2D Shading for Cel Animation

Matis Hudon
mhudon@scss.tcd.ie
V-SENSE, School of Computer Science and Statistics, Trinity College Dublin
Dublin, Ireland

Rafael Pagés
pagesscr@scss.tcd.ie
V-SENSE, School of Computer Science and Statistics, Trinity College Dublin
Dublin, Ireland

Mairéad Grogan
mgrogan@scss.tcd.ie
V-SENSE, School of Computer Science and Statistics, Trinity College Dublin
Dublin, Ireland

Jan Ondřej
jan.ondrej@scss.tcd.ie
V-SENSE, School of Computer Science and Statistics, Trinity College Dublin
Dublin, Ireland

Aljoša Smolić
smolica@scss.tcd.ie
V-SENSE, School of Computer Science and Statistics, Trinity College Dublin
Dublin, Ireland

Figure 1: ***Example of semi-automatic shaded animation. After setting up the light position to the left side of the character, the user specifies where he wants the shades to appear via single clicks in the first frame, then the tool can propagate the shading throughout the whole sequence. After the propagation, the light can still be modified as well as the shade color and the whole sequence is affected by the modifications. (Original drawing by Matis Hudon, used under CC BY.)***

ABSTRACT

We present a semi-automatic method for creating shades and self-shadows in cel animation. Besides producing attractive images, shades and shadows provide important visual cues about depth, shapes, movement and lighting of the scene. In conventional cel animation, shades and shadows are drawn by hand. As opposed to previous approaches, this method does not rely on a complex 3D reconstruction of the scene: its key advantages are simplicity and ease of use. The tool was designed to stay as close as possible to the natural 2D creative environment and therefore provides an intuitive and user-friendly interface. Our system creates shading based on hand-drawn objects or characters, given very limited guidance from the user. The method employs simple yet very efficient algorithms to create shading directly out of drawn strokes. We evaluate our system through a subjective user study and provide qualitative comparison of our method versus existing professional tools and state of the art.

Expressive '18, August 17–19, 2018, Victoria, BC, Canada

ACM ISBN 978-1-4503-5892-7/18/08…$15.00
https://doi.org/10.1145/3229147.3229148

CCS CONCEPTS

• **Computing methodologies** → **Animation**; **Image-based rendering**; *Non-photorealistic rendering*; • **Human-centered computing** → *User interface design*;

KEYWORDS

2D Shading, Cel Animation, Hand-drawn Animation, Self-Shadowing, Image-Based Rendering

ACM Reference Format:
Matis Hudon, Rafael Pagés, Mairéad Grogan, Jan Ondřej, and Aljoša Smolić. 2018. 2D Shading for Cel Animation. In *Expressive '18: The Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering, August 17–19, 2018, Victoria, BC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3229147.3229148

1 INTRODUCTION

Although arduous to create, hand-drawn animation is still a significant art communication medium. The freedom of expression brought by manual drawing continues to draw expert, professional and casual creatives. Furthermore, new affordable mediums of digital creation such as tablets, touch screens or pen compatible displays, give new impetus to both casual and professional digital hand-drawn art. Spatial arrangement, perspective and temporal coherency are daunting challenges that are hard to tackle when drawing frame by frame animation. Nevertheless, the freedom it brings remains unchallenged and hand-drawn animation still has a long life ahead of it. However, recently the pace of digital creation has risen considerably. In order to exist, artists, whether professional or casual, have to be highly productive which is not quite compatible with the long-term endeavour that producing a hand-drawn animation can be. For these reasons, there is a need for interactive tools, such as [Feng et al. 2016; Henz and Oliveira 2017; Sýkora et al. 2011, 2009a,b; Xing et al. 2014, 2015; Yeh et al. 2015], supporting the creation of hand-drawn digital art and animation.

Besides bringing appeal and style to animations, shades and shadows provide important visual cues about depth, shapes, movement and lighting [Kersten et al. 1997; Petrović et al. 2000; Wanger et al. 1992]. Shading a hand-drawn animation manually is challenging as it requires not only a strong comprehension of the physics behind the shades, but also spatial and temporal consistency respectively within and between the different frames. The two basic components required to illuminate a point are the light position with respect to the point and the surface normal. In hand-drawn artwork, the surface normal is unknown and only the 2D position of the point is available (there is no depth information).

In this paper we propose a semi-automatic method for shading and creating local self-cast shadows in a hand-drawn animation. Our system provides an easy light positioning tool and creates shades based on hand-drawn objects or characters, the user just has to click/tap/touch where he wants a shade to appear on the drawing. Lighting can be modified after the shades are drawn, and even animated throughout the frames. The benefits of our system are an increased flexibility and the reduction of the effort required to create plausible shades and local self-shadows in a hand-drawn animation. As our method remains completely in the 2D space, it maintains the 2D feel of animating by hand.

2 PREVIOUS WORK

2.1 Shading, shadows and illumination for cel animation

Most prior works in this area rely on reconstructing 3D geometry. Works like Teddy [Igarashi et al. 1999] provide interactive tools for building 3D models from 2D data, automating the "inflation" to a 3D model. Petrovic's work [Petrović et al. 2000] applies this idea to create shades and shadows for cel animation. While this work reduces the labor of creating the shadow mattes compared to traditional manual drawing, it also demonstrates that a simple approximation of the 3D model is sufficient for generating appealing shades and shadows for cel animation. However, it still requires an extensive manual interaction to obtain the desired results. Instead of reconstructing a 3D model, Lumo [Johnston 2002] only estimates surface normals by interpolating from the line boundaries to render convincing illumination, however this normal estimation is not fully automatic and requires a separation of drawing contours in different mattes and also makes use of an inner-outer line representation to determine boundary conditions. Later, further improvements were made such as handling T-junctions and cups [Karpenko and Hughes 2006], also using user drawn hatching/wrinkle strokes [Bui et al. 2015; Jayaraman et al. 2017] or cross section curves [Shao et al. 2012; Tuan et al. 2017] to guide the reconstruction process. Recent works exploit geometric constraints present in specific types of line drawings [Pan et al. 2015; Schmidt et al. 2009; Xu et al. 2014], however, these sketches are too specific to be generalized to 2D hand-drawn animation. In TexToons [Sýkora et al. 2011], depth layering is used to enhance textured images with ambient occlusion, shading, and texture rounding effects. Recently, Sýkora et al. [2014] make use of user annotation to recover a bas-relief with approximate depth from a single sketch of a hand-drawn character, which they use to reconstruct a full 3D model and then apply global illumination effects while still retaining the 2D style of 2D animation. While Sýkora shows an example of how toon-shading can be applied to their 3D estimated models, the method is more oriented to support the production of a richer look for traditional hand-drawn animation as opposed to our work which aims at mimicking manual shading.

2.2 Non-realistic rendering

Given a 3D model, global illumination rendering can give an overly realistic shading style to 2D-based animation. Petrovic et al. [2000] stated that, in traditional animation, hand-drawn shadows and shades are often abstract rather than realistic and proposed to simplify the rendered shadows as a post process to their 3D inflating pipeline. Several works try to simplify global illumination shading to match the style of cartoon animation. Non-realistic cartoon like rendering has inspired a lot of work [Anjyo et al. 2006; Barla et al. 2006; Lee et al. 2007; Todo et al. 2007]. Recently Fišer et al. [2016] proposed to shade a 3D model mimicking the shade drawn by a user on a simple sphere.

