Computers & Graphics: X 1 (2019) 100003

Contents lists available at ScienceDirect

Computers & Graphics: X

journal homepage: www.elsevier.com/locate/cagx

Special Section on Expressive 2018

2DToonShade: A stroke based toon shading system

Matis Hudon*, Mairéad Grogan, Rafael Pagés, Jan Ondřej, Aljoša Smolić

V-SENSE, School of Computer Science and Statistics, Trinity College Dublin, Ireland

ARTICLE INFO

Article history:
Received 23 November 2018
Revised 25 April 2019
Accepted 13 May 2019
Available online 24 May 2019

Keywords:
Toon shading
Cel shading
Hand-drawn animation
Image-based rendering
Non-photorealistic-rendering

ABSTRACT

We present 2DToonShade: 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 recent state of the art.

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 tools for 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 coherence 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 endeavor that producing a hand-drawn animation can be. For these reasons, there is a need for interactive tools, such as [1–10], 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 [11–13]. Manually shading a hand-drawn animation 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 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 present 2DToonShade: a semi-automatic method for creating different types of shading and shadows in a hand-drawn animation. Our system provides an easy 2D 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.

This paper embodies an extended journal version of the Expressive 2018 paper by Hudon et al. [14]. Compared to the original work, the major additional contributions are the following:

- An updated user interface allowing the creation of more shading effects.
- A reworked and simplified shade construction technique that removes artifacts generated in our previous version [14].
- A new shade enhancement system allowing the deformation of shades around detail strokes, such as, but not limited to, facial elements or cloth folds.

* Corresponding author.
E-mail address: hudonm@scss.tcd.ie (M. Hudon).

https://doi.org/10.1016/j.cagx.2019.100003

- An improved propagation algorithm making use of shape context from Belongie et al. [15] for region matching.
- An extension of the shape context computation to handle small strokes for the propagation of shade enhancements around details.
- Additional comparisons with the most recent state of the art.

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 [16], provide interactive tools for building 3D models from 2D data, automating the "inflation" to a 3D model. In Petrović et al. [11], this idea is applied 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 extensive manual interaction to obtain the desired results. In Lumo, instead of reconstructing a 3D model, Johnston [17] only estimate surface normals by interpolating from the line boundaries to render convincing illumination. However, this normal estimation is not fully automatic. It requires a separation of drawing contours in different mattes and makes use of an inner-outer line representation to determine boundary conditions. Later, further improvements have been made such as handling T-junctions and cups [18], also using user drawn hatching/wrinkle strokes [19,20] or cross section curves [21–23] to guide the reconstruction process. Recent works exploit geometric constraints present in specific types of line drawings [24–26], however, these sketches are too specific to be generalized to 2D hand-drawn animation. In TexToons [3], depth layering is used to enhance textured images with ambient occlusion, shading, and texture rounding effects. Recently, Sýkora et al. [27] make use of user annotation to recover a bas-relief with approximate depth from a single sketch of a hand-drawn character. They use this to reconstruct a full 3D model and then apply global illumination effects while still retaining the 2D style of 2D animation. This reconstruction was further improved in Dvorožňák et al. [28]. A fully automatic method to reconstruct normal maps from line drawings using convolutional neural network was presented by Hudon et al. [29]. While [27,29] both show examples of how toon shading can be applied to their reconstructions, their methods are 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. Petrović et al. [11] 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 [30–33]. Recently Fišer et al. [34] proposed to shade a 3D model mimicking the shading drawn by a user on a simple sphere.

2.3. Image registration and region matching

Propagation is a key element in reducing the workload in traditional animation and motivates a lot of work in areas such as inbetweening and image registration. Recent methods [35,36] use shape descriptors and machine learning techniques to automatically compute stroke correspondences. Object-level segmentation for cartoon video tracking was also studied in [37] but takes roughly 30 s to create correspondences in a simple 11 frame animation. Sýkora et al. [3,5] 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. [38] proposed to find stroke correspondences using a set of user-guided semi-automatic techniques to guide automation of tight inbetweening. Overall image registration, stroke correspondences and region matching is still an active field of research. State of the art works either rely on additional user input or long processing times to perform complete correspondences between frames. Many of the presented techniques make use of shape descriptors to perform the correspondences. Kanamori [39] present a comparative study of shape descriptors including to associate pairs of close regions between consecutive frames of an animation. They studied ellipses, Fourier descriptors and shape context and showed that shape context was slightly better overall.

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, seems overly complex for the task at hand. By relying only on 2D-based algorithms instead of 3D models or normal map estimation, our system bypasses the major state-of-the-art work-flow. It tries to preserve the intuition-based creative process of hand-drawn animation while still producing appealing and plausible self-shadows and shades.

3. Interface

We developed our 2DToonShade tool on top of Pencil2D animation software [40]. 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 (Fig. 1). 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. 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 different types of shades and shadows 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). A drop down menu is available in the 2DToonShade option menu, allowing the user to choose from: *Shade, Highlight, Shade&Highlight* and *Concave shade* (see examples in Fig. 2). 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 are rendered in the shading layer and can be propagated throughout all frames in the sequence. Note that our method can produce different shades depending on the input line drawing. Therefore, the user is free to choose to add invisible lines to close some shapes

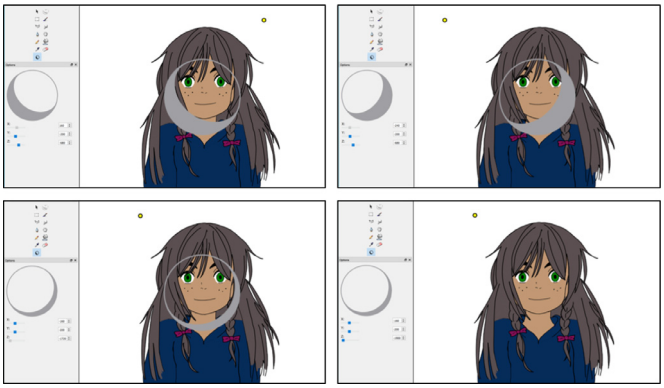

Fig. 1. Light positioning. While the user freely moves the light in the 3D space using the *X*, *Y* and *Z* sliders the system overlays a preview sphere, shaded as if it was lit from the current light position. This allows the user to directly evaluate how the shade would look on his drawings and simply 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.).

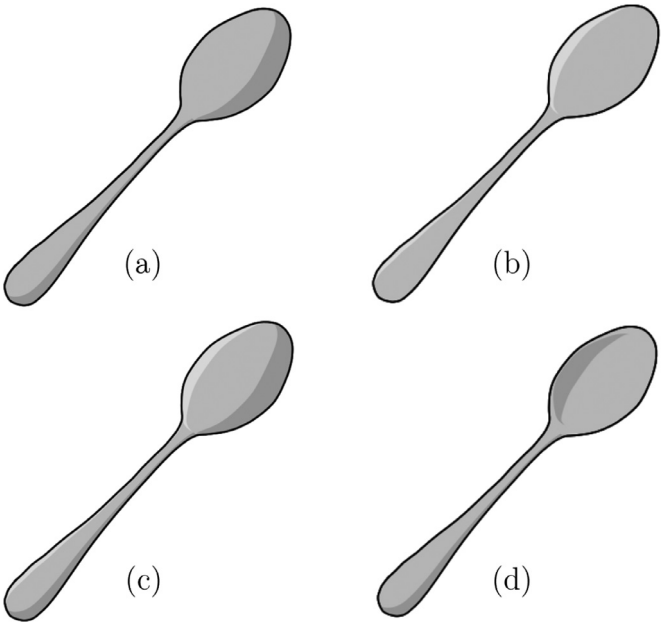

Fig. 2. Different types of shades. (a) Round shade, (b) highlight shade, (c) round shade and highlight, (d) concave shade in the hollow of the spoon and round shade on the handle.

or add details (see Fig. 26). All these features will be explained in the following sections.

