An interactive forest

Thomas Di Giacomo
iMAGIS-GRAVIR (joint project CNRS-INRIA-INPG-UJF)

Stéphane Capo
Infogrames Interactive SA

François Faure
iMAGIS-GRAVIR (joint project CNRS-INRIA-INPG-UJF)

Abstract. We present a prototype of a forest in which a video game player can move and interact physically with the trees.
The trees are procedurally built on-the-fly at each redraw. Two animation approaches are combined: a procedural method which handles most of the trees efficiently, and a physically-based method which allows user interaction with the trees. The physically-based method is dynamically applied only where needed. Physical data is computed only where the physical method is applied, and deleted afterwards. Smooth transitions between animation methods are performed.
Levels of detail are used for rendering and for procedural animation. Our method allows the display and the animation, including user action, of a 256-tree forest at interactive rates.

1 Motivation

We address the problem of displaying and animating large scenes with vegetation in the context of a collaboration with a video game company namely Infogrames. More specifically, our aim is to model a forest in which the player can move and interact with the scene, e.g. grab a branch and shake the tree. Such an application has been a challenging problem in computer graphics due to geometric and dynamic complexity. Rendering and animating a single tree may require thousands of polygons and degrees of freedom, or more. The complexity of a forest may thus be practically intractable. However, level-of-detail (LOD) approaches can help us adapt the complexity of the elements according to their visual importance. A tree in the foreground may be modeled using ten thousands polygons, while a tree in the background may be modeled using only a few hundred polygons.

In applications with a fixed point of view, the geometric and dynamic elements can be modeled offline, provided that we are able to refine them according to their visual contribution. The scene is then rendered and animated using standard techniques. The problem is more difficult when the point of view changes over time, because the LODs have to be dynamically adapted. We thus have to perform smooth transitions between LODs to avoid "popping" effects. This applies to geometry as well as animation.

When using standard graphics hardware acceleration, rendering complexity mainly depends on the number of polygons and the amount of texture data. In contrast, animation complexity is not only a function of the number of degrees of freedom in the model, it also depends on the method used. Procedural methods can be fast but do not guarantee physical plausibility, especially when interactions occur. In contrast, physically-based

methods generate more realistic motion and allow interactions but they are generally compute-intensive.

In this paper, we present our approach for animating and rendering a forest. We mainly focus on animation. The geometric model is procedurally computed on-the-fly at each refresh using a small set of parameters which entirely define each individual tree. This results in a very compact geometrical data set. A procedural method is used to animate the trees at different levels of detail to model the action of wind. A simplified dynamic model of tree is used to handle user interaction. Both animation methods are closely related to well-known approaches. Our main contribution is to combine them so that physically-based animation is used only where needed. Physical data is dynamically generated to handle interaction and deleted thereafter. Transitions between procedural and physical animation are performed to avoid popping. Combined with our procedural LODs, the method allows the animation of an interactive forest made of a few hundreds of trees with industrial video game quality (textures, lights, high frame rates).

The remainder of the paper is organized as follows. In section 2 we present previous work on related topics. In section 3 we outline our geometric modeling method. In section 4 we present our animation method. Results are shown in section 5. We finally conclude and discuss future work in section 6.

2 Related Work

2.1 LOD for animation

As for geometry, animation techniques can be adapted and simplified in certain cases i.e., when motions are too fast, too far away, or too numerous for human sight, [Ber97], or when motions are of low interest and do not need complex calculations. Though [GCB95] applied this idea to procedural animation by decreasing sampling frequency of motions, and degrees of freedom of human articulated figure, it seems more appropriate to apply it to physically-based animation because of its high cost in computations. To simplify or refine motions, adaptive models must be created, and smooth transitions between their levels of detail ensured [Val99].

One can mix different models, or construct a single multiresolution model. Multiple models are used by [CH97] to simulate monopodes with three different models, dynamic, kinematic and single particle motion, where transitions are possible only at certain times. [PC01] also simulates a prairie with three models, 3D blade of grass, texels and 2D textures, by interpolating 3D motions to match the texel when needed.

