Spline Interface for Intuitive Skinning Weight Editing

SEUNGBAE BANG and SUNG-HEE LEE, Korea Advanced Institute of Science and Technology, South Korea

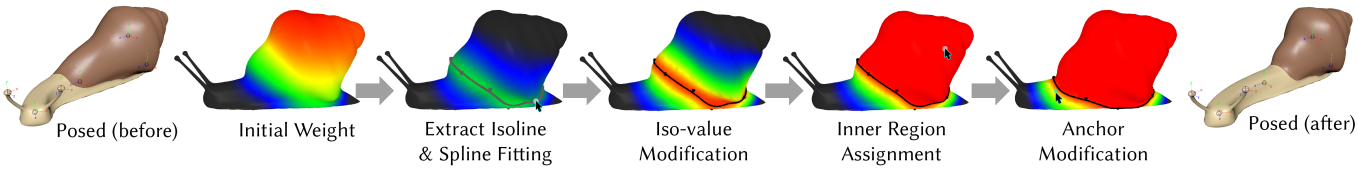


Fig. 1. Left two: initial deformation and its skinning weight computed by an automatic algorithm. Our spline interface allows for convenient and intuitive editing of skinning weights by using a small number of splines defined on a mesh surface.

Despite the recent advances in automatic methods for computing skinning weights, manual intervention is still indispensable to produce high quality character deformation. However, current modeling software does not provide efficient tools for the manual definition of skinning weights. Widely used paint-based interfaces gives users high degrees of freedom, but at the expense of significant efforts and time. This paper presents a novel interface for editing skinning weights based on splines, which represent the isolines of skinning weights on a mesh. When a user drags a small number of spline anchor points, our method updates the shape of the isolines and smoothly interpolates or propagates the weights while respecting the given iso-value on the spline. We introduce several techniques to enable the interface to run in real-time, and propose a particular combination of functions that generates appropriate skinning weight over the surface. Users can create skinning weights from scratch by using our method. In addition, we present the spline and the gradient fitting methods that closely approximate initial given weights, so that a user can modify the weights with our spline interface. We show the effectiveness of our spline-based interface through a number of test cases.

CCS Concepts: • **Computing methodologies** → **Animation**; *Mesh geometry models*;

Additional Key Words and Phrases: Character Modeling, Rigging, Skinning, Spline

ACM Reference Format:
Seungbae Bang and Sung-Hee Lee. 2018. Spline Interface for Intuitive Skinning Weight Editing. *ACM Trans. Graph.* 1, 1, Article 1 (January 2018), 14 pages. https://doi.org/10.1145/3186565

1 INTRODUCTION

Among many approaches for object and character deformation, closed-form skinning methods, such as Linear Blend Skinning (LBS) and Dual Quaternion Skinning (DQS), are widely used as they are fast and intuitive. The quality of these skinning methods highly depends on specifying appropriate skinning weights to vertices, which requires the intensive efforts of professional artists in production animation.

Many studies have proposed automatic skinning algorithms (e.g., [Baran and Popović 2007; Dionne and de Lasa 2013; Jacobson et al. 2011]) and they have been useful for generating initial weights. However, since most of these algorithms only deal with geometric properties of the rest mesh, e.g., geometric deformation energy, they cannot reflect other important factors related to skinning, such as physical properties that might have different values over a mesh and cause non-uniform mesh deformation. Even if there were an advanced automatic method that takes these factors into account, it is unlikely that automatic algorithms can exactly satisfy user needs. Therefore, manual intervention is almost always necessary for high quality skinning.

Software such as Maya and 3DS Max provides two kinds of methods for modifying skinning weights. One method uses modifiable primitive shapes, such as ellipsoids, that define the influence range of joints. This method is effective as long as the desired configuration of skinning weights is similar to the primitive shapes, which is not always the case. In general, this interface lacks fine control of skinning weights. The other standard method uses a paint-based interface, by which a user manually draws skinning weights on a mesh surface. This tool provides high degree of freedom in specifying the skinning weights, but is not convenient. Users need to paint repeatedly with many strokes to reach a satisfying result while frequently changing many parameters, such as brush size and smoothness. Despite the advances in skinning technology, the computer graphics community still lacks in a method that allows for detailed control of skinning weights with ease.

This paper introduces a novel interface for editing skinning weights by using splines. In our interface, a spline curve defines an isoline of skinning weights. When a user manipulates the spline curves on a mesh surface, which changes the desired shape of the isolines, our method determines the skinning weights on the entire mesh such that the desired isolines are satisfied while the skinning weights between the isolines are smoothly interpolated or propagated. A spline-based skinning interface can inherit the strong advantages of splines, e.g., controllable smoothness and level of detail. One can also change the overall skinning weights without

Authors' address: Seungbae Bang; Sung-Hee Lee, Korea Advanced Institute of Science and Technology, 291 Daehak-ro Yuseong-gu, Daejeon, 34141, South Korea, be2848@kaist.ac.kr, sunghee.lee@kaist.ac.kr.

0730-0301/2018/1-ART1 $$15.00
https://doi.org/10.1145/3186565

losing smoothness by only dragging a few anchor points. Creating a more complex weight configuration is possible by assigning more anchor points. Specifically, we use the method of [Panozzo et al. 2013] that computes spline curves on surface meshes in real-time.

Figure 1 shows an example of how splines are defined on a mesh surface and modified to edit the overall skinning weight of the surface. A basic workflow of our system is shown in Fig. 2. A user can generate splines from scratch by using a GUI or extract them from given initial skinning weight. As the splines are edits, skinning weight is updated to satisfy the spline constraints.

Fig. 2. Workflow of our system.

The contributions of this paper are as follows:

- We present a first spline-based interface for defining skinning weights and show the effectiveness of the interface.
- We introduce techniques that enable the spline-based skinning interface to run in real-time. These include a set of methods that efficiently compute skinning weights while respecting user-specified initial weights and an efficient spline fitting algorithm.
- Among several candidate smoothing functions, we propose a particular combination of functions that generates appropriate skinning weight distribution over the surface given splines.

The remainder of this paper proceeds as follows. After reviewing related work in Sec. 2 and the basic theories of splines in Sec. 3, we introduce a set of methods to compute skinning weights in Sec. 4. Then, Sec. 5 explains the spline editing and fitting procedures. Section 6 describes our experiments on skinning weight generation, and Sec. 7 introduces further applications that our spline interface can be used for besides the skinning weight editing. Finally, Sec. 8 concludes the paper.

2 RELATED WORK

Compared with intensive efforts for developing automatic algorithms that generate skinning weights, studies on efficient interface for manual weight design have been more or less ignored in computer graphics research. In this section, we first review important previous work on the automatic computation of skinning weights. Spline curves that can control the skinning weights must be defined on a surface domain. Thus, we subsequently review several methods for modeling splines on surface meshes.

Automatic Generation of Skinning Weights. After the introduction of closed-form skinning methods, such as LBS and DQS [Kavan et al. 2008], researchers have developed methods that automatically generate appropriate skinning weights for given meshes. One of the pioneering work is [Baran and Popović 2007] that introduced heat diffusion weight, see Sec. 4.3 for its details. Wareham and Lasenby [2008] developed a modified heat diffusion weight by introducing the concept of light illumination to skinning weights, and Bang et al. [2015] added a bi-Laplacian term to the heat diffusion weight to control the smoothness of the weight. Borosan et al. [2012] introduced a local computation method for the heat diffusion weight. The mean value coordinates [Floater 2003; Ju et al. 2005], harmonic coordinates [Joshi et al. 2007], and green coordinates [Lipman et al. 2008] were introduced for cage-based deformation. The Bounded Biharmonic Weight (BBW) method [Jacobson et al. 2011] achieved improved smoothness and local controllability with non-linear optimization of the Laplacian energy with constraints. An additional advantage of the method is that it can be used with arbitrary rig types, including bones, points, and cages. Dionne and de Lasa [2013] introduced the Geodesic Voxel Binding (GVB) method that is capable of computing skinning weights for a geometric model with multiple non-manifold meshes. Kavan and Sorkine [2012] proposed a method for optimized skinning weights that approximate the skin deformations produced by nonlinear variational deformation methods [McAdams et al. 2011]. Wang et al. [2015] introduced skinning weights that satisfy the linear deformation subspace. The weight reduction technique [Landreneau and Schaefer 2010] can be applied to existing skinning weights to reduce the number of influencing controllers per vertex.

If multiple example poses or shapes are available, for example, through shape capturing, inverse methods [Le and Deng 2012, 2014] can be applied to find optimal skinning weights that reproduce the captured shapes. Statistical shape models such as [Anguelov et al. 2005; Pons-Moll et al. 2015] typically learn identity-dependent and pose-dependent components that can create realistic deformations without resorting to skinning. A notable difference is [Loper et al. 2015], which learns identity and pose-dependent skinning weights for LBS. Mohr et al. [2003] noted the tedious process of paint interface and proposed a method to compute skinning weight from example deformations created manually by users. For facial animation, where blendshapes are used for creating mesh deformation, Lewis and Anjyo [2010] introduced an inverse problem of solving blendshape weights with direct manipulation of geometry.

All these automatic methods for generating skinning weights can be used to provide an initial solution, upon which our spline-based skinning interface can be applied to reflect a user's particular need.

Spline Curves on Surface Mesh. A spline curve on general surfaces has been realized by either using a variational approach [Hofer and Pottmann 2004] or as an iso-contour of a scalar field [Jin et al. 2009]. Both approaches require solving a relatively costly optimization. Thus, they are not suitable for interactive applications such as ours. Feng and Warren [2012] developed biharmonic B-splines with irregular knots by constructing a discrete bi-Laplacian, and Hou et al. [2016] improved the method such that biharmonic B-splines can be defined on arbitrary compact 2-manifolds. Their work provides a theoretically robust generalization of univariate B-splines to curved surfaces, but the computational load is too heavy to be used for our purposes.

Some fast algorithms to generate splines on surfaces take an approach that first computes an initial position by using weighted averaging in Euclidean space (thus, the position is not constrained to a surface), and then projects the position onto the surface. Wallner and Pottmann [2006] simply used 3D Euclidean space for this purpose. Panozzo et al. [2013] developed a high dimensional Euclidean embedding technique to approximate weighted averages

N_v	number of vertices
N_s	number of spline points
N_a	number of anchor points
N_o	number of isoline points
$\mathbf{w}^s$	Skinning weight
$\mathbf{w}^g$	Gradient weight
$\mathbf{K}$	Anchor weight
$\mathbf{G}$	Gradient anchor weight
$\mathbf{C}^p$	position of anchor points
$\mathbf{c}^i$	iso-value of anchor points
$\mathbf{c}^g$	gradient coefficient value of anchor points
$\mathbf{S}^p$	position of spline points
$\mathbf{s}^i$	iso-value of spline points
$\mathbf{O}^p$	position of isoline points
$\mathbf{o}^i$	iso-value of isoline points

Table 1. Frequently used symbols

of points on surfaces. Their method creates smooth spline curves with arbitrary anchor points on a surface mesh and performs fast enough to be used for real-time applications. We used their method to define splines in our work.

