Please see the color plate on page 285.

JEcoSys - A Framework for Interactive Plants Simulation

Luis Carlos Yano Endo*
Instituto de Matemática e Estatística,
Universidade de São Paulo,
São Paulo - SP - Brazil

Carlos Hitoshi Morimoto†
Instituto de Matemática e Estatística,
Universidade de São Paulo,
São Paulo - SP - Brazil

Antonio Elias Fabris‡
Instituto de Matemática e Estatística,
Universidade de São Paulo,
São Paulo - SP - Brazil

Figure 1: Interation on a virtual ecosystem

ABSTRACT

This paper introduces JEcoSys, an extensible framework developed in Java to generate interactive simulations of plants. JEcoSys uses a simple physical model of plants, based on dynamic constraints, and pre-computations to achieve realism and efficiency during animation. The framework also includes a force field based technique to model the interaction of plants with different agents, that can simulate natural phenomena such as wind and rain. Experimental results show that JEcosys is able to simulate complex scenes containing hundreds of tufts of grass and flowers in real-time.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Hierarchy and geometric transformations, Physically based modeling; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; C.3 [Special-purpose and application-based systems]: Real-time and embedded systems

Keywords: Computer graphics, Hierarchy and geometric transformations, Physically based modeling, Interaction techniques, Animation, Interactive systems

1 INTRODUCTION

Interactive and realistic animation of complex scenes and natural phenomena is still a very challenging problem due to the conflict of requirements of interactive systems and realistic animation. Interactive systems require the application to be responsive to the user's actions, however the computational cost of realistic rendering and animation of complex scenes in real-time is still beyond the capacity of current machines.

In this paper we introduce JEcoSys, a framework that allows the rapid development of interactive, fast and realistic simulations of plants. This framework can be used by games, biological simulators, and virtual reality systems. Plants scenes are very complex to model, render and simulate. For example, the difficulty in the animation of a forest scene is not only due to the number of elements that must be rendered, but also due to the complexity of each plant and their interaction with natural phenomena and other agents.

We use a simplified physical model of plants, based on dynamic constraints, to achieve realism during animation. This plant model is very flexible and has been used to model prairies, flowers and little bushes, but it can be easily extended to other kinds of plants. Efficient rendering and animation are obtained from pre-computing several properties of the plant model. JEcoSys allows the user to change such properties to create different looking plants based on the same model.

The framework also includes a force field based technique that allows any agent to interact with plants spread over a large region. This technique allows the real-time simulation of natural phenomena, such as wind and rain for example, over a large region containing hundreds of plants.

The next section describes other techniques for plant modeling and simulation, and discusses their applicability to interactive systems. Section 3 introduces the Force field technique, and shows how it has been used to model wind interaction with plants. Section 4 describes how the framework handles the modeling, interaction, animation and distribution of plants. Implementation issues and experimental results are presented in Section 5. Section 6 discusses advantages and limitations of this framework and future work.

2 RELATED WORK

Several techniques were developed for modeling and animation of plants with outstanding realism [4, 7], although computationally too expensive for interactive systems, requiring several minutes or hours to render complex scenes.

Particles systems are commonly used to model natural phenomena, including plants [10], which are modeled by the path of several particles. Complex scenes would require thousands of particles, which makes the animation very expensive.

Level of details (LOD) techniques and pre-computations have been used in order to improve performance. The idea is to use more realistic models to render objects that are closer to the viewer, while less detailed, yet more efficient models, are used for rendering dis-

*e-mail: lye@ime.usp.br
†e-mail: hitoshi@ime.usp.br
‡e-mail: aef@ime.usp.br

Proceedings of Computer Graphics International 2005 (CGI'05)
June 22-24, 2005, Stony Brook, NY, USA
0-7803-9330-9/05/$20.00 ©2005 IEEE

tant objects. Points and lines representations are a very popular approach on LOD algorithms, due to their efficiency of rendering complex scenes comparing with Splines or even triangles. Deussen *et al.* [3] use this technique to model plants with interactive rates.

Image based algorithms, such as billboards, are used to model complex objects by approximation. A pre-generated image or a picture of the real object is mapped onto a tridimensional structure as a texture. This technique is used in [8, 9] to model and animate plants at interactive rates.

