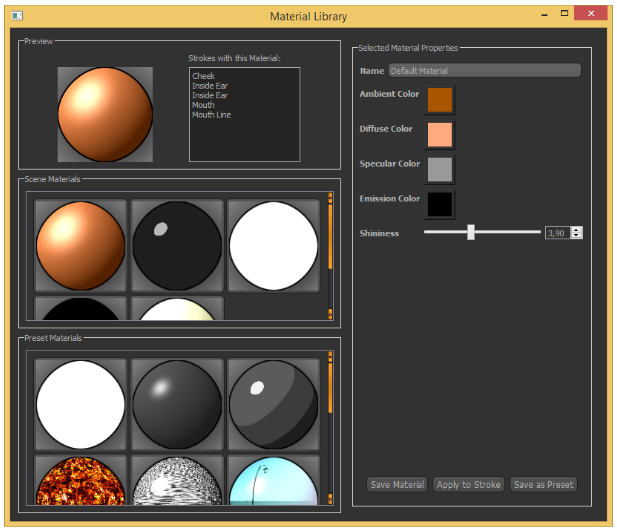


Figure 8: The modeling and shading tool: material library window containing the scene materials used in Fig. 7-(a) and the list of preset materials.

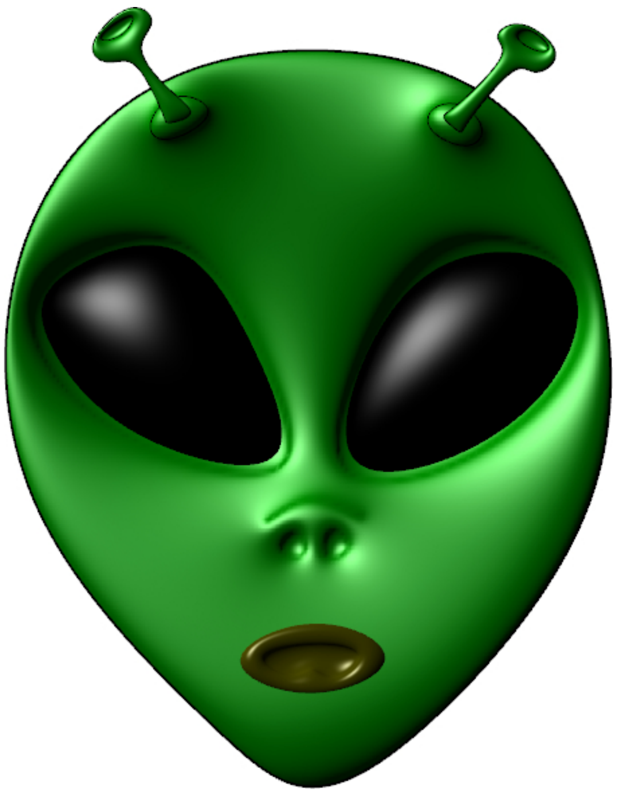


Figure 9: Example of use of curves for creating shading constraints (regions of the eyes, nose and mouth).

4.5 Contour Clipping

So far, we presented our approach considering that shading is applied only in the interior of a shape. Actually, we implemented it in a way that the entire area inside the bounding rectangle is shaded. To discard the exterior of the shape, we perform a per-pixel clipping operation in screen space (Fig. 6).

Another approach could be the creation of a mesh constrained to the interior of the shape, thus avoiding the clipping and the shading of the exterior of the shape. However, this would require a remeshing operation whenever the shape changed, imposing an additional overhead for updating the corresponding vertex buffer object to the GPU. Our method is simpler since it uses a single static quad as input geometry, and more effective since the clipping is per-pixel accurate.

4.6 Shading Effects

We demonstrate a set of well-known 3D shading effects to our 2.5D modeling: Phong and Gooch shadings [8], cel shadings, environment mapping, static texture mapping, animated texture mapping (lava shading using parallax scroll [3]) and texture hatching [19] based both in screen space parametrization and 3D reference model parametrization (object-space). We also demonstrate an additional effect, the fur simulation, that is computed in the geometry shader and thus does not depend on a 3D reference model for surface parametrization.

