Mesh Ensemble Motion Graphs: Data-driven Mesh Animation with Constraints

Doug L. James
Cornell University[1]

Christopher D. Twigg
Andrew Cove
Carnegie Mellon University

Robert Y. Wang
Massachusetts Institute of Technology

We explore the use of space-time cuts to smoothly transition between stochastic mesh animation clips involving numerous deformable mesh groups while subject to physical constraints. These transitions are used to construct *Mesh Ensemble Motion Graphs* for interactive data-driven animation of high-dimensional mesh animation datasets, such as those arising from expensive physical simulations of deformable objects blowing in the wind. We formulate the transition computation as an integer programming problem, and introduce a novel randomized algorithm to compute transitions subject to geometric nonpenetration constraints. We present examples for several physically based motion datasets, with real-time display and optional interactive control over wind intensity via transitions between wind levels. We discuss challenges and opportunities for future work and practical application.

Categories and Subject Descriptors: G.1.6 [**NUMERICAL ANALYSIS**]: Optimization—*Integer programming*; I.3.5 [**COMPUTER GRAPHICS**]: Computational Geometry and Object Modeling—*Physically based modeling*; I.6.8 [**SIMULATION AND MODELING**]: Types of Simulation—*Animation*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: collision detection, constraint satisfaction, integer programming, motion graphs, nonpenetration

1. INTRODUCTION

Dynamic mesh ensembles can be used to model complex deforming environments, such as a garden of flowers or thousands of colliding flags driven by stochastic wind forces, and provide a challenge for real-time computer animation given the complexity of computing realistic solid dynamics, collision processing and frictional contact for high-dimensional deforming objects. To avoid runtime simulation costs, compressed mesh animations are attractive for real-time hardware-accelerated playback of such complex mesh deformation phenomena, especially when computing dynamics on the fly would not be feasible [James and Twigg 2005]. Such examples are appealing candidates for interactive data-driven animation given (a) their often secondary role in animation, and (b) the high cost of physical simulation, especially using robust contact handling [Bridson et al. 2002]. Motion graph techniques [Kovar et al. 2002; Lee et al. 2002] provide the basic abstraction for data-driven animation, and rely on the fundamental editing/blending operation of splicing together motion clips (to loop or transition within the dataset). Unfortunately, simple blending

[1]Research completed while at Carnegie Mellon University.

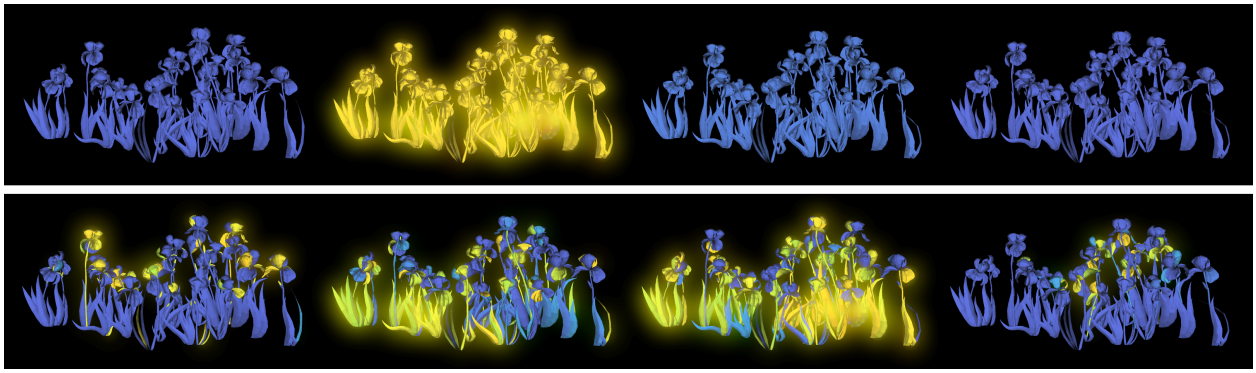


Fig. 1: **Asynchronous Transitions:** *(Top) Synchronized blending of scene motions from low to high wind levels causes all plant groups to unnaturally and suddenly start moving faster all at the same time. These spatially and temporally correlated transition errors tend to appear jerky and unnatural. Here frames are shown at 1/3 second intervals, with highlights indicating groups undergoing transition blends. (Bottom) Mesh Ensemble Motion Graphs allow scene components to transition at slightly different times, and thus avoid unnatural transition artifacts by (i) reducing the total transition error, and (ii) diffusing group transition errors in both space and time.*

can lead to transition artifacts since in high-dimensional deformation state spaces it can be exceedingly rare to find nearby states, and transitions can introduce undesirable artifacts with clear spatiotemporal correlation.

Data-driven animation techniques, such as "video textures" [Schödl et al. 2000] and "motion graphs" [Lee et al. 2002; Kovar et al. 2002], have proven to be very successful in recent years, permitting interactive play-back, looping, and control, of a wide range of rich, measured or simulated, multimedia datasets. A major challenge for data-driven methods is the curse of dimensionality: it can be difficult to animate high-dimensional phenomena, such as deformations, partly because of insufficient data. This limitation is particularly onerous for data-driven approaches to physically based animation of dynamic, deformable ensembles, such as a banner of flags colliding in the wind. In such cases, it can be hard to simulate and store a motion database that samples the space of dynamics adequately to permit data-driven animation without artifacts. Although affected by dimensionality concerns, video-based animation techniques have seen wonderful success for stochastic, textural phenomena, e.g., broadly called stationary datasets, using a variety of sophisticated, pixel-blending techniques [Kwatra et al. 2003; Bhat et al. 2004; Agarwala et al. 2005]. One common theme (that we also exploit for mesh ensembles) is to use space-time cuts to allow pixels to "transition" at different times. However, unlike video-based animation, transitions for high-dimensional mesh animations have the added constraint that the resulting 3D mesh geometry must have a valid geometric configuration that does not self-intersect unnaturally. Thus, although we find that asynchronous transitions can result in smoother dynamic motions that produce more good transitions for small datasets, they also exacerbate a serious and fundamental problem for data-driven mesh animation using interpolated motion: nonphysical mesh interpenetration between scene elements. These artifacts can be terrible for large motions in complex scenes, especially between contacting meshes, or meshes in close-proximity to the viewer, such as nearby plants colliding in a strong breeze. In short, interpenetrations must be addressed for high-quality, artifact-free, data-driven mesh animation.

In this paper, we introduce *Mesh Ensemble Motion Graphs* (MEMGs), a precomputation-based approach for data-driven animation of high-dimensional, deformable mesh ensembles that exploits asynchronous motion, and can address the fundamental issue of nonpenetration. First, we show how asynchronous transitions can enable the reuse of modest amounts of expensive, precomputed simulation data for high-quality mesh animation in real-time contexts. Asynchronous scene motions are parameterized using local-frame coordinates of nonoverlapping, mesh groups, e.g., of leaves and petals. Asynchronous mesh-group transitions allow effective diffusion of transition errors in space-time, which can partly masquerade as natural stochastic forcing (see Figure 1). The resulting data-driven MEMG models can be displayed efficiently, and are suitable for data-driven, physically based animation in interactive, virtual environments.

Second, we show how to impose physical constraints on transitions, such as nonpenetration constraints, using boolean equality constraints. We formulate the general MEMG transitions as solutions to integer programming problems, and provide a randomized algorithm for computing transitions for the important special case of nonpenetration constraints. We present several interactive examples that illustrate interesting interpenetration-free contact phenomena,

and interactive wind-level modification via transitions between datasets of differing wind conditions.

1.1 Problem Overview

We now give a high-level problem overview before delving into details. We consider mesh animations composed of G mesh groups, such as G distinct grass blades (see Figure 2). The animation's time series of deformed shape coordinates may be considered as a multi-track motion recording with G tracks. The core problem we address is how to splice these motion tracks such that the resulting motion transition is smooth and interpenetration free. We do not force all groups to transition at the exact same time instant, but rather permit each group to transition at some temporal offset. Given a candidate transition location, we denote the integer temporal offset of group k by τ_k frames, and assemble all G integer splice offsets into a *temporal offset vector*, $\tau = (\tau_1, \ldots, \tau_G)^T \in \Omega$ in the integer hypercube, $\Omega \equiv \{\tau | \tau \in \mathbb{Z}^G, \|\tau\|_\infty \leq \tau_{max}\}$ for some user-specified bound, τ_{max} (see Figure 3).

Fig. 2: *Scenes are hierarchically decomposed into mesh "groups" to support asynchronous transitions ("Large Irises" with 547 groups).*

Interpenetration artifacts are introduced easily when nearby mesh groups are animated asynchronously while in close proximity, and we seek to avoid these by specifying constraints on the transition optimization problem. Mathematically, we formulate the optimal temporal offset vector, τ, as the solution to a nonlinear integer programming problem [Nemhauser and Wolsey 1999] of the form,

$$\min_{\tau \in \Omega} \ \Phi(\tau) \quad \text{subject to} \quad \text{no mesh interpenetrations} \tag{1}$$

for a suitable data-driven transition cost function, $\Phi(\tau)$, that measures the "jump" in the scene position and velocity for a given τ-parameterized transition. Key challenges are that the nonpenetration constraints are expensive to even evaluate (using collision detection), and that many asynchronous transition programming problems encountered simply have no solution. We present a robust randomized algorithm that (a) avoids getting stuck on hard, or infeasible, transition programs, (b) exploits the geometric structure of nonpenetration constraints for intelligent local search, and (c) is efficient enough to compute thousands of asynchronous transitions for high-dimensional, deformable animations. In the following sections we show how to exploit the structure of the objective function Φ, and the geometric nonpenetration constraints, to yield an approximate, randomized algorithm that is efficient in practice.

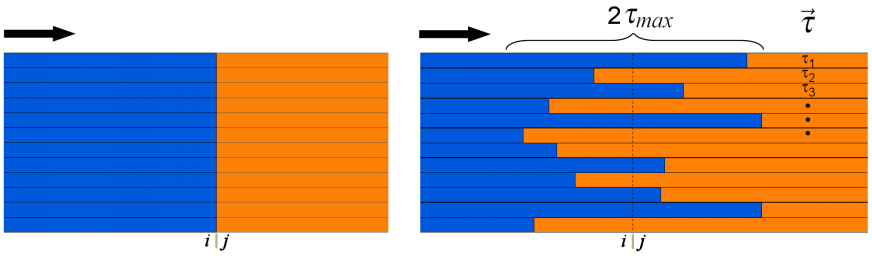


Fig. 3: **Multi-track motion splicing:** *Temporal transition offsets allow each motion group to splice its motion track with the destination clip at slightly offset times: (Left) Synchronous transitions have all groups transition at the same time ($\tau = 0$), whereas (Right) asynchronous MEMG transitions ($\tau \neq 0$) allow different transition blend times for each group. Determining good τ that also avoid scene interpenetrations is the difficulty our algorithm addresses.*

1.2 Related Work

Mesh Ensemble Motion Graphs (MEMGs) are related to data-driven animation techniques, such as "Video Textures" [Schödl et al. 2000; Schödl and Essa 2002; Kwatra et al. 2003; Bhat et al. 2004; Agarwala et al. 2005],

and "motion graphs" for character animation [Lee et al. 2002; Kovar et al. 2002; Arikan and Forsyth 2002]. We address computing interpenetration-free transitions in high-dimensional, 3D mesh animation datasets, but the resulting transitions can be used within existing graph-based frameworks discussed in prior work. Similar in spirit to recent video-based approaches [Kwatra et al. 2003; Bhat et al. 2004; Agarwala et al. 2005], we use asynchronous transitions to reduce transition errors, and diffuse artifacts in space and time. In contrast to video-based methods that must address finding error-minimizing transitions in huge video datasets [Kwatra et al. 2003; Agarwala et al. 2005], we can exploit the 3D scene parameterization to simplify this task. Additionally, asynchronous transitions for 3D meshes necessitate the introduction of difficult nonpenetration constraints, whereas most motion graph approaches can safely ignore self-collisions during transitions; such problems do not arise directly in video-based animation, however interacting or overlapping structures can be a problem in some cases, such as when they have different frequencies or nonstationary behavior [Agarwala et al. 2005].