Physically based techniques [12] are known for producing very realistic animations, as well as, for being very computationally expensive. One of such techniques that is particularly relevant for this work is Dynamic Constraints [2]. This technique creates constraint forces that act on system's objects until some defined constraints are satisfied, and when this occurs they try to keep them satisfied. Section 4.2 describe a simpler extension that considerably enhances the performance of this model.

3 Force Field: Real-time interaction technique for complex scenes

The animation of complex scenes, such as those containing hundreds of plants, by interaction are computationally very expensive, thus very hard to implement in real-time. Most current real-time solutions trade realism for efficiency. Since vegetation is in general not part of the foreground, little or no resources can be allocated to animate them. Nevertheless, background animation considerably enhances the realism of the scene.

In [6] a Force field was modeled as a volumetric mesh that divide the space into regions. All objects inside a region receive the action of the force defined there. The interaction is calculated by the action of external forces. Many agents can act over those regions, changing the resultant force there, and the objects receive the resultant force in each region. This generated poor representation on very populated regions while inefficient one on isolated regions.

Next we present a more efficient algorithm that uses a similar representation for the force field but with a LOD algorithm. Section 3.2 presents a wind model and shows how to adapt it to act over the force field.

3.1 LOD Force Field

A LOD technique for the Force field was created to enhance the efficiency and realism. In this new approach, regions that are more populated or closer to the viewer receive interaction a large number of force vectors, defining a more detailed interaction. Furthermore, few forces are considered for distant or less populated regions, which implies on more efficient interaction.

3.1.1 LOD Technique

The data structure used for the Force field's information storage is a Quadtree. With this structure we are able to create child nodes, that are represented by smaller volumes, only for regions where a greater precision is required.

The algorithm for the quadtree creation considers an initial big region, that represents all scene. For each object in the scene four new children nodes are created if the node, that this object should be, is already occupied by a configurable number of objects. In this case, the objects that belongs to the node that was composed in four should be distributed between the children nodes. These children node must have $\frac{1}{4}$ of the volume of the original one. This is done on the pre-computation phase, however new objects can be easily added later.

After the Force field creation it is considered that all the objects inside the same node will receive the action of the same force vector defined by the node.

Using this LOD algorithm we can animate regions with more objects using more vectors. However there can be regions with many vectors that are not even in the view frustum of the viewer.

A distance LOD algorithm was also developed to improve the technique's performance. This algorithm calculates more precisely the interaction on regions closest to the viewer, disconsidering non visible nodes.

The quadtree structure aids this algorithm, since parent nodes (regions) can represent four or more children nodes. Thus, $4^h - 1$ interaction operations can be spared, where h is the difference of height between the parent node used to represent the children.

The LOD algorithm first verifies if the node is visible, and if it is defines which level of the tree will define the action over this area. If the node is far from the viewer, parent nodes can represent larger areas with less details. An array with the distance ranges defines how many level are disconsidered according to the viewer's distance. After this algorithm the Force field is represented only by visible cells, where some cells may be representing several others.

Figure 2 shows the Force field with nodes painted in different gray scale. Each gray scale defined represents a different height of nodes in the tree, the lighter gray scale represents the leaves, the next lighter represents the parent of the leaves, and so on. The figure shows several regions represented by their parents (darker colors) due to their distance from the viewer. The darker areas represents the non visible nodes.

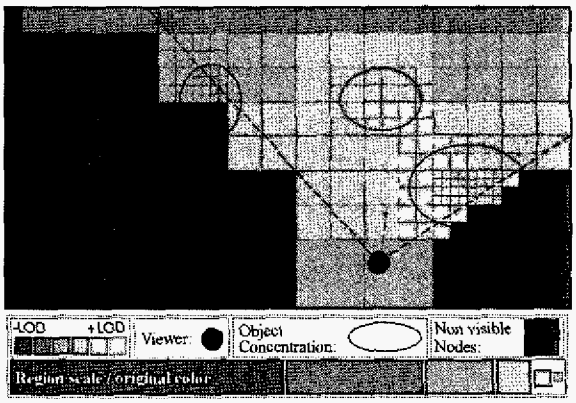

Figure 2: Force field quadtree with LOD. The box bellow shows the original gray scale of each square. If smaller volumes are represented with a darker gray scale, this means that one of its parents is being used to represent this node.