Methods working on a single multiresolution model are mainly physically-based approaches : [HPH96] and [HH98] refine mass-spring networks, [DDBC99] and [DDCB01] proposed a multiresolution FEM, the dynamic multiresolution response is compared to other models, to simulate deformations of a liver.

In this paper we present a hybrid method which uses two different models, the procedural part being multiresolution.

2.2 Wind, trees

Forests with blowing winds have been studied by [SF92] and [Sta96], both using stochastic vector fields for the wind, and a modal analysis of beams vibrations for the response of the trees. However, [Sta96] overrides the integration by directly synthetising the motions. We could not apply such a method because of our need for interactions.

We generate wind with the same kind of primitives as [PC01], that is to say 2D masks containing a vector field, with velocity and influence area.
We also make simplifying assumptions similar to [Ono97] on the dynamics of the tree : the density of the tree is uniform, no collisions occur between branches or leaves, branches move by bending, not torsion.

3 Geometric model

Our trees are made of skeleton nodes defining topology and meshes defining geometry. Skeleton nodes define the topology, lengths and angles of the branches. Figure 1 shows the geometric parameters (h, θ, ϕ) used to define the position of a branch with respect to its parent. The reference frame of the child is first translated by h and rotated by θ along the z axis of the parent. A rotation of angle ϕ along the axis y of the child is then applied. Meshes define branch shapes and textures. In order to reduce the geometric

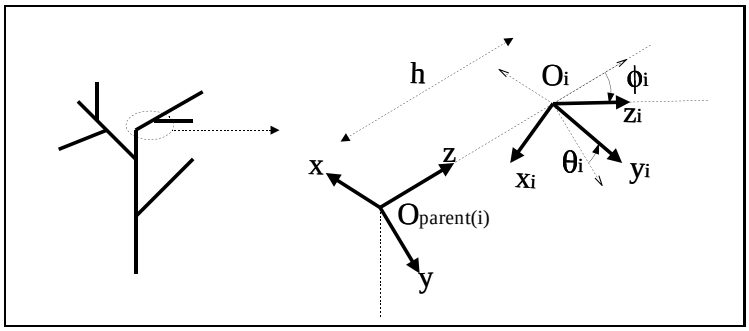

Fig. 1. A branch i positioned with respect to its parent.

data set, skeleton nodes and meshes are recomputed procedurally at each screen refresh using small sets of parameters which model seeds. The set of seeds totally defines the forest and is the only permanent data. Compared with progressive mesh approaches used in other applications, our approach has the following advantages:

- we can generate a whole forest, with all different trees, at a very low memory cost;
- the topological and geometrical information allows us to simplify topology as well as shape;
- smoothness is not limited by a reference model and can be arbitrarily high.

The additional computation cost is partially balanced by the reduction of data flow. Moreover, an increasing number of platforms (Playstation 2, recent PC cards) use specialized processors for mesh generation, which saves a lot of computation time.

Our goal is to generate visually convincing trees for games rather than botanical applications. We have thus made the following choices :

- The skeleton generation is controlled by parameters given by the game designer through a graphic interface (size, number of children, number of leaves,...).
- Simple rules found from real tree observation are used by the system and can not be altered. For example, the maximum angle between a branch and its mother is a function of their relative diameters.

- The number of parameters is as small as possible. This makes modeling easier and reduces the data needed to model an entire unique tree to less than one kilobyte.
- Perlin noise is used to obtain a more natural look.

We adjust the LODs according to the size of the elements projected to the screen. The LODs are modulated by a global factor which is a function of the total complexity of the scene at the previous frame. This feedback mechanism allows us to approximately set the global complexity to an arbitrary level.

4 Tree animation

We animate the trees by controlling the angles θ and ϕ (see fig. 1) for each branch. In this section, we first present the procedural animation method, then the physically-based method, and finally the combined method.

4.1 Procedural animation

Our procedural motion is a function of wind primitives traversing the scene. We first describe the wind primitives, the equations of motion, and how we apply levels of detail to procedural motion.

