Virtual Climbing Plants Competing for Space

Bedřich Beneš* Erik Uriel Millán

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Ciudad de México
Calle del Puente 222
Tlalpan 14380 Mexico City

Abstract

An old algorithm for visual simulation of climbing plants is extended here. Plants are modeled as systems of oriented particles that are able to sense their environment. Particles move to the best locations using directed random walk. We use the phenomenon of traumatic reiteration for critical cases. If there is no location for further growth possible the particle dies, but before sends a signal that is propagated down in the plant structure. This signal activates the closest possible sleeping particle that takes its job. We use an associated voxel space for collisions and space occupancy detection as well as for evaluating the illumination of the plant organs. The algorithm is fast, easy to implement, and runs interactively even for quite large scenes on a medium-class computer. We believe that this approach can be used as an interactive technique in architecture, computer games, computer animation, etc.

Keywords: virtual climbing plant, visual simulation, artificial life, collision detection, voxel space, particle system, oriented particles

1 Introduction and Previous Work

There are no doubts that the most realistic way of modeling real things is their simulation. To give a specific example, terrain modeling and morphology is the best achieved by erosion simulation. Simulations are context sensitive, respect underlying structures, and give realistic results reasonable fast. On the other hand, *ad hoc* interactive techniques usually provide very high level of control although they are not based on simulation and therefore have certain limits in words of visual plausibility.

Visual simulation of plant development has been found as the best way of plant modeling many years ago. Before, the fractal structure of plant topology has been an inspiration for visual models of plants. Probably one of the first approaches, but certainly a very good example, was introduced by Bloomenthal [4]. He uses fractal-based techniques to generate plant topology, generalized cylinders for plant geometry, and photographs for textures of bark and leaves. Many fractal-based approaches and interactive techniques were introduced later but they are out of the scope of this paper. Their comprehensive review can be found for example in [11].

The principal disadvantage of fractal-based techniques was mentioned above. They do not respect the underlying topology; in other words they are not adaptive. Practically it means that the plant has the same (or better said self-similar) appearance regardless to the place it is located. Certainly a maple grown in a corner of a yard would have different shape than the same plant grown in an open landscape. There are some approaches aiming to eliminate this drawback (for example [16]). The most reasonable way to model realistic plants is based on simulation of the plant development in their real conditions.

There are two approaches typically used in computer graphics: L-systems and particle systems.

1.1 L-systems

L-systems (Lindenmayer's systems) were introduced by Lindenmayer in 1968 [13] and later extended by Prusinkiewicz and his collaboratives (see [11, 17] for review and [16, 18] for the latest progress).

L-system is a parallel string rewriting system. It rewrites a starting symbol (so-called axiom) to a sequence of modules (L-string) that are interpreted by a geometric turtle. The turtle interprets every module as a command and reacts in a predefined way. The path that the turtle travels in the space and the actions it performs correspond to the plant geometry. The rewriting process is controlled by a sequence of production rules, in the case of context sensitive

*beda@campus.ccm.itesm.mx

L-systems by a context of a module, or by conditions, by parameters, etc.

An important step forward to realistic simulations was introduced in [15]. So called Open L-systems control the rewriting process also by external conditions, for example by light or proximity of an obstacle. This communication flow is mutual; plants are influenced by the surrounding environment and vice versa.

The latest progress in visual simulation of plant development using L-systems is devoted to combining interactive techniques and L-system. In the work [16], the plant is endowed with certain physical quantities (stiffness, torsion, etc.) that are used in the inverse kinematics optimization. The geometry of the plant is changed interactively whereas the underlying topological structure remains unchanged.

The last work of Prusinkiewicz *et. al* [18] enhances visual plausibility of plants integrating three important elements: posture, gradual variation of features, and the progression of drawing process from silouhette to local details.

L-systems are extremely powerful tools because they are closest to the biological simulations and provide visually extremely plausible results. On the other hand there are some objections and drawbacks. One of them is that it is not easy to predict the final shape of the plant and there is a very low control over the rewriting process. Second and more important objection is that L-systems are complex. A simulation of a simple plant requires a number of production rules that influence each other. A simple error can destroy a lot of work and is hard to detect. Even more, construction of the production rules requires a lot of experiences. L-systems seem to be a kind of programming language but without data structures and with very low control over the production process.

1.2 Particle Systems

The second group of algorithms used for virtual plants modeling is based on particle systems. Particle systems were used for the first time for plant modeling by Reeves and Blau [19]. As in the case of L-systems, the particle systems were closed allowing no interaction between particles and an environment.

Arvo and Kirk [1] and later Greene [10] use similar approach to simulate climbing plants. Particles are able to sense the surrounding environment and perform certain actions. The proximity of a supporting tool is detected using ray-casting and the plant tries to keep close. At the same time the amount of incoming light is evaluated and the plant tends to leave shadowed parts of the scene. The greatest difference between the work of Greene [10] and Arvo [1] is that Greene uses voxel space to enhance collision detection and proximity of objects whereas the later use ray-casting to achieve it. These papers are extended by our work here.

