COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2014; **25**:353–362
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cav.1580

SPECIAL ISSUE PAPER

Flock morphing animation

Xinjie Wang[1], Linling Zhou[1], Zhigang Deng[2] and Xiaogang Jin[1]*

[1] State Key Lab of CAD&CG, Zhejiang University, Hangzhou 310058, Zhejiang Province, China
[2] Computer Science Department, University of Houston, Houston, TX, USA

ABSTRACT

We propose a new animation technique, called flock morphing, to create special morphing effects between two arbitrary 3D objects by combining the features of 3D morphing and flock animation. Its core idea is first to tetrahedralize the source 3D mesh and regard each tetrahedron as an agent in a flock and then continually generate the flock morphing animation until the target mesh emerges, formed by the same set of tetrahedra. By applying plausible trajectory planning scheme and smooth deformation algorithm, we demonstrate that our proposed method can simultaneously achieve visually desired morphing effects. Copyright © 2014 John Wiley & Sons, Ltd.

KEYWORDS

tetrahedralization; shape morphing; flock simulation; flock morphing; special effects; computer animation

***Correspondence**

Xiaogang Jin, State Key Lab of CAD&CG, Zhejiang University, Hangzhou, Zhejiang Province, China.
E-mail: jin@cad.zju.edu.cn

1. INTRODUCTION

Special effects, which blur the boundaries between art and technology, are playing an important role in entertainment and film industries. During the past several decades, there have been many developments of creating special effects using computer animation techniques. Shape morphing and flock simulation are two important ones among them.

In this paper, we propose a novel flock morphing method to create visually desired special effects that share the visual appearances of 3D shape morphing and flock simulation. Flock morphing produces aggregate motion of tetrahedron particles, and it visually meets velocity matching and flock centering rules. To produce visually pleasing morphing animation, we incorporate many features of flock simulation into our morphing model. To the best of our knowledge, similar techniques have not been reported in the literature to date.

Our algorithm can be described as follows. First, two 3D objects (called source and target) are located in different positions in the working space. Then, we subdivide the two objects until they possess the same number of tetrahedra. Second, each individual tetrahedron in the source object moves in a plausible trajectory and smoothly morphs to its corresponding tetrahedron, along a user-specified global path. After arrival at their destinations, these tetrahedra gradually form the target object. Figure 1 shows one example morphing animation result by our approach.

Similar effects that we have seen in movies are more like extensions of particle systems. Particle systems have the following limitations: (i) correspondences between two mesh objects are not considered; (ii) particles can only form an approximate shape of an object; and (iii) particle systems do not support complex and autonomous behavior patterns.

Our method has four main contributions: (i) it provides a complete solution to this new morphing animation effect by seamlessly combining 3D shape morphing and flock simulation, including tetrahedralization, path planning and control, deformation, and so on; (ii) we introduce a new style control into local tetrahedron trajectory planning, which is useful for changing the patterns of tetrahedra when they are regarded as agents in a flock; (iii) we design a new obstacle avoidance algorithm for velocity generation of the flock, and this algorithm can also be easily integrated into any velocity-based multi-agent simulation systems; and (iv) we introduce a new smooth shape interpolation algorithm between two arbitrary tetrahedra, which can be potentially used for other morphing applications, not limited to the proposed approach.

2. RELATED WORK

Although techniques for producing the visual effect presented in this paper has never been reported in the literature, there exists a large volume of relevant research. It is

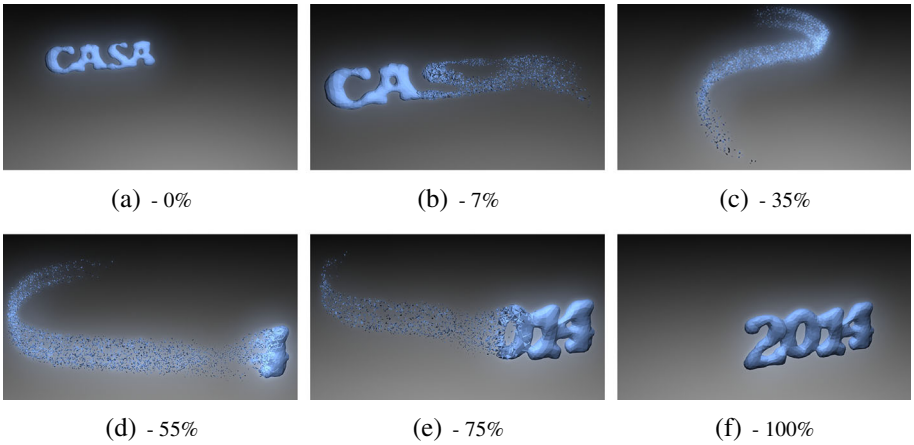


Figure 1. A flock of tetrahedra morphs between two-word models and follows a user-specified path. The percentage value below each frame denotes its corresponding *t* value in the morphing process.

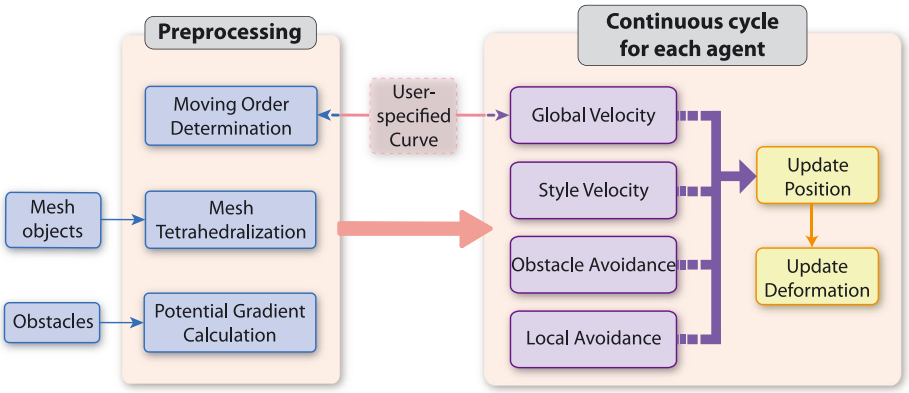


Figure 2. The pipeline of the proposed approach.

beyond the scope of this paper to comprehensively review all previous morphing, motion planning, and crowd simulation methods; as such, we will keep our focus on a few recent, most relevant approaches.