Wind. A wind primitive (see fig. 2) is defined by an area of influence (disk (C, r)), a force vector F and a pulsation ω. Each branch of a tree inside the area of influence

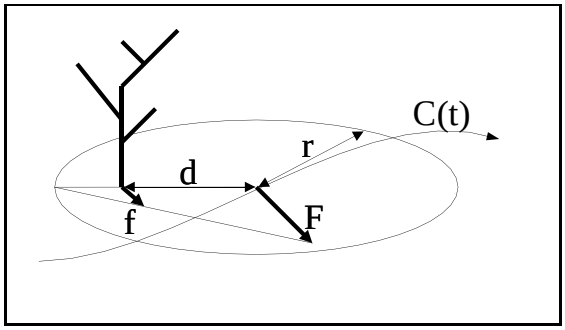

Fig. 2. A wind primitive applied to a tree.

undergoes a force f given by:

$$f = \frac{r-d}{r} sin(\omega t) F$$

if $d < r$, zero elsewhere. Phase angle is omitted for simplicity. The disk can move along a given trajectory $C(t)$. Such primitives allow us to test our framework. In future work we will implement more sophisticated functions.

Response to wind. Our procedural method is not physical since it is not based on the time integration of an acceleration. However, it is designed to ease the transitions to the physically-based method. Each wind force applied to a branch creates a torque at its base, which is responsible for angular motion. We approximate the torque by:

$$\tau(t) = Lz \times f(t)$$

where L is the length of the branch and z its axis. The torque is then projected to the rotation axes to obtain a rotation amplitude. The motion due to a wind primitive is thus given by

$$\begin{aligned} \theta &= \theta_0 + \frac{1}{m}\tau(t) z_{parent} \\ \phi &= \phi_0 + \frac{1}{m}\tau(t) y \end{aligned}$$

where z_{parent} and y are the axes of rotation of the branch with respect to its parent (fig. 1). The term m is an approximation of the inertia of the branch(see section 4.2). The action of the wind thus acts as a force integrated twice. By summing the motions due to all the primitives applied to a given branch we obtain:

$$\theta = \theta_0 + \frac{1}{m} z_{parent} \sum_i \tau_i(t) \quad (1)$$

$$\phi = \phi_0 + \frac{1}{m} y \sum_i \tau_i(t) \quad (2)$$

Angles are directly set to the value of the torque applied. The wind direction is taken into account by torque projections, along with force magnitude and pulsation.

Levels of detail. The animation LODs have been developed independently of rendering. For now, the level of detail is related to recursion depth rather than screen size, this corresponds to the transition between dark and light grey levels of the fig. 3. Branch joints beyond the given level are fixed. This level can be dynamically set for each tree independently, according to eye distance and to the presence of wind. Transitions are implemented by linear interpolation over a time interval $[t_0, t_1]$ as follows:

- fixed to mobile : $\theta = \alpha\tilde{\theta}$ with $\alpha = \frac{t-t_0}{t_1-t_0}$
- mobile to fixed : $\theta = \alpha\tilde{\theta}$ with $\alpha = \frac{t_1-t}{t_1-t_0}$

where $\tilde{\theta}$ is the response to wind actions and θ the angle actually applied. At any time during the transition, we can switch back to the previous LOD by reversing the evolution of α.

4.2 Physically-based animation

Physically-based animation uses physical data such as force, mass and stiffness. Though the procedural model is not designed to generate such data, we build it using the geometrical data available. Here we first briefly explain how to build a physical model using the procedurally generated geometrical data. We then present a simplified method for the physical animation.

Physical model. As previously mentioned, no physical data is kept permanently. We derive on-the-fly mass, stiffness and damping using geometrical data. The forces involved are caused by the wind, by the user's action and by the joints.

In contrast to procedural animation, we now consider wind action as a real force. To vary continuously from procedural to physical animation, we apply the second time derivative of wind action. Using our current wind primitives, we get:

$$\tau_{physical} = -\omega^2 \frac{r-d}{r} sin(\omega t) Lz \times F$$

The motion due to the wind is thus theoretically the same as the motion computed procedurally. In practice, due to explicit time integration, divergence would eventually occur if joint damping were not applied.

User action is modeled as an external force applied to a branch. This force is handled similarly to wind force.

