Interactive Authoring of Simulation-Ready Plants

Yili Zhao Jernej Barbič
University of Southern California

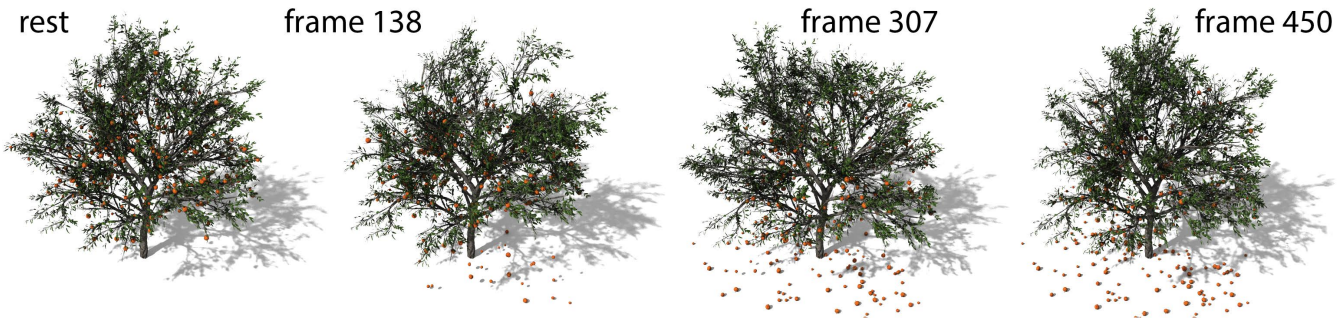

Figure 1: Simulation of a peach tree with anatomically realistic geometry (Prunus Persica), with fracture. *Peaches fall from the tree swaying in the space-time Perlin wind. 299,707 triangles, 237 branches, 3,556 twigs, 18,536 leaves, 330 fruits, 2,950 reduced DOFs, 7 hierarchy levels, 5 msec of simulation per graphical frame.*

Abstract

Physically based simulation can produce quality motion of plants, but requires an authoring stage to convert plant "polygon soup" triangle meshes to a format suitable for physically based simulation. We give a system that can author complex simulation-ready plants in a manner of minutes. Our system decomposes the plant geometry, establishes a hierarchy, builds and connects simulation meshes, and detects instances. It scales to anatomically realistic geometry of adult plants, is robust to non-manifold input geometry, gaps between branches or leaves, free-flying leaves not connected to any branch, spurious geometry, and plant self-collisions in the input configuration. We demonstrate the results using a FEM model reduction simulator that can compute large-deformation dynamics of complex plants at interactive rates, subject to user forces, gravity or randomized wind. We also provide plant fracture (with prespecified patterns), inverse kinematics to easily pose plants, as well as interactive design of plant material properties. We authored and simulated over 100 plants from diverse climates and geographic regions, including broadleaf (deciduous) trees and conifers, bushes and flowers. Our largest simulations involve anatomically realistic adult trees with hundreds of branches and over 100,000 leaves.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques, I.6.8 [Simulation and Modeling]: Types of Simulation—Animation

Keywords: botanical, authoring, interactive, large deformations, model reduction, domain decomposition, FEM

Links: DL PDF WEB VIDEO CODE

1 Introduction

A large fraction of our world is covered by vegetation. Botanical environments are both diverse and very common; therefore, they are crucial for special effects, games and virtual reality applications. Previous computer graphics research on plants has focused on plant geometry creation and appearance (rendering), as well as efficient simulation. Simulation of complex plants is challenging, however, because plant meshes are typically designed for rendering, not simulation. In this system paper, we demonstrate how to robustly and quickly pre-process complex plants in the presence of imperfections in the input geometry, for subsequent fast physically based simulation. Because plants naturally decompose into their constituent parts (branches, twigs, leaves, etc.), we focus on simulators that employ domain decomposition. Such authoring of simulation-ready plants augments and completes simulation, by making it easy to apply simulation to general, complex plant models.

We give an efficient, systematic approach to convert anatomically realistic "polygon soup" plant triangle meshes to a format suitable for physically based simulation. In a manner of minutes and with minimal user intervention, our system can pre-process virtually any plant, which we then simulate in an efficient domain decomposition simulator, accelerated using model reduction [Barbič and Zhao 2011]. Such a combination of authoring and fast simulation enables us to produce quality large-deformation *dynamics* of complex plants, under any given external forces, such as impulsive forces, gravity or wind. Our authoring is robust to non-manifold input geometry, gaps between branches or leaves, free-flying leaves not connected to any branch, spurious geometry ("debris") left in the model, and plant self-collisions in the input configuration. We remove loops in the domain graph using a new user-assisted algorithm to select a minimum spanning tree in a general undirected graph. Our domain graph creation algorithm, instancing, and spanning tree selection procedure apply to any domain decomposition plant simulation method, including those that do not employ model reduction [Twigg and Kačić-Alesić 2010].

Our system supports plants represented using triangle meshes and alpha-masked billboards. Real-time fracture along the domain boundaries is supported, enabling our plants to shed leaves or drop fruits (Figure 1). We also present an approach to perform physically-based inverse kinematics, enabling the user to adjust the

geometry of existing plants by dragging plant vertices. We simulated over 100 plants from diverse climates and geographic regions, including many broadleaf (deciduous) trees and conifers, bushes and flowers. Our system can simulate both simple plants and plants as complex as entire anatomically realistic adult trees with several hundreds of branches and over 100,000 leaves (Figure 5).

2 Related Work

Botanical modeling has a long history in computer graphics and we refer the reader to good survey work [Deussen and Lintermann 2005]. There are many strategies to create and edit plant geometry [Prusinkiewicz 1986; Pirk et al. 2012; Longay et al. 2012]. User-assisted plant modeling [Lintermann and Deussen 1999] has been demonstrated to scale and produce realistic models of complex plants, as seen in the commercial Xfrog system [Xfrog 2009]. We use Xfrog models extensively in our work, as well as models from other sources, such as the SpeedTree [Interactive Data Visualization 1999] system, and generic online content providers [Turbosquid 2000]. Our method works with "triangle soup" input meshes and is agnostic of the specific modeling approach.

Perhaps the simplest approach to animating plants is to drive the deformations kinematically using a stochastic wind [Wong and Datta 2004]. Animations can also be created using pre-recorded motion graphs [James et al. 2007; Zhang et al. 2007], optionally combined with stochastic motion [Zhang et al. 2008]. Physically based simulation can provide dynamics, secondary motion and easier runtime control. A common approach is to model branches as rigid rods connected with angular springs [Sakaguchi and Ohya 1999; Zhang et al. 2006; Weber 2008]. Simulations can be augmented with a proper randomized wind model [Ota et al. 2004], accelerated using level-of-detail techniques [Beaudoin and Keyser 2004], and stabilized using recursive fully implicit methods [Hadap 2006]. Flexible branches have been modeled using 1D linear Euler-Bernoulli beams [Habel et al. 2009; Hu et al. 2012] or nonlinear Kirchhoff rods [Bergou et al. 2008; Bertails 2009]. Plants consisting of deformable branches and leaves can be simulated using oriented particles [Müller and Chentanez 2011], elastons [Martin et al. 2010], and the three-dimensional solid Finite Element Method (FEM) [Twigg and Kačić-Alesić 2010; Lu et al. 2011]. Three-dimensional FEM simulations can easily support volume preservation and spatially-varying material properties. They also automatically incorporate branch thickness (thicker branches are harder to bend), and crooked undeformed branches. Our simulator builds upon the three-dimensional solid FEM. It augments it with frame-aware domain decomposition to support large deformations, modularity, interactive design and fracture, and model reduction to increase the speed of computation. We note, however, that much of our pre-processing pipeline applies broadly to plant simulation: loops in the input triangular geometry can be avoided, instances detected, leaves connected and domain decomposition enabled.