AMAP, another particle system oriented approach, was described in details for the first time in [8] and later extended for example [2, 12]. AMAP is a biologically based system using "intelligent" particles. Every module of a simulated plant has a certain behavior assigned that depends on its internal state and external conditions. For example a bud contains apical meristem and can grow only if it is sufficiently fed by soils from the root and by CO_2 from the adjacent leaves. Leaves produce this material only if they are exposed to light; root can pump up water with soils only if there are resources available, etc. Originally, AMAP was the name for a computer graphics model, in these days it is a commercially available product either as a standalone program or as a plug-in for MAYA or Softimage.

The work of Chiba *et. al* [5, 6, 7] uses particle system based plant modeling as well. The main focus is an interaction of the plant with light and tree pruning. The approach is similar to AMAP, although the rules used for plant development simulation are simpler.

This paper continues with the plant model and plant development description. Section 3 describes in depth the algorithm and the data structures. Section 4 deals with the results of our work and the last section concludes the paper and discusses the future work.

2 Plant Model

2.1 Modular Structure

Plant can be modeled at different levels of details. The original work of Lindenmayer [13] focuses the cellular structure and development, whereas some applications, for example flight or military simulators, require models of entire plants [20]. There are also approaches focusing plant ecosystems [9].

The most common approach is taken from biology [8] and is used in computer graphics to model the plant on level of so called *modules*. Figure 1 gives a description of the plant modules.

A module is a basic unit that usually corresponds to certain plant organ or a group of organs that have some important properties from the viewpoint of functionality of the plant. Modules can be distinguished easily on the plant and also serve for the plant identification.

The most important plant module is a *bud*, which can assume one of two forms: an *apical bud* is always located at the extremity of the main trunk or lateral branch, whereas a *lateral bud* is situated at the leaf's axil (it is also called axilar bud). A *leaf* is always adjacent to a lateral bud. A *node* consists of one or more lateral buds and an identical number of leaves. The lateral leaves feed the bud. An *internode* is a piece of stem located between two successive nodes. The node is either situated between two internodes, or at the tip

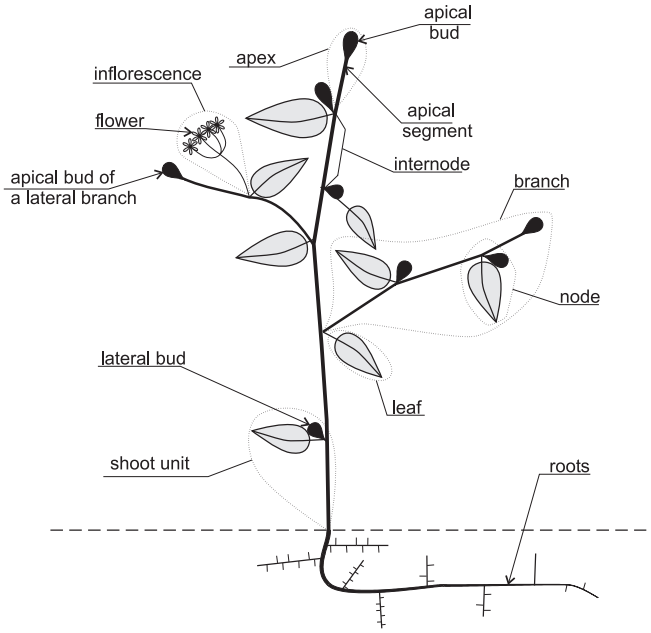

Figure 1. Modular structure of a plant

of the branch. An *apex* is an apical segment with its apical bud.

The attention paid to the geometrical representation of the plant modules is reflected in visual plausibility of the resulting three-dimensional model. The common approach is to represent modules roughly, internodes as cylinders, leaves as sets of triangles, buds as spheres, etc. These simplifications cannot be easily distinguished in large distance views because plants are quite complex, but disturb in the close-ups. An interesting approach to a semi-interactive geometrical representation is described in [14].

2.2 Plant Development

Buds are the basic units causing a plant to grow (see in Figure 2). Every bud has a special cellular tissue called *apical meristem* that extensively splits and replicates. This activity depends on the level of chemical substances and on its DNA. A bud can perform different actions depending on both environmental conditions of the plant, light, water, nutrients, etc., and internal ones, age, amount and viability of the meristem, etc.

A bud can either die, or bloom and die, or become dormant, or become an internode. The last mentioned option, the process of becoming an internode, is the most important one because it causes growing and branching. This process consists of three steps (see Figure 2). At first, one or more lateral leaves appear beside the bud and the same number of new buds appears at their axils. Then the apical bud produces a piece of stem - the new internode. Every branch is the result of the activity of its apical bud. Apparently, no bud can grow to infinity and this ability of the apical meristem differs with the bud age. The ability to grow can be the best characterized by a bell-shaped function of time.

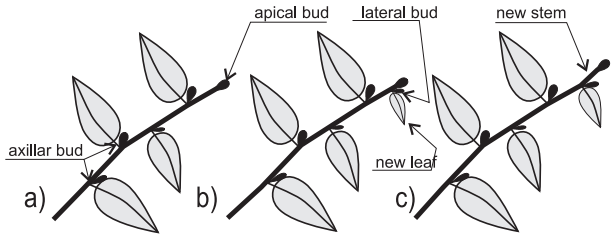

Figure 2. Apical bud produces new internode and a lateral bud (from [2])

A bud seeks for light (a phenomenon called *phototropism*) and its abundance causes the bud to produce more lateral buds. If there is a lack of the light buds search for the light intensively. Less light causes buds to become dormant and a long-term insufficiency of light causes them to die. The sensitivity of plants to the light is well described in computer graphics literature [2, 7, 15] and we will not deal with this phenomenon in more details here.