2.3 Image registration and propagation

Sýkora and colleagues [Sýkora et al. 2011, 2009a] use an image-based approach to register and morph cartoon frames. Their approach works well for rigid pose changes but is not designed to handle the precise interpolation of details. Whited et al. [2010] proposed to find stroke correspondences using a set of user-guided semi-automatic techniques to guide automation of tight inbetweening.

At the origin of our idea is the observation that state-of-the-art automatic shading methods usually rely on a very complex pipeline, whereas manual, but still appealing, shading is often quite simple. A pipeline such as: building a 3D model from 2D drawn art (for each frame), then rendering using 3D global illumination to finally simplify rendered shadows to match the animation hand-drawn style, sounds like using a sledgehammer to kill a fly. By relying only on 2D-based algorithms, no 3D model or normal map estimation, our system bypasses the major state-of-the-art work-flow, trying

to preserve the intuition-based creative process of hand-drawn animation and is still able to produce appealing and plausible self-shadows and shades.

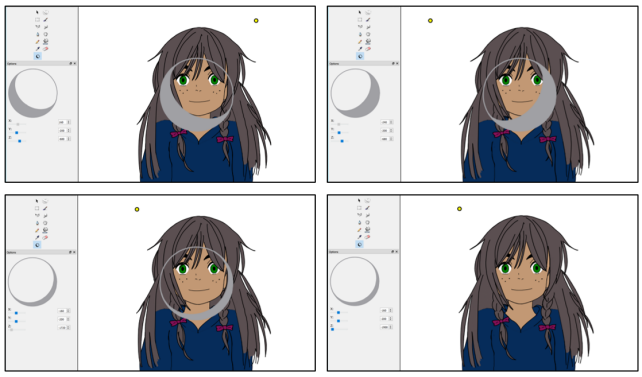

Figure 2: Light positioning. While the user freely moves the light in the 3D space using the X, Y and Z sliders the system overlays a preview sphere, shaded as if it was lit from the current light position. This allows the user to directly evaluate how the shade would look on his drawings and simply position the light with a more artistic approach. Top Row, the light is moved along the X axis, changing the preview shade direction. Bottom left, when moving the Z slider the user directly plays with the light inclination, changing the preview shade span. Bottom right, when the user is not moving the light, the preview sphere disappears. (Original drawing by Matis Hudon, used under CC BY.)

3 INTERFACE

We developed our semi-automatic shading tool on top of Pencil2D animation software. Pencil2D is a free and open-source hand-drawn animation software developed in C++ and Qt. Pencil2D has the standard visual appearance of a keyframe animation authoring tool. With basic features of Pencil2D users can draw in one or several layers and play the frames as an animation. We added our tool to the collection of tools Pencil2D already provides. When selecting the shading tool, a light and a shading layer are added to the scene. Light can either be moved by clicking in the canvas when the light layer is selected, or using the X, Y, Z sliders provided in the tool options panel. To ease the light positioning, a live shade previewing sphere is overlaid when moving the light (Figure 2). This preview sphere is shaded as if it was lit from the current light position; this provides a less physical but more artistic and intuitive way of positioning the light, based on the preview. Rather than positioning the light with respect to 3D space, our method allows the user to adjust the light position based on what the shades would look like. On the drawing layer, the user can add shades via a 2D position input, such as pen press, finger touch or mouse click depending on the interface available (from now these 2D positions will be referred to as user clicks). The user can also create local self-cast shadows by clicking on the surface casting the shadow and dragging to the surface receiving the shadow. Shades will be rendered in the shading layer and can be propagated throughout all frames in the sequence.

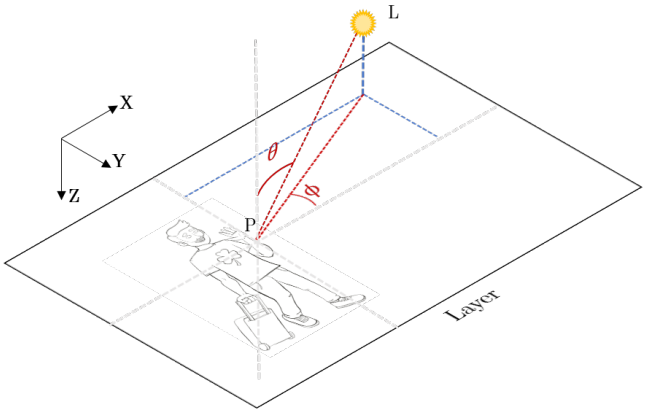

Figure 3: Light coordinates representation. (Original drawing by Rafael Pagés, used under CC BY.)

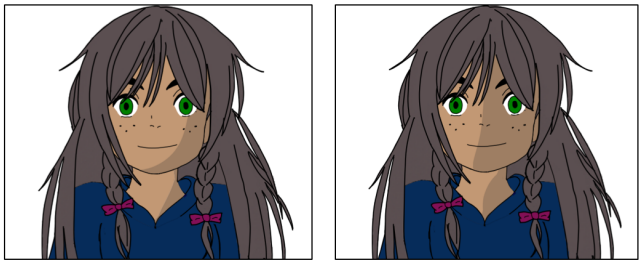

Figure 4: The effect of inclination θ on shading. On the left a low inclination is used: the light is almost facing the character; on the right the inclination is higher: the light is coming more from the left side. (Original drawing by Matis Hudon, used under CC BY.)

4 PROPOSED TECHNIQUE

This section describes our process for creating shade contours and local self-cast shadows directly from hand-drawn art, and propagating results through the sequence. The process is mainly stroke-based and remains completely in the 2D domain. Our system was implemented for vector images. We make the common assumptions that the line drawings are clean and shapes were carefully closed by the artist. Under these assumptions, our method could be adapted to raster images.

4.1 Light

As illustrated in Figure 3, we make use of a spherical coordinate system to represent the light position with respect to any point in the drawn layer. A light L can be positioned in the 3D space using the Cartesian coordinate system at (X_L, Y_L, Z_L). Then for any point $P(X_P, Y_P, 0)$ in the drawn layer we can specify the inclination θ and the azimuth ϕ. The azimuth ϕ represents the incoming light direction as projected on the canvas. The inclination θ will be directly responsible for the shade size: a low inclination implies that the light direction is almost orthogonal to the drawing, therefore the shade contour will be very narrow, whereas a higher inclination implies a wider shade (Figure 4).

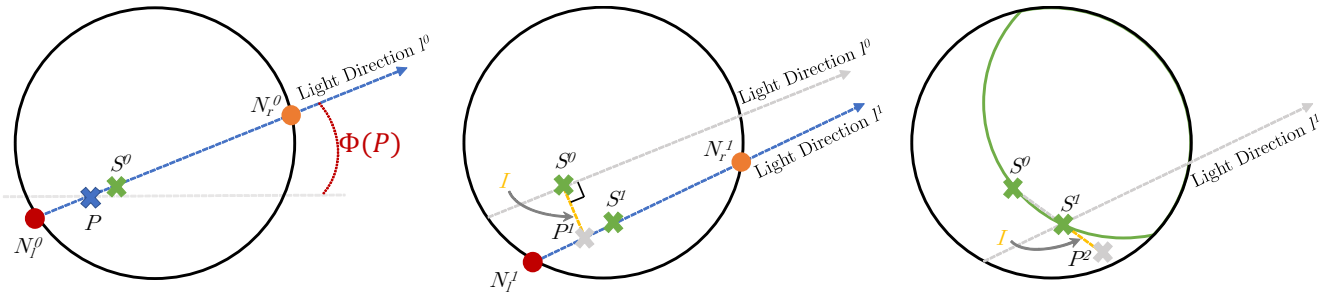

Figure 5: Algorithm overview of shade construction. Left: Given the click P, the first shade border point S^0 is found on the line segment $[N_f^0 N_n^0]$ taking into account the local inclination θ. Middle: First iteration towards the bottom: the next search point P^1 is found using a direction orthogonal to l^0 and the shade border point S^1 is found on the line segment $[N_f^1 N_n^1]$ taking into account the local inclination θ. Right: All remaining search points are computed using the direction spawned by the last two shade border points found, for example P^2 is found along the direction $\overrightarrow{S^0 S^1}$.