Our approach is related to motion graph techniques for animating crowds, such as [Lai et al. 2005; Sung et al. 2005], where the motion of a high-DOF system must be planned, while characters are animated using motion graphs subject to collision avoidance. One notable difference is that we do not coordinate plans of multiple interacting motion-graph objects, but rather our system is driven by a single (asynchronous) motion graph with very high dimensionality. The name "Mesh Ensemble Motion Graph" may suggest that our technique could be adapted to motion graphs for humanoid characters, but this is not our intention. Fully asynchronous transitions would appear undesirable given that human movement is not stochastic, and has a high degree of purposeful coordination [Safonova et al. 2004].

Our use of optimization to compute interpenetration-free transitions is related to collision-free path planning in robotics and related areas [Latombe 1991; Kavraki et al. 1996; Choset et al. 2005; Moll and Kavraki 2006]. One major difference is that we control motion using abstract transition parameters (τ) instead of more traditional actuation of kinematic structures. Also, we are planning self-collision-free motions for high-DOF deformable objects, whereas motions are more commonly planned for articulated robot manipulators in static environments, although plans for deformable objects also exist (e.g., see [Moll and Kavraki 2006]).

Regarding data-driven mesh animation, Perbet and Cani [2001] use procedural animation of a single, physics-based, deformable, grass mesh primitive (with texture-based level of detail), to animate massive grass prairies at interactive rates. Although we also demonstrate instanced mesh primitives (including grass), our work is complementary in that we address how to support data-driven animation for *interacting* mesh ensembles (instead of a single mesh), by addressing the fundamental problem of nonpenetration.

James and Fatahalian [2003] also address data-driven animation of deformable scenes, but limit their focus to specialized impulse responses that enable transitions between motion clips, and can be played (without looping) until the object comes to rest. Highly discontinuous, impulse-based transitions allow physical disturbances to mask data-driven transition errors (typical of small simulation datasets), whereas we rely on asynchronous transitions to diffuse errors in time. In this paper we address high-dimensional motions (tens of thousands of DOF), whereas they used datasets with low-dimensional motion (tens of DOF) that resulted from the use of rigid-motion impulses. Finally, we explore asynchronous transitions and address related nonpenetration issues, whereas impulse responses are inherently synchronized.

Constraint programming concerns the problem of finding a *feasible point*, any point, that satisfies a list of constraints. Such a problem is often referred to as a constraint satisfaction problem (CSP) [Apt 2003], and they are very common in AI, combinatorial optimization, numerical analysis, finance, etc. Although not heavily used in computer graphics, constraint programming actually had early roots in Sutherland's Sketchpad system [1963]. Our nonpenetration constraint programming algorithm exploits the particular structure of contact constraints to provide an efficient iterative algorithm, and this is related to rigid body contact solvers that exploit structure in nonpenetration, linear complementarity, and other constraints for efficiency [Baraff 1994; Kaufman et al. 2005]. The major difficulty in our problem is that our constraints are highly nonlinear, unlike linear complementarity conditions, and their values are only defined implicitly by costly collision detection. For such difficult CSPs, the best approach is often to just "try and see," and sophisticated random sampling approaches are often used to perform a combination of global and local search. Of particular relevance, very successful techniques have been developed for solving a difficult and important class of NP-Complete CSPs known as k-satisfiability (k-SAT) problems [Garey and Johnson 1979]; briefly, these problems involve constraints on boolean variables, where each constraint or clause involves no more than k boolean variables. A variety

of successful local search methods have been developed that iteratively modify variables associated with unsatisfied constraints, so as to reduce their overall number [Gomes and Selman 2003]. As we will discuss later, our random-sampling algorithm is analogous to these approaches, and in particular the k-SAT solver known as WalkSAT [Selman et al. 1996], however since our nonpenetration constraints involve collision detection and can be orders of magnitude more expensive to evaluate than boolean clauses, our algorithm takes a significantly different approach that avoids the explicit evaluation of constraints at each iteration.

2. ASYNCHRONOUS TRANSITIONS

We will assume that each scene mesh ensemble consists of G non-overlapping mesh groups, e.g., G blades of grass, specified by the user (more on this later). To permit each group to transition between motion clips at different times, the deformed shape of each mesh group is defined in a local frame of reference. To accommodate interesting structures, we permit groups to be arranged in tree-like kinematic structures; hierarchies with loops introduce synchronization constraints on τ and are not discussed for simplicity. The following section describes the details of the reduced-coordinate, tree-structured kinematics that we exploit throughout.

2.1 Kinematics of mesh ensembles

Our MEMG algorithm can be applied to a range of motion datasets, however we briefly describe the particular kinematics used in our examples. The shape of each group, $k = 1 \ldots G$, is defined in a local coordinate frame of reference, with groups arranged in a tree-structured kinematic hierarchy. The deformed position vector of the k^{th} group's v_k vertices is

$$\mathsf{T}_{Wk}(\mathsf{p}_k + \mathsf{u}_k) = \mathsf{T}_{Wk}(\mathsf{p}_k + \mathsf{U}_k \mathsf{q}_k) \ \in \mathbb{R}^{3v_k}, \tag{2}$$

with the rigid-body transformation of this group to the world coordinate frame denoted by the operator T_{Wk}. The *local-frame* displacement field is parameterized by $\mathsf{u}_k = \mathsf{U}_k \mathsf{q}_k$ for a low-rank displacement matrix, $\mathsf{U}_k \in \mathbb{R}^{3v_k \times r_k}$ for a group with v_k vertices, and rank r_k. Here $\mathsf{q}_k = \mathsf{q}_k(t) \in \mathbb{R}^{r_k}$ is the reduced-coordinate vector (SVD thresholded at 10^{-4}) which describes the deformation in a local frame, and t denotes an integer time index for a uniform animation time step. The concatenation of all reduced displacement coordinates is the *ensemble displacement coordinate*, $\mathsf{q} = (\mathsf{q}_1^T, \ldots, \mathsf{q}_G^T)^T \in \mathbb{R}^r$ where the total rank of the shape model is $r \equiv r_1 + \ldots + r_G$. The deformed scene's shape is then described by q, along with frames of reference, $\{\mathsf{T}_{Wk}\}_{k=1\ldots G}$. Similarly, the *ensemble velocity coordinate*, $\dot{\mathsf{q}} \in \mathbb{R}^r$, describes the rate of change of local-frame shape. During MEMG transitions, we linearly blend q_k values over $\tau_{blend} = 10$ frames using the τ_k transition offset. The transition frames T_{Wk} are computed hierarchically, with blending of each local transform's transition time determined by the appropriate group's τ component, analogous to interpolating a rigid character's motion during an asynchronously blended transition of joint angle data.

Finally, in our implementation dynamic coordinate frames are determined by deformations of parent groups. Consequently, if two shapes have the same local-frame deformation coordinates, $\mathsf{q} = \mathsf{q}'$, then they have the same frames as well. We will exploit this when computing shape-shape distances in our transition cost function.

2.2 Transition Cost Function

Hereafter, we can encapsulate all state details by assuming that for each group k, we simply have a transition cost metric, $D_k(i, j) \in \mathbb{R}$, that measures group k's additive cost for an $i \rightarrow j$ transition, where i and j are integer time indices corresponding to the input time series[2]. In our implementation we use the squared state-space distance metric from [James and Fatahalian 2003],

$$D_k(i, j) = \|\mathsf{q}_k^{(i)} - \mathsf{q}_k^{(j)}\|_2^2 \ + \ \beta^2 \|\dot{\mathsf{q}}_k^{(i)} - \dot{\mathsf{q}}_k^{(j)}\|_2^2. \tag{3}$$

We define the asynchronous transition cost for a given temporal offset vector $\tau \in \Omega$, by

$$\Phi(\tau;\, i, j) = \Phi(\tau) = \textstyle\sum_{k=1}^{G} \Phi_k(\tau_k), \tag{4}$$

[2]For simplicity we will describe transitions within a time series, but of course transitions will be computed between different time series as well.

where $\Phi_k(\tau_k) \equiv D_k(i+\tau_k, j+\tau_k)$; we will drop the (i,j) arguments when they are clear. This is a generalization of the synchronous distance metric in [James and Fatahalian 2003], which corresponds to $\Phi(0) = \Phi(0;i,j)$ (or $G=1$). The exact definition of the scene shape during the asynchronous transition results from tree-structured hierarchical interpolation of local group transforms, and interpolated shape parameters at group-specific transition times.

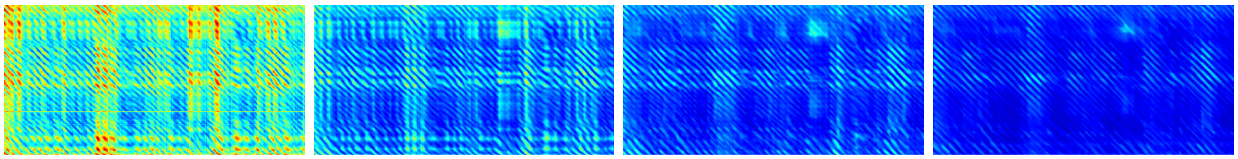
2.3 On finding minimum of $\Phi(\tau)$

In the absence of nonpenetration constraints, it is easy to perform unconstrained minimization of the objective function, $\Phi(\tau)$, $\tau \in \Omega$, since it is a *separable function* [Nocedal and Wright 1999] expressible as the sum of simple univariate element functions, $\Phi_k(\tau_k)$. Separability implies that element functions can be optimized independently, since

$$\tau^* = \arg\min \Phi(\tau) \quad \Leftrightarrow \quad \tau_k^* = \arg\min \Phi_k(\tau_k). \tag{5}$$

Consequently, unconstrained minimization is very fast, involving only a scan to find the minimum of the $\Phi_k(\tau_k)$ data vector. Unfortunately, nonpenetration constraints couple the integer variables, making minimization substantially harder. Nevertheless, we will exploit unconstrained minimization of the separable objective functions to find a good initial guess for our randomized hill-climbing algorithm. Also, we exploit separability during random sampling, to importance sample each τ_k variable independently.