Morphing is a widely-used graphical technique [1]. Two-dimensional image-based morphing [2] has the limitation of lacking the underlying model's spatial information, which may cause visually unpleasant artifacts when it is implemented into 3D animations. As a comparison, 3D mesh-based morphing can produce more realistic and vivid visual effects (A comprehensive survey is provided in [3]). However, most of the existing morphing approaches focus on establishing the one-to-one correspondence between two meshes beforehand, without dealing with crowd (flock) algorithms.

To date, extensive researches have been carried out on crowd simulation [4]; for example, a large volume of existing studies are focused on modeling various patterns of real-world crowds, including flock behavior [5], social force model [6], example-based approaches [7], and so on. Recently, increasing attentions have been drawn to control or generate crowd formations and transitions. For example, Takahashi *et al.* produce sophisticated group formations by combining heuristic rules with explicit hard constraints [8]. Along a different direction, Gu and Deng introduce an automated approach to generate various free-form group formations using a novel sketching interface [9]. Ju and colleagues present a method that blends different crowd data to create a morphable crowd simulation [10]. Furthermore, Xu *et al.* propose an agent-based flock animation system that is capable of satisfying dynamic 3D shape constraints [11]. But all the methods mentioned earlier do not integrate with shape morphing algorithms. In other words, individuals in these methods just group together so that the flock looks like a shape, without changing into another shape during the transformation process.

3. APPROACH OVERVIEW

The pipeline of our approach can be illustrated in Figure 2. At the beginning, two mesh models are specified as source and target. Then, they are tetrahedralized into two sets, which have the same number of tetrahedra. During the morphing process, the source tetrahedron set is moving

DOI: 10.1002/cav

toward the target as a flock simulation, where each tetrahedron can be regarded as an individual agent. At each time step, we compute the velocity of each agent and update its position accordingly. Once a tetrahedron approaches the target area, we will dynamically identify its corresponding (target) tetrahedron. Then, by adjusting its arriving velocity, it will gradually morph to the target tetrahedron. The animation stops once all the tetrahedra reach their goals and together form the target model.

4. MESH TETRAHEDRALIZATION

We adopt "Tetgen" [12] to obtain a tetrahedron set that accurately tetrahedralizes a mesh model. Given a piecewise linear complex (PLC) without intersecting segments or facets, Tetgen can generate a quality mesh of Delaunay tetrahedra with radius-edge ratios no greater than 2.0. Most 3D meshes can be converted into PLCs easily.

Once we have tetrahedralized the source mesh and the target mesh into two tetrahedron sets $\mathbb{S}$ and $\mathbb{T}$, we continue subdividing them until they share the same number of tetrahedra. Let n_1 and n_2 be the number of tetrahedra in $\mathbb{S}$ and $\mathbb{T}$, respectively. Let n be a user-specified parameter, which is the total number of tetrahedra during the morphing, and $n \geq \max(n_1, n_2)$. The default value for n is $\max(n_1, n_2)$. Three different schemes can be used for subdivision, as illustrated in Figure 3.

We first apply one-to-four subdivision scheme (Figure 3(a)) to all the tetrahedra in $\mathbb{S}$. So $\mathbb{S}$ is replaced by a new set with $4^1 \times n$ tetrahedra after one level of subdivision. We repeatedly apply one-to-four subdivision scheme k times where k should satisfy:

$$4^k \times n_1 \leq n < 4^{k+1} \times n_1 \tag{1}$$

It is easy to know that k equals to $\left|\log_4^{\frac{n}{n_1}}\right|$. So far, the subdivided $\mathbb{S}$ has $m = 4^k \times n_1$ tetrahedra. Let

$$d = \left\lfloor \frac{n-m}{3} \right\rfloor, \; r = n - m - 3 \times d \tag{2}$$

We first sort the tetrahedra of the subdivided $\mathbb{S}$ according to their volumes in a descending order and then subdivide the preceding d tetrahedra using one-to-four subdivision scheme. If r equals to 2, we subdivide the tetrahedra with index d with the one-to-three subdivision scheme (Figure 3(b)). If r equals to 1, we subdivide the tetrahedron with index d by the one-to-two subdivision scheme (Figure 3(c)). To this end, both $\mathbb{S}$ and $\mathbb{T}$ have n tetrahedra, respectively.

5. MOVING SCHEME

In this work, we assume that the source mesh and the target mesh do not overlap. Thus, the simplest way to generate morphing animation is to guide all agents to move straightly from their source positions to their goals. However, many scenarios require agents to follow a more flexible and controllable path.

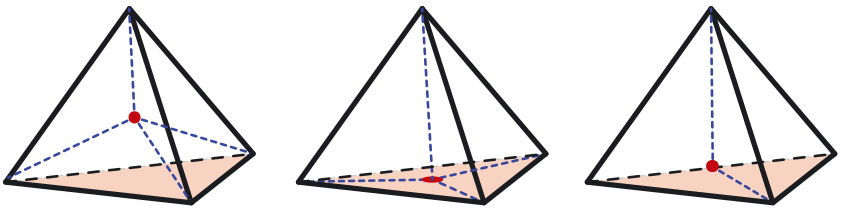
In this section, we describe a two-phase velocity control method that can create many different kinds of plausible trajectories. Agents will apply different velocities according to their positions during different phases, as illustrated in Figure 4. During phase 1, we generate velocity $\vec{V}_1$ for each agent by using a two-level velocity method $\left(\vec{V}_g \text{ and } \vec{V}_l\right)$ and two collision avoidance schemes $\left(\vec{V}_{oa} \text{ and } \vec{V}_{la}\right)$. During phase 2, a simple velocity $\vec{V}_2$ is applied to each agent according to its corresponding target tetrahedron. The next four sections give the details of four velocity fields needed for calculating $\vec{V}_1$, and Section 5.5 describes the detailed computation of $\vec{V}_1$ and $\vec{V}_2$.