Application of model reduction to linear systems are common [Pentland and Williams 1989], and offer several advantages such as rapid simulation rates, as well as easy adjustments of the simulated and rendered level of detail. In the context of plant simulation, Stam [1997] used a modal basis to compute noise in the frequency domain, and Diener [2009] used a wind projection basis to increase computation speed. Low-dimensional (three of less) linear modal simulations on individual, fully decoupled, branches modeled as Euler-Bernoulli beams were presented in [Habel et al. 2009; Hu et al. 2012], as well as methods to tune the models to match recorded real tree motion. These approaches produce quality high-frequency motion of trees, e.g., leaves and branches rapidly fluttering in the wind, and are as such complementary to our nonlinear model reduction simulator. Because they employ linear models,

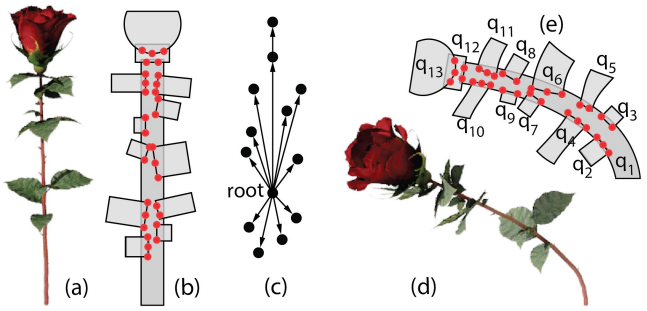

Figure 2: Domain decomposition and model reduction: *(a) undeformed triangle mesh, (b) domain simulation meshes with fixed vertices shown in red, (c) domain graph, (d) deformed mesh, (e) deformed domain simulation meshes, with reduced deformation vectors q_i shown for each domain.*

they cannot, however, support large deformation plant dynamics, characteristic of non-wooden (herbaceous) plants or trees blowing in strong winds. Research on how to quickly simulate plants undergoing large deformations, especially the ones with large structural and geometrical complexity, appears very limited. In this paper, we develop a robust system which allows us to quickly pre-process many complex botanically accurate models and launch them in our interactive simulation program.

Because plants typically consist of well-defined subparts with only limited inter-part interaction, domain decomposition is a natural approach to tackling plant complexity. Twigg and Kačić-Alesić [2010] simulated deformable objects consisting of many parts, including trees, by gluing them together using the Procrustes transform. Their method simulates each deformable part in the full space. Such simulations can be accelerated by orders of magnitude using model reduction, which makes it possible to timestep complex plants at interactive rates. Recently, the combination of domain decomposition and model reduction has received significant attention in the literature [Barbič and Zhao 2011; Kim and James 2011; Yang et al. 2013]. Barbič and Zhao [2011] employed gradients of rotation matrices computed using polar decomposition, for structures without loops in the domain graph. Kim and James [2011] efficiently simulated deformations of characters using inter-domain spring forces. Recent work of [Yang et al. 2013] uses modes obtained from unit displacements of interface vertices, and inertia modes, combined with modal warping. We adopt [Barbič and Zhao 2011] as our runtime simulation method because it does not require a skeleton, pre-existing motion, or well-defined domain interfaces. We augment it with fracture, inverse kinematics, frequency tuning, and bottom-up meshing: unlike [Barbič and Zhao 2011], a global volumetric mesh is never constructed, as each domain is meshed individually, enabling component re-use and modularity.

3 Background: Domain Decomposition and Model Reduction

We now give key model reduction and domain decomposition concepts necessary for our work. In domain decomposition, the object whose deformations are to be simulated is divided into parts, called *domains* (Figure 2, b). In the context of plant simulation, domains are usually the plant logical parts, such as leaves, branches, petals or fruits. The domains form a graph where two domains are connected if they are neighbors. For example, the trunk is connected to the branches emanating from the trunk, and a branch is connected to the leaves growing from the branch (Figure 2, c). The idea of

domain decomposition is to timestep each domain separately, but add proper coupling forces or some other mechanism to keep the domains connected and properly affect each other's motion. This is especially beneficial when the object consists of many repetitive parts (leaves, petals, twigs, fruits, etc.), enabling data re-use and memory savings. For the vast majority of plants, domain graphs are connected and free of cycles (graph-theoretical trees). Some algorithms, such as the one adopted by our method [Barbič and Zhao 2011], can exploit the tree property for faster computation.

Model reduction (see [Sifakis and Barbič 2012] for a survey) is an orthogonal technique to domain decomposition. Given a flexible object (in our context, an individual domain), the key idea is to substitute the high-dimensional dynamics where every vertex has three degrees of freedom, with a projection to some representative, low-dimensional space of vertex displacements. There are several approaches to selecting the low-dimensional basis and efficiently projecting the full-space dynamics to it. We use linear vibration modes, modal derivatives and cubic polynomials for geometrically nonlinear FEM deformable models [Barbič and James 2005] because they can be computed automatically from the elastic material properties (mass density, Young's modulus, Poisson's ratio), and do not require any simulation snapshots.

Domain decomposition can be combined with model reduction. Kim and James [2011] coupled domains with spring forces, whereas Barbič and Zhao [2011] couple them kinematically, essentially employing the reduced coordinates of every domain as (generalized) "joints" (Figure 2, d). We briefly restate their method here; for more details, we refer the reader to [Barbič and Zhao 2011]. The domain tree is first oriented by picking a root, then at every timestep the method proceeds from root to the leaves, timestepping the reduced dynamics of each domain under the internal elastic forces, system forces, interface forces, and any given external forces such as gravity, collisions or user forces. The computed deformations of a domain are employed to compute the new position and orientation for all its child domains, by fitting undeformed interface vertices to their deformed positions, using polar decomposition (Figure 2, d). At the end of the timestep, all domains have been assigned a new position, orientation, and local deformation of its vertices. The method assumes that the interfaces between domains are small, a condition which is typically satisfied with plants. The coupling between the domains is approximated with two types of forces. *System forces* model the effect that child domains undergo system forces due to the acceleration of the parent (similar to passengers in an accelerating car). In the context of tree simulation, such forces produce natural secondary motion of smaller branches and leaves due to the motion of their parent branches. *Interface forces* model the effect that the motion of a domain is affected by the mass of the subtrees attached to it; therefore, for example, the main trunk vibrates more slowly due to the branches attached to it. In [Barbič and Zhao 2011], interface forces are approximated by the assumption that the mass of attached subtrees is concentrated at the location of the interface. Such an approximation improves simulation speed and stability, as it leads to symmetric systems of equations, but it also causes a frequency shift. We show that the shifts can be corrected to match the lowest frequencies of full FEM simulation, by scaling Young's moduli for each domain (Section 7).

4 Plant Preprocessing

Given an input triangle mesh of a plant, we present an efficient and easy to use user-assisted pipeline to organize the plant into domains, create the domain hierarchy and simulation meshes, and pre-process reduced models. Our initial attempt was to perform the entire pre-process without a user, in a console application without a GUI, but such an approach quickly proved to be impractical. All of the steps of our pipeline except the model reduction pre-processing in Step 10 apply generally to plant domain decomposition, and could be used with other domain decomposition methods [Twigg and Kačić-Alesić 2010; Kim and James 2011]. Our procedure starts with a triangle mesh of a plant. We make no assumptions on mesh topology or connectivity and support arbitrary "triangle soups" which may contain cracks, T-vertices or duplicated triangles. Such plant models are very common in practice. The various plant parts need *not* be properly connected to each other in terms of sharing vertices; for example, it is sufficient if adjacent branches simply collide with each other slightly at their common intersection (see Figure 3, right). Our system supports "billboards", i.e., texture-mapped (usually simple) triangle meshes with transparency, commonly used to model leaves, fronds and smaller branches (twigs). We now explain our pre-processing pipeline; each consecutive step is described in a subsection below. Steps performed by the user are marked as **(U)**, whereas fully automated steps are marked as **(A)**. Table 1 analyzes the time needed for each of the steps.

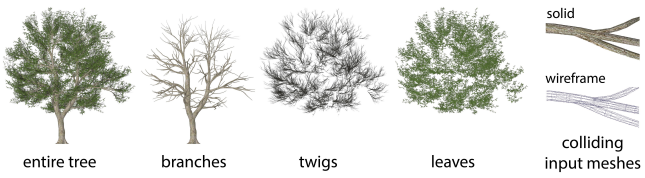

Figure 3: Branches (F), twigs (R1) and leaves (R2). *Branches can simply "sink" into each other in the input geometry (right).*

4.1 (U) Organize input mesh into domains

Given a "polygon soup" mesh, the polygons must be grouped into domains. Each of our domains is characterized by the user as one of the three types: (i) F, (ii) R1 or (iii) R2 (Figure 3). Domains F are flexible, and typically incorporate meshed, non-billboard 3D geometry, e.g., the trunk, branches, or flowers. Domains of type Rx are rigid, and are in practice often instanced. They are typically used for fruits, billboard twigs, fronds, leaves, small decorative geometry or even unwanted geometry left in the model by artists ("debris"; e.g., small unconnected triangles). We use two levels R1 and R2 because in some plants, the artist intended billboard domains to be parented to other billboard domains, e.g., conifer billboard needles attached to billboard twigs. If a single level R1 was employed, some domains may be parented to flexible domains that are too far away, which can cause neighboring domains to separate at runtime. A typical example of the decomposition is to assign all branches into F, twigs into R1 and leaves into R2 (Figure 3). The user is free to deviate from such guidelines, however. In some of our examples, leaves are modeled as flexible triangle meshes, and assigned to F.

