User's Manual for Environmental programs

Radomír Měch
Przemyslaw Prusinkiewicz

May 7, 1998

Contents

1 Collisions

1.1 Environmental program arvo

This program is used for simulation of climbing around surfaces. For a given segment, it determines whether the segment collides with an object and if it does the program computes a new orientation of the segment so that the collision is avoided and the segment's tip keeps a given distance from the surface. The algorithm is based on the paper by Arvo and Kirk [1].

1.1.1 Execution

Name of the executable: arvo

Command line parameters:
arvo [-e environment_file] environment_argument_file

Commands in the environment argument file

`domain size:` $xrange\ yrange\ zrange$ specifies the range (in world coordinates) of a regular grid used to store objects for the intersection test. The numbers can be delimited also by ',', ';', or '×'.

`position:` $xpos\ ypos\ zpos$ specifies the position of the lower front left corner of the grid. The numbers can be delimited also by ',', ';', or '×'.

`grid size:` $x\ y\ z$ size of the grid in voxels. The numbers can be delimited also by ',', ';', or '×'.

`verbose:` on/off switches on or off verbose mode (the default is off).

`seed:` n sets the seed (an integer value) for the random number generator.

`surface distance:` D defines a distance (a real value) from the surface that all segment enpoints are trying to keep.

`max surface distance:` D maximum distance from the surface segment that an enpoint can have (usually around $2 * surface\_distance$).

`tries for Q:` n the number of randomly generated candidates for selecting a new position of the segment;s tip Q.

`tries for surface:` n the number of tries (rays) traced in order to find the closest surface.

`obstacles:` $filename$ the file $filename$ contains a specification of obstacles in aGLS format (see Section 3.2).

`add objects:` on/off If on, objects specified by the symbol following the communication module $?E$ are added to the grid (and removed at the end of each simulation step). The default is off. So far, the following modules (located just after $?E$) are recognized:

$S(rad)$	defines a sphere with radius rad;
$C(rad, height)$	defines a cylinder with radius rad and height $height$;
$C(rad_1, rad_2, height)$	defines a cone with base radius rad_1, top radius rad_2, and height $height$.

`remove objects:` on/off If on, all objects are removed from the grid at the beginning of each simulation step. The default is on.

1.1.2 Communication

Turtle parameters sent to the field: position, heading and up vectors.

Communication symbol

`with 0 parameters` adds an object to the grid according to the module following $?E$ (S—sphere, C—cylinder or cone).

`with 7 parameters` Parameters sent to the environment:

`1` desired segment length;

`2-7` not used (can be zeros or whatever).

Parameters set by the environment:

`1-3` the new heading vector (of a unit length);

`4` the length of the new segment. It is equal to 0 if the segment is not found. In that case, values of other parameters are undefined;

`5-7` the new up vector (basically the normal of the closest point on a surface).

`with 1 or 4 parameters` In this case, the communication symbol MUST be followed by F,f,G, or g! Parameters sent to the environment:

`1 or 1-4` ignored;

Parameters set by the environment:

`1` equal to 1 if there is an intersection of the segment with an object. Otherwise returns 0.

`2-4` the surface normal in the intersection point.

1.1.3 Algorithm

The program stores all incoming queries in a dynamically allocated array (prepared for the case that some queries will only add a new object - part of the plant).

After the string is processed, queries are answered. For each point P a new point Q at a given distance $length$ (specified by the communication symbol) is found, on a line perpendicular to the up vector. There is a certain number of tries for the point Q, for which the whole circle is swept starting in the heading direction and then +- angle where angle is increasing up to 180 degrees.

For each trial point Q a closest surface is found — a given number rays is shot and the closest intersection is sought. If it is not found a new point Q is generated. Otherwise, the new end point P is taken, specified as the intersection of the trial ray with the closest surface plus the normal vector in the intersection times the desired minimum distance from the surface. The field program then returns the new heading vector (P minus the turtle location T), its length $|P-T|$, and the up vector (the surface normal).

NOTE: In the case of 1 or 4 parameters, when only segment intersection is tested, the parameter $surface\ distance$ influences the returned intersection (this can be used to keep the plant a little away from the surface - to account for stem width). Just make sure that the size of one voxel is bigger than this $surface\ distance$.

Not finished: Well, out of all primitives in GLS format, so far it works only with cylinders, spheres, prisms, and rectangles.

1.1.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $arvo$ are located in node $environment$, subnode $arvo$.

1.2 Environmental program collisions

The program tests for collisions between balls of a given radius (specified by the first parameters of a communication module $?E$). If two balls collide, the program returns a force the other ball is initiating.

The program is rather specific and is used only in one type of models — simulations of a dynamic systems like a cluster of cherries, for example. Compare with program *honda81* (Section 1.4).

1.2.1 Execution

Name of the executable: collisions

Command line parameters:
collisions [-e environment_file] environment_argument_file

Commands in environment argument file

`verbose:` on/off switches on or off verbose mode (the default is off).

`radius:` rad defines a default radius of a ball.

1.2.2 Communication

Turtle parameters sent to the field: position.

Communication symbol

`with 2 or 3 parameters` Parameters sent to the environment:

`1` radius of the ball (if 0, the default radius is used);

`2-3` ignored.

Parameters set by the environment:

`1-2 or 1-3` force acting on the colliding ball.

1.2.3 Algorithm

The environment stores all balls corresponding to communication modules in a linked list. After all queries are inserted, the program goes through the list and computes the distance from a given ball to each other ball. If there is a colliding ball, the force acting on the tested ball (with index *t*) is computed as:

$$\vec{F}_t = \sum_{i \in C_\sqcup} 2 \cdot (C_t - C_i) \left(\frac{|C_t - C_i|}{r_t + r_i} - 1 \right)$$

where $\mathcal{C}_{\sqcup}$ is a set of balls colliding with ball t, and C and r denotes the center and the radius of a ball, respectively. The force is then return as the first two or three parameters of the communication module.

1.2.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $collisions$ are located in node $environment$, subnode $collisions$.

1.3 Environmental program ecosystem

This program determines collisions between a set of plants growing in a field. Internally, the plants are represented as disks with a given radius. If two disks representing two plants partially overlap, the plant with the lower radius (or possibly also lower vigor) is reported as colliding. It is also possible to determine the collisions in three dimensions, in which case the plants are represented as spheres.