5.1. Global Velocity

First, we find out the centroid C_S (or C_T) of the source (or target) mesh. The vector $\overrightarrow{C_S C_T}$ is regarded as the main direction of the global velocities of all the agents in $\mathbb{S}$.

Second, our approach allows users to specify the global path by sketching an arbitrary curve in the 3D space. We sequentially sample m key points (m can be specified by users) on the curve, $Q(Q_0, Q_1, \ldots, Q_m, Q_{m+1})$. Specially, $Q_0 = C_S$ and $Q_{m+1} = C_T$, which indicates that the navigation curve must start from the source and end at the target. Each pair of two adjacent points derives a unit vector $\vec{f}_i$, from the former to the latter (Figure 4). Then, we calculate the global velocities for all the agents:

$$\begin{aligned} &\vec{V}_g(X,t) = \vec{f}_i \cdot speed_0, \\ &\text{where } i = argmin(distance(\vec{x}(t), Q_i) \end{aligned} \tag{3}$$



(a) 1-to-4 (b) 1-to-3 (c) 1-to-2

Figure 3. Three subdivision schemes for a tetrahedron.

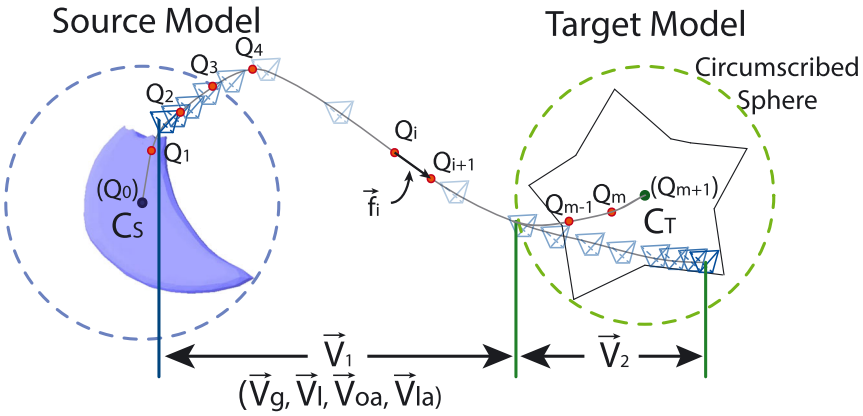


Figure 4. Moving scheme for flock morphing.

In Equation 3, $speed_0$ is a user-specified parameter, which defines the base speed of the flock. X is a tetrahedron, and $X \in \mathbb{S}, \vec{x}(t) = \vec{x}(t, x, y, z)$ means the current 3D position of X's centroid at moment t. The function *distance*() calculates the distance value between two points. We can also define the width of the path and add an attractive velocity on $\vec{V}_g$ to force all agents to move in a fixed width. It is unnecessary to go into the details because it is easy to implement. Besides, it is noteworthy that the path should not be self-intersecting. Otherwise, we may not obtain the correct path.

Moreover, we design the specific moving order of $X \in \mathbb{S}$ according to the path. Generally, when animation starts, all of the tetrahedron agents are stationary in the source model. Those who are in front along the global moving direction will move earlier. On the other hand, at the end of the animation, agents will form the target model by starting with the farthest locations. A concrete example is shown in Figure 1. To achieve this, we define an order $D_1(X)$ for any $X \in \mathbb{S}$, and $D_2(X')$ for any $X' \in \mathbb{T}$:

$$\begin{aligned} D_1(X) &= (Q_0 - Q_1) \cdot \left(\vec{x}(t_0) - Q_1\right) \\ D_2(X') &= -(Q_{m+1} - Q_m) \cdot \left(\vec{x'}(t_0) - Q_m\right) \end{aligned} \tag{4}$$

In the previous equations, t_0 denotes the first time step of the animation, when all the tetrahedra in $\mathbb{S}$ have not started to move yet. As a preprocessing step, we calculate D_1 and D_2 and sort them in an ascending order, respectively.

5.2. Style Velocity

While the global velocity can guarantee agents to move along the path, we still expect that they can exhibit characteristic motion patterns. Therefore, a few existing velocity fields can be integrated with our approach, such as stochastic wind fields, gravitational fields, or noise fields. These velocity fields are used to generate local behaviors, so our approach is able to create a variety of animation scenarios. For example, we can employ the following curl-noise velocity field, inspired by the work of Wang *et al.* [13] and Bridson *et al.* [14], to obtain swarm-like behaviors of the flock:

$$\vec{P}(\vec{x}) = \left(P_1\left(\frac{\vec{x}}{scale}\right), P_2\left(\frac{\vec{x}}{scale}\right), P_3\left(\frac{\vec{x}}{scale}\right)\right) * gain$$

$$\vec{V}_l(X, t) = \nabla \times \vec{P}\left(\vec{x}(t)\right) \tag{5}$$

Here, $\vec{x}$ is a 3D point, and P_1, P_2, and P_3 are three Perlin noise functions with different random seeds. $\vec{x}(t)$ is still X's centroid at moment t. We use two noise parameters: *scale* (for controlling the grid density indirectly) and *gain* (for adjusting the magnitude of the Perlin noise).