Every triangle must be assigned into exactly one domain, and each domain is assigned one of the F,R1,R2 types. We employ a user interface similar to that in, say, Maya, where the user can select triangles or domains, and show / hide / add / subtract / delete / merge them. Domains can be selected with the mouse and then tagged as either F, R1 or R2. In some models, polygons are pre-grouped by the artists into individual logical parts, e.g., each leaf is a separate domain already in the input, in which case the domains must only be selected, and their type identified. Often, however, the parts of the same kind are grouped together, e.g., all leaves are initially in one domain, and must therefore be separated into individual leaf domains. Therefore, another operation that we support is to break an existing domain into connected components, where two triangles are considered connected if they share a vertex. We compute the connected components using the union-find datastructure [Cor-

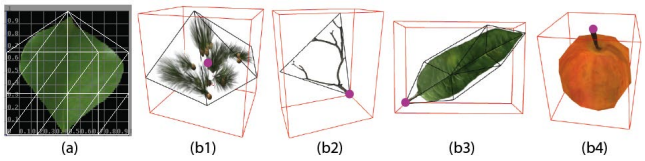

Figure 4: Instancing and anchors: *(a) One-to-one texture map with transparency. (b) User-selected anchor points (in purple).*

Figure 5: Instancing: *The 120,000 alpha-masked billboard leaves of this Oregon white oak tree (Quercus Garryana, 2,360,868 triangles) are replicated copies (instances), detected automatically from the input triangle oak mesh (shown left). 871 branches, 1 sec of simulation for one graphical frame. We note that previous methods [Barbič and Zhao 2011] generated positions of leaves procedurally. To the best of our knowledge, this is the first fully mechanically simulated tree with realistic adult geometry anywhere in science.*

men et al. 1990]. When breaking domains into components, the user can choose to impose a rule that all triangles in the domain may use at most one texture image; otherwise, the domain is broken further, into one domain for each texture image. Such a rule is useful with instancing, but is rarely needed because most models already satisfy this requirement as is. Although we did not encounter it in practice, the connected components may also be useful when the input geometry is unorganized, such as each triangle initially forming a separate domain.

4.2 (U) Instancing

Many plants consist of repeated parts. The instances are translated, rotated, scaled and sometimes nonlinearly deformed copies of each other, e.g., replicated leaves or flower petals. Instancing decreases authoring time and runtime memory footprint, as each instance needs to be preprocessed and stored only once. Furthermore, it greatly aids with hierarchy creation, as the user needs to specify the anchor points (described below) only once per instance. Complex examples are too tedious to author without instancing. Our first attempt to perform instancing used shape matching directly [Müller et al. 2005]. This worked on some plants, but not on others, because of scalings, nonlinear instance deformations, and occasional lack of vertex correspondence. Instead, we automatically identify instances using a combination of shape matching and texture map analysis, as follows. We assume that the triangle mesh of each instanceable domain is texture-mapped with a single image, with a one-to-one texture map (Figure 4, a). In particular, vertices cannot have different texture coordinates (u,v) if they appear in more than one triangle, and triangles cannot "fold over" or cross each other in the *uv* space. In our model databases, we did not encounter texture maps that would not be one-to-one.

We first inspect the texture image and the number of vertices; domains that do not match in both, are not instanced copies. For each vertex i of the first domain, we then find the vertex $j = \mathcal{P}(i)$ of the second domain whose texture coordinates are closest to those of vertex i, by performing a nearest neighbor search in the *uv* space. If the map $\mathcal{P}$ is not one-to-one and onto (a permutation), or the distance to the nearest neighbor is greater than $\varepsilon = 10^{-3}$ for any vertex, we deem the domains not instanced. Finally, we check that the triangle mesh topology is same for both domains: if vertices i, j, k form a triangle in the first domain, so must vertices $\mathcal{P}(i), \mathcal{P}(j), \mathcal{P}(k)$ in the second domain, and vice-versa. If topology is the same, domains are deemed instanced copies of each other. At first, we did not seek the permutation $\mathcal{P}$, but simply checked the *uv* distance between vertex i of the first and second domain. This approach did not work well because modeling packages sometimes arbitrarily reorder triangle mesh vertices before exporting them. Note that our definition of instancing relies only on the *uv* space, and therefore permits arbitrary (nonlinear) world-space object transformations.

Example	1	2	3	4	5	6	7	8	10	total
dahlia	16	40	0.2	0	3	27	0.01	0.45	141	228
jasmine	29	14	0.6	20	3	63	0.09	1.13	124	255
white pine	36	3	7	10	3	60	3.50	17	170	311
broadleaf	7	10	1.4	0	3	269	1.14	3.3	136	431

Table 1: Timings *for each pre-processing step, in* **seconds**, *for four representative models (shown in Figure 10), including user and computer time. Four users were involved in the experiments. Timings for steps 9 and 11 are not shown to save space, but are included in totals; they are less than 0.7 sec in all examples.*

All the domains that are instanced copies of one another are placed into an *instance set*. A single domain is chosen as a representative instance; we choose the one that appears first in the input mesh. For each instance set, the user selects a world-coordinate "anchor point" on the representative instance (see Figure 4, b), by clicking with the mouse on the model. We then determine and store the corresponding (u,v) coordinates. The anchor point will be used to determine the instance transformation, and also in Step 7 to assign a parent to each Rx domain. The anchor point should typically be selected at the end of the botanical piece, e.g., on leaf's stem (petiole), next to the attachment point to a branch (see Figure 4, b). Sometimes, billboard domains are not designed to attach to anything, but are free-floating in great numbers, e.g., clusters of needles on a conifer tree (Figure 4, b1). In such cases, the user should select the center of the billboard polygon. Next, for each instance from an instance set, we determine the *linear* transformation that best aligns the vertices of the representative instance to this instance, using shape matching [Müller et al. 2005]. Optionally, the user can force the linear transformation into a rotation, computed using polar decomposition. It is often preferable to use linear transformations, however, because the leaves are often scaled in size to add variety. The instance transformation is stored to disk, and used at runtime to properly transform the domain. Only a single mesh must be stored for each instance set.

4.3 (A) Computing the F domain graph

Next, our system automatically builds the *domain graph* for the F domains. The nodes of the graph are the F domains, and two nodes are connected if the two triangle meshes intersect in the undeformed configuration. We determine the graph edges using collision detection, accelerated by bounding volume hierarchies and spatial hashing [Lin and Gottschalk 1998]. Because we only need to perform collision detection once (on static shapes), the domain graph construction only takes a few seconds at most, even for our most complex examples. Initially, we attempted to use collision detection on volumetric meshes (computed using voxelization) to create our

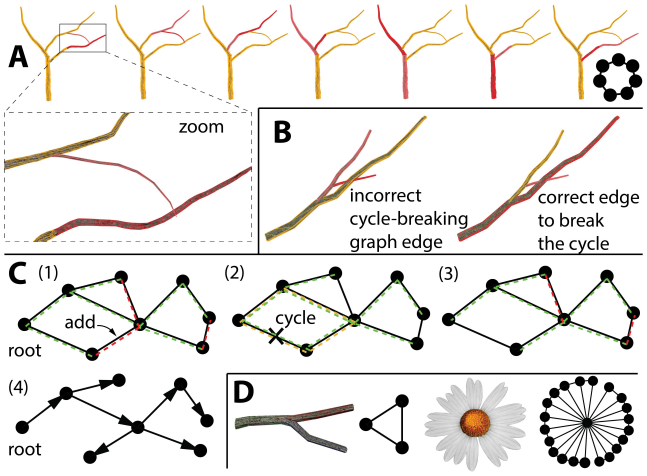

Figure 6: Loop resolution: A: *a 7-cycle.* **B**: *an example of a 3-cycle where a computer may make a mistake and that requires user intervention. Because both smaller branches collide with the bigger branch, computer initially suggests an incorrect loop-breaking graph edge (center), whereas the correct edge is shown on the right.* **C**: *the minimum spanning tree selection algorithm. (1): Initial minimum spanning tree (green) and redundant edges (red). The edge to be resolved next is marked by "add". (2): The loop. User decides to remove the edge marked with "X". (3): The minimum spanning tree after deleting the edge. (4): The hierarchy after the remaining two redundant edges were processed.* **D**: *Examples of loops in input geometry: "Y"-bifurcation and flower petals.*

hierarchy. Although such an approach avoided gaps between domains, it created many spurious graph edges, which greatly complicated the spanning tree selection in Step 6.

