Pacific Graphics 2009
S. Lee, D. Lischinski, and Y. Yu
(Guest Editors)

Volume 28 (2009), Number 7

ProcDef: Local-to-global Deformation for Skeleton-free Character Animation

Takashi Ijiri[1, 2], Kenshi Takayama[2], Hideo Yokota[1], and Takeo Igarashi[2, 3]
[1]Riken, [2]The University of Tokyo, and [3]JST ERATO

Abstract

Animations of characters with flexible bodies such as jellyfish, snails, and, hearts are difficult to design using traditional skeleton-based approaches. A standard approach is keyframing, but adjusting the shape of the flexible body for each key frame is tedious. In addition, the character cannot dynamically adjust its motion to respond to the environment or user input. This paper introduces a new procedural deformation framework (ProcDef) for designing and driving animations of such flexible objects. Our approach is to synthesize global motions procedurally by integrating local deformations. ProcDef provides an efficient design scheme for local deformation patterns; the user can control the orientation and magnitude of local deformations as well as the propagation of deformation signals by specifying line charts and volumetric fields. We also present a fast and robust deformation algorithm based on shape-matching dynamics and show some example animations to illustrate the feasibility of our framework.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Animation.

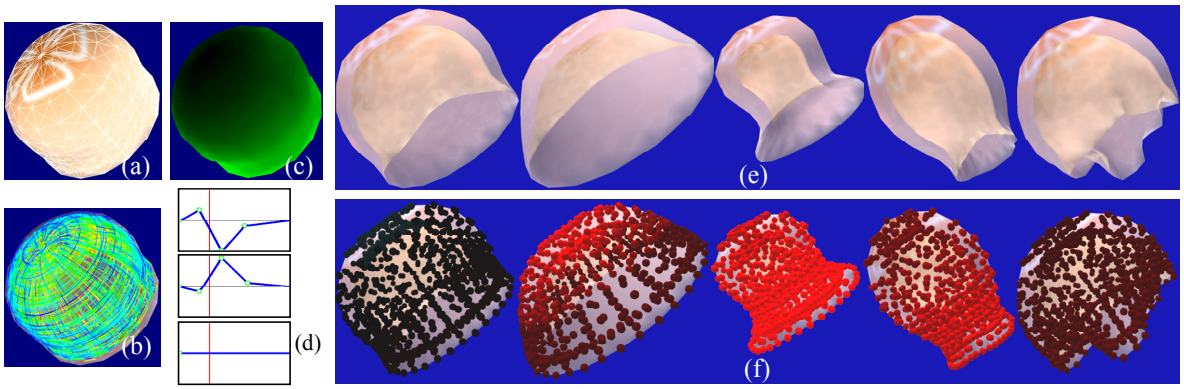

Figure 1: *Swimming jellyfish. The user constructs an orientation field (b) and a phase-shift field (c) on a tetrahedral model (a) and sets deformation charts (d). Based on the specified parameters, the system synthesizes global motions induced by the accumulation of local deformations (e). We visualize the propagation of deformation signals in (f), where excited vertices are highlighted in red.*

1. Introduction

Most animation authoring methods are designed for rigged characters whereby the character body is divided into near-rigid parts such as arms and legs, and the user controls the joint angles between them. However, little work has dealt with the animation of flexible objects without specific skeletal structures such as jellyfish, snails, slugs, hearts, and stomachs. While the typical approach to this type of object is to set a sequence of discrete key poses and interpolate them in time, specifying individual key poses by manipulating many control points is tedious work. Furthermore, making these objects respond to external forces such as contacts and collisions after all key poses are complete is very difficult.

To address this issue, we propose a procedural deformation framework (ProcDef) for designing animations of non-articulated objects that are difficult to handle using existing

Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

skeleton- or keyframing-based methods. The key observation is that the deformations of such flexible objects are driven by expansion and contraction of the local tissues. Local tissues receive some excitation signals and transform their shapes individually, and then the accumulation of the local deformations induces the global motion. For example, a heart is a muscular organ in which muscle fibers are aligned in a spiral direction. When beating, the heart muscles receive an electronic signal arising in the sinoatrial node and locally contract along their fiber orientations. This induces a twisting global motion. Based on this observation, we propose to create the global motion of a flexible object by controlling its local deformations. The user specifies local deformations such as contraction and expansion, and the system synthesizes the global motion by assembling the local deformations and taking the external forces into account. We also show that stimuli–response deformations can be naturally designed within this framework.

We present an efficient design scheme for complicated local deformation patterns. We use line charts and three volumetric fields over the model; these are an orientation field, an amplitude field, and a phase-shift field. The line charts, *called deformation charts*, define cyclic time-varying stretching and contraction of a local element in three directions. The orientation field defines the direction of the local stretching and contraction. The amplitude field allows the space-dependent modulation of the deformation magnitude. This field is particularly useful for designing bending motions. The phase-shift field allows the user to design excitation propagation phenomena. We provide a direct manipulation interface for the line charts and painting interfaces [TOII08] for constructing these three volumetric fields.

We also introduce a robust and efficient algorithm based on shape matching dynamics [MHTG05; RJ07] for computing global motions induced by the accumulations of local deformations. Our algorithm is an extension of lattice shape matching (LSM) [RJ07] with two main differences. First, while LSM uses the original lattice shape as a rest configuration, we locally deform the original shape based on the user-specified parameters and use the deformed shape as a rest configuration. Second, we use a tetrahedral mesh instead of a regular lattice. Since we rely on shape matching dynamics, we can share the same advantages with them: unconditional robustness and low computational cost.