3 PRELIMINARY: SPLINE MODEL

3.1 B-spline

We use the standard cubic B-spline model in our method. The B-spline curve is widely used in many applications because of its smoothness and local controllability. By moving an anchor point, we can selectively modify the B-spline curve without losing its geometric continuity. In Euclidean space, a B-spline curve is defined as a weighted sum of B-spline basis functions $R_{i,r}(u)$, which are C^{r-2}-continuous $(r-1)$-th order piecewise polynomials ($r = 4$ in our experiment). Given a set of anchor points $\mathbf{c}_i^p$, a B-spline curve $s(u)$ is given by:

$$s(u) = \sum_{i=0}^{n-1} c_i^p R_{i,r}(u). \tag{1}$$

With N_a number of anchor points, basis functions $R_{i,r}(u)$ are defined on a knot vector $\mathbf{t} = t_0, t_1, ..., t_{N_a+r-1}, t_{N_a+r}$.

For a set of sample parameter values $U = \{u_0, \cdots, u_{N_s-1}\}$, we can represent the positions of the corresponding points on the spline $\mathbf{S}^p (\in \mathbb{R}^{N_s \times 3})$, dubbed *spline points* in this paper, in a matrix form:

$$\mathbf{S}^p = \mathbf{B}\mathbf{C}^p, \tag{2}$$

where $\mathbf{C}^p (\in \mathbb{R}^{N_a \times 3})$ is the position of anchor points and $\mathbf{B} (\in \mathbb{R}^{N_s \times N_a})$ is the basis matrix that corresponds to U. Table 1 summarizes the symbols used in the paper.

Open and Closed Loops. We use splines as either a closed loop curve or a clamped open curve, and both types require constraining the knots.

For a closed loop curve, the loop closure is ensured by duplicating the first three points and adding them as virtual anchor points. However, this trick breaks the form of (2). Instead, we achieve the loop closure by modifying the basis function $\mathbf{B}$ by adding the first three columns value to the last three columns: $\forall k : \mathbf{B}_{k,j} = \mathbf{B}_{k,j} + \mathbf{B}_{k,N_a-j-1}$ (for $j = 0, 1, 2$).

For an open curve, we want our open spline curve to be clamped such that the curve's end points match their first and last anchors because they give the user a more intuitive control. This is achieved by defining knot vector such that the first and last knots are repeated with multiplicity equal to the order r as follows: $u_0 = ... = u_{r-1}, u_{N_a+1} = ... = u_{N_a+r}$.

In both case, only the basis matrix $\mathbf{B}$ in (2) is changed. With the anchor points being fixed, the topology of the spline can be easily changed by replacing the basis matrix. Figure 3 shows splines generated with closed and open conditions. More details on the B-spline can be found in [De Boor 1978].

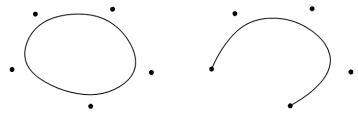


Fig. 3. Spline generated with closed and open with same anchor points.

3.2 Spline on Surface

In [Panozzo et al. 2013], an Euclidean-embedding metric is defined by an embedding $\mathbf{e} : \mathcal{M} \to \mathbb{R}^D$ where $\mathcal{M}$ is a metric space and D is set to 8. The distance between two points on the surface is calculated as the Euclidean distance between their embeddings $d(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{e}(\mathbf{x}_1) - \mathbf{e}(\mathbf{x}_2)\|$. By using the Fréchet mean with a number of anchors $\mathbf{x}_i$ and weights ξ_i, the weighted average on surface is defined as:

$$\hat{\mathbf{x}} = \underset{\mathbf{x}\in\mathcal{M}}{\operatorname{argmin}} \sum_i \xi_i d(\mathbf{x}, \mathbf{x}_i)^2. \tag{3}$$

Using (3), one can define a spline curve on the surface as a function of a parameter $u \in [0, 1]$:

$$\hat{\mathbf{x}}(u) = \underset{\mathbf{x}\in\mathcal{M}}{\operatorname{argmin}} \sum_{i=0}^{N_a-1} B_i(u) d(\mathbf{x}, \mathbf{c}_i^p)^2 \tag{4}$$

After solving (4), the original method [Panozzo et al. 2013] uses the Phong projection technique to project the $\hat{\mathbf{x}}$ to surface. To accelerate this process, a local search method is used that combines a greedy search for the locally nearest vertex and a breadth-first search for a target triangle to which $\hat{\mathbf{x}}$ is projected. We found that sometimes this searching scheme fails and creates unnaturally sharp protrusions in our spline, a possible problem mentioned in their paper [2013], and hence developed a more robust local search scheme customized for our purpose. We first find a Phong projection with a global search for the first spline generation. Subsequently, when an existing spline is edited, we examine the faces that have the current isoline as well as their adjacent faces (in three rings on each side of the isoline) to find the closest face for the Phong projection.

As a result, we obtain spline points on the surface. The position of a spline point will be represented in terms of the barycentric coordinates of the face the point belongs to.

Sampling Spline Points. We visualize the spline as connected line segments made by sample points $\mathbf{S}^p = \{\mathbf{s}_0, \mathbf{s}_1, ., , .\mathbf{s}_{N_s-1}\}$ of the spline, where $\mathbf{s}_k^p = \mathbf{x}(u_k)$ is a sample point for a parameter value of u_k. In order to obtain more or less uniformly sampled points on the spline, the number of sampling parameters u_k between two anchor points $\mathbf{c}_k^p$ and $\mathbf{c}_{k+1}^p$ is set to be proportional to $d(\mathbf{c}_k^p, \mathbf{c}_{k+1}^p)$. In Sec.

4.1, we will use these spline points to set constraints on weight computation.

4 WEIGHT COMPUTATION

Given a set of splines, our proposed technique computes a smooth scalar field that interpolates or approximates the spline constraint. For this, we introduce two methods for computing skinning weights: an interpolation-based method and a diffusion-based method. If one iso-value spline is specified, the skinning weight is computed by the diffusion-based method. On the other hand, if there is more than one iso-value spline, the hybrid of both methods is used. We also investigate local computation that reduces the region of non-zero weight and accelerates the computation. A method to decompose the influence of anchor points, dubbed *anchor weight*, on a vertex' skinning weight is introduced for faster weight editing. For a more sophisticated control of weight, the gradient control technique is introduced to add gradient weight to the skinning weight. Final weight is computed as sum of skinning weight of gradient weight. The overall framework of the weight computation is illustrated in Fig. 4.

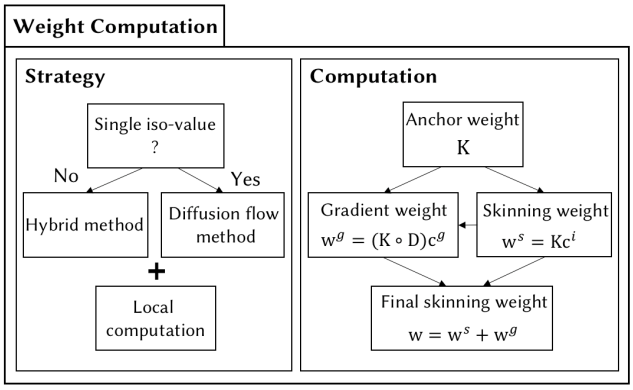


Fig. 4. Overview of weight computation.

Figure 5 shows the color codes for the anchor weight (a) and skinning weight (b).

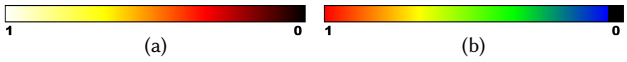


Fig. 5. Color codes for anchor weight (a) and for skinning weight (b).

4.1 Spline Constraint

Our primary condition to be satisfied is the spline constraint, i.e., the skinning weights of the vertices must be determined to agree with the given skinning weight of the spline. We use barycentric coordinates to interpolate the vertex skinning weights for each mesh face. That is, the weight $w(s_k^p)$ of a spline point s_k^p is expressed as:

$$w(s_k^p) = b_{k_0} w_{k_0} + b_{k_1} w_{k_1} + b_{k_2} w_{k_2} = s_k^i, \tag{5}$$

where w_{k_j} denotes the weight of a vertex k_j of a face (for $j = 0, 1, 2$) to which the spline point s_k^p belongs, and b_{k_j} are the barycentric coordinates of s_k^p with respect to the vertex. Therefore, the vertex weights w_{k_j} must be determined to satisfy $w(s_k^p)$.[1] Stacking the barycentric equation (5) for a set of desired weights of spline points into a matrix constitutes a linear equality constraint equation:

$$\mathbf{A}\mathbf{x} = \mathbf{s}^i, \quad \text{where} \quad \mathbf{s}^i = \mathbf{B}\mathbf{c}^i \tag{6}$$

The matrix $\mathbf{A}(\in \mathbb{R}^{N_s \times N_v})$ is a matrix with $A_{k,k_j} = b_{k_j}$ that stacks the barycentric conditions (5) in each row, $\mathbf{x}$ is the skinning weights of all vertices, and $\mathbf{s}^i$, $\mathbf{c}^i$ is a vector of the iso-value of the spline points and anchor points respectively.

The skinning weight $\mathbf{w}^s$ can be computed by the minimizing the energy $E(\mathbf{x})$ that satisfies the constraints of the linear equality equation in the form:

$$\mathbf{w}^s = \underset{\mathbf{x}}{\operatorname{argmin}}\, E(\mathbf{x}) \quad \text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{s}^i \tag{7}$$

The decision of energy function $E(\mathbf{x})$ determines the overall shape of the skinning weight. We first introduce two types of methods that satisfy the spline constraint: the interpolation-based method and diffusion flow method. We compute the skinning weight using the diffusion flow method if there is only one iso-value spline and use both methods together if there is more than one spline.

4.2 Interpolation-based Method

The technique of interpolating the scalar field on a surface given point constraints has been introduced for generalized barycentric coordinates [Floater 2003; Ju et al. 2005; Lipman et al. 2007; Rustamov 2010]. For the specific application for the skinning weight of articulated characters, methods based on harmonic functions [Joshi et al. 2007] and bi-harmonic functions [Jacobson et al. 2011] have been developed. We can similarly formulate our problem within a surface with the linear equality constraint of spline. Specifically, we minimize the quadratic energy under the spline constraint:

$$\mathbf{w}^s = \underset{\mathbf{x}}{\operatorname{argmin}}\, \mathbf{x}^T\mathbf{Q}\mathbf{x} \quad \text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{s}^i, \tag{8}$$

where $\mathbf{Q}$ $(\in \mathbb{R}^{N_v \times N_v})$ is a positive semi-definite matrix of quadratic coefficients. We can solve this equation by using Lagrange multipliers with a hard constraint problem of the form: $\begin{pmatrix} \mathbf{Q} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{pmatrix}\begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{s}^i \end{pmatrix}$. However, with this hard constraint, the positive semi-definiteness is broken, and thus a fast Cholesky solver cannot be used. Alternatively, we enforce the linear equality constraint weakly by appending a quadratic penalty term to the original energy. This results in a new quadratic energy of the form:

$$E(\mathbf{x}) = \alpha\mathbf{x}^T\mathbf{Q}\mathbf{x} + (\mathbf{A}\mathbf{x} - \mathbf{s}^i)^T\mathbf{U}(\mathbf{A}\mathbf{x} - \mathbf{s}^i) \tag{9}$$

where the scalar coefficient α controls the importance between the two terms and the matrix $\mathbf{U}$ $(\in \mathbb{R}^{N_s \times N_s})$ determines the weights among the linear equality constraints. We set $\mathbf{U}$ to the identity matrix to give equal weights to every spline constraint. The minimizer $\mathbf{w}^s$ of the energy can be obtained from $\frac{dE(\mathbf{x})}{d\mathbf{x}} = 0$, i.e.,

$$(\alpha\mathbf{Q} + \mathbf{A}^T\mathbf{A})\mathbf{w}^s = \mathbf{A}^T\mathbf{s}^i \tag{10}$$

[1] The spline points are obtained by (4). However, if more than one spline point of the iso-value are given in one face, no vertex weights can satisfy the constraints in general. Therefore, we simply select only one spline point at each face and discard the rest, if any, on that face. The remaining spline points constitute S^p.