In this way, we combine $\vec{V}_g$ (Equation 3) and $\vec{V}_l$ (Equation 5) to obtain the governing velocity as follows:

$$\vec{V}_{grn} = \omega_1 \cdot \vec{V}_g + \omega_2 \cdot \vec{V}_l \tag{6}$$

where ω_1 and ω_2 are adjustment factors.

5.3. Obstacle Avoidance

In many scenes, collisions will happen when the user-specified path passes through the obstacles. Thus, the agents should be aware of the environment and find out a collision-free new path in time. Many obstacle avoidance algorithms have been proposed in the crowd simulation community. Treuille *et al.* [15] design the continuum crowd simulation based on shortest path finding. But their approach does not support user-defined paths. Patil *et al.* [16] introduce a path finding algorithm with a user-sketched guidance curve. However, in their work, path following may fail if the guidance curve intersects with obstacles. In addition, agents distant away from the curve will not be effected strongly by the curve. In our flock morphing animation, all the agents should follow the path as strictly as possible while avoiding obstacles; otherwise, the visual result would not be satisfying. We address this issue by introducing a new obstacle avoidance scheme, described as follows.

First, we calculate a discrete potential field d_t using 3D grids in the environment, where $d_t(i)$ denotes the shortest distance from the center of a grid cell i to the obstacles. Those cells whose centers are inside an obstacle will have $d_t = 0$. d_t distributes in the space and forms a gravitational field, as illustrated in Figure 5. Any points in this field tend to move from higher potential to lower potential. The moving direction of an agent X can be obtained from *potential gradient* $\vec{g}(d_t(X)$ equals to $d_t(i)$ when X is in the grid cell i). When we need to calculate an obstacle-avoidance velocity, we first solve a plane that passes through X and perpendicular to $\vec{g}$, and then we calculate a *repulsion direction* of X by finding a projection unit vector $\vec{D}$ of the V_{grn} on this plane. Figure 5 illustrates this computing process (2D view is chosen for simplicity). Note that if $\vec{D}$ is a zero vector, the repulsion direction of X can be any unit vector on the plane. The obstacle-avoidance velocity V_{oa} is defined as follows:

$$\vec{V}_{oa}(X) = \frac{d_t(X)}{k_a} \cdot \vec{V}_{grn} + \left(1 - \frac{d_t(X)}{k_a}\right) \cdot \vec{D} \cdot speed_0 \quad (7)$$

where k_a is an adjustment factor and denotes the range of the influenced area. As described in Equation 7, agents that are close to an obstacle will be more affected by $\vec{V}_{oa}$ (the agent X_1 in Figure 5). Conversely, agents that are located outside the influenced area will maintain their original velocities (the agent X_2 in Figure 5).

The obstacle avoidance algorithm introduced earlier can be straightforwardly integrated into any velocity-based multi-agent simulation systems by regarding their governing velocity fields as V_{grn} in Equation 7.

5.4. Local Avoidance

In a path planning scheme, the local collision avoidance between agents is important. Considering that a large number of agents may appear in one scene, we address the local inter-agent avoidance issue using a highly efficient method called minimum distance enforcement (MDE) rule to compute a local avoidance velocity $\vec{V}_{la}$, as suggested in [15]. This rule can be briefly summarized as follows: If the distance between two agents is smaller than a threshold (i.e., the minimum distance), we add two symmetrical opposite velocities on them, respectively (illustrated in Figure 6).

5.5. Velocity Synthesis

As illustrated in Figure 4, we divide the moving scheme into two phases, by checking whether the agent is inside the minimal bounding sphere of the target model (phase 2) or not (phase 1). We calculate the final velocities for agents during phase 1 and phase 2, respectively.

Phase 1: For an agent X that is moving, the governing velocity of X should be $V_{grn}(X, t)$ at moment t. We advance X's position by this velocity for δt time. If X is to collide with obstacles, it needs to change its direction at t. Otherwise, it will remain its direction. The final velocity during

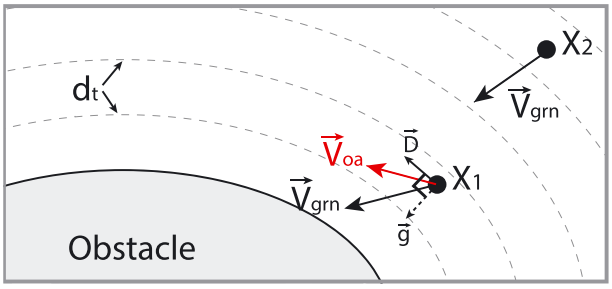
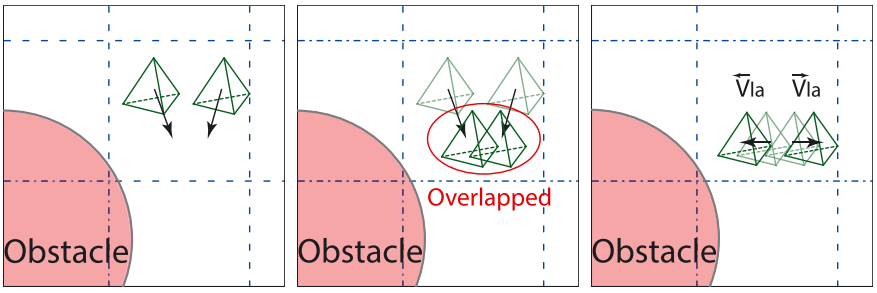


Figure 5. An illustration of computing the obstacle-avoidance velocity.