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, although it can also be easily applied to raster images. We make the common assumptions that the line drawings are clean and shapes were carefully closed by the artist. This process can however be very tedious, therefore for 2DToonShade we implemented an automatic line closing loop (inspired by Gangnet et al. [41]) in Pencil2D which greatly improves the overall experience of the user.

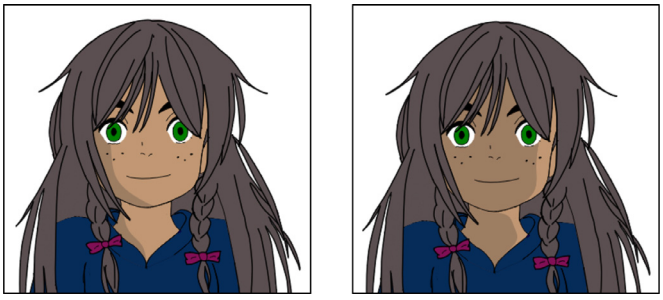

Fig. 3. The effect of inclination θ on shading. On the left a low inclination is used: the light is almost facing the character; on the right the inclination is higher: the light is coming more from the character's left side and the back. (Original drawing by Matis Hudon, used under CC BY.).

4.1. Light

As illustrated in Fig. 5, 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 and therefore the shade contour will be very narrow, whereas a higher inclination implies a wider shade (Fig. 3).

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 recomputed.

4.2. Shading

It is natural to assume that each shape of a drawing has an implicit 3D curved surface, as shown in Fig. 6. By definition, a contour line is composed of points with their normals orthogonal to the eye vector. For simplicity, we assume that input contour lines are exterior contours lines. Therefore, the profile of the 3D shape can be approximated as a blob whose height is directly related to the local width of the 2D shape, as shown in Fig. 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, often referred to as terminator, using only the local 2D geometry and the local light inclination θ.

4.2.1. Terminator construction

Having specified light coordinates, the next step is to create shade contours for the hand-drawn art. Our goal is to find a set of control points describing an enclosing area for our shade. Note that our algorithm was designed to be able to process both raster and vector images (implemented for vector images). Fig. 4 shows an example of our algorithm when applied to a drawn circle. Let P be any point in a shape. Using the 2D light direction l at P, we can locate two points called *far neighbor* N_f and *near neighbor* N_n representing the local width of the enclosing drawn shape. The terminator point S, on the line segment $[N_f N_n]$, can be computed following Algorithm 1:

where θ_P is the inclination of the light at a point P.

Using this principle, our algorithm iteratively draws the complete terminator spreading in two opposite directions from a user

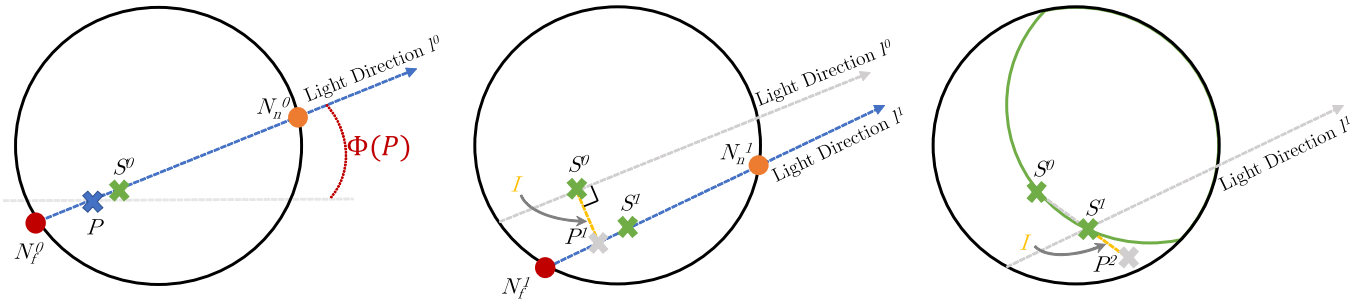

Fig. 4. Algorithm overview of shade construction. *Left*: given the click P, the first shade border point S^0 is found on the line segment $[N_f^0 N_n^0]$ taking into account the local inclination θ. *Middle*: first iteration towards the bottom: the next search point P^1 is found using a direction orthogonal 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 $\overrightarrow{S^0 S^1}$.

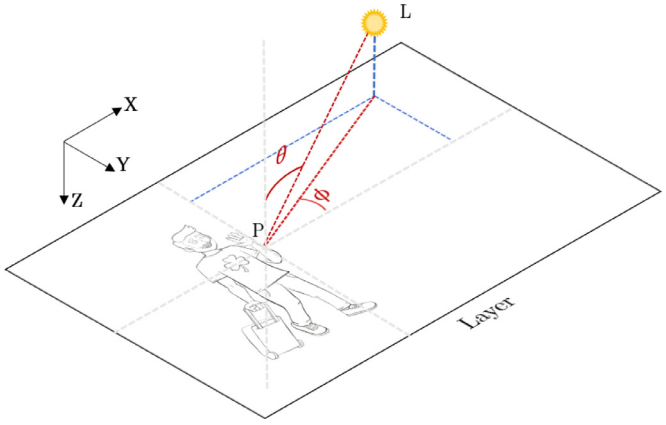

Fig. 5. Light coordinates representation. (Original drawing by Rafael Pagés, used under CC BY.).

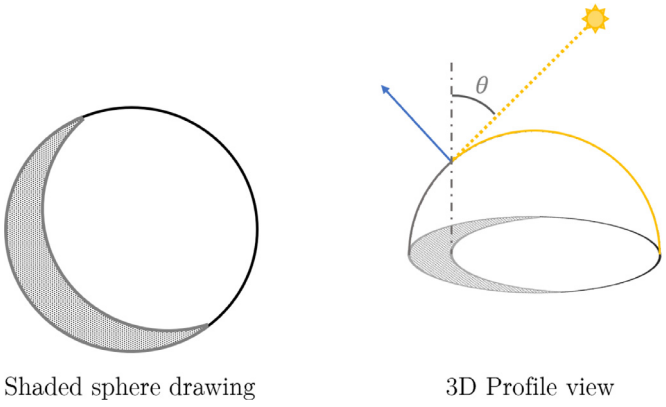

Fig. 6. On the left: an example of shading for a drawn sphere. On the right: a 3D representation giving an insight into how our tool might be thought of. The boundary between light and shade often called terminator is the 2D projection of the 3D shape profile line [17] whose normals are orthogonal to the light direction.

Algorithm 1 Compute terminator point S.

1: Initialize $\theta = \theta_P$
2: **repeat**
3: $S = N_f + (1 - \cos(\theta))\left(\frac{N_n - N_f}{2}\right)$
4: $\theta = \theta_S$
5: **until** convergence

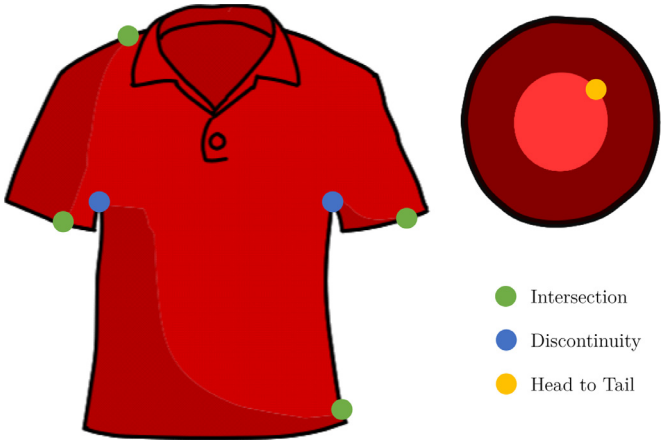

Fig. 7. Stopping criteria illustrated. Discontinuities in the direction away from the light are maintained. (Original drawing by Matis Hudon, used under CC BY.).