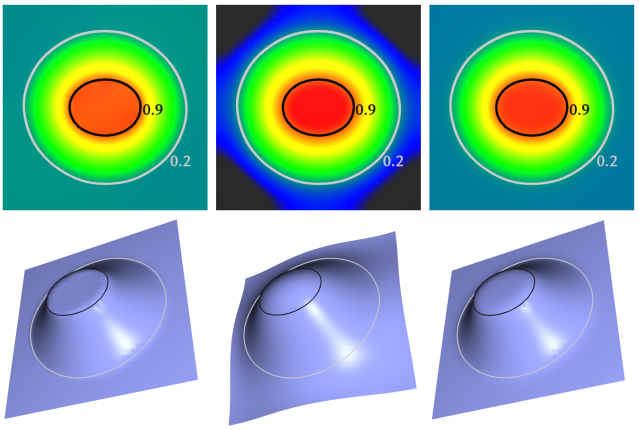


Fig. 6. Skinning weight generated using interpolation-based method with Laplacian (left), bi-Laplacian (middle), and sum of Laplacian and bi-Laplacian (right) operators, with two iso-values of $[0.2, 0.9]$. In the second row, the weight is multiplied to the normal direction of the vertices to deform the mesh to visualize the weight effect. Anchor points are omitted from visualization.

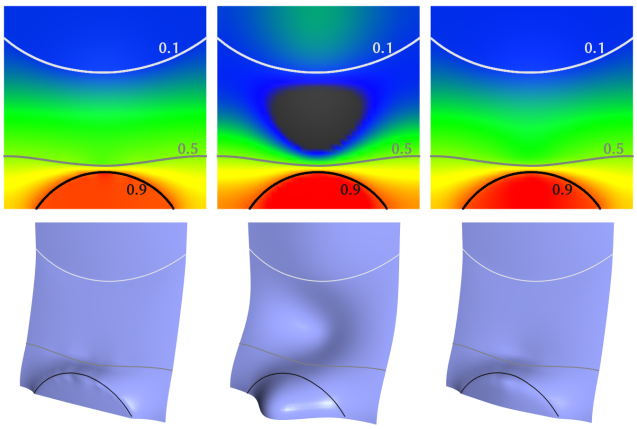


Fig. 7. Skinning weight generated using interpolation-based method with Laplacian (left), bi-Laplacian (middle), and sum of Laplacian and bi-Laplacian (right) operators, with iso-values of $[0.1, 0.5, 0.9]$.

By setting a proper coefficient of α (= 0.1 in our experiment), (10) generates results similar to the solution with the hard constraint.

Choice of Operator. We now review the choice of energy for the interpolation-based method. We first consider a common energy in computer graphics: the Dirichlet energy, which is a quadratic functional that measures how much a function $\mathbf{x}$ changes over domain: $E(\mathbf{x}) = -\mathbf{x}^T\mathbf{L}_s\mathbf{x}$, where $\mathbf{L}_s$ is a symmetric matrix containing the edge weights. The corresponding Euler-Lagrange equation is the Laplace equation: $\mathbf{L}_s\mathbf{x} = 0$. Another popular choice is the squared Laplacian energy, which measures the magnitude of curvature [Jacobson 2013]: $E(\mathbf{x}) = \mathbf{x}^T\mathbf{L}_s\mathbf{M}^{-1}\mathbf{L}_s\mathbf{x}$, where $\mathbf{M}$ is a diagonal mass matrix. The corresponding Euler-Lagrange equation is the bi-Laplace equation: $\mathbf{L}_s\mathbf{M}^{-1}\mathbf{L}_s\mathbf{x} = 0$. Thus, we review the choice for the matrix $\mathbf{Q}$ with the Laplacian operator $\mathbf{Q} = -\mathbf{L}_s$ and the bi-Laplacian operator $\mathbf{Q} = \mathbf{L}_s\mathbf{M}^{-1}\mathbf{L}_s$.

Figure 6 shows different skinning weight distributions solved with Laplacian and bi-Laplacian operators when two splines of iso-values $[0.2, 0.9]$ are given. As can be seen in the figure, the in-between regions of two splines show a similar skinning weight distribution while the bi-Laplacian results in a slightly smoother result. Differences are notable on the boundaries of the splines and beyond as the bi-Laplacian creates a much smoother result and is also capable of extrapolation.

With this observation, it may seem that the bi-Laplacian is a natural choice to be the operator. However, the extrapolating nature of the bi-Laplacian often generates unintuitive results. Figure 7 shows the skinning weights with three splines of iso-values $[0.1, 0.5, 0.9]$. The steep decrease of weights from 0.9 to 0.5 makes the bi-Laplacian operator create a deep downfall below 0.1 in the region $[0.1, 0.5]$, which is counter-intuitive for the skinning as one would expect the weight distribution in that region to be between 0.1 and 0.5. This is due to the nature of the C^1-continuity of the bi-Laplacian, which propagates the slopes outside the boundaries. For a region bounded by two iso-value splines, it is more intuitive to have the skinning weights interpolating between the two iso-values, which is achieved by the Laplacian operator.

We have observed that both the Laplacian and bi-Laplacian have advantages and disadvantages. Aiming to take the benefits of both operators, we propose to use the combination of two operators $-a\mathbf{L}_s + (1-a)\mathbf{L}_s\mathbf{M}^{-1}\mathbf{L}_s$. The last columns of Figs. 6 and 7 show the results for $-0.5\mathbf{L}_s + 0.5\mathbf{L}_s\mathbf{M}^{-1}\mathbf{L}_s$. One can see that the combination inherits the interpolating nature of the Laplacian, with guaranteed smoothness across the splines.

From the viewpoint of skinning weight distribution, the Laplacian-based interpolation has another problematic phenomenon. While it is natural that the weight decreases to zero as the vertex gets farther from the lowest iso-value spline, the actual results in Figs. 6 and 7 show that the weights of the far region still remains similar to the minimum iso-value. A solution to this would be to add another spline with zero iso-value to where the weights should be zero, but this is somewhat cumbersome. Instead, this problem is addressed in Sec. 4.4 with the diffusion-based method that is described next.

4.3 Diffusion Flow Method

Diffusion flow is a mathematically well-understood model for the time-dependent process of smoothing a given signal value [Botsch et al. 2010]. Diffusion flow with the implicit Euler integration of matrix is formulated as $(\mathbf{I} - h\Delta)f(t+h) = f(t)$, where h denotes a time period for the smoothing process. As h increases, the signal value $f(t+h)$ will be propagated more smoothly. The computation of the skinning weight by using the diffusion-based method is introduced by [Baran and Popović 2007] with the famous heat equation: $(\mathbf{H} - \mathbf{L})\mathbf{w} = \mathbf{H}\mathbf{p}$, where $\mathbf{L} = \mathbf{M}^{-1}\mathbf{L}_s$ is a discrete approximation of the Laplacian at each vertex. In their work, the initial values are assigned to the $\mathbf{p}$ vector and different time parameters are given to each vertex by using the diagonal matrix $\mathbf{H}$. In our case, $\mathbf{H}$ can be simplified to the identity matrix. By generalizing the Laplacian $\mathbf{L}_s$ with $-\mathbf{Q}$ and by multiplying the mass matrix $\mathbf{M}$ to the equation, the

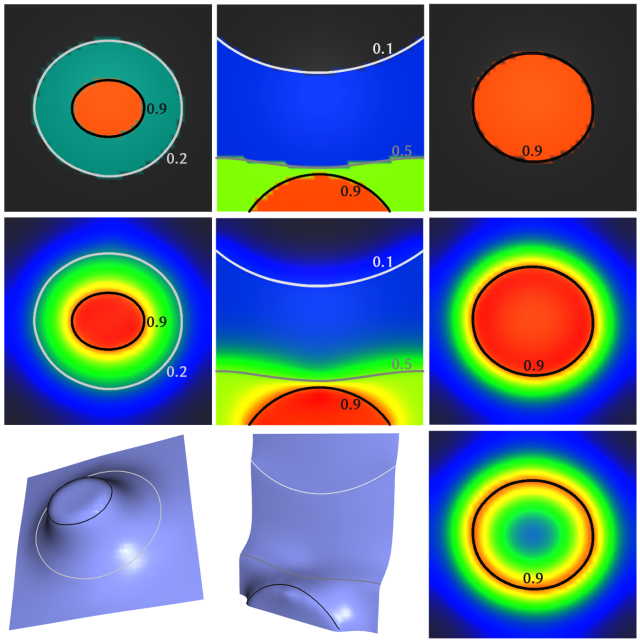


Fig. 8. Top: Input splines with iso-values marked in the figure and the visualization of the initial value $\mathbf{p}$. Middle: Resulting skinning weight generated with the diffusion-based method using sum of Laplacian and bi-Laplacian. Bottom: Left two images show corresponding deformation applied on normal direction, and right image shows the skinning weight solved without assigning vector $\mathbf{p}$.

diffusion flow equation can be written as:

$$(\beta\mathbf{M} + \mathbf{Q})\mathbf{w}^s = \beta\mathbf{M}\mathbf{p}, \tag{11}$$

where β (= 0.1 in our experiment) is a time parameter for diffusion. Formulating a quadratic optimization problem to satisfy the above equation with the spline constraint leads to:

$$\mathbf{w}^s = \underset{\mathbf{x}}{\operatorname{argmin}}\, \mathbf{x}^T(\beta\mathbf{M} + \mathbf{Q})\mathbf{x} - 2\mathbf{x}^T\beta\mathbf{M}\mathbf{p} \quad \text{s.t.} \quad \mathbf{A}\mathbf{x} = \mathbf{s}^i. \tag{12}$$

Similarly to the interpolation-based method, we treat the constraint as a soft constraint and obtain the solution as:

$$(\alpha\beta\mathbf{M} + \alpha\mathbf{Q} + \mathbf{A}^T\mathbf{A})\mathbf{w}^s = \mathbf{A}^T\mathbf{s}^i + \alpha\beta\mathbf{M}\mathbf{p}. \tag{13}$$

Figure 8 shows the results for different spline configurations. We set the initial value of vector $\mathbf{p}$ such that $p_j = s^i_k$ if a vertex j is located inside the spline of s^i_k, and 0 otherwise. The method to find the surrounding spline will be explained later in this section. The $\mathbf{Q}$ matrix is set to $-0.5\mathbf{L}_s + 0.5\mathbf{L}_s\mathbf{M}^{-1}\mathbf{L}_s$ to make it consistent with the interpolation-based method. One can see that the diffusion flow method smoothly attenuates the skinning weights to the outer region without requiring the 0 valued isoline, which is a desired property as a skinning weight generator. A notable limitation of the diffusion flow method is that, as can be seen on the bottom row of Fig. 8, it does not smoothly interpolate the weights between the two iso-values, which give a staircase-like weight distribution

The above observations suggest that the diffusion-based method is a good choice for the case where the skinning weights can be simply controlled by a single iso-value spline. In such a case, the diffusion-based method exhibits natural weight attenuation properties while avoiding spline interpolation.