2.3 Traumatic Reiteration

The focus of this paper is the usage of the phenomenon of the *traumatic reiteration*. Traumatic reiteration was mentioned in computer graphics for the first time in [5] and later in [15].

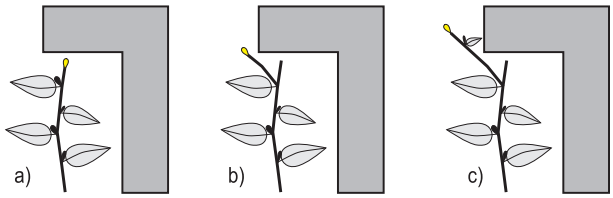

Figure 3. Traumatic reiteration a) the apical bud has no way to grow and dies b) this causes cease of the production of the inhibitor so the closest possible bud starts growing c) the new apex produces new lateral bud and a leaf

Leading apex of a plant produces new internodes and lateral buds and at the same time produces certain chemical substance that inhibits the other buds so they cannot grow.

This propagation has a certain limit so the not affected buds produce lateral branches. Suppose (see in Figure 3) the apical bud has been cut or cannot grow and therefore dies. The inhibitor is not produced anymore because the leading apex is missing so the lack of the inhibitor is propagated down in the plant. The first bud in the way down that is not inhibited anymore starts to grow. The new bud takes the role of the main apex that is why this effect is also called *the change in a leadership*. This growth causes that the same chemical substance inhibiting the other buds is produced and is propagated down in the plant. This important phenomenon guarantees that the substantial changes in the plant structure will not affect its viability.

2.4 Climbing Plants

We are primarily interested in climbing plants here. Some voluble species like (by the way toxic) English ivy (*hedera helix*) (see Figure 4), grapevine (*vitis vinifera*), or night-blooming cestrum (*cestrum nocturnum*) produce two lateral buds. One of them produces new lateral branch in the way described in the Section 2.2, whereas the other develops some kind of a claw-like organ. This can surround some nearby objects but can even enter some shallow plaster or bark of another plant. This keeps the bud, and therefore the entire branch, close to the supporting object.

Figure 4. Hedera helix

The voluble plants are sensitive to two phenomena. Some of them prefer to grow in areas that have high level of calcium, i.e., plaster of a house, some of them like to be exposed to intensive light, whereas the other prefer to stay in a shadow; for example tombs are frequently covered by the English ivy.

3 Algorithm and Data Structures

We use traumatic reiteration and a collision detection simulation to assure homogeneous covering of the areas where the climbing plants grow.

Climbing plants grow just by elongating the apical areas. They do not elongate already grown areas (stem internodes, see in Figure 1) because they are fixed in the supporting objects. These parts just increase their diameter.

We simulate the plant as a system of oriented particles. Every particle is represented by a sphere of certain radius and has associated a local coordinate system. The vector x of the coordinate system corresponds to the direction of the growth. Particles travel in the three-dimensional space forming a path that corresponds to a branch. Branches, i.e., successive particle positions, are represented as generalized cylinders. We represent leaves as a set of Bézier bicubic patches and the supporting tools consist of triangles, spheres, cylinders, and line segments.

A plant tends to grow into the areas with the best conditions. To simulate this we should evaluate a *fitness function* (see Section 3.2) for each point of the three-dimensional space depending on the scene. This function informs us how convenient the new location is. This is certainly time consuming. Instead, we perform a stochastic sampling of the fitness function and evaluate the best case just from the performed samples. Apparently the higher the number of samples the better the function is approximated. To achieve this sampling efficiently we apply modification of a random walk, so called *directed random walk* [3] that is explained in the Section 3.1.

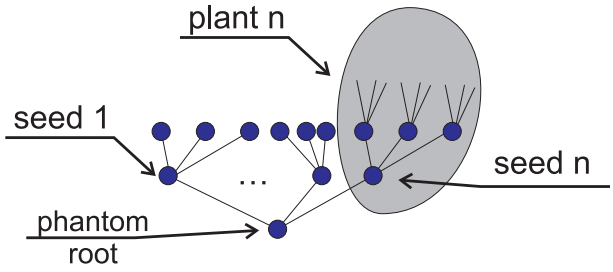

Figure 5. The data structure representing the virtual plants

If there is no growth possible, for example the branch is in the corner having no space to grow as in the Figure 3, or there is not enough light, the bud dies and we perform traumatic reiteration task to activate the closest bud.

The algorithm uses a data structure called the *list of active buds* that corresponds to the buds that are actually growing and are not dormant. The entire plant is represented as a mathematical n-ary tree. It is important to notice that the

active buds are not always located in the leaves of this mathematical tree, so the list of the active buds helps to maintain the algorithm fast.