This approach of force interaction is very suitable for plants, that are fixed on the ground, which implies that the node of each object does not change. If the objects are allowed to move, the algorithm should need to compute which node that this object belongs to at each update of the force field.

Using the quadtree approach this search is very fast, and if we define a fixed and balanced quadtree the current LOD could be defined by checking the population of each visible region when the field is updated.

The Force field must be constantly updated because of changes of the user position and view direction. The technique is very adaptive, both interaction agent and object that receives the interaction can be easily adapted to it. In the next section, a wind model is adapted to the Force field as an interaction agent. Section 4 describes a plant model that receives the action of the Force field.

3.2 Fast wind simulation using Force field

This section presents a simple wind model, integrated to the Force field technique, that achieves fast animation of objects by the wind simulation. In this model each wind flow is represented by a single vector representing the velocity on each visible node of the Force field's quadtree.

The idea behind this technique is to rely on the Force field representation, since it already defines appropriate levels of detail according to their location relative to the viewer. The flow will be the set of vectors defined in the visible nodes.

Several flows can still act on a node, and the Force field is responsible for the computation of the resultant force applied on that node.

To calculate the action of the flow over some node, the flow value is approximated at the center of gravity of the node, considering that the flow do not change its direction. Figure 3 describes a simple scheme of this approach.

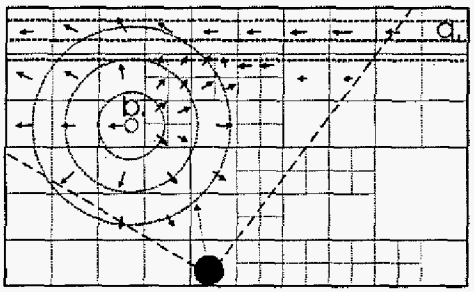

Figure 3: A wind flow simulation over Force field quadtree. Nodes with wind action have a vector describing the resultant force over it. (a) An uniform flow; (b) A repulsing flow.

The wind flows are generated by wind sources, that act like the model describes, applying force on the particles. The system allows the implementation of sources of any shape, and each source can also be under the action of other flows, moving under their action or not. This option allows the user to create complicated flows as the result of the combination of sources.

Primitives described in [11], like uniform, sink and source flow were implemented with realistic and fast results. Furthermore, it is possible to define different kinds of flows. Each flow can be activated/deactivated using an user interface, or can have a predefined duration.

4 Interactive Plants Simulation

This section describes the techniques that allow the generation of interactive simulation of plants, and discusses some of their advantages and drawbacks.

The same geometrical plant model, described on Section 4.1, is used for all distances to the viewer. However, these models can be represented using a smaller number (or even different) primitives. Transitions between level of details can be easily handled, avoiding abrupt shape changes.

Section 4.2 presents the animation algorithm, based on a Dynamic Constraints model to achieve realism and a pre-computated one integrated to the Force field for interaction and efficiency. Other features if this framework are shown in Section 4.3.

4.1 Plants Modeling

A vector based model was created to pre-computes the geometry of plants components, such as leaf, flower, stalk, etc. This model defines a skeleton, composed by a set of vectors, for each component. A shape is created by changing the properties of these vectors such as scale and direction, and then generating triangles or lines with the points defined by the model. The basic shape of the skeleton varies for each plant or component, and each can be fully configured by the user, so that a large variety of plants can be created.

A physically based model defines the appearance of the models by bending their components, simulating the action of gravity over them. Some parameters, like stalk width and mass, are used to define the model deformation. The resultant models are then pre-computed in several different angles.

As the components of plants of the same species are very similar, a small changes in their skeletons can generate a great variety of components shapes for that plant. Pre-computing the skeletons considerably improves the system performance during animation. More details about this model can be found in our previous paper [6].

The LOD technique implemented in this framework allows the configuration of as many levels the user requires. Its basic idea is to select a subset of the pre-computed vectors for each level of detail created. This subset is used to create the shape of the final component, according to the distance from the viewer. Furthermore, it also allows the usage of triangles or lines to create the model at each level.

The animation technique only updates the vectors that belongs to the subset defined by the current level of detail used to render the plant.