(a) Before update (b) MDE applied (c) After update

Figure 6. An illustrative example of the minimum distance enforcement rule.

phase 1 can be described in Equation 8, where P_X denotes the current position of X, and R_X denotes the radius of the minimal bounding sphere of tetrahedron X.

$$\vec{V}_1(X,t) = \begin{cases} \vec{V}_{oa}(X,t) + \vec{V}_{la}(X,t) & d_t\left(x(t) + \vec{V}_{syn}(X,t) * \delta t\right) < R_X, \\ \vec{V}_{syn}(X,t) + \vec{V}_{la}(X,t) & \text{Otherwise} \end{cases} \tag{8}$$

Phase 2: When X moves into the minimal bounding sphere of the target model, we immediately find the corresponding tetrahedron X' in the target set $\mathbb{T}$ using Equation 4, and we predict the duration t_f before it arrives at X'. t_f needs to be used to determine V_2 as shown in Equation 9. Note that V_2 is a constant, and we only need to calculate it once.

$$\vec{V}_2(X) = \frac{\overrightarrow{P_{X'}P_X}}{\| \overrightarrow{P_{X'}P_X} \| t_f}, \text{ where } t_f = \frac{\| \overrightarrow{P_{X'}P_X} \|}{speed_0} \tag{9}$$

6. MORPHING SCHEME

Besides the path planning, how to morph between the source and the target tetrahedra is also a non-trivial issue in our approach. Based on our experiments, we found that applying appropriate deformations (shrink, transform, and enlarge) to an individual tetrahedron could achieve visually pleasing flock morphing animations.

Figure 7(a) illustrates our morphing scheme. In order to explain the process more clearly, two key parts of the morphing scheme are zoomed in Figure 7(b) and Figure 7(c), respectively. As in Figure 7(b), a tetrahedron quickly shrinks to one-eighth of its original size when it starts to move. The situation is somewhat different when the tetrahedra tend to form the target. As shown in Figure 7(c), first, a smooth shape transformation is performed between the shrinked tetrahedron X_s and the corresponding shrinked target tetrahedron X'_s. Then, X'_s is enlarged gradually until it becomes the same as X'. Meanwhile, X'_s arrives at the target position. We present an efficient way to process shape transformations based on an affine mapping algorithm.

An affine transformation is a transformation that preserves collinearity and ratios of distances, and it can be uniquely determined by four pairs of points. Inspired by the work of Farin [17], an affine transformation can be expressed as follows:

$$\mathbf{x}' = \mathbf{d} + \mathbf{A}(\mathbf{x} - \mathbf{o})$$

where $\mathbf{o}$ is the origin of $\mathbf{x}$'s coordinate system and $\mathbf{x}'$ is the image of under affine map $\mathbf{x}$ displaced by vector $\mathbf{d}$. Let $T(\mathbf{p}_1\mathbf{p}_2\mathbf{p}_3\mathbf{p}_4)$ be the source tetrahedron and $T'(\mathbf{p}'_1\mathbf{p}'_2\mathbf{p}'_3\mathbf{p}'_4)$ be the target tetrahedron. We assume that $\mathbf{p}_i$ corresponds to $\mathbf{p}'_i$. To calculate the affine transformation from T to T', we set up a coordinate system at T by taking $\mathbf{P}_1$ as the origin and $\mathbf{p}_i - \mathbf{p}_1 (i = 2, 3, 4)$ as three axes. The coordinate

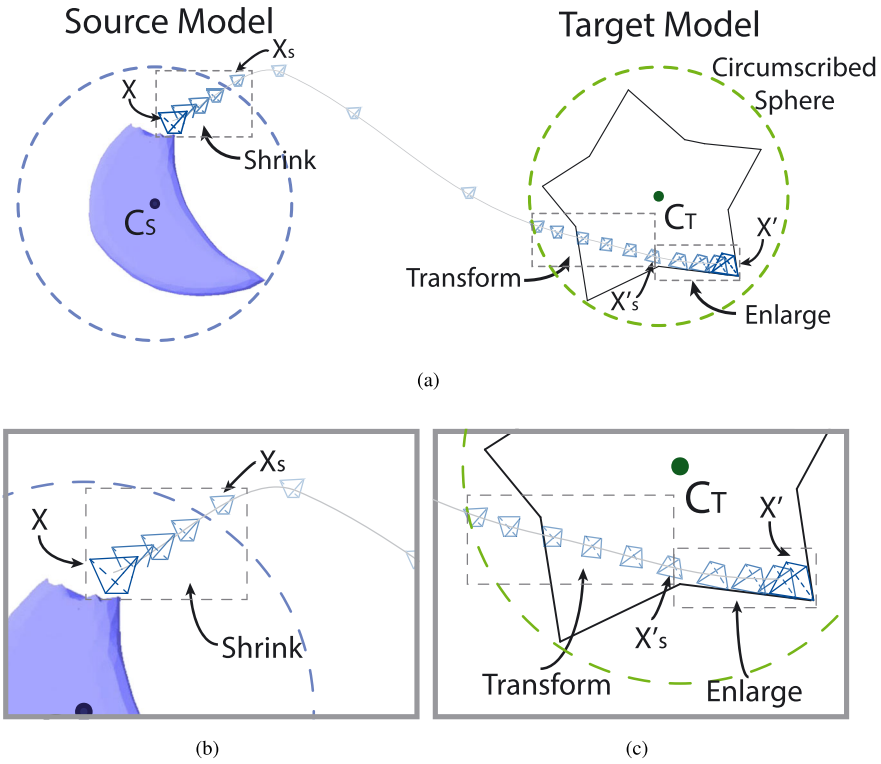


Figure 7. Morphing scheme for flock morphing.