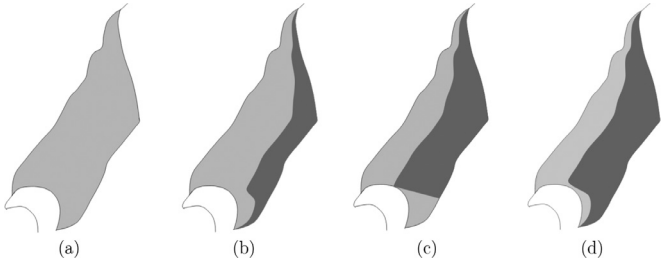

Fig. 8. (a) Input shape, (b) shading from Hudon et al. [14], (c) failure case from Hudon et al. [14] where the intersection criterion is met but leaving an unnatural shade profile, (d) our improvement.

click point P^0 (*seed point*). In each direction, given a shade point S^i, a new search point P^{i+1} is computed as :

$$P^{i+1} = S^i + \mathbf{l^i} \tag{1}$$

where $\mathbf{l^i}$ is a 2 pixel long line segment in the direction $(S^i - S^{i-1})$ for all $i > 0$. When $i = 0$, $\mathbf{l^0}$ is instead oriented in the two directions perpendicular to the light direction.

4.2.2. Stopping criteria

In our previous work [14], three stopping criteria were used: *intersection, discontinuity* and *head to tail*, see Fig. 7.

For the first stopping criterion, as expected, the shade border line should not extend beyond the edges of the surrounding drawn shape. Therefore, the terminator control points S^i cannot be outside of this shape (see Fig. 8(b)). However, we noticed that in some

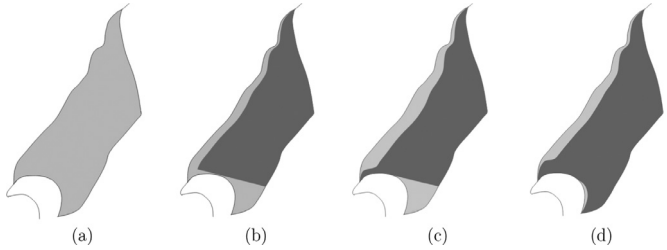

Fig. 9. (a) Input shape, (b) far neighbors discontinuity criterion failure case in Hudon et al. [14], (c) removing the discontinuity criterion, (d) no discontinuity criterion and our new filling improvement.

cases, when the shape has concave curvatures this direct implementation can produce unnatural shade profiles (see Fig. 8(c)). In order to resolve these issues, whenever the *intersection* criterion is met (let i be the index corresponding to this intersection), a new search point $\widetilde{P}^i$ is used as follows:

$$\widetilde{P}^i = \frac{S^i + N_f^i}{2} + \mathbf{I}^i \tag{2}$$

where $\mathbf{I}^i$ is the last search interval used before the intersection. Note that in the case of a concave shade or a highlight, N_f^i is replaced by N_n^i. This improvement allows the shade construction to continue as in Fig. 8(d) and avoids the unnatural shading artifact shown in Fig. 8(c).

Regarding the *discontinuity* criterion, 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 $d^i = \|N_f^i - N_f^{i-1}\|$. This distance is then compared to the previous one $d^{i-1} = \|N_f^{i-1} - N_f^{i-2}\|$. We consider that there is a discontinuity if $d^i > \beta \times d^{i-1}$, where $\beta = 10$ in our experiments. However, this criterion was causing artifacts such as that shown in Fig. 9(b). Therefore, we keep this stopping criterion only when the discontinuity is in the direction moving away from the light and completely removed it in the other case shown in Fig. 9(b). Removing the discontinuity criterion in that particular case allows the shade construction to continue as in Fig. 9(c). Unfortunately, in that case another artifact appears as the discontinuity has been ignored. Therefore, we added a routine after the shade creation where the discontinuities are found and filled if necessary, using another dedicated shade point search, starting from the middle of the discontinuity (see Fig. 9(d)).

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.

4.2.3. Shade rendering

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 ith point is filtered as follows:

$$S_f^i = \sum_{k=i-\eta}^{i+\eta} \exp\left(-\frac{\left\| S^i - S^k \right\|_2}{\sigma_1^2} - \frac{|\, i-k \,|}{\sigma_2^2} \right) \tag{3}$$

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, shade border points S^i are used 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 terminator control points S^i, forming a set of control points. As our search interval is small, the final shade contains too many points, thus cannot be edited easily by the artist if necessary. Therefore, we apply the Douglas–Peucker algorithm [42] to reduce the number of points. Finally we inter-connect the points with a Bezier curve to form the shade area. The color and transparency of the shade can be selected/changed by the user.

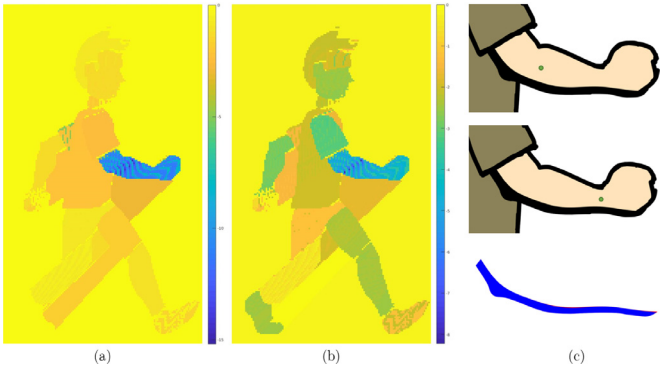

Fig. 10. Measured log similarities of shades created with different seed point locations. (a) similarity measured using Haussdorff distance, (b) similarity measured using our shade similarity metric. (c) Top: reference shade, Middle: least similar shade, Bottom: shades from top and middle superposed.

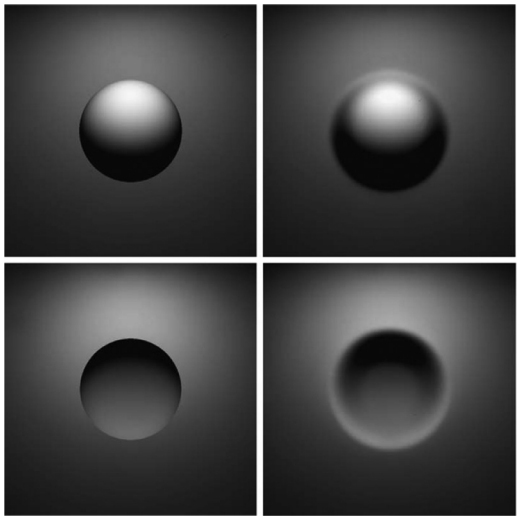

Fig. 11. Example images taken from both experiments in [44]. On the left experiment 1 with sharp edges and on the right experiment 2 with smoothed edges. Top images were obtained by rendering convex surfaces and inversely bottom images were obtained by rendering concave surfaces. Liu and Todd [44] showed an improved perception of concavity when using smoother edges.

4.2.4. Seed point invariance

Our terminator construction algorithm is by nature very robust to the seed point location. Any 2D position will create a visually identical shade, as long as it is located inside the 2D area. In Fig. 10, we measure the similarities between the shade created from a seed point (shown in Fig. 10(c) top row) and all the shades originating from all other possible seed points in the input drawing. Fig. 10 shows logged similarities measured with (a) the Hausdorff distance as in [43] and (b) our boundary shape context similarity metric presented in Section 5.2.2. In Fig. 10 (c) middle row we show the shade created by a user click in the arm region with the least similarity to the original and in (c) bottom its superposition with the original shade. This shows that the input click position within the 2D area can vary and does not visually influence the shade line result.