At the beginning of the simulation we define the scene and put some seeds there. Every seed corresponds to one plant. To simplify the data structures we simulate the entire scene just as a one plant that has just one root (Figure 5 illustrates this). This "phantom root" corresponds in fact to the root of the mathematical tree. The first level of this tree keeps the seeds. The entire algorithm is described as follows:

1. Put all seeds into the list of active buds
2. While the list is not empty:
3. Perform the following actions in parallel for every bud from the list of active buds:
   (a) Generate n sample positions
   (b) Evaluate the fitness function for every sample
   (c) Pick the best sample
   (d) Is the best position viable?
   YES$\rightarrow$ continue growing
      i. Grow there
      ii. Sometimes generate lateral buds and do not put them into the list of active buds
      iii. Put every m-th lateral bud into the list of active buds to achieve branching
   (e) NO $\rightarrow$ perform traumatic reiteration
      i. Remove the bud from the list of active buds
      ii. Find the closest bud down on the same branch
      iii. Put this bud into the list of active buds

3.1 Directed Random Walk

A bud has its growing direction represented as a vector given by its actual and the previous position. At the beginning we define the growth direction of the seeds from the normal vector to the ground.

To get a new location we randomly generate n new positions for the corresponding particle at a certain distance from the actual position inside an angular area given by an allowed distribution (see Figure 6). The statistical characteristic of the internode length of real plants obeys the Gaussian distribution that is why we use the Gaussian random number distribution to achieve higher visual plausibility of the resulting model. The alteration of the angle assures the direction of growth is kept and modified just slightly. Since the particles are oriented, we can generate new positions efficiently just by perturbing the direction of their motion.

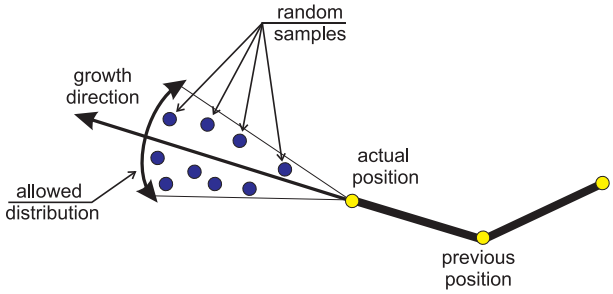

Figure 6. Directed random walk. New random positions are generated only within a predefined area. The direction is given by the actual and the previous position

3.2 Fitness Function

We generate n random positions using the directed random walk and then select the best one. The criterion for this choice depends on the value of so called fitness function f. The fitness function

$$f : Real^3 \rightarrow 0 \leq f \leq 1,$$

can be defined arbitrarily. The proximity of the object and the amount of incoming light (see an example in the Figure 9) is used as a criterion here.

The value $f = 1$ corresponds to the best choice, i.e., the new position is exactly on the surface of a supporting object and is illuminated by all lights in the scene. The new growth position is then taken as the maximum from the given samples i.e.,

$$max\{f_i, \forall i, i = 1, \ldots, n\}$$

where n is the number of samples. It is important to note that the maximum can also be zero in the case if there is no way of growth possible. It corresponds to the step 3e) from the above outlined algorithm.

The function f_i is the fitness function for the particular sample and consists of two components. The first is denoted by f_i^d and reflects the proximity of the objects whereas the second, denoted by f_i^l, corresponds to the amount of incoming light. So the

$$f_i = \frac{1}{2}(f_i^d + f_i^l), i = 1, \ldots, n;\ 0 \leq f_i^d, f_i^l \leq 1 \qquad (1)$$

The sum is divided by two in order to keep the result normalized.

The distance component is computed as

$$f_i^d = 1 - \frac{d_j}{max}, j = 1, \ldots, m \qquad (2)$$

where d_j is the actual distance from the $j-$th object, m is the number of the objects in the scene, and max is the maximum distance in the scene. If the object is exactly on a surface the distance $d_j = 0$ so the function returns one. If there are no objects in the scene the distance function should be modified to return max and not infinity.

The light component f_i^l of the i-th sample from the equation (1) is evaluated as

$$f_i^l = \frac{1}{p} \sum_{k=1}^{p} \eta(i,k), \tag{3}$$

where p is the number of the light sources and $\eta(i,k)$ is an illumination function determining if the k-th light source illuminates the bud in the i-th sample position.

$$\eta(i,k) = \begin{cases} 1 & \text{the } i\text{-th sample is illuminated by the } k\text{-th light} \\ 0 & \text{otherwise} \end{cases}$$

Better evaluation of this term would involve real values from the continuous interval from zero to one. The binary case we use just corresponds to two cases - the light shines on the bud or not and does not calculate with partial occlusions. The algorithms [2, 5] or [15] could be used.

There are special "penalty cases" of the fitness function. The function $f_i = 0$ if the sample is on the other side of a surface, because plants do not grow through the objects. Another "penalty case" occurs when the sample is in collision with some other object. This is explained in-depth in the Section 3.4.