1.3.1 Execution

Name of the executable: ecosystemR

Command line parameters:
ecosystemR [-e environment_file] environment_argument_file

Commands in environment argument file

`grid size:` $x\ y\ z$ Specifies the size of a regular grid used for reducing the time necessary for determining the colliding disks. The numbers can be delimited also by ',', ';', or '$\times$'. The third value is used in the case the plants are represented as spheres in three dimensions.

`verbose:` on/off switches on or off verbose mode (the default is off).

`vigor:` on/off if on, an additional parameter is send with each communication module. This parameter specifies the vigor of a plant. If two disks (or spheres) representing two plants collide, the plant with the lower vigor (even if its radius is bigger) is reported as colliding. The default is off.

`3d case:` $on(yes)/off(no)$ if on, each plant is represented by a sphere and the collision tests are performed in three dimensions. The default is off.

1.3.2 Communication

Turtle parameters sent to the field: position.

Communication symbol

`with 1 or 2 parameters` The communication symbol has to have at least one parameter if the vigor is not used or two parameters otherwise.

Parameters sent to the environment:

`1` radius of the disk (sphere) representing the plant;

`2` plant's vigor (if applicable).

Parameters set by the environment:

`1` 0 if the plant collides with another plants with the bigger radius or bigger vigor, 1 otherwise;

`2` unchanged.

1.3.3 Algorithm

Each plant is represented by a single communication module with at least one parameter (if the vigor is not used) or two parameters (if the vigor is used). Initially these modules are stored into a linked list. After all of them are processed (in a given simulation step), a regular grid of a given resolution is created so that it tightly encompasses all disks (or spheres) representing the plants. Note that the grid is rebuilt after each step.

The grid is used to speed up the collision tests. Each voxel of the grid contains a linked list of disks (spheres) occupying a portion of the voxel. For each disk it is then determined, whether any object stored in any of the voxels occupied by the given disk (or sphere) intersects this disk.

If a collision is found and the vigor is not considered, the program returns to the plant simulator `cpfg` a value 0, if the radius of the given disk is less than or equal to the radius of the colliding disk. Otherwise, it returns 1.

If a collision is found and the vigor is considered, the program returns to the plant simulator `cpfg` a value 0, if the vigor of the given disk is less than the radius of the colliding disk. If both vigors are equal, the program returns 0, if the radius of the given disk is less than or equal to the radius of the colliding disk. Otherwise, the program returns 1.

1.3.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $ecosystem$ are located in node $environment$, subnode $ecosystem$. Note that in /usr/u/vlab/bin/, the program is called $ecosystemR$.

1.4 Environmental program honda81

The program tests for collisions between leaf clusters, represented by disks of a fixed radius. If two clusters collide, the one with lower vigor (specified by the first parameters of a communication module $?E$) is marked for removing (by setting the first parameter of $?E$ to 0).

Compare with program *collisions* (Section 1.2).

1.4.1 Execution

Name of the executable: honda81

Command line parameters:
honda81 [-e environment_file] environment_argument_file

Commands in environment argument file

`verbose:` on/off switches on or off verbose mode (the default is off).

`radius:` rad defines the radius of a disk represending each leaf cluster (all clusters have the same radius, they differ only in vigor).

`3d case:` on/off if set to on instead of disks, each cluster is represented as a sphere. The default is off.

1.4.2 Communication

Turtle parameters sent to the field: position.

Communication symbol

`with 1 or 2 parameters` Parameters sent to the environment:

- `1` vigor of the cluster;
- `2` if present, specifies the index of the cluster (collisions are tested only with clusters of the same index).

Parameters set by the environment:

- `1` 0 if there is a collision, 1 otherwise.
- `2` unchanged.

1.4.3 Algorithm

The environment stores all queries corresponding to communication modules in a linked list. The first parameter of the communication module specifies the vigor of the segment. After all queries are inserted, the program goes through the list and computes the distance from a given query to each other having higher or equal vigor (it means that if a segment end point is obstructed by a leaflet with lower vigor it would continue growing — consequently $?E(1)$ will always stop growth of other branches). This repeats for each query. The cluster is tested only with clusters of the same index (if the second parameter of $?E$ is not present, the index defaults to 0).

The response is 0 (cannot grow further) or 1 (can grow).

1.4.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $honda81$ are located in node $environment$, subnode $honda81$.

1.5 Environmental program ulam

This program determines whether a point in the given set of two-dimensional points occupies the same place as other points. Basically, the program determines collisions in a discrete grid of points.

1.5.1 Execution

Name of the executable: ulam

Command line parameters: ulam [-e environment_file]

There is no environment argument file.

1.5.2 Communication

Turtle parameters sent to the field: position.

Communication symbol

`with 1 parameter` Parameters sent to the environment:

`1` not used.

Parameters set by the environment:

`1` value 0 if there is a collision, value 1 if the growth can continue.

1.5.3 Algorithm

The environment stores all queries, representing two dimensional points (the z coordinate is ignored) points, in a linked list. After all points are inserted, the program goes through the list and computes the distance from a given point to each other point. This repeats for each query.

The response is 0 (can't grow further) or 1 (can grow).

1.5.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $ulam$ are located in node $environment$, subnode $ulam$.

2 Light environment

2.1 Environmental program clover

This program determines the amount of direct light coming from the top, reaching apices (represented as points on the ground) of a clover. The light can be obstructed by leaves, stored as disks in a high-resolution grid.

2.1.1 Execution

Name of the executable: clover

Command line parameters:
clover [-e environment_file] environment_argument_file

Commands in environment argument file

`grid range:` x y specifies the range (in world coordinates) of a regular two-dimensional grid used to store leaves to speed up the light test. The numbers can be delimited also by ',', ';', or '×'.

`grid position:` $xpos$ $ypos$ specifies the position of the lower front left corner of the grid. The numbers can be delimited also by ',', ';', or '×'.

`grid size:` x y size of the grid in voxels. The numbers can be delimited also by ',', ';', or '×'.

`verbose:` on/off switches on or off verbose mode (the default is off).

`transmittance:` t transmittance of leaves (a value in the interval $(0, 1]$).

`input image:` $imagename$ defines the name of an image used to specify the light intensities coming from the top. Only the green channel of the image is used.

`remove old leaves:` yes/no if set to yes all leaves are removed from the grid after each simulation step. The default is no.

`z is up:` yes/no as a default, the z axis is considered up and the program uses only x and y coordinates for specifying location of leaves and queries. If the parameter `z is up` is set to no, the y coordinate is considered being up and the program uses x and z coordinates of each point.

2.1.2 Communication

Turtle parameters sent to the field: position.

Communication symbol

`with 1 parameter` Parameters sent to the environment:

`1-st` ignored

Parameters set by the environment:

`1-st` set to the light intensity reaching the point.