In practice, we will seek to minimize transition error for a bounded range of τ values: $\|\tau\|_\infty \leq \tau_{max}$. In practice, even modestly increasing τ_{max} can quickly reduce transition distances (see Figure 4).



$\tau_{max} = 0$ frames $\tau_{max} = 5$ frames $\tau_{max} = 10$ frames $\tau_{max} = 20$ frames

Fig. 4: **MEMG transition distance** $\Phi(\tau;\, i,j)$ *exhibits quick decay with moderate increases in* τ_{max} *(units of # frames at 30 FPS). Horizontal and vertical axes represent a range of* i *and* j *values, respectively. A tenfold reduction in* Φ *is observed for some transitions at just* $\tau_{max} = 20$*. Unfortunately, these transitions can often exhibit major interpenetration artifacts. Dataset is for irises at moderate wind levels.*

2.4 Motion Graph Construction

Given a transition cost function, a motion graph could be constructed similar to approaches from character animation [Kovar et al. 2002; Lee et al. 2002]. However, anticipating the random-sampling algorithm in following sections, for each wind level we generate a sparse set of candidate transitions then extract the largest strongly connected component. Transitions between wind levels are then generated by sampling transitions between each of the strongly connected components. In either case, transitions are generated to ensure that each animation interval, e.g., of one second, has at least one transition to elsewhere in the same wind level, and other wind levels. As described later, random sampling is used to introduce variability.

3. CONSTRAINING ASYNCHRONOUS TRANSITIONS

The asynchronous transitions generated in the previous section can work well for very small deformations, however the optimization for τ is ignorant of contact physics and the geometric configuration space governing the underlying motion. As a result, transitions can introduce interpenetrations and other artifacts not present in the original motion (see Figure 5), and will be unsuitable for certain applications. We now show how to put some of the physics back in by adding constraints to the integer optimization that determines τ. For the remainder of the paper we address the important case of geometric nonpenetration constraints.

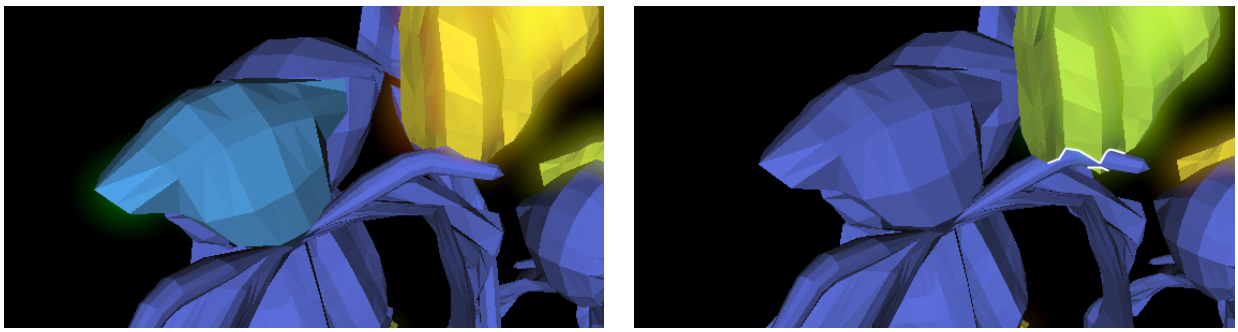


Fig. 5: A typical interpenetration during an asynchronous transition: *(Left) an asynchronous transition begins and leads to (Right) a petal passing through an adjacent iris's pistil without the contact being resolved. Equality constraints can be used to discourage such artifacts, which ranged from zero to hundreds per transition in our datasets. (Yellow glow indicates a group undergoing a transition blend.)*

3.1 MEMG Transitions as Integer Programs

Finding the best temporal offset, τ, for a given (i, j) involves integer optimization, and we formulate the MEMG transition as the solution to a nonlinear integer program involving geometric nonpenetration constraints between each pair of groups, $k \neq l$. In practice, to limit search costs and provide expedient transitions, we bound the range of each integer component of τ to lie in $[-\tau_{max}, \tau_{max}]$. Mathematically, this results in the optimal temporal offset vector, τ, being the solution to our governing MEMG integer program:

$$\begin{array}{l} \underline{\text{MEMG TRANSITION INTEGER PROGRAM}} \\ \qquad \min_{\tau \in \Omega} \ \Phi(\tau) \\ \text{subject to} \\ \qquad L^t_{kl}(\tau) = 0, \quad \forall t \in \mathscr{T}, \ \forall k \neq l. \end{array} \tag{6}$$

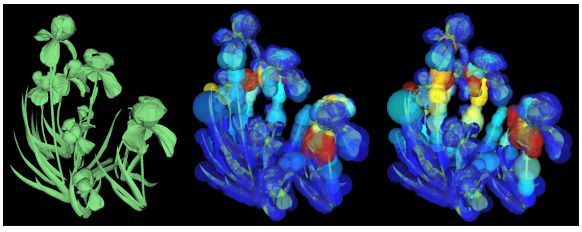
where $L^t_{kl} = 0$ are nonpenetration equality constraints (discussed hereafter), and the time steps of the asynchronous transition are denoted by $\mathscr{T} \equiv \{-(\tau_{max} + \tau_{blend}), \ldots, (\tau_{max} + \tau_{blend})\}$, where τ_{blend} are the number of frames used for linear blending ($\tau_{blend} = 10$ frames at 30 FPS in our examples). The remainder of this paper concerns the efficient approximation of the solutions to this integer program.

3.2 Ensuring nonpenetration using boolean equality constraints

Nonpenetration constraints exist between each pair of groups k and l ($k \neq l$, and $k, l \in 1, \ldots, G$), at each time index of the blended transition frames, $t \in \mathscr{T}$, and have been written as

$$L^t_{kl}(\tau) = 0 \tag{7}$$

where L^t_{kl} is a measure of interpenetration. For our fast solver algorithm in Section 4, the L^t_{kl} need only be a boolean indicator function that returns a nonzero value if the meshes k and l overlap at transition frame t, and zero otherwise.



Evaluating nonpenetration constraints involves costly collision detection operations that are the algorithm bottleneck. Fortunately, we can exploit the reduced-coordinate parameterization of the scene shape and use output-sensitive collision detection approaches for deformable geometry, based on Bounded Deformation Trees [James and Pai 2004]. Although constraints are most heavily evaluated in a pairwise local manner in our algorithm, a uniform subdivision is used to accelerate the broad-phase part of global tests. While our current constraint solver implementation checks for overlaps at discrete times, extensions to interval-based collision tests are straightforward.

3.3 On the existence of transitions

For interesting scenes involving mesh groups moving in close proximity, merely finding a feasible τ that satisfies the nonpenetration constraints can be hard. Given the combinatorially many choices for τ (there are $O(\tau_{max}^{G})$ possibilities), in general it seems impossible to even know if the problem can be solved in polynomial time, and performing collision detection to test constraints is expensive. Worse still, one challenge intrinsic to the asynchronous mesh animation problem is that the existence of solutions to (6) is not even guaranteed: there may not even exist a feasible point, τ, that satisfies all nonpenetration constraints. These infeasible transition programs, where it's not possible to transition without introducing interpenetration, correspond to the classic saying, "you can't get there from here" (see Figure 6). We address nonexistence as follows. Given a single i frame, we sample j to obtain multiple (i, j) integer programs, with some having no solution, but many having solutions, with the latter being a mix of easy and hard programs. To avoid not getting stuck on hard or infeasible problems, we use random sampling with restarts, so that the solver aborts solving (i, j) after a fixed amount of time, and changes to another (i, j') problem. Fortunately, for many datasets the majority of the transition programs we encountered are solvable, or can be solved with sufficient data. In the following section (§4) we present an algorithm that exploits key problem structure for speed.

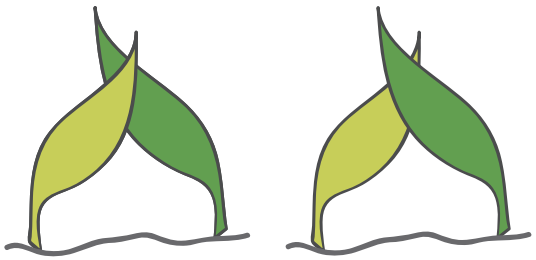


Fig. 6: **"Can't get there from here":** *Mesh shapes can appear close in state space (based on vertex positions and velocities), but be far in configuration space where interpenetrations are disallowed.*

3.4 Discussion

Mathematically we are using boolean equality constraints to identify when something "bad" goes wrong during a transition so that τ may be modified to correct it. If the mathematical program is feasible, then the numerical algorithm given in the next section stands a chance of finding such a transition. Showing how to impose such constraints on asynchronous transitions for data-driven animation is our main research contribution. We have considered nonpenetration constraints as a prototypical example of a equality constraint, since interpenetrations were the most obvious artifact we encountered. However, boolean equality constraints can be used to model a wide range of other behaviors and contact conditions. For example, although it was beyond the scope of our study, one could also suggest contact compatibility conditions to encourage contact forces to be nonzero when contact is not present, e.g., since larger contact forces are less well masked by stochastic wind forces.

Lastly, nonpenetration constraints should not be misunderstood to mean that contact can not occur during the transition. Contacts occur in our input motion without interpenetration, and these can also occur in the transition, along with blended motions that avoid interpenetration, e.g., sliding leaves in our banana plant. Furthermore, the equality constraint can be used to specify other types of feasible geometric contact configurations, such as allowing small interpenetrations in some bounded measure if so desired, e.g., to accomodate difficult contact configurations.

4. NUMERICAL ALGORITHM

Although the basic iterative constraint satisfaction approach is conceptually simple there are many details for efficient implementation. We will first provide a high-level overview of our MEMG transition solver algorithm (§4.1), then proceed with a detailed description and pseudo-code implementation to ensure a concise and implementable description of our algorithm.

4.1 Brief Overview of Algorithm

We solve (6) by performing global minimization of Φ over feasible point samples (§4.2), with samples generated by our nonpenetration constraint solver (§4.3), with the latter being the most complicated part. Feasible points are determined by starting with a unconstrained minimum, then a full collision detection pass is run on the transition frames to determine unsatisfied constraints. Iterative importance-based local search is used to modify τ components associated with unsatisifed constraints, while all other components are left at their global minimum values. Local search continues until either no more interpenetrations can be found, or (a) the iteration aborts due to a suboptimal Φ value, or (b) the maximum allotted solver time is exceeded. Throughout, judicious use of collision detection queries are used to avoid computational bottlenecks. A visual summary of the algorithm is given in Figure 8, and we now described it in detail.