3.3 Distance Evaluation

To evaluate the best position of every sample we have to evaluate the distance of the active bud from the objects in the scene. Some kind of a spatial subdivision could be used. Many techniques have been described in the ray-tracing literature and many techniques for evaluating point-to-object distance (sphere-to-object distance can be reformulated to this problem) can be found in books of computational geometry. We do not evaluate the distance to the plant itself (they do not grow on themselves), but just to the others objects in the scene. The number of tests performed in every step is $O(b * m)$ where b is the number of active buds and m is the number of the objects in the scene (see equation (2)).

3.4 Collision Detection and the Failure Test

We have to evaluate collision of every bud from the active bud list potentially with every object in the scene including the plant itself. Algorithms for collision detection have usually quite high overhead for the scene precomputation and data structure management. We deal with the dynamic scene so we have decided to use static voxelization of the three-dimensional space similar to the work [10]. The difference between this work and ours is that we do not precompute distance from the objects, but we use this additional data structure just for the collision detection.

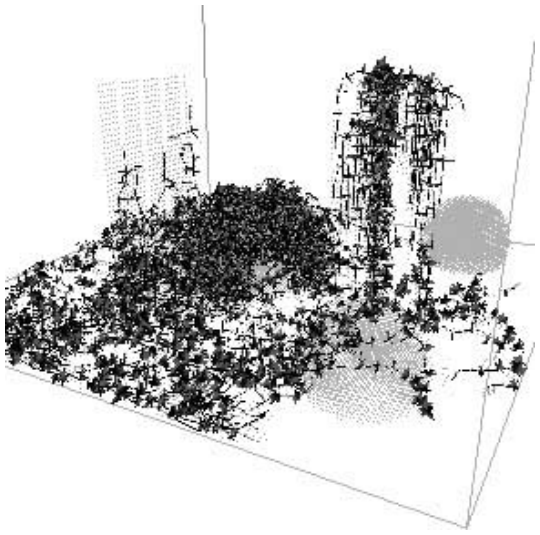

Figure 7. OpenGL preview of the voxelized scene where the used voxels and the plant are displayed

When the scene is loaded the voxels that are allocated by the objects are marked as occupied (see in Figure 7). This is done just once for the supporting tools. This process is somewhat complicated in the case of the plant. When something new appears in the scene, we must allocate the corresponding voxels and mark them as occupied. This is enhanced in the following way. For every object in basic position we precompute the occupied voxels. When the object appears in the scene, we just rotate and scale the pattern and map it into the voxel space. This is more complex in the case of Bézier surfaces used for leaves. We perform the de Casteljau adaptive subdivision that stops at the size of the voxel. Figure 7 shows an OpenGL preview of the scene where occupied voxels and the plant are displayed.

Since we use B-rep of the objects we must be careful not to let the particle pass through or inside the objects. Figure 8 shows the possible collision cases. The case a) is on the other side of an object and will be rejected by the 3D DDA algorithm, because the branch would grow through the occupied voxels. The case b) is located inside an occupied voxel and therefore failures as well.

This voxel space based collision detection solution is very fast and does not depend on the complexity of the scene since the number of the voxels is constant. The voxel allo-

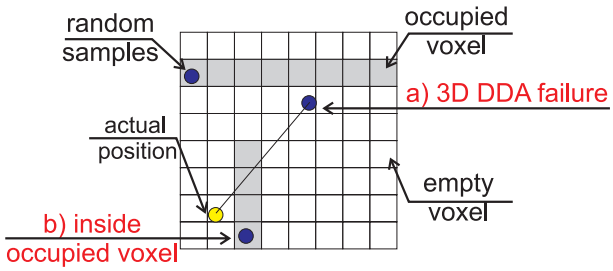

Figure 8. Failure of all samples

cation does depend on the number of the objects. It would be interesting to see how the size of the voxel influences the final distribution of the objects. We use 256x256x256 voxels for the scenes that are on the figures of this paper.

3.5 Light

The fitness function for the light component was described in the Section 3.2. We perform the 3D DDA ray casting to the point light sources to evaluate the $\eta(i,k)$ term from the equation (3). We evaluate the incoming light just for buds. Precisely done, this should be evaluated for the entire area of leaves. Leaves should produce chemical material that should be delivered to the adjacent buds. We have not found this simplification as the cause of a great error, although this statement should be quantified and we work on it.

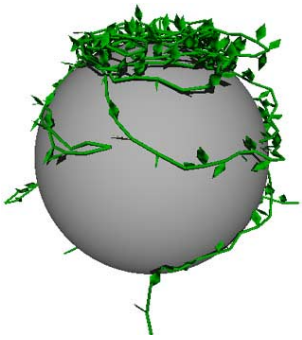

Figure 9. Sphere covered by a growing plant that tends to stay at the illuminated area. Light is arriving from the top

On the example in the Figure 9 the seed is located in the front of the sphere and the scene is illuminated by one parallel light source from the top. Once the plant reaches the sphere it tends to keep in the illuminated areas and it competes for the best position on the light. At the same time the traumatic reiteration together with the collision detection assures the top of the sphere is more-or-less uniformly occupied. As can be seen in the Figure 9, if there is no position on the sphere available the plant also tends to grow into the free space that corresponds to reality.

4 Results

We have implemented the algorithm in C using OpenGL and we run it on PC equipped with PentiumIII/500MHz. We use OpenGL for preview and Persistence of Vision raytracer (`www.povray.org`) for the photorealistic images.