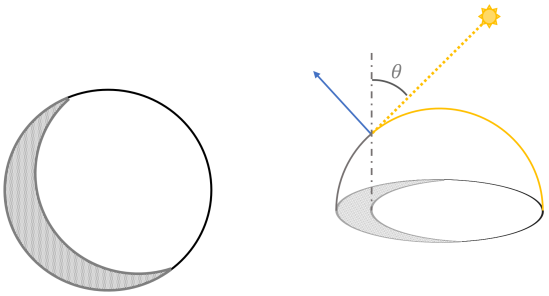

Figure 6: On the left: an example of shading for a drawn sphere. On the right: a 3D representation giving an insight into how our tool might be thought of. The boundary between light and shade is the 2D projection of the 3D sinus blob profile line [Johnston 2002] whose normals are orthogonal to the light direction.

Once shades and local self-shadows are rendered, the user can still adjust the lighting conditions to perfectly match his needs. As the process of creating a shade line is straightforward and fast, every time the light position is changed, every shade or local self-shadow affected is reconstructed.

4.2 Shading

It is natural to assume that each shape of a drawing has an implicit 3D curved surface, as shown in Figure 6. In [Johnston 2002], it is supposed that for curved surfaces, points where normals are orthogonal to the eye vector (i.e. in the drawing plane) are on the exterior silhouette. Therefore, the profile of the 3D shape can be approximated as a sinus blob whose height is directly related to the local width of the 2D shape, as shown in Figure 6. This representation gives an insight into how our tool might be thought of. However, our method completely remains in the 2D space and thus does not compute any 3D blob or height. The idea is to directly estimate the shade borders using only the local 2D geometry and the light inclination θ.

4.2.1 Shade line construction. Having specified light coordinates, the next step is to create shade contours for the hand-drawn art. Our goal here is to find a set of control points describing an enclosing area for our shade. Figure 5 shows an example of our algorithm when applied to a drawn circle. The start point (*seed point*) is the user input click P. One of the strengths of our method is that this point is arbitrary as long as it is within the same enclosed region of the drawing (e.g. within the circle in Figure 5). Let l^0 be the line representing the light direction as projected on the drawing plane, passing through P. We define two points called *far neighbor* N_f^0 and *near neighbor* N_n^0, as the two first intersections of l^0, with the drawing strokes, starting from P in both directions (see Figure 5). Note that, whatever the drawing or the light position, the far neighbor is always taken as the furthest of the two intersections with respect to the light, inversely the near neighbor is the nearest of the two intersections with respect to the light. The distance between these two points represents the local width of the element to shade. We compute the shade border point S^0, on the line segment $[N_f^0 N_n^0]$, following Equation 1 with $i = 0$:

$$S^i = N_f^i + (1 - \cos(\theta)) \left(\frac{N_n^i - N_f^i}{2} \right) \tag{1}$$

where θ is the local inclination of the light. When $\theta = 0$, light is directly facing the object, thus there is no shade and $S = N_l$. When $\theta = \pi/2$, light is coming from the side and $S = (N_n + N_f)/2$.

Once the first shade border point S^0 has been found, we look for the second search point $P^{\pm 1}$ using a fixed search interval I, perpendicular to l^0 (in both directions, *top* P^{-1} and *bottom* P^1). Then we can compute shade border points $S^{\pm 1}$ using Equation 1. For all remaining shade border points $P^{\pm i}$, we use the direction spawned by the last two shade border points found $S^{\pm(i-1)}$ and $S^{\pm(i-2)}$, with the same search interval I. In order to obtain a correct sampling of shade points S^i, the interval length (in pixels) used is directly proportional to the length of $[N_f^0 N_n^0]$ (10 times smaller in our experiments) and is adapted throughout the iterative process.

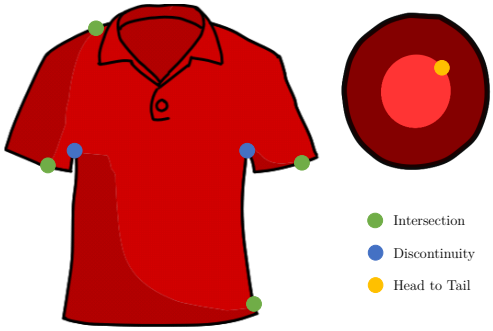

Figure 7: Stopping criteria illustrated. (Original drawing by Matis Hudon, used under CC BY.)

4.2.2 Stopping criteria. We make use of three stopping criteria: *intersection, discontinuity* and *head to tail*. For the first one, as expected, the shade border line should not extend beyond the edges of the surrounding drawn shape. Therefore, shade border control points S^i cannot be outside of this shape. For the second criterion, an example can be seen on the right sleeve of the T-shirt in Figure 7 (blue point). Here, the shape silhouette is discontinuous around the blue point, so constructing the shade (upward) after this point, would result in a very strange and unrealistic shade. More generally, the shade construction should stop when there is a large discontinuity between the far neighbor construction points N_f^i. Whenever we add a new far neighbor N_f^i during the construction, we compute the distance $d^i = \left\| N_f^i - N_n^{i-1} \right\|$. This distance is then compared to the previous one $d^{i-1} = \left\| N_f^{i-1} - N_n^{i-2} \right\|$. We consider that there is a discontinuity, if $d^i > \beta \times d^{i-1}$, where $\beta = 3$ in our experiments. The last stopping criterion occurs if the shade line intersects itself (*head to tail*), occurring in particular when the light is placed close and in the middle of the object (see right of Figure 7). Whenever a stopping criterion is met, the length of the search interval I is recursively decreased by two to improve the sampling precision (the interval I cannot be smaller than 1 pixel).

4.2.3 Final steps. In order to deal with small discontinuities and obtain a clean and smooth shade border, we lightly filter the positions of our shade border control points, applying a 1D bilateral filter along the shade points. Letting S be the ordered set of control points, the i^{th} point is filtered as follows:

$$S_f^i = \sum_{k=i-\eta}^{i+\eta} \exp\left(-\frac{\left\| S^i - S^k \right\|_2}{\sigma_1^2} - \frac{|\, i-k \,|}{\sigma_2^2} \right) \tag{2}$$

where σ_1 and σ_2 are standard deviation parameters, and η is half of the filtering window size. In our experiments $\sigma_1 = 11$, $\sigma_2 = 7$ and $\eta = 2$. Finally, the last step of our algorithm is to use the shade border points S^i to form a closed area to colorize. We add (in the right order) the far neighbor points N_f^i, found during the construction process, to the shade control points S^i, forming a set of control points that we inter-connect with a Bezier curve to form the shade area. The color and transparency of the shade can be selected/changed by the user.