Diffusion Flow for a Single Spline. In order to compute (13), the vector $\mathbf{p}$ must be initialized with proper values as in Fig. 8 (top right). If $\mathbf{p}$ is initialized with zero values and solved with one spline, the value on the spline will be propagated from the spline, as in Fig. 8 (bottom right). This is not a desired weight distribution for skinning. In general, if a spline creates a closed region, the user would want the region to share similar weights as the iso-value of the spline and the weights to smoothly propagate to the outside, as shown in Fig. 8 (middle right). For this, the system needs to know the region, and this is achieved by asking a user to select any vertex in the region. The region could be found by collecting reachable vertices in a flood-filling manner from the selected vertex. In our implementation, we applied the interpolation-based method to find the bound region. Specifically, we solve (8) with an additional constraint that the selected vertex has a very high weight (1000). This makes weights higher than the isovalue of the vertices inside the region, and we collect these vertices. Subsequently, we assign $\mathbf{p}$ with $p_j = s^i_k$ for the vertex j in the region and solve (13). The above identification step is performed only once when a spline is created. During the following spline editing process, the region is efficiently updated by collecting vertices with weights higher than s^i_k.

Another Derivation. We can also derive the equation for the diffusion flow method purely from the energy minimization viewpoint. In addition to the quadratic energy $\mathbf{x}^T\mathbf{Q}\mathbf{x}$ and the spline constraint $\mathbf{A}\mathbf{x} = \mathbf{s}^i$, we encourage the weight to be similar to the original value $\mathbf{x} = \mathbf{p}$. Putting these together leads to the energy function of the form:

$$E(\mathbf{x}) = \alpha\mathbf{x}^T\mathbf{Q}\mathbf{x} + (\mathbf{A}\mathbf{x} - \mathbf{s}^i)^T\mathbf{U}(\mathbf{A}\mathbf{x} - \mathbf{s}^i) + \alpha\beta(\mathbf{x} - \mathbf{p})^T\mathbf{M}(\mathbf{x} - \mathbf{p}) \tag{14}$$

The matrix $\mathbf{U}$ determines the importance among the spline constraints and is set to $\mathbf{I}$ as in the interpolation-based method. The mass matrix $\mathbf{M}$ is used to scale the importance of each vertex for $\mathbf{x} = \mathbf{p}$ with the area associated with the vertex. Minimization of (14) results in the same equation as (13).

4.4 Hybrid Method of Interpolation and Diffusion

We have discussed the advantages and disadvantages of the interpolation-based method and the diffusion-based method. To combine their strengths, we develop a hybrid method for a mesh with more than one spline. The idea is to divide the regions for the interpolation-based method and diffusion-based method.

Specifically, to take advantage of its good capability of interpolating iso-values of spline constraint, we set a interpolation-based computing region $\mathbf{V}_I (\in \mathbb{R}^{N_I \times 3})$ as the vertices that will be assigned with weights higher than the smallest iso-value among the

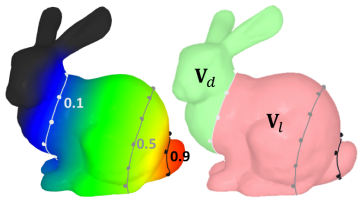


Fig. 9. Interpolation-based region $\mathbf{V}_I$ and diffusion-based region $\mathbf{V}_d$ divided (right) with given spline constraint (left)

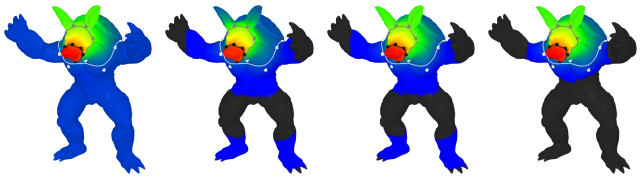


Fig. 10. Skinning weight generated with iso-values [0.1, 0.5, 0.9] using the interpolation-based method (a), the diffusion-based method (b), hybrid of both (c), and hybrid with local computation (d). For a clear distinction on zero-valued weight, the black color is used only on the vertices of zero weight for this example.

splines. Then we set the remaining region as a diffusion-based region $\mathbf{V}_d (\in \mathbb{R}^{N_d \times 3})$ to enjoy its good capability of diffusing iso-value to surrounding region. Figure 9 depicts the division into regions.

If we rearrange the equation for skinning weight computation (10) or (13) as a linear system $\mathbf{C}\mathbf{w}^s = \mathbf{d}$, and divide $\mathbf{w}^s$ into weights for the interpolation-based region $\mathbf{w}^s_l (\in \mathbb{R}^{N_l})$ and for the diffusion-based region $\mathbf{w}^s_d$, the linear system can be written as:

$$\begin{pmatrix} \mathbf{C}_{ll} & \mathbf{C}_{ld} \\ \mathbf{C}_{dl} & \mathbf{C}_{dd} \end{pmatrix} \begin{pmatrix} \mathbf{w}^s_l \\ \mathbf{w}^s_d \end{pmatrix} = \begin{pmatrix} \mathbf{d}_l \\ \mathbf{d}_d \end{pmatrix}, \tag{15}$$

where $\mathbf{C}_{ll}, \mathbf{C}_{ld}, \mathbf{C}_{dl}$, and $\mathbf{C}_{dd}$ are corresponding block submatrices of $\mathbf{C}$. This equation is solved sequentially in the following order. We first solve the interpolation region with a diffusion region specified as a boundary condition.

$$\mathbf{C}_{ll}\mathbf{w}^s_l = \mathbf{d}_l - \mathbf{C}_{ld}\tilde{\mathbf{w}}^s_d, \tag{16}$$

where $\tilde{\mathbf{w}}^s_d$ is the previous weight of the diffusion region before updating the anchor positions. Note that the difference between $\tilde{\mathbf{w}}^s_d$ and $\mathbf{w}^s_d$ is negligible as the weights are continuously updated while a user modifies the anchor position. To solve the interpolation region, we set $\mathbf{C} = \alpha\mathbf{Q} + \mathbf{A}^T\mathbf{A}$, and $\mathbf{d} = \mathbf{A}^T\mathbf{s}^i$. Then, we solve the diffusion region with the interpolation region as a boundary condition:

$$\mathbf{C}_{dd}\mathbf{w}^s_d = \mathbf{d}_d - \mathbf{C}_{dl}\mathbf{w}^s_l, \tag{17}$$

where $\mathbf{C} = \alpha\beta\mathbf{M} + \alpha\mathbf{Q} + \mathbf{A}^T\mathbf{A}$ and $\mathbf{d} = \mathbf{A}^T\mathbf{s}^i + \alpha\beta\mathbf{M}\mathbf{p}$. Note that we set $\mathbf{p} = 0_{N_v}$ because the diffusion region is only computed in the region outside of the minimum iso-value spline.

When the spline is initialized for the first time, no skinning weight is available for distinguishing the regions. Thus, we use the interpolation-based method with (10) once to identify the regions. The hybrid method is applied thereafter. In Fig. 10, we compare the results of the interpolation-based method (a), diffusion flow method (b), and hybrid method (c).

4.5 Local Computation

As can be observed from Fig. 10 (c), even the hybrid method may have unnecessarily large areas with small but non-zero weights. The non-zero weight areas may even spread to the unrelated areas (e.g. foot) because of the effect of the bi-Laplacian term.

Such areas do not exhibit perceivable deformation due to the corresponding controller and thus can be clamped to zero and excluded from weight computation for computational efficiency. To this end,

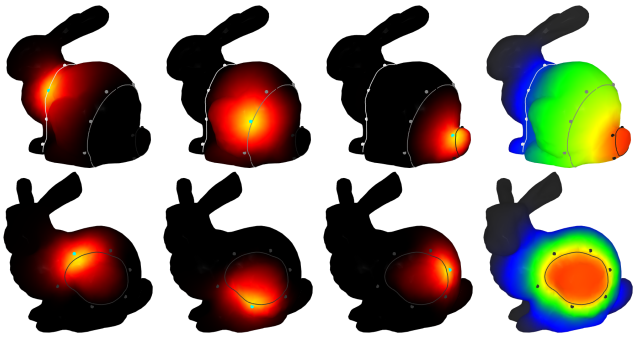


Fig. 11. Anchor weight and its applied skinning weight on a bunny model. Top: The results of the hybrid method of isolines [0.1, 0.5, 0.9]. Bottom: The results of the diffusion-based method with only one isoline [0.8].

we define a local computing region, as the vertices with the weight values above a threshold (0.05), with extra number (4) of rings of neighbor vertices added. The extra neighbors are added to respond to the possible increase in the local computing region as a user edits the splines. The local computing region consists of interpolation and diffusion regions. The remaining region is defined as the zero-clamp region $\mathbf{V}_z$, in which the vertices are forced to have zero weights.

This distinction allows for local computation to remove the zero-clamp region from solving for the weights. This local computation scheme contributes greatly to enabling real-time computation, which is essential for the plausible visualization skinning weights while a user manipulates the spline interface.

Introducing the zero-clamp region turns the system into:

$$\begin{pmatrix} \mathbf{C}_{ll} & \mathbf{C}_{ld} & \mathbf{C}_{lz} \\ \mathbf{C}_{dl} & \mathbf{C}_{dd} & \mathbf{C}_{dz} \\ \mathbf{C}_{zl} & \mathbf{C}_{zd} & \mathbf{C}_{zz} \end{pmatrix} \begin{pmatrix} \mathbf{w}^s_l \\ \mathbf{w}^s_d \\ \mathbf{w}^s_z \end{pmatrix} = \begin{pmatrix} \mathbf{d}_l \\ \mathbf{d}_d \\ \mathbf{d}_z \end{pmatrix} \tag{18}$$

where $\mathbf{w}^s_z$ denotes the weights for the zero-clamp region. Similarly to (15), the equation can be solved with the interpolation region first as:

$$\mathbf{C}_{ll}\mathbf{w}^s_l = \mathbf{d}_l - \mathbf{C}_{ld}\tilde{\mathbf{w}}^s_d - \mathbf{C}_{lz}\mathbf{w}^s_z \tag{19}$$

Then the diffusion region is solved with the interpolation region as the boundary condition:

$$\mathbf{C}_{dd}\mathbf{w}^s_d = \mathbf{d}_d - \mathbf{C}_{dl}\mathbf{w}^s_l - \mathbf{C}_{dz}\mathbf{w}^s_z \tag{20}$$

In the above equations, $\mathbf{w}^s_z$ is set to zero to enforce zero weight to the zero-clamp region.