Scenes consist of at most one hundreds thousands plant modules (leaves, internodes) and it runs in real time.

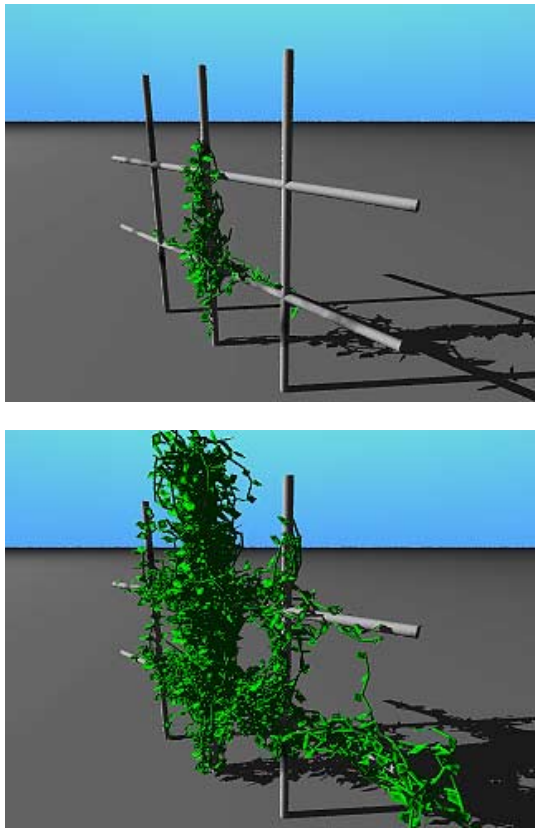

Figure 10. There is one plant seeded in the middle of the fence that grows and branches intensively

One plant is seeded in the middle part of the fence in the Figure 10. The plant tends to branch intensively and rapidly captures the entire middle cylinder. Due to the intensive branching and frequent reiterations the plant does not capture the rest of the supporting structures very fast. The an-

imation consists of 120 frames and the simulation runs for approximately 30 seconds.

Similar example on Figure 11 shows the same plant with less intensive branching. The plant easily spreads over the entire structure. One flaw can be noticed there. As we do not simulate the gravity at the moment the plant reaches the end of the supporting tool structure, it bends in a spiral closest to the occupied voxels and grows back. In reality the branches are falling down.

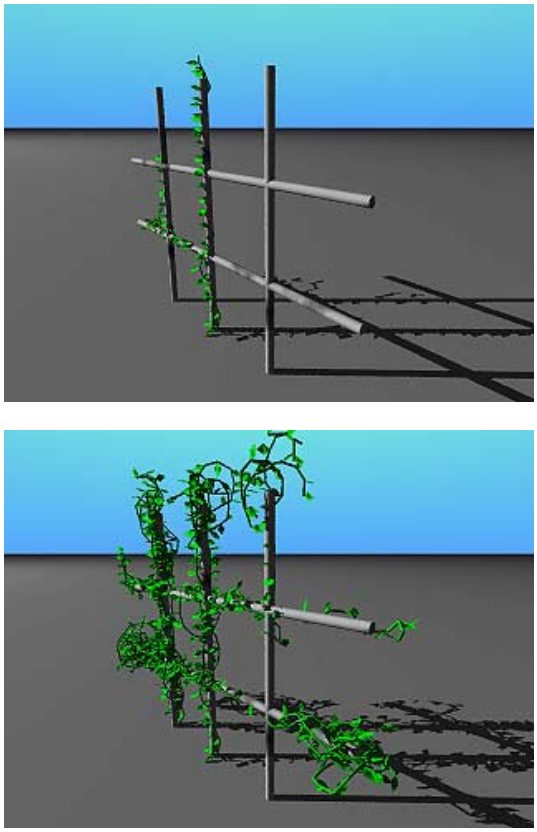

Figure 11. The same plant with less intensive branching easily covers the entire fence structure very fast

An example in Figure 12 displays fence covered by a climbing plant. The top image shows the complete plant whereas the other displays just the branches. The animation has 20 frames and it was computed in less than ten seconds.

The last example in Figure 13 shows a simulation of a house that is covered by ivy. There are twenty seeds around the house that reach it fast. Once the plants find the house they are competing for the space intensively performing the reiteration task. The first two images show the process of the growth whereas the others show the structure of the plants without the house. The entire animation consists of 500 frames and has run two minutes. Visit paginas.ccm.itesm.mx/beda/research to see the animations.