system at T' can be set up similarly based on the same indices. Points in the two coordinate systems can be mapped by

$$\mathbf{x}' = \mathbf{p}'_1 + \mathbf{A}(\mathbf{x} - \mathbf{p}_1)$$

Then, the non-translational portion $\mathbf{A}$ of the affine transformation from T to T' can be computed as follows:

$$\mathbf{A} = [\mathbf{p}'_2 - \mathbf{p}'_1, \mathbf{p}'_3 - \mathbf{p}'_1, \mathbf{p}'_4 - \mathbf{p}'_1] * [\mathbf{p}_2 - \mathbf{p}_1, \mathbf{p}_3 - \mathbf{p}_1, \mathbf{p}_4 - \mathbf{p}_1]^{-1}$$

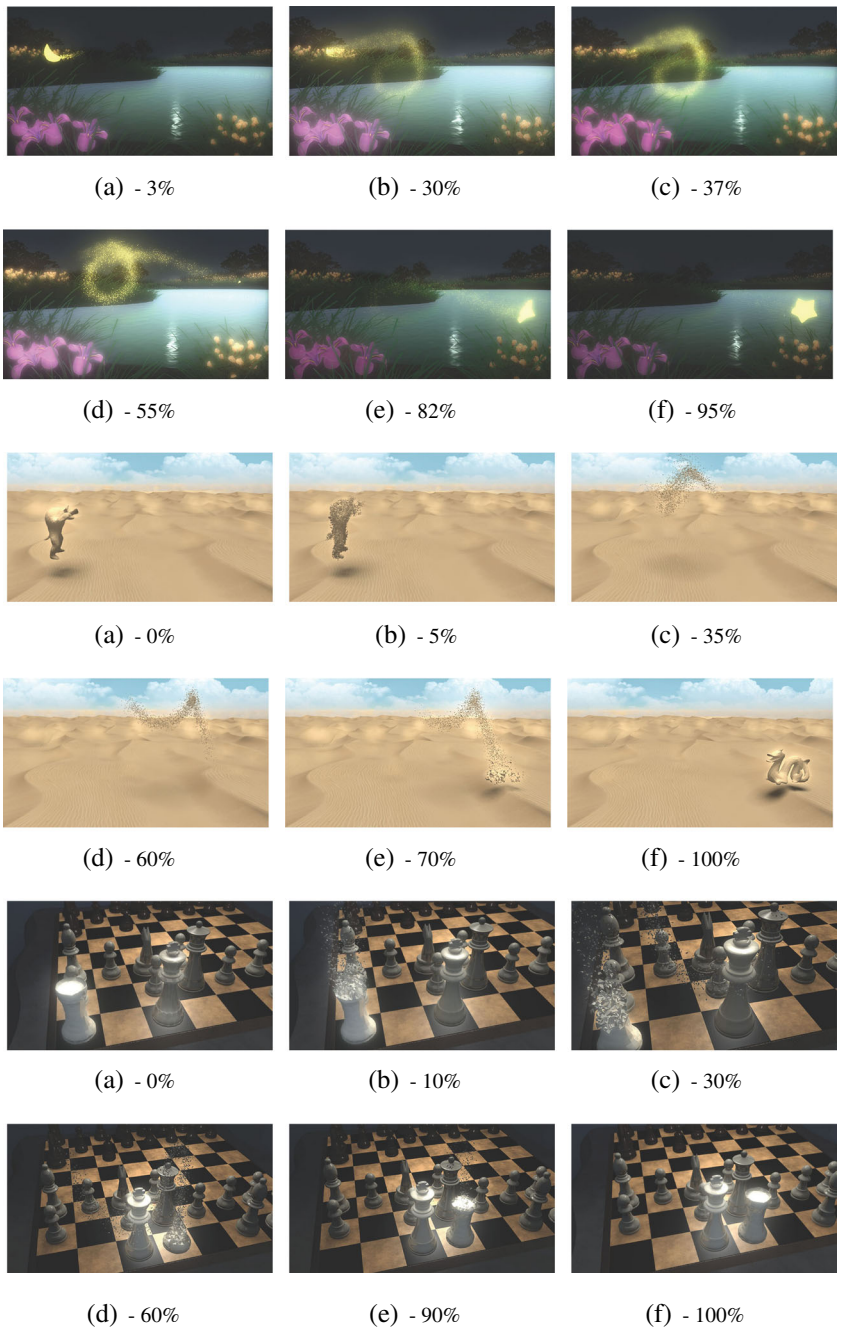


Figure 8. Results of our approach: (top example) a moon-like model morphs to a star; (middle example) two imaginary creatures transform from one to another in a desert; (bottom example) in a castling situation of a chess game, the rook and the king morph to the target subtly instead of moving rigidly. The percentage value below each frame denotes its corresponding *t* value in the morphing process.

A is a composite transformation that encodes rotation, scaling, and shear transformations. If we interpolate it directly with an identity matrix **I**, that is, $(1-t)\mathbf{I}+t\mathbf{A}$, the results are not valid. In 1994, Shoemake showed that the polar decomposition is a good way to interpolate an affine transformation [18]. The polar decomposition factors **A** are represented as follows:

$$\mathbf{A} = \mathbf{RS}$$

where **R** is an orthogonal matrix and **S** is a symmetric positive definite matrix. The factor **S** is always unique and can be expressed as $\mathbf{S} = (\mathbf{A}^T\mathbf{A})^{\frac{1}{2}}$, where exponent $\frac{1}{2}$ denotes the principal square root. The factors produced by polar decomposition are unique and coordinate-independent.

In order to proceed the polar decomposition, we first converse **R** to its unit quaternion representation $q = (w,x,y,z)$ and then further express q as $q = \left(cos\frac{\theta}{2}, \mathbf{n}sin\frac{\theta}{2}\right)$, where θ is the rotation angle and **n** is the unit rotation axis. Thus, we can interpolate the rotation between two tetrahedra explicitly by interpolating the unit quaternion. q can be converted into the rotation matrix $\mathbf{R}(\mathbf{n},\theta)$.