4.2 Global Minimization

Global minimization is performed by minimizing over randomly sampled feasible points as follows. Given i, and set of candidate frames to transition to, J, our algorithm finds $j \in J$ and τ that (approximately) minimizes $\Phi(\tau) = \Phi(\tau;\, i, j)$ subject to the constraints in (6). The entire algorithm of our paper is decomposed into a number of simpler functions for clarity, but the main, user-level function performs this simple global minimization:

```
COMPUTE-MEMG-TRANSITION(i, J, N_samples)
 1  φ_min ← ∞
 2  X ← NIL
 3  for n ← 1 to N_samples
 4     j ← importance sample j ∈ J using 1/Φ(0; i, j)
 5     φ_cutoff ← α φ_min
 6     (φ, τ) ← SATISFY-MEMG-CSP(i, j, φ_cutoff)
 7     if ((φ, τ) ≠ NIL)  and  (φ < φ_min)
 8        then φ_min ← φ
 9             X ← (i, j, τ)
10  return X
```
(8)

This function simply determines the minimum ϕ sample of at most $N_{samples}$ feasible points, ignoring NIL values. All difficulty is contained in the SATISFY-MEMG-CSP() function, which finds feasible points to the MEMG constraint satisfaction problem (CSP), or failing returns NIL. The minimization uses importance-based random sampling (line 4) to find $j \in J$ that tend to have small $\Phi(0;\, i, j)$ values; these small synchronized transition distances are easy to compute, and also tend to be good locations to look for asynchronous transitions. While this randomness does not aid us in finding the j that results in the global minimum, introducing randomness into the found (i, j) pair is desirable, since in many of our datasets, we find that near-optimal (unconstrained) ϕ values are plentiful, but it is constraint satisfaction that is hard. Finally, to avoid spending time finding feasible points with similar or suboptimal ϕ values, we use a cutoff (lines 5-6) slightly smaller than the best known minimum, $\phi_{cutoff} = \alpha\phi_{min}$, where $\alpha \in (0, 1]$; our examples used $\alpha = 0.97$.

4.3 Nonpenetration Constraint Programming

We now describe the computation performed by the core function SATISFY-MEMG-CSP(), that given an (i, j) pair finds a feasible solution τ (or NIL if none can be found) to the MEMG Transition Integer Program (6). To find a feasible point, we start with the solution to the unconstrained minimization and use random sampling and local search techniques to repair unsatisfied nonpenetration constraints. A key insight that makes our problem solvable for complex examples is that randomized constraint programming can be done efficiently by (a) exploiting the sparse functional structure of the geometric nonpenetration constraints to reduce constraint checks during local search, and (b) using output-sensitive collision detection algorithms to evaluate the constraints.

4.3.1 *Structure of Nonpenetration Constraints.* The main contribution of our optimization algorithm is the efficient handling of numerous nonpenetration constraints. We exploit the fact that each nonpenetration constraint, $L^t_{kl}(\tau)$, can

only depend on a minimal subset of variables, or *kinematic clique*, $\aleph_{kl}$:

$$L^t_{kl}(\tau) = L^t_{kl}(\tau') \text{ if } \tau_i = \tau'_i \text{ for each } i \in \aleph_{kl}. \tag{9}$$

The exact definition of the kinematic clique, $\aleph_{kl}$, is implied by scene kinematics, and is simply an enumeration of groups in a kinematic chain connecting k and l for which changes in τ components can affect the value of L^t_{kl} (see illustration in Figure 7). In our algorithm we will exploit the corollary that if L^t_{kl} is nonzero, then only changes to $\tau_i : i \in \aleph_{kl}$ need be considered to remove the interpenetration.

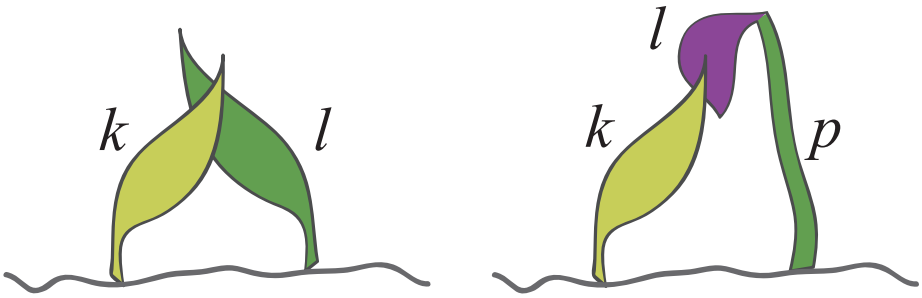
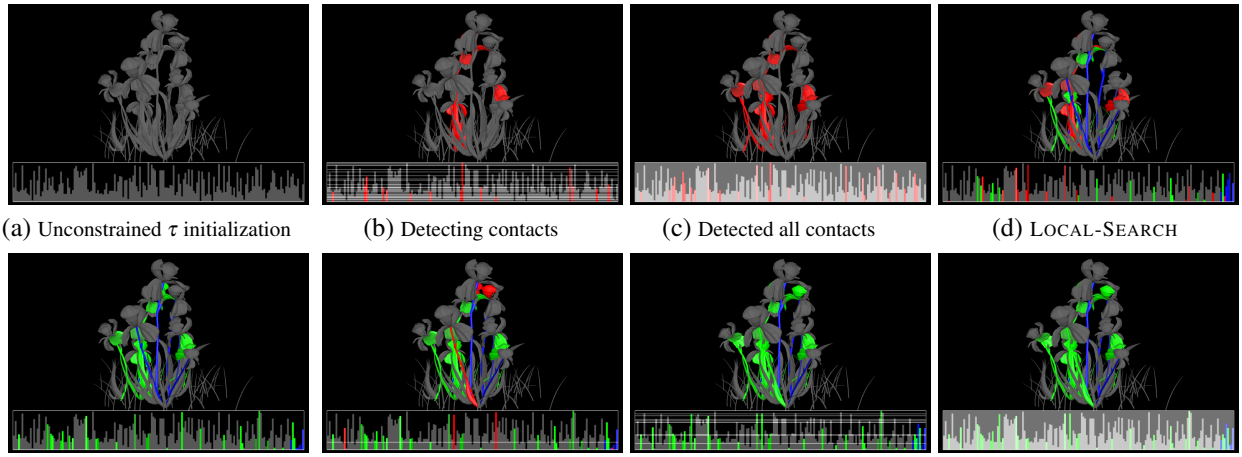


Fig. 7: **Illustration of the kinematic clique, $\aleph_{kl}$:** *(Left) In the absence of a kinematic hierarchy, transition-induced contact between groups k and l can only be removed by adjusting τ components of the simple kinematic clique, $\aleph_{kl} = \{k,l\}$. (Right) More generally, the kinematic clique is composed of the chain of mesh groups connecting the contactees, k and l. The example shown also involves l's parent group, p, so that $\aleph_{kl} = \{k,l,p\}$.*



(a) Unconstrained τ initialization (b) Detecting contacts (c) Detected all contacts (d) LOCAL-SEARCH

(e) Made $B = \emptyset$ (f) DETECT-NEW-CONTACTS (g) Verifying no contacts exist (h) Success!

Fig. 8: **Overview of iterative nonpenetration constraint satisfaction** *performed by* SATISFY-MEMG-CSP() *in (11). Iterative modification of τ (illustrated as G vertical bars) is used to remove all interpenetrations (or time-out) as follows: (a) τ is initialized to the optimal unconstrained minimum, $\tau = \arg\min \Phi$ in line 2 of (11) (denoted in gray); (b-c) collision detection is performed at each time step (each test represented as a white horizontal bar on τ) to determine initial bad contacts, $B = B_{init}$ (red highlights), and the initial contact set, $C = B$ (lines 7-8 of (11)); (d)* LOCAL-SEARCH() *is repeatedly invoked (lines 9-12 of (11)) modifying τ in order to (e) remove all bad (red) contacts, $B = \emptyset$; at this point only known contacts, C (in green), are guaranteed to be collision free, so (f) new contacts are detected at random time instants (lines 9-10 of (12)), incrementing B and C, and* LOCAL-SEARCH() *is invoked to fix them as they arise. (h) The algorithm terminates successfully when no new contacts exist. In practice, the majority of CSPs will be aborted early due to suboptimal ϕ values, or timed-out if the problem is too hard or infeasible. Blue indicates noncontacting stems which are in active kinematic cliques, e.g., their petals are in C.*

4.3.2 *Contact Constraint Sets.* Merely detecting if any of the $O(G^2|\mathscr{T}|)$ nonpenetration constraints are unsatisfied is an expensive collision detection query, even using optimized broad and narrow phase strategies. Fortunately, in practice often only a small fraction of nonpenetration constraints complicate the CSP solution process. By tracking observed group-group interpenetrations (which we call *contacts*), and exploiting their kinematic structure, we can

speed up our algorithm by using fast collision checks on the known contact set. These quick tests provide a necessary (but not sufficient) condition for interpenetration removal that can be used before resorting to costly global nonpenetration tests.

We maintain three types of contact constraint sets for each (i, j) transition computation, SATISFY-MEMG-CSP$(i,\ j,\ \phi_{cutoff})$. Each set is composed of a group-group contact pair, (k,l) $(k<l)$ (encoded as an integer), which we will hereafter refer to as "contacts." The contact constraint sets are:

Contact set, C The contact set, C, is a running set of all encountered bad contact pairs for this (i, j) program. It serves as a list of "known contact hot spots" that can be tested orders-of-magnitude faster than a global collision test, and is used to quickly estimate how the number of unsatisfied constraints will change during local search.

Bad contacts, B The current list of known bad contact pairs, B, is computed by collision detection, and used to expand the contact set, C. Note that $B \subset C$. The set B comprises the frontier of local search activities, and in general, B is only a subset of all existing scene interpenetrations that may have been introduced by adjustments to τ. When B is empty, this MEMG constraint satisfaction problem is solved.c

Kinematically affected contacts, A_* Finally, we also track G sets, $A_k,\ k=1\ldots G$ to describe which contacts, $A_k \subset C$, may be kinematically affected by changes to the k^{th} variable, τ_k. This is used extensively during local search to quickly check if the number of known bad contacts has increased, without resorting to testing all of C, let alone perform a global collision test! These G sets will be denoted as A_* for brevity.

