Pacific Graphics 2016
E. Grinspun, B. Bickel, and Y. Dobashi
(Guest Editors)

Volume 35 (2016), Number 7

Flow Curves: an Intuitive Interface for Coherent Scene Deformation

L. Ciccone[1] M. Guay[2] R. Sumner[1,2]
[1]ETH Zurich [2]Disney Research Zurich

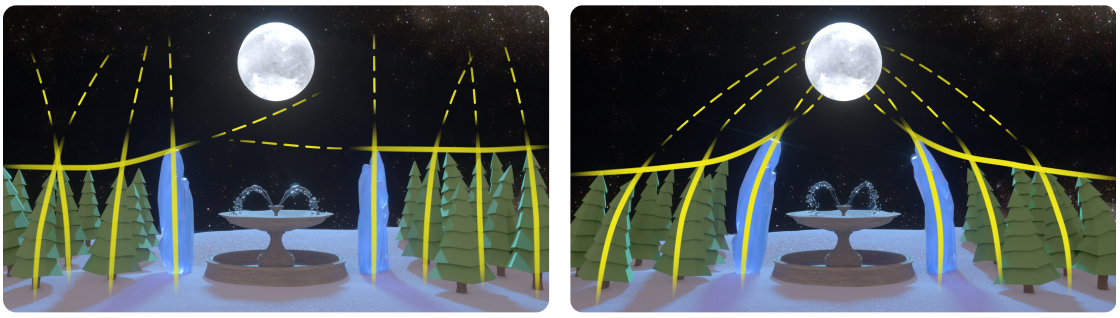


Figure 1: *Our new Flow Curves interface is designed to help artists to take a scene with an ambiguous flow (left), and quickly turn it into a compelling scene (right) by simply sketching strokes—inducing whole-scene, multi-object deformations.*

Abstract

Effective composition in visual arts relies on the principle of movement, where the viewer's eye is directed along subjective curves to a center of interest. We call these curves subjective because they may span the edges and/or center-lines of multiple objects, as well as contain missing portions which are automatically filled by our visual system. By carefully coordinating the shape of objects in a scene, skilled artists direct the viewer's attention via strong subjective curves. While traditional 2D sketching is a natural fit for this task, current 3D tools are object-centric and do not accommodate coherent deformation of multiple shapes into smooth flows. We address this shortcoming with a new sketch-based interface called Flow Curves which allows coordinating deformation across multiple objects. Core components of our method include an understanding of the principle of flow, algorithms to automatically identify subjective curve elements that may span multiple disconnected objects, and a deformation representation tailored to the view-dependent nature of scene movement. As demonstrated in our video, sketching flow curves requires significantly less time than using traditional 3D editing workflows.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

1. Introduction

"The central point of interest has nothing to do with the center of the frame, but everything to do with where you wish the audience's eye to eventually rest." [But02]

When viewing a piece of visual art or design, our eye is often naturally drawn towards one main area of the scene, typically where the main character, object or action is located. This effect is not accidental. Rather, it results from skilled artists cleverly exploiting mechanisms of our visual system in order to more effectively communicate the important elements of an image. In particular, they use our tendency to pick up contrast edges across multiple objects and join them together into longer imaginary *subjective curves* that direct the eye's movement across the image. Hence, by carefully shaping and coordinating contrast edges across objects in a scene, visual artists are capable of controlling the very way we look at an image and bring our attention to a *center-point* (see Fig. 2). This concept is referred to as the principle of *movement* in visual arts, *flow* in design, and *good continuation* in Gestalt perceptual theory.

Traditional 2D sketching offers the freedom to naturally craft and design scene-wise subjective curves. In contrast, current 3D tools are designed in an object-centric fashion which does not easily accommodate building a coherent movement, for a given view-point. In fact, digital artists are obliged to shape each object separately, using several rig and deformation handles that typically differ from object to object. In other words, these tools are agnostic to the artistic principle of movement and provide no means of expressing co-

Figure 2: *Illustration of movement in existing artwork (Left: book cover of Predator's Gold, Middle: screenshot from Mickey's Christmas Carol, Right: poster of Ratatouille). Yellow lines represent subjective curves pointing to a center-point.*

herence across multiple objects in a scene. Hence, despite the dramatic importance of movement in visual design, 3D artists are left with a cumbersome, indirect and time consuming workflow.

Our work addresses this shortcoming by introducing a new sketch-based interface called *Flow Curves* that provides a direct coordination control across multiple objects. From raw input geometry, we automatically compute a deformation embedding for objects in the scene that allows directly deforming shapes without rig setup time. To provide the user with the possibility to deform objects with a single stroke, we automatically compute *subjective curve elements* (or *SEcurves*) spanning multiple objects in the form of *principal curves* and *abstract contours*. By defining a *flow curve* as a shape constraint over close-by SEcurves, we can optimize for a scene deformation that conforms to the sketched flow curves. When compared to traditional workflows, our interface offers a significant increase in efficiency (see video and Table 1 for comparison).

Our main contribution is a definition of flow curves as shape constraints over SEcurves formed from contrast edges in a scene. We additionally contribute automatic techniques for computing two classes of often-found SEcurves: principle curves and abstract contours. We demonstrate the efficiency of our approach with a direct deformation implementation that allows deforming scenes without manual rig setup time.

2. Related Work

Artistic principles. The principles of visual arts reflect the different aspects of the human visual system and serve as guidelines for shaping scenes into aesthetically pleasing images. Several principles contribute to the overall coherence and aesthetic of a scene; some of which have already been integrated into computational tools, such as *balance* for photo composition [LCWCO10] and scene layout placement [LMLF15], or *emphasis* for directing gaze [CDF*06, BMSG09]. In this paper, we focus on the principle of *movement* where subjective curves flow across contrasts towards a center-point, leading the viewer's attention to the important parts of the scene. "Movement is the way a viewer's eye is directed to move through a composition, often to areas of emphasis." [Gla13]. To our knowledge, this principle has never been used for whole scene deformation, but only for individual characters with the *line of action* drawing concept [OBP*13, GCR13], where the whole shape of a character forms a smooth (skeletal) curve pointing towards the main action in the scene. Our flow curves subsume lines of action and allow deforming whole scenes—including characters—using a single stroke.

Subjective curves are imaginary curves that we perceive when grouping together salient contrast edges and points. Gestalt theory refers to this perceptual phenomenon as the principle of "good continuation" [Wer38]. Due to its subjective nature, a precise mathematical characterization of good continuation remains elusive. Additionally, the bulk of academic research in computer vision is geared towards identifying "*existing*" subjective curves, either via fixed primitives such as circle arcs and Euler spirals [Ull76], minimal curvature splines [Hor83, Mum94], or probabilistic models [WJ97] where a scalar (probability) field is computed from all the pairwise interactions between edge segments (curve formed by tracing paths along probability peaks). These methods identify subjective curves in existing images, while our work provides a practical tool for shaping and forming new subjective curves.