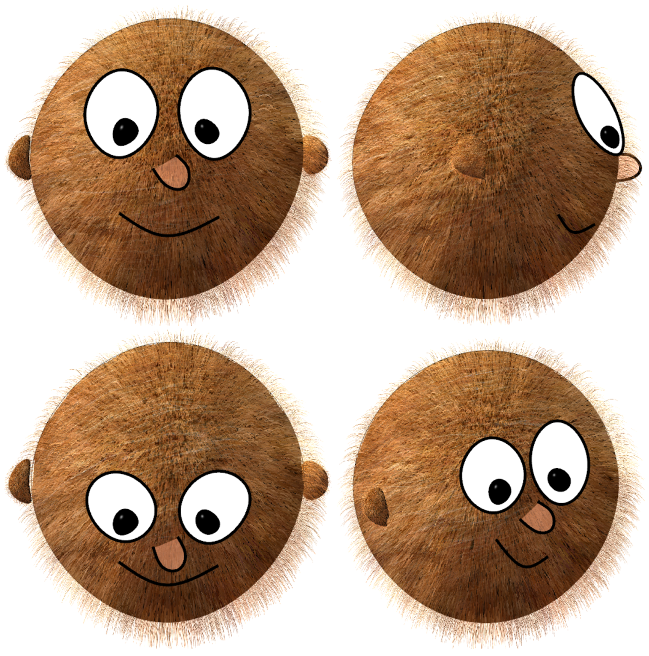


Figure 10: Shaded 2.5D Coconut Head model: Fur shading effect applied on the head and ears of the model.

Phong, Gooch and cel shadings, environment mapping, static and animated texture mappings, and object-space texture hatching are applied on the 3D reference model. For that reason, they are the same shaders used for shading 3D models, *i.e.*, they do not require any special adjustment to our 2.5D modeling approach. Once the 3D reference model is rendered to a texture, it is mapped to the interior of the 2D shape using the propagated normals. On the other hand, instead of using the 3D reference model for texture mapping, screen-space texture hatching uses the pitch-yaw parametrized orientation space to translate the hatching textures directly in screen space. This effect only employs the 3D reference model for simulating the highlights on the surface.

Different from the previous effects, the fur simulation cannot be applied to the 3D reference model because the new primitives generated for the simulation of fur would be distorted and clipped after illumination mapping in the interior of the shape. Fur is computed in the geometry shader after tessellation. We create a set of line strips for each quad of the tessellated bounding rectangle that represents the fur. The line strips follow the direction of the propagated normals. For a more realistic effect, each line strip is slightly perturbed and bended to simulate the influence of gravity. As the fur is created in the geometry shader, it is only valid for the current view and must be regenerated whenever the shape changes. Since the fur is normally not dense enough to cover all the fragments inside the shape, we can use any of the previous shaders to fill the background before applying the fur, for instance, Phong shading (Fig. 10) or texture mapping (Fig. 11-(c)). In such cases, the color of the fur follows the color of the chosen background.

To avoid clipping the line strips that extrapolate to the exterior of the shape, we must know if they start inside the shape. This is determined by setting, for each vertex of the line strip, the texture coordinate of the initial vertex in the tessellated bounding rectangle. In the fragment shader, a binary mask texture indicating the inside and outside of the shape is sampled for each fragment of each line strip using the texture coordinate. If the sample corresponds to an outside region, the fragment is part of a line strip that started outside the shape and therefore is discarded.

5 Modeling and Shading Tool

We implemented an interactive user interface for demonstrating our 2.5D modeling and shading approach (Figs. 7 and 8). The 2D drawings are created similarly as in vector-art applications with support to open and closed cubic splines. The user can navigate in the 3D space with a virtual trackball or by manipulating a *view cube* located in the top-right corner of the viewport. This functionality is based on the camera rotation widgets normally found in 3D modeling applications, which makes the 2.5D modeling navigation more intuitive.