Figure 1 shows an example animation of a swimming jellyfish. The user can easily design such animations by specifying the volumetric fields and the deformation charts. The resulting animations are computed in real time and the user can interactively add external forces by dragging part of an object with the mouse during the animation.

Contributions:

- Introducing a new procedural animation framework whereby global motions are induced by accumulations of local deformations (Sec. 3).

- Presenting a sophisticated scheme for designing local deformation patterns using line charts and three volumetric fields (Sec. 4).

- Showing that shape matching dynamics are useful for synthesizing motions driven by local deformations (Sec. 5).

- Showing some example animations demonstrating the effectiveness of the approach (Sec. 6).

2. Related Work

Local-to-global deformation: The concept of local-to-global deformations was originally presented by Barr [Bar84]. However the paper only dealt with relatively simple cases where local deformations are globally consistent. When local deformations contain some conflicts, local shapes need to compromise each other. Such deformations were presented in biologically motivated studies. Rolland et al. [RBC03] studied the asymmetric growth of the petals of a snapdragon in two dimensions. They represented a petal with an elastic triangular mesh and emulated its growth by modifying the rest length of each edge. Ijiri et al. [IYKI08] extended this model to three dimensions, and Combaz and Neyret introduced a semi-interactive modeling system for wrinkles of fabrics [CN02] and organic shapes such as leaves [CN06]. Their systems deform an elastic surface by locally expanding the user-specified regions and relaxing the accumulated energies. In these systems, the user specifies the local growth rate by textures or painting interfaces. However, these studies are limited to thin membranes and are intended only for growth (expansion) of tissues. In medical science, finite element-based systems have been developed to simulate volumetric deformations of organs such as the heart [AKS*04; WSSH04]. These systems, however, are either for off-line simulations or require specific hardware.

Detail-preserving deformations: Gradient domain mesh deformation method is an active research area. The key concept is to cast mesh deformation as an energy minimization problem. The energy function contains the terms for detail preservation and positional constraints. The detail preservation term involves both the local differentials and local transformations. We refer the reader to surveys of this field [Sor06; BS08]. Note that these methods only consider static surface geometry; it is difficult to apply them to dynamic deformations driven by internal forces.

Deformable models: After Terzopoulos et al. [TPBF87] introduced a finite difference approach, many physically based methods and models have been developed to simulate deformable objects including mass-spring models, and finite element or volume methods. More information is contained in surveys of this field [GM97; NMK*05].

Shape-matching methods have been recently introduced to achieve fast and unconditionally stable deformations. Müller et al. [MHTG05] first applied shape matching to deformable models. They find the best matching rigid (or linear) transformation from an undeformed rest shape to the current deformed shape and determine the goal position

of each particle by the transformed rest shape. They then pull particles to their goal positions. Rivers and James [RJ07] proposed an extension of this approach called lattice shape matching (LSM) to simulate more degrees of freedom. They construct overlapped local regions for all particles of a volumetric lattice. They then apply shape matching to each local region, and blend the results to obtain the smoothed goal positions. Steinemann et al. [SOG08] extended LSM by applying dynamic adaptive sampling to represent heterogeneous stiffness, and Stumpp et al. [SSBT08] apply a similar approach to cloth.

Virtual character design environment: Some platforms exist in which users can create virtual characters and design their motions from scratch. *Modulobe* [EHW*08] provide LEGO-like interfaces. The user can construct virtual creatures by connecting primitives such as rectangular solids or shafts. The user can also control the angles of joints by cyclic functions to design motions. However, these systems are limited to articulated models. *Sodaplay* [Bur] and *Springs World 3D* [Fal] allow the user to create elastic characters. In these systems, the user constructs a model by connecting springs and designs animation by updating the rest length of each spring using a cyclic function. Although these systems are capable of creating elastic character animations, the spring model is unstable for large deformations or strong external forces, and easily generates undesired oscillations. In addition, these systems require the user to control the motion of each spring one-by-one, which makes creating characters with many springs both tedious and difficult.

3. ProcDef: Local-to-Global Animation Framework

We introduce a new procedural deformation framework (ProcDef) for animating flexible objects. Although many studies on deformable models have been carried out, some of which are mentioned above, most of them focus on *passive* deformations caused by external forces. In contrast, this paper focuses mainly on *self-activated* motions. Our framework is useful for designing animations of actively moving flexible characters (e.g., jellyfish and worms) or organs (e.g., hearts and stomachs).

We assume that global motion of a flexible object is driven by the accumulation of local deformations. Figure 2 shows an overview of the ProcDef framework. Before the animation, we prepare a volumetric tetrahedral mesh and define a local region $\boldsymbol{N}_i$ around each mesh vertex $\mathbf{x}_i$ by connecting immediate (1-ring) neighborhood vertices. Neighboring local regions overlap each other (a). In each animation step, we first deform each region $\boldsymbol{N}_i$ of the original mesh based on the user-specified deformation function $\mathbf{T}_i(t)$, and then synthesize a new global shape that satisfies the deformed local regions as much as possible. For example, if we horizontally expand the upper regions and contract the lower regions (b), the global shape bends downward (c). If we contract the upper regions and expand the lower regions (d), the global shape bends in the opposite direction (e).

The local deformation function $\mathbf{T}_i(t)$ depends on vertex i and time t. We define $\mathbf{T}_i(t)$ as a linear transformation to simplify the problem. Note that even locally linear transformations can generate globally complicated nonlinear deformations (Sec. 6). A naive approach is to have the user define the individual $\mathbf{T}_i(t)$ manually as described by [Bur]. However, designing expressive motions in this way is too time-consuming and difficult. In the next section, we therefore present an efficient scheme for defining $\mathbf{T}_i(t)$ with few global controls.