`with 2 parameters` Parameters sent to the environment:

`1-st` specifies the type of the operation. The recognized values are:

- `0` query the light reaching the point;
- `1` add a leaf with a specified leaf area (see the second parameter);
- `2` remove a leaf with a specified leaf area (see the second parameter).

`2-nd` specifies the leaf area. The leaves are assumed to be circular.

Parameters set by the environment:

`1-st` set to the light intensity reaching the point — only if the first parameter was set to 0, otherwise not changed.

`2-nd` set to 0 — only if the first parameter was set to 0, otherwise not changed.

2.1.3 Algorithm

As the program receives all communication modules in a given simulation step, it stores those that represent query for the amount of light in a linked list. If a query represents a leaf, which should be added to the data structures, this leaf is stored as a disk in a high-resolution grid (usually 2000×2000).

Each voxel of the grid contains information about the number of leaves obstructing the given voxel. Thus for a given leaf, the values in all voxels which are covered by a disk representing the leaf are incremented by one. Similarly, if a leaf is removed from the grid, values in the corresponding voxels are decreased by one.

In addition to the number of leaves obstructing the voxel, the voxel contains also the initial intensity of light. This intensity defaults to 1, but it can also be specified at the beginning of the simulation using an image file. In this case, the green channel of

the image specified the intensity at a given voxel. Note that the resolution of the grid does not have to match the resolution of the input image.

After the grid is updated, all queries, stored in the linked list, are processed one by one. For each query, a corresponding voxel is determined. The initial intensity associated with the voxel is then multiplied by a factor t^n, where t is the leaf transmittance and n is the number of leaves obstructing the voxel. The resulting value is then returned to `cpfg` as the intensity reaching the point of the query.

2.1.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $clover$ are located in node $environment$, subnode $clover$.

2.2 Environmental program chiba

This program determines the amount of direct light reaching leaf clusters represented as spheres (leaf balls). Based on a paper by Chiba *et. al.* [5].

2.2.1 Execution

Name of the executable: chiba

Command line parameters:
chiba [-e environment_file] environment_argument_file

Commands in environment argument file

`grid size:` $x\ y\ z$ specifies the size of the grid in voxels (the grid range in world coordinates is determined according to the location of leaf clusters so that the grid tightly encloses them). The numbers can be delimited also by ',', ';', or '$\times$'.

`verbose:` on/off switches on or off verbose mode (the default is off).

`number of samples:` n The parameter n specifies the number of samples of the light sphere. The closest higher number to $8 * k^4$ (where k is an integer) is taken. 128 is usually high enough.

`lower to upper ratio:` U (0.0 - 1.0) defines the ratio of intensities of lower and upper hemisphere. Values in the interval 0.7-0.9 are generally good. The default is 0.7.

`use CIE formula:` yes/no If set to no, all light sources in the same hemisphere have the same intensity. If set to yes, the light intensity of sources in the upper hemisphere is determined from the standard CIE formula for overcast sky [6] (based on the direction towards the light source). The default is no.

`source direction:` $x\ y\ z\ intensity$ defines a light source with a given intensity. This command have precedence over the previous 3 commands. After all sources are defined their intensities are normalized so that their sum is 1. Also, the direction is normalized.

`transmittance:` t (float) specifies the transmittance of the leaf balls (0.0-1.0). The default is 0.6.

`radius:` rad specifies the default leaf ball radius for those leaf clusters for which it is not explicitly given by *cpfg*. The default is 25.

`beam radius:` rad radius of a beam of rays traced to determine which leaf ball obstruct the light coming from the light sources. The radius is expressed as a part of the current leaf ball radius (i.e. rad is from the interval $\langle 0, 1 \rangle$). The default is 0.

`estimate intersection area:` on/off Determines the way other leaf balls affect the intensity reaching the leaf ball (see section Algorithm below for more details). The default is off.

2.2.2 Communication

Turtle parameters sent to the field: position.

Communication symbol

`with 1 or 4 parameters` Parameters sent to the environment:

- `1` if zero, radius defined in the environment argument file is taken, otherwise it specifies the radius;
- `2-4` not used (can be anything).

Parameters set by the environment:

- `1` percentage of light perceived by the centre of the leaf ball (0-1);
- `2-4` the brightest direction (of a unit length).

2.2.3 Algorithm

The program stores all incoming queries in a dynamically allocated array. After all queries are in a regular grid (usually of the size $64 \times 64 \times 64$) is built to speed up ray casting.

The environment then determines the amount of incoming light and the brightest direction (if the corresponding communication module have 4 parameters) for all queries. The amount of light is computed by shooting *number of samples* (in the case of a sky hemisphere) or *number of light sources* (when a fixed amount of light sources is specified) rays from the center of each leaf ball. Each intersected leaf reduces the perceived light intensity. The brightest direction is the sum of all sample rays multiplied by their intensities.

With `estimate intersection area` off, the ray is tested against a sphere with the radius $intersected\_leaf\_radius + beam\_radius$ and the function returns the length L of the line segment inside this sphere. The intensity associated with the ray is multiplied (reduced) by:

$$pow(transmittance, L/2 * intersected\_leaf\_radius).$$

Unfortunately, this formula is not very correct — at least it should be:

$$pow(transmittance, L/2.0 * (intersected\_leaf\_radius + beam\_radius))$$

but also the fixed value of the beam radius shouldn't be used with variable size leaf balls.

The better approach is then to switch the `estimate intersection area` on. In this case, the intersection function returns the ratio of the area of intersection of disks, resulting from projecting leaf clusters to a plane perpendicular to the ray, to the area of the projected leaf cluster (the one from whose center the ray is traced).

2.2.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $chiba$ are located in node $environment$, subnode $chiba$.

2.3 Environmental program takenaka

This program determines the amount of direct light reaching leaf clusters represented as spheres (leaf balls). The algorithm is based on Takenaka's paper [41].

2.3.1 Execution

Name of the executable: takenaka

Command line parameters:
takenaka [-e environment_file] environment_argument_file

Commands in environment argument file

`grid size:` $x\ y\ z$ specifies the size of the grid in voxels (the grid range in world coordinates is determined according to the location of leaf clusters so that the grid tightly encloses them). The numbers can be delimited also by ',', ';', or '$\times$'.

`verbose:` on/off switches on or off verbose mode (the default is off).

`parameter s:` $value$ this parameter controls the sparsity of leaf distribution on the leaf ball. If r is the radius of a sphere with surface area equal to the leaf area LA, then the radius of the leaf ball is $s \cdot r$. The default is 1.5.