The local computing scheme requires some time to identify the non-zero weight region, but the overall computation time is significantly reduced. In the initial stage, we solve (10) and identify the local computing region. Figure 10 (d) shows the result of applying the local computation.

4.6 Decomposition into Anchor Weights

Until now, we have computed the skinning weight given the iso-value constraints of spline $\mathbf{s}^i$. Since the spline is determined by a small number of anchor points, we can decompose the skinning weight according to the effect of the anchor points, which leads to increased computational efficiency as will be shown here.

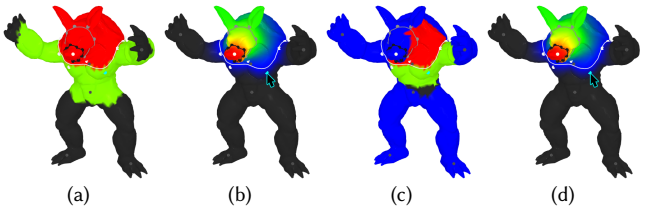


Fig. 12. From the left, local computation region identified with the skinning weight divided into interpolation region (red), diffusion region (green), and zero-clamp region (black) (a), and its skinning weight (Sec. 4.5) (b). Local computation region identified with the anchor weight divided into each region with boundary region (blue) added (c), and its skinning weight (Sec. 4.6) (d).

For the interpolation-based method, from (10) and $\mathbf{s}^i = \mathbf{B}\mathbf{c}^i$, we can relate $\mathbf{w}^s$ and $\mathbf{c}^i$ as

$$\mathbf{w}^s = \mathbf{K}\mathbf{c}^i, \tag{21}$$

$$(\alpha\mathbf{Q} + \mathbf{A}^T\mathbf{A})\mathbf{K} = \mathbf{A}^T\mathbf{B}, \tag{22}$$

where each column of the $\mathbf{K}$ ($\in \mathbb{R}^{N_v \times N_a}$) matrix represents the influence of the anchor point on the skinning weight of every vertex. Thus, the matrix $\mathbf{K}$ will be termed *anchor weight matrix*. Note that $\mathbf{K}$ remains constant if the anchor point positions do not change. Therefore, if only the iso-value of an anchor is changed, $\mathbf{w}^s$ can be computed simply by multiplying the pre-computed matrix $\mathbf{K}$ with the new $\mathbf{c}^i$ instead of solving the linear system (10). [2] If an anchor position is manipulated, the anchor weight matrix should be updated by solving (22).

It can be shown straightforwardly that (21) holds for the diffusion method if $\mathbf{K}$ is defined to satisfy:

$$(\alpha\beta\mathbf{M} + \alpha\mathbf{Q} + \mathbf{A}^T\mathbf{A})\mathbf{K} = \mathbf{A}^T\mathbf{B} + \alpha\beta\mathbf{M}\mathbf{P}, \tag{23}$$

where $\mathbf{P}(\in \mathbb{R}^{N_v \times N_a})$ is a matrix with every column being the $\mathbf{p}$ vector. Here, $p_j = 1/N_a$ for the diffusion flow method with a single spline, and $\mathbf{p} = 0_{N_v}$ for the hybrid method with multiple splines.

Figure 11 shows the anchor weights and the resulting skinning weights with the hybrid method (top) and diffusion-based method (bottom). Naturally, the anchor weight is the highest for the vertex closest to the anchor point and diminishes with the distance.

Local Computation for Anchor Weight. When a user moves the position of an anchor point, only the vertices with a non-zero anchor weight corresponding to the anchor point are affected. Therefore, we can further reduce the local computing region introduced in Sec. 4.5 by using the anchor weight. In our experiment, we set the local computing region as the vertices of which the absolute value of the selected anchor weight is above a threshold (0.05), with an extra number of rings (4) of neighbor vertices added. (Due to the bi-Laplacian operator, the vertices influenced by an anchor may have negative anchor weights.) We divide the local computing region into a interpolation region $\mathbf{V}_I$, a diffusion region $\mathbf{V}_d$, and a zero-clamp region $\mathbf{V}_z$ and set non-local computing region as a boundary region $\mathbf{V}_b$. We then compute the equation (18) with the boundary region $\mathbf{V}_b$ as boundary constraint. Figure 12 shows the local regions computed by the skinning weight and by the anchor weight, as well as and the resulting skinning weight distributions. Despite a much reduced local computing region that enables a shorter computing time, the method using anchor weights produces similar results as the one that uses the skinning weights.

[2] However, the anchor weight of the minimum iso-value must be computed again because the zero-clamp region needs to be updated according to the iso-value.

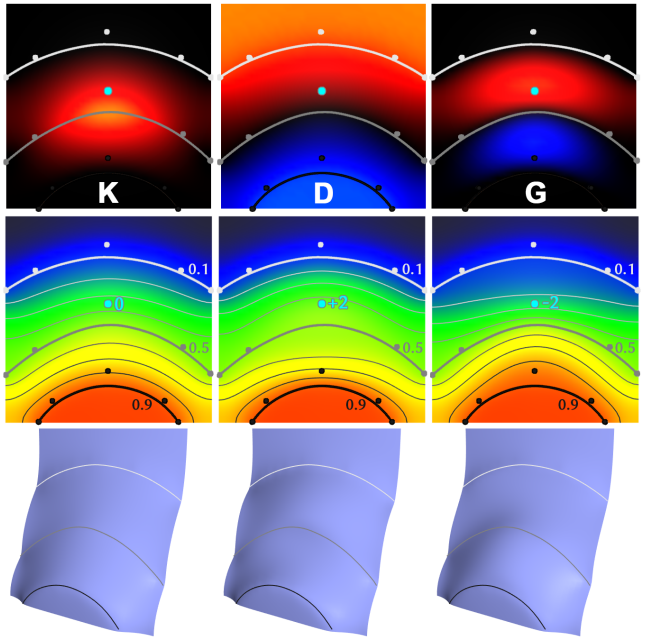


Fig. 13. Top: from the left, anchor weight matrix $\mathbf{K}$, difference matrix $\mathbf{D}$, and gradient weight matrix $\mathbf{G}$ for a selected anchor (cyan). For the difference and gradient weight matrices, negative values are visualized with blue color. For the gradient matrix, values are scaled 10 times for clear visualization. Middle: different skinning weights obtained using gradient coefficient set to 0 (left), 2 (middle), −2 (right) for the anchor colored cyan. Additional isolines are visualized to emphasize the change. Bottom: corresponding deformation applied on normal direction.

4.7 Gradient Control

The methods introduced so far allow for solving skinning weights that smoothly interpolate the spline constraints. In these methods, the interpolation characteristics are determined by the operators and their modification is not possible. However, fine control of the weight distribution is often necessary. For example, with two given splines, a user may want the weights to change only slightly near the splines but change sharply in the middle of the splines. This could be achieved by placing additional splines of new iso-values between the two splines, but this increases the number of user parameters considerably and is inconvenient. A better approach would be to allow users to control the slope of the weight distribution across the spline, which is similar to the tangent editing function of spline curve editing. To this end, we develop a method to control the difference between the computed skinning weight $\mathbf{w}^s$ and the iso-value $\mathbf{c}^i$ of the anchors. The method is achieved by the following

equations:

$$\begin{aligned} \mathbf{w}^g &= \mathbf{G}\mathbf{c}^g, \\ \mathbf{G} &= \mathbf{K} \circ \mathbf{D}, \\ \mathbf{D} &= 1_{N_v} \mathbf{c}^{i\,T} - \mathbf{W}^s, \end{aligned} \tag{24}$$

where $\mathbf{G}(\in \mathbb{R}^{N_v \times N_a})$ is a matrix where each column is a gradient weight vector for each anchor point (hence termed *gradient anchor weight matrix*) and $\mathbf{c}^g$ is the coefficient vector that manipulates the slope for each anchor. If $\mathbf{c}^g_k > 0$, the slope around the k-th anchor becomes flatter, and if $\mathbf{c}^g_k < 0$, the slope gets steeper. The matrix $\mathbf{D}(\in \mathbb{R}^{N_v \times N_a})$ encodes the difference between the iso-value $\mathbf{c}^i$ of the anchor and skinning weight $\mathbf{w}^s$ in each column, where 1_{N_v} is a vector filled with 1, and $\mathbf{W}^s(\in \mathbb{R}^{N_v \times N_a})$ is a matrix that juxtaposes the N_a copy of the weight vector $\mathbf{w}^s$. The operator $\circ$ performs component-wise multiplication. For a matrix $\mathbf{K}$, we use the anchor weight matrix (21), with slight modification that clamps negative and small positive values to zero. The final skinning weight is computed by adding a gradient skinning weight on the original skinning weight:

$$\mathbf{w} = \mathbf{w}^s + \mathbf{w}^g. \tag{25}$$

Figure 13 visualizes the value of each matrix $\mathbf{K}, \mathbf{D}, \mathbf{G}$, as well as the effect of gradient control on the skinning weight.

5 SPLINE EDITING AND FITTING

A user can dynamically build splines from scratch by picking anchor positions on the mesh through a GUI, as shown in the accompanying video. Splines can also be built by extracting isolines from the given skinning weight. Once built, our system supports anchor editing operations, such as repositioning, deletion, insertion, splitting, and merging. During the editing process, our system performs local computations for real-time user interactions.

5.1 Spline Fitting from Given Weight

If skinning weights are specified on mesh surface, generated by a paintbrush interface, an automatic skinning algorithm, or a spline interface, our system can extract isoline for a selected iso-value, and then fit a spline to the isoline. Finally anchor coefficient optimization are performed to minimize the difference between user given weights. A user can manipulate this newly generated spline to modify the existing skinning weight. The overview of spline fitting is illustrated in Fig. 14.

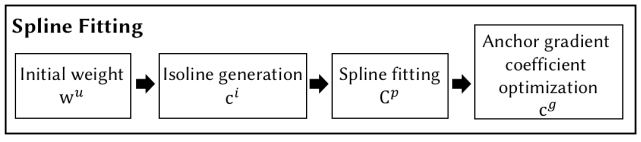


Fig. 14. Overview of spline fitting.

5.1.1 Isoline Generation. We will represent an isoline as a sequence of points on the edge of a mesh, i.e., $\mathbf{O}^p = \{\mathbf{o}_0, \mathbf{o}_1, ..., \mathbf{o}_{N_o-1}\}$. Let $\mathbf{o}^i$ denote a vector of iso-values for isolines. The position of isoline points can be identified by linear interpolation between the vertices of each edge $\mathbf{o}^p_k = l_{k_0}\mathbf{x}_{k_0} + (1 - l_{k_0})\mathbf{x}_{k_1}$, with the interpolation weight l_{k_j} $(j = 0, 1)$ satisfying $o^i_k = l_{k_0} w(\mathbf{x}_{k_0}) + (1 - l_{k_0}) w(\mathbf{x}_{k_1})$. We search through every edge to find isoline points and sort them by the adjacency.