Generally, the user draws three 2D views of the model. Front, top and side views suffice for most cases. These orthogonal views can be quickly accessed by clicking on the sides of the view cube. Oblique views are accessed by freely rotating the view cube.

We also provide an advanced navigation interface that presents the pitch-yaw map (Figure 7-(b)) as a grid of points that indicates where the 2D views are placed in the orientation space, as proposed by Rivers and colleagues [21].

The editing tools are accessed by an edit toolbar composed of four buttons, shown in the top-left part of the main window (Fig. 7). The first three correspond to pen, move and select functionalities. The fourth allows the definition of splines that produce shading constraints in the shapes (Figs. 7-(b) and 9).

We also provide a material library (Fig. 8) that contains preset shaders. The user can manipulate the parameters of the shaders in the interface and save them as new effects.

6 Results

We implemented a cross-platform application that integrates 2.5D modeling capabilities with our interactive 2.5D shading approach. The user interface is presented in Section 5. For this implementation, we employed the Qt framework [20] version 5.3.2. The shaders were implemented in OpenGL Shading Language (GLSL), version 4.2. Our tests were run on a PC Intel i5-4670K, with 8GB RAM and ATI Radeon R9 290 graphics card.

Fig. 11 presents a set of different effects applied on a 2.5D Turtle model. Fig. 10 illustrates the use of fur simulation to create the 2.5D Coconut-Head model. Fig. 12 shows the shaded 2.5D Chair model corresponding to the 2D drawing inputs presented with solid colors in Fig. 2. It can be observed that, for this model, despite its simplicity, the shading strongly enriches the perception of a 3D shape.

Table 1: Timing results for rendering the shaded 2.5D Turtle model. All the shapes were rendered using the same effect.

Rendering effect	fps
Solid colors	1035
Phong shading	103
Cel shading	104
Animated texture	102
Environment mapping	100
Gooch shading	99
Hatching (object-space)	96
Hatching (screen-space)	71
Fur simulation	41

For performance tests (Tabs. 1–2), we rendered the shaded 2.5D Turtle model, which is composed of 15 shapes, in a viewport with resolution of 800×600. Each contour was approximated by a polyline of 500 vertices. We found this number suitable for both ensur-