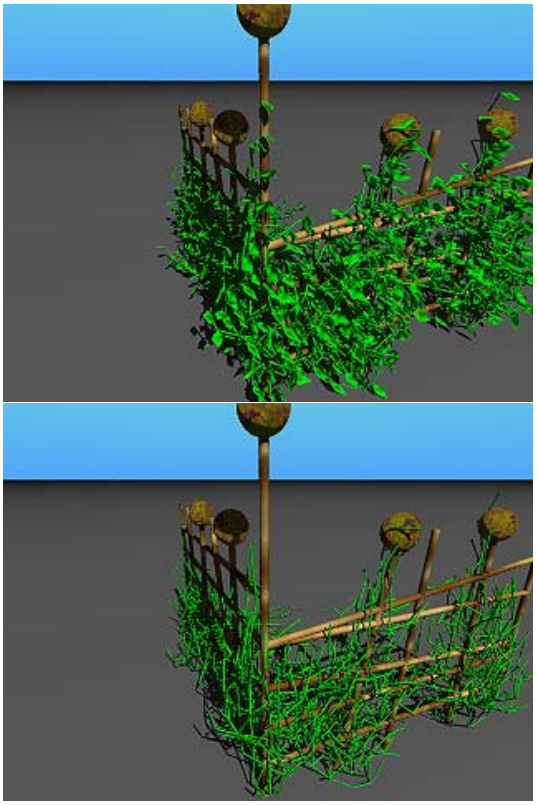

Figure 12. An example of a fence with a climbing plant. The top image shows the plant with leaves and the other shows just the branches

5 Conclusions

An old algorithm [1, 10] for climbing plants simulation was recalled and enhanced here. We simulate the plant as an ”intelligent” particle system, where every particle representing the basic growing module of a plant senses its environment. Using the mechanism of directed random walk we sample the space and the fitness function assigns a number to every sample. We select the best sample and let the plant grow there. If the best case is not viable we run the traumatic reiteration task that activates the closest possible bud on the same branch. The scene is complex and is changing fast so we use an associated voxel space to solve the collision detection task efficiently. Buds are illuminated by light that is also computed in the voxel space using 3D DDA sampling. Leaves are modeled as Bézier patches, branches as a generalized cylinders, and the supporting tools in the scene

consist of triangles, quadrilaterals, line segments, spheres, and cylinders.

Our implementation runs in real time on medium-class PC. We believe that this algorithm could be used as an interactive technique for plant modeling. The slowest part of the program is the distance evaluation so the future work includes some kind of spatial subdivision for efficient distance calculation, better algorithm for illumination evaluation, and more precise geometric representation.

This work is supported by the ITESM grant No.32391.

References

[1] J. Arvo and D. Kirk. Modeling Plants with Environment-Sensitive Automata. In *Proceedings of Ausgraph'88*, pages 27–33, 1988.

[2] B. Beneš. Visual Simulation of Plant Development with Respect to Influence of Light. In *Computer Animation and Simulation'97*, Springer Computer Science, pages 125–136. Springer–Verlag Wien New York, 1997.

[3] B. Beneš and E. Espinosa. Using Particles for 3D Texture Sculpting. *The Journal of Visualization and Computer Animation*, 12:191–201, 2001.

[4] J. Bloomenthal. Modeling the Mighty Maple. In *Proceedings of SIGGRAPH '85*, volume 19(3) of *Annual Conference Series 1985*, pages 305–311, 1985.

[5] N. Chiba, K. Ohshida, K. Muroaka, and S. Nobuji. A Growth Model Having the Abilities of Growth-Regulations for Simulating Visual Nature of Botanical Trees. *Computer & Graphics*, 18:469–479, 1994.

[6] N. Chiba, K. Ohshida, K. Muroaka, and S.. Nobuji. Visual Simulation of Leaf Arrangement and Autumn Colors. *The Journal of Visualization and Computer Animation*, 7:79–93, 1996.

[7] N. Chiba, S. Okawa, K. Muroaka, and M. Muira. Visual Simulation of Botanical Trees Based on Virtual Heliotropism and Dormancy Break. *The Journal of Visualization and Computer Animation*, 5:3–15, 1994.

[8] P. de Reffye, C. Edelin, J. Fraçon, M. Jaeger, and C. Puech. Plants Models Faithful to Botanical Structure and Development. In *Proceedings of SIGGRAPH '88*, volume 22(4) of *Annual Conference Series*, pages 151–158, 1988.

[9] O. Deussen, P. Hanrahan, B. Lintermann, R. Měch, M. Pharr, and P. Prusinkiewicz. Realistic Modeling and Rendering of Plant Ecosystems. In *Proceedings of SIGGRAPH'98*, *Annual Conference Series 1998*, pages 275–286, 1998.

[10] N. Greene. Voxel Space Automata: Modeling with Stochastic Growth Processes in Voxel Space. In *Proceedings of SIGGRAPH '89*, volume 23(4) of *Annual Conference Series*, pages 175–184, 1989.

[11] H. Jones. Modelling of Growing Natural Forms. In *Eurographics'00 Tutorials*. Springer–Verlag, 2000.

[12] R. Lecoustre, P. de Reffye, M. Jaeger, and P. Dinouard. Controlling the Architectural Geometry of Plant's Growth – Application to the Begonia Genus. In *Computer Animation'92*, pages 199–214, 1992.

[13] A. Lindenmayer. Mathematical Models for Cellular Interaction in Development. *Journal of Theoretical Biology*, Parts I and II(18):280–315, 1968.

[14] B. Lintermann and O. Deusen. Interactive Modelling and Animation of Branching Botanical Structures. In *Computer Animation and Simulation'96*, Springer Computer Science, pages 139–151. Springer–Verlag Wien New York, 1996.