The torques applied by the joints are modeled using linear damped angular springs. The stiffness k and damping ν are estimated from branch diameter and length. In the theory of linear elasticity, stiffness is related to the inverse of the section area. For a given force, the displacement at the end of a branch is proportional to its length. We thus set k proportional to d^2/l where d is the average branch diameter and l the length. We make the common assumption that damping is proportional to stiffness. The torque generated by a given joint is given by:

$$(k\theta + \nu\dot{\theta})z_p + (k\phi + \nu\dot{\phi})y$$

Mass is closely related to volume. We thus set the term m proportional to ld^2. In future work we will investigate the use of l^3d^2, which models the inertia of a bar.

Equations of motion. In order to save computation time, we apply simplified dynamics and consider only rotations. The laws of dynamics give:

$$\begin{aligned} \tau &= J\dot{\Omega} + \Omega \times J\Omega \\ &\simeq m\dot{\Omega} \end{aligned}$$

where τ is the net torque applied to a body, Ω its angular velocity in world coordinates, $\dot{\Omega}$ its angular acceleration and J its inertia matrix. The second equation is derived using the simplifying assumption $J \simeq mI$ where I is the identity matrix.

The net torque applied to a given branch is the sum of the actions of the wind and joint forces:

$$\begin{aligned} \tau = & -\sum_{i \in winds} \omega_i^2 \tau_i \\ & -(k\theta + \nu\dot{\theta})z_p - (k\phi + \nu\dot{\phi})y \\ & + \sum_{j \in children} (k_j\theta_j + \nu_j\dot{\theta}_j)z - (k_j\phi_j + \nu_j\dot{\phi}_j)y_j \end{aligned}$$

where subscript p denotes the parent of the branch. We then project the angular acceleration of the body with respect to its parent to the two rotation axes to obtain the angular

joint accelerations:

$$\begin{aligned} \dot{\Omega}_{rel} &= \frac{1}{m}\tau - \dot{\Omega}_p \\ \ddot{\theta} &= z_p \dot{\Omega}_{rel} \\ \ddot{\phi} &= y \dot{\Omega}_{rel} \end{aligned}$$

Time integration is performed using the standard Euler method:

$$\begin{aligned} \dot{\theta}(t+dt) &= \dot{\theta}(t) + \ddot{\theta}(t)dt \\ \theta(t+dt) &= \theta(t) + \dot{\theta}(t)dt \end{aligned}$$

and similarly for ϕ. Note that position and velocity must be stored frame to frame to perform time integration. This results in a temporary memory overhead of four values per branch animated physically.

4.3 Hybrid animation

We use hybrid animation to combine wind influence and user action such as branch grabbing. We first show how the two methods can coexist within the same tree. We then explain how transitions are performed.

Coexistence of the two methods. Procedural animation is computed by applying equations (1) and (2), which do not depend on how the other branches are animated. Procedural animation can thus be trivially applied within a mixed animation. Physically-based animation is based on net force computation. Spring forces depend on object positions and velocities. For each branch animated physically, we thus need to know the position and velocity of the neighboring branches. If a neighboring branch is animated physically, then its velocity is a part of its dynamic state as well as position. If it is animated procedurally, we compute its velocity by finite differentiation $\Delta x/\Delta t$. Figure 3 illustrates how methods coexist within a tree.

Transitions between the two methods. Transition from procedural to physical animation is done instantaneously using the positions computed procedurally and the velocities deduced. However, an instantaneous transition from physical motion to procedural would generate a popping effect since position is procedurally computed as a function of time, without considering previous position and velocity. We thus compute both motions and perform a smooth blending during a time interval, typically one and half a second. Transition starts at time t_0 and ends at time t_1. We use a parameter α smoothly varying from 0 to 1 during the given time interval:

$$\begin{aligned} u(t) &= \frac{t - t_0}{t_1 - t_0} \\ \alpha(t) &= 3u^2 - 2u^3 \\ \theta(t) &= (1-\alpha)\bar{\theta}(t) + \alpha\tilde{\theta}(t) \\ \phi(t) &= (1-\alpha)\bar{\phi}(t) + \alpha\tilde{\phi}(t) \end{aligned}$$

where the tilde denotes motion computed procedurally and the bar denotes motion computed physically. At any time, we can switch back to physically-based motion by reversing the evolution of α.