4.4 (U) Connecting the F domain graph

The graph computed in step 3 may not be connected. In practice, we encountered such situations in about 25% of all models. Most often, this occurs for one of two reasons: (1) there is a small gap between two domains, often visually (nearly) invisible and unimportant for rendering, or (2) the domains are "debris": small pieces inadvertently left in the model by the artist, often invisible inside branches. We let the user connect the graph as follows. We compute the connected components using the union-find datastructure. The user can then select arbitrary two domains in arbitrary components, and connect them. At any time of this process, she can recompute the graph and the connected components. In most cases, when the graph was initially not connected, the total number of components was less than 10. We encountered a few cases of "debris" with approximately 50 components, which we deleted one by one. When visualizing the connected components, we sort them based on size. At any moment, the user can simply delete the remaining components, if their size is deemed insignificant.

4.5 (U) User selects the root domain

Plants in nature are rooted. Although most models come pre-oriented so that the Y-axis is up, this is not guaranteed, so the user has to specify the root domain (typically, stem or trunk). We provide the domain with the largest diameter as the initial suggestion. We compute the diameter approximately, by first computing the domain centroid, followed by the tightest ball centered at the centroid.

4.6 (U) User-assisted resolution of loops

Although in principle the output of Step 4 should have no cycles (a tree), this is rarely the case in practice. For example, an artist may have accidentally left a branch colliding with another branch, forming a cycle (see Figure 6, A). There are three very common causes of cycles: collisions of non-neighboring branches in the input (very common with complex trees), "Y-bifurcations" where a branch splits into two branches and all three meshes collide with each other, forming a 3-cycle, and petals on flowers (Figure 6, D). We found cycles to be common in commercial plant model libraries. They must be removed so that a tree hierarchy can be computed. It is difficult to remove cycles automatically, e.g., the small transverse branch in Figure 6 (magnified in A), may as well be connected to either of the main branches. A human, however, can look at the branch and recognize its intended direction.

Each edge in the domain graph is either correct or spurious, and we rely on the user to classify it. For complex plants, the number of edges ranks in tens of thousands, and it is not practical for the user to visit and classify every edge. Therefore, we designed an algorithm for user-assisted removal of loops, which provably always removes exactly the spurious edges, and in practice greatly decreases the number of edges that the user must classify. Our algorithm applies generally to any problem where a minimum spanning tree avoiding the "bad" edges must be selected out of the many minimum spanning trees of an undirected graph with cycles. It assumes that we are given an undirected graph with n vertices and $n-1+k$ edges, $k \geq 0$ of which are spurious, but it is not known which ones without asking the user to classify the edges. The goal is to minimize the number of user classifications.

We first compute an initial candidate minimum spanning tree, using a breadth-first traversal starting from the root domain. There are exactly k edges which are not in the minimum spanning tree. They form the set of *redundant edges*, and we prioritize them by their breadth-first order (Figure 6, C1). Note that the redundant edges are not necessarily spurious, and that the spurious edges may be in the non-redundant set. We then ask the user to resolve the loop formed by adding each redundant edge, one by one. For each redundant edge, we add it to the current minimum spanning tree, and therefore exactly one cycle appears in the resulting subgraph $\mathbb{S}$ (Figure 6, C2). At least one spurious edge must appear in this cycle; otherwise, the domain graph has a genuine cycle, which we assume is not possible with plants. The cycle is discovered by traversing the minimum spanning tree from each of the two vertices of the redundant edge towards the root of the tree, until a common ancestor is detected, thereby detecting a cycle. The cycle is then visualized to the user, by coloring the cycle domains in a golden color (as in Figure 6, A). The user is then asked to break the cycle by removing exactly one spurious edge. The user does so by traversing the cycle (as in Figure 6, A; the two node domains that are the endpoints of the edge are shown in red and pink), and selecting the edge to be removed. After the edge removal, the working graph $\mathbb{S}$ is a tree again, and the number of redundant edges has decreased by one (Figure 6, C3). This process is repeated k times until the redundant set becomes empty, i.e., the working graph $\mathbb{S}$ is a tree and all the redundant edges were resolved. Because we repeated the process k times, each time removing one spurious edge, there can be no more spurious edges in $\mathbb{S}$. We then orient $\mathbb{S}$ to form a tree hierarchy, starting from the root and proceeding to the leaves (Figure 6, C4). Because redundant edges are prioritized by their breadth-first traversal order, the user resolves cycles closer to the root first.

In our plant database, virtually all plants initially had cycles in the domain graph, and required user intervention. Most loops, however, occur due to "Y" bifurcations, and can be resolved very quickly. For small / moderate examples, these were often the only loops.

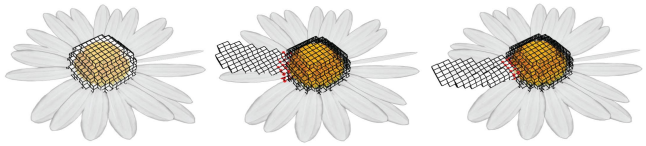

Figure 7: Voxel simulation meshes. *Left: root domain. Middle, Right: two representative domains. The meshes for the different domains are completely independent and do not need to meet in a common interface. Adjacent petals can thus be meshed independently, even though they collide in the input triangle mesh (middle, right). Fixed vertices are shown in red, and need not be vertices of the parent volumetric mesh; some are even outside the parent mesh.*

The vast majority of redundant edge sets that we have encountered in our examples had less than 100 edges. A few large examples, such as large trees, had approximately 500 redundant edges. Even for the most complex trees, the redundant edge set removal was manageable and was completed within minutes of user time. We were able to significantly shorten the user time by implementing an auto-focus feature where the camera automatically focuses on each cycle while the user is processing it.

4.7 (A) Add domains Rx to the domain tree

We now assign parents to domains Rx. For complex plants, it is too tedious to do this manually. Domains R1 are always parented to F domains, whereas domains R2 can be parented to R1 or F. Such parenting can never introduce new cycles into the domain graph. We initially tried assigning a parent to each Rx domain by performing a minimum distance query to all the F domains, and selecting the closest (perhaps colliding) domain as the parent. However, some Rx domains collided with more than one F domain, requiring user intervention. Even worse, sometimes there was only one colliding but incorrect parent, resulting in an undetected mistake. This typically occurred when a leaf petiole is separated from the branch by a small gap, whereas the tip of the leaf is (accidentally) colliding with another branch. Instead, we use anchor points to robustly determine the parent. The anchor points were selected by the user in Section 4.2, once per instance set. The anchor positions on the instanced copies are now computed by finding the world-coordinate location on the domain mesh whose texture coordinates are (u,v), where (u,v) are the texture coordinates of the anchor point on the representative instance. Such anchor position computation was very robust in practice. We then perform a nearest neighbor search, seeking the nearest triangle in all domains in F to the anchor position; the closest domain becomes the parent. For R2 domains, we also search in R1. Our algorithm is robust to cracks and can accommodate (intentionally) "floating" domains that are common with billboarding. For example, tree leaves are sometimes simply accumulated in close proximity to give a space-filling perception, and are far from any branch. The nearest neighbor queries are accelerated using a hierarchy of axis-aligned bounding boxes, and only take a few seconds, even for complex models (Table 1).

4.8 (A) Build simulation meshes

We build the volumetric mesh for each domain by voxelizing the domain's triangle mesh [James et al. 2004] (Figure 7). We chose this approach because it is completely automatic and supports arbitrary (potentially ill-formed) input "polygon soup" geometry. Alternatively, one could employ automatic tetrahedral meshes such as [Labelle and Shewchuk 2007]. The user specifies the maximum voxelization resolution (it defaults to 100 in our system) for the expanded bounding box of the input triangle mesh (we use expansion factor of $1.2\times$). The voxelization resolution of each domain is then set automatically, by assigning to each domain a resolution proportional to the longest edge of its bounding box. Using such an adaptive resolution, the meshes of different domains grade approximately uniformly across the entire plant (see Figure 7).

The voxel meshes for the individual domains are separate meshes; they need not topologically connect to meshes of other domains using any well-defined interface. Such meshing flexibility is possible in domain decomposition simulators that do not need the connectivity requirement, such as [Twigg and Kačić-Alesić 2010; Barbič and Zhao 2011]. Our "bottom-up" meshing has the advantage that an entire simulation mesh never needs to be constructed. It is also very modular. Input triangle mesh collisions of neighboring domains (e.g., petals on a flower) are not a concern, because each domain is simply meshed separately (see Figure 7). If a new domain needs to be added, it can be pre-processed in isolation and added to the domain graph. In [Barbič and Zhao 2011], domain decompositions were created by forming a global tetrahedral mesh, which was then manually subdivided into domains. Although such a "top-down" approach results in well-defined interfaces, it requires careful manual work to produce quality interfaces. For example, the choice of whether to place a tetrahedron into the left or the right domain at the interface can affect the bendability of short branches. Also, top-down approaches require special handling in the presence of collision in the input triangle mesh; otherwise, the colliding branches or petals will simply be welded in the global volumetric mesh.