Intuitive editing interfaces. Today, the prevailing interaction metaphor for object deformation interfaces is click-and-drag of individual controls. For example, skeletal bones are used for characters [BW76, MTLT88], while lattice grids (FFD) are used for free-form objects such as point clouds or triangle soups [SP86, SF98, MJBF02]. Unfortunately, this type of interaction does not accommodate artistic principles. Recently many researchers investigated sketching as a more natural way of editing 3D content [DIC*03, HQ03, KG05, NSACO05, ZNA07, KSvdP09, GCR13, HMC*15].

A deformation is defined via sketching a pair of curves: a reference and a target. Several works exploit the mesh geometry [NSACO05, ZNA07, KSvdP09] or rig of the character [DIC*03, GCR13] (e.g. its skeleton) to compute the reference curve. Unfortunately, these techniques do not allow deforming multiple objects simultaneously with a single stroke. In contrast, we automatically compute multi-objects abstractions (subjective curve elements described in section 4) which allow the user to directly deform multiple objects with a single stroke.

Automatic shape abstractions. To allow the user to deform raw scenes with a single stroke, we automatically compute abstractions in the form of principle curves and abstract contours. While it would be straightforward to use rigged objects with our method (skeletons representing a good abstraction for many shapes), we decided to work in the general case where most objects of the scene are not rigged (trees, fences, furniture, etc.). Automatic skeletonisation technics compute a skeleton from a mesh or point cloud [ATC*08, TZCO09] but yield skeletons with noisy branches that can rarely be sketched directly. Another approach is to assume a parametric curve such as a spline and optimize its shape as to conform to the object using an iterative closest point framework [KWT88]. We build upon this approach as it is well suited for sketching the smooth splines. Other works on shape abstractions seek to remove details while preserving a coarse version of the shape [MDS09, MZL*09]. However, pruning parts of objects solely based on geometric features does not result in proper outer-abstractions such as contours in the case of pointy objects, e.g. branches of a tree or poles of a fence.

Direct deformation of scenes. One of the challenges of scene deformation is the fact that different objects are typically deformed with different algorithms (e.g. skeleton, lattice-FFD, etc). Hence the ability to quickly edit an existing raw scene requires a unified approach to multi-object deformation. One approach to direct deformation of 3D objects is to formulate the deformation directly on

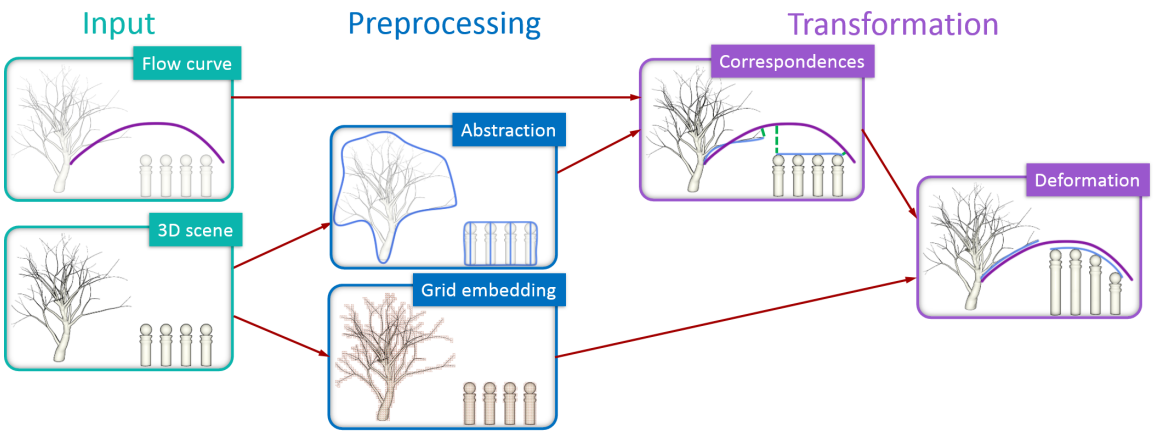


Figure 3: *Overview of our approach applied on a simple scene. The user gives as input a 3D scene, on which our system automatically computes SEcurves (in the form of objects abstractions) and a 2D grid embedding of the objects (used for deformation). Then, when the user provides a flow curve, SEcurve pieces are selected and associated to corresponding flow curve pieces based on a geometric closest-point approach, and the grids are deformed as to have these SEcurve pieces match the shape of the flow curve, in screen-space.*

the mesh vertices, typically by penalizing the distortion of triangle elements [SCOL*04a, SA07] while matching some vertex position constraints. Unfortunately, this approach does not support disconnected meshes and point clouds, such as fluids. Additionally, the complexity of the deformation is proportional to the number of vertices, which can impede interactive refinement. An alternative is to automatically compute an embedding of the geometry [SSP07, BPWG07, SDC09] and use similar deformation formulations on the grid embedding elements instead. We build upon this approach by computing a 2D grid embedding of the 3D elements, in sceen-space, that reflects the intrinsic shape of the objects.

Others have used whole scene 2D grid embeddings, mainly in the context of image warping [CAA10, OEYK12]. Unfortunately, a single 2D grid does not reflect the structural components of the objects in the scene and can lead to undesirable distortions. For example, bending a branch will distort the space around it. Similar to spatial warping, rendering using a nonlinear projection [CS04, CSB*05, BSCS07] allows to deform images. These techniques do not allow deforming individual parts of objects with ease and, as with spatial warping, do not allow deforming objects without warping the space around them (e.g. the background). In contrast, we benefit from the fast computation of 2D deformation, while preserving the intrinsic shape of objects.

3. Overview

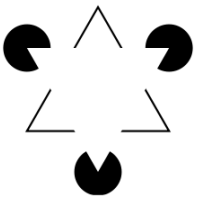
Our interface is designed to allow users to apply the principle of *movement* in a fast and natural way. Our approach is based on deforming the geometry of a scene as to form *subjective curves* aligned with user-provided *flow curves* and/or smoothly pointing towards a user-specified *center-point*. Devising such a tool requires overcoming several challenges.

First, we need to determine which contrasts in the scene could be deformed to form strong subjective curves. Since our goal is different from computing existing subjective curves in images, we compute partial subjective curve elements that we call *SEcurves*, short for *Subjective Elements*. While detecting these is highly ambiguous, we observed that in many cases, subjective curves are formed from a skeletal curve of objects, or from parts of abstract outer-contours. Based on these observations, we compute *principal curves* for thin objects and *abstract contours* for complex ones, as well as groups of objects (Section 4). While these SEcurves cover a wide range of cases, users may sketch additional ones if desired.

As a means of forming coherent subjective curves via scene deformations, we introduce a sketch-based interface called *Flow Curves*. The flow curves are defined as *shape* constraints for SEcurves in their vicinity (Section 5). We offer two intuitive ways of specifying flow curves: 1-by sketching strokes, which provides direct control onto the subjective curves of the scene and 2-by sketching a center-point which automatically generates a coherent network of flow curves converging towards it. This center-point provides coordinated control onto the scene's movement as a whole.