The algorithm to update C and A_* given some newly detected contact pairs, B, is simply

```
UPDATE-CONTACTS(C, B, A*)
1  C ← C ∪ B
2  for (m,n) ∈ B
3     for k ∈ ℵ_mn
4        A_k ← A_k ∪ (m,n)
```
(10)

4.3.3 *Iterative Nonpenetration Constraint Satisfaction.* We use an iterative local search algorithm to detect and resolve unsatisfied nonpenetration constraints (see overview in Figure 8). The main algorithm (SATISFY-MEMG-CSP() in (11)) proceeds by first initializing (ϕ, τ) to the simple unconstrained minimum (line 2), and the contact sets (B, C, A_*) to contacts found during a full scan of the τ-transition frames (lines 5-8). Iterative constraint satisfaction is then used (lines 9-13) to incrementally find new contacts, and modify τ to ultimately reduce the number of bad contacts, B, until no more bad contacts can be found ($B=\emptyset$), and τ is a feasible point to the CSP.

```
SATISFY-MEMG-CSP(i, j, φ_cutoff)
 1  t_start ← TIME()
 2  (φ,τ) ← unconstrained minimization of Φ
 3  if φ > φ_cutoff
 4     then return NIL
 5  C ← ∅
 6  A_k ← ∅ ∀k
 7  B ← detect all contacts in τ-transition (EXPENSIVE)
 8  UPDATE-CONTACTS(C,B,A*)
 9  while B ≠ ∅
10     if (φ > φ_cutoff) or (TIME() − t_start > t_cutoff)
11        then return NIL
12     (φ,τ) ← LOCAL-SEARCH(C,B,A*,φ,τ)
13  return (φ,τ)
```
(11)

If the algorithm reaches line 13, the constraints have been satisfied, and a feasible solution point is returned. While this function is designed to detect and resolve all bad contacts using LOCAL-SEARCH (discussed in the next section), it also pessimistically aborts this (i, j) CSP and returns NIL for one of two reasons:

(1) *Suboptimal:* NIL is returned if the unconstrained minimum ϕ (line 4) or current ϕ (line 10) exceeds ϕ_{cutoff}. Note that aborting on line 4 avoids all collision detection queries.

(2) *Time-out:* LOCAL-SEARCH is invoked until success (lines 9-12), unless ϕ is suboptimal, or the elapsed process time exceeds a cutoff, t_{cutoff}. The latter is important to avoid getting stuck on a hard or infeasible CSP.

We use a time-out instead of aborting after a fixed number of iterations (like popular hill-climbing algorithms for k-SAT), since we found time is a more robust way to handle the highly variable collision detection costs, and iteration counts, of each dataset and (i,j) program. In our implementation, we manually specify t_{cutoff} as a multiple of the collision detection time, t_{CD}, to process all group collisions in the τ-transition frames (timing of line 7 of SATISFY-MEMG-CSP), since t_{CD} is related to the overall solution complexity (an issue discussed in Section 4.3.7).

4.3.4 *Local Search.* The procedure LOCAL-SEARCH() selects a component of τ that affects an unsatisified constraint in B, then randomly modifies it provided it doesn't increase the number of unsatisfied active constraints, $|B|$. In other words, LOCAL-SEARCH() modifies a single variable in a bad constraint $(\phi,\tau,B) \rightarrow (\phi',\tau',B')$ iff $|B'| \leq |B|$. LOCAL-SEARCH() with also find any new bad contacts to refill B if it is emptied, such that when LOCAL-SEARCH() returns with $|B| = \emptyset$, the CSP solver SATISFY-MEMG-CSP() can terminate and return a guaranteed-feasible point. The details are as follows:

```
LOCAL-SEARCH(C, B, A*, φ, τ)
 1  k ← SELECT-VARIABLE(B,A*)
 2  (φ′,τ′) ← importance sample τ_k using 1/Φ_k
 3  B_k ← B ∩ A_k
 4  B′_k ← GET-BAD-CONTACTS(A_k, τ′)
 5  if |B′_k| ≤ |B_k|
 6    then (φ,τ) ← (φ′,τ′)
 7         B ← (B − B_k) + B′_k
 8         if B = ∅
 9           then B ← DETECT-NEW-CONTACTS(C,τ)
10                UPDATE-CONTACTS(C,B,A*)
11  return (φ,τ)
```
(12)

Random variable selection (line 1) is used to determine which τ_k to consider modifying (discussed in next section). The variable is randomly sampled (line 2) using a $1/\Phi_k$ importance distribution to bias the sample to have small $\Phi_k(\tau_k)$ values. Collision detection is used to check which of the affected contacts, A_k, are bad (line 4). If the change does not increase the bad contacts (line 5), then the change to τ is committed (lines 6-7). As in WalkSAT [Selman et al. 1996], allowing changes that do not change the total number of unsatisfied constraints ($|B'_k| = |B_k|$) helps avoid getting stuck in local minima, allowing the sampling process to walk around in the high-dimensional infeasible solution space toward a feasible point. Finally, if LOCAL-SEARCH empties B in line 7, global collision detection attempts to detect some new contacts (lines 8-10), but if no new contacts exist B remains empty and SATISFY-MEMG-CSP() terminates successfully.

4.3.5 *Random Variable Selection.* Careful selection of the variable to sample (line 1 of (12) is critical for good performance. Our strategy is reminiscent of the WalkSAT algorithm [Selman et al. 1996], wherein a two-step random process is employed: first an unsatisfied constraint (k,l) is picked randomly from B, followed next by a variable within that constraint, $m \in \aleph_{kl}$, with the latter determined at random, or by a heuristic. In our implementation, we use importance sampling to select the sample variable, m, in a simple way that tends to reduce the number of collision pairs, $|A_m|$, that must be tested during local search:

```
SELECT-VARIABLE(B ≠ ∅, A*)
1  (k,l) ← random collision pair ∈ B
2  return variable m ∈ ℵ_kl with probability p_m ∝ C^(−|A_m|)
```
(13)

Here $C \geq 1$, and we used $C = 1.1$ in all examples, except for the flag banner where we used $C = 2$ to avoid sampling the rope (more on this later).

4.3.6 *Contact Detection.* Collision detection performance determines the expected speed of LOCAL-SEARCH(). Therefore we are careful to only perform necessary checks on the active contact pairs, C, to ascertain whether $|B|$ has

increased. Two functions help us do this, with the first being:

$$\left.\begin{array}{l}\text{GET-BAD-CONTACTS}(S,\ \tau)\\ 1\quad S_{bad} \leftarrow \emptyset\\ 2\quad S_{left} \leftarrow S\\ 3\quad \textbf{for } frame \in transitionFrames(\tau)\\ 4\qquad P \leftarrow \text{ detect } S_{left} \text{ collisions in } frame\\ 5\qquad S_{bad} \leftarrow S_{bad} \cup P\\ 6\qquad S_{left} \leftarrow S_{left} - P\\ 7\quad \textbf{return } S_{bad}\end{array}\right] \tag{14}$$

which simply returns the bad contacts in a given set, S, by checking these contacts in all transition frames. To save time, it avoids retesting known bad pairs, S_{bad}, and only tests the remaining pairs, S_{left}. The second collision detection function determines new contacts (if they exist) by performing a randomized global test:

$$\left.\begin{array}{l}\text{DETECT-NEW-CONTACTS}(C,\ \tau)\\ 1\quad \textbf{for } frame \in randomizedTransitionFrames(\tau)\\ 2\qquad D \leftarrow \text{ detect new contacts in } frame \text{ (ignoring } C)\\ 3\qquad \textbf{if } D \neq \emptyset\\ 4\qquad\quad \textbf{then return } D\\ 5\quad \textbf{return } \emptyset\end{array}\right] \tag{15}$$

This function avoids testing the entire animation (which is very costly), by returning contacts found in the first interpenetrating transition frame encountered during *randomized* frame traversal. Note that previously tested contact hotspots, C, can be ignored for speed. If no contacts are returned (line 5), the transition is globally interpenetration free, and a feasible point has been found.

4.3.7 *Complexity.* By virtue of the time-out after $t_{cutoff} = \eta t_{CD}$, the running time of the algorithm is bounded by a constant times the time to perform $2N_{samples}$ collision detection passes on the dynamic transition geometry (see Table I), or $O(N_{samples}\eta t_{CD})$. Unfortunately, the difficulty of the integer programs encountered is strongly problem dependent, and may even be infeasible; even with sufficiently large η, the algorithm may not be able produce any feasible solutions. In practice, our statistics indicate that feasible integer programs are typically solved in time proportional to the transition's collision detection time, t_{CD} (see Table I).

The memory requirements of the preprocess algorithm are only dominated by the underlying simulation dataset, and basic collision detection structures. The dominant memory cost for computed transitions, is only the τ vector, which is encoded as G bytes. Finally, the runtime MEMG memory footprint is determined by the underlying animation data, which in our implementation is compressed using principal component analysis (in local transform frames) and quantized.

Accelerating collision detection would have a direct impact on the speed of our algorithm. The performance of the integer programming algorithm can be improved in several ways. The most obvious way is to accelerate the collision processing kernel used. The BD-Tree collision bounds may be too conservative for close-proximity collision tests, such as in dense foliage, and alternate approaches may perform better in collision "hot spots." Many of the collision tests may be redundant since importance sampling may re-testing $L^t_{kl}(\tau)$ for τ values that are identical as far as the kinematic clique, $\aleph_{kl}$, is concerned. Overall, it is probably not unreasonable to expect that an order of magnitude speedup in the integer programming preprocess is possible with optimization.

5. RESULTS

To illustrate our algorithms we have constructed several examples based on custom physically based mesh ensemble databases. All dynamic ensembles are comprised of between 32-1005 mesh groups inherited from the simulation preprocess, and the PCA-compressed animations have many thousands of local degrees of freedom. Complete dataset statistics are available in Table I. To assist the research community, we have also made an online repository at `http://graphics.cs.cmu.edu/projects/memg/data` that contains the over 15 GB of PCA-compressed

Example	Vtx/Tri	G	DOF	Frames	Wind	T_{Motion}	T_{MEMG}	$N_{samples}$	τ_{max}	t_{CD}	η	*nLocal*	$\|C\|$	*nWin/nIP*	*OK/Cull/TOut*
Dense Grass	47412/59866	1005	14162	2726	1	790 hr	26 hr	5	30	21 sec	12	6652	206	42/88	.11/.18/.71
Flag Banner	9904/15792	101	10170	1667	1	690 hr	66 hr	10	30	27 sec	20	1426	85	22/53	.05/.24/.71
Banana Tree	8088/13746	32	10170	13880	3	1390 hr	50 hr	4	30	20 sec	5	37	5	946/1340	.28/.33/.39
Large Irises	33331/51468	547	15652	2739	1	827 hr	32 hr	8	40	15 sec	14	1276	49	58/88	.10/.24/.66
Iris Clump 1	12469/19824	173	7610	13196	3	1656 hr	86 hr	5	30	13 sec	16	233	9	798/1127	.22/.38/.40
Iris Clump 2	11068/17388	159	5976	10800	3	1718 hr	18 hr	4	30	4.5 sec	10	119	7	888/1001	.38/.45/.27
Iris Clump 3	12448/19796	173	7226	10800	3	1662 hr	36 hr	4	30	5.3 sec	10	253	9	653/1001	.25/.32/.43
Iris Clump 4	12749/20292	177	6078	10800	3	1419 hr	15 hr	4	30	4.6 sec	10	80	5	950/1005	.41/.45/.14

Table I: **Statistics** *illustrate that interpenetration-free transitions with approximately two seconds of asynchronous motion ($2\tau_{max} \approx 60$) can be found in a variety of complex datasets even with modest numbers of* COMPUTE-MEMG-TRANSITION() *samples ($N_{samples}$), and high-dimensional motions ($DOF = \dim(\mathsf{q}, \dot{\mathsf{q}})$). Times to precompute all transitions (T_{MEMG}) are a fraction of the simulation preprocess times (T_{Motion}), but both codes could be optimized greatly. We also provide average animation collision detection times (t_{CD}); time-out factors (η); average number of times* LOCAL-SEARCH() *was invoked on successful solves (nLocal) and the final contact set size ($|C|$); the number of successful* COMPUTE-MEMG-TRANSITION() *and total integer programs (nWin/nIP); and the fraction of times* SATISFY-MEMG-CSP() *finds a feasible point (OK), or is culled with $\phi > \phi_{cutoff}$ (Cull), or timed out (TOut).*

articulated source motion assets related to this paper. Please see the accompanying video and website for representative animations and transitions.