Now, the interpolated composite matrix can be written as follows:

$$\mathbf{A}(t) = (1-t)\mathbf{I} + t\mathbf{R}(\mathbf{n},t\theta)\mathbf{S}, 0 \le t \le 1$$

Decomposing matrix **A** into the polar form makes it possible to interpolate the intrinsic rotation separately, which leads to smooth transformation. Now, using the previous formula, the positions of the linearly interpolated tetrahedron, $T(\mathbf{p}_1(t)\mathbf{p}_2(t)\mathbf{p}_3(t)\mathbf{p}_4(t))$, can be computed as follows:

$$\begin{aligned}
\mathbf{q}_1(t) &= (1-t)\mathbf{p}_1 + t\mathbf{p}_1' \\
\mathbf{q}_2(t) &= \mathbf{q}_1(t) + \mathbf{A}(t)(\mathbf{p}_2 - \mathbf{p}_1) \\
\mathbf{q}_3(t) &= \mathbf{q}_1(t) + \mathbf{A}(t)(\mathbf{p}_3 - \mathbf{p}_1) \\
\mathbf{q}_4(t) &= \mathbf{q}_1(t) + \mathbf{A}(t)(\mathbf{p}_4 - \mathbf{p}_1)
\end{aligned}$$

7. EXPERIMENTAL RESULTS

We used our approach to generate a variety of flock morphing animations and found that our approach can produce visually satisfactory effects. Our method can be applied to any 3D objects with arbitrary topology to create visually pleasing evolving effects from the source object to the target object. All the transitions are visually smooth during the whole animation. Several selected simulation scenarios are discussed in this section. Please refer to the accompanying demo video for animation results.

Words morphing: Figure 1 clearly shows the process of our approach. Tetrahedra detach from the word "CASA" in order and find their paths to form the target word "2014." The transformations and scalings between tetrahedra can be obtained during the animation. Because there is no style velocity field (i.e., $\vec{V}_l = 0$), agents move and follow the user-specified curve strictly.

Table I. Parameter settings and performance statistics of our approach for selected simulation scenarios.

Scenario	Number of agents	$Speed_0$	Parameters	Grid resolution	Style factors	Preprocessing time (second)	Simulation FPS (CPU)	Average MDE percentage per frame (%)
Words	13,818	10.0	$\omega_1 = 1.0$ $\omega_2 = 0.0$	80 × 120 × 60	—	1.536	185.4	0.13
The moon	3474	8.0	$\omega_1 = 0.7$ $\omega_2 = 0.3$	120 × 40 × 40	Stochastic wind: *WindSpeed* = 0.5	0.454	31.3	0.72
Creatures	4811	10.0	$\omega_1 = 0.5$ $\omega_2 = 0.5$	120 × 100 × 65	Curl-noise: *gain* = 0.3 *scale* = 0.6	0.508	17.8	0.63
Castling - King	4146	6.5	$\omega_1 = 1.0$ $\omega_2 = 0.0$	100 × 80 × 60	—	0.412	322.4	0.02
Castling - Rook	4146	5.5	$\omega_1 = 0.8$ $\omega_2 = 0.2$ $k_a = 0.01$	80 × 120 × 60	Curl-noise: *gain* = 0.006 *scale* = 0.015	2.159	13.1	1.27

The calculation of style velocities consumes extra time in scenarios 2, 3, and 5.

Comp. Anim. Virtual Worlds 2014; **25**:353–362 © 2014 John Wiley & Sons, Ltd.
DOI: 10.1002/cav

The moon: In this simulation, we employ a stochastic wind velocity field as a style velocity field. While agents are constrained by the curve, they are also affected by the wind field. In Figure 8 (top example), the moon vanishes as it is blown away by the wind, creating a flow-like morphing effect. Note that the navigation curve in this scenario does not actually intersect with itself, just an illusion when the scene is rendered into 2D.

Imaginary creatures: In Figure 8 (middle example), the curl-noise style velocity field is employed to make agents move like insect swarms (described in Section 5.2).

Castling: This scenario shows how agents deal with obstacles. After the king moves two squares toward the rook, it becomes an obstacle blocked in the morphing path of the rook. Then, the rook starts to morph, and all the tetrahedron agents can be aware of the king and bypass it. In Figure 8 (bottom example), the rook's morphing process is demonstrated. We encourage readers to check the complete castling animation in our companion video.

All the previous scenarios were simulated on an off-the-shelf computer with an Intel 2 DUO Central Processing Unit (CPU) E7500 and 4 G main memory. Table I shows the performance statistics of the previous four scenarios. The regular 3D grid defined in each scenario is employed for storing style velocities (at the centers) and MDE calculation. The obtained simulation Frames Per Second (FPS) shows that our approach can run in near real-time on an off-the-shelf computer. Besides, the average percentage of MDE per frame is recorded in the last row of Table I. In all the previous scenarios, the percentage of calling the MDE rule is significantly below 2%.

8. DISCUSSION AND CONCLUSIONS

In this paper, we introduce a highly efficient approach to create visually pleasing flock morphing animations. Unlike existing shape morphing and flock simulation methods, our approach seamlessly combines the two research directions to produce a new type of visual special effect. Our approach provides flexible controls to users including tuning those parameters or employing pre-defined velocity fields. Another advantage of our approach is its time efficiency. For example, our approach real-time supports large-scale simulations (i.e., over 10^4 tetrahedra agents) on an off-the-shelf computer.

However, a number of caveats need to be noted regarding our current approach. First, the minimal bounding sphere that we use for determining whether the agents is close enough to the target model may not work well when the target model is too narrow. But adopting a minimal bounding box, instead of the minimal bounding sphere, would cause some moving artifacts, because the bounding box is anisotropic and agents may fail to move in order. We will leave this issue to the future work.