`transmittance:` t transmittance of the leaf balls (0.0-1.0). The default is 0.1.

`efficiency:` e multiplicative parameter influencing the resulting light product [41]. The default is 0.015.

`source:` $x\ y\ z\ intensity$ defines a ligth source with a given intensity. After all sources are defined their intensities are normalized so that their sum is 1.

`beam radius:` rad radius of a beam of rays traced to determine which leaf ball obstruct the light coming from the light sources. The radius is expressed as a part of the current leaf ball radius (i.e. rad is from the interval $\langle 0, 1 \rangle$). The default is 0.

2.3.2 Communication

Turtle parameters sent to the field: position.

Communication symbol

`with 2 parameters` Parameters sent to the environment:

`1` leaf area (the area of the leaf ball);

2 amount of leaf product necessary for the leaf maintenance.

Parameters set by the environment:

1 unchanged (leaf area);

2 product of photosynthesis (after the maintenance cost is subtracted).

2.3.3 Algorithm

The program stores all incoming queries, representing the leaf balls, in a dynamically allocated array. After that a grid is build to speed up ray casting.

The program then determines the amount of incoming light for each leaf ball. The amount of light is computed by shooting a beam of rays from the leaf ball centre towards each light source. Each intersected leaf reduces the perceived light intensity. The weight of the final product is computed according to the appendix of the paper [41].

2.3.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $takenaka$ are located in node $environment$, subnode $takenaka$.

2.4 Environmental program MonteCarlo

This program determines the amount of light reaching objects in a scene. The objects are defined using the modules following the communication modules $?E$. The program `MonteCarlo` recognizes two objects, a triangle or a polygon, defined by modules T and P, respectively. In addition a set of polygons representing the module following $?E$ can be sent by the plant simulator.

The amount of light reaching all or selected objects is computed using the path tracing algorithm, based on Monte Carlo techniques. Generally, the light coming from the light sources (in form of rays) is traced through the scene. Every time the ray reaches an object, it is determined whether the light is absorbed, reflected, or transmitted through the object, based on the surface parameters associated with the object.

2.4.1 Execution

Name of the executable: MonteCarlo

Command line parameters:
MonteCarlo [-e environment_file] environment_argument_file

2.4.2 Commands in the environment argument file

Specifying the grid

`domain size:` $xrange\ yrange\ zrange$ specifies the range (in world coordinates) of a regular grid used to store objects for speed up the intersection test. The numbers can be delimited also by ',', ';', or '×'.

`position:` $xpos\ ypos\ zpos$ specifies the position of the lower front left corner of the grid. The numbers can be delimited also by ',', ';', or '×'.

`grid size:` $x\ y\ z$ size of the grid in voxels. The numbers can be delimited also by ',', ';', or '×'.

`remove objects:` $on(yes)/off(no)$ if set to on or yes (the default) all objects are removed from the grid after each simulation step.

`obstacles:` $filename$ additional objects are added to the grid. The objects are input from a text file containing GLS commands. There can be only one command `obstacles` in the environment argument file.

Controlling generation of initial rays

`light source:` $xpos\ ypos\ zpos\ weight$ defines a light source at a given point with a given weight. Several light sources can be specified by

including several commands `light source` in the environment argument file. In this case, the initial rays are generated according to the weights associated with the light sources.

`sky file:` $filename$ specifies the file defining intensities coming from all directions of a sky hemisphere. The file contains $n \times m$ numbers defining intensity of the sky at different directions (specified by two angles θ, ψ, where $\theta \in [0, \pi/2]$ and $\psi \in [-\pi, \pi)$. The file can be generated by program *computesky* (see Section 2.4.6). The command `sky file:` takes precedence over the command `light source`. If neither of these two commands is specified all initial rays are coming from the top.

`ray density:` n controls the number of initial rays. The density specifies how many rays will be generated per unit area. Thus the bigger the scene, the more rays will be generated. Make sure, that you properly change this value if you scale the scene up or down.

`stratified sampling:` $on(yes)/off(no)$ if on and all rays are coming from the top (in the case that neither of the commands `light source` and `sky file` is specified), the rays are generated using stratified sampling, which provides better distribution of rays.

`spectrum samples:` s this command specifies the number of wavelengths for which it is necessary to compute the light distribution. This command has to precede the command `source spectrum` and all commands defining materials!

`source spectrum:` $\lambda_1 \ w_1 \ \lambda_2 \ w_2 \ \ldots \ \lambda_s \ w_s$ specifies the wavelengths (currently not used) and weight for each spectral sample. The weight distribution is the same for all light sources.

`one ray per spectrum:` yes/no as a default, the light intensities reaching the objects for different wavelengths are computed separately, one set of initial rays for each wavelength. If this command is set to yes, a special algorithm is used, in which each rays carries the information about all wavelengths and the light intensities can be determined after one set of initial rays is traced (see below for more details).

`rays from objects:` yes/no as a default, the rays are initiated from the light sources or from the sky hemisphere. A large number of rays is usually necessary to obtain a sufficient precision of results (the amount of light reaching the objects). If there are only few objects, for which it is necessary to determine the light reaching them, it is more efficient to trace the rays backwards, from the object, towards the light source. This mode is switched on by specifying yes after the command `rays from objects` (see below for more details about the method).

Tracing of a ray

`periodic canopy:` $yes(on)/no(off)$ if on, the rays which leave the scene on the left, right, back, or front, are transformed to the opposite side of the scene and tre tracing continues. This simulates and infinite canopy with periodically repeating set of objects (make sure that the grid is tight enough not to have too big spaces between the sets). It is more suitable to consider only light coming directly from the top and not to use the option of tracing the ray from the objects (although both are not necessary). The default is off — in which case, if a ray leaves the scene, it is not traced any more.

`maximum depth:` d limits the number of hits (intersection of a ray with an object) for each ray. If set to -1, there is no limit. The default is 3. Note that is is better to set this limit high (*e.g.* 10) and to use the command `Russian roulette`.

`Russian roulette:` $thr\ \ prob$ an optional method used for determining the termination of rays. After each hit, the ray's intensity is compared with the threshold intensity thr (a value between 0 and 1). If it is below thr, the rays is terminated with the probability $prob$ (a value between 0 and 1). The intensity of rays which are not terminated is increased by factor $1/(1-prob)$.

`reflectance model:` $blinn/phong/parcinopy$ there are three different ways the reflected rays are generated (see below for more details). The Blinn-Phong model ($blinn$) is the best one to use. The default is $parcinopy$.