Figure 4 shows some results for a rose model, three level of details are illustrated. The difference between shapes generated using the same vector model is very clear. The complexity of each model is described in Table 1.

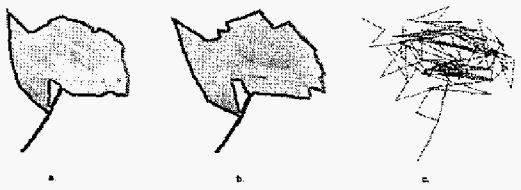

Figure 4: Three different representation of a rose model.

Model	*Primitive*	*Number of primitives*
a	triangle	177
b	triangle	105
c	lines	118

Table 1: Rose model complexity table for each level of detail.

4.2 Plants Animation

Using the plant component pre-computed models, the animation is performed very fast, since we only need to choose the model, given the interaction received from the Force field. The changing of models automatically defines the animation.

To generate a smoother animation with more realistic movements we use the technique of Dynamic Constraints. Constraint forces can animate models by moving their components to a desired configuration. The movement has different accelerations and velocities which warrants realism of the animations.

All the components are defined with an initial position. An elastic force ($\vec{F}_{el}$) opposes to the changing of this position. This force

only exists when some external force displace the component from its initial position, and it becomes stronger as the displacement increases. Figure 5 illustrates the behaviour of $\vec{F}_{el}$ for a component represented by its main axis.

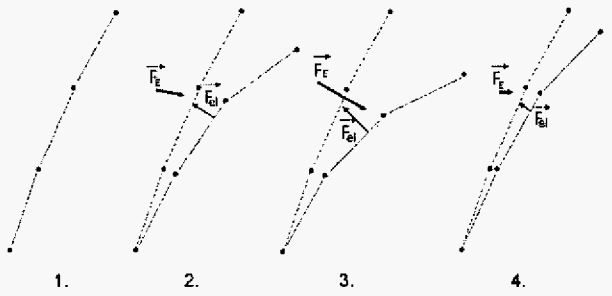

Figure 5: (1) The component in its initial position. (2) The component receives the action of a external force $\vec{F}_E$. $\vec{F}_{el}$ tries to restore the component's initial position. (3) $\vec{F}_E$ increases and the component bends more until the equilibrium between $\vec{F}_E$ and $\vec{F}_{el}$ is established. (4) $\vec{F}_E$ decreases, thus $\vec{F}_{el}$ take the component closer to its original position.

Subsection 4.2.1 explains the simplifications assumed to create a model based on Dynamic Constraints technique that can be used in interactive applications. The animation algorithm is described in Subsection 4.2.2, it uses both modeling pre-computations and Force field to generate faster and more realistic results.

4.2.1 A Dynamic Constraint Model for Plants

This section introduces a variation of the Dynamic Constraints model [2] to animate plants, discussing some of the possible simplifications required for an interactive execution. Equation 1 defines a simple constraint force calculation.

$$\sum_{constraints\ j} (\sum_{bodies\ i} (\Gamma^i G^i_j + \Lambda^i H^i_j)\vec{F}_{e_j}) +$$

$$\sum_{bodies\ i} (\Gamma^i \vec{F}^i_E + \Lambda^i \vec{T}^i_E) + \vec{\beta} + \frac{2}{\tau}\vec{D}^{(1)} + \frac{1}{\tau^2}\vec{D} = 0 \qquad (1)$$

This is a very complex equation to be solved for each plant component in an interactive system. In the plant model we assume that the torque does not exist, which implies that T, H and Λ, all related to the torque of the body, are null. τ is a time constant that is used to control the time spent for the constraint to be satisfied.

We define the action of the forces over a component at it center of mass, similar to the Point-to-Nail constraint described in [2]. This example illustrates a body under effect of gravity and without rotational terms.