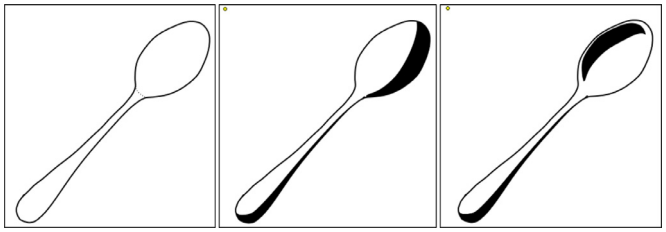

Fig. 12. Simple concave example. From left to right, input line drawing, convex shading of the spoon and concave shading of the spoon. In both cases light is coming from the yellow dot in the upper left corner. (Original drawing by Matis Hudon, used under CC BY.).

4.3. What about concavity?

The method described in the previous section works for convex surfaces. However, it is sometimes necessary to handle concave surfaces. In our case, concave and convex surfaces can have the same line drawing. For example, in Fig. 12 – left, even for a human it is impossible to determine 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 at its line drawing is an even more 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 Fig. 12. In this case, we modify equation line 3 in Algorithm 1 to use near neighbors instead of far neighbors as follows:

$$S^i = N_n^i + (1 - \cos(\theta))\left(\frac{N_f^i - N_n^i}{2}\right) \tag{4}$$

When generated this way, the concave shade could just look like a convex shade on the wrong side. Liu and Todd [44] presented interesting results on the human perception of shape from shading: they made two experiments showing concave and convex surfaces rendered with different properties (ambient light, specularities, inter-reflections). They showed, inter alia, that there was an increased perception of the concavity when they smoothed the edges of the surface (see Fig. 11). The reason given was that it is possible that sharp edges of the concave surface regions may had been perceptually misinterpreted as smooth occlusion contours. When the sharp edges were removed, the observers' accuracy at judging concave surfaces increased significantly. We applied this principle to our shading technique. To achieve this, we translate shade control points towards the barycenter B_s of far and near neighbors by a distance dependent 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 $[S^iB_s]$, towards B_s by a distance:

$$distance^i = \left(\zeta + \gamma\left(1 - \frac{\left\|N_n^i - S^i\right\|}{M_T}\right)\right)\left\|S^i - B_s\right\| \tag{5}$$

where the scaling factor γ is 0.1 and the constant ζ is 0.05 in our experiments. M_T is the maximum thickness $\left\|N_n^i - S^i\right\|$ found in the set S. This exaggeration of the shading simulates smoother edges and leads to an improved perception of the concavity, as shown in [44].

4.4. Shade enhancements with details

One of the main improvements compared to our previous work [14], is the possibility of adding shade deformations where the shades intersect simple detail strokes. Supposing that all the detail strokes are located in a special layer, the user is free to select detail strokes (highlighted when hovered over by the cursor, if the detail layer is selected); once selected, a stroke will automatically deform the shade curve that it intersects.

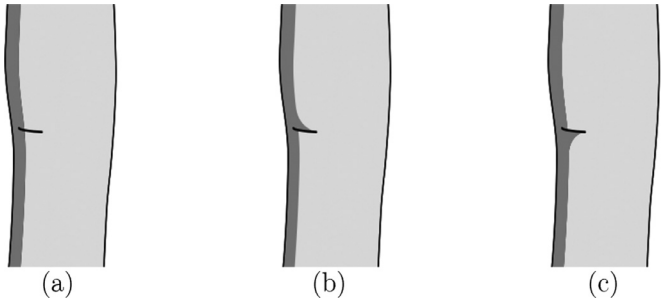

Fig. 13. Example of shade deformation on a detail stroke. Before deformation (a), deformation with light coming from bottom right (b), and deformation with light coming from top right (c).

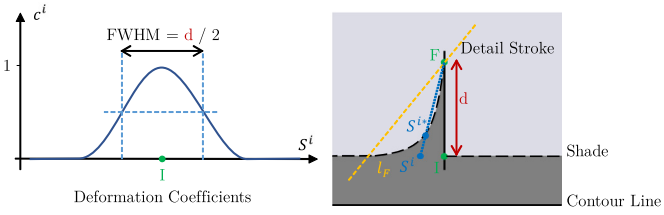

Fig. 14. Representation of the Gaussian coefficients c^i (blue curve) and the corresponding deformation around the detail stroke. The full width half maximum of the Gaussian *FWHM* used to generate the deformation coefficients c^i is set to $d = \|F - I\|$. Here, only the left side of the shade will be displaced as the light direction l_F falls on the left of $[FI]$.

4.4.1. Design

We noticed that, in hand-drawn and hand-shaded animations, even the simplest shades can sometimes be deformed following detail strokes, suggesting local geometric deformations in the global shape. For example, in the case of hand-drawn characters, those details deforming the shades are often face elements (eyes, mouth, nose, eyebrows etc.) or cloth folds. While there are many subjective ways of drawing/deforming the shades around those details, in the most common case the shade will only be deformed on one side of the detail stroke, as shown Fig. 13. Therefore, we chose to orient the shade deformation on the detail stroke using the light position: the shade will lie on the opposite side of the light with respect to the detail stroke. Another design concept is related to the size of the deformation caused by a detail stroke. Indeed, we noticed that in general, the longer the detail stroke the bigger the deformation.

4.4.2. Modeling the shade deformation

We model the deformation using two points: I the intersection point between the detail stroke and the shade line, and F, the furthest point on the detail stroke following the light direction. The shade points S^i of the shade line around I will be displaced towards F to new shade points S^{i*} using Gaussian coefficients c^i represented in Fig. 14:

$$S^{i*} = S^i + (F - S^i) \times c^i \tag{6}$$

where the Gaussian coefficients are defined by:

$$c^i = \exp\left(-\frac{\left\|(S^i - I)\right\|_2}{\sigma^2}\right) \tag{7}$$

where the standard deviation σ is chosen as:

$$\sigma = \frac{\|F - I\|}{\sqrt{4ln2}} \tag{8}$$

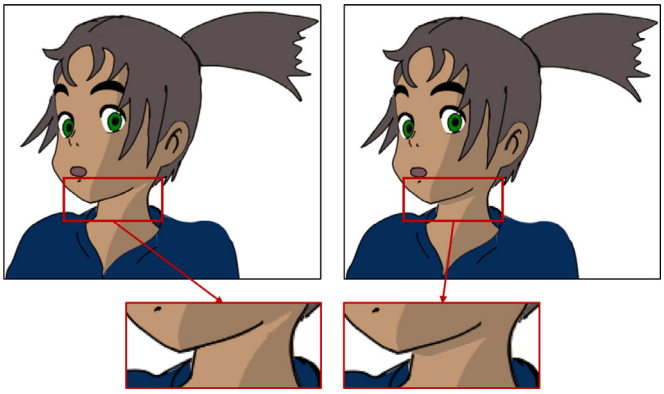

Fig. 15. Example of the same drawing with and without self-shadowing (top), magnification of the interesting areas (bottom). (Original drawing by Matis Hudon, used under CC BY.).

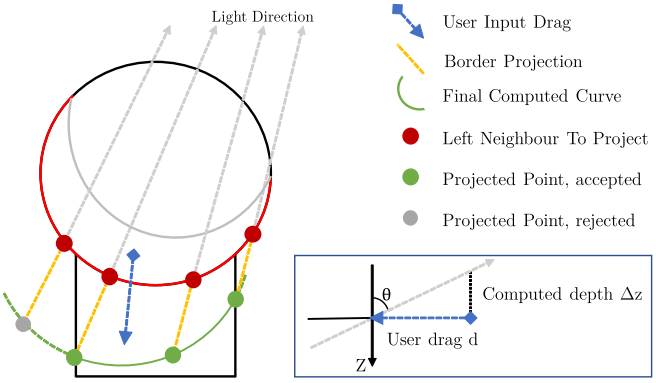

Fig. 16. Self shadowing construction overview. In this example, the circle area is casting a shadow onto the squared area.

so that the full width half maximum *FWHM* of the Gaussian is equal to $\|F - I\|/2$.