To match the SEcurves to their corresponding flow curves, we need a deformation method that allows deforming different types of objects (including point clouds such as liquids), requires little manual setup time, and preserves the intrinsic shape of objects (i.e. does not distort space). Our solution to these requirements is to compute a 2D grid embedding in screen-space for each individual object in the scene (Section 6.1), and to formulate the constrained deformation on the grid embedding (Section 6.3). Constraints are the target position of SEcurves and the conservation of the objects placement layout.

4. Subjective Curve Elements



Our visual system perceives curves that are subjective and not entirely explicit: we fill-in the space between contrast edges and points, and even skip over parts of objects at high curvature points, in favor of longer smoother curves. For example, in the image on the left, we naturally see two solid triangles while none is entirely present. In this section, we identify strong edges and

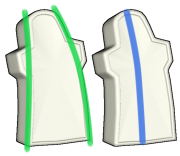
shape features that could be elements of subjective curves—we call them SEcurves.

The goal of our tool is to facilitate coordinating SEcurves into forming long and coherent subjective curves—as opposed to finding existing subjective curves as is done in the computer vision literature. Computing edges is often noisy as well as impractical for sketching. Additionally, computing every possible edges that could be used for creating movement in a scene would make the selection process ambiguous (how to pick the right SEcurve from a soup of edges?). By observing users sketching their own subjective curves, we found that they are often formed from two categories of recurrent SEcurves. The first is *principal curves* which abstract thin objects, and the second is *abstract contours* which join the tips and edges of complex shapes such as trees, plants and fences, as well as join together groups of nearby objects.

Based on these observations, we first compute for each object in the scene both their principal curve (a generalization of the principal axis, detailed in Section 4.1) and abstract contour (Section 4.2). Then we decide which is best suited based on a measure of thinness. We estimate thinness as the variance of the point-wise distance d_i between each point of the principal curve and the contour. A small variance means that the principal curve is a better approximation of the object than the abstract contour. Hence, if $\sum d_i^2/N - (\sum d_i/N)^2$ is below a threshold, we keep the principal curve, otherwise we keep the contour.

Shapes may form a principal curve individually, but when located close to each other join and form a stronger contour, as shown by the four poles in Fig. 3. Hence, in a second step, we measure whether multiple principal curves are close to one another, in which case we compute an abstract contour around this group of objects. As a result, the group holds individual principal curves *and* a group-wise abstract contour.

4.1. Principal Curves



The tombs on the left illustrate how principal curves (the blue curve on the right) are a practical approximation of side contrast edges (the green curves on the left). We propose a method to automatically compute principal curves for arbitrary objects in the scene. Principal curves have been studied in statistics as a non-parametric model of curved manifolds and a generalization of the principal (vector) component [Sil85]. We follow a spline regression framework, where the correspondence between the data points and the spline is not known a priori but must be estimated—typically in an iterative closest point (ICP) fashion.

Applying this technique to 3D meshes leads to problems of its own. First, the distribution of points along the surface can influence the shape of the principal curve, hence we need to re-sample the surface uniformly. Second, minimizing the distance alone can lead to spurious curves when applied to shapes that are not equally spaced-out w.r.t. their center-line (such as the tree in Fig. 4). We address this issue with iterative regularization, i.e. we penalize the solution w.r.t. the last computed curve between consecutive ICP iterations.

Our first step is to compute a uniform sampling of the surface mesh. As we work in screen space, we use the depth map and sample pixels whose depth is lower than 1, yielding 2D surface points $X = \{x_1, x_2, ..., x_N\}$. Note, we believe our approach easily extends to the 3D case with 3D surface re-sampling. We initialize the spline $c_0(s)$ to the principal axis x_{axis} of the surface by computing the largest eigen vector of the sample covariance matrix formed from all the surface points $Cov(X,X) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T$, using principal component analysis. We use cubic Hermite splines in our implementation, and initialize the length of $c_0(s)$ to the size of the shape in its principal direction.

Then, we refine the curve by iteratively solving the following problem. At each iteration, we compute the parametric correspondence s_i^* of every surface point x_i to the current principal curve by computing the closest point ($s_i^* = \text{argmin}_s \|x_i - c_k(s)\|$). And then we minimize the euclidian distance between both points, w.r.t. the spline degrees of freedom, while penalizing deviations in shape from the previous curve:

$$c_{k+1}(s) = \min_{c(s)} w_1 \sum_i \left\| x_i - c(s_i^*) \right\|^2 + w_2 \int_s \left\| \frac{\partial c(s)}{\partial s} - \frac{\partial c_k(s)}{\partial s} \right\|^2 .$$

We then increase k and iterate until no more improvement is gained. In our implementation, we used a weight w_2 1000 times bigger than w_1. To compute this integral, as well as all the following ones, we discretize the Hermite curve into equally-spaced points and sum over the desired values at these points. Examples of principal curves computed by our algorithm are shown in Fig. 4.

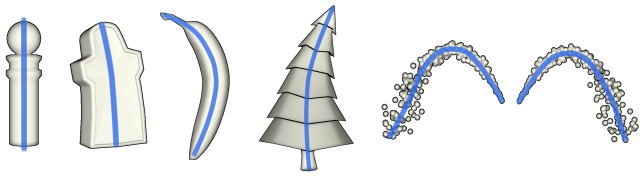


Figure 4: *Thin objects can be abstracted by a principal curve, which approximates their side edges.*

4.2. Abstract Contours

Subjective elements are often perceivable around the tips and edges of complex shapes, or joining groups of close objects. Outer-contour groupings are often what we naturally sketch when drafting the shape of objects and as such are an important class of SEcurves (see Fig. 3 middle).

We also use spline fitting to compute the abstract (outer) contour of an object or group of objects, but here using a closed curve. We minimize the area $E_A(c)$ enclosed by the curve $c(s)$ while preventing points of the object from moving outside the curve, resulting in a penalizing energy term $E_O(c)$. The level of abstraction is controlled by weighting the curvature penalization term $E_C(c)$. For the closed curve $c(s)$, we use a cubic Hermite spline and initialize it to the bounding box around the object. We detail bellow the optimization problem we solve.

Area $E_A(c)$. We penalize the area enclosed by the curve, scaled by the object's bounding box area a_0. We discretize the curve into equal-length segments and compute the enclosed area by summing

all the signed areas under these segments, using the determinant of the 2×2 matrix formed by setting the consecutive curve samples as columns: $[c(s_i)\, c(s_{i+1})]$, resulting in:

$$E_A(c) = \frac{1}{2 \cdot a_0} \left| \sum_i Det\,[c(s_i)\, c(s_{i+1})] \right|.$$

Outside points $E_O(c)$. We compute the number of edge pixels p_i outside the curve $c(s)$. To do so, we trace vertical lines (in the $\vec{y}$ direction) starting at each edge pixel p_i, and count the number $I(p_i)$ of intersections with $c(s)$. The parity of $I(p_i)$ determines whether it is inside (odd) or outside (even). Note that this term acts as a hard constraint and thus has a large weight w_O.

$$E_O(c) = \sum_i 1 - (I(p_i) \% 2).$$

Curvature $E_C(c)$. We penalize the curvature of the contour $c(s)$, which corresponds to its second derivative (we approximate it numerically using equally-spaced samples). Controlling a soft curvature penalty weight controls the level of abstraction of the contour. Indeed, if $c(s)$ is allowed to be flexible, it can move inside cavities of the object in order to minimize area.

$$E_C(c) = \int_s \left\| \frac{\partial^2 c(s)}{\partial^2 s} \right\|^2.$$

The total energy is non-linear and the term E_O is discontinuous. Hence we minimize the sum of energies using stochastic optimization with Covariance Matrix Adaptation (CMA):

$$E(c) = w_A E_A(c) + w_O E_O(c) + w_C E_C(c),$$

setting the parameters to $w_A = 100$, $w_O = 10^5$, and $w_C = 5$ in our results. Note that there can be infinite reparameterizations of $c(s)$ that yield the same total energy values; this can cause global stochastic optimization to loop without significant gains. We alleviate this issue with a stopping criteria based on the relative improvement of the objective function. Examples of abstract contours computed by our algorithm are shown in Fig. 5.