4.9 (A) Assign fixed vertices

Fixed vertices connect a domain to its parent. The root domain is rooted to the ground, either by fixing all vertices that are below a user-provided height, or by manually selecting its fixed vertices. Fixed vertices for the remaining domains are set automatically as follows. For each parent-child pair in the hierarchy, we constrain the child vertices that are located close to the parent. Specifically, we detect child voxels that intersect parent voxels, and constrain all the vertices in those child voxels. Because we always constrain all eight vertices of a voxel, we avoid degenerate planar sets of fixed vertices. The intersection test is accelerated using a bounding volume hierarchy. Such an assignment handles both the case where the child voxels are smaller than parent voxels and may be completely inside a parent voxel, and the case where child voxels are large and may subsume parent voxels. It can happen that all the vertices of a domain are deemed fixed; e.g., with small, in-grown, branches close to a bigger branch; in such cases, we declare the domain to be rigid. When no fixed vertices are detected, the domain is also made rigid. Such cases are rare in our data, but sometimes occur with domains not properly connected to the rest of the plant, e.g., small branches added for decoration in the tree crown. Note that for our domain decomposition approach, it is not necessary to establish a well-defined interface between the two domains; the fixed vertices of a child domain may even be outside of the parent mesh (Figure 7, middle, right). Because the interfaces between branches and/or leaves often have small surface areas, it is common to neglect their bending. Often, plant simulators even assume that all the parts (the branches) are completely decoupled [Habel et al. 2009; Hu et al. 2012]. In our work, the domains are coupled, with the assumption that the interface deformations are small.

4.10 (A) Compute low-dimensional simulation basis and pre-process reduced dynamics

Given the fixed vertices, we compute linear modes, their large-deformation correction (modal derivatives [Barbič and James

2005]), and the simulation basis. By default, we use the first 10 linear modes, and a 20-dimensional basis. We use a default (and tunable), spatially uniform, mass density $\rho = 1000 kg/m^3$, Young's modulus $E = 10^6 N/m^2$ and Poisson's ratio of $\nu = 0.45$, corresponding to a fairly incompressible material. We then precompute a geometrically nonlinear FEM reduced model for each domain [Barbič and James 2005]. Each of these models is a compact, low-dimensional representation of the FEM dynamics of each domain. It supports large deformations, and can be timestepped rapidly, in microseconds. The domains are coupled to each other as described in Section 3. The entire process is automatic. In order to save space and increase speed, we use an adaptive number of modes per domain, by pre-processing smaller bases for smaller domains. The user sets the maximum size of a domain simulation basis $r_{\max}$ (we use $r_{\max} = 20$). For a domain with e elements, the number of modes is then computed as $r = \max(5, \lfloor r_{\max} \log(e)/\log(e_{\max}) \rfloor)$, where $e_{\max}$ is the maximum number of elements in a domain (see also [Barbič and Zhao 2011]).

4.11 (A) Tune Young's modulus

Reduced models incorporate the geometry of each domain and automatically produce correctly scaled domain frequencies, e.g., longer branches vibrate slower. However, the global stiffness scale is arbitrary; one can globally scale ρ and E with an arbitrary constant without affecting the basis, and the precomputed reduced polynomials only scale by a constant [Barbič and James 2005]. Because we are using a FEM method, it would be possible to set the material properties for wood, stems, leaves, etc. Although progress on measuring elastic material properties has been made [Bickel et al. 2009], the parameters for plants are typically not available. Therefore, we assign default, spatially constant, E, ν, ρ. We then pre-process the reduced models, and exploit the fact that natural vibration frequencies scale linearly with E. Next, we globally rescale the Young's modulus so that the lowest natural vibration frequency of the root domain becomes 1 Hz, i.e., $E' = E/f_0^2$, where E is the default Young's modulus, and f_0 is the lowest natural frequency of vibration of the root domain, determined using a sparse eigenvalue solver [Lehoucq et al. 1997]. This computation is fast; it takes less than one second even for complex models. The user can later further adjust E of each domain, to make it stiffer or softer (Section 5).

5 Interactive Plant Design

Because our method is fast and modular, it is possible to use it as an interactive shape editor for plants. The user can easily delete or add new parts at runtime, either by duplicating existing plant parts, or importing parts from a pre-processed library of plant parts. The editing process is local and interactive. The mass, stiffness, position and orientation of each branch, twig or leaf, as well as a linear scale (geometric size), can all be adjusted interactively at runtime, with immediate physically based simulation feedback to the user, without any additional pre-processing. It is possible to randomize these choices, creating an arbitrary number of variations of the same plant. In our interactive editor, we can select an arbitrary domain, and linearly scale the stiffness (Young's modulus) of all of its elements by any factor $\alpha > 0$. It can be shown that under such a scaling, the reduction basis does not change [Barbič and James 2005], whereas the reduced forces scale by α; therefore, the scaling is instant, and there is no need to maintain an explicit volumetric mesh or its element material properties. Because stiffness is proportional to the square of the lowest natural vibration frequency, such scaling makes it possible to cause a branch to oscillate faster or slower. For cinematic effect (to show more secondary motion), we sometimes found it useful to make smaller branches oscillate more slowly than dictated directly by physics. We achieve this by scaling every domain with a factor $\alpha = \beta^d$, where $\beta > 0$ is a constant and d is the depth of domain in the hierarchy. Because our method is fast, the user can tune β interactively with immediate feedback; typically, values close to $\beta = 0.1$ were producing good results.

Figure 8: Editing tree shapes using point constraints: *The tree shape was adjusted to avoid collision with the house. The six manipulated points are shown in red. Bottom row shows the tree without the leaves. 45 branches, 125 twig billboards, 1154 leaf billboards, 8 msec of simulation for one graphical frame.*

In our system, the user can also manipulate the plant using inverse kinematics-like handles. Using such a tool, it is possible to constrain and drag plant vertices, while the rest of the plant automatically re-adjusts using physics to a good-looking, minimal strain energy configuration. For example, a tree can be made to lean in a certain direction or avoid external objects (Figure 8). The user can also use it to resolve any unwanted collisions in the rest configuration, or simply re-adjust the plant shape to increase scene geometric variety. In our IK tool, the user can select or deselect an arbitrary number of vertices (IK handles). As the user drags the mouse, a three-dimensional force is applied to the active IK vertex, whereas the remaining IK vertices are kept fixed to their positions using linear springs. The mouse force is applied in the "screen plane", i.e., plane orthogonal to the view direction and cutting through the manipulated vertex. The force magnitude is proportional to the number of pixels traveled by the mouse since the beginning of the drag. We use our standard real-time dynamics solver for such IK manipulation; no special code is required. Because the model normally undergoes dynamic motion, we must employ some mechanism to quickly stabilize the motion to a limit equilibrium configuration. One approach is to perform a Newton-Raphson iteration to seek the model equilibrium under the IK linear spring forces (a static solver [Mezger et al. 2008; Barbič et al. 2009]). However, such a solver often suffers from a high linear system condition number, which has lead to instabilities in our experiments. We found much better results by simply using a high level of stiffness-proportional Rayleigh damping, in a *dynamic* simulation. Although the model does not reach the equilibrium configuration instantly, with proper gains and damping the convergence is rapid and stable. A single stiffness gain for all the linear springs was sufficient in our examples. Because we already injected sufficient damping into our simulator, linear springs did not need any additional damping.

We note that we first attempted to perform inverse kinematics by enforcing *exact* user control over the handle positions. This was performed by solving, at every timestep, an optimization problem that

minimized the total strain energy of the plant subject to the exact IK constraints. This approach did not work very well in practice. Because the handles must be specified in the three-dimensional space which can be difficult to visualize on a 2D screen, it was easy for the user to accidentally command unreasonable handle positions. As the solver was trying to meet the constraint exactly, the plant would stretch unnaturally, which led to vibrations and instabilities. Instead, when positions are enforced via springs, the solver has the ability to selectively "yield" on each constraint as needed. This tends to produce much smoother, natural-looking shapes. We were able to tune the IK stiffness gains to reach both good output shapes and minimal deviation of the constrained vertices from their prescribed positions (Figure 8). After the user is satisfied with the shape, she can save it to disk, and re-process the reduced models with respect to the new rest shape. For small/moderate edits where the old reduced basis is still sufficient, a re-process is not necessary: one can simply compute the reduced forces for the new shape for each domain, and then offset reduced forces so that the new shape is the rest configuration.