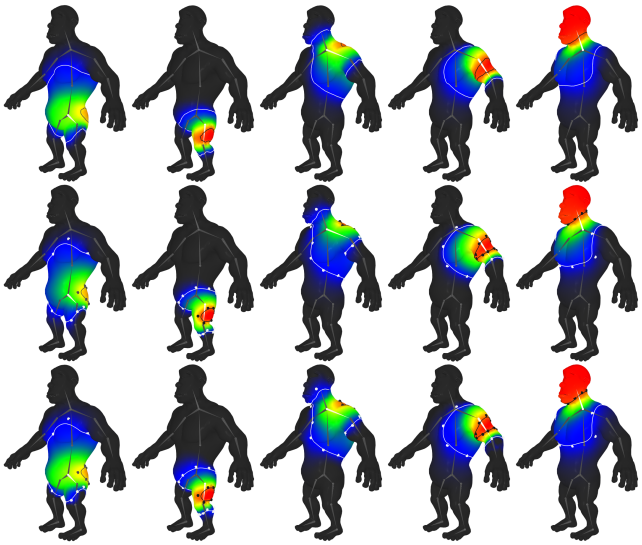


Fig. 15. Ogre model. Top: Automatically generated skinning weight using BBW and extracted isolines of iso-values $[0.1, 0.9]$. Middle: Spline fitted to isoline and computed skinning weight. Bottom: The skinning weight after gradient coefficient optimization.

5.1.2 Spline Fitting. The spline fitting method should ideally be able to determine the optimal number of anchor points and their positions, but it may require heavy computation for a real-time applications. Since real-time interactivity is critical in our method, we take a more simplified approach. Specifically, we set the number of anchor points N_a to be proportional to the number of isoline points N_o and the magnitude of curvature o^{cm} of the isoline. The magnitude of the curvature o^{cm} is approximated as

$$o^{cm} = \sum_{k=1}^{N_o-1} \left(\frac{\mathbf{o}^p_k - \mathbf{o}^p_{k-1}}{\|\mathbf{o}^p_k - \mathbf{o}^p_{k-1}\|} \cdot \frac{\mathbf{o}^p_k - \mathbf{o}^p_{k+1}}{\|\mathbf{o}^p_k - \mathbf{o}^p_{k+1}\|} + 1 \right). \tag{26}$$

Empirically, we set the optimal number of anchor point as $N_a = \lfloor \frac{N_o}{15} \rfloor + \lfloor o^{cm} \rfloor + 3$.

To compute anchor point positions, we aim for the spline points to coincide with the isoline points, $\mathbf{S}^p \approx \mathbf{O}^p$. Let $\mathbf{u}^o = u^o_0, u^o_1, ..., u^o_{N_o-1}$ $(N_o = N_s)$ denote the parameter values for spline points $\mathbf{S}^p$. We set the spline parameter value u^o_k for $\mathbf{s}^p_k$ to its arc length divided by the total length of isoline:

$$u^o_k = \frac{\sum_{t=0}^{k-1} \|\mathbf{o}^p_{t+1} - \mathbf{o}^p_t\|}{\sum_{t=0}^{N_o-1} \|\mathbf{o}^p_{t+1} - \mathbf{o}^p_t\|} \tag{27}$$

Then we build the basis function $\mathbf{B}(\mathbf{u}^o)$ based on the parameter value $\mathbf{u}^o$.

If we simplify the relation between the spline points and the anchor points to be linear, like a regular spline in Euclidean space, isoline points can be represented with linear equation: $\mathbf{O}^p = \mathbf{B}(\mathbf{u}^o)\mathbf{C}^p$.

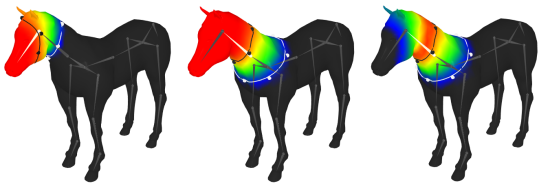
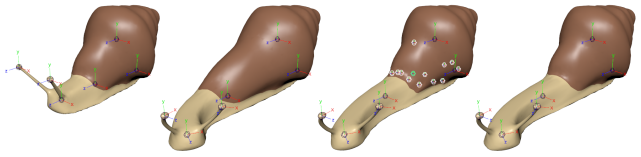


Fig. 16. Left: Skinning weight for the head bone, defined by two splines of iso-values [0.05, 0.9] with weight freezing turned on. Middle: Skinning weight for the neck bone, defined with two splines of iso-values [0.05, 0.9]. Right: Resulting weight for the neck after subtracting the fixed weight due to the head bone.



(a) (b) (c) (d)

Fig. 17. Snail model in a rest pose (a). Deformation result with BBW (b) and BBW with 13 additional point controllers around shell boundary (c). Our result (d).

We can compute anchor points as $\mathbf{C}^p = \mathbf{B}^+(\mathbf{u}^o)\mathbf{O}^p$, where $\mathbf{B}^+$ is a pseudo inverse of the basis matrix. However, this does not guarantee that the anchor points are on the surface. Instead, we regard $\mathbf{C}^p$ as the weighted ($\mathbf{B}^+$) average of $\mathbf{O}^p$, and once again we use (3) with $\mathbf{B}^+$ as the weights and $\mathbf{O}^p$ as the anchor points to compute $\mathbf{C}^p$. For an isoline with an open loop, we constrain the first and last anchor points to coincide with isoline points at both ends. Finally, we use the Phong projection as a post-process to project the anchor points on the surface. This approximation does not guarantee that the fitted spline matches the target isoline exactly, but it gives a satisfying performance in terms of computation time and accuracy. Figure 15 show some results of splines fitted to the isolines of automatically generated skinning weights.

5.2 Anchor Gradient Coefficient Optimization

Since spline fitting in Sec. 5.1.2 is performed only on isolines, the weight distribution made by the fitted spline may not exactly be the same as the input skinning weights $\mathbf{w}^u$, which may have been obtained by manual painting or by automatic skinning algorithms. In particular, the in-between region where the spline is not specified will show different skinning weight results. To reduce the discrepancy, we perform anchor gradient coefficient optimization. We have introduced the gradient control, which adds a gradient skinning weight $\mathbf{w}^g$ on top of skinning weight $\mathbf{w}^s$. We find the optimal gradient anchor coefficient $\mathbf{c}^g$ to minimize the difference by using the equation:

$$\mathbf{c}^g = \mathbf{G}^+(\mathbf{w}^u - \mathbf{w}^s), \tag{28}$$

Where $\mathbf{G}^+$ is the pseudo inverse of the gradient anchor weight matrix $\mathbf{G}$ in (24). Figure 15 shows the fitting results when only the spline fitting is applied and when the gradient optimization is additionally performed. The results show that the gradient optimization significantly reduces the gap of the computed weight from the input weight.

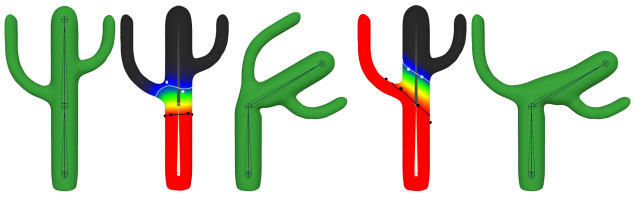
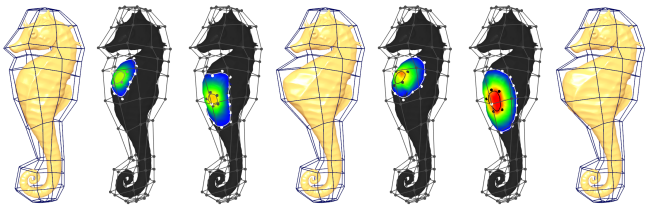


Fig. 18. Cactus model. Skinning weight and its corresponding deformation with different shapes of splines with the same iso-values [0.1, 1.0].



(a) (b) (c) (d) (e) (f) (g)

Fig. 19. From the left, a seahorse model with the cage-based deformation (a). Initial weights obtained by BBW and a resulting deformation (b-d). Weights modified with the spline interface and the corresponding deformation (e-f).

5.3 Anchor Editing

A user modifies the configuration of the spline by editing the anchors. We provide a set of tools that can intuitively edit the anchors:

(1) **Anchor Positioning** allows the user to change the position of the anchor point. As the anchors are repositioned, the skinning weights are recomputed.
(2) **Anchor Deletion** reduces the number of anchors. We delete the entire closed-loop spline if its number of anchors becomes less than 4.
(3) **Anchor Insertion** adds a new anchor point to the user-selected position. The gradient coefficient for the anchor is set to the average of the two adjacent anchors.
(4) **Anchor Splitting** divides an open-loop spline into two or change the topology of a closed loop spline into an open loop.
(5) **Anchor Merging** combines two anchor points into one. Using this operation, two open-loop splines can be merged into one, or an open-loop spline can be changed to a close-loop spline. The anchor gradient coefficient is set to the average of the two merged anchors.

5.4 Weight Freezing

So far, we have focused on the skinning weights for one bone. Since the total sum of weights over the bones must be one for every vertex, the weight modification for a bone necessitates the adjustment of the weights for other bones, typically through the normalization. This may lead to an undesirable consequence where previously edited weights get changed. This problem is also somewhat unavoidable for any skinning weight methods, including the paint interface. To alleviate this problem, we introduce the concept of *weight freezing*. If a user is satisfied with the skinning weights of a specific bone and does not want it to be modified, she can simply fix the skinning

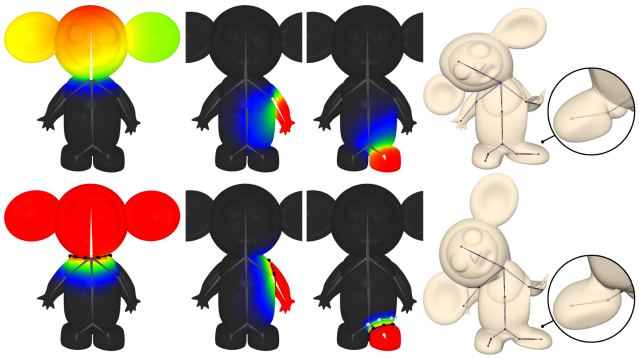


Fig. 20. Skinning weight on the head, arm, and foot of the Cheburashka model and its applied deformation generated using an automatic solution of the BBW (top) and the spline interface (bottom).

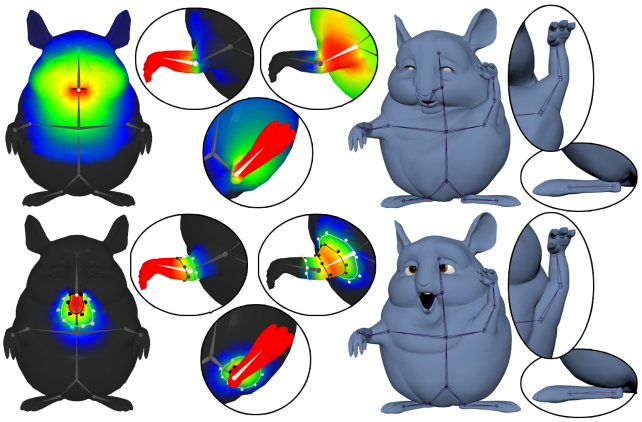


Fig. 21. Skinning weight on the mouth, arm, and foot of a Chinchilla model and its applied deformation generated using an automatic solution of the heat diffusion weight (top) and the spline interface (bottom).

weights for the bone. Normalization then changes the values of only the unfixed skinning weights.