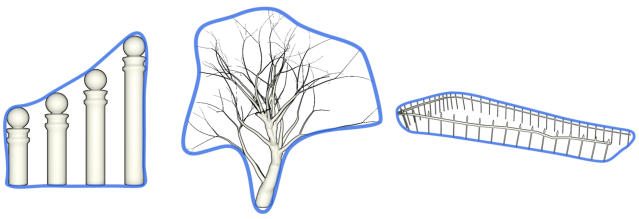


Figure 5: *Complex shapes, as well as groups of objects, form abstract contours. We compute them using spline fitting, i.e. by minimizing the area of a closed spline curve while ensuring that points of the object remain inside the closed curve. The level of abstraction is controlled by penalizing curvature.*

5. Flow Curves

We define flow curves as shape constraints over SEcurves. The goal of flow curves is to design smooth movement and we thus first describe how to control the smoothness of the sketched flow curve—if desired by the user. A second goal is coherence, not only for individual subjective curves, but also for the scene as a whole. Hence, the second control we provide to the user is a center-point, which generates a coherent network of smooth flow curves.

5.1. Flow Curve from Sketched Stroke

We can control the smoothness of a flow curve $\gamma(s)$ drawn by the user via spline fitting and curvature penalization. We fit a cubic Hermite curve to the stroke samples f_i while penalizing the second derivative of the curve:

$$\min_{\gamma(s)} \sum_i w_1 \|f_i - \gamma(s_i)\|^2 + w_2 \int_s \left\| \frac{\partial^2 \gamma(s)}{\partial^2 s} \right\|^2, \qquad (1)$$

which we optimize w.r.t. all the spline degrees of freedom, i.e. the position and tangent of every control point. The number of control points n_γ is determined by the length l_γ of the sketched stroke. Observing that a flow curve of length $w_s/4$ (w_s being the width of the screen) is well represent by 4 control points, we used $n_\gamma = \left\lceil 3 \cdot \frac{4 \cdot l_\gamma}{w_s} \right\rceil + 1$.

5.2. Flow Curves Network from Center-Point

When the user sketches a circle shape, we interpret its center of mass as the center-point and generate a network of flow curves. To be consistent with the current scene, the generated network depends on the scene configuration and the present SEcurves. But its definition is unfortunately not unique; for example, on the top row of Fig. 6, we can see that each subjective element individually points towards the center-point, while on the second row, they move in accordance, eventually bending towards the center-point. Hence, we compute a parametric space of flow curves and provide the user with a flow radius parameter d_{Max}, indirectly controlling the number of flow curves generated.

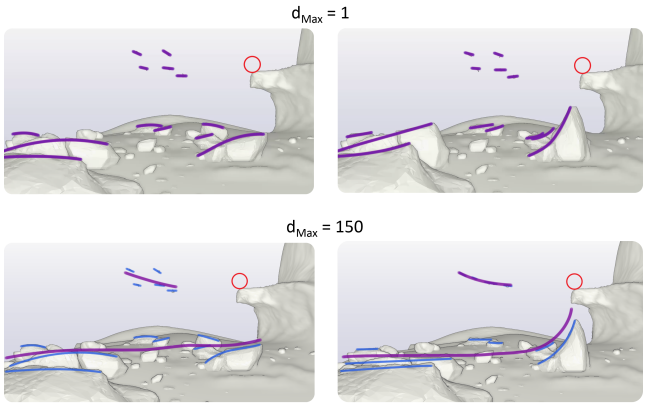


Figure 6: *We allow the user to specify only a center-point, from which several flow curves are computed—each converging to the center-point. We first initialize the flow curves (in purple) from the SEcurves (in blue), as shown in the left column. The user can control the amount of grouping for the generated flow curves through a parameter d_{Max}. The right column shows the deformed curves and scenes.*

Our idea is to first compute average curves from the SEcurves present in the scene, before bending them in order to make them coherently flow towards the center-point.

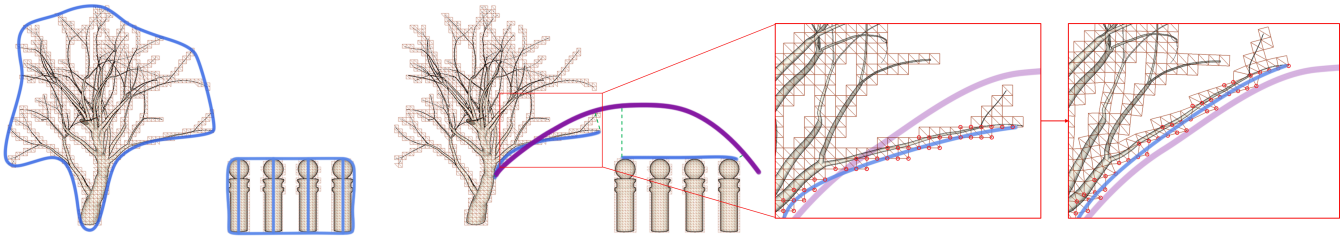


Figure 7: *To deform arbitrary objects in the scene, we compute a 2D embedding of the object, in screen-space, which is then used for deformation. The grid is a coarse approximation that preserves its intrinsic shape when deformed. The SEcurves are expressed as linear functions of the 2D grid triangles, shown with red points. We optimize for a deformation of the grid as to match the shape of DEcurves (in blue) to the shape of their corresponding flow curves (in purple).*

Average curves. To estimate the average flow curve in a region of the scene, we first extrapolate each SEcurve in both directions by interpolating the direction of the nearest SEcurves. To do so, we start with an extremity $c(1)$ (or $c(0)$) and look for the nearest points $c_j(s^*)$ on the other SEcurves at a distance below d_{Max}. Then we integrate the curve's position by averaging the directions with an inverse distance weighting scheme:

$$c(s+\Delta s) = c(s) + \Delta s \frac{\sum_{j=1}^{N} r(\|c_j(s^*) - c(s)\|) \frac{\partial c_j(s^*)}{\partial s}}{\sum_{j=1}^{N} r(\|c_j(s^*) - c(s)\|)},$$

where r is a radial kernel function used to interpolate the directions of the neighboring SEcurves; we used $r(d) = \frac{1}{1+d^2}$. Note that we only include in the sum the points that verify $\|c_j(s^*) - c(s)\| < d_{Max}$ and $\angle\left(\frac{\partial c_j(s^*)}{\partial s}, \frac{\partial c(s)}{\partial s}\right) < \theta_{Max}$. This condition on the angle between tangents allows avoiding influence from inappropriate curves and supporting curves crossing each other (we used $\theta_{Max} = \frac{\pi}{3}$). We stop when there is no more point verifying these conditions.
Finally, knowing which SEcurves influenced the extrapolation of which other ones, we cluster the resulting curves. We then compute their average curve $\bar{c}(s)$ (by imposing a common parametrization on every clustered curve), that we smooth into a Hermite curve using eq.(1).