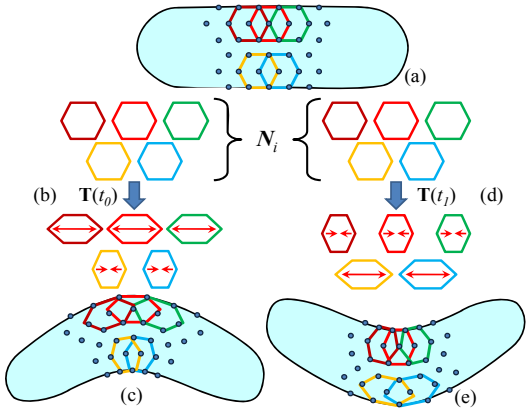

Figure 2: *Overview of ProcDef. We first deform individual local regions $\boldsymbol{N}_i$ (b)(d) of the original mesh (a) by the user-specified $\mathbf{T}(t)$ and synthesize the global shape that satisfies the deformed local shapes (c)(e).*

4. Local Deformation Pattern Design

This section introduces an efficient and intuitive scheme for designing local deformation patterns $\mathbf{T}_i(t) \in \mathbf{R}^{3\times3}$. We assume that a single *static* orientation field (a set of three orthogonal vector fields) is defined inside the model. This orientation field determines the orientations of the local deformations. We believe that this assumption is natural, since organic objects usually have fixed fiber orientations and the local deformations are aligned to the orientations (e.g., the heart). We provide a graph interface called *deformation charts* to control the time-varying cyclic deformations. The system allows the user to optionally specify an amplitude field and a phase-shift field for more complicated deformation patterns. We provide painting interfaces for constructing these fields. We also introduce a *stimuli–response framework* that allows the user to design specific behaviors when the object contacts obstacles.

4.1 Orientation Field Design

In our definition, an orientation field is a set of three orthogonal vector fields in which three orthogonal unit vectors ($\mathbf{d}_1$, $\mathbf{d}_2$, $\mathbf{d}_3$) are defined at each vertex. Since designing appropriate orthogonal vector fields over the whole model is generally difficult for the user, we divide the design process into two steps [TOII08].

In the first step, the user constructs a smooth *layer field* by placing constraint points with a painting interface. The user selects a color that represents a layer value; blue and red correspond to the outermost and innermost parts respectively. The user then paints the color on the model using three tools: a single face fill tool, a flat region fill tool, and a stroke tool [TOII08]. The single face fill tool allows the user to paint the selected color onto a triangle under the mouse cursor, and the system places constraint points with the selected layer value on three vertices of the triangle. The flat region fill tool allows painting on multiple triangles that construct a near-flat surface (the user can choose the angle threshold); the system places constraint points with the selected value at all vertices of the near-flat surface. The stroke tool allows the user to draw a colored stroke on the model surface and the system places constraint points along the stroke. During painting, the user can interactively cut the model by drawing a cutting stroke and paint on the cross sections. After each painting action, the system automatically interpolates the layer values associated with the constraints over the model so that it generates a smooth layer field. We apply a radial basis function (RBF) for the interpolation [TO99], and we use the gradients of the obtained layer field as secondary directions of the orientation field.

In the second step, the user specifies the primary directions of the orientation field by drawing strokes. The user selects a *layer* (iso-surface of the layer field) and draws strokes on the layer. This ensures that the stroke direction is always perpendicular to the gradient direction of the layer field. After drawing each stroke, the system immediately interpolates the stroke orientation over the model; we individually interpolate x, y, and z values of the tangent directions of the input strokes in three-dimensional space using the RBF and then normalize the interpolated x, y, and z values to obtain a unit vector at each vertex. Finally, the system estimates the third directions of the orientation field by taking the cross product of the primary and secondary directions.

Figure 3 shows a design process of an orientation field for the jellyfish model. The user first paints blue (outermost) on the outer surface and red (innermost) on the inner surface of the model using the flat region fill tool (a) to generate the smooth layer field (b). The user can cut the model by drawing a cutting stroke (the red line in (b)) to examine the layer structure. The user next draws four strokes onto a layer to specify the primary direction of the orientation field. Two strokes are placed on the near side and the others are on the far side (c). The system finally constructs the smooth orientation field (d). The three figures in (d) show the primary, secondary, and tertiary orientations from left to right.

4.2 Deformation Charts

After designing the smooth orientation field over the model, the user specifies the local expansion/contraction rate using three line charts that we call deformation charts (Fig. 4). Each of the three charts represents an expansion / contraction rate in the primary, secondary, or tertiary direction of the orientation field. While the horizontal axis corresponds to the phase [0, T], where T is the user-specified time cycle, the vertical axis represents the expansion/contraction rate [$1/A$, A], where $A > 1$ is the user-specified amplitude value. The left and right edges of the charts are connected to represent cyclic deformation behavior. The vertical red bar indicates the timeline and moves from left to right repeatedly. The user can modify the three charts by adding or removing control points or by dragging existing control points in the charts.