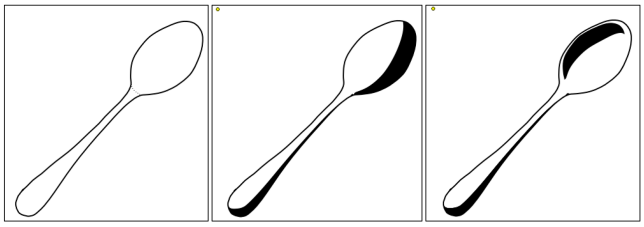

Figure 8: Simple concave example. From left to right, input line drawing, convex shading of the spoon and concave shading of the spoon. In both cases light is coming from the yellow dot in the upper left corner. (Original drawing by Matis Hudon, used under CC BY.)

4.3 What About Concavity?

The method previously described works for convex surfaces. However, it is sometimes necessary to handle concave surfaces. Concave and convex shapes can have the same line drawing. For example in Figure 8-left, it is impossible to determine, even for a human, whether the spoon is meant to be seen from the front or the back. Determining the concavity or convexity of a shape just by looking to its line drawing is an under-constrained problem that cannot be tackled without additional human input. Provided with this input, our system can easily be adapted to create concave shading as shown in Figure 8. In this case, we modify Equation 1 to use near neighbors instead of far neighbors as follows:

$$S^i = N_n^i + (1 - \cos(\theta)) \left(\frac{N_f^i - N_n^i}{2} \right) \tag{3}$$

When generated this way, the concave shade could just looks like a convex shade on the wrong side, in our experiments we found that accentuating this concavity effect was giving more insight about the underlining shape. In order to accentuate the concavity effect we translate shade control points towards the barycenter B_s of far and near neighbors by a distance depending on the relative local thickness of the shade, therefore the smaller the relative thickness the longer the translation distance. As an example, control points S^i are translated along the segment $[S^i B_s]$, towards B_s by a distance:

$$distance^i = \left(\zeta + \gamma \left(1 - \frac{\left\| N_n^i - S^i \right\|}{M_T} \right) \right) \left\| S^i - B_s \right\| \tag{4}$$

where the scaling factor $\gamma = 0.1$ and the constant $\zeta = 0.05$ in our experiments and M_T is the maximum thickness $\left\| N_n^i - S^i \right\|$ found in the set S.

4.4 Local Self-Cast Shadows

Self-cast shadows occurs when an area of a drawing is casting a shadow onto another area. This implies, within the meaning of 3D, that the area casting a shadow is in front of the one receiving it, with respect to the light position. Self-cast shadows play an important role in creating plausible shades and shadows, giving consistency to the scene as shown in Figure 9.

Our process for creating local self-cast shadows is straightforward. The user can create local self-cast shadows by clicking on the area casting the shadow (the one in the front) and dragging and

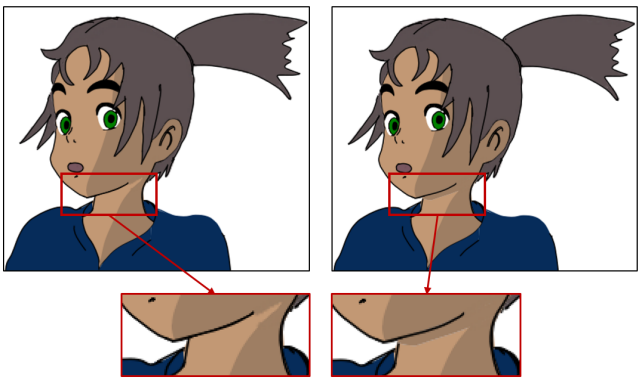

Figure 9: Example of the same drawing with and without self-shadowing (top), magnification of the interesting areas (bottom). (Original drawing by Matis Hudon, used under CC BY.)

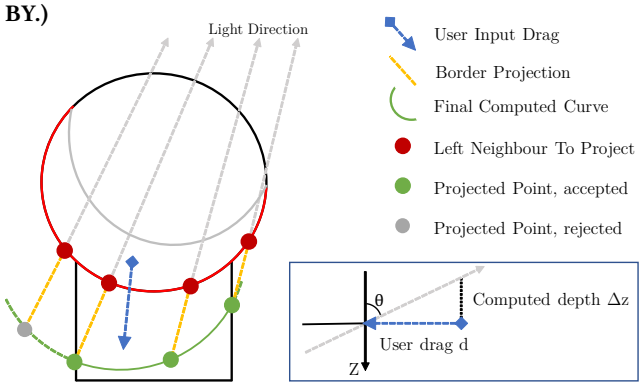

Figure 10: Self shadowing construction overview. In this example, the circle area is casting a shadow onto the squared area.

releasing in the area receiving the shadow. For example, in Figure 9 the user would click on the chin, drag to the neck and release. The wider the dragging segment the wider the shade casting. The local self-cast shadows algorithm relies heavily on the shading algorithm presented Section 4.2, as described below.

4.4.1 Border shadow casting. The profile of the cast-shadow depends on the profile of the area casting the shadow. As an example, in Figure 9, the profile of the cast shadow on the neck is related to the profile of the chin. This profile can be calculated using the shade construction algorithm described in Section 4.2. Indeed, the far neighbors found when constructing a shade line, using the first point of the user drag as seed, are good representatives of the edge of the casting area (Figure 10). To create the shadow we need to find the projections of those points and determine whether they fall into the receiving area or not.

4.4.2 Depth difference computation. If an area casts a shadow onto another, it means it is closer to the light source. The difference between the distances to the light source of the two areas plays an important role in the final appearance of the shadow: the larger this difference is, the larger the shadow. However, a 2D drawing alone does not convey directly how much closer the casting area is. This is why we let the user adjust this depth difference Δz by taking into account the length of his drag d:

$$\Delta z = \frac{d}{\tan\theta} \tag{5}$$

where $\theta \in]0, \frac{\pi}{2}[$ is the angle of light inclination with respect to the point at the intersection of the user input segment and the stroke between the area casting the shadow and the one receiving it.

Using the depth difference Δz we can now project the points from the border. If a projection falls into the area to be shaded, we accept the point, if not we reject it. Finally, the shadow area is closed using the border points whose projection was accepted during the construction process.

5 PROPAGATION

Propagation is a key element to reduce the workload in traditional animation and motivated a lot of work in areas such as inbetweening [Whited et al. 2010] and image registration [Sýkora et al. 2009a]. However, as our semi-automatic shading tool computation is fast (on average 148 milliseconds for a 56 control points shade line), no complex image registration is needed to propagate the shades from one frame to the next. Remember that, in order to create a shade line (Section 4.2), our algorithm requires a user input click inside a 2D area in the sketch. As stated before, this input is not strictly constrained: any 2D position will create a similar shade, as long as it is located inside the 2D area. Consequently, the input click position within the 2D area can vary and does not influence the shade line result. Considering this remark, propagating a shade line from a 2D area can rely on finding just one point in the following frame, which is located anywhere within the corresponding area. This point is then used as the new seed. Our propagation algorithm relies on simple assumptions and is very robust to motion and non-rigid deformations, which is one of the main issues in alternative methods based on image registration. Furthermore, as consecutive propagated shades are computed in each frame, temporal coherency is ensured by the consistency of the artist's hand drawn frames rather than by a complex spatio-temporal filtering method. For the sake of clarity, this section presents our propagation as applied to shades, however a similar algorithm can be applied to shadows. Objects to be shaded in consecutive frames must maintain the same color as well as some spatial and temporal coherency. The closer the inbetweens are the better the propagation result will be. To propagate a shade to the next frame, our method only needs to propagate the seed point (similar to color seed points in [Sýkora et al. 2009b]), that is finding a seed point that generates the right shade in the next frame. From here, the idea is to generate a list of points (seed candidates), that have a high probability of being in the right shape in the next frame (Figure 11).