Bending curves. We now bend the computed average curves as to have them smoothly join the center-point (Fig. 6 right). For each curve $\bar{c}(s)$, we fix an extremity control point (its position and tangent) and deform the remaining part as to point towards the center-point, giving a flow curve $\gamma(s)$. To know in which direction to deform the curve (i.e. choosing between fixing $\bar{c}(0)$ or $\bar{c}(1)$), we deform both cases and measure the deformation magnitude (difference in shape), and we select the case yielding the smallest deformation. Note that we also use this measure to remove flow curves that would deform the scene too drastically.

Having a flow curve *join* a center-point means that if we extrapolate its path it should eventually touch the center-point. We approximate this measure with the angle between the curve's tangent at its tip and the vector between the tip position and the center-point, resulting in $\theta_p = \angle(\gamma(1) - x_p, \frac{\partial \gamma(1)}{\partial s})$. Also, the length of the curve $l(\gamma)$ is constrained to stay close to the initial length $l(\bar{c})$. Hence by minimizing these values and controlling the smoothness via curvature, we obtain the following problem:

$$\min_{\gamma(s)} \quad w_1 \theta_p^2 + w_2 |l(\gamma) - l(\bar{c})| + w_3 \int_s \left\| \frac{\partial^2 \gamma(s)}{\partial^2 s} \right\|^2 \tag{2}$$

$$\text{subject to} \quad \gamma(0) = \bar{c}(0), \ \frac{\partial \gamma(0)}{\partial s} = \frac{\partial \bar{c}(0)}{\partial s},$$

where the direction term has a larger weight ($w_1 = 500$) than length conservation ($w_2 = 0.5$) and curve smoothing ($w_3 = 7$), to ensure pointing towards the center-point.

6. Direct Deformation of Scene Objects

Deforming the scene objects as to have SEcurves match the shape of flow curves would require re-computing them at each step of the process. We circumvent this problem by using a common 2D grid embedding of both the object geometry *and* SEcurves (Section 6.1). The correspondence between SEcurves and flow curves is automatically computed using proximity and shape criteria (Section 6.2). From the correspondence, we derive position constraints on grid vertices for matching the curves, and then minimize an as-rigid-as-possible energy expressed on the grid embedding vertices (Section 6.3).

6.1. 2D Grid Embedding

Since movement edits are screen-space refinements, we utilize a deformation representation designed to operate in screen-space. Given a 3D object with vertices $V = \{v_1, v_2, ..., v_N\}$, we build a 2D triangle grid embedding defined by vertices x_g by uniformly tessellating the object's bounding box in screen space at a fixed resolution (15 pixels in our examples). We then remove all triangles not occupied by the mesh so that the grid closely conforms to the object's projected shape (see Fig. 7, left). We parameterize the 3D object vertices v_i to their projected position $\mathbf{P}v_i = \phi_i(x_g)$ in the grid using barycentric coordinates, together with depth values in view space $d_{z_i} = \|v_i - \mathbf{P}v_i\|$. Here, $\mathbf{P}$ is the perspective projection matrix. After deforming grid elements x_g into x'_g (section 6.3), we recover the deformed mesh position v'_i with:

$$v'_i = \phi_i(x'_g) + d_{z_i} x_{dir}, \tag{3}$$

where x_{dir} is the unit vector between the camera position and point $x'_g(i)$ on the screen.

6.2. Automatic Correspondence

Given a flow curve γ and SEcurves c_i, we automatically select parts of SEcurves that will be transformed and compute their correspondence to the flow curve. The result are new curve *segments* c_l and γ_l, sharing a common parameterization s: $c_l(s) \rightarrow \gamma_l(s)$. Our solution consists of two steps. We first select curve segments $\tilde{c}_k$ and $\tilde{\gamma}_k$ based on a closest point approach combined with a curve similarity measure [CG97]. Because this initial selection may include SEcurve segments that are either too small or in conflict with one another (an example is the tree's contour in Fig. 7, which could have both parts of its base selected), we filter undesirable segments based on a score.

The initial segmentation process computes all the corresponding SEcurve segments $\tilde{c}_k$ and flow curve segments $\tilde{\gamma}_k$, by going through all the points of the discretized curve c_i and storing the ones $c_i(s_j)$ whose closest point $\gamma(s_j^*)$ is under a threshold $\|\gamma(s_j^*) - c_i(s_j)\| < d_m$ (we use $d_m = 100$ pixels), and whose angle between tangents is under a threshold $\angle\left(\frac{\partial c_i(s_j)}{\partial s}, \frac{\partial \gamma(s_j^*)}{\partial s}\right) < \theta_m$ (we use $\theta_m = \frac{2}{3}\pi$). The result is a set of corresponding curve segments $\tilde{c}_k \rightarrow \tilde{\gamma}_k$.

Then, to filter undesirable segments, we compute a score $S(\tilde{c}_k)$ for each segment, based on its length and mean distance to the flow curve (i.e. mean of $\|\tilde{\gamma}_k(s_j) - \tilde{c}_k(s_j)\|$):

$$S(\tilde{c}_k) = w_1\, l(\tilde{c}_k) + w_2\, \frac{1}{1 + d(\tilde{\gamma}_k, \tilde{c}_k)}.$$

We keep only segments whose score is above S_{min}, resulting in the set of segments $\{c_l, \gamma_l\}$. We use values $w_1 = 1$, $w_2 = 10^5$ and $S_{min} = 150$ in our implementation.

6.3. Deformation

Given a flow curve segment γ_l and its corresponding SEcurve segment c_l (computed in section 6.2), our goal is to deform the grids as to match the shape of c_l to the shape of γ_l:

$$\frac{\partial c_l(s)}{\partial s} = \frac{\partial \gamma_l(s)}{\partial s}. \tag{4}$$

We achieve this by computing an as-rigid-as-possible grid deformation that satisfies this differential constraint. For increased speed, we use a fast implementation of ARAP which only solves for hard position constraints. We thus turn the differential constraints into hard position constraints of the grid vertices w_i. We also add positional constraints to preserve the initial placement of objects in the scene.

Layout positions constraints. We compute intersections between objects and create position constraints for all grid vertices to which these intersections project: $w_i^*, \forall i \in L$, where L is the set of all constrained positions' indices on a given object grid. When no position constraints are detected for a particular object, the natural behavior is to translate, allowing their SEcurves to match the flow curve, as is the case for the birds in Fig. 8.