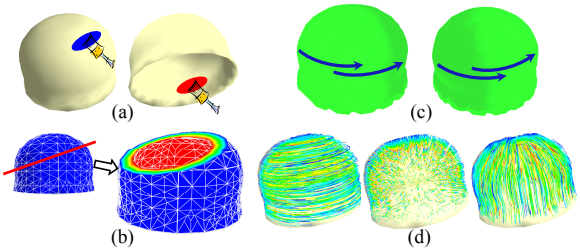

Figure 3: *User interface for designing an orientation field.*

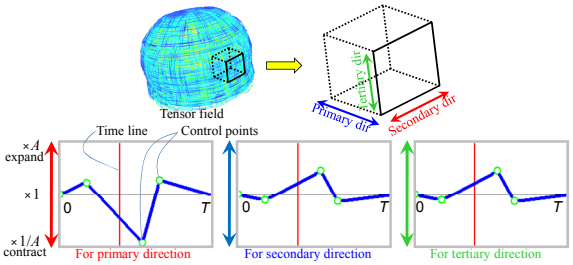

Figure 4: *Deformation charts to define expansion / contraction rate. Top row shows an orientation field and a local region (cube). The three charts define the deformation rate in the primary, secondary, and tertiary directions of the orientation field.*

We also provide a *volume preservation mode*, whereby the user can modify the chart for the primary direction, and the system automatically adjusts the remaining secondary and tertiary charts so that the local transformation always preserves the original volume. The system adjusts the secondary and tertiary charts to maintain $c_1(t) \times c_2(t) \times c_3(t) = 1$, where $c_1(t)$, $c_2(t)$, and $c_3(t)$ are the values of the primary, secondary, and tertiary charts at time t. We provide two options for the user: *adjusting both charts* in which we set $c_2(t) = c_3(t) = 1/\sqrt{c_1(t)}$, or *adjusting the secondary chart only* in which we set $c_2(t) = 1/c_1(t)$ and $c_3(t) = 1$.

Calculation of local linear transformation: Given three directions of the orientation field ($\mathbf{d}_1$, $\mathbf{d}_2$, $\mathbf{d}_3$) and expansion/contraction rates from the deformation charts ($c_1(t)$, $c_2(t)$, $c_3(t)$) for a vertex i at time t, the local linear transformation $\mathbf{T}_i(t)$ is defined as

$$\mathbf{T}_i(t) = \mathbf{D}\,\mathrm{diag}(\,c_1(t), c_2(t), c_3(t)\,)\,\mathbf{D}^{\mathrm{T}}, \qquad (1)$$

where $\mathbf{D} = (\mathbf{d}_1\ \mathbf{d}_2\ \mathbf{d}_3) \in \mathbf{R}^{3\times3}$ and $\mathrm{diag}(c_1, c_2, c_3)$ is a 3×3 diagonal matrix of which the diagonal elements are c_1, c_2, and c_3.

4.3 Amplitude/Phase Shift Fields

The orientation field and the deformation charts deform the entire local regions in the same way and lack the ability to represent rich motions. To support more complicated control, we allow the user to optionally specify amplitude and phase-shift fields [KA08]. We provide painting interfaces to design both of these fields similar to that provided for layer field construction.

The amplitude field allows modifying the magnitude of the local deformations depending on the space. One can bend an object by increasing the local deformation on one side and reducing it on the other side as shown in Figure 5. We define an amplitude field as being a smooth scalar field over the model in which all vertices have values ranging over [–1, 1]. To design this, the user first selects red (value 1), black (value 0), or blue (value –1) and then paints on the model with the selected color using the three painting tools. When a vertex has an amplitude field value A_s, the system modifies a deformation chart value c as $c' = c^{As}$. Thus, when $A_s = 1$, the system does not change the original deformation rate. When $A_s = 0$, the system eliminates the deformation effect by making the rate = 1. When $A_s = -1$, the system inverts the deformation rate.

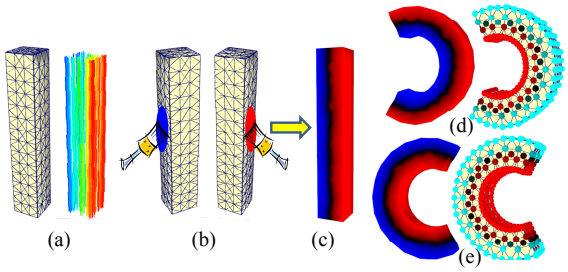

Figure 5: *Bending effect due to an amplitude field. A bar model and its orientation field are shown in (a). The user constructs an amplitude field (c) with the painting interfaces (b). The amplitude field efficiently bends the target model (d) and (e). The expansion rate in the primary orientation is ×1.5 in (d) and ×1/1.5 in (e). Deformed regions are highlighted in red (contraction) or light blue (expansion).*

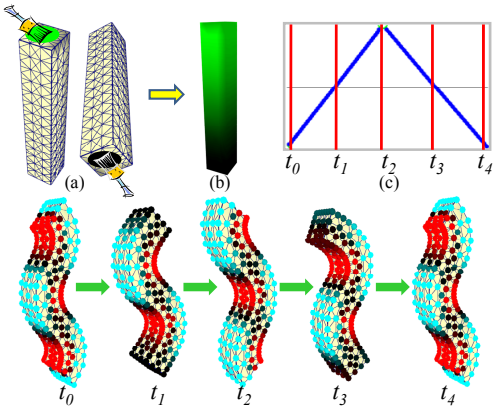

Figure 6: *Deformation propagation due to a phase-shift field. The user constructs a phase-shift field (b) with a painting interface (a) and sets the deformation chart for the primary direction (c). The bottom row shows the sequential poses at times t_0, t_1, t_2, t_3, and t_4 in (c). Contracted and expanded regions are highlighted in red and light blue, respectively.*

The phase-shift field allows the user to vary the timing of local deformations depending on the position. This field is useful for representing motion propagating through the body, such as a swimming motion of a jellyfish or a peristaltic movement of a bowel. Figure 6 shows the effect of this field. In our definition, a phase-shift field is a smooth scalar field ranging over [0, 1]. The user can design this field in a way similar to the amplitude field. In this field, black (value 0) and green (value 1) correspond to the source and sink of the deformation signals. In the animation process, the system shifts phases of the deformation charts according to the specified field as $c'(t) = c(t - P_s/v_p)$, where P_s is the value of the phase-shift field and v_p is the propagation speed specified by the user. The combination of the phase-shift and amplitude fields is useful for generating a wavy motion (e.g., crawling snake).