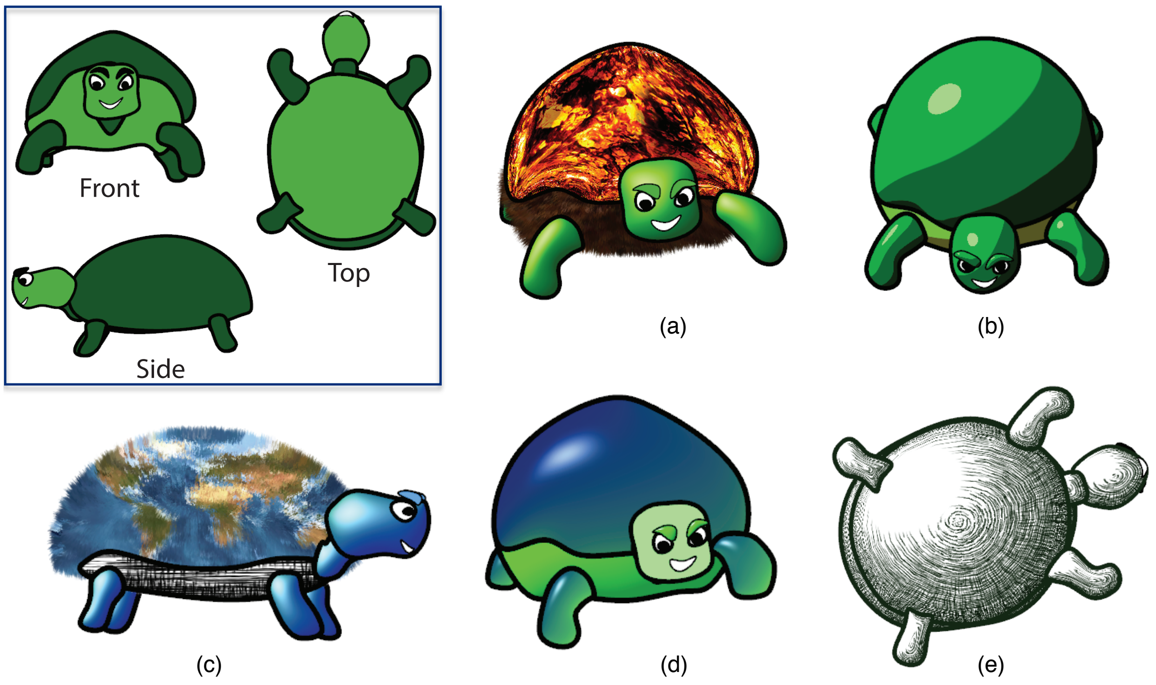


Figure 11: The 2.5D Turtle model rendered with different effects: (a) lava shading, Phong shading and fur simulation; (b) cartoon shading; (c) texture mapping with fur simulation, Gooch shading, and screen-space texture hatching; (d) Gooch shading; (e) object-space texture hatching. The 2D inputs are shown in the top-left box.

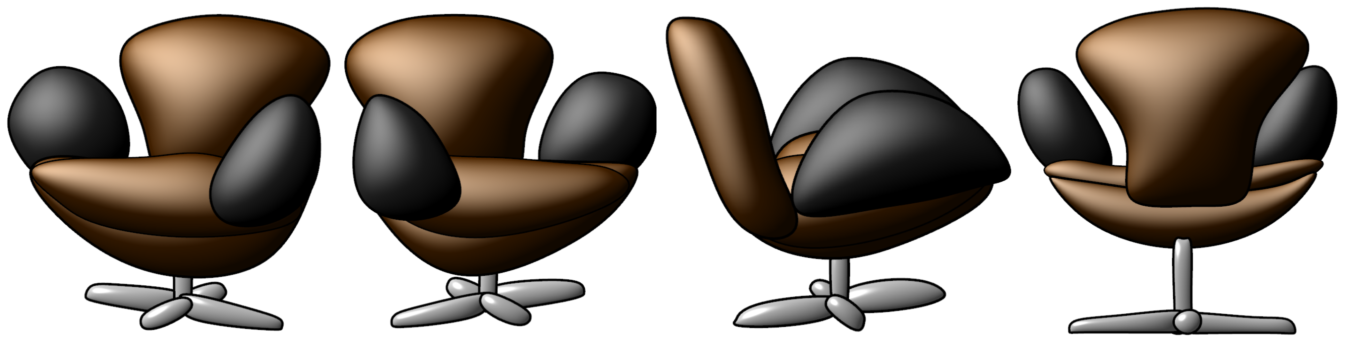


Figure 12: Shaded 2.5D Chair model rendered with Phong shading. The 2D inputs are presented in Fig. 2.

ing interactivity while generating smooth results for this viewport resolution.

Table 1 presents the *frames-per-second* (fps) for this model where all 15 shapes were rendered using the same effect. Most shaded effects achieve around 100 fps. As it can be observed, solid colors are the most efficient. This is caused by the estimation of normals, the use of 3D reference models, tessellation and mappings are not required. Screen-space texture hatching is more time consuming because, aside from the use of the 3D reference model to compute the highlights, it requires additional processing time to map the hatching textures in screen space. Lastly, fur simulation is the most time-consuming effect because it demands the generation of new primitives in the geometry shader.

Table 2 presents performance results showing the impact of the number of shapes shaded with Phong shading for the 2.5D Turtle model. The first column shows the number of shapes filled with Phong shading. The remaining shapes of the Turtle model are filled with solid colors. We also performed an analogous test where, instead of applying the Phong shading for all shapes, we used different (randomly chosen) effects for each shape. In that case, the timing results were similar to the previous test, which indicates that the use of distinct shading effects in the same model does not impact performance.