Shape constraints into position constraints. We approximate the differential expression above (eq. 4) in terms of positional constraints by translating the flow curve γ_l to the position on the subjective curve that is closest to a constrained grid point: $\gamma_l(s) := \gamma_l(s) + (c_l(s^*) - \gamma_l(s^*))$, where $s^* = \text{argmin}_s \|c_l(s) - w_i^*\| \forall i \in L$. When there are no constrained grid positions ($L = \emptyset$), we translate the SEcurve to the nearest flow curve point: $c_l(s) := c_l(s) + (\gamma_l(s^*) - c_l(s^*))$, where $s^* = \text{argmin}_s \|c_l(s) - \gamma_l(s)\|$. We then use γ as a position matching constraint. We derive the appropriate constrained grid positions similarly to other sketch-based deformation methods [ZNA07], by mapping the relative position of vertices close to c_l onto γ_l.

Given these constraints, we wish to compute a deformed grid that respects the constraints while minimizing the local distortion of grid elements. To do so, we compute a deformation energy $E_{shape}(\mathbf{T})$ that measures the non-rigidity of triangle transformations $\mathbf{T} = T_1, T_2, ...$, where T_i transforms triangle i from its undeformed to its deformed state. As such deformations are well studied, we employ an as-rigid-as-possible deformation [SCOL*04b] to solve the following problem:

$$\begin{aligned} \min_{x_g} \quad & E_{shape}(\mathbf{T}) \\ \text{subject to} \quad & c_l(s) = \gamma_l(s) \qquad \forall l \\ & w_i = w_i^* \qquad \forall i \in L \end{aligned} \tag{5}$$

Vertex position constraints are enforced by construction by removing them from the set of unknown variables. Due to the 2D formulation, the overall system is solved efficiently, leading to interactive performance. After solving for the grid deformation, we recover the mesh vertex positions using eq. (3).

7. Results and Discussion

Fig. 8 shows how four different scenes were edited using sketched flow curves. We refer to them as 1-Fountain, 2-Octopus, 3-Cliff and 4-Cemetery, following the top-to-bottom order. The last three scenes (2,3,4) are inspired by existing artworks (the ones shown in Fig. 2). The various scenes allowed us to evaluate our interface on different types of objects (connected and disconnected meshes such as the poles of a fence) and scene configurations.

The second functionality provided by flow curves is the ability to sketch a center-point circle. The center-point—taken as the center of the sketched curve—automatically generates coherent flow curves to deform the scene (as described in Section 5). In both Fig. 9 and accompanying video, we show the possibility for the user to interactively manipulate the center-point while our system generates coherent flow curves that directly deform the scene.

To evaluate the efficiency of our tool, we invited two artists with more than 7 years of professional experience. We asked them to deform 5 scenes using their favorite Software (Maya) in order to create a personalized movement. Then they used our flow curves to create similar deformations on the same initial scenes. An example of this evaluation is shown in Fig. 10, as well as in the beginning of our accompanying video. The additional 4 deformation results are available in a supplementary document.

Both artists were impressed by the ease of use of our tool and appreciated the direct use (i.e. no manual setup). Table 1 compares the times taken both in Maya and using our Flow curves, as well as the required number of mouse clicks: our interface is on average 6 times faster and require 8 times less clicks. Flow Curves was manifestly much more intuitive and faster than deformers available in

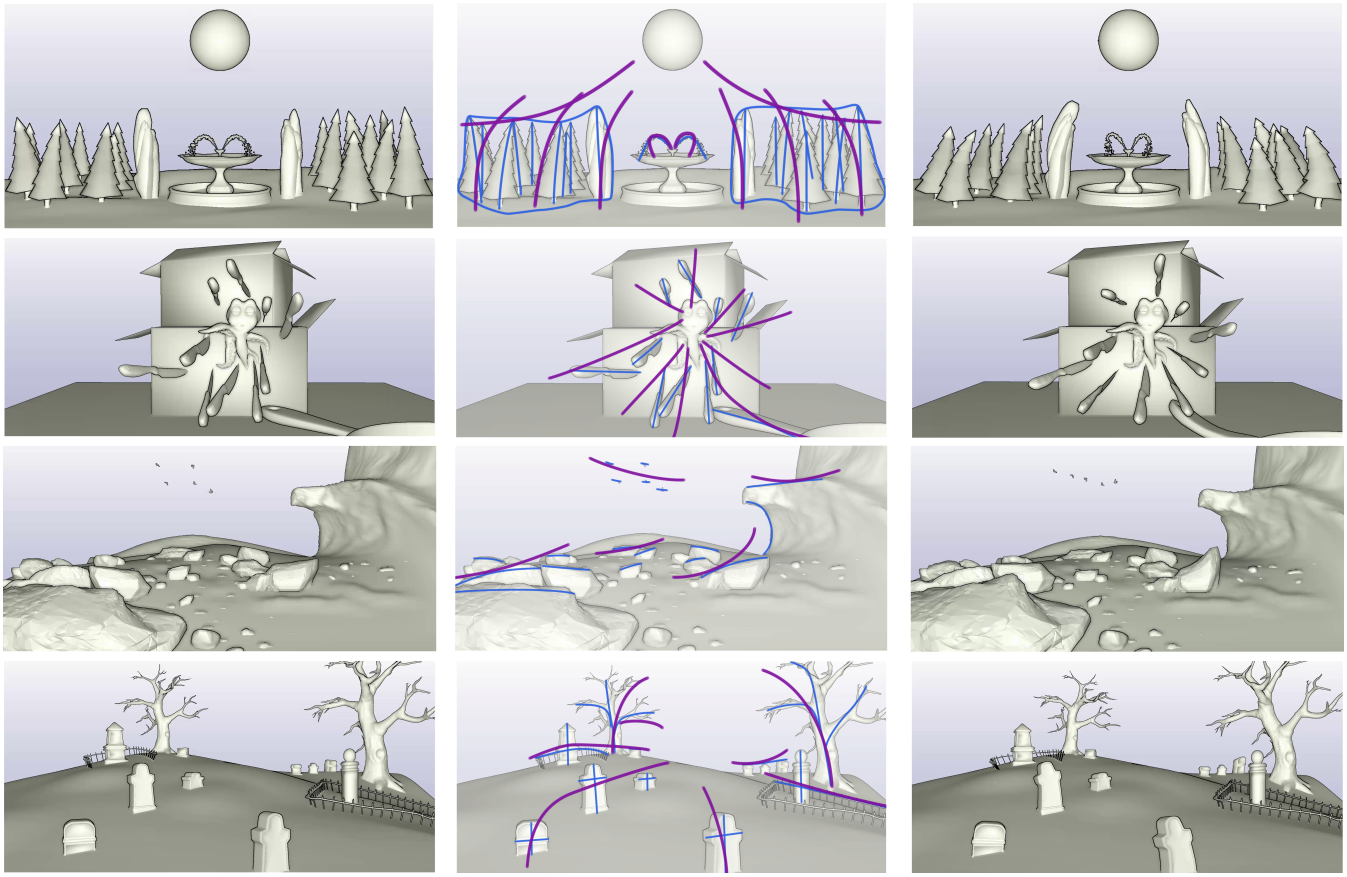


Figure 8: *The left column is the input scene. In the middle column are the SEcurves in blue and the sketched flow curves in purple. The SEcurves were computed automatically in the first two rows, and manually specified in the last two rows. The last column is the final deformed scene. Note that we preserve the initial object placement layout: objects in contact remain so during deformation, as is the case with the trees and ground, or the knives and boxes. When there are no contacts, such as the small birds in the third row, the flow curves automatically become position constraints. See section 6.3 for more details.*