The shade points must be displaced only if they are on the right side of the detail stroke. To determine this side, we use the light direction l_F (see Fig. 14). Whatever side of the detail segment l_F lies on will determine which part of the shade will be deformed. Finally, the vector points of the line segment [*FI*] are inserted correctly in the shade line.

4.5. Local self-cast shadows

Self-cast shadows occur 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 Fig. 15.

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 releasing in the area receiving the shadow. For example, in Fig. 15 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.

4.5.1. Border shadow casting

The profile of the cast-shadow depends on the profile of the area casting the shadow. As an example, in Fig. 15, 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 (Fig. 16). To create the shadow we need to find the projections of those points and determine whether they fall into the receiving area or not.

4.5.2. 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:

$$\Delta z = \frac{d}{\tan\theta}. \tag{9}$$

Here, $\theta \in]0, \frac{\pi}{2}[$ 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 in reducing the workload in traditional animation and motivates a lot of work in areas such as inbetweening [38] and image registration [5].

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 shown in Section 4.2.4 before, this input is not strictly constrained: any 2D position will create a visually identical shade, as long as it is located inside the 2D area. 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 basic shades, however a similar algorithm can be applied to shadows, highlights and concave shades. 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 [4]), 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.

5.1. Generation of seed candidates list

Typically, in two consecutive frames, corresponding areas are spatially close or even overlap. Therefore, a reasonable sampling

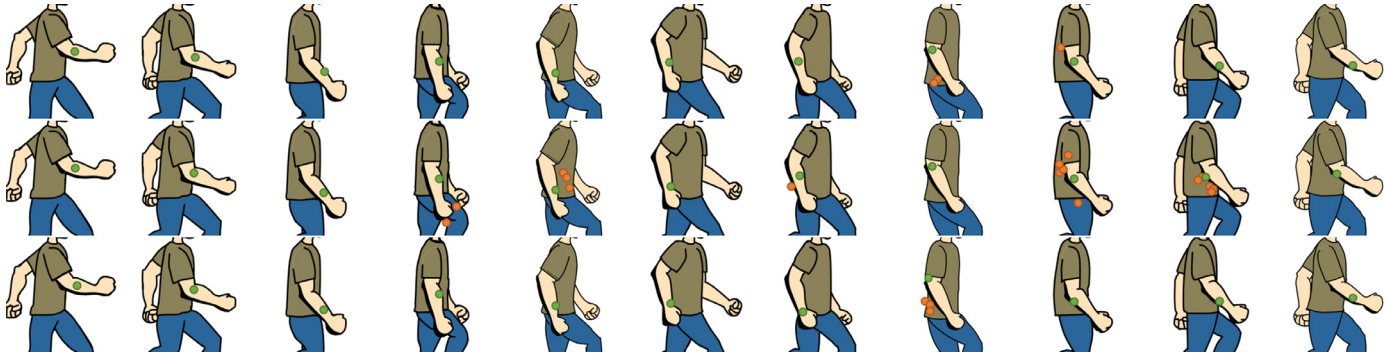

Fig. 17. Propagation results using different shape metrics. The green point is the selected candidate, the orange points are the points that were ranked higher than the selected candidate after pre-sorting. (Top row) Hausdorff distance, (Middle row) shape context, (Bottom row) boundary shape context. Color matching was not used to generate these results.

of the area where the shade is located in *frame n* has a high probability of containing good seed candidates. While the seed candidates list could be generated by evenly sampling in a bounding box around the concerned shape, we found it convenient to directly extract the sampling from the shape's boundary points found during the shade construction. Following experiments we found that taking a sampling of 9 points within this shape is a good compromise between accuracy and efficiency. When candidate points from the two previous frames are available (i.e. at least one propagation has already been performed), nine other candidates are computed through motion prediction by taking the difference between two corresponding candidate positions in the current and previous frame. Finally, candidate points whose color does not match with the previous seed point color are rejected.

5.2. Selecting one candidate

Let C be our set of candidates, s the shade we want to propagate from *frame n*, A the region around s and A^{+1} the corresponding region 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.

5.2.1. Pre-sorting candidates

In our previous work [14], in order to find a good seed point among the set of candidates C, we were processing them one by one until a correct seed point is found. In order to increase the robustness and the rapidity of the method, we propose to first sort the candidates in C based on how similar their surrounding region/boundary is in comparison to that around the seed pixel of s. This maximizes the probability of finding a good seed candidate at the top of the candidates list and thus minimizes the overall computation. To do so, we tested and compared three different shape descriptors for region/boundary matching: The Hausdorff distance as presented [43], the shape context inspired by Belongie et al. [15] and a local boundary version of the shape context using only the boundary curves about the considered point instead of the full sketch. We show a comparison of these metrics in Fig. 18. The shape context based metric performs well except when occlusions occur, such as in Fig. 18(c) bottom row. In this case, some lines disappear from the drawing leading to false positives. In contrast, the Hausdorff and the boundary shape context based metrics seem to perform equally well and robustly to occlusions. Fig. 17 shows propagation results on the same sequence using the three mentioned metrics. Note that color matching has not been use to generate these propagation results. Once again the Hausdorff distance and the boundary shape context based metrics show similar performance while the shape context metric is more prone to errors.

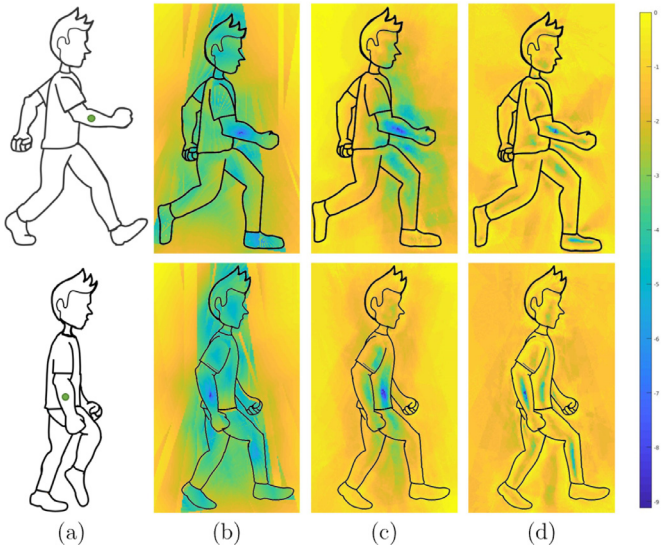

Fig. 18. Comparison of shape metrics. (a) original frame and seed point, (b) Hausdorff distance, (c) shape context, (d) boundary shape context. All similarities measured were logged for this figure.

We chose to use the boundary shape context. In our implementation, we construct the shape context histograms using 10 equally spaced angle bins and 10 equally spaced log-radius bins. to compare two shape context histograms we use the χ^2 metric as presented in Belongie et al. [15].