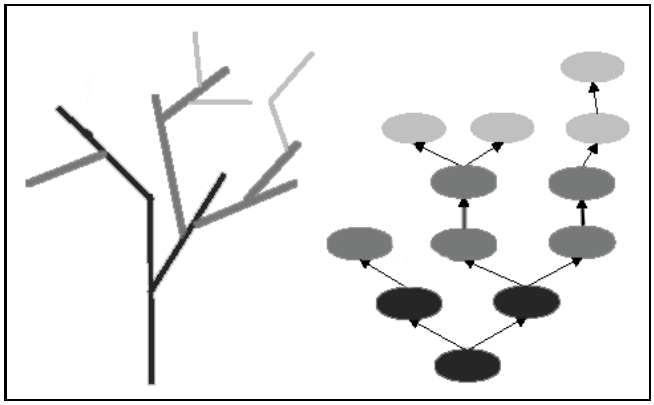

Fig. 3. Hybrid animation. The branches at the bottom (black) are animated physically. The intermediate levels (dark grey) are animated procedurally. The higher levels (light grey) are fixed.

5 Results

The experiments were made on a PIII 800MHz with 512MO RAM computer, with a NVIDIA GeForce256 board using OpenGL.

- Procedural animation of a tree with level of details. The motion of the tree is computed with respect to the direction, the magnitude and its position in the area of influence of the winds (these areas are the red circles of fig. 4a). We can simplify or refine the depth level in which branches are animated (red branches on fig. 4a), ensuring smooth transitions (shown in green). The transitions can be adapted to gain smoothness by increasing their durations.
- Procedurally animated forest (fig. 4b, 4c). We choose the following criteria for individual LODs : if the tree is not under any wind influence, its level is set to zero and the tree does not move. Levels are refined depending on the number of influent windfields, the distance to the camera and the position of the tree in the area of influence. Fig. 4c shows that we obtain interactive frame rates (27fps) using LODs, while it was not the case without LODs (around 1fps on the same scene).
- Physically-based animation of a single tree. All branches are physically handled, showing that they come back to their original positions after a gust of wind. This experiment is mainly done to calibrate our physic parameters to respond to the wind.
- Hybrid animation of a tree. We combine the two method : the multiresolution procedural method and the physically-based method. Continuity is achieved when switching the motion of a branch from procedural to physics, and from physics to procedural.
- Force and wind applied to a tree. The user applies an external force to a selected branch (yellow branch on fig. 4d at rest, and 4e pulled) which is pulled to the right while the tree is influenced by the wind. The physical LOD is then set, at least, at the same depth level as the branch grabbed. The motion is therefore the combination of the action of the wind and the applied force.

- Full forest. Finally, we animate a forest including 256 trees (fig. 4f), in which physics is applied where needed. We can select any branch of the scene and interact with it.

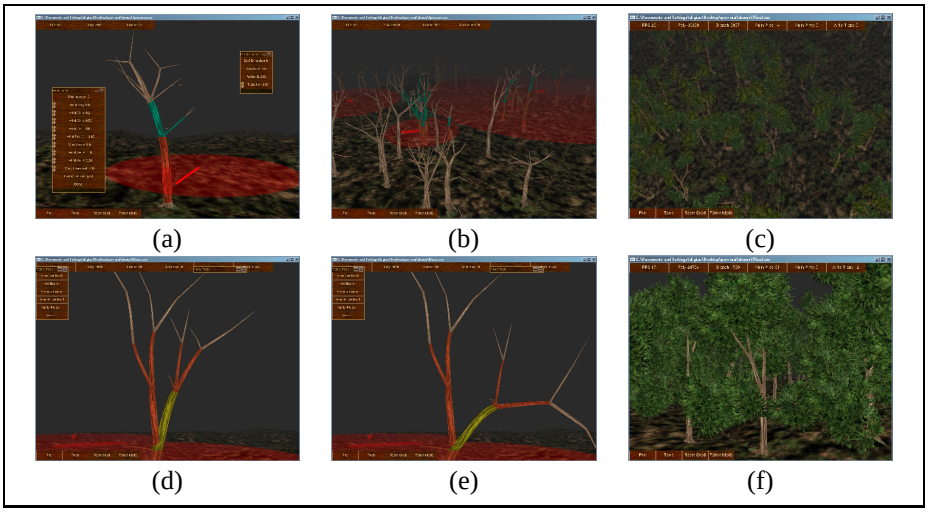

Fig. 4. Results.