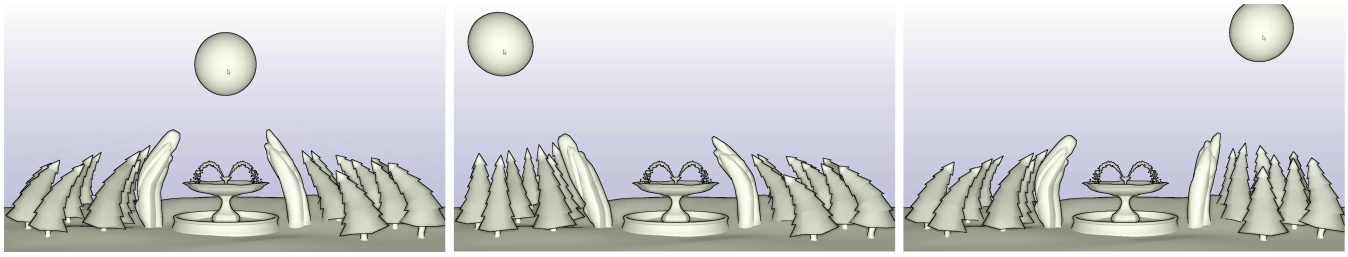


Figure 9: *The user can control the center-point, thereby interactively inducing flow-preserving deformations over the whole scene.*

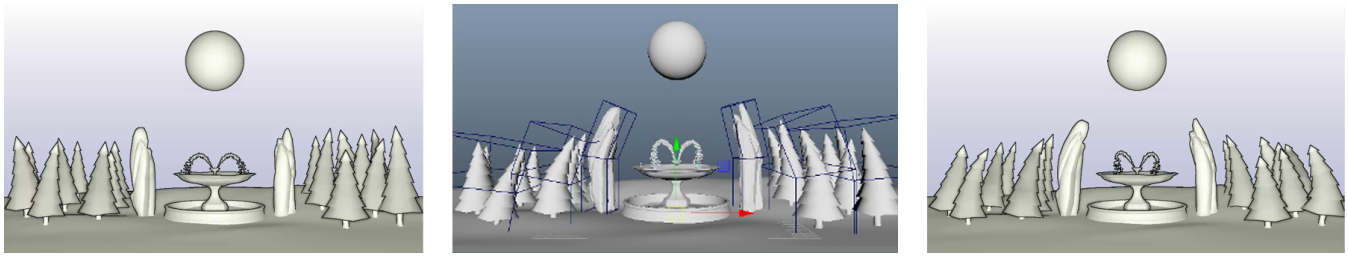


Figure 10: *Footage from our comparison between Flow Curves and Maya. Starting from the input scene on the left, an expert animator took 6 minutes to complete the scene in the middle using Maya, and only 23 seconds to complete the scene on the right using Flow Curves.*

Maya, however it did not offer the same level of control. In fact, both artists mentioned that they would like to use Flow Curves as a fast and natural way to deform their scenes, and to then use additional deformers if more detailed refinements are needed.

	Artist 1				Artist 2			
	Time		Mouse clicks		Time		Mouse clicks	
Scene	Maya	Ours	Maya	Ours	Maya	Ours	Maya	Ours
1	5m 58s	23s	174	10	5m 02s	26s	138	12
2	1m 37s	25s	52	12	2m 22s	26s	103	13
3	1m 50s	51s	61	21	2m 07s	33s	75	19
A	4m 01s	59s	121	19	5m 06s	37s	144	13
B	1m 48s	17s	64	6	3m 10s	14s	92	6

Table 1: *Comparison of the time and mouse clicks required by professional artists to deform the same scenes in Maya and with Flow Curves. The scenes and deformations are shown in the supplementary material.*

8. Limitations and Future Work

In our effort to offer a fully automatic interface that works on raw scene geometry with little manual setup, we made assumptions about the user's intentions. In particular, we automatically compute the correspondence between SEcurves and flow curves (Section 6.2), but this may not reflect the expectation of the user (he or she may only want to modify closer SEcurves, or maybe sketch larger deformations). This issue is easily removed with manual intervention, such as by sketching over the desired SEcurve portions that one would like to modify. Also, when computing SEcurves in the scene, the performance of our algorithm depends on the segmentation of the scene. For example, the fence in Fig. 5 is one object and we obtain an outer-contour, while if it was segmented into several objects, a principal curve would be obtained for each individual pole.

In this work we have focused on static scenes, and left dynamic aspects of movement for future work. For example, moving objects can form subjective curves over time due to visual persistence—our mind keeps track of previous positions and traces imaginary paths over time. It would be interesting to investigate modeling such dynamic subjective curves for improved animation quality and appeal. In particular, liquids have to this day little artistic guidance, and we could envision using flow curves to dynamically guide the surface over time, as to form visually pleasing imaged shapes (or paths).

We believe that our characterization of flow curves as constraints over feature subjective curves in the scene opens new opportunities beyond scene deformation. For instance, the same constraints could be used to guide the modeling or lighting of a scene. For example, a procedural modeling algorithm could use our constraints to generate objects whose SEcurves match user-specified flow curves through shading control.

9. Conclusion

We introduced a new sketch-based interface called Flow Curves that provides intuitive ways of deforming whole scenes for visual coherence. With our interace, the user can more directly and efficiently design and edit the movement of a scene, simply by sketching flow curves, or sketching a circle center-point. Our direct deformation method requires little to no manual setup time and preserves the intrinsic shape of objects together with the initial objects placement layout. We demonstrated that our approach takes considerably less time to operate than current modern 3D digital tools, which require setting up diverse deformers and performing multiple edits from different view-points.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 642841. We would like to thank the artists Maurizio Nitti and Alessia Marra for their useful input on the project and their help making results. We would also like to thank the BlendSwap users OliverMH and pndrdm for respectively the rocks and fountain models in scene 1.