`no direct light:` $yes(on)/no(off)$ to compare the effect of reflected and transmitted light on objects, it is possible to set this parameter to yes and the direct light will not be included in the amount of light reaching an object. The default is no (or off).

Materials

`leaf material (top):` $r\ \ n_r\ \ t\ \ n_t\ \ n$ specifies the parameters of the default material. For each used wavelength, a set of 5 parameters has to be included: reflectance r (0-1), reflectance scattering exponent n_r (0-∞), transmittance t (0-1), transmittance scattering exponent n_t (0-∞), and refractive index n (ignored, but must be present!). To determine the number of spectral samples, the command `spectrum samples` has to be placed before the leaf material specification.

`leaf material (bottom):` $r\ \ n_r\ \ t\ \ n_t\ \ n$ if included, it specifies the bottom part of each surface. Otherwise, the top part is used for both sides.

`material:` $r\ n_r\ t\ n_t\ n$ specifies an additional material. The materials are indexed, the leaf top material has index 1, the leaf bottom index 2 and each subsequent material (specified using the command `material` has index 3, 4, and so on). The default material can be changed by the 3-rd and fourth parameter of modules T and P (following the communication module $?E$) and representing a triangle or polygon, respectively (see command `material parameter` below for other possibility).

`material parameter:` n in case that the polygons representing the module X following the communication module $?E$ are sent directly from the plant simulator, the value n specifies what parameter of the module X determines the index of the object's material (1 for the first parameter, 2 for the second parameter, *etc.*). If set to 0 (default), the default material is used and all parameters of the module X are ignored.

Miscellaneous commands

`verbose:` on/off switches on or off verbose mode (the default is off).

`seed:` v specifies the seed for the random number generator (an integer value).

`number of runs:` n if above 1 (default) the computation of light reaching objects is performed n times and the program returns the mean value and the standard deviation of the ratio of the light intensities for the first two wavelengths as the first and second parameter of the communication module associated with the object (thus at least two wavelengths have to be specified). If the communication module has more than two parameters, the program returns the mean intensity and the standard deviation for the first wavelength (3-rd and 4-th parameter of $?E$), second wavelength, and so on. Used to determine the precision of the algorithm for a given ray density.

`version:` ver if the parameter ver is bigger than 1, it is possible to specify the number of rays shot from each object, for which the light calculation has to be done (in the mode when rays are shot from objects — see command *rays from objects* above). The default version is 1.

2.4.3 Communication

Turtle parameters sent to the field: position, heading, and up. In case the rays are generated from objects, also the left vector has to be sent.

Communication symbol

`to environment` the plant simulator always sends the communication module and the module following it:

`communication module` All parameters of the communication module are ignored, unless the version is set to a value above 1 and the rays are shot from objects. In this case the parameters of $?E$ specify the number of rays per unit area shot from each object for each wavelength. Consequently, less rays can be traced for selected wavelengths (even 0).

`the following module` The module following the communication module $?E$ is used to specify the objects in the scene or to set the light spectrum. There are two ways how to define objects:

1. for a module P or T following the communication module $?E$, the environmental process constructs a parallelogram or a triangle of a specified orientation (based on turtle vectors). The size of the parallelogram or triangle is controlled by the parameters of the module P or T.
   Specifically, the module P defines a polygon constructed in the following way. The polygon is a parallelogram with one diagonal in the direction of the turtle heading vector $\vec{H}$ and length $2a$, and the second diagonal in the direction of the left vector $\vec{L}$ and length b. The turtle position $\vec{P}$ defines one vertex of the polygon.
   The module T defines a triangle constructed in the following way. One edge (of length a) of the triangle is in the direction of the turtle left vector. The middle point of this edge corresponds to the turtle position. The third vertex of the triangle is placed on an axis, corresponding to the turtle heading, at the distance b from the turtle's position.
   The third and fourth parameter of the module P or T, if present, specify the index of the material for the front and back sides of the polygon or triangle.
2. For any other module, the environment expects that the plant simulator sends a set of polygons representing the module. To define a complex organ, homomorphism productions can specify the geometry of the module and all the polygons representing the module are transferred to the environment, which considers them as part of a single object. The material indexes can be specified by the parameters of the module following the module $?E$, as specified by command `material parameter` in the environment argument file.
   This option takes precedence over the previous one, that is if the polygons are send for modules P or T, these polygons are used to define the object.

If a module L follows the communication module, its parameters set the current intensities for different wavelengths emitted from all light sources (these intensities are shared by all light sources). The param-

eters specify the intensities in the order given by values following the command `source spectrum` in the environment argument file.
If the rays are traced from objects, the the object (sensor) can be defined only as polygon P (using the first approach).

`from environment` Parameters set by the environment:

`single run` the first n parameters of the communication module $?E$ are set to the amount of light reaching the object, defined by the module following $?E$, for each of the n wavelengths.

`multiple runs` the first two parameters of the communication module $?E$ are set to the mean value and the standard deviation of the ratio of the amount of light reaching the object for the first two specified wavelengths (computed after the given number of runs). The next $2n$ parameters contain the mean value and the standard deviation of the amount of light reaching the object for each of the n wavelengths.

2.4.4 Algorithm

The environmental process first receives information about all communication modules and the geometry of the modules that follow. These modules, representing leaves, stems, the ground, or objects around the plant, are interpreted as a set of polygons which are transferred to the environment.

After all scene polygons are input, they are stored in a regular grid to speed up the algorithm. The computation of the light distribution starts with generating initial rays representing light of a certain power emitted from the light sources. Each initial ray is then traced in the grid. If the intersection with an object is detected, the light carried by the ray is either absorbed by the object, reflected by the surface, or transmitted through it. The fate of the ray depends on the parameters of the surface material specified in a specification file processed by the environmental program at the beginning of the simulation.

Each material is defined by four parameters, controlling the probability of tracing a reflected ray, its scattering coefficient (affecting the direction of the reflected ray), the probability of tracing a transmitted ray, and its scattering coefficient (see Appendix A.2.5 for more details).

Reflected or transmitted rays are traced further until the ray depth (equal to the number of hits on the ray's path) reaches the maximum user-specified value or its power is below a certain threshold and it is terminated according to a method called *Russian roulette* (see Appendix A.2.4). After all initial rays have been traced, parameters storing the light flux absorbed by an object are sent to the plant model.

2.4.5 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $MonteCarlo$ are located in node $environment$, subnode $MonteCarlo$.