$\vec{D}$ is the distance of the component center of mass from the current position to its initial position. $\vec{D}^{(1)}$ is defined as the rate of change of $\vec{D}$, the velocity of the movement ($\vec{D}^{(1)} = \vec{v}$). $\vec{D}^{(2)}$ is the acceleration of the movement and is defined by equation 2.

$$\vec{D}^{(2)} = \sum_{bodies\ i} (\Gamma^i(y)\vec{F}^i + \Lambda^i(y)\vec{T}^i) + \vec{\beta}(y) = \frac{\vec{F}}{m} \qquad (2)$$

Let $\vec{X}$ be the position of the center of mass when the constraint is satisfied and $\vec{X}_0$ the current position (constraint point), the deviation $\vec{D}$ is given by: $\vec{D} = \vec{X} - \vec{X}_0$

As only one constraint force, $\vec{F}_c$, was defined and one external force, $\vec{F}_E$, is returned by the force field as the resultant force that acts over the component, equation 1 can be rewrited with the assumptions and notations above.

$$\Gamma\vec{F}_c + \Gamma\vec{F}_E + \vec{\beta} + \frac{2}{\tau}\vec{v} + \frac{1}{\tau^2}(\vec{X} - \vec{X}_0) = 0$$

Furthermore, from 2 we have $\beta = 0$ and $\Gamma = 1/m$. The constraint force needed to take the body from $\vec{X}$ to $\vec{X}_0$ is show bellow:

$$\vec{F}_c = -\vec{F}_E - \frac{2}{\tau}m\vec{v} - \frac{1}{\tau^2}m(\vec{X} - \vec{X}_0) \qquad (3)$$

The Dynamic Constraint model calculates the constraint force needed to take the body to the constraint point defined. Observe that the constraint force may be decomposed into three components: one opposing to the external force, one opposing to the body's velocity and another that pulls the body towards the direction of the constraint point.

The framework uses the same model but instead of calculating the constraint force, it defines a constraint force to get the object position when the forces reach the state of equilibrium. This constraint force is defined as an elastic force given by equation 4, where PC is a component property defined by the user.

$$\vec{F}_c = \mathrm{PC}(\vec{X} - \vec{X}_0) \qquad (4)$$

The problem is then simplified to find out $\vec{X}$, given the component center of mass at its initial position ($\vec{X}_0$), the external force ($\vec{F}_E$), the constraint force ($\vec{F}_c$), the previous component velocity ($\vec{v}$) and the time interval passed since last position update (τ).

From 3 and 4 we have:

$$\mathrm{PC}(\vec{X} - \vec{X}_0) = -\vec{F}_E - \frac{2}{\tau}m\vec{v} - \frac{1}{\tau^2}m(\vec{X} - \vec{X}_0)$$

$$(\mathrm{PC} + \frac{1}{\tau^2}m)(\vec{X} - \vec{X}_0) = -\vec{F}_E - \frac{2}{\tau}m\vec{v}$$

Let $\mu = \mathrm{PC} + \frac{1}{\tau^2}m$, then 5 describes the deviation of the component center of mass as a function of the time elapsed, external force and current velocity. Calculations of constants related to the component can be performed only once to optimize animation time.

$$\vec{X} - \vec{X}_0 = -\frac{1}{\mu}\vec{F}_E - \frac{2m}{\tau\mu}\vec{v} \qquad (5)$$

The meaning of equation 5 may be illustrated by the following sentence: *"The deviation of center of mass from current to initial position is due to the action of some external forces and/or as result of a previous movement"*. Therefore, if some component is not on its initial position, some external force or previous movement caused the deviation.

We have calculated the effects of the constraint force for subintervals of τ, and the position the leaf will be in each time interval. This creates a more realistic animation since the leaf will move under different velocities and accelerations.

4.2.2 Fast Animation Algorithm

Equation 5 is very simple compared to 1, however, if it is applied to many plant components the performance of the system may degenerate. This section shows how to animate plants using the technique described in the previous section.

At each time interval τ, μ and $\frac{2m}{\tau\mu}$ are calculated once for each component, because both depend only on τ and some properties of the components. This cache of values avoid many redundant calculations. The deviation described in equation 5 can be calculated by

the product of a "constant" ($-\frac{1}{\mu}$) and the external force ($\vec{F}_E$) added to the product of another "constant" ($\frac{2m}{\tau\mu}$) and the previous move velocity ($\vec{v}$).

The animation is performed by two algorithms, one that updates the plant components according to the Force field, and other that is responsible for rendering the appropriate model.

The first algorithm obtains the external force from the Force field for each plant visible (on a visible node of the Force field). The deviation D is defined by the equation 5, given the external force action; current velocity, assuming it is in the direction of axis P; and the cached values of $-\frac{1}{\mu}$ and $\frac{2m}{\tau\mu}$.