5.1 Generation of Seed Candidates List

Our objective is to compute a set of points (called seed candidates), which have a high probability of being located inside the right area of the next frame. As an example, in Figure 11 the objective is to propagate the shade of the arm from *frame n* to *frame n* + 1 (i.e.

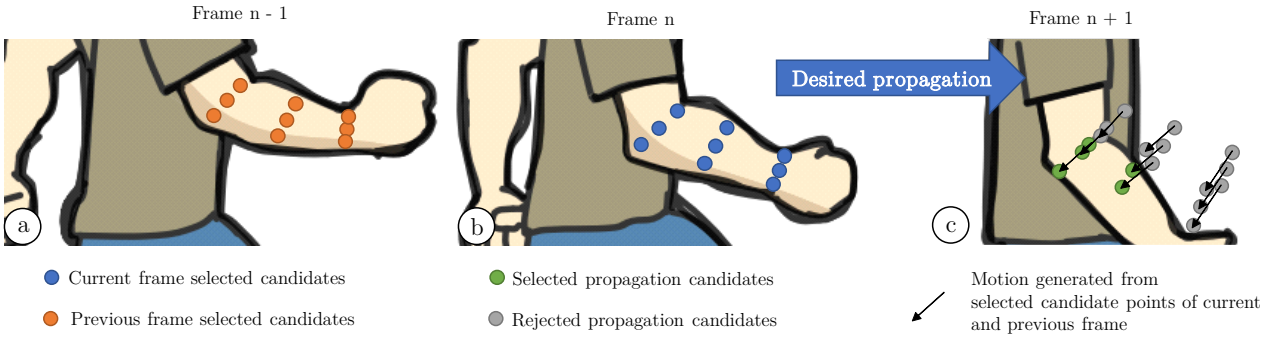

Figure 11: Example of the selection of propagation seed points when propagating from *frame* n (b) to *frame* $n+1$ (c). 9 candidates are directly generated from the current *frame* n (b) and 9 other points are computed thanks to motion vectors between the previous *frame* $n-1$ (a) and the current *frame* n (b). Finally, in the next *frame* $n+1$ candidates are selected (green) or rejected (grey) via color matching (c). (Original drawing by Rafael Pagés, used under CC BY.)

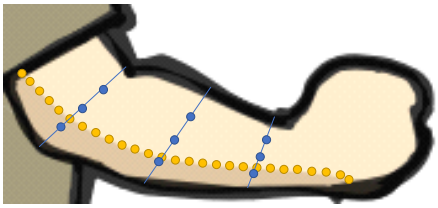

Figure 12: Candidate local selection along shade control points. The yellow points are the shade border control points S^i and the blue points are the selected candidates. (Original drawing by Rafael Pagés, used under CC BY.)

finding points on the arm in *frame* $n+1$). Typically, in two consecutive frames, corresponding areas are spatially close or even overlap. Therefore, a reasonable sampling of the area where the shade is located in *frame* n has a high probability of containing good seed candidates. Remember that a shade line is defined through a list of construction points composed of far neighbor N_f^i, shade point S^i and near neighbor N_n^i. We make use of these shade construction points to sample the area around a shade. We first choose the three shade points which divide the shade line into equally sized sections as in Figure 12. For each of these shade points, we extract three candidates by dividing the segment between left and right neighbors into three equally sized sub-segments, as shown in Figure 12. These nine points (Figure 11b) cover a reasonably wide area in the surrounding shape. We could extend this computation to a higher number of construction points, but following experiments we found that nine is a good compromise between accuracy and efficiency. Nine other points are computed through motion prediction by taking the difference between two corresponding candidate positions in the current (Figure 11b) and previous frame (Figure 11a). Candidate seed points whose color does not match with the previous color, exact same color in our implementation, are rejected (grey points in Figure 11c).

5.2 Selecting One Candidate

Let C be our set of candidates, s the shade we want to propagate from *frame* n, A the area around s and A^{+1} the corresponding area in the next *frame* $n+1$. Candidates in C have a high probability of being in A^{+1}. However, we only need to find one seed point in A^{+1} for the propagation. To find a good seed point among the set of candidates C, we process them one by one until a correct seed point is found. Let c_k be our k^{th} candidate and B_k the area around it in *frame* $n+1$. We suppose that *frame* n and *frame* $n+1$ are temporally and spatially close, thus we can make the assumption that A and A^{+1} have similar shapes. To validate that B_k is actually A^{+1}, we designed a shape/area similarity metric robust to small non-rigid deformations (very frequent in hand-drawn animations). As stated before, construction points of a shade are good representatives of the surrounding area. Therefore, we first compute the shade s_{c_k} using c_k as a seed point in *frame* $n+1$. We measure the similarity Π between the two areas A and B_k by comparing the two series of distances between right and left neighbor construction points, $\left\|N_l^i - N_r^i\right\|$, for s and s_{c_k}. If the two areas are similar enough, we can then consider that c_k is a good candidate (i.e. $B_k = A^{+1}$). As the size and shape of areas may vary from one frame to another, the number of shade control points might not be the same. We define our similarity Π as the minimum root mean squared error found when sliding the shorter set of right-left distances Ω_s over the longer set of right-left distances Ω_l. Let n_s and n_l be respectively the cardinals of S_s and S_l:

$$\Pi = \min\{RMSE_\tau(\Omega_s, \Omega_l), \tau \in [0, n_l - n_s]\} \tag{6}$$

where τ is a sliding offset.

$$RMSE_\tau(\Omega_s, \Omega_l) = \sqrt{\frac{1}{n_s}\sum_{i=1}^{n_s}(\Omega_s(i+\tau) - \Omega_l(i))^2} \tag{7}$$

Finally, we select a candidate providing Π does not exceed a maximum acceptable threshold ξ:

$$\xi = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\alpha \times \Omega(i))^2} \tag{8}$$

where Ω is the set of right-left distances of s, n is the cardinal of Ω and α is a fixed coefficient, in our implementation $\alpha = 0.02$, which represent an acceptable error of 2% along right-left distances representing A. In the rare cases where no correct seed point is found among all candidates, the propagation process stops, giving the user the opportunity to input a 2D point manually.

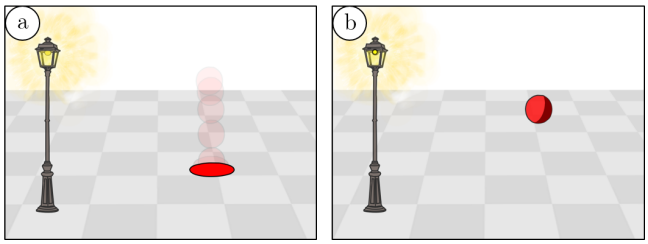

Figure 13: Bouncing ball scene: (a) Onion skin view of the animation, (b) One frame shaded by our shading tool. (Original drawing by Matis Hudon, used under CC BY.)