2.4.6 Program computesky

The program *computesky* is used to generate a file with a density functions capturing the light intensities at different parts of the sky. The function is then directly used for generating initial rays in program *MonteCarlo* so that more rays are traced from the parts of the sky with higher intensity of light.

The density file corresponds to an average light intensity of a sky at a given location on Earth (the sun position during a day differs for different locations) for a given period of time (the sun is lower in the winter han in the summer). It would be interesting to consider a real time skylight distribution based on the timing of the plant simulation. Currently, all simulations have to use a static skylight distribution, which is computed by averaging light intensities over a certain time interval. Specifically, the program *computesky* takes as an input the longitude and latitude of the place of the simulated experiment, the first and the last date of the simulation, and the percentage of overcast days during the time period.

The intensities in a specified number of directions are averaged over the given time period. For each day it is randomly decided whether the day is cloudy or with a clear sky. If it is cloudy, the intensity is assumed constant throughout the day and is determined using the CIE formula (1), specified below. If the day is clear, it is subdivided into small time intervals, according to a user-specified time step, and at each time the intensities are computed using the CIE formula (2) for the correct sun position in a given time at the given position, specified below.

To compute the intensity of light at a given part of the sky, the following formula has been proposed by the CIE [6] (see also [27, 30, 33, 40]). The distribution on an overcast sky is relative to the zenith intensity, also called luminance, L_z. Intensity of light coming from a certain direction from an overcast sky is defined as luminance $L(\theta)$, where θ is the angle from zenith to the considered direction:

$$L_z = \frac{1}{0.203} \cdot (8.6 \cdot \cos\theta + 0.123)$$

$$L(\theta) = L_z \cdot \frac{1}{3} \cdot (1 + 2\cos\theta) \tag{1}$$

Formula (1) reflects the quantitative tests measuring the ratio of luminance $L(\theta)$ to zenith luminance $L_z = L(0)$ presented in [26].

For a clear sky, the distribution also depends on the sun position. The zenith luminance L_z is then a function of the angle θ_s of the sun from the zenith and a *turbidity* factor[1]:

$$L_z = \frac{1}{0.203} \cdot \left((1.367 \cdot turbidity - 1.81) \cdot \tan\left(\frac{\pi}{2} - \theta_s\right) + 0.38\right).$$

[1] As of now I do not have more information on this *turbidity* factor.

The luminance in direction (θ, φ) is:

$$L(\theta,\varphi) = L_z \cdot \frac{(0.91 + 10e^{-3\gamma} + 0.45 \cdot \cos^2\gamma)(1 - e^{-0.32/cos\theta})}{0.27385 \cdot (0.91 + 10e^{-3\theta_s} + 0.45 \cdot \cos^2\theta_s)} \qquad (2)$$

where θ is the angle from the zenith, φ is the azimuth angle going from 0 at north to $\frac{\pi}{2}$ at east, θ_s and φ_s are the zenith and azimuth angles of the sun, and γ is the angle between the sun direction and the direction (θ, φ) that can can be computed by:

$$\cos\gamma = \cos\theta_s \cdot \cos\theta + \sin\theta_s \cdot \sin\theta \cdot \cos(\varphi - \varphi_s).$$

The sun position is computed using a routine based on a java code written by Rafael Wiemker[2] based on a working paper No. 162 by B. K. P. Horn from March 1978. The input parameters of the routine are the latitude and longitude of a point on Earth, the date, and the time of the observation.

To be able to generate the direction of an initial ray according to the skylight distribution, it is necessary to compute the probability distribution function $F(\theta, \varphi)$ from the probability density function $p(\theta, \varphi)$ equal to normalized function of intensity:

$$p(\theta,\varphi) = \frac{I(\theta,\varphi)}{\int_{-\frac{\pi}{2}}^{-\frac{\pi}{2}} \int_0^{2\pi} I(\alpha,\beta) sin(\alpha)\, d\alpha d\beta}$$

The double integral in the denominator is summing the intensity values over the whole hemisphere. The distribution function F is then:

$$F(\theta,\varphi) = \int_{-\frac{\pi}{2}}^{\theta} \int_0^{\varphi} p(\theta,\varphi) \sin(\alpha)\, d\alpha d\beta$$

The distribution function is computed by the program *computesky* and output into a file as an array of values. The file is read by the simulation program.

The file format is described at the beginning of each density file. See an example object in node $environment$, subnode $MonteCarlo$, subnode $Clover$, file $portulaca.sky$.

Having the distribution function F, a random direction (θ, φ) can be generated according to function F [38]:

1. A uniformly distributed random number x in the interval $\left\langle F\left(\frac{\pi}{2}, 0\right), F\left(\frac{\pi}{2}, 2\pi\right)\right\rangle$ is selected and angle φ is found such that $F(\frac{\pi}{2}, \varphi) = x$. If function F is specified as an array of values, the angle φ is equal to φ_i such that $F(\frac{\pi}{2}, \varphi_i)$ is closest to x.

[2] http://kogs-www.informatik.uni-hamburg.de/ wiemker/

2. A uniformly distributed random number y in the interval $\langle F\left(-\frac{\pi}{2},\varphi\right), F\left(\frac{\pi}{2},\varphi\right)\rangle$ is selected and angle θ found such that $F(\theta,\varphi) = y$. If function F is specified as an array of values, angle θ is found by linear interpolation between values $F(\theta_i,\varphi) \leq y < F(\theta_{i+1},\varphi)$.

The number of initial rays N is determined similarly as in case of multiple light directions (see the previous section):

$$N = 2\pi r d \int_{-\frac{\pi}{2}}^{-\frac{\pi}{2}} \int_{0}^{2\pi} I(\alpha,\beta) sin(\alpha)\ d\alpha d\beta,$$

where r is the radius of a bounding sphere and d is the ray density. The light intensity is summed over the whole sky hemisphere.

The program *computesky* has the following command line parameters:

```
computesky [-verbose] [-from YY/DD/MM] [-to YY/DD/MM] [-pos
latitude longitude] [-tzone HH] [-clear %] [-withsun] [-samples
Nx Ny Mx My] [-tstep MM]
```

specifying:

- the verbose mode,
- the start and end date of the intensity averaging,
- the location on Earth,
- the time zone with respect to the European time (negative values going eastward),
- the percentage of the clear days during the given time interval, whether the sun itself is included in the sky (this option is not working properly at the time due to the lack of references in the literature),
- the number of samples $Nx \times Ny$ output in the density file with additional $Mx \times My$ samples computed between each 4 output points to increase the precision of the results,
- the time step in case a clear day is considered.