5.2.2. Similarity check

The candidates have been sorted by probability of being a good candidate. We suppose that *frame n* and *frame* $n+1$ are temporally and spatially close, thus the shade generated by a candidate has to be close to the original shade. Letting c_k be our kth candidate, we 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 shades s and s_{c_k} by comparing their series of distances between far construction points and terminator points. If the two shades are similar enough, we can then consider that c_k is a good candidate. As the size and shade 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 far-terminator distances Ω_s over the longer set of far-terminator distances Ω_l. Let n_s and n_l be the cardinals of S_s and S_l, respectively:

$$\Pi = \min\{RMSE_\tau(\Omega_s, \Omega_l), \tau \in [0, n_l - n_s]\} \quad (10)$$

where τ is a sliding offset.

$$RMSE_\tau(\Omega_s, \Omega_l) = \sqrt{\frac{1}{n_s}\sum_{i=1}^{n_s}(\Omega_s(i+\tau) - \Omega_l(i))^2} \quad (11)$$

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

$$\xi = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\alpha \times \Omega(i))^2} \quad (12)$$

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 $\alpha = 0.3$. In the rare cases where no correct seed point has been found among all candidates, the propagation process stops, giving the user the opportunity to input a 2D point manually.

5.3. Propagating details

As we add the shade enhancements with details as a new contribution in this work, we also need a way to propagate those shade enhancements along with the shade propagation. Our main goal here is to find the correspondences between strokes in the detail layers of two consecutive frames of an animation sequence. As stated in Section 4.4, strokes in the "Details" layer are assumed to be simple. For this purpose, we present an accumulative shape context descriptor that allows for the matching of detail stokes between frames. Instead of computing the shape context histogram for one point, we combine the shape context histograms computed for each of the points on the detail stroke into one histogram. We then normalize the histogram. When a shade contains detail enhancements, we first compute the accumulative shape context histogram for each detail stroke selected by the user for shade enhancements. We then compare these histograms, using the χ^2 metric presented in [15] context histogram computed for each strokes intersecting the propagated shade in the next frame. Note that the accumulative shape context is computed with respect to drawing-layer strokes and not the detail layer strokes. Finally if the best χ^2 score is below a certain threshold (0.1 in our experiments) we add the detail enhancement to the propagated shade.

6. User evaluation

We conducted a user experiment in order to evaluate the utility and usability of the basic features 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 (Fig. 19). 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 (Fig. 20). This experiment allows us to measure the effectiveness of our method 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.

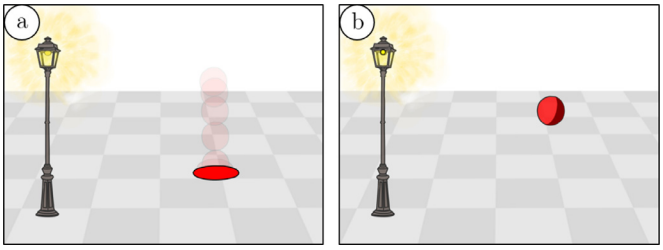

Fig. 19. Bouncing ball scene: (a) onion skin view of the animation, (b) One frame shaded by our shading tool. (Original drawing by Matis Hudon, used under CC BY.).

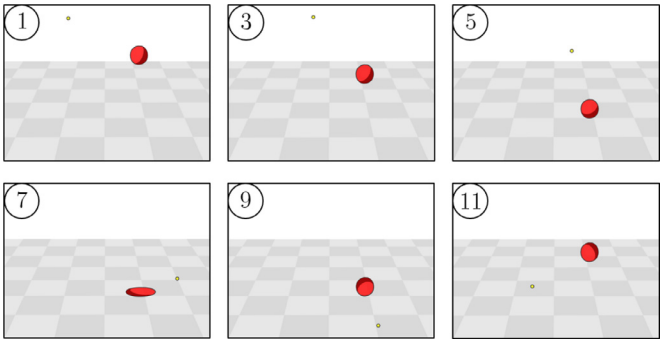

Fig. 20. Bouncing-ball free session: light position animation of one participant (One frame over two 1–11). (Original drawing by Matis Hudon, used under CC BY.).

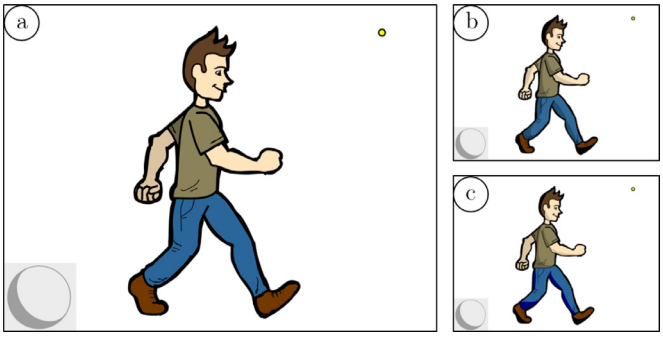

Fig. 21. The walking cycle scene: (a) original frame to shade, (b) example of manual shading by a participant, (c) example of semi-automatic shading by the same participant. (Original drawing by Rafael Pagés, used under CC BY.).

In the second part of this experiment, the user is asked to shade a human cartoon character in a walking cycle, as shown in Fig. 21. 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

Fig. 22 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

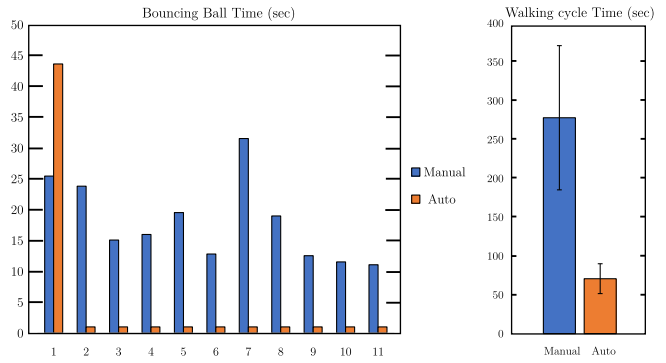

Fig. 22. Performance results. Average timings per frame for the bouncing-ball experiment (left). Average timings for the walking-cycle experiment ± standard deviation (right).

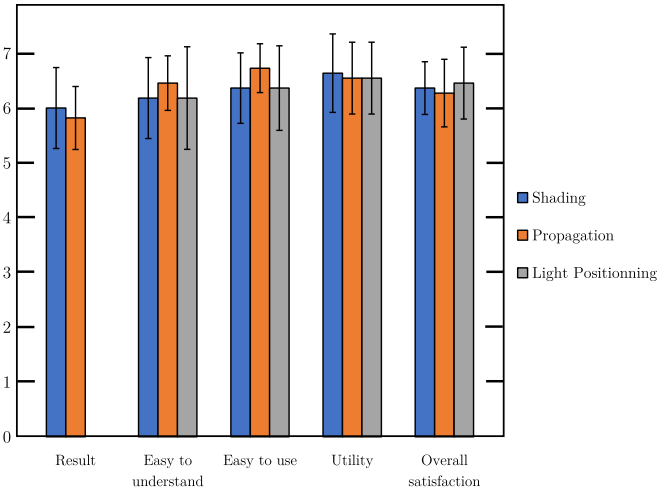

Fig. 23. User feedback. All quantities are expressed as mean ± standard deviation in a 7-point Likert scale.

shade. For a simple scene such as the bouncing ball with only one shade to draw per frame, as it can be seen in Fig. 22, 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 s whereas it was only 55 s for the semi-automatic shading. The ratio of completion time by our system versus manual drawing is 3.64, which greatly demonstrates the labor 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. Fig. 22 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. Fig. 23 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

Fig. 28 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.

We also show a qualitative comparison of our method versus several state of the art methods in Fig. 24. We used the Stomping Man scene from Petrović et al. [11]. 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 artwork. 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 Fig. 24 and took approximately 6–7 min to create. TVPaint Pro provides an automatic and efficient shading tool called Toon Shading which is an implementation of the method from Textoon tool [3]. 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 s (this is the user time, the algorithm itself is very fast – see Sýkora et al. [3]). 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 Fig. 24 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.