Table 2: Timing results for rendering the shaded 2.5D Turtle model for different numbers of shapes rendered with Phong shading.

Phong-shaded shapes	fps
0	1035
1	550
2	410
7	212
15	103

7 Conclusion

In this work, we proposed an approach for interactive shading of 2.5D models. Previous 2.5D modeling tools only supported shapes filled with solid colors. Our approach estimates 3D geometric properties to simulate 3D shading effects such as, but not limited to, Phong, Gooch and cel shadings, environment mapping, static and animated texture mappings, texture hatchings, and fur simulation. The availability of shading for 2.5D models not only enriches the perception of depth but also allows the use of a broader range of artistic effects. As a byproduct, we implemented an interactive computational tool for demonstrating the different 3D shading effects in the 2.5D modeling.

The proposed technique presents some limitations. The first is the use of a sphere as the 3D reference model for any type of shape. For contours with deep concavities, the use of a sphere as 3D reference model may present distortions and aliasing artifacts in the illumination and object-space texture mapping. Another limitation can be observed in the fur effect. Since the fur must be regenerated whenever the shape changes and we do not track the positions of previous fur, a shower door effect may be noticeable.

Focusing on tackling the aforementioned limitations, we envision two main directions to follow as future work. The first aims at investigating more sophisticate techniques for estimating relief and parameterizations for the interior of the shapes [22, 15, 9]. The second aims at providing new shading effects, for instance, global illumination [24] and vector graphics effects [7]. In both cases, we must be aware about the computational cost, since these techniques and effects are not directly suitable for real-time rendering.

Finally, we plan to improve and enrich the user interface of our application by incorporating more types of curves and objects as well as brushes for the shape outlines, and a timeline tool for producing animations.

Acknowledgements

The authors thank to Fundação de Amparo à Pesquisa do Estado de São Paulo – FAPESP (proc. 2014/11067-1), the National Council of Technological and Scientific Development (CNPq), and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) for the financial support of this work. The authors also thank to Emil Persson for making available the cube map texture "FootballField 3" in the website http://www.humus.name/index.php?page=Textures.

References

[1] F. An, X. Cai, and A. Sowmya. Automatic 2.5d cartoon modelling. In *International Conference Image and Vision Computing*, 2011.

[2] F. An, X. Cai, and A. Sowmya. Perceptual evaluation of automatic 2.5d cartoon modelling. In *Proceedings of the 12th Pacific Rim conference on Knowledge Management and Acquisition for Intelligent Systems*, PKAW'12, pages 28–42, Berlin, Heidelberg, 2012. Springer-Verlag.

[3] A. Balkan, J. Dura, A. Eden, B. Monnone, J. Palmer, J. Tarbell, and T. Yard. Parallax Scrolling. In *Flash 3D Cheats Most Wanted*, pages 121–164. Apress, 2003.

[4] E. V. Brazil, I. Macêdo, M. C. Sousa, L. H. de Figueiredo, and L. Velho. Sketching variational hermite-rbf implicits. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, SBIM '10, pages 1–8, Aire-la-Ville, Switzerland, 2010. Eurographics Association.

[5] F. Di Fiore, P. Schaeken, K. Elens, and F. Van Reeth. Automatic in-betweening in computer assisted animation by exploiting 2.5d modelling techniques. In *Proceedings of the Fourteenth Conference on Computer Animation*, pages 192–200. IEEE, 2001.

[6] F. Di Fiore and F. Van Reeth. Employing approximate 3d models to enrich traditional computer assisted animation. In *Proceedings of the Fifteenth Conference on Computer Animation*, pages 183–190. IEEE, 2002.

[7] M. Finch, J. Snyder, and H. Hoppe. Freeform vector graphics with controlled thin-plate splines. *ACM Trans. Graph.*, 30(6):166:1–166:10, Dec. 2011.