For example, the file used in most of the simulation was created using parameters:

```
computesky -v -from 88/07/01 -to 88/09/30 -pos 31.783 35.133
-tzone -1 -clear 80 -samples 100 100 4 4 -tstep 30
```

based on values inspired by [31] (location is in Izrael).

3 Diffusion environment

3.1 Environmental program axons

This program determines the value and gradient of a substance used for quiding the development of axons in retina. Based on a paper by Gierer [13].

3.1.1 Execution

Name of the executable: axons

Command line parameters:
axons [-e environment_file] environment_argument_file

Commands in environment argument file

`verbose:` on/off switches on or off verbose mode (the default is off).

`tectal surface:` $\alpha\ \beta\ u\ v$ four parameters specifying the function representing the concentration of a guiding substance p (see the algorithm below).

`range:` $xrange\ yrange$ specifies the range (in world coordinates) of the area considered for the output of an image. The numbers can be delimited also by ',', ';', or '$\times$'.

`position:` $xpos\ ypos\ zpos$ specifies the position of the lower front left corner of the area. The numbers can be delimited also by ',', ';', or '$\times$'.

`outputimage:` $min\ max\ xsize\ ysize\ imagename$ Controls the visualization of the given area of the field. The values min and max define the range which is convented into interval $\langle 0, 255\rangle$. Values $xsize$ and $ysize$ specify the size of the image output into file $imagename$.

`frame intervals:` $\{n|\ n_1 - n_2|\ n_3 - n_4\ \ step\ \ s_1\}$ to save time consuming output of an image after each step, specific frames can be selected. The step number is obtained from *cpfg*.

3.1.2 Communication

Turtle parameters sent to the field: position.

Communication symbol

`with 1 or 3 parameters` Parameters sent to the environment:

`1 or 1-3` all ignored.

Parameters set by the environment:

`1` amount of the guiding substance at the point

`2-3` gradient of the quiding substance

3.1.3 Algorithm

The environment provides the axon with a guiding parameter p representing the amount of a certain substance available at a given point. The production of the substance is affected by the concentration of other substances, one spreading along the x axis and one along the y axis, according to an exponential distribution with the center at point (u, v) [13]:

$$q_x(x) = e^{\alpha(u-x)/xrange},$$
$$q_y(y) = e^{\beta(v-y)/yrange}.$$

Parameters α and β control the steepness of the distribution curves. Values of $q_x(x)$ and $q_y(y)$ affect the production of the guiding substance by linear activation and superlinear inhibition:

$$p(x, y) = \frac{q_x(x)}{(1 + q_x(x))^2} + \frac{q_y(y)}{(1 + q_y(y))^2}.$$

The coordinates (gx, gy) of the field gradient at a point (x, y) are computed approximately by averaging values $p(x + \Delta, y)$, $p(x - \Delta, y)$ and $p(x, y + \Delta)$, $p(x, y - \Delta)$:

$$\begin{aligned} gx = &\ (p(x + \Delta, y) - p(x - \Delta, y))/(2 * \Delta) \\ = &\ \left(\frac{q_x(x+\Delta)}{(1+q_x(x+\Delta))^2} - \frac{q_x(x-\Delta)}{(1+q_x(x-\Delta))^2}\right)/(2 * \Delta) \\ gy = &\ (p(x, y + \Delta) - p(x, y - \Delta))/(2 * \Delta) \\ = &\ \left(\frac{q_y(y+\Delta)}{(1+q_y(y+\Delta))^2} - \frac{q_y(y-\Delta)}{(1+q_y(y-\Delta))^2}\right)/(2 * \Delta). \end{aligned}$$

3.1.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $axons$ are located in node $environment$, subnode $axons$.

3.2 Environmental program soil

This program simulates diffusion processes in the soil together with a simple obstacle avoiding mechanism.

3.2.1 Execution

Name of the executable: soil

Command line parameters:
soil [-e environment_file] environment_argument_file

3.2.2 Commands in environment argument file

General

`domain size:` $xrange\ yrange\ zrange$ specifies the range of a grid used for the simulation of the diffusion (in world coordinates). The values can be delimited also by ',', ';', or $\times$.

`position:` $xpos\ ypos\ zpos$ specifies the position of the lower front left corner of the grid. The values can be delimited also by ',', ';', or $\times$.

`verbose:` on/off switches on or off verbose mode.

Input

`image:` $min\ max\ input\_image$ [$output\_image$] specifies an rgb image file from which all input values are read and converted into a range (min, max). The values of the field (in this case 2-dimensional) can be output into image specified as output_image. The size of the grid (in voxels) in this case is $image\_xsize \times image\_ysize \times 1$.

`array:` $xsize\ ysize\ zsize\ min\ max$ [$output\_image$] specifies the size of a grid (in voxels), its minimal and maximal value, and an optional output image name. This command is followed by a list of values (starting on the following line) which will be stored in the grid ($xsize \cdot ysize \cdot zsize$ values).

`layer thicknesses:` $th1\ +\ th2$ `...` defines specific layers in which the geotropic angle may differ (by a set of thicknesses — along the axis y) [7].

`geotropic angles:` $a1\ a2$ `...` Specifies a geotropic angle for each layer (see [7]). This command should be after the command `layer thicknesses`!

`3D primitives:` $xsize \times ysize \times zsize\ \ min\ \ max\ \ file1\ [file2\ ...]\ out\_file$ specifies the size of the grid (in voxels), its minimal and maximal value, a set of input files (containing a set of object defined in the GLS format — see below), and an output filename. Regardless of the parameter set by the command `obstacles and sources`, opaque objects represent obstacles and transparent objects define areas of concentration equal to $1 - alpha$, where $alpha$ is the fouth channel associated with the surface diffuse color. If $alpha$ is equal to 0, the object is a source.

If the input file defines a rectangle of a size is $a \times b$ and the output file is specified, the program *soil* outputs a set of rectangles of size 1×1 (using the GLS format), each followed by a value specifying the concentration withing the range (min, max). Otherwise the output file stores a set of triangles representing a transparent surface (contour) of a given concentration (see the command `concentration contour value` below).

Diffusion control

`diffusion:` $step\ \ [tolerance]$ if step is -1, diffusion is simulated until all values in one step are not changed by more than the given tolerance (the default tolerance is 0.001). Otherwise the program performs $step$ steps of diffusion. The diffusion can be switched off by setting the step to 0 (the default).

`depletion:` on/off if on, a root can actually reduce a concentration in a particular voxel. Otherwise, the concentration stays unchanged. The default is on.