As stated in Section 2, state of the art methods such as Sýkora et al. [27], later improved in Dvorožňák et al. [28], 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 Petrović et al. [11], 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. Fig. 25 shows a visual comparison of our shading versus classical Toon-shader from an artifact free 3D model (first row), an estimated 3D model from Dvorožňák et al. [28] (middle row) and a reconstructed normal map from [29] (bottom row).

When comparing our result to that generated using a 3D model which contains all the accurate geometric information about the shape, the result can appear overly realistic in comparison to manual toon shading. Our tool provides a less realistic shading, however it is closer to a traditional and appealing cartoon shading style.

In the second row, we are using a 3D reconstruction from [28] as input to the classic toon shader. This model appears to contain enough, but not too many, geometric details for a nice toon shaded result. However, the 3D reconstruction of [28] is the result of a more complex interactive modeling scenario, where the

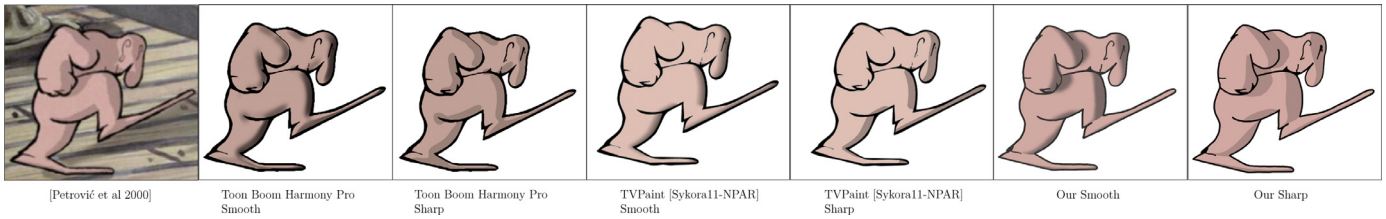

Fig. 24. Qualitative shading comparison between [11], our method(< 20s), Toon Boom Harmony pro© tone shader (6–7min) and TVPaint Pro© Toon shader [3] (< 20s). The Stomping Man drawing was reproduced from [11] with authors' consent. Light was approximately placed to mimic the lighting from the original 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.

Fig. 25. Visual comparison of Classic Toon-Shading versus our shading result. From left to right, *Top row:* Freestyle plugin (Blender©) applied to a 3D model, a toon shading obtained from the 3D model and our shading result. *Middle row:* original drawing from Dvorožňák et al. [28], shading result obtained in Blender©and our result. *Bottom row:* original input drawing, toon shading result obtain from Hudon et al. [29], our result. (Source drawings: Top row 3D model by Andy Goralczyk, Middle row Anifilm©, bottom row by Rafael Pagés).

input drawing can be separated into a set of semantically meaningful parts, of which relative depth order is known beforehand, such as in their previous work [27].

In the last row we compare our shading system to the fully automatic one in [29]. In this paper the authors reconstruct a high detailed normal map from the input line drawing. However, similar to the results of the 3D model in the first row, the shades generated using reconstructed normal map are in some places too

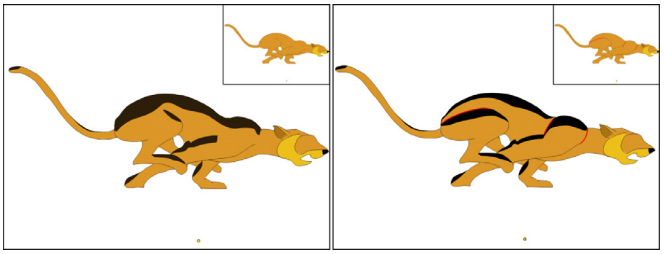

Fig. 26. Shading with (right) and without (left) invisible line comparison. Invisible lines were highlighted in red for this figure. In both cases, light is coming from the yellow dot. (Source drawing by Esther Huete, used under CC BY.).

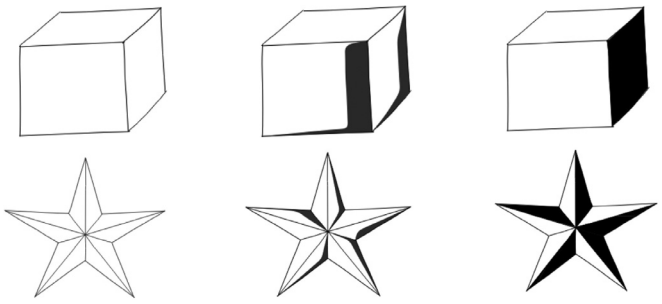

Fig. 27. From left to right: Input shapes with sharp edges, our shading tool, bucket tool colouring.

complex for a nice toon shading result, and does not deal with shadows.

Overall, in the state of the art that rely on geometric models, the quality of the toon shading result is closely related to the amount of geometric detail in the 3D model/normal map. One key advantage of our system is that the user is free to choose where to create the shades. Also, rather than giving a rendered result, shades can be edited/modified one by one. Furthermore, shading colors can be carefully chosen by the artist to obtain the most appealing result possible rather than a physically realistic one.

Our method can produce different shading results depending on the input line drawing. Therefore, an even more finely tuned shading result can be created by adding invisible lines in the line drawing, as shown in Fig. 26. 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 Fig. 26), 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.

Fig. 28. *Results of artworks shaded with our method. Invisible lines were highlighted in red for this figure. First row*: From left to right: input drawing, input detail layer, result from Hudon et al. [14], new result, artist hand-drawn shading. *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. Hair, face details and main body/head are split on three different layers. (Source drawings: Row 1 by Matis Hudon, Row 2 by Rafael Pagés, Row 3–5 by Esther Huete, Row 6 by Matis Hudon; all used under CC BY).

This work presents a method using only one light source but could be applied to a scene with multiple light sources at the cost of more complexity for the user. Also, different types of lights such as area lights or spot 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. While flat surfaces are not handled by our system, we think that a flat surface in toon-shading is either completely shaded or not shaded. Therefore the artist can simply shade a flat surface by using the bucket

tool Fig. 27. Our system struggles to automatically handle local small discontinuities and sharpness differences such as bumps and dents. 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 currently not possible, but this could be a compelling feature. Also employing more advanced shade edits than simply moving vector control points could be investigated.

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 results, it is still limited to points of view where the whole shade is visible – self occlusions in concave objects are currently not supported. While the exaggeration of the concave effect presented in this paper is accentuating the perception of concavity, it would be interesting to let the user scale this effect by adding additional control on the parameter λ in (5).

Some failure cases can be seen in Fig. 28. 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. An automatic correction for this particular case could also be investigated.

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. In our previous version [14], very similar and spatially close objects were likely to make the propagation process fail. However, since the introduction of shape contexts, those cases are less problematic.

9. Conclusion

This paper presents 2DToonShade: a complete system 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 plausible cartoon style shades and self-shadows. The process drastically reduces the labor of drawing shades and self-shadows by hand. It also brings more flexibility to the user as the lighting can be moved in retrospect. 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 software packages for professional animation. We are convinced that it can motivate future work on non-physical manipulation of shading and shadows.

Conflict of interest

None.

Acknowledgments

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

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.cagx.2019.100003.