4.4 Stimuli-Response Deformation

In some natural objects, deformation signals are triggered by environmental stimuli. For example, a snail deforms its body when something touches it, and the deformation usually starts at the contact point. ProcDef can easily emulate such stimuli–response phenomena. Here, we show an example of stimuli–response interaction (Fig. 7).

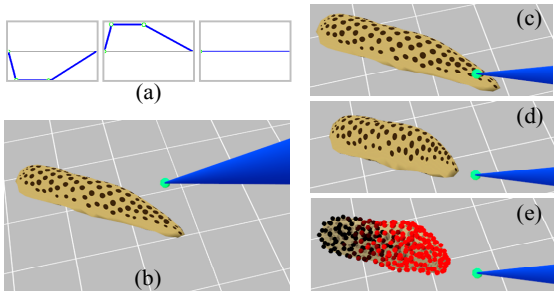

Figure 7: *Stimuli–response interactions. The deformation pattern is specified beforehand (a). When the user touches the object (c), a deformation signal arises from the touched point (d). We visualize excited regions in red (e).*

Before animating an object, the user sets the propagation speed v_p and damping magnitude D_m of a deformation signal, and specifies a deformation pattern using the deformation charts (Fig. 7(a)). In the animation process, the system allows the user to touch the object by mouse clicking (Fig. 7(b)). When the user touches the object, the system constructs a distance field from the touched point over the object, and then propagates the deformation signal along the distance field; the system dynamically constructs an amplitude and phase shift for local regions according to the distance field. When a vertex has a normalized distance field value d, we define its phase-shift value as $P_s = d$, and amplitude value as $A_s = D_m^{-d}$; larger phase shift and smaller

magnitude are applied to distant locations. A stimulus response is not a cyclic motion, so the system sweeps the deformation charts only one cycle. This simple framework successfully emulates deformations triggered by external stimuli (Fig. 7(c) and (d)).

5. Implementation Detail

This section explains our algorithm that synthesizes global motions from local deformations. An object is represented by a volumetric tetrahedral mesh. We construct overlapping local regions $\boldsymbol{N}_i$ at each vertex $\mathbf{x}_i$ by connecting the immediate (1-ring) neighborhood. The deformation of each local region $\boldsymbol{N}_i$ at time t is defined by the user-specified linear transformation $\mathbf{T}_i(t)$.

Our algorithm is based on LSM [RJ07]; we first apply shape matching to all regions and blend the results to obtain smoothed goal positions $\mathbf{g}_i$ of vertex $\mathbf{x}_i$, and then pull vertices to their goal positions. The main differences between LSM and our approach are as follows. While LSM used the original lattice as a rest state, we transform each local region $\boldsymbol{N}_i$ of the original tetra mesh by $\mathbf{T}_i$ and use the deformed shape as a rest state (Fig. 2). Another difference is the use of the tetrahedral mesh instead of a regular lattice. In the lattice representation that embeds an actual model, outer particles of a lattice are usually outside of the model, and thus many regions do not fit inside the actual model. Designing global motions by indirectly controlling such unfitted regions is difficult and unintuitive. In our approach, the tetrahedral mesh, in which all local regions are inside the actual model, allows directly and intuitively specifying local deformations.

Unfortunately, a fast summation operator of fastLSM is not available in our case, since constructing a hierarchical structure for the fast summation in a tetra mesh is difficult. Note that the benefits of the fast summation would not be significant even if it were available, since we use only small local regions (1-ring neighborhood).

Finding goal position by shape matching: We denote the mass of the ith vertex by m_i. To ensure that vertices belonging to many regions are not weighted more than others, we use modified vertex masses, $\tilde{m}_i = m_i / |\boldsymbol{N}_i|$, for shape matching, where $|\boldsymbol{N}_i|$ is the number of vertices in $\boldsymbol{N}_i$. For each local region $\boldsymbol{N}_r$, [RJ07; MTHG05] find the best-fitting rotation matrix $\mathbf{R}_r$ that minimizes

$$\underset{\mathbf{R}_r}{\operatorname{argmin}} \sum_{i \in \boldsymbol{N}_r} \tilde{m}_i \left(\mathbf{R}_r (\mathbf{x}_i^0 - \mathbf{c}_r^0) - (\mathbf{x}_i - \mathbf{c}_r) \right)^2 , \qquad (2)$$

where $\mathbf{x}_i^0$ and $\mathbf{x}_i$ are vertex positions of the undeformed mesh and the current deformed mesh, respectively, and $\mathbf{c}_r^0$ and $\mathbf{c}_r$ are the mass centers of $\boldsymbol{N}_r$ of the undeformed and current mesh,

$$\mathbf{c}_r^0 = \frac{1}{M_r} \sum_{i \in \boldsymbol{N}_r} \tilde{m}_i \mathbf{x}_i^0 , \quad \mathbf{c}_r = \frac{1}{M_r} \sum_{i \in \boldsymbol{N}_r} \tilde{m}_i \mathbf{x}_i , \quad M_r = \sum_{i \in \boldsymbol{N}_r} \tilde{m}_i . \qquad (3)$$