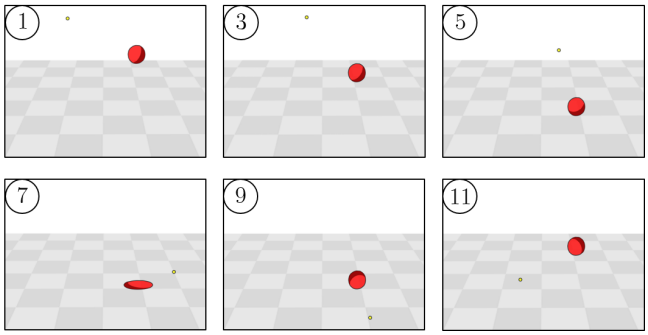

Figure 14: Bouncing-ball free session: light position animation of one participant (One frame over two 1-11). (Original drawing by Matis Hudon, used under CC BY.)

6 USER EVALUATION

We conducted a user experiment in order to evaluate the utility and usability of our system by comparing fully traditional manual shading to our semi-automatic shading system.

6.1 Experiment

A total of 12 users participated in our experiment, among them 3 were skilled in drawing and/or animation and 9 were new to the field. All tasks were conducted using a Wacom Cintiq 21UX pen display, which provides the closest digital drawing experience to pen and paper with almost no adaptation period. After a short warm-up session designed to familiarize the user with the pen tablet, Pencil2D animation software and basics about shading in cel animation, the experiment is split into three different parts. The bouncing ball experiment (12 min), the walking cycle experiment (12 min), and the final interview (2 min).

In the first part a simple bouncing ball animation scene is provided to the participants. In the scene, a lamp post and the ball have been drawn and placed in the scene with respect to a grid patterned floor background, giving the illusion of depth (Figure 13). First, the user is asked to manually draw the shade of the ball with respect to the lamp post position, for each of the 11 frames of the animation cycle. The second task is to create the same scene, this time using our semi-automatic shading system. In the third task, the lamp post is removed, and participants can freely try to animate the light position over the whole animation cycle (Figure 14). This experiment allows us to measure the effectiveness of our method

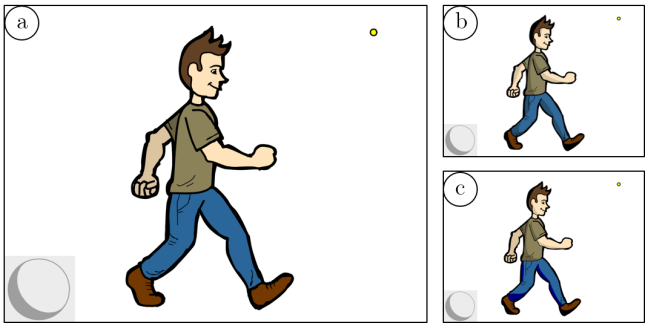

Figure 15: The walking cycle scene: (a) Original frame to shade, (b) Example of manual shading by a participant, (c) Example of semi-automatic shading by the same participant. (Original drawing by Rafael Pagés, used under CC BY.)

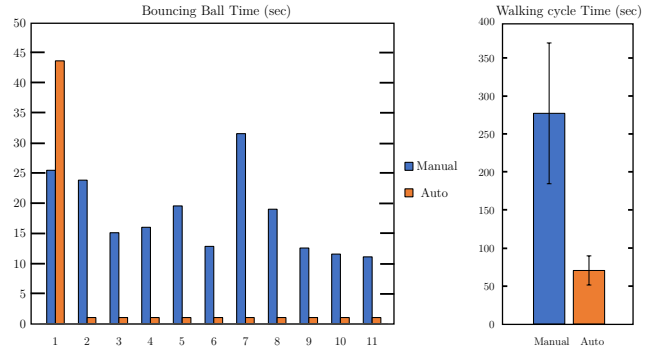

Figure 16: Performance results. Average timings for the bouncing-ball experiment (left). Average timings for the walking-cycle experiment ± standard deviation (right).

on a simple scene (11 frames), giving us an actual measure of the reduced workload of our method versus manual editing. Also, users get a chance to experience our light positioning system interface.

In a second this experiment the user is asked to shade a human cartoon character in a walking cycle, as shown in Figure 15. The participants are first asked to draw the shades manually on the first frame and then using our semi-automatic tool. The second task is a free session: participants are free to try the propagation tool over the eleven frames of the animation and also free to modify the light. This experiment allows us to measure the effectiveness of our method on a more complex scene.

6.2 Outcome

Figure 16 provides measurements of average completion time for the bouncing ball shading tasks with and without our semi-automatic shading tool. Timings include drawing and colorizing the shade. For a simple scene such as the bouncing ball with only one shade to draw per frame, as it can be seen in Figure 16, our method takes longer on the first frame as there is an additional task consisting of positioning the light in the scene. However the propagation takes less than one second per frame. On average, total completion time for the manual shading was 198 seconds whereas it was only 55

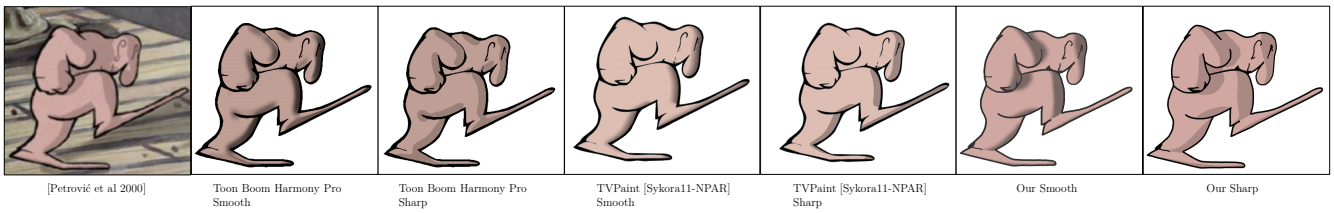

Figure 17: Qualitative shading comparison between [Petrović et al. 2000], our method(<20sec), Toon Boom Harmony pro© tone shader (6-7min) and TVPaint Pro© Toon shader [Sýkora et al. 2011] (<20sec). The Stomping Man drawing was reproduced from [Petrović et al. 2000] with authors' consent. Light was approximately placed to mimic the lighting from the original artwork. For our result, shapes suggested by the drawing, such as the two ankles and the shoulder, have been closed with an invisible line. The smooth version of our result has been post-processed for comparison purposes.

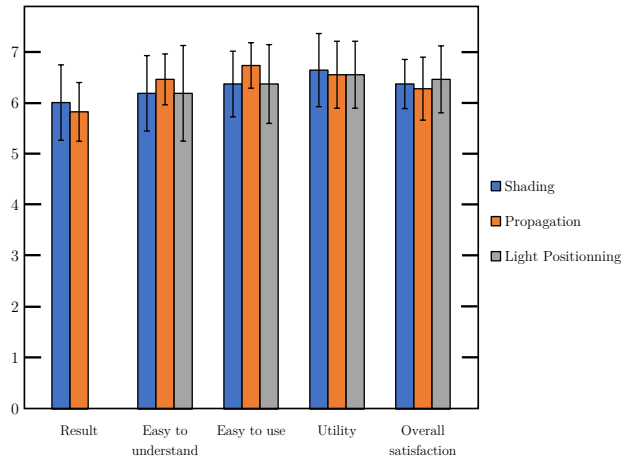

Figure 18: User feedback. All quantities are expressed as mean ± standard deviation in a 7-point Likert scale.