References

[1] Xing J, Wei L-Y, Shiratori T, Yatani K. Autocomplete hand-drawn animations. ACM Trans Gr (TOG) 2015;34(6):169.
[2] Xing J, Chen H-T, Wei L-Y. Autocomplete painting repetitions. ACM Trans Gr (TOG) 2014;33(6):172.
[3] Sýkora D, Ben-Chen M, Čadík M, Whited B, Simmons M. Textoons: practical texture mapping for hand-drawn cartoon animations. In: Proceedings of the ACM SIGGRAPH/eurographics symposium on non-photorealistic animation and rendering. ACM; 2011. p. 75–84.
[4] Sýkora D, Dingliana J, Collins S. Lazybrush: Flexible painting tool for hand–drawn cartoons. In: Proceedings of the computer graphics forum, 28. Wiley Online Library; 2009a. p. 599–608.
[5] Sýkora D, Dingliana J, Collins S. As-rigid-as-possible image registration for hand-drawn cartoon animations. In: Proceedings of the seventh international symposium on non-photorealistic animation and rendering. ACM; 2009b. p. 25–33.
[6] Feng L, Yang X, Xiao S, Jiang F. An interactive 2d-to-3d cartoon modeling system. In: Proceedings of the international conference on technologies for e-learning and digital entertainment. Springer; 2016. p. 193–204.
[7] Henz B, Oliveira MM. Artistic relighting of paintings and drawings. Vis Comput 2017;33(1):33–46.
[8] Yeh C-K, Jayaraman PK, Liu X, Fu C-W, Lee T-Y. 2.5 d cartoon hair modeling and manipulation. IEEE Trans Vis Comput Graph 2015;21(3):304–14.
[9] Simo-Serra E, Iizuka S, Ishikawa H. Real-time data-driven interactive rough sketch inking. ACM Trans Gr (SIGGRAPH) 2018a;37(4).
[10] Simo-Serra E, Iizuka S, Ishikawa H. Mastering sketching: adversarial augmentation for structured prediction. In: Proceedings of the Transactions on Graphics Presented at SIGGRAPH, 37.
[11] Petrović L, Fujito B, Williams L, Finkelstein A. Shadows for cel animation. In: Proceedings of the twenty-seventh annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co.; 2000. p. 511–16.
[12] Wanger LR, Ferwerda JA, Greenberg DP. Perceiving spatial relationships in computer-generated images. IEEE Comput Gr Appl 1992;12(3):44–58.
[13] Kersten D, Mamassian P, Knill DC. Moving cast shadows induce apparent motion in depth. Perception 1997;26(2):171–92.
[14] Hudon M, Pagés R, Grogan M, Ondřej J, Smolić A. 2d shading for cel animation. In: Proceedings of the joint symposium on computational aesthetics and sketch-based interfaces and modeling and non-photorealistic animation and rendering. ACM; 2018a. p. 15.
[15] Belongie S, Malik J, Puzicha J. Shape matching and object recognition using shape contexts. Technical Report. California Univ San Diego la Jolla Dept of Computer Science and Engineering; 2002.
[16] Igarashi T, Matsuoka S, Tanaka H. Teddy: A sketching interface for 3d freeform design, siggraph. In: Proceedings of the Conference ACM; 1999.
[17] Johnston SF. Lumo: illumination for cel animation. In: Proceedings of the second international symposium on non-photorealistic animation and rendering. ACM; 2002. 45–ff.
[18] Karpenko OA, Hughes JF. Smoothsketch: 3d free-form shapes from complex sketches. ACM Trans Gr (TOG) 2006;25:589–98.
[19] Jayaraman PK, Fu C-W, Zheng J, Liu X, Wong T-T. Globally consistent wrinkle-aware shading of line drawings. IEEE Trans Vis Comput Gr 2017.
[20] Bui MT, Kim J, Lee Y. 3d-look shading from contours and hatching strokes. Comput Gr 2015;51:167–76.
[21] Shao C, Bousseau A, Sheffer A, Singh K. CrossShade: shading concept sketches using cross-section curves. ACM Trans Gr 2012;31(4). doi:10.1145/2185520.2185541.
[22] Tuan BM, Kim J, Lee Y. Height-field construction using cross contours. Comput Gr 2017.
[23] Iarussi E, Bommes D, Bousseau A. Bendfields: regularized curvature fields from rough concept sketches. ACM Trans Gr 2015;34(3):24:1–24:16. doi:10.1145/2710026.
[24] Schmidt R, Khan A, Singh K, Kurtenbach G. Analytic drawing of 3d scaffolds. ACM Trans Gr (TOG) 2009;28:149.
[25] Xu B, Chang W, Sheffer A, Bousseau A, McCrae J, Singh K. True2form: 3d curve networks from 2d sketches via selective regularization. ACM Trans Gr 2014;33(4).
[26] Pan H, Liu Y, Sheffer A, Vining N, Li C-J, Wang W. Flow aligned surfacing of curve networks. ACM Trans Gr (TOG) 2015;34(4):127.
[27] Sýkora D, Kavan L, Čadík M, Jamriška O, Jacobson A, Whited B, et al. Ink-and-ray: bas-relief meshes for adding global illumination effects to hand-drawn characters. ACM Trans Gr (TOG) 2014;33(2):16.
[28] Dvorožňák M, Nejad SS, Jamriška O, Jacobson A, Kavan L, Sýkora D. Seamless reconstruction of part-based high-relief models from hand-drawn images. In: Proceedings of the joint symposium on computational aesthetics and sketch-based interfaces and modeling and non-photorealistic animation and rendering. ACM; 2018. p. 5.
[29] Hudon M, Pagés R, Grogan M, Smolić A. Deep normal estimation for automatic shading of hand-drawn characters. In: Proceedings of the ECCV Workshops.

[30] Anjyo K-i, Wemler S, Baxter W. Tweakable light and shade for cartoon animation. In: Proceedings of the fourth international symposium on non-photorealistic animation and rendering. ACM; 2006. p. 133–9.
[31] Barla P, Thollot J, Markosian L. X-toon: an extended toon shader. In: Proceedings of the fourth international symposium on non-photorealistic animation and rendering. ACM; 2006. p. 127–32.
[32] Lee Y, Markosian L, Lee S, Hughes JF. Line drawings via abstracted shading. ACM Trans Gr (TOG) 2007;26:18.
[33] Todo H, Anjyo K-i, Baxter W, Igarashi T. Locally controllable stylized shading. ACM Trans Gr (TOG) 2007;26(3):17.
[34] Fišer J, Jamriška O, Lukáč M, Shechtman E, Asente P, Lu J, et al. Stylit: illumination-guided example-based stylization of 3d renderings. ACM Trans Gr (TOG) 2016;35(4):92.
[35] Liu D, Chen Q, Yu J, Gu H, Tao D, Seah HS. Stroke correspondence construction using manifold learning. In: Proceedings of the computer graphics forum, 30. Wiley Online Library; 2011. p. 2194–207.
[36] Yu J, Bian W, Song M, Cheng J, Tao D. Graph based transductive learning for cartoon correspondence construction. Neurocomputing 2012;79:105–14.
[37] Zhu H, Liu X, Wong T-T, Heng P-A. Globally optimal toon tracking. ACM Trans Gr (TOG) 2016;35(4):75.
[38] Whited B, Noris G, Simmons M, Sumner RW, Gross M, Rossignac J. Betweenit: an interactive tool for tight inbetweening. In: Proceedings of the computer graphics forum, 29. Wiley Online Library; 2010. p. 605–14.
[39] Kanamori Y. A comparative study of region matching based on shape descriptors for coloring hand-drawn animation. In: Proceedings of the 2013 twenty-eighth international conference on image and vision computing New Zealand (IVCNZ 2013). IEEE; 2013. p. 483–8.
[40] Pencil2d animation. https://www.pencil2d.org/.
[41] Gangnet M, Van Thong J-M, Fekete J-D. Automatic gap closing for freehand drawing. ACM SIGGRAPH, 94; 1994.
[42] Douglas DH, Peucker TK. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartogr: Int J Geogr Inf Geovis 1973;10(2):112–22.
[43] Noris G, Sỳkora D, Coros S, Whited B, Simmons M, Hornung A, et al. Temporal noise control for sketchy animation. In: Proceedings of the ACM SIGGRAPH/eurographics symposium on non-photorealistic animation and rendering. ACM; 2011. p. 93–8.
[44] Liu B, Todd JT. Perceptual biases in the interpretation of 3d shape from shading. Vis Res 2004;44(18):2135–45.