Weight freezing can also be beneficial when defining the skinning weight because the previously fixed weight serve as a constraint. For example, Fig. 16 (left) shows the fixed skinning weight defined on the head region. Fig. 16 (middle) shows the skinning weight computed on the neck region without considering the fixed weight. Due to the effect of the pre-defined fixed weight on the head region, the resulting skinning weight on the neck is computed by subtracting the fixed weight, as can be seen in Fig. 16 (right). Figure 23 also shows the edited skinning weight leveraging the weight freezing technique. The skinning weights of the mouth and ear were defined by fixing the weights from distal to proximal bones. These examples show that the weight freezing is applied to the joints connecting two bones, but the method can also be applied to joints with more bones.

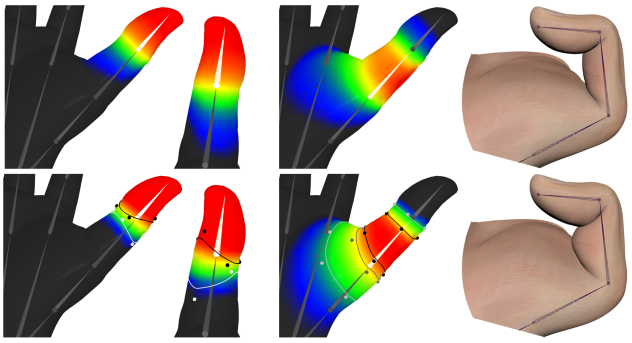


Fig. 22. Skinning weight on the thumb of the hand model and its applied deformation generated using an automatic solution of the heat diffusion weight (top) and the spline interface (bottom).

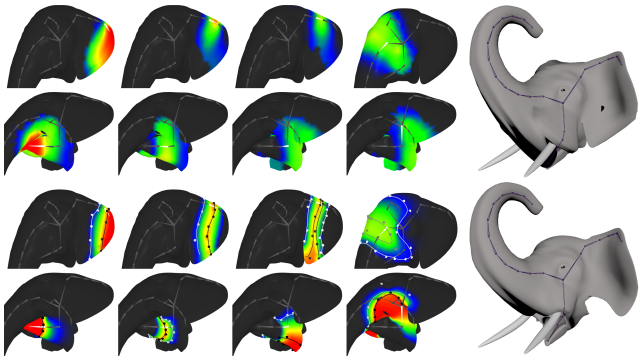


Fig. 23. Skinning weight on the forehead, ear, and mouth of an elephant model and its applied deformation generated using an automatic solution of the GVB (top) and the spline interface (bottom).

6 EXPERIMENTS

We implemented our system as a stand-alone application using NanoGUI as the graphical user interface, and the libIGL library [Jacobson et al. 2016] for most of the geometry processing algorithms, such as Laplacian matrix construction and BBW. We used open source code provided by Panozzo et al. [2013] for computing (3). All of the experiments were performed on a MacBook Pro with 2.8 GHz Intel Core i5 CPU and 16 GB of memory.

The effectiveness of our interface is well demonstrated in a snail example (Fig. 1), which is rigged with 6 point controllers. In order to keep the shell part rigid when deforming the soft body part, ideal skinning weight should change sharply along the boundary between the two parts. We could obtain a satisfying result by generating a spline curve with high iso-value on the boundary of the shell. Automatic methods that do not consider the desired deformation properties of surfaces can only obtain smoothly varying deformation as shown in Fig. 17 (b). To alleviate the problem, one needs to add significantly more controllers. In the snail example, adding 13 more controllers (Fig. 17 (c)) could reproduce similar result as ours (Fig. 17 (d)) although the shell part could not be kept perfectly rigid.

Our spline interface system aims to provide a user with a convenient tool for editing skinning weights while guaranteeing a

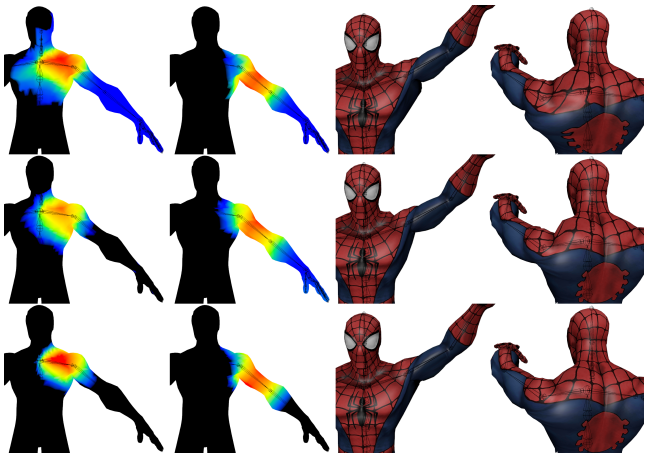


Fig. 24. A Spiderman model with the automatic skinning method using GVB and its applied deformation (top). Modification of the skinning weight using paint interface (middle), and our spline interface (bottom), and their resulting deformation.

smoothness of skinning weight distribution. For example, in Fig. 18, we show that significantly different deformations can be obtained by modifying the skinning weights with our spline interface. Figure 19 (b-d) shows initial weights obtained by an automatic method using BBW and its resulting deformation. Figure 19 (e-g) shows a modified skinning weights and the corresponding deformation by using our spline interface. While the automatic solution gives plausible results, our method can produce different candidates for the skinning weight configuration.

We also present some examples of our method generating natural deformation results, which could not be obtained with existing automatic methods. We show the skinning weights and its deformation results compared with BBW (Figs. 1 and 20), heat weight (Figs. 21 and 22), and GVB (Fig. 23).

Our spline interface is applicable to any type of rigs, such as bones, points, and cages, This is because our method does not depend on the particular features of controllers. We show more results of our skinning weights and corresponding deformations with the cage based deformer in Fig. 19, with point handles in Fig. 1, and with combination of skeleton and point handles in Fig. 25 and 26.

Comparison with Paint-based Interface. We compare our spline interface with the traditional paint-based interface by conducting a user test where a user fixes the original skinning weight obtained with an automatic method. Figure 24 (top) shows the initial skinning weight computed by the GVB method. The excessively spread out weights of the shoulder and the arm skeleton cause unnatural deformation. A user is asked to fix this initial skinning weight with a paint interface method and our spline interface method, respectively. Figure 24 (middle) shows the results of modification by paint interface method, which took 2 minutes and 14 seconds. Figure 24 (bottom) shows the results using the spline interface method, which took 58 seconds. Even with a shorter operation time, the spline interface created a much more plausible result. This is an expected result because moving an anchor point changes the weights

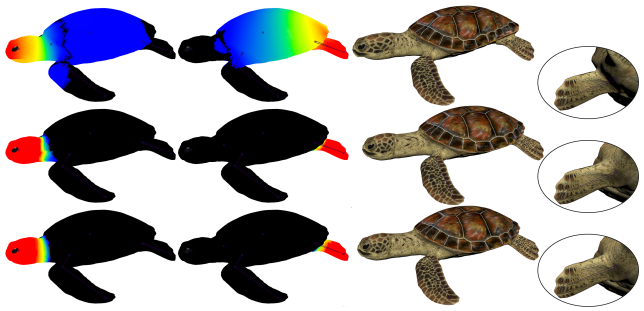


Fig. 25. A turtle model with the automatic skinning method using BBW and its applied deformation(top). Modification of the skinning weight using paint interface (middle), and our spline interface (bottom), and their resulting deformation.

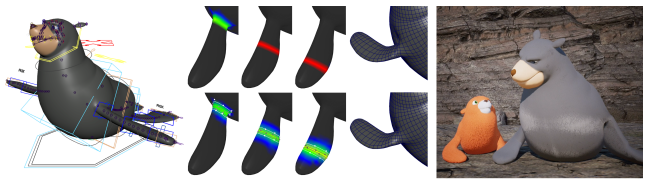


Fig. 26. Seal character used in animation production. Our spline interface was used to improve the skinning weight.

of suitably wide regions while preserving smoothness of the weights, which cannot be easily obtained by the paint-based interface. Figure 25 shows another example with a turtle model. A user is asked to fix an initial skinning weight that excessively spreads out from the head and leg bones. The paint interface took 3 minutes and 15 seconds, but ours took only 33 seconds and produced a more plausible result.

Production Rig. We applied our spline interface to a rig used in the short film production through collaboration with an animation studio. Figure 26 (left) shows a seal character that consists of 205 controllers and complex rig settings ready to be animated. A rigging artist with 1 year of professional experience spent 10 hours to complete the skinning task with a paint interface. Figure 26 (middle, top) shows the skinning weights on the front leg and its applied deformation. Figure 26 (middle, bottom) shows modified skinning weights and the deformation by using our spline interface for 30 minutes. The improved quality of the deformation can be better seen in the accompanying video. Figure 26 (right) shows the final rendered shot of the posed rig.

Local Computation. Table 2 shows a performance comparison between global and local computations with the models presented in this paper. Time for the global computation solely depends on number of vertices (N_v), and our local computation technique mostly depends on the number of anchor points (N_a) and the number of spline points (N_s). As can be seen in the table, local computation schemes achieve significantly better performance than the global computation, and the local computation using anchor weight is more efficient than the local computation using skinning weights.

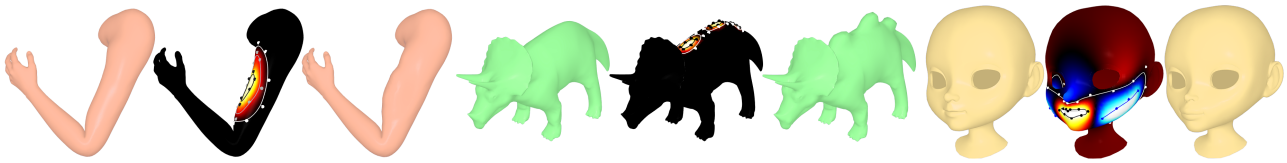


Fig. 27. Mesh deformation by moving vertices in the normal direction by the amount of scalar field generated with our spline interface.

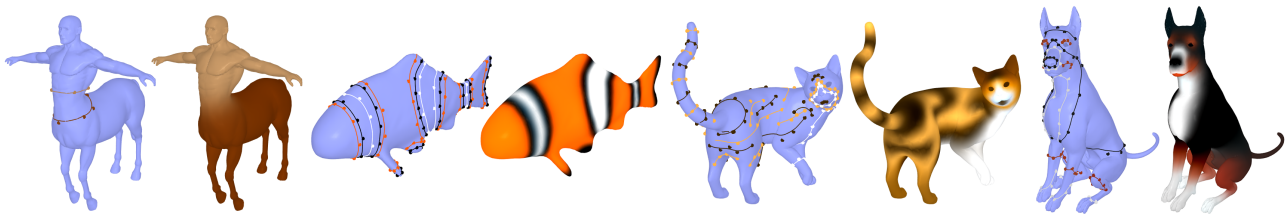


Fig. 28. Result of color interpolation on centaur, fish, cat, and dog models. Splines are visualized with given colors.