Second, global avoidance and local avoidance are not supported when agents move inside the minimal bounding spheres. Controls of the agents' velocities are restricted because moving to the target positions is the primary task of the agents. In other words, agents may collide with those obstacles close to the target model or collide with each other before they reach their destinations. The latter seems less critical because agents will collide with each other anyway at the final step. We plan to explore new and better path-control schemes to further improve the results. As the future work, we would like to extend our approach to handle 3D models in other forms (not tetrahedra), such as irregular polyhedra, meta-balls, or more complex elements.

ACKNOWLEDGEMENTS

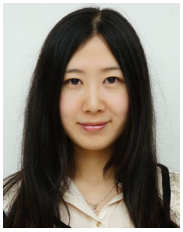
This work was supported by Zhejiang Provincial Natural Science Foundation of China (grant no. Z1110154), the Joint Research Fund for Overseas Chinese, Hong Kong and Macao Young Scientists of the National Natural Science Foundation of China (grant no. 61328204), and the National Natural Science Foundation of China (grant no. 61272298). The authors would like to thank Junjie Chen for English proofreading.

REFERENCES

1. Gomes J, Darsa L, Costa B, Velho L. *Warping and Morphing of Graphical Objects*. Morgan Kaufmann Publishers Inc.: San Francisco, USA, 1998.
2. Wolberg G. Image morphing: a survey. *The Visual Computer* 1998; **14**(8–9): 360–372.
3. Alexa M. Recent advances in mesh morphing. *Computer Graphics Forum* 2002; **21**(2): 173–198.
4. Zhan B, Monekosso DN, Remagnino P, Velastin SA, Xu LQ. Crowd analysis: a survey. *Machine Vision and Applications* 2008; **19**(5–6): 345–357.
5. Reynolds CW. Flocks, herds and schools: a distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '87. ACM: New York, USA, 1987; 25–34.
6. Helbing D, Molnár P. Social force model for pedestrian dynamics. *Physical Review E* 1995; **51**: 4282–4286.
7. Lee KH, Choi MG, Hong Q, Lee J. Group behavior from video: a data-driven approach to crowd simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '07. Eurographics Association: Aire-la-Ville, Switzerland, 2007; 109–118.
8. Takahashi S, Yoshida K, Kwon T, Lee KH, Lee J, Shin SY. Spectral-based group formation control. *Computer Graphics Forum* 2009; **28**(2): 639–648.
9. Gu Q, Deng Z. Generating freestyle group formations in agent-based crowd simulations. *IEEE Computer Graphics and Applications* 2013; **33**(1): 20–31.

10. Ju E, Choi MG, Park M, Lee J, Lee KH, Takahashi S. Morphable crowds. *ACM Transactions on Graphics* 2010; **29**(6): 140:1–140:10.
11. Xu J, Jin X, Yu Y, Shen T, Zhou M. Shape-constrained flock animation. *Computer Animation and Virtual Worlds* 2008; **19**(3–4): 319–330.
12. Si H. Tetgen: a quality tetrahedral mesh generator and three-dimensional Delaunay triangulator. *Technical Reports 9*, Weierstrass Institute for Applied Analysis and Stochastic, Berlin, Germany, 2004.
13. Wang X, Jin X, Deng Z, Zhou L. Inherent noise-aware insect swarm simulation. *Computer Graphics Forum* 2014. DOI: 10.1111/cgf.12277. (published online).
14. Bridson R, Houriham J, Nordenstam M. Curl-noise for procedural fluid flow. *ACM Transactions on Graphics* 2007; **26**(3): 1–4.
15. Treuille A, Cooper S, Popović Z. Continuum crowds. *ACM Transactions on Graphics* 2006; **25**(3): 1160–1168.
16. Patil S, van den Berg J, Curtis S, Lin M, Manocha D. Directing crowd simulations using navigation fields. *IEEE Transactions on Visualization and Computer Graphics* 2011; **17**(2): 244–254.
17. Farin G. *Curves and Surfaces for CAGD: A Practical Guide* 5edn. Morgan Kaufmann Publishers Inc.: San Francisco, USA, 2002.
18. Shoemake K. Graphics gems IV. In *Graphics Gems IV*, Heckbert Paul S. (ed.). Academic Press Professional, Inc.: San Diego, USA, 1994; 207–221.

AUTHORS' BIOGRAPHIES



Xinjie Wang is a PhD candidate of the State Key Lab of CAD&CG, Zhejiang University, China. She received her BSc degree in computer science and technology from Wuhan University in 2010. Her research interests include computer animation, flock simulation, and crowd simulation.

Linling Zhou received her BSc degree in digital media technology in 2011 and MSc degree in computer technology in 2014, all from Zhejiang University. Her research interests include computer animation and interactive design.

Zhigang Deng is an associate professor of Computer Science at University of Houston (UH) and the Founding Director of the UH Computer Graphics and Interactive Media (CGIM) Lab. His research interests are in the broad areas of computer graphics, computer animation, human-computer interaction, virtual human modeling and animation, and visual computing for biomedical applications. Besides the CASA 2014 general co-chair, he currently serves as an associate editor of Computer Graphics Forum, and Computer Animation and Virtual Worlds Journal. He earned his PhD in computer science at the University of Southern California in 2006. Prior to that, he completed his BS degree in Mathematics from Xiamen University (China) and MS in Computer Science from Peking University (China). Over the years, he had worked at the Founder Research and Development Center (China) and AT&T Shannon Research Lab.

Xiaogang Jin is a professor of the State Key Lab of CAD&CG, Zhejiang University, China. He received his BSc degree in computer science in 1989 and MSc and PhD degrees in applied mathematics in 1992 and 1995, respectively, all from Zhejiang University. His current research interests include traffic simulation, insect swarm simulation, physically based animation, cloth animation, special effects simulation, implicit surface computing, non-photorealistic rendering, computer-generated marbling, and digital geometry processing.