The component's displacement angle is calculated using the deviation D. The current velocity must be estimated (defined by D/τ) for the next iteration.

If the angle of displacement was changed, the second algorithm is called. It defines the appropriate model to render the component, according to its distance, and get the vector model (from precomputed set of vectors), according to the new displacement angle.

The animation results are illustrated in Figure 6

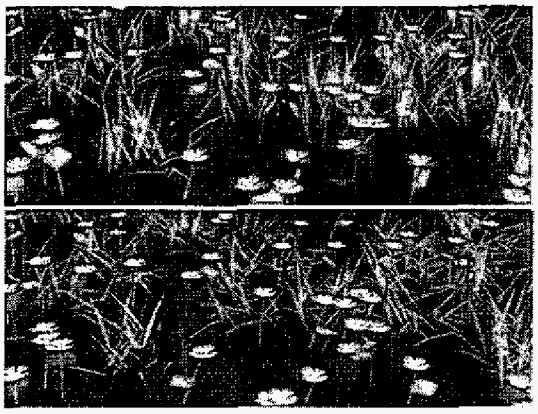

Figure 6: Animation of flowers and grass.

4.3 Other framework's features

The framework also have a simple terrain mesh generator and a plant distribution algorithm that considers some properties of the terrain, like irrigation and nutrients and their influence over each plant specie.

The algorithm creates plants concurrently for different plant distributions. One plant can occupy the area that other one could be occupying, removing the nutrient or water needed for the development of other distribution. The algorithm verifies if the terrain has the proper conditions of development for each plant distribution configured.

5 The Framework

5.1 Implementation Issues

JEcoSys was developed using Java[1] because of its portability and scalability. It can be easily adapted to Internet applications (Applets[2] and Java Web Start Technology[3]) and has several libraries that can be used to aid some system's functionality. Java Threads are used in other to process the Force field, simulation and animation of plants concurrently.

The system's architecture is based on object-oriented concepts, which makes it very reusable and adaptable to other applications or algorithms. The system is also very extensible, since it defines several Java interfaces and abstract classes. New techniques can be added or replace the ones defined in order to solve another problem or just for comparison of techniques.

Java $3D^{TM}$[4] [1] API (Application Programming Interface) was used for the system's implementation. The system's parameter input is done by using a XML (Extensible Mark-up Language)[5] based language.

5.2 Experimental Results

This section presents some test results that show the efficiency of the algorithms presented in this paper. The system prototype was tested on a Pentium III 900 MHz with 512 Mb and a 64Mb graphics video card. The results are presented in tables that allows comparison, analysis and discussion of the performance of the framework's components.

Three different kind of plant distributions were created for testing. Table 2 describes the geometrical complexity of each plant. The three level of details were used are described on Table 3.

Plant	*Type*	*LOD1*	*LOD2*	*LOD3*
A	grass tuft	250	150	175
B	yellow flower	72	52	37
C	purple flower	78	58	39

Table 2: Geometrical complexity of plants used for the system's analysis. B and C have similar complexity and represent only one flower. The grass tuft (A) is composed of 25 leaves.

Level of Detail	*Distance from viewer*	*Primitive*
LOD1	0 a 10 meters	triangle
LOD2	10 a 15 meters	triangle
LOD3	more than 15 meters	line

Table 3: Level of Detail's description

Three different scenes were created using these plant models. The scenes properties are described in Table 4 and shown in Figure 7. The following subsections use these test scenes to evaluate the performance of the system.

Scene	*# of A*	*# of B*	*# of C*
C1	50	15	15
C2	75	0	0
C3	0	100	100

Table 4: Test scenes description.

5.2.1 Performance x Geometrical complexity

Table 5 shows the system's performance analysis given the geometrical complexity of scenes. The number of primitives used is an approximation since it changes when the user moves along the scene, or the view direction changes. The measure is given in frames per second (fps) and considering that both Force field and animation algorithms are activated.