In our case, we use the local shape deformed by the linear transformation $\mathbf{T}_r(t)$ as the rest configuration,

$$\underset{\mathbf{R}_r}{\operatorname{argmin}} \sum_{i \in \boldsymbol{N}_r} \tilde{m}_i \left(\mathbf{R}_r \mathbf{T}_r (\mathbf{x}_i^0 - \mathbf{c}_r^0) - (\mathbf{x}_i - \mathbf{c}_r) \right)^2 . \qquad (4)$$

The best-fitting rotation matrix $\mathbf{R}_r$ can be estimated using the rotation part of

$$\mathbf{A}_r \equiv \sum_{i \in \boldsymbol{N}_r} \tilde{m}_i (\mathbf{x}_i - \mathbf{c}_r) \left(\mathbf{T}_r (\mathbf{x}_i^0 - \mathbf{c}_r^0) \right)^{\boldsymbol{T}} \in \mathbf{R}^{3\times 3} . \qquad (5)$$

We can obtain the rotation matrix $\mathbf{R}_r$ via the polar decomposition $\mathbf{A}_r = \mathbf{R}_r \mathbf{S}_r$, where $\mathbf{S}_r$ is a symmetric matrix $\mathbf{S}_r = \sqrt{\mathbf{A}_r^{\boldsymbol{T}} \mathbf{A}_r}$. Given $\mathbf{R}_r$, $\mathbf{c}_r$, and $\mathbf{c}_r^0$, we estimate the goal position of the ith vertex in the local region $\boldsymbol{N}_r$ as,

$$\mathbf{x}_i = \mathbf{R}_r \mathbf{T}_r (\mathbf{x}_i^0 - \mathbf{c}_r^0) + \mathbf{c}_r \quad i \in \boldsymbol{N}_r . \qquad (6)$$

Finally, we obtain the goal positions by blending the shape- matching results of all local regions,

$$\mathbf{g}_k = \frac{1}{|\boldsymbol{N}_k|} \sum_{i \in \boldsymbol{N}_k} \left(\mathbf{R}_i \mathbf{T}_i (\mathbf{x}_k^0 - \mathbf{c}_i^0) + \mathbf{c}_i \right) . \qquad (7)$$

To accelerate computation, we precompute M_r and $\mathbf{c}_r^0$ and apply a *warm start* [RJ07] when computing $\sqrt{\mathbf{A}_r^{\boldsymbol{T}} \mathbf{A}_r}$.

Iterative approach for stiffness control: The original LSM modifies the size of local regions to control the stiffness. However, this stiffness control is not applicable to our method for the following reason. We assume that a local region is uniformly deformed by a single linear transformation. Thus, a large local region would contain large errors and induce inappropriate global deformations. Deforming large local regions nonuniformly may be possible, for example, with multiple linear transformations. However, this requires the same local-to-global deformation assembly to calculate the nonuniform deformation of a large region, which is too costly. We therefore control the stiffness by repeatedly applying shape matching to small local regions.

When calculating the goal position, we first apply shape matching to the current animated shape $\mathbf{x}_i$, to obtain the goal position $\mathbf{g}^0{}_i$. Next, we use the obtained $\mathbf{g}^0{}_i$ as the target shape and apply the shape matching to $\mathbf{g}^0{}_i$ to obtain $\mathbf{g}^1{}_i$. We can iteratively calculate the $\mathbf{g}^N{}_i$ and use them as the goal shape. N is a user-defined parameter. For a larger value of N, a force applied on a vertex affects vertices farther away, resulting in stiffer deformations (Fig. 8). Note that the computational cost is proportional to N.

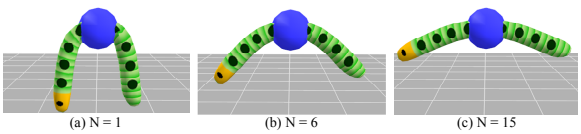

Figure 8: *Stiffness control by iteration. We fix the center of a worm in the air. Each figure shows the rest global shape with different iteration size N.*

Dynamics: When the goal positions are obtained, we update the vertex positions $\mathbf{x}_i$ and velocities $\mathbf{v}_i$ as described by [MHTG05],

$$\mathbf{v}_i(t+h) = \mathbf{v}_i(t) + \frac{\mathbf{g}_i(t) - \mathbf{x}_i(t)}{h} + h\frac{\mathbf{f}_i^{ext}(t)}{m_i} \quad (8)$$

$$\mathbf{x}_i(t+h) = \mathbf{x}_i(t) + h\,\mathbf{v}_i(t+h), \quad (9)$$

where h is the time step of the simulation and $\mathbf{f}_i^{ext}$ is the external force applied to the i-th vertex. We also apply the damping model introduced by [MHHR06; RJ07].

6. Results and Discussion

ProcDef supports general deformations that are induced by muscular tissues and it covers a large variety of possible motions. Figures 1 and 9 show swimming jellyfish and crawling worms designed with ProcDef. The user can easily design these animations by setting the orientation field and the three deformation charts. Our phase-shift field supports the design of deformation propagation effects (Figs. 1(c) and 9(a)). We set the amplitude field to create a bending motion of the worm in Figure 9(c) and 9(e). Figure 10 shows scenes containing many moving objects. In these examples, the system computes both the deformations and collision avoidances between objects in real time. Note that these characters have not been frequently used in video games so far, because it was difficult and costly to design and compute their motions. We hope our method can make such currently unpopular characters to be heavily used in the future.