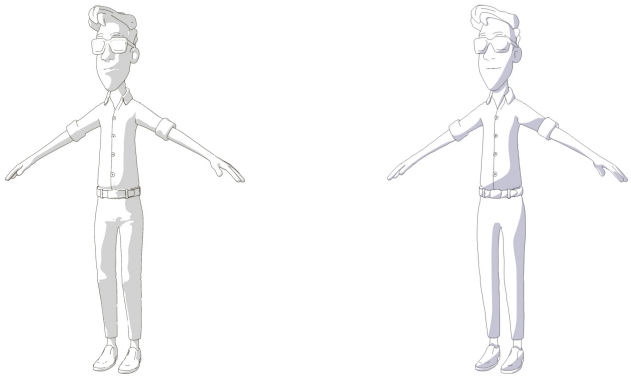

Figure 19: Visual comparison of Classic Toon-Shading versus our shading result. A toon shading obtained from a 3D model (left) and our shading result (right). The line drawing was obtained from a 3D model with sketch-like rendering (Freestyle plugin in Blender ©). (3D model is an original creation by Andy Goralczyk, used under CC BY.)

seconds for the semi-automatic shading. The ratio of completion time by our system versus manual drawing is 3.64, which greatly demonstrates the labour reduction induced by our system. When asked, participants either preferred the semi automatic shading result over their drawing or stated that the two methods had comparable results. Figure 16 also shows measurements of average completion time of the second tasks. The completion time ratio of 3.92 shows that in the case of a complex scene, with several shades and colors, semi-automatic shading for one frame is a lot faster than manual drawing. Figure 18 outlines the subjective feedback of our participants (in a 7-point Likert scale) about the results and the different functionalities of our semi-automatic shading tool. Overall, participants were satisfied by our system. Participants familiar with using creative tools and software particularly liked the light positioning tool, while novice users had more trouble linking the 3D space position with the preview shaded sphere. We also presented our work to a professional animator who stated that she would gladly use our tool for shading animations, but also digital still artwork. She enjoyed our tool for positioning the light and particularly liked the shade previewing sphere.

7 RESULTS

Figure 21 shows different results of drawn artwork and animation shaded with our system. This validates that our tool can handle different types of drawings and animations. Results show how our system provides good shading lines in most cases.

As stated in Section 2, state of the art methods such as [Sýkora et al. 2014] focus on estimating an accurate 3D model from sketches. Given a 3D model, classical rendering techniques can give an overly realistic shading style to 2D based animation. As stated by Petrovic et al. [2000], in traditional animation, hand-drawn shadows are often abstract rather than realistic. This might be a direct consequence of the lack of efficient tools and the amount of manual labor that would require the creation of realistic shades consistently throughout an entire animation. Nevertheless, simplistic manual shading remains undeniably appealing. Our tool tries to mimic how a human artist would draw a simplistic but appealing shading. Figure 19 shows a visual comparison of our shading versus classical Toon-shading from an artifact free 3D model. Our tool provides a less realistic shading compared to a classic Toon-shader, however it is closer to a traditional cartoon shading style.

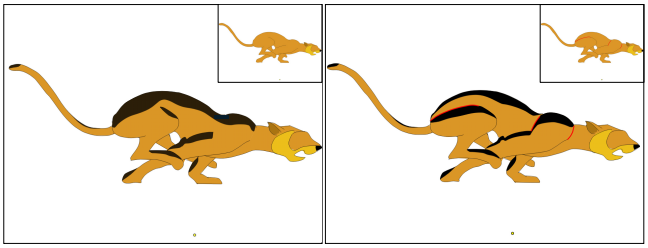

Figure 20: Shading with (right) and without (left) invisible line comparison. Invisible lines were highlighted in red for this figure. In both cases, light is coming from the yellow dot. (Source drawing by Esther Huete, used under CC BY.)

We also show a qualitative comparison of our method versus several state of the art methods in Figure 17. We used the Stomping Man scene from [Petrović et al. 2000]. We shaded the scene with two of the most widely used animation softwares: Toon Boom Harmony Pro© and TVPaint Pro©. Toon Boom Harmony provides a very complete interface to create a normal map from a 2D drawn art. As the number of customisable parameters is large, the software has the potential for creating very high quality normal maps and shading, however this is at the cost of simplicity and speed. Note that much higher quality can be obtained by splitting the different parts of the character into several layers, and inflating them one by one, however doing this for each drawn frame can easily become very tedious. The sharp and smoothed results can be seen in Figure 17 and took approximately 6-7 minutes to create. TVPaint Pro provides an automatic and efficient shading tool called Toon Shading which is an implementation of the method from Sýkora's Textoon tool [Sýkora et al. 2011]. The tool almost requires no user interaction (except for light positioning) and the whole process of creating shades is very simple, taking a whole image as input and can be performed in less than 20 seconds (this is the user time, the algorithm itself is very fast - see [Sýkora et al. 2011]). There is also the possibility of editing the shade profile which is very interesting. However, without pre-segmentation and layering of the body parts, inner parts like the arm in Figure 17 are not shaded. We show the results of this method with both a linear profile (smooth) and a discontinuous one (sharp). We show here that our shading method is qualitatively comparable to what can be achieved with state of the art methods in widely used professional animation software. This validates that our semi-automatic system is capable of creating plausible shades and self-shadows.

Our method is faster and easier to use than the one used in Harmony (even without pre-segmentation of body parts). TVPaint and our method can shade a drawn art at comparable speed.

Our method can produce different shading results depending on the input line drawing. Therefore, the shading result can be more detailed by adding invisible lines in the line drawing, as shown in Figure 20. In both cases the shading looks plausible, the inclusion of invisible lines to add more detail to the shading result is more often an artistic choice, to emphasise parts of the drawing (such as the back leg muscle in Figure 20), rather than a compulsory pre-processing step.

8 DISCUSSION AND FUTURE WORK

Although our approach works quite well for many hand-drawn sketches and animations, the current prototype has some limitations that need to be taken into account.

This work presents a method using only one light source but could easily be applied to a scene with multiple light sources. Also, different types of lights such as area lights or point lights could be investigated.

One of the strengths of our method is that the seed point is arbitrary as long as it is within the right portion of the drawing. One could think of generating arbitrary seed points in all of the drawing to create all the shades without requiring any user interaction. While this feature might be interesting, sometimes, some shades are not necessary and we prefer to give more creative possibilities to the artist rather than directly imposing a final result.

The presented implementation assumes hand-drawn objects and animations mainly have rounded shapes. Thus our system cannot automatically handle local discontinuities and sharpness differences. To overcome this issue, the user can always edit the shade curves by moving control points, however the propagation of shade edits through the different frames is not currently possible, but this could be a compelling feature. Also employing more advanced shade edits than simply moving vector control points could be investigated. Another improvement could be to handle details, by for example, deforming shades and shadows where they intersect detail strokes such as the eyes, nose, and mouth but also cloth folds.

In this paper we have shown that our method can be easily adapted to handle concave rounded shapes providing an additional user input. While this feature showed convincing and promising results it is still limited to points of view where the whole shade is visible - occlusions in concave objects are currently not supported.

Also, some failure cases can be seen in Figure 21. For example, the first frame of the Feline sequence shows a slightly wrong shading in one if its back paws. In this case, the problem is that the leg is represented as a long narrow shape; so, when the light is aligned with it, the wrong intrinsic thickness is found and, therefore, the wrong profile. This results in an incorrect shading line. A possible manual solution for this kind of error could be adding an invisible line to separate the leg and paw.