[15] R. Měch and P. Prusinkiewicz. Visual Models of Plants Interacting With Their Environment. In *Proceedings of SIGGRAPH '96*, volume 30(4) of *Annual Conference Series 1996*, pages 397–410, 1996.

[16] J. L. Power, A. J. B. Brush, P. Prusinkiewicz, and D. Salesin. Interactive Arrangement of Botanical L-system Models. In *Proceedings of SIGGRAPH'99*, volume 22(4) of *Annual Conference Series*, pages 175–182, 1999.

[17] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer–Verlag, New York, 1990. With J.S.Hanan, F.D. Fracchia, D.R.Fowler, M.J.de Boer, and L.Mercer.

[18] P. Prusinkiewicz, L. Mundermann, and B. Lane. The Use of Positional Information in the Modeling of Plants. In *Proceedings of SIGGRAPH'01*, volume 22(4) of *Annual Conference Series*, pages 289–300, 2001.

[19] W. Reeves and R. Blau. Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems. In *Proceedings of SIGGRAPH '85*, volume 19(3) of *Annual Conference Series*, pages 313–322, 1985.

[20] J. Weber and J. Penn. Creation and Rendering of Realistic Trees. In *Proceedings of SIGGRAPH '95*, volume 22(4) of *Annual Conference Series*. pages 291-300, 1995.

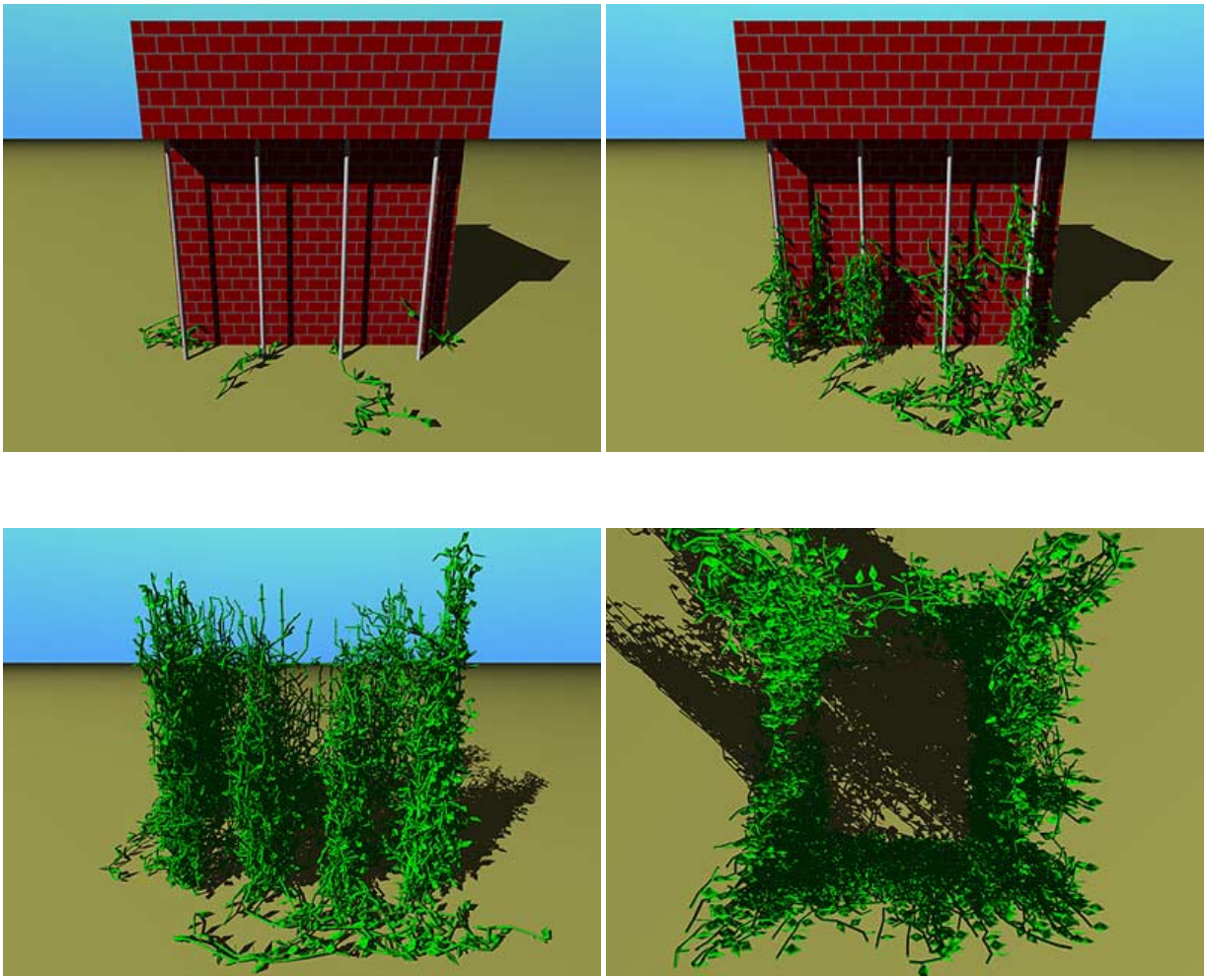

Figure 13. An example of a house covered by a plant. The two upper images show the plant development, the lower images show the plant without the house