[1] http://java.sun.com

[2] http://java.sun.com/applets/

[3] http://java.sun.com/products/javawebstart/

[4] http://java.sun.com/products/java-media/3D/

[5] http://www.w3.org/XML/

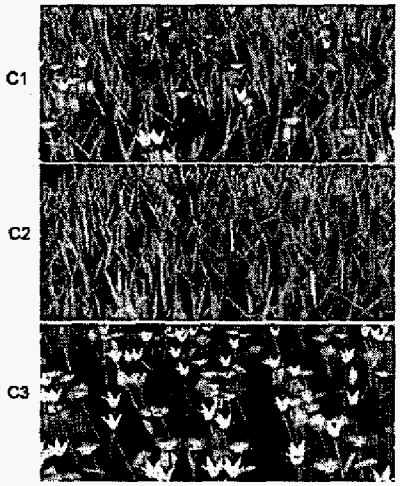

Figure 7: Scenes considered for testing.

Scene	*~ # of triangles*	*~ # of lines*	*Frame rate*
C1	7300	4000	29 ±11 fps
C2	10000	4400	26 ±09 fps
C3	8000	3750	33 ±06 fps

Table 5: Performance x scene's geometrical complexity.

The results show that the system's performance depends on the number of primitives and plant components being animated (just the movement of the primitives have a significant cost as shown in Table 7). For example, scene C3 has a better performance than C2, despite flowers have a more complex geometry, because A is composed by 25 components (leaves) and B or C only by 7 components. The results have shown that the system can animate around 4000 leaves of grass or 400 flowers in real-time in a typical personal computer.

5.2.2 Force field analysis

The Force field must handle efficientilly interaction, its benefits should be greater than the time spent for it treatment. The system has been tested with a random based animation, because with the Force field disabled there would be no animation. Table 6 shows the results. Frame rate 1 stands for Force field active, while Frame rate 2 for Force field disabled.

Scene	*Update time*	*Frame rate 1*	*Frame rate 2*
C1	40 ±30 ms	44 ±10 fps	47 ±5 fps
C2	30 ±20 ms	42 ±7 fps	44 ±6 fps
C3	45 ±35 ms	36 ±15 fps	38 ±5 fps

Table 6: Force field performance.

During the tests, the viewer was moving along the scene and the plants by the action of the animation technique. The results have shown that the Force field technique is very efficient, therefore it can be used in other systems that require real-time generic interaction.

5.2.3 Performance x Animation algorithm

Table 7 describes the system's performance for three situations: S1 - static scenes (without animation, but user movement); S2 - random animation; and S3 - framework's animation technique. The Force field is activated on all situations.

Scene	*S1*	*S2*	*S3*
C1	50 ±10 fps	44 ±10 fps	29 ±11 fps
C2	52 ±08 fps	42 ±07 fps	26 ±09 fps
C3	49 ±11 fps	36 ±15 fps	33 ±06 fps

Table 7: Animation algorithm performance.

The results have shown that the physically based animation algorithm have a good performance. Even on scene C1 the number of plants created would not be much more greater than the obtained by this system, for real-time results. The results from scene C2 (that represents the time spent only for the plants movement) and C3 are very similar. The time variation is very small, considering that a realistic animation based on a physical model is used in scene C3.

6 Conclusion and Future Work

We have presented JEcoSys, an extensible, highly adaptable, and efficient framework for the simulation of plants. The framework allows real-time user interaction with the simulation, and was developed completely in Java.

JEcoSys uses a simple physical model of plants. Interactive performance is basically achieved from the pre-computation of several parameters and the use of Level of details to control the simulation.

The study of several techniques has allowed the selection of the most suitable ones for each specific problem. Some of these were successfully extended or adapted to satisfy more realistically and efficiently the proposed problem. One good results achieved was the development of independent techniques for interaction and animation that can be used to solve other similar problems. Furthermore, when they are used together, in JEcoSys, the performance has been enhanced.

The framework also includes a Force field based technique that may be used for fast interaction over large scenes, with objects spread all over it, by several different agents. It was used to model the interaction of plants with an agent that simulates the wind.

The animation technique, based on a dynamic constraint model, generates realistic results and is still efficient, since some simplifications and pre-computations are assumed. It also can be used by other articulated models for fast animation.

Experimental results have shown that JEcosys is able to simulate complex scenes containing hundreds of tufts of grass, flowers and simple bushes in real-time, using a typical personal computer. The quality of the simulation can be seen in Figure 8.