Our system is also useful for animating organs. We prepared a heart ventricle model and defined its orientation field as described by [TAI*08]. The primary directions of the orientation field are aligned in a spiral form and the secondary directions are oriented in the thickness directions of the ventricular wall (Fig. 11(a)). We then specify the deformation charts to make the local regions contract along the primary directions and expand in the secondary directions (Fig. 11(b)). These parameters generate highly realistic twisting motions of the heart (Fig. 11(c)). We also designed animations of a bowel. We defined a ring-shaped orientation field (Fig. 11(d)) and deformation charts (Fig. 11(e)). We then propagate deformation signals to generate peristaltic motions (Fig. 11(f)). Figure 11(g) highlights the deformation signals; red and light blue indicate contractions and expansions, respectively, in the primary orientation. Since these organ animations are computed in real time on a standard PC and the user can interact with the model, we believe ProcDef will be a useful tool for medical applications such as surgery simulations or electronic charts.

Finally, Table 1 summarizes the performance of our current implementation for each scene in this paper. All timings are generated on a 2.4-GHz Intel Core 2 Duo CPU.

7. Conclusions

In this paper, we presented ProcDef, a new rig-less animation design framework for flexible objects. The key concept is to design the global motions by controlling local deformations. ProcDef permits designing animations of flexible organic objects that have been difficult to be dealt with by skeleton- or keyframing-based approaches. ProcDef also allows interactively adding external forces and stimuli during animation. We provided an efficient scheme for designing local deformation patterns by setting charts and volumetric fields. We applied the shape-matching method to synthesizing motions induced by local deformations robustly and efficiently.

One limitation of our method is the representation of stiffness. Although we introduced an iterative method for stiffness control, its computational cost is linearly proportional to the iteration size N. Development of a more sophisticated stiffness model remains an item for future work. Another item for future research is *inverse* animation design. Currently, we control local deformations, and a global motion only emerges afterward. Allowing a designer to set a global motion and to have the local deformation configurations computed automatically would be appealing and useful. We would also like to combine ProcDef with skeleton- or keyframing-based frameworks for designing more complicated animations.

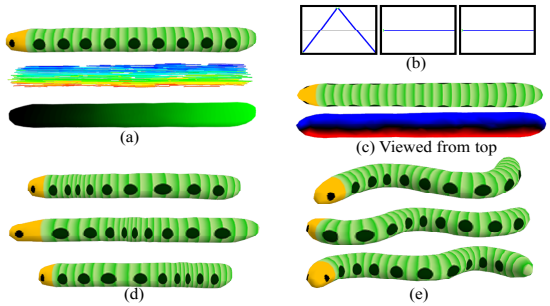

Figure 9: *Crawling worms. We construct an orientation and phase shift field on a worm model (a) and set deformation charts (b) to create a crawling motion (d). We can also design bending motions(e) by setting an amplitude field (c).*

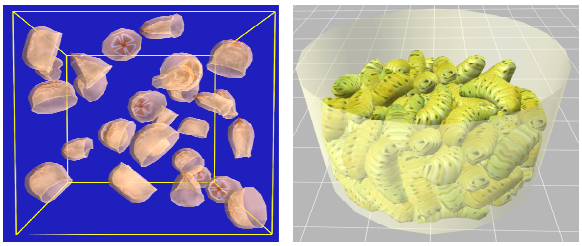

Figure 10: *Multiple moving objects: 31 swimming jellyfishes (left) and 101 short worms (right).*

model	Figure	#Vertices (#Regions)	#Models	Iteration N	Time(msec)
Jellyfish	Fig. 1	854	1	2	4.37
Worm	Fig. 8	470	1	6	5.04
Jellyfishes	Fig. 10left	315	31	1	31.64
Worms	Fig. 10right	73	101	1	34.01
Heart	Fig. 11 a-c	1302	1	6	14.79
Bowel	Fig. 11 d-g	1576	1	6	17.49

Table 1: *Performance of ProcDef. The time row shows the timings (millisecond) for computing local deformations* $\mathbf{T}_i$*, global motions, and collision avoidance, but for rendering.*

Acknowledgements. We thank Dr. Kazuo Nakazawa, Prof. Takashi Ashihara and Dr. Ryo Haraguchi for their helpful comments and kindly providing the heart model (Fig. 11). This paper was funded in part by Adobe Systems Inc. and JSPS research fellowship.

References

[AKS*04] AMANO A., KANDA K., SHIBAYAMA T., KAMEI Y., MATSUDA T.: Model Generation Interface for Simulation of Left Ventricular Motion. *IEEE EMBC. 2004,* 3658-3661.

[Barr84] BARR, A. H. Global and Local Deformations of Solid Primitives, *Computer Graphics* 17, 3(1984), 21-30.

[BS08] BOTSCH, M., SORKINE, O.: On linear variational surface deformation methods. *IEEE TVCG*, 14, 1(2008), 213-230.

[Bur] BURTON E.: Sodaplay. www.sodaplay.com/.

[CN02] COMBAZ J., NEYRET F.: Painting Folds Using Expansion Textures. In *Proc. Pacific Graphics 2002*, 176–182.

[CN06] COMBAZ J., NEYRET F.: Semi-interactive Morphogenesis. In *Proc. Shape Modeling International 2006, 35.*

[EHW*08] ETO K., HAMASAKI M., WATANABE K., KAWASAKI Y., MATSUO Y., NISHIMURA T.: Modulobe: A New 3D Model Creation Platform for Complex Motion Design. In *Proc. CCASNS 2008.*

[Fal] FALCO M.: Springs World 3D. www.sw3d.net.