Motion Library Precomputation. Although our contribution is the MEMG algorithm, and not the generation of realistic motions, we briefly describe the simulator used to generate the motion datasets. For simplicity, all objects were either modeled as thin shells directly (similar to [Grinspun et al. 2003]), or embedded in a shell lattice, e.g., plant stems. An implicit numerical integrator was used to handle stiffness associated with kinematic group-group constraints, the mixture of stiff/soft and heavy/light (botanical) elements, as well as penalty-based contact constraints. Linear systems arising in the Newton iterations were solved using a PCG iterative solver with an ILU preconditioner that was only updated periodically for speed. Wind forcing was based on a simple lattice-based volumetric noise driven by a 1D input intensity time series.

Motion precomputation timings are given in Table I, and they can be tens of thousands of times slower than real-time, and far exceed the cost of computing MEMG transitions–which is also more pleasantly parallel. The speed of the MEMG preprocess times can be substantially reduced in a number of ways to enable practical usage. First, many motions are precomputed in parallel on a cluster, and only the total computation times were reported. Second, the goal of our precomputation solver was not speed, but rather robust dynamics and contact handling (similar to [Bridson et al. 2002]); low-tolerances in the iterative solvers and Newton iterations combined with adaptive time-stepping were used to guarantee that the long simulation runs did not contain any artifacts, such as "popping" contacts, such as when petals were crushed by heavier elements under high winds. Faster simulation algorithms and/or lower quality settings stand to accelerate precomputation dramatically. Particularly promising candidates for accelerating database precomputation are statistical model reduction techniques [Barbič and James 2005], or more agressive approximations [Müller et al. 2006]; such approaches could lead to orders-of-magnitude speed improvements bring dynamics preprocess costs closer to integer programming costs, and allowing much larger examples in practice. In the end, whatever simulation model is used, one typically need only precompute a few minutes of motion (for each wind level). Whatever the speed of the offline simulator, the resulting compressed animation can be rendered efficiently; hardware acceleration can be used to display the collection of local-frame modal deformation models [James and Pai 2002; James and Twigg 2005].

As an implementation detail, the PCA-compressed, multiple wind-level, mesh animation datasets can be well over a gigabyte in size. Therefore we quantize the memory-dominant PCA coordinates, q, to be consistent with the PCA threshold used. In addition, we quantize quaternion and translation vectors used to represent any group transforms, T_{Wk}. Our final MEMG datasets range from ten to one hundred megabytes using conservative ($\approx 5\times$) compression settings.

Grass. This example is comprised of $G = 1005$ blades of grass blowing in a strong breeze (see Figure 9). Each blade of grass is modeled as a V-grooved blade embedded in a prismatic, thin-shell lattice. Synchronized transitions lead to highly correlated transition errors that appear very unnatural for grass. On the other hand, asynchronous transition errors are spread out in time, and have much smaller transition jumps. The most challenging aspect of this dataset is the resolution of nonpenetration constraints, given that *every* unconstrained transition (as well as all synchronous transitions) involved very many interpenetrations. We were surprised how the MEMG integer program solver found clever ways to smoothly transition each grass blade while avoiding interpenetration. Experiments with shorter grass

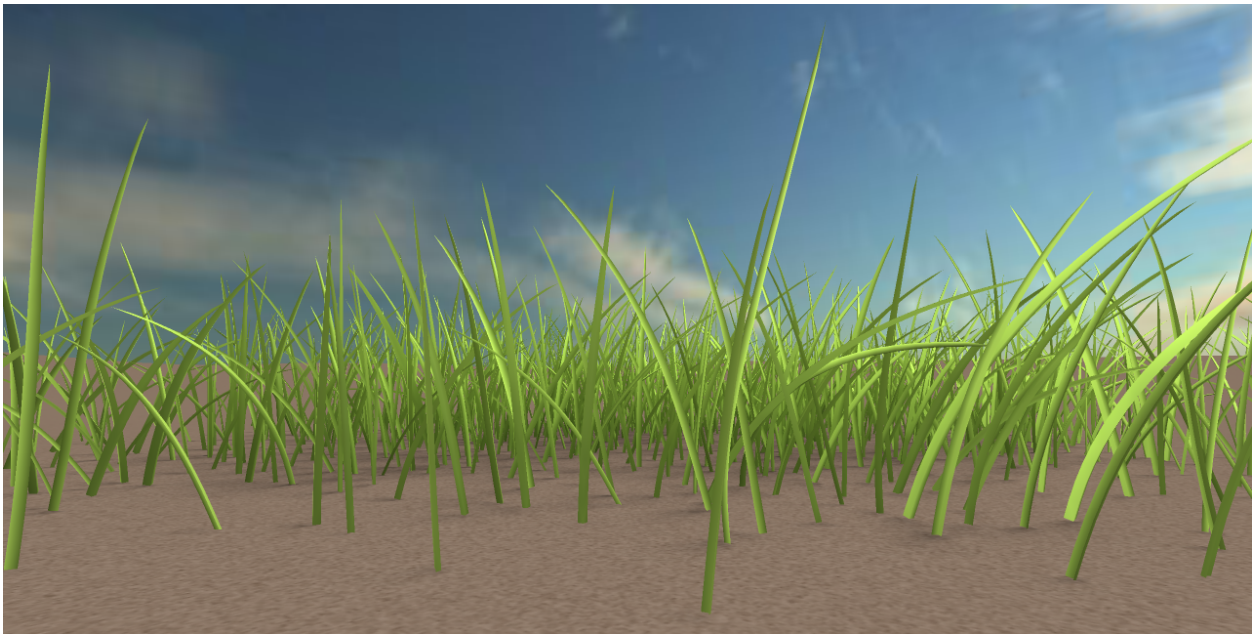


Fig. 9: **Nonpenetrating data-driven grass**

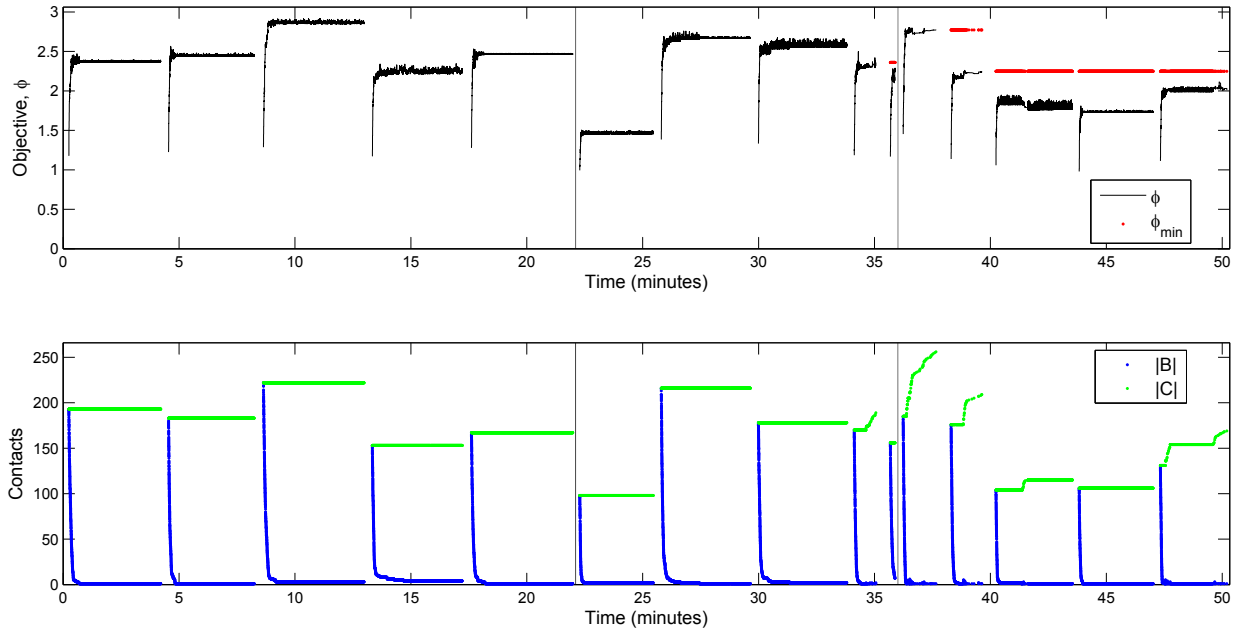


Fig. 10: **Illustration of CSP solver progress** *for the 1005-blade grass example as a function of computation time. A sequence of three calls to* COMPUTE-MEMG-TRANSITION *(three different i values) are shown separated by vertical lines, each with* $N_{samples} = 5$ *calls to* SATISFY-MEMG-CSP *(five j values). (Top) The values of the objective function* ϕ *(in black) and* ϕ_{min} *(in red)–the latter only when feasible points exist. (Bottom) Progress counters for bad contacts,* $|B|$ *(in blue), and the active contact set,* $|C|$ *(in green). Notice that at the beginning of each* SATISFY-MEMG-CSP *call we see* $|B|$ *decreasing quickly (as easy constraints are satisfied rapidly), and consequently* ϕ *increases from its initial unconstrained minimum. As B is emptied, new bad contacts are found by* DETECT-NEW-CONTACTS*, thereby increasing* $|C|$*. Many runs get stuck and time-out due to a handful of bad contacts. The three runs of* COMPUTE-MEMG-TRANSITION *behave as follows: (Run 1) all five samples time-out; (Run 2) first three samples time-out, fourth sample is feasible, and fifth sample is culled; (Run 3) first sample is feasible, second sample is feasible and further decreases* ϕ_{min}*, and the remaining three samples time-out. Minor note: Small temporal gaps between curves are due to scene collision processing (line 7 of (11), and feasible* $(B = 0)$ *runs of* DETECT-NEW-CONTACTS *in (15)).*

and lower wind levels greatly reduced collision complications, and were much easier; conversely, longer grass was much harder, and lead to difficulties with insufficient data, and/or infeasible transitions. Progress of the constraint solver is shown in Figure 10.