[8] A. Gooch, B. Gooch, P. Shirley, and E. Cohen. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 447–452, New York, NY, USA, 1998. ACM.

[9] J. Hahn, J. Qiu, E. Sugisaki, L. Jia, X.-C. Tai, and H. S. Seah. Stroke-based surface reconstruction. *Numer. Math. Theor. Meth. Appl.*, 6(1):297–324, 2013.

[10] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3d freeform design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 409–416, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[11] S. F. Johnston. Lumo: Illumination for cel animation. In *Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering*, NPAR '02, pages 45–ff, New York, NY, USA, 2002. ACM.

[12] B. Jones, J. Popovic, J. McCann, W. Li, and A. Bargteil. Dynamic sprites. In *Proceedings of Motion on Games*, MIG '13, pages 17:39–17:46, New York, NY, USA, 2013. ACM.

[13] O. A. Karpenko and J. F. Hughes. Smoothsketch: 3d free-form shapes from complex sketches. *ACM Trans. Graph.*, 25(3):589–598, July 2006.

[14] M. J. Kilgard. A practical and robust bump-mapping technique for today's gpus. In *Game Developers Conference*, 2000.

[15] J. Lopez-Moreno, S. Popov, A. Bousseau, M. Agrawala, and G. Drettakis. Depicting stylized materials with vector shade trees. *ACM Trans. Graph.*, 32(4):118:1–118:10, July 2013.

[16] R. Nascimento, F. Queiroz, A. Rocha, T. I. Ren, V. Mello, and A. Peixoto. Colorization and illumination of 2d animations based on a region-tree representation. In *24th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 9–16, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[17] L. Olsen, F. Samavati, and J. Jorge. Naturasketch: Modeling from images and natural sketches. *IEEE Comput. Graph. Appl.*, 31(6):24–34, Nov. 2011.

[18] L. Olsen, F. F. Samavati, M. C. Sousa, and J. A. Jorge. Sketch-based modeling: A survey. *Computers & Graphics*, 33(1):85–103, Feb. 2009.

[19] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 581–, New York, NY, USA, 2001. ACM.

[20] Qt. Qt Project http://qt-project.org, 2014.

[21] A. Rivers, T. Igarashi, and F. Durand. 2.5d cartoon models. *ACM Trans. Graph.*, 29(4):59:1–59:7, July 2010.

[22] C. Shao, A. Bousseau, A. Sheffer, and K. Singh. Crossshade: Shading concept sketches using cross-section curves. *ACM Trans. Graph.*, 31(4):45:1–45:11, July 2012.

[23] D. Sýkora, M. Ben-Chen, M. Čadík, B. Whited, and M. Simmons. Textoons: Practical texture mapping for hand-drawn cartoon animations. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, pages 75–83, 2011.

[24] D. Sýkora, L. Kavan, M. Čadík, O. Jamriška, A. Jacobson, B. Whited, M. Simmons, and O. Sorkine-Hornung. Ink-and-ray: Bas-relief meshes for adding global illumination effects to hand-drawn characters. *ACM Transaction on Graphics*, 33(2):16, 2014.

[25] D. Sýkora, D. Sedlacek, S. Jinchao, J. Dingliana, and S. Collins. Adding depth to cartoons using sparse depth (in)equalities. *Computer Graphics Forum*, 29(2):615–623, 2010.

[26] T.-P. Wu, C.-K. Tang, M. S. Brown, and H.-Y. Shum. Shapepalettes: Interactive normal transfer via sketching. *ACM Trans. Graph.*, 26(3), July 2007.

[27] B. Xu, W. Chang, A. Sheffer, A. Bousseau, J. McCrae, and K. Singh. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM Trans. Graph.*, 33(4):131:1–131:13, July 2014.

[28] C.-K. Yeh, P. Song, P.-Y. Lin, C.-W. Fu, C.-H. Lin, and T.-Y. Lee. Double-sided 2.5d graphics. *IEEE Transactions on Visualization and Computer Graphics*, 19(2):225–235, 2013.