6 Pre-specified Fracture

In nature, plants often fracture at the joints between its parts. For example, leaves or fruits detach from branches (see Figure 9), or branches crack away from the main stem. Such *fracture with pre-specified patterns* is useful in interactive applications because it is controllable and artist-directable [Parker and O'Brien 2009]. We support such fracture by monitoring the (reduced) interface forces between adjacent domains. If the L_2-norm squared of reduced interface force vector exceeds a user-adjustable threshold, we fracture the entire subtree from the main structure. The subtree (in many applications a single domain) then undergoes a ballistic trajectory under gravity, e.g., peach tree fruits land on the ground (Figure 1). Such fracture is computationally extremely inexpensive as the L_2-norm test can be performed in nanoseconds. Because each part has its own rendering mesh independent of all the other parts, the objects are automatically free of holes after separation and there is no need for any re-meshing. The fracture events could also be scripted / keyframed, e.g., to simulate trees undergoing an explosion. Our domains can only fracture at the interfaces to other domains. For more detailed fracture, domains can be divided during the pre-process, e.g., the trunk can be pre-cut into two pieces.

Figure 9: Real-time Fracture: *The tea bush (Camellia Sinensis) leaves are shaken from the bush by the user-applied force in real time. Left: Before fracture, with applied user force indicated. Middle: during fracture. Right: after fracture. Note that the leaves close to the user force location fractured in greater numbers than elsewhere on the model because they underwent higher accelerations. 6 msec of simulation per graphical frame.*

7 Results

We pre-processed over 100 plants; we provide a selected subset of 32 pre-processed and simulated models in Figure 10. It can be seen that the performance is interactive even for very complex plants. Plants of small to moderate complexity are fast enough for real-time systems such as computer games. Inverse kinematics and fracture were demonstrated in Figures 8 and 9, respectively. Plant motion resulting from user (mouse) forces, followed by free vibration, is demonstrated in Figure 11.

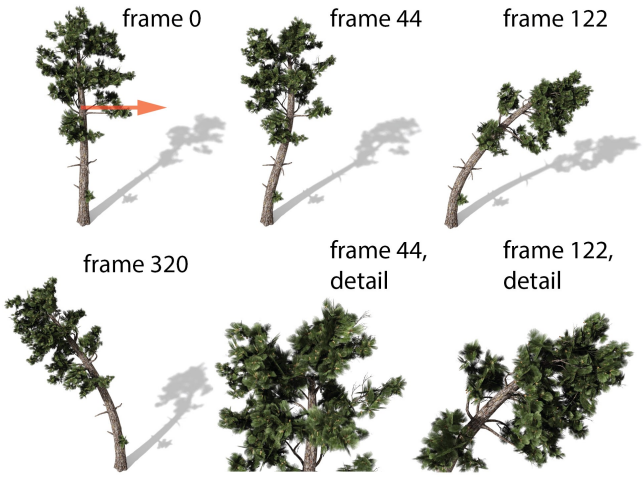

Figure 11: Space-time instantaneous (localized) external force followed by free vibration. *Secondary motion in the smaller branches due to motion of main trunk is clearly visible. Simulation time: 5 msec per graphical frame.*

We deform our plants using wind, user forces and gravity. Our wind consists of two components: a wind with a (tunable) constant direction and magnitude, and a randomized wind, implemented as a 4D space-time Perlin noise [Perlin 2002], with standard parameters: number of frequencies and how quickly they decay (persistence). All the wind parameters are easily adjustable at runtime without any precomputation. Our method can model the entire spectrum of winds from gentle breezes, moderate winds (Figure 1) to hurricanes (Figure 12). Stochastic noise is often used to animate trees by directly driving the deformations [Ota et al. 2004]. We employ Perlin noise to create spatially varying and controllably turbulent wind forces, causing large deformations and secondary motion of branches due to the motion of parent branches. We use Perlin noise directly, but other wind generators such as the $1/f^{\beta}$ noise [Ota et al. 2004] or the Navier-Stokes equations [Akagi and Kitajima 2006; Selino and Jones 2012] could be used instead. In order to apply the wind, we must sample it at properly selected locations on the model. One could evaluate the wind at every plant vertex, but doing so would require many wind evaluations and subspace force projections. Another alternative that we considered but did not pursue due to prohibitive cost was to sample the wind on a regular grid, and then seek the nearest vertex to each grid point. Instead, we select a representative set of volumetric mesh vertices on each branch domain, and then sample the wind at those locations (see Figure 12, top-left). Each wind sample is scaled with the volume of each branch. The user supplies the desired number of samples s per domain. We use a constant number of samples for all the branches (typically $s = 5$), but s could be scaled with domain size, e.g., $s = 1$ for leaves. The sampled vertices for a domain are then determined automatically, as follows. We first build a mesh graph for our voxel domain mesh, where vertices are nodes and nodes are connected if they are adjacent mesh vertices. We then use Dijkstra's algorithm to compute the minimum graph distance of each volumetric mesh vertex to the set of *fixed* vertices for this domain. Let D denote the maximum distance. Then, for each $i = 1, \ldots, s$, we select any vertex with distance $\lfloor iD/(s+1) \rfloor$ as our sample. Because plant branches

Figure 10: Representative subset of simulated models *including flowers, bushes, broadleaf and conifer trees. Input meshes are from Xfrog, SpeedTree and 3dmolier (Turbosquid) model libraries. Models were deformed either by pulling on vertices, gravity, or using a 4D Perlin wind. Each plant reports #triangles, #domains, #flexible domains, total # of DOFs, the simulation frame rate and memory. The simulation frame rate includes all computation to produce the next graphical frame, except the rendering itself. The pre-processing times range from a minute for simple models to 20 minutes for the most complex trees. Intel Xeon 2x8 cores 2.9 GHz CPU, 32GB RAM. GeForce GTX 680, 2GB RAM. Models from Table 1 are shown in (1-indexed) (rows, columns) (1,4), (2,1), (8,3), (5,3), respectively. Selected animations are shown in the main video and Supplementary Material 1.*

are long and slender, and the running time of Dijkstra's algorithm is linear in the number of vertices of each domain, such a strategy produces well-distributed samples in negligible time.

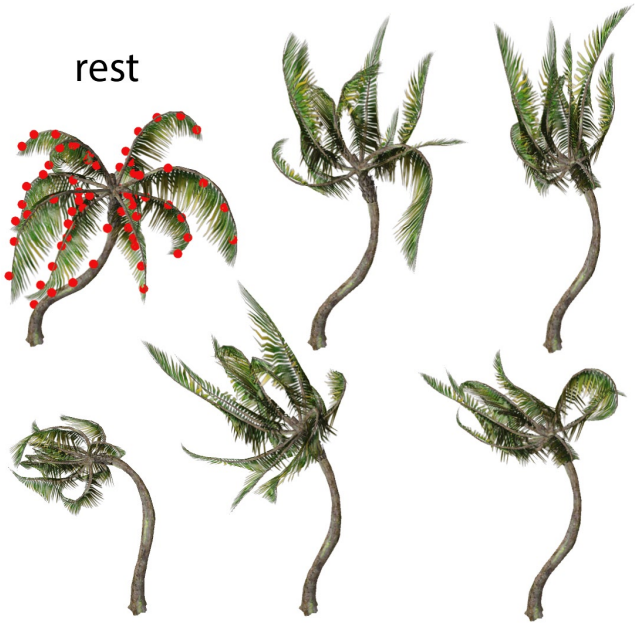

Figure 12: Palm tree (Cocos Nucifera) in strong Perlin wind: *Top-Left: the locations (in red) where the wind is sampled. Other images: selected animation frames. Simulation time: 4 msec per graphical frame. In this demo, palm leaves are not simulated, but are skinned to the palm branches. Each leaf is skinned entirely to one branch, with each vertex copying the displacement (in local branch frame) of the closest branch vertex in the undeformed configuration. Such skinning can greatly enrich the plant visual appearance at a minimal computational cost.*

Figure 13: Gravity: *Left: undeformed. Right: under gravity.*

Our system also supports plants loaded by gravity (Figure 13). Because gravity acts in a constant direction, it needs to be rotated into the frame of reference of each domain, $f_i^{\text{ext}} = U_i^T R_i f$, where U_i and R_i are the matrix specifying the low-dimensional space, and world-coordinate rotation of domain i, respectively, and $f = [0,\ -g,\ 0,\ 0,\ -g,\ 0,\ \ldots 0,\ -g,\ 0]^T$ is the gravity vector. Because f is constant, f_i^{ext} can be efficiently precomputed using the sandwich transform [Kim and James 2011], by evaluating f_i^{ext} for $R_i = e_k e_\ell^T$, where e_k is the k-th standard basis vector in $\mathbb{R}^3$, for all $k, \ell = 1, 2, 3$. At runtime, for each non-rigid domain, one then merely has to multiply a pre-computed $r_i \times 9$ matrix (r_i = #reduced DOFs of domain i) with the 9-vector of the entries of R_i, imposing a negligible overhead. A nonlinear optimization in the space of domain reduced coordinates could be employed to pre-load plants so that the input configuration is the rest configuration under gravity [Derouet-Jourdan et al. 2010; Twigg and Kačić-Alesić 2011].