[GM97] GIBSON S. F., MIRTICH B.: A Survey of Deformable Models in Computer Graphics. *Tech. Rep. TR-97-19, Mitsubishi Electric Research Laboratories, Cambridge, MA.*

[IYKI08] IJIRI T., YOKOO M., KAWABATA K., IGARASHI T.: Surface-based Growth Simulation for Opening Flowers. In *Proc. Graphics Interface 2008,* 227–234.

[KA08] KASS M., ANDERSON J.: Animating Oscillatory Motion with Overlap: Wiggly Splines. *ACM Trans. Graph.* 27, 3(2008).

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless Deformations Based on Shape Matching. *ACM Trans. Graph.* 24, 3(2005), 471–478.

[MHHR06] MÜLLER M., HEIDELBERGER B., HENNIX M., RATCLFF J.: Position Based Dynamics. In *Proc. VRIPhys 2006*, 71–80.

[NMK*05] NEALEN A., MÜLLER M., KEISER R., BOXERMAN E., CARLSON M.: Physically Based Deformable Models in Computer Graphics. *Computer Graphics Forum,* 25, 4(2005), 809–836.

[RJ07] RIVERS A., JAMES D. L.: FastLSM: Fast Lattice Shape Matching for Robust Real-Time Deformation. *ACM Trans. Graph.* 26, 3(2007), 82.

[RBC03] ROLLAND A.-G., BANGHAM J. A., COEN E.: Growth Dynamics Underlying Petal Shape and Asymmetry. *Nature, 422,* 161–163.

[Sor06] SORKINE, O.: Differential representations for mesh processing. *Computer Graphics Forum*, 25, 4(2006), 789–807.

[SOG08] STEINEMANN D., OTADUY M., GROSS M.: Fast Adaptive Shape Matching Deformations. In *Proc. SCA 2008,* 87–94.

[SSBT08] STUMPP T., SPILLMANN J., BECKER M., TESCHNER M.: A Geometric Deformation Model for Stable Cloth Simulation. In *Proc. VRIPHYS 2008,* 13–14.

[TAI*08] TAKAYAMA K., ASHIHARA T., IJIRI T., IGARASHI T., HARAGUCHI R., NAKAZAWA K.: A Sketch-based Interface for Modeling Myocardial Fiber Orientation that Considers the Layered Structure of the Ventricles. *J. Physiol. Sci.* 58, 7(2008), 487–492.

[TOII08] TAKAYAMA K., OKABE M., IJIRI I., IGARASHI T.: Lapped solid textures: filling a model with anisotropic textures. *ACM Trans. Graph.* 27. 3(2008).

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *Proc. ACM SIGGRAPH '87*, 205–214.

[TO99] TURK G., O'BRIEN J. F.: Shape Transformation Using Variational Implicit Functions. In *Proc. ACM SIGGRAPH '99*, 335–342.

[WSSH04] WATANABE H. SUGANO T., SUGIURA S., HISADA T.: Finite Element Analysis of Ventricular Wall Motion and Intra-Ventricular Blood Flow in Heart with Myocardial Infarction. *JSME.* 47, 4(2004). 1019–1026.

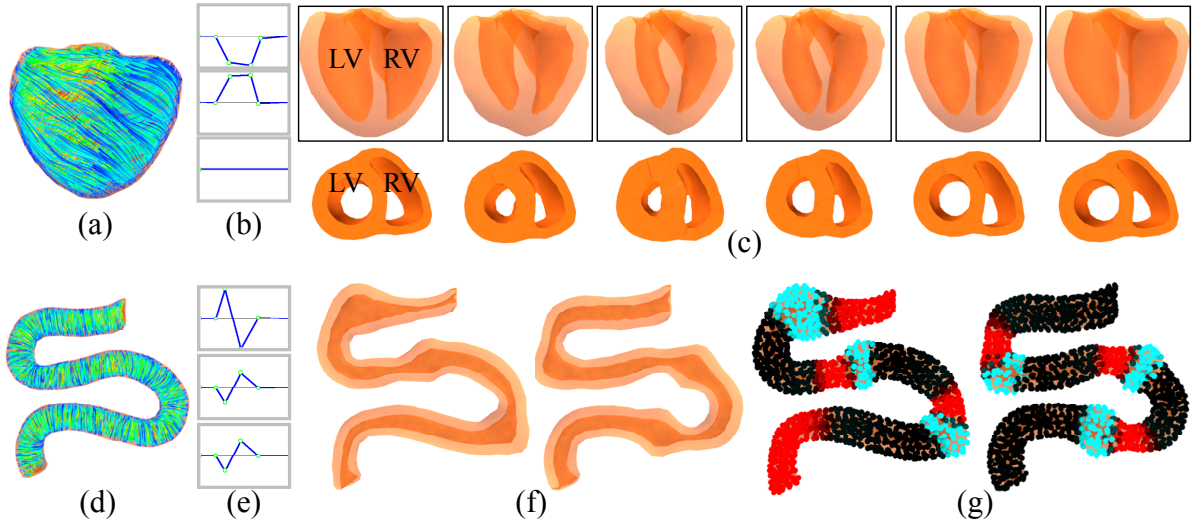

Figure 11: *Animations of a heart and an S-shaped bowel. LV and RV indicate the left and right ventricles of the heart. We show the representative poses of the heart motion in the top row of (c). The bottom row of (c) is an overhead view of the sliced heart model. When beating, the left ventricle (LV) wall strongly thickens to reduce the size of the left ventricle (c).*