	Complexity			Time (ms)			Time (s)
Model	N_v	N_a	N_s	Global	Local (skin)	Local (anchor)	MMDS
Snail	6902	9	91	62	35	26	377
Armadillo	7784	29	216	80	54	24	333
Bunny	3485	22	177	39	37	12	105
Ogre	7846	19	128	71	23	14	364
Horse	3856	11	67	20	10	5	265
Cactus	5261	11	89	47	30	11	111
Seahorse	8559	16	111	104	22	13	395
Cheburashka	6659	14	102	64	15	8	197
Chinchilla	3907	8	45	22	6	4	434
Hand	7707	10	74	78	34	10	285
Elephant	10559	15	111	114	21	12	395
Spiderman	4483	29	176	30	11	8	276
Turtle	8850	12	106	93	43	11	418

Table 2. Computation time of the skinning weights with the global computation, the local computation on skinning weight (Sec. 4.5), the local computation using anchor weight (Sec. 4.6), and the MMDS pre-computation time.

7 OTHER APPLICATIONS

We have introduced the spline interface for defining skinning weights so far. In this section, we show other useful applications that are possible with just small modifications on the algorithm.

7.1 Offset Deformation

Our spline interface can be used to define a scalar field on a surface that encodes the amount of displacement in the normal direction. This gives the effect of the displacement map without resorting to mapping. Figure 27 shows the results of deformation. Scalar fields are defined to create the biceps bulging in the arm example and to create humps on the back of a dinosaur. For the face example, the range limit of $[0, 1]$ is disabled to allow contraction. The spline of the iso-value of 1 is defined on the mouth region and -1 on the chin region to create a face with a protruded mouth. We use the color codes from Fig. 5a for this example with the blue color as negative values.

7.2 Color Interpolation

We can also define color on a surface by specifying the RGB color on splines and interpolating the color. Let $\mathbf{W}^c (\in \mathbb{R}^{N_v \times 3})$ denote the color at every vertex map on a surface, where each column represents R, G, and B colors, respectively. With the anchor weight computed by using (22), the surface color can be computed using the equation:

$$\mathbf{W}^c = \mathbf{K}\mathbf{C}^c, \tag{29}$$

where $\mathbf{C}^c (\in \mathbb{R}^{N_a \times 3})$ is a matrix of color on the anchors. For this application, the partition of the unity constraint is not needed. Figure 28 shows the results of color interpolation on various models.

The results are comparable with the ones obtained with the diffusion curve methods [Jeschke et al. 2009]. The difference is that the diffusion curve method defines curves in texture space whereas our curves are directly controllable on the mesh surface.

8 DISCUSSION AND FUTURE WORK

In this paper, we introduced a novel spline-based interface for editing skinning weights for mesh animation. Our interface brings the benefits of splines, i.e., the ability to create arbitrary curves with controlled smoothness, to the process of defining skinning weights. We have shown its effectiveness through a number of examples.

Our work has some limitations that can be interesting topics for future work. First, the Weighted Average technique [Panozzo et al. 2013], which is used as our basic spline algorithm, requires a one-time computation of Metric Multidimensional Scaling (MMDS) [Cox and Cox 2000], which takes a significant amount of time as shown in the last column of Table 2. While it can be done as a preprocess to maintain real-time performance, an improved spline method that does not require preprocessing would increase the utility of the spline interface.

Second, if a user manipulates control points too quickly, the resulting spline may not be found from our local searching scheme, described in Sec. 3.2. In this case, the global Phong projection needs to be applied, but the computation time increases significantly. A more robust searching scheme will be beneficial for our system.

Third, our spline interface only works on a single component mesh. However, many rigs in the production consist of multiple

mesh patches. Fixing the polygonal soup using previous methods [Jacobson et al. 2013; Xu and Barbič 2014] or building a single enclosing cage using the nested cages [Sacht et al. 2015] can alleviate this limitation. In addition, if a mesh contains complex parts, e.g., small holes, a typical picking-based GUI is inefficient as it may require too many changes of viewing angle to draw a spline. A recently developed method [Krs et al. 2017] that supports 3D interactive curve modeling from a single view can alleviate this problem.

Lastly, we have only considered conventional skinning weights that define deformation configurations to correspond to quasi-static pose changes. While physical simulation can create secondary dynamic deformation of surfaces [Kim et al. 2017], the editing of their deformation characteristics is mostly possible through cumbersome operations, such as changing material parameters. Developing an intuitive tool for editing the global and local characteristics of dynamic deformation remains an important future research problem, and the spline interface may be further improved to contribute to this direction.

ACKNOWLEDGMENTS

We thank Motif Imagenery animation studio for providing the production rig and participating in the user study. We also thank the anonymous referees for their valuable comments and helpful suggestions. This work was supported by Giga Korea Project (GK17P0200) and Basic Science Research Program (NRF-2017R1A2B2006160) funded by MSIT, Korea.

REFERENCES

Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. 2005. SCAPE: Shape Completion and Animation of People. *ACM Trans. Graph.* 24, 3 (July 2005), 408–416.

Seungbae Bang, Byungkuk Choi, Roger Blanco i Ribera, Meekyoung Kim, Sung-Hee Lee, and Junyong Noh. 2015. Interactive Rigging with Intuitive Tools. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 123–132.

Ilya Baran and Jovan Popović. 2007. Automatic rigging and animation of 3D characters. In *ACM Transactions on Graphics (TOG)*, Vol. 26. ACM, 72.

Péter Borosán, Ming Jin, Doug DeCarlo, Yotam Gingold, and Andrew Nealen. 2012. Rigmesh: automatic rigging for part-based shape modeling and deformation. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 198.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon mesh processing*. CRC press.

Trevor F Cox and Michael AA Cox. 2000. *Multidimensional scaling*. CRC press.

C. De Boor. 1978. *A Practical Guide to Splines*. Number V. 27 in Applied Mathematical Sciences. Springer-Verlag. https://books.google.co.kr/books?id=mZMQAQAAIAAJ

Olivier Dionne and Martin de Lasa. 2013. Geodesic voxel binding for production character meshes. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 173–180.

Powei Feng and Joe Warren. 2012. Discrete bi-Laplacians and Biharmonic B-splines. *ACM Trans. Graph.* 31, 4, Article 115 (July 2012), 11 pages. https://doi.org/10.1145/2185520.2185611

Michael S Floater. 2003. Mean value coordinates. *Computer aided geometric design* 20, 1 (2003), 19–27.

Michael Hofer and Helmut Pottmann. 2004. Energy-minimizing splines in manifolds. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 284–293.

Fei Hou, Ying He, Hong Qin, and Aimin Hao. 2016. Knot Optimization for Biharmonic B-splines on Manifold Triangle Meshes. *IEEE Transactions on Visualization and Computer Graphics* (2016).

Alec Jacobson. 2013. *Algorithms and interfaces for real-time deformation of 2d and 3d shapes*. Ph.D. Dissertation. ETH Zurich.

Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.* 30, 4 (2011), 78.

Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 33.

Alec Jacobson, Daniele Panozzo, et al. 2016. libigl: A simple C++ geometry processing library. (2016). http://libigl.github.io/libigl/.

Stefan Jeschke, David Cline, and Peter Wonka. 2009. Rendering surface details with diffusion curves. In *ACM Transactions on Graphics (TOG)*, Vol. 28. ACM, 117.

Jingyi Jin, Michael Garland, and Edgar A Ramos. 2009. MLS-based scalar fields over triangle meshes and their application in mesh processing. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. ACM, 145–153.

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic coordinates for character articulation. In *ACM Transactions on Graphics (TOG)*, Vol. 26. ACM, 71.

Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean value coordinates for closed triangular meshes. In *ACM Transactions on Graphics (TOG)*, Vol. 24. ACM, 561–566.

Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. 2008. Geometric Skinning with Approximate Dual Quaternion Blending. *ACM Trans. Graph.* 27, 4, Article 105 (Nov. 2008), 23 pages.

Ladislav Kavan and Olga Sorkine. 2012. Elasticity-inspired deformers for character articulation. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 196.

Meekyoung Kim, Gerard Pons-Moll, Sergi Pujades, Seungbae Bang, Jinwook Kim, Michael J. Black, and Sung-Hee Lee. 2017. Data-driven Physics for Human Soft Tissue Animation. *ACM Trans. Graph.* 36, 4, Article 54 (July 2017), 12 pages.

Vojtěch Krs, Ersin Yumer, Nathan Carr, Bedrich Benes, and Radomír Měch. 2017. Skippy: single view 3D curve interactive modeling. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 128.

Eric Landreneau and Scott Schaefer. 2010. Poisson-Based Weight Reduction of Animated Meshes. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 1945–1954.

Binh Huy Le and Zhigang Deng. 2012. Smooth skinning decomposition with rigid bones. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 199.

Binh Huy Le and Zhigang Deng. 2014. Robust and accurate skeletal rigging from mesh sequences. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 84.

JP Lewis and Ken-ichi Anjyo. 2010. Direct manipulation blendshapes. *IEEE Computer Graphics and Applications* 30, 4 (2010), 42–50.

Yaron Lipman, Johannes Kopf, Daniel Cohen-Or, and David Levin. 2007. GPU-assisted positive mean value coordinates for mesh deformations. In *Symposium on geometry processing*.

Yaron Lipman, David Levin, and Daniel Cohen-Or. 2008. Green coordinates. In *ACM Transactions on Graphics (TOG)*, Vol. 27. ACM, 78.

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. 2015. SMPL: A skinned multi-person linear model. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 248.

Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient elasticity for character skinning with contact and collisions. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 37.

Alex Mohr, Luke Tokheim, and Michael Gleicher. 2003. Direct manipulation of interactive character skins. In *Proceedings of the 2003 symposium on Interactive 3D graphics*. ACM, 27–30.

Daniele Panozzo, Ilya Baran, Olga Diamanti, and Olga Sorkine-Hornung. 2013. Weighted averages on surfaces. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 60.

Gerard Pons-Moll, Javier Romero, Naureen Mahmood, and Michael J Black. 2015. Dyna: A model of dynamic human shape in motion. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 120.

Raif M Rustamov. 2010. Barycentric coordinates on surfaces. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 1507–1516.

Leonardo Sacht, Etienne Vouga, and Alec Jacobson. 2015. Nested cages. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 170.

Johannes Wallner and Helmut Pottmann. 2006. Intrinsic subdivision with smooth limits for graphics and animation. *ACM Transactions on Graphics (TOG)* 25, 2 (2006), 356–374.

Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. 2015. Linear subspace design for real-time shape deformation. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 57.

Rich Wareham and Joan Lasenby. 2008. Bone glow: An improved method for the assignment of weights for mesh deformation. In *International Conference on Articulated Motion and Deformable Objects*. Springer, 63–71.

Hongyi Xu and Jernej Barbič. 2014. Signed distance fields for polygon soup meshes. In *Proceedings of Graphics Interface 2014*. Canadian Information Processing Society, 35–41.