Flag Banner. This is a large, flapping, banner of 100 flags attached to a rope (see Figure 11). Given the degree of uncorrelated flag motion, synchronized transitions produced undesirable visual artifacts. Nontrivial collisions and contacts occur between neighboring flags: the average number of interpenetrations introduced by unconstrained minimization of Φ was 81 flag-flag pairs. Interpenetration artifacts occur without nonpenetration constraint programming, but are removed using the algorithm from Section 4. Mathematically, the flag dataset has a very interesting A_* structure: the rope group is contained in every kinematic clique of the many contacting flag pairs. Given that the rope had a $|A_{rope}|$ value proportional to G, this stresses the importance of SELECT-VARIABLE(B,A_*): the exponentially lower probability of picking the rope, $p_{rope} \propto 2^{-|A_{rope}|}$, relative to a flag, means that time is not wasted checking for collisions on *all* C pairs if the rope is jiggled, especially given that the rope has a very small probability of reducing the number of bad constraints as the solution nears completion. Consequently, τ_{rope} remains at its group's unconstrained minimum. If SELECT-VARIABLE() had chosen the random variable from the unsatisfied constraint with uniform probability (as in WalkSAT for k-SAT), the transitions for the flag example would be extremely inefficient–expected $O(|C|)$ slower.

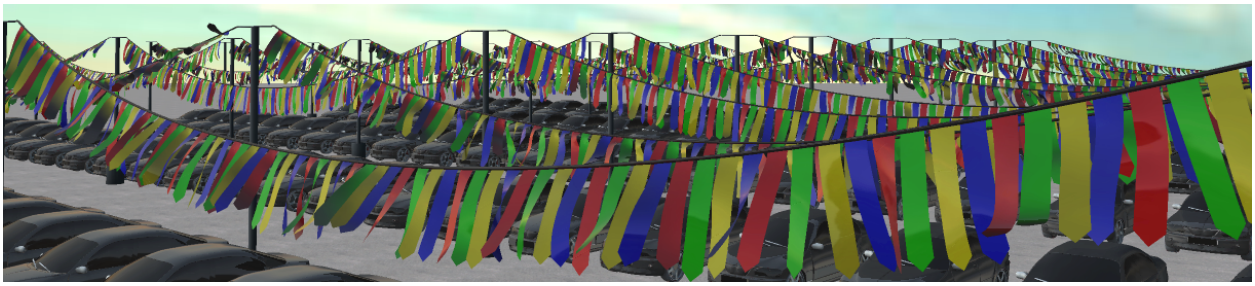


Fig. 11: **3000 flags blowing in the wind:** *Interactive scene comprised of 30 instanced, MEMG-simulated banners, each with 100 thin-shell flags.*

Banana Tree. This is a banana tree blowing at three wind levels, with the highest a strong gale (see Figure 12). Although there are only $G=32$ mesh groups, the large leaves have complex deformations, and repeatedly contact and slide on each other thereby making it a challenging but necessary case for nonpenetration handling. Although many transitions appear to be infeasible up until time-out, the solver still manages to find enough feasible solutions.

Large Irises. This is a large dataset involving substantial collisions approximately 20 irises with grass at a high wind levels (see Figure 13). Without nonpenetration constraints, adjacent irises and grass pass into one another during asynchronous motions.

Fig. 12: **Banana tree transition** *from a moderate to high wind level, with large deformations and contact. First (left) and last (right) frames are 1.5 seconds (90 frames) apart. Yellow indicates groups that are currently blending their group motion data, and highlight how asynchronous motions are computed to provide smooth transitions while avoiding interpenetration. Dynamic precomputed radiance transfer lighting with translucency is used, similar to [James and Fatahalian 2003].*

Fig. 13: **"Large Irises"** *dataset with grass elements.*

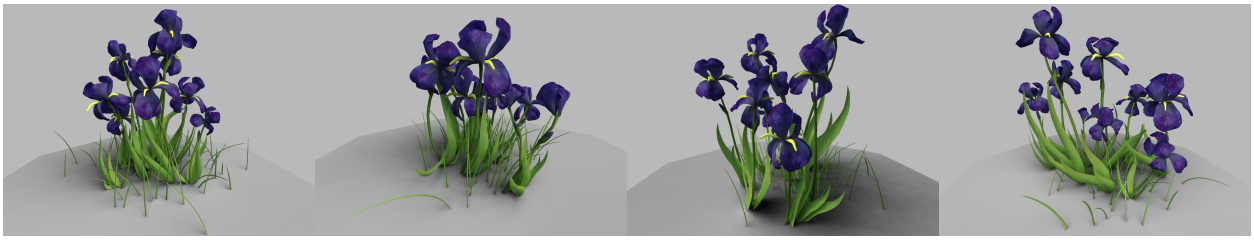


Fig. 14: **Four grassy iris datasets** *that were scaled and instanced to build a large data-driven dynamic environment.*

Large Botanical Environments. Instancing approaches are commonly used for efficient modeling and rendering of botanical environments [Prusinkiewicz and Lindenmayer 1990; Hart 1992; Deussen et al. 1998; Perbet and Cani 2001; Cohen et al. 2003; Decaudin and Neyret 2004]). Our interactive, precomputed, data-driven, plant models are well-suited to instance-based modeling of large dynamic scenes. To illustrate this fact, we constructed a very large, interactive iris field using MEMG models for grassy irises (see Figure 14) and smaller grass-only clumps. The results are shown in Figure 16. Although high-quality, near-field animation is our primary goal, to enable interactive display we used a simplification of geometric and dynamics models to avoid rendering bottlenecks. We use full-quality dynamics for near-field instances, and progressively reduce the rank of dynamics models at farther distances, with the latter compiled as vertex shaders. For very distant LOD models, we use low-rank PCA approximations, and simplified geometric models (analogous to [James and Twigg 2005]). To render huge iris fields with dynamic spherical harmonic lighting, at each frame we compile a small display list for each instance, with vertex attributes for (a) vertex position, (b) a couple PCA displacements, and (c) a single PRT color to modulate the texture. Scaling parameters are used to add simple biological size variability.

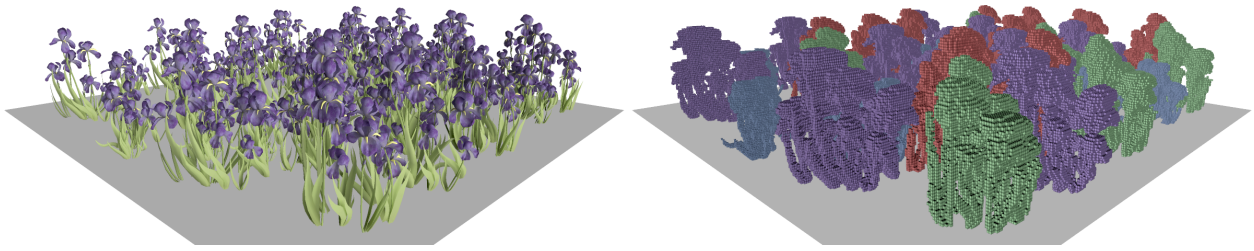


Fig. 15: *Illustration of (Left) MEMG plants placed to avoid interpenetration using (Right) voxelizations of each mesh animation's swept volume.*

Mesh ensemble motion graphs guarantee that no self-interpenetrations will occur, and runtime nonpenetration between

Fig. 16: **Large iris field** *composed of approximately 26,000 MEMG-animated iris and grass clumps, with dynamic wind variations. This example illustrates that MEMG preprocessing can make high-quality, physically based motion available for large scale, interactive virtual environments at hardware-accelerated rendering costs (approx. 3 FPS using Java-based JOGL renderer on a GeForce 6800, Opteron 280 system).*

neighboring MEMG models is guaranteed during a placement phase: a conservative voxelization of each MEMG model's swept volume for overlap testing of randomly inserted models (see Figure 15). At runtime, each model's MEMG frame pointer is advanced as usual, with on-screen models drawn. Spatially varying wind conditions, based on a simple particle model, were used to control MEMG wind levels (see Figure 16): MEMG objects attempt to transition to the integer wind level of the nearest wind particle.

6. CONCLUSION

In summary, we have introduced Mesh Ensemble Motion Graphs for data-driven animation of dynamic mesh ensembles that are decomposed into an ensemble of submesh groups. First, we proposed to splice together mesh animations using spatially asynchronous transitions, where mesh group transitions can occur at different times. In the simplest case, these transitions can be computed by integer minimization of a simple, separable cost functional.

In addition, we have provided an integer programming formulation and randomized algorithm for imposing physical constraints on the asynchronous transition. We illustrate this for the case of nonpenetration constraints, thereby providing a way to avoid unnatural interpenetration artifacts. We have presented some examples to provide evidence that the algorithm can plan interpenetration-free transitions with cost proportional to performing collision detection on the particular transition's mesh animation sequence. Our main technical contribution is a detailed description of an efficient algorithm for computing asynchronous transitions using a randomized integer programming approach that resolves nonpenetration constraints. Our algorithm is effective enough to handle a variety of interesting deformable motion datasets with high dimensionality, leveraging limited data without introducing interpenetration artifacts.

Limitations and Future Work

This work is a step toward the practical goal of data-driven animation of mesh ensembles, and the main contribution (integer programming formulation of asynchronous transitions with equality constraints) is nontrivial but still also somewhat theoretical. For example, we are not proposing the "right way" to simulate, e.g., grass, but simply exploring what is possible with a new data-driven animation approach. Consequently, there are several limitations, but also many exciting areas for future work and practical application.

Data-driven Animation and Content: Presently, we believe that data-driven algorithms for physically based simulation are sufficiently unconventional that it is unclear what areas to develop. However, in the future we expect that related algorithmic approaches will be used to enable the data-driven paradigm to provide interactive content for increasingly complex, and compute-intensive physically based animations of a much wider variety. As mentioned, our examples are "proofs of concept," and it should be expected that orders-of-magnitude improvements are possible in preprocess speed to enable more complex examples in practice–especially for dynamics preprocessing where our simulator was ten thousand times slower than real time. One far-reaching potential benefit of precomputed mesh ensembles for graphics is that they could also permit other complex multimodal phenomena, such as sound and lighting, to be precomputed and co-parameterized for real-time playback.

One interesting extension of our algorithm would be for space-time animation editing and synthesis, permitting motions from different space/time contexts to be reused and combined. This would allow greater use of small datasets for finding transitions (similar in spirit to [Agarwala et al. 2005]), but it could also open the door for animators to edit, adjust and optimize physically based motion databases. For example, MEMGs benefit from asynchronous transitions between two times, $i \rightarrow j$, but a simple generalization would be to allow handfuls of (problematic) groups to transition to yet another intermediate frame, i.e., $i \rightarrow m \rightarrow j$.

Resolving External Interactions: One challenge for data-driven animation is handling external contact interactions. Precomputed mesh animations and transitions can provide interactive playback at greatly accelerated rates, but how to best resolve contact interactions, e.g., between the scene and an avatar, is unclear. One obvious solution is to use a hybrid approach where (low-fidelity) simulation elements help resolve the likely spatially and temporally localized interactions (analogous to [Zordan et al. 2005]), while data-driven elements comprise the rest of the scene, say with one-way coupling to drive the online models. Sparse dynamics computations could also be used to fill in missing data, adapt datasets to different goals, or handle difficult contact scenarios.