The framework details are available at my homepage: <http://www.ime.usp.br/~lye>. We hope this framework will help other researchers to prototype other highly efficient algorithms for plant simulation. For further information please check my master dissertation [5], unfortunatelly it is only available in portuguese.

Our main goal is to continue the development of JEcoSys to improve the efficiency and realism for simulating plant ecosystems. Some of these enhancements include the use of: real plants textures, collision detection treatment, realistic biological data; and an image based algorithm for modeling.

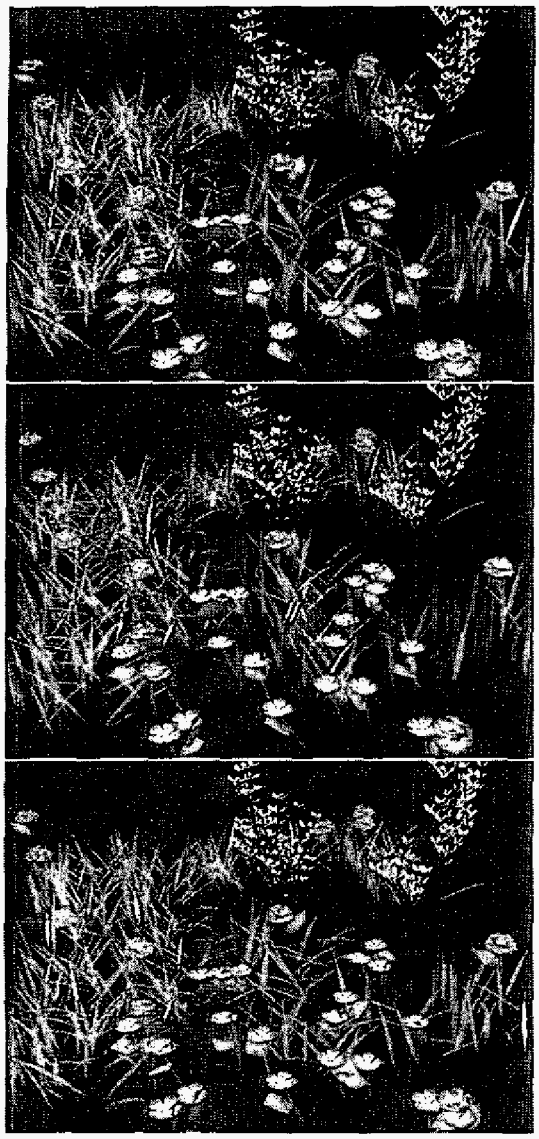

Figure 8: Plants animated in real-time.

References

[1] Sun Microsystems. Java 3DTM API Tutorial.

[2] R. Barzel and A. H. Barr. A Modeling System Based On Dynamic Constraints. *ACM SIGGRAPH*, 22(4):179–188, August 1988.

[3] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. *Proceedings of the IEEE Visualization Conference*, October 2002.

[4] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic Modeling and Rendering of Plant Ecosystems. *ACM SIGGRAPH*, 32:275–286, 1998.

[5] L. C. Y. Endo. Master dissertation: Simulaćão de Mini-Ecossistemas Vegetais em Tempo Real.

[6] L. C. Y. Endo, C. H. Morimoto, and A. E. Fabris. Real-time Animation of Underbrush. *WSCG*, 11:41–48, 2003.

[7] D. R. Fowler, P. Prusinkiewicz, and J. Battjes. A Collision-based model of spiral phyllotaxis. *ACM SIGGRAPH*, 26(2):361–368, 1992.

[8] A. Jakulin. Interactive Vegetation Rendering with Slicing and Blending. In *Eurographics*. Eurographics, August 2000.

[9] F. Perbet and M. P. Cani. Animating Prairies in Real-Time. In *ACM Interactive 3D Graphics*, March 2001.

[10] W. T. Reeves and R. Blan. Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems. *ACM SIGGRAPH*, 19(3):313–322, 1985.

[11] J. Wejchert and D. Haumann. Animation Aerodynamics. *ACM SIGGRAPH*, pages 19–22, July 1991.

[12] A. Witkin and D. Baraff. Physically Based Modeling: Principles and Practice (SIGGRAPH'97 Course Notes).

Color Plates