The propagation of the shades through the whole sequence is, in some cases, prone to errors. For example when an object becomes partially occluded during an animation, automatically finding the next shade position is more likely to fail. However, whenever the propagation fails, it stops on the frame triggering the error and lets the user manually input the next shade position. Also, very similar and spatially close objects can make the propagation process fail. For these complex cases the user can run the propagation in safe mode, where propagated shades must be validated manually at each frame.

9 CONCLUSION

This paper presents a new way for the semi-automatic creation of shading and self shadowing. While previous works focused on reconstructing 3D models or normal maps from 2D artwork, the tool tries to directly construct animation style shade lines, while staying in the natural 2D drawing environment. The system yields

Figure 21: Results of artworks shaded with our method. Invisible lines were highlighted in red for this figure. First row: Character in a walking cycle with fixed lighting. Second and third rows: leftmost image pairs are the detail and main drawing layers. Images on the right show different lighting configurations. Fourth and fifth rows: Feline animation with relatively moving light.Last two rows: girl animation with fixed light. Hairs, face details and main body/head are split on three different layers. (Source drawings: Raw 1-2 by Rafael Pagés, Raw 3-5 by Esther Huete, Raw 6 by Matis Hudon; all used under CC BY.)

plausible cartoon style shades and self-shadows. The process drastically reduces the labour of drawing shades and self-shadows by hand. It also brings more flexibility to the user as the lighting can be moved in retrospect. The system was developed for convex shapes but we have shown that it can be easily adapted to concave shapes. We also introduced a new way to position a light in the 3D space without leaving 2D space which was greatly appreciated by experienced users. Moreover, we compared our method to the state of the art and two widely used professional animation softwares. We are convinced that it can motivate future work on non-physical manipulation of shading and shadows.

ACKNOWLEDGMENTS

The authors would like to thank Esther Huete, professional animator, for sharing her original creations as well as for her valuable comments. The authors would also like to thank the anonymous reviewers for their valuable comments and helpful suggestions. This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under the Grant Number 15/RP/2776.

REFERENCES

Ken-ichi Anjyo, Shuhei Wemler, and William Baxter. 2006. Tweakable light and shade for cartoon animation. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*. ACM, 133–139.

Pascal Barla, Joëlle Thollot, and Lee Markosian. 2006. X-toon: an extended toon shader. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*. ACM, 127–132.

Minh Tuan Bui, Junho Kim, and Yunjin Lee. 2015. 3D-look shading from contours and hatching strokes. *Computers & Graphics* 51 (2015), 167–176.

Lele Feng, Xubo Yang, Shuangjiu Xiao, and Fan Jiang. 2016. An Interactive 2D-to-3D Cartoon Modeling System. In *International Conference on Technologies for E-Learning and Digital Entertainment*. Springer, 193–204.

Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. 2016. StyLit: illumination-guided example-based stylization of 3D renderings. *ACM Transactions on Graphics (TOG)* 35, 4 (2016), 92.

Bernardo Henz and Manuel M Oliveira. 2017. Artistic relighting of paintings and drawings. *The Visual Computer* 33, 1 (2017), 33–46.

T Igarashi, S Matsuoka, and H Tanaka. 1999. Teddy: A Sketching Interface for 3D Freeform Design, SIGGRAPH. In *Conference Proceedings), ACM*.

Pradeep Kumar Jayaraman, Chi-Wing Fu, Jianmin Zheng, Xueting Liu, and Tien-Tsin Wong. 2017. Globally Consistent Wrinkle-Aware Shading of Line Drawings. *IEEE Transactions on Visualization and Computer Graphics* (2017).

Scott F Johnston. 2002. Lumo: illumination for cel animation. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*. ACM, 45–ff.

Olga A Karpenko and John F Hughes. 2006. SmoothSketch: 3D free-form shapes from complex sketches. In *ACM Transactions on Graphics (TOG)*, Vol. 25. ACM, 589–598.

Daniel Kersten, Pascal Mamassian, and David C Knill. 1997. Moving cast shadows induce apparent motion in depth. *Perception* 26, 2 (1997), 171–192.

Yunjin Lee, Lee Markosian, Seungyong Lee, and John F Hughes. 2007. Line drawings via abstracted shading. In *ACM Transactions on Graphics (TOG)*, Vol. 26. ACM, 18.

Hao Pan, Yang Liu, Alla Sheffer, Nicholas Vining, Chang-Jian Li, and Wenping Wang. 2015. Flow aligned surfacing of curve networks. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 127.

Lena Petrović, Brian Fujito, Lance Williams, and Adam Finkelstein. 2000. Shadows for cel animation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 511–516.

Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. 2009. Analytic drawing of 3D scaffolds. In *ACM Transactions on Graphics (TOG)*, Vol. 28. ACM, 149.

Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. 2012. CrossShade: Shading Concept Sketches Using Cross-Section Curves. *ACM Transactions on Graphics* 31, 4 (2012). https://doi.org/10.1145/2185520.2185541

Daniel Sýkora, Mirela Ben-Chen, Martin Čadík, Brian Whited, and Maryann Simmons. 2011. TexToons: practical texture mapping for hand-drawn cartoon animations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*. ACM, 75–84.

Daniel Sýkora, John Dingliana, and Steven Collins. 2009a. As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*. ACM, 25–33.

Daniel Sýkora, John Dingliana, and Steven Collins. 2009b. LazyBrush: Flexible Painting Tool for Hand-drawn Cartoons. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 599–608.

Daniel Sýkora, Ladislav Kavan, Martin Čadík, Ondřej Jamriška, Alec Jacobson, Brian Whited, Maryann Simmons, and Olga Sorkine-Hornung. 2014. Ink-and-ray: Bas-relief meshes for adding global illumination effects to hand-drawn characters. *ACM Transactions on Graphics (TOG)* 33, 2 (2014), 16.

Hideki Todo, Ken-ichi Anjyo, William Baxter, and Takeo Igarashi. 2007. Locally controllable stylized shading. *ACM Transactions on Graphics (TOG)* 26, 3 (2007), 17.

Bui Minh Tuan, Junho Kim, and Yunjin Lee. 2017. Height-field Construction using Cross Contours. *Computers & Graphics* (2017).

Leonard R Wanger, James A Ferwerda, and Donald P Greenberg. 1992. Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications* 12, 3 (1992), 44–58.

Brian Whited, Gioacchino Noris, Maryann Simmons, Robert W Sumner, Markus Gross, and Jarek Rossignac. 2010. BetweenIT: An interactive tool for tight inbetweening. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 605–614.

Jun Xing, Hsiang-Ting Chen, and Li-Yi Wei. 2014. Autocomplete painting repetitions. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 172.

Jun Xing, Li-Yi Wei, Takaaki Shiratori, and Koji Yatani. 2015. Autocomplete hand-drawn animations. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 169.

Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. 2014. True2Form: 3D curve networks from 2D sketches via selective regularization. *ACM Transactions on Graphics* 33, 4 (2014).

Chih-Kuo Yeh, Pradeep Kumar Jayaraman, Xiaopei Liu, Chi-Wing Fu, and Tong-Yee Lee. 2015. 2.5 D cartoon hair modeling and manipulation. *IEEE Transactions on Visualization and computer graphics* 21, 3 (2015), 304–314.