References

[ATC*08] Au O. K.-C., Tai C.-L., Chu H.-K., Cohen-Or D., Lee T.-Y.: Skeleton extraction by mesh contraction. *ACM Trans. Graph. 27*, 3 (2008), 44:1–44:10. 2

[BMSG09] Bailey R., McNamara A., Sudarsanam N., Grimm C.: Subtle gaze direction. *ACM Trans. Graph. 28*, 4 (2009), 100:1–100:14. 2

[BPWG07] Botsch M., Pauly M., Wicke M., Gross M.: Adaptive space deformations based on rigid cells. *Computer Graphics Forum 26*, 3 (2007), 339–347. 3

[BSCS07] Brosz J., Samavati F. F., Carpendale M. S. T., Sousa M. C.: Single camera flexible projection. In *Proceedings of the 5th International Symposium on Non-photorealistic Animation and Rendering* (2007), pp. 33–42. 3

[But02] Button B.: *Nonlinear Editing: Storytelling, Aesthetics, and Craft*. CMP Books, 2002. 1

[BW76] Burtnyk N., Wein M.: Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *ACM Transactions on Graphics (TOG) 19*, 10 (1976), 564–569. 2

[CAA10] Carroll R., Agarwala A., Agrawala M.: Image warps for artistic perspective manipulation. *ACM Trans. Graph. 29*, 4 (2010), 127:1–127:9. 3

[CDF*06] Cole F., DeCarlo D., Finkelstein A., Kin K., Morley K., Santella A.: Directing gaze in 3d models with stylized focus. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques* (2006), pp. 377–387. 2

[CG97] Cohen S. D., Guibas L. J.: Partial matching of planar polylines under similarity transformations. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (1997), pp. 777–786. 7

[CS04] Coleman P., Singh K.: Ryan: Rendering your animation nonlinearly projected. In *Proceedings of the 3rd International Symposium on Non-photorealistic Animation and Rendering* (2004), pp. 129–156. 3

[CSB*05] Coleman P., Singh K., Barrett L., Sudarsanam N., Grimm C.: 3d screen-space widgets for non-linear projection. In *Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia* (2005), pp. 221–228. 3

[DIC*03] Davis J., Igarashi M., Chuang E., Popovic' Z., Salesin D.: A sketching interface for articulated figure animation.

Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation (2003), 320–328. 2

[GCR13] GUAY M., CANI M.-P., RONFARD R.: The Line of Action: an Intuitive Interface for Expressive Character Posing. *ACM Transactions on Graphics 32*, 6 (2013). 2

[Gla13] GLATSTEIN J.: *Formal Visual Analysis: The Elements & Principles of Composition*. 2013. 2

[HMC*15] HAHN F., MUTZEL F., COROS S., THOMASZEWSKI B., NITTI M., GROSS M., SUMNER R. W.: Sketch abstractions for character posing. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (2015), pp. 185–191. 2

[Hor83] HORN B. K. P.: The curve of least energy. *ACM Trans. Math. Softw. 9*, 4 (1983), 441–460. 2

[HQ03] HUA J., QIN H.: Free-form deformations via sketching and manipulating scalar fields. In *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications* (2003), pp. 328–333. 2

[IMH05] IGARASHI T., MOSCOVICH T., HUGHES J. F.: As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*, 3 (2005).

[KG05] KHO Y., GARLAND M.: Sketching mesh deformations. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games* (2005), pp. 147–154. 2

[KSvdP09] KRAEVOY V., SHEFFER A., VAN DE PANNE M.: Modeling from contour drawings. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling* (2009), pp. 37–44. 2

[KWT88] KASS M., WITKIN A., TERZOPOULOS D.: Snakes: Active contour models. *International Journal of Computer Vision 1*, 4 (1988), 321–331. 2

[LCWCO10] LIU L., CHEN R., WOLF L., COHEN-OR D.: Optimizing photo composition. *Computer Graphics Forum 29*, 2 (2010), 469–478. 2

[LMLF15] LIU T., MCCANN J., LI W., FUNKHOUSER T.: Composition-aware scene optimization for product images. *Comput. Graph. Forum 34*, 2 (2015), 13–24. 2

[MDS09] MI X., DECARLO D., STONE M.: Abstraction of 2d shapes in terms of parts. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering* (2009), pp. 15–24. 2

[MJBF02] MILLIRON T., JENSEN R. J., BARZEL R., FINKELSTEIN A.: A framework for geometric warps and deformations. *ACM Trans. Graph. 21*, 1 (2002), 20–51. 2

[MTLT88] MAGNENAT-THALMANN N., LAPERRIÈRE R., THALMANN D.: Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics Interface '88* (1988), pp. 26–33. 2

[Mum94] MUMFORD D.: Elastica and computer vision. In *Algebraic Geometry and its Applications* (1994), pp. 491–506. 2

[MZL*09] MEHRA R., ZHOU Q., LONG J., SHEFFER A., GOOCH A., MITRA N. J.: Abstraction of man-made shapes. *ACM Trans. Graph. 28*, 5 (2009), 137:1–137:10. 2

[NSACO05] NEALEN A., SORKINE O., ALEXA M., COHEN-OR D.: A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph. 24*, 3 (2005), 1142–1147. 2

[OBP*13] ÖZTIRELI A. C., BARAN I., POPA T., DALSTEIN B., SUMNER R. W., GROSS M.: Differential blending for expressive sketch-based posing. In *Proceedings of the 2013 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2013). 2

[OEYK12] ORBAY G., ERSIN YÜMER M., KARA L. B.: Sketch-based aesthetic product form exploration from existing images using piecewise clothoid curves. *J. Vis. Lang. Comput. 23*, 6 (2012), 327–339. 3

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing* (2007), pp. 109–116. 3

[SCOL*04a] SORKINE O., COHEN-OR D., LIPMAN Y., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Symposium on Geometry Processing* (2004), vol. 71, pp. 175–184. 3

[SCOL*04b] SORKINE O., COHEN-OR D., LIPMAN Y., ALEXA M., RÖSSL C., SEIDEL H.-P.: Laplacian surface editing. In *Proceedings of the EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing* (2004), pp. 179–188. 7

[SDC09] SÝKORA D., DINGLIANA J., COLLINS S.: As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering* (2009), pp. 25–33. 3

[SF98] SINGH K., FIUME E.: Wires: A geometric deformation technique. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (1998), pp. 405–414. 2

[Sil85] SILVERMAN B.: Some aspects of the spline smoothing approach to nonparametric regression curve fitting (with discussion). *Journal of the Royal Statistical Society, Ser.B 47* (1985), 1–52. 4

[SP86] SEDERBERG T. W., PARRY S. R.: Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph. 20*, 4 (1986), 151–160. 2

[SSP07] SUMNER R. W., SCHMID J., PAULY M.: Embedded deformation for shape manipulation. *ACM Trans. Graph. 26*, 3 (2007). 3

[TZCO09] TAGLIASACCHI A., ZHANG H., COHEN-OR D.: Curve skeleton extraction from incomplete point cloud. In *ACM SIGGRAPH 2009 Papers* (2009), pp. 71:1–71:9. 2

[Ull76] ULLMAN S.: Filling the gaps: The shape of subjective contours and a model for their generation. *Biological Cybernetics* (1976). 2

[Wer38] WERTHEIMER M.: *Laws of organization in perceptual forms*. 1938. 2

[WJ97] WILLIAMS L. R., JACOBS D. W.: Stochastic completion fields: A neural model of illusory contour shape and salience. *Neural Comput. 9*, 4 (1997), 837–858. 2

[ZNA07] ZIMMERMANN J., NEALEN A., ALEXA M.: Silsketch: Automated sketch-based editing of surface meshes. In *Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling* (2007), pp. 23–30. 2, 7