6 Discussion

We have presented a prototype of an interactive forest. It is made of trees built on-the-fly at each redraw using a small set of parameters defining position, topology and shape. This method reduces memory requirements and allows the fine tuning of the levels of detail. We have developed a new animation method well-suited for this approach. It combines a fast procedural method including levels of detail with a physically-based model used to handle interactions. Transitions between the animation methods can be performed at any time. The physical method is dynamically applied only where needed. As a result, high frame rates are obtained as well as interactivity.

Additional development is necessary for practical use. The partitioning of the tree according to the animation method applied needs be more finely tuned than our current level-based approach. We will enhance the procedural and physically-based animation methods. More sophisticated wind primitives will be implemented. Implicit integration will be applied to stiff objects. We also plan to investigate noise functions to treat leaf animation in addition to branches. Collision detection will also be included.

From a more fundamental point of view, we will further investigate how procedural and physically-based methods can coexist. In our current implementation, physical branches are parents of procedural branches. The opposite would be useful also, e.g. if a bird lands on a small branch we may not need to animate all parent branches physically, but only those really influenced by the bird. More generally, we need a criterion to determine which branch needs to be animated physically. We also plan to apply LODs to the physically-based animation, probably using sub-tree clustering.

This work is partly funded by the PRIAMM project called ”Modèles multi-échelles de végétation animée et interactive pour le jeu vidéo”.

References

Ber97. R. Berka. Reduction of computations in physic-based animation using level of detail. In *Spring Conference of Computer Graphics*, 1997.

CH97. D. Carlson and J. Hodgins. Simulation levels of detail for real-time animation. In *Graphics Interface*, 1997.

DDBC99. G. Debunne, M. Desbrun, A. Barr, and M.-P. Cani. Interactive multiresolution animation of deformable models. In *Computer Animation and Simulation*, Sep. 1999.

DDCB01. G. Debunne, M. Desbrun, M.-P. Cani, and A. Barr. Dynamic real-time deformations using space and time adaptive sampling. To appear in the SIGGRAPH'01 conference proceedings, 2001.

GCB95. J. P. Granieri, J. Crabtree, and N. I. Badler. Production and playback of human figure motion for visual simulation. *Modeling and Computer Simulation*, 5(3):222–241, 1995.

HH98. P. Howlett and W. T. Hewitt. Mass-spring simulation using adaptive non-active points. *Computer Graphics Forum*, pages 345–354, 1998.

HPH96. D. Hutchinson, M. Preston, and T. Hewitt. Adaptative refinement for mass-spring simulations. In *Eurographics Workshop on Animation and Simulation*, pages 31–45, Sep. 1996.

Ono97. H. Ono. Practical experience in the physical animation and destruction of trees. In *Eurographics Workshop on Animation and Simulation*, pages 149–159, Sep. 1997. Industrial Light and Magic.

PC01. F. Perbet and M.-P. Cani. Animating prairies in real-time. In *Symposium on Interactive 3D Graphics*, 2001.

SF92. M. Shinya and A. Fournier. Stochastic motion-motion under the influence of wind. *j-CGF*, 11(3):C119–C128, C469, sept 1992.

Sta96. J. Stam. Stochastic dynamics: Simulating the effects of turbulence on flexible structures. Technical Report RR-2847, INRIA Rocquencourt, March 1996.

Val99. Bernard Valton. *Gestion de la complexite de scenes animees et interactives : contributions a la conception et a la representation.* PhD thesis, Univ. de Rennes(FR), 1999.