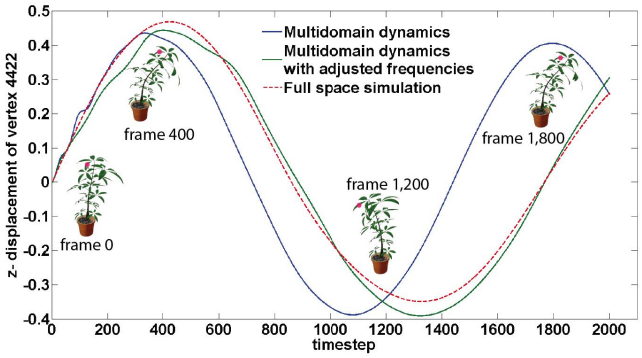

Figure 14: Comparison to full simulation. 91 flexible domains, 1183 modes, 11,396 triangles. *All three simulations visually look similar. The unscaled reduced simulation has a higher natural frequency, for two reasons: (1) reduced systems lack degrees of freedom and are typically somewhat stiffer than unreduced systems with equal material properties, and (2) interface lumping [Barbič and Zhao 2011] increases frequency, much like a pendulum with a shorter length oscillates faster. After scaling, the frequency and amplitude match full simulation closely.*

Comparison to full simulation: One practical approach to animate vegetation is to build a simulation mesh for the entire plant, and then timestep the dynamics using a deformable object simulator. In Figure 14, we compare our method to a geometrically nonlinear full simulation [Capell et al. 2002]. For this experiment, we generated a global tetrahedral mesh for the shefflera plant, and then manually subdivided it into domains. This process took 8 hours of work, whereas our pre-processing pipeline takes less than 10 minutes. In order to make the motion more natural, we made the stem 2x stiffer than the rest of the mesh. We then simulated the shefflera under an identical force load, material and simulation properties. In this example, our method (including time for $u = Uq$ displacement computation) is 23x faster than the full simulation. Visually, the two motions differ slightly in frequency, but appear qualitatively similar. If a close frequency match is desired, it is possible to scale the frequency spectrum of each domain so that the lowest frequency matches some externally prescribed frequency, such as frequency from full simulation, or real measurements of plants [James et al. 2006]. We illustrate this concept with a 1D harmonic oscillator, $m\ddot{x} + d\dot{x} + kx = f$, whose natural angular frequency (without damping) is $\omega = \sqrt{k/m}$. Suppose a different frequency $\omega' \neq \omega$ is desired. The modified mass and stiffness must satisfy $\alpha^2 k m' = k' m$, where $\alpha = \omega'/\omega$. If we impose an additional condition that the new oscillator attains the same maximum amplitude after a fixed initial impulse, we obtain $k' = \alpha k$ and $m' = m/\alpha$. With plants, we perform the same scaling, to match the lowest vibration frequency, separately for each domain. We obtain the desired frequency for each branch by creating a volumetric mesh for the entire subtree rooted at that branch, and computing (unreduced) mass and stiffness matrices M and K. The desired angular velocity is then the square root of the lowest eigenvalue of $Mx = \lambda Kx$, which we find using a sparse eigensolver [Lehoucq et al. 1997]. We scale the reduced mass matrix by $1/\alpha$ and internal elastic forces by α. The result is shown in Figure 14. We note that, instead of matching the lowest frequencies to full FEM simulation, in the future one could attempt to match them to real-world plant observations.

Rendering: We render the plants interactively using OpenGL. Billboards are in practice often partially transparent, with alpha values continuously ranging from zero to one. Therefore, one-pass alpha-testing results in noticeable aliasing, for example, at the leaf edges. We use two-pass rendering where we first render all the geometry with alpha-testing enabled, i.e., fragments where the alpha value is strictly less than 1.0 are discarded. We then make the depth buffer read-only, disable alpha-testing, enable alpha blending, and re-render all domains that use transparent texture maps. Note that in many models transparent textures represent a large fraction of the geometry, e.g., leaves, conifer needles, twigs. Our simulation is substantially faster than rendering; more optimized rendering pipelines could be designed [Sousa and Crytek 2007]. Before our models can be rendered, the displacements (in local domain frame of reference) of all the mesh vertices must be computed via the modal equation $u_i = U_i q_i$ [James and Pai 2002]. Their world-coordinate positions are then constructed using the domain's current position vector and rotation matrix. Our reported simulation timings include the time necessary to compute u_i and world coordinate triangle mesh vertex positions. We compute u_i on the CPU, but GPU implementations [James and Pai 2002] would be readily possible. We note that matrix U_i here contains the modes that were already interpolated (during pre-process) from the volumetric mesh to the plant triangular geometry, using barycentric interpolation. We found that such a strategy is usually faster than interpolating volumetric mesh displacements to the triangle mesh at runtime; the tradeoff depends on how finely the triangle meshes are tessellated. Offline renderings were performed using the Yafaray ray tracer.

8 Conclusion

We presented a system for stable physically based simulation of anatomically realistic botanical systems. We demonstrated a robust pipeline to pre-process "polygon soup" plant geometry for domain decomposition simulations. Our system scales to the complexity of real-world adult trees, flowers and bushes. We have pre-processed over 100 plants from several publicly available vegetation model libraries. Our system accommodates unorganized, unprocessed triangle input geometry, including billboards, and is enabled by recent advances in model reduction and domain decomposition. We support fracture, interactive plant design and frequency tuning.

Limitations and future work: Our system cannot handle plants that are in continuous contact with their natural environment, such as a vine climbing a mesh fence or ivy climbing a tree. For plants in simpler (ground) contact such as zucchini or watermelons, frictional contact could be handled using constraint solvers, or even penalty forces. Our system avoids loops, which has not been a problem in practice as the vast majority of plants do not have loops. Loops could be addressed using penalty forces. Our material parameters are tuned by the user to achieve a specific effect and the models are then simulated, but could in the future incorporate mechanical properties or other observation data from real plants. Our instancing can be easily extended to domains that consist of several disjoint components, each texture-mapped with a distinct texture map. A more challenging case, however, would be to support *hierarchical instancing*, where instances themselves consist of replicated copies, e.g., replicated blooms, each consisting of replicated but otherwise identical petals. We note, however, that many blooms in practice are modeled as a single global quadrilateral billboard, replicated in different orientations, where our single-level instancing is sufficient. Methods that use alternative simulation bases [Gilles et al. 2011] may be an interesting approach to simulate plants. We use a semi-implicit integrator to timestep our models which provides stability but also introduces artificial damping. It would be interesting to seek integration schemes that can work with model reduction and that can avoid artificial damping. Simulator realism would be improved by handling self-collisions, and simulating two-way coupling between the wind and the plant.

Acknowledgements: We thank Somya Sharma for help with plant authoring, Intel Corporation for donating two workstations to perform this research, and Zoran Kačić-Alesić and ILM for feedback on our system. This research was sponsored in part by the National Science Foundation (CAREER-53-4509-6600).

References

AKAGI, Y., AND KITAJIMA, K. 2006. Computer animation of swaying trees based on physical simulation. *Computers & Graphics 30*, 4, 529–539.

BARBIČ, J., AND JAMES, D. L. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. on Graphics 24*, 3, 982–990.

BARBIČ, J., AND ZHAO, Y. 2011. Real-time large-deformation substructuring. *ACM Trans. on Graphics 30*, 4, 91:1–91:7.

BARBIČ, J., DA SILVA, M., AND POPOVIĆ, J. 2009. Deformable object animation using reduced optimal control. *ACM Trans. on Graphics 28*, 3.

BEAUDOIN, J., AND KEYSER, J. 2004. Simulation levels of detail for plant motion. In *Symp. on Computer Animation (SCA)*, 297–304.

BERGOU, M., WARDETZKY, M., ROBINSON, S., AUDOLY, B., AND GRINSPUN, E. 2008. Discrete elastic rods. *ACM Trans. on Graphics 27*, 3, 63:1–63:12.

BERTAILS, F. 2009. Linear time super-helices. *Comput. Graphics Forum 28*, 2, 417–426.