A related problem is how to resolve interactions between MEMG objects. This is a concern when placing them in close proximity, such as in the iris field, where noninterference currently implies that the swept volumes of the MEMG objects do not intersect. For low wind levels tight packings can be obtained (see Figure 17), but higher wind levels require larger spacings (see Figure 16). Possible approaches to address this could be adaptive hybrid simulation (both online and/or preprocessed) to resolve interactions, co-planning compatible motions and transitions for the merged ensembles, or even just using static geometry as a buffer.

Fig. 17: *Densely packed irises using only low wind level motion.*

Issues of Interpenetration: In some applications, it may be admissible or necessary to permit small amounts of interpenetration due to (a) interpenetration artifacts in the input animation data, e.g., due to animation compression, (b) the lack of infeasible points due to the difficulty of the problem, and/or lack of sufficient animation data, or (c) less stringent quality requirements. In these cases, a strict boolean interpenetration test is too stringent. A small modification to the algorithm can find interpenetration-free transitions where possible, but also gracefully degrade to allow small interpenetrations in the hardest parts of the scene if allowed. We have also experimented with giving particularly problematic contacts more than the unit weight currently used in the (line 5) comparison of LOCAL-SEARCH() so that once a difficult contact constraint has finally been satisfied, the result is not easily undone thereby accelerating the search for feasible solutions.

Psychophysical Studies: It would be useful to determine exactly how much interpenetration can be allowed, e.g., during transitions, via perceptual studies similar to work on collision processing [O'Sullivan et al. 2003]. The seriousness of any interpenetration seems to depend in a complicated way on many factors, including the spatial and temporal extend of the interpenetration, user view point, visibility, associated geometry, appearance, lighting, dynamics at the time of interpenetration, and intended application.

Contact Complexity and Hysteresis: Our examples were created to illustrate our technique and be reasonably compelling, but of course they have short-comings. One short-coming is that they only convey a certain range of contact behavior. Systems involving numerous and persistent contact, such as tall dense grass pose special challenges. As

mentioned earlier, the boolean equality constraints can be used to more generally define acceptable contacts, which may or may not involve some interpenetration. For systems with more complicated contact behaviors, hysteresis can be introduced wherein the system may get tangled or otherwise deformed in a way that restricts the range of transitionable frames. This is also related to plasticity and destructive effects which change the system in some irreversible way–which is inconsistent with our assumption of quasi-stationary dynamics. In general, dealing with hysteresis appears difficult, and may suggest a motion graph construction strategy that reasons about the reachability of the dataset. For example, instead of selecting the largest connected component for a single wind level, one might pick a smaller connected subcomponent that exhibits less hysteresis. This problem is also related to how much motion is required to build such a MEMG, and the answer to this appears to be strongly example dependent.

Imposing Other Constraints: We have experimented with imposing nonpenetration constraints, however these are but one simple way to require asynchronous transitions to be physically plausible. Additionally, one might add other boolean equality constraints or modify the objective function to introduce other physics, such as encouraging contact forces to be nonzero only during contact, i.e., unilateral contact constraints, or to enforce causality as a wind wave passes over. The randomized constraint solver could also be used to satisfy hard boolean constraints, such as nonpenetration, while reducing other continuous constraints, such as unilateral contact constraints.

Automating the Method: An algorithm for automatic construction of tree-structured scene segmentation, as opposed to manual or simulation-specific segmentations, would be useful for practical use. In addition, support for more general scene kinematics with less restrictive group-group interface definitions, would help generalize transition programming techniques to other deformable models. Our algorithm also involves setting some dataset-specific parameters, of which τ_{max}, $N_{samples}$, and η_{cutoff} are the most important. Although we processed all datasets using roughly similar values, automatic parameter determination is desirable, as are ways to guarantee sufficient transition density and quality.

ACKNOWLEDGMENTS

We would like to thank Moshe Mahler for geometric modeling, especially of the banana tree and irises; Sriram Vaidhyanathan for early wind models; reviewers for helpful comments; and internal feedback by many at Carnegie Mellon retreats. This material is based upon work supported by the National Science Foundation under Grant CCF-0347740; any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Additional support by Pixar and the Alfred P. Sloan Foundation; hardware donations from NVIDIA, computers from Intel, and Maya licenses from Autodesk; Chris Twigg was supported partially by an NVIDIA Graduate Fellowship

REFERENCES

AGARWALA, A., ZHENG, K. C., PAL, C., AGRAWALA, M., COHEN, M., CURLESS, B., SALESIN, D., AND SZELISKI, R. 2005. Panoramic Video Textures. *ACM Trans. Graph. 24,* 3, 821–827.

APT, K. 2003. *Principles of Constraint Programming*. Cambridge University Press.

ARIKAN, O. AND FORSYTH, D. A. 2002. Synthesizing Constrained Motions from Examples. *ACM Transactions on Graphics 21,* 3 (July), 483–490.

BARAFF, D. 1994. Fast Contact Force Computation for Nonpenetrating Rigid Bodies. In *Proceedings of SIGGRAPH 94*. Computer Graphics Proceedings, Annual Conference Series. 23–34.

BARBIČ, J. AND JAMES, D. 2005. Real-Time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Transactions on Graphics 24,* 3 (Aug.), 982–990.

BHAT, K. S., SEITZ, S. M., HODGINS, J. K., AND KHOSLA, P. K. 2004. Flow-based Video Synthesis and Editing. *ACM Trans. on Graphics 23,* 3 (Aug.), 360–363.

BRIDSON, R., FEDKIW, R. P., AND ANDERSON, J. 2002. Robust Treatment of Collisions, Contact, and Friction for Cloth Animation. *ACM Transactions on Graphics 21,* 3 (July), 594–603.

CHOSET, H., LYNCH, K. M., HUTCHINSON, S., KANTOR, G., BURGARD, W., KAVRAKI, L. E., AND THRUN, S. 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang Tiles for Image and Texture Generation. *ACM Trans. on Graphics 22,* 3 (July), 287–294.

DECAUDIN, P. AND NEYRET, F. 2004. Rendering Forest Scenes in Real-Time. In *Rendering Techniques '04 (Eurographics Symposium on Rendering)*. 93–102.

DEUSSEN, O., HANRAHAN, P. M., LINTERMANN, B., MECH, R., PHARR, M., AND PRUSINKIEWICZ, P. 1998. Realistic Modeling and Rendering of Plant Ecosystems. In *Proceedings of SIGGRAPH 98*. 275–286.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco.

GOMES, C. AND SELMAN, B. 2003. Hill climbing search. In *Encyclopedia of Cognitive Science*, L. Nadel, Ed. Nature Publishing Group, London.

GRINSPUN, E., HIRANI, A. N., DESBRUN, M., AND SCHRÖDER, P. 2003. Discrete Shells. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, D. Breen and M. Lin, Eds. 62–67.

HART, J. C. 1992. The Object Instancing Paradigm for Linear Fractal Modeling. In *Graphics Interface '92*. 224–231.

JAMES, D. L. AND FATAHALIAN, K. 2003. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics 22,* 3 (July), 879–887.

JAMES, D. L. AND PAI, D. K. 2002. DyRT: Dynamic Response Textures for Real Time Deformation Simulation With Graphics Hardware. *ACM Transactions on Graphics 21,* 3 (July), 582–585.

JAMES, D. L. AND PAI, D. K. 2004. BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models. *ACM Trans. Graph. 23,* 3, 393–398.

JAMES, D. L. AND TWIGG, C. D. 2005. Skinning Mesh Animations. *ACM Transactions on Graphics 24,* 3 (Aug.), 399–407.

KAUFMAN, D. M., EDMUNDS, T., AND PAI, D. K. 2005. Fast Frictional Dynamics for Rigid Bodies. *ACM Transactions on Graphics 24,* 3 (Aug.), 946–956.

KAVRAKI, L. E., SVESTKA, P., LATOMBE, J.-C., AND OVERMARS, M. H. 1996. Probabilistic roadmaps for fast path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation 12,* 4 (June), 566–580.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion Graphs. *ACM Transactions on Graphics 21,* 3 (July), 473–482.

KWATRA, V., SCHÖDL, A., ESSA, I., TURK, G., AND BOBICK, A. 2003. GraphCut Textures: Image and Video Synthesis Using Graph Cuts. *ACM Transactions on Graphics 22,* 3 (July), 277–286.

LAI, Y.-C., CHENNEY, S., AND FAN, S. 2005. Group Motion Graphs. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. 281–290.

LATOMBE, J.-C. 1991. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA.

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive Control of Avatars Animated With Human Motion Data. *ACM Transactions on Graphics 21,* 3 (July), 491–500.

MOLL, M. AND KAVRAKI, L. E. 2006. Path Planning for Deformable Linear Objects. *IEEE Transactions on Robotics*, to appear.

MÜLLER, M., HEIDELBERGER, B., HENNIX, M., AND RATCLIFF, J. 2006. Position Based Dynamics. In *Proceedings of Virtual Reality Interactions and Physical Simulations (VRIPhys)*. 71–80.

NEMHAUSER, G. L. AND WOLSEY, L. A. 1999. *Integer and Combinatorial Optimization*. Series in Discrete Mathematics and Optimization. John Wiley and Sons, Inc., New York.

NOCEDAL, J. AND WRIGHT, S. J. 1999. *Numerical Optimization*. Springer series in operations research. Springer-Verlag New York, Inc.

O'SULLIVAN, C., DINGLIANA, J., GIANG, T., AND KAISER, M. K. 2003. Evaluating the Visual Fidelity of Physically Based Animations. *ACM Transactions on Graphics 22,* 3 (July), 527–536.

PERBET, F. AND CANI, M.-P. 2001. Animating Prairies in Real-Time. In *2001 ACM Symposium on Interactive 3D Graphics*. 103–110.

PRUSINKIEWICZ, P. AND LINDENMAYER, A. 1990. *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc.

SAFONOVA, A., HODGINS, J. K., AND POLLARD, N. S. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics 23,* 3 (Aug.), 514–521.

SCHÖDL, A. AND ESSA, I. A. 2002. Controlled Animation of Video Sprites. In *ACM SIGGRAPH Symposium on Computer Animation*. 121–128.

SCHÖDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. 2000. Video Textures. In *Proceedings of ACM SIGGRAPH 2000*. Computer Graphics Proceedings, Annual Conference Series. 489–498.

SELMAN, B., KAUTZ, H., AND COHEN, B. 1996. Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. Vol. 26. American Mathematical Society, 521–532.

SUNG, M., KOVAR, L., AND GLEICHER, M. 2005. Fast and accurate goal-directed motion synthesis for crowds. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. 291–300.

SUTHERLAND, I. E. 1963. Sketchpad: A man-machine graphical communication system. Ph.D. thesis, Massachusetts Institute of Technology, Boston, MA, USA.

ZORDAN, V. B., MAJKOWSKA, A., CHIU, B., AND FAST, M. 2005. Dynamic Response for Motion Capture Animation. *ACM Transactions on Graphics 24,* 3 (Aug.), 697–701.

Received August 2006