BICKEL, B., BAECHER, M., OTADUY, M., MATUSIK, W., PFISTER, H., AND GROSS, M. 2009. Capture and modeling of non-linear heterogeneous soft tissue. *ACM Trans. on Graphics 28*, 3, 89:1–89:9.

CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. A Multiresolution Framework for Dynamic Deformations. In *Symp. on Comp. Animation 2002*, 41–48.

CORMEN, T. H., LEISERSON, C. E., AND RIVEST, R. L. 1990. *Introduction to Algorithms*. MIT Press/McGraw-Hill.

DEROUET-JOURDAN, A., BERTAILS-DESCOUBES, F., AND THOLLOT, J. 2010. Stable inverse dynamic curves. *ACM Trans. on Graphics 29*, 6, 137:1–137:10.

DEUSSEN, O., AND LINTERMANN, B. 2005. *Digital Design of Nature: Computer Generated Plants and Organics*. Springer-Verlag, New York.

DIENER, J., RODRIGUEZ, M., BABOUD, L., AND REVERET, L. 2009. Wind projection basis for real-time animation of trees. *Computer Graphics Forum 28*, 2, 533–540.

GILLES, B., BOUSQUET, G., FAURE, F., AND PAI, D. K. 2011. Frame-based elastic models. *ACM Trans. on Graphics 30*, 2, 15:1–15:12.

HABEL, R., KUSTERNIG, A., AND WIMMER, M. 2009. Physically Guided Animation of Trees. *Computer Graphics Forum 28*, 2, 523–532.

HADAP, S. 2006. Oriented strands: dynamics of stiff multi-body system. In *Symp. on Computer Animation (SCA)*, 91–100.

HU, S., CHIBA, N., AND HE, D. 2012. Realistic animation of interactive trees. *The Visual Computer 28*, 859–868.

INTERACTIVE DATA VISUALIZATION, 1999. Speedtree. www.speedtree.com.

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

JAMES, D. L., BARBIČ, J., AND TWIGG, C. D. 2004. Squashing Cubes: Automating Deformable Model Construction for Graphics. In *Proc. of ACM SIGGRAPH Sketches and Applications*.

JAMES, K. R., HARITOS, N., AND ADES, P. K. 2006. Mechanical stability of trees under dynamic loads. *American J. of Botany 93*, 10, 1522–1530.

JAMES, D. L., TWIGG, C. D., COVE, A., AND WANG, R. Y. 2007. Mesh Ensemble Motion Graphs: Data-driven Mesh Animation with Constraints. *ACM Trans. on Graphics 26*, 4.

KIM, T., AND JAMES, D. 2011. Physics-based character skinning using multi-domain subspace deformations. In *Symp. on Computer Animation (SCA)*, 63–72.

LABELLE, F., AND SHEWCHUK, J. R. 2007. Isosurface Stuffing: Fast Tetrahedral Meshes with Good Dihedral Angles. *ACM Trans. on Graphics 26*, 3, 57:1–57:10.

LEHOUCQ, R., SORENSEN, D., AND YANG, C. 1997. ARPACK Users' Guide: Solution of large scale eigenvalue problems with implicitly restarted Arnoldi methods. Tech. rep., Comp. and Applied Mathematics, Rice Univ.

LIN, M. C., AND GOTTSCHALK, S. 1998. Collision Detection Between Geometric Models: A Survey. In *Proc. of IMA Conference on Mathematics of Surfaces*, 37–56.

LINTERMANN, B., AND DEUSSEN, O. 1999. Interactive modeling of plants. *IEEE Comp. Graphics and Applications 19*, 1, 56–65.

LONGAY, S., RUNIONS, A., BOUDON, F., AND PRUSINKIEWICZ, P. 2012. Interactive procedural modeling of trees on a tablet. In *Proc. of Eurographics Symp. on Sketch-Based Interfaces and Modeling*, 107–120.

LU, H., GUO, X., ZHAO, C., AND LI, C. 2011. Physical model for interactive deformation of 3d plant. *International Journal of Virtual Reality 10*, 2, 33.

MARTIN, S., KAUFMANN, P., BOTSCH, M., GRINSPUN, E., AND GROSS, M. 2010. Unified simulation of elastic rods, shells, and solids. *ACM Trans. on Graphics 29*, 4, 39:1–39:10.

MEZGER, J., THOMASZEWSKI, B., PABST, S., AND STRASSER, W. 2008. Interactive physically-based shape editing. In *Proc. of the ACM Symp. on Solid and physical modeling*, 79–89.

MÜLLER, M., AND CHENTANEZ, N. 2011. Solid simulation with oriented particles. *ACM Trans. on Graphics 30*, 4, 92:1–92:10.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless Deformations Based on Shape Matching. *ACM Trans. on Graphics 24*, 3, 471–478.

OTA, S., TAMURA, M., FUJIMOTO, T., MURAOKA, K., AND CHIBA, N. 2004. A hybrid method for real-time animation of trees swaying in wind fields. *The Visual Computer 20*, 613–623.

PARKER, E. G., AND O'BRIEN, J. F. 2009. Real-time deformation and fracture in a game environment. In *Symp. on Computer Animation (SCA)*, 156–166.

PENTLAND, A., AND WILLIAMS, J. 1989. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics (Proc. of ACM SIGGRAPH 89) 23*, 3, 215–222.

PERLIN, K. 2002. Improving Noise. *ACM Trans. on Graphics 21*, 3, 681–682.

PIRK, S., STAVA, O., KRATT, J., SAID, M. A. M., NEUBERT, B., MĚCH, R., BENES, B., AND DEUSSEN, O. 2012. Plastic trees: interactive self-adapting botanical tree models. *ACM Trans. on Graphics 31*, 4, 50:1–50:10.

PRUSINKIEWICZ, P. 1986. Graphical applications of l-systems. In *Graphics Interface / Vision Interface*, 247–253.

SAKAGUCHI, T., AND OHYA, J. 1999. Modeling and animation of botanical trees for interactive virtual environments. In *Proc. of the Symp. on Virtual reality software and technology*, 139–146.

SELINO, A., AND JONES, M. D. 2012. Large and Small Eddies Matter: Animating Trees in Wind Using Coarse Fluid Simulation and Synthetic Turbulence. *Comp. Graphics Forum 32*, 1, 75–84.

SIFAKIS, E., AND BARBIČ, J. 2012. FEM Simulation of 3D Deformable Solids: A practitioner's guide to theory, discretization and model reduction, Part 2: Model reduction. In *SIGGRAPH Course Notes*. www.femdefo.org.

SOUSA, T., AND CRYTEK. 2007. *GPU Gems 3, Chapter 16. Vegetation Procedural Animation and Shading in Crysis*. Addison-Wesley Professional, Boston.

STAM, J. 1997. Stochastic Dynamics: Simulating the Effects of Turbulence on Flexible Structures. *Comp. Graphics Forum 16*, 3, 159–164.

TURBOSQUID, 2000. www.turbosquid.com.

TWIGG, C., AND KAČIĆ-ALESIĆ, Z. 2010. Point cloud glue: constraining simulations using the procrustes transform. In *Symp. on Computer Animation (SCA)*, 45–54.

TWIGG, C., AND KAČIĆ-ALESIĆ, Z. 2011. Optimization for sag-free simulations. In *Symp. on Computer Animation (SCA)*, 225–236.

WEBER, J. P. 2008. Fast simulation of realistic trees. *IEEE Computer Graphics and Applications 28*, 3, 67–75.

WONG, J. C., AND DATTA, A. 2004. Animating real-time realistic movements in small plants. In *Proc. of GRAPHITE 2004*, 182–189.

XFROG, 2009. www.xfrog.com.

YANG, Y., XU, W., GUO, X., ZHOU, K., AND GUO, B. 2013. Boundary-aware multi-domain subspace deformation. *IEEE Trans. on Visualization and Computer Graphics, to appear*.

ZHANG, L., SONG, C., TAN, Q., CHEN, W., AND PENG, Q. 2006. Quasi-physical Simulation of Large-Scale Dynamic Forest Scenes. In *Advances in Computer Graphics*, Springer, vol. 4035 of *Lecture Notes in Computer Science*, 735–742.

ZHANG, L., ZHANG, Y., JIANG, Z., LI, L., CHEN, W., AND PENG, Q. 2007. Precomputing data-driven tree animation. *Computer Animation and Virtual Worlds 18*, 4-5, 371–382.

ZHANG, L., ZHANG, Y., CHEN, W., AND PENG, Q. 2008. Real-time simulation of large-scale dynamic forest with gpu. In *IEEE Conf. on Circuits and Systems*, 614–617.