`relaxation factor:` $omega$ controls the speed of diffusion (see [22]). The default is 0.5.

`keep depleted cells:` on/off It is more correct (as found later) to keep values of the depleted cells constant (the parameter is on) throughout the diffusion. The dafualt is off (for backward compatibility with older models).

`obstacles and sources:` on/off Influences only 2d mode, when the field is input using the command `image` or `array`.

If on, white color specifies sources (they have always concentration 1), color (40,55,70) represents obstacles. Otherwise the green component of a color specifies concentration levels. In the case of `array`, voxels with values equal to min are obstacles, and those equal to max are sources.

If off (the default), each color is converted to grey and its intensity specifies the concentration.

Output

`frame intervals:` $\{n|\ n_1 - n_2|\ n_3 - n_4\ \ step\ \ s_1\}$ to save time consuming output of an image after each step, specific frames can be selected. The step number is obtained from *cpfg*.

`output normals:` on/off (default off) in case of triangle output normals may be included for Gouraud shading

`3D output:` $polygons/triangles$ (default polygons:) the contour is output either as a collection of triangles approximating the surface or polygons representing voxel faces close to the surface

`concentration contour value:` $val\ \ alpha$ The parameter val defines the value of the implicit field on the output surface (the surface is defined by all field points with the given value). The parameter $alpha$ specifies the alpha value of the default contour material (a woody looking material):

$ambient\ 0.1\ 0.1\ 0.1\ alpha$
$diffuse\ 0.1\ 0.6\ 0\ alpha$
$specular\ 0.5\ 0.5\ 0.5\ alpha$
$emissive\ 0\ 0\ 0\ alpha$
$spec\ exponent\ 25$

`contour material:` `17` $floats$ specifies the contour material (in the order as the example above)

`source material:` `17` $floats$ specifies the material of a source (because sources are specified with alpha 0)

`section material:` `17` $floats$ material of the rectangular sections (2d output along given rectangles).

3.2.3 Communication

Turtle parameters sent to the field: position (2d or 3d), heading (required only if there is a communication module with more than 4 parameters — used for obstacle avoiding).

Communication symbol

`with 1,3,6 parameters (2D case)`

Parameters sent to the environment:

`1` desired amount of nutrients;

`2` geotropic weight (between 0 and 1);

`3` soil gradient vector (between 0 and 1);

`4` ignored;

`5` internode length (necessary only for obstacle avoiding);

`6` an angle by which the current heading is diverted from the previous heading.

Parameters set by the environment:

`1` received amount of nutrients;

`2-3` the sum of weighted geotropic and gradient vector;

`4` set to 0;

`5` returns the new heading angle by which the current heading should be diverted from the previous heading in order to avoid obstacles. Returns 0 and prints a warning if such angle is not found.

`6` unchanged.

`with 1,4,7 parameters (3D case)`

Parameters sent to the environment:

`1` desired amount of nutrients;

`2` geotropic weight (between 0 and 1);

`3` soil gradient vector (between 0 and 1);

`4` ignored;

`5` internode length (necessary only for obstacle avoiding);

`6-7` ignored.

Parameters set by the environment:

`1` received amount of nutrients;

`2-4` the sum of weighted geotropic and gradient vector;

`5-7` returns the new heading vector to avoid obstacles or (0,0,0) if such vector is not found.

3.2.4 Algorithm

After the program *soil* is started, it processes the environment argument file and sets up its data structures.

During the simulation, each communication module is processed as soon as it is received by the program *soil*. At first, the position of the module in the grid is determined. Then, according to the value of the depletion parameter (see above), either the amount of nutrients present in the voxel is returned or the lower of the desired amount of nutrients and the amount in the voxel.

According to the actual geotropic layer, the desired geotropic direction is determined and added to the weighted field gradient in the given point. This vector (note that the gradient is not normalized!) is returned as the second and third (and in the 3d case also fourth) parameters of the communication module.

In the 2D case ($zsize$ in the environment argument file command `array` is 1 or the command `image` is used), 2d obstacle avoiding is applied to modules which have more than 3 parameters. The end point of the internode segment (the length must be sent to the environment as the 4-th parameter of $?E$) is tested whether it is inside or outside of an obstacle. If it is inside is is rotated by different angles to find the first available position. The angles are +2, -2, +4, -4 *etc.* up to +180, -180 degrees. Unfortunately, this sequence of angles is fixed. Moreover, some longer segments may intersect corners of obstacles.

If the 3D case, 3D obstacle avoiding is applied to modules which have more than 3 parameters. It works the same as 2D avoiding where the plane of rotation is given by the heading vector and the surface normal closest to the end point (which is inside, otherwise the segment is would not intersect an obstacle).

Diffusion is simulated according to [22].

Tip: use the command `frame intervals` to save only those contours you need (*e.g.* the last one).

3.2.5 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $soil$ are located in node $environment$, subnode $soil$.

4 Miscellaneous environments

4.1 Environmental program implicit

This environmental program creates a contour surface defined as an implicit surface around a set of skeleton points. The surface is polygonized and output to a file in the GLS format.

4.1.1 Execution

Name of the executable: implicit

Command line parameters:
implicit [-e environment_file] environment_argument_file

Commands in environment argument file

`domain size:` $xrange\ yrange\ zrange$ specifies the range (in world coordinates) of a regular grid used to store objects for the intersection test. The numbers can be delimited also by ',', ';', or '$\times$'.

`position:` $xpos\ ypos\ zpos$ specifies the position of the lower front left corner of the grid. The numbers can be delimited also by ',', ';', or '$\times$'.

`3D primitives:` $xsize \times ysize \times zsize\ filename$ specifies the size of the grid in voxels and a filename for the output of the contour in the GLS format (see Section 3.2).

`3D output:` $polygons/triangles$ the contour is output either as a collection of triangles approximating the surface or polygons representing voxel faces close to the surface. The default is $polygons$.

`output normals:` on/off in case of triangle output, the normals at each vertex may be included for Gouraud shading. The default is off.

`concentration contour value:` $val\ alpha$ The parameter val defines the value of the implicit field defining the contour surface. The parameter $alpha$ specifies the alpha value of the default contour material (woody looking):

```
ambient  0.1 0.1 0.1 alpha
diffuse  0.1 0.6 0   alpha
specular 0.5 0.5 0.5 alpha
emissive 0   0   0   alpha
spec exponent   25
```

`contour material:` $17\ values$ specifies the contour material (17 values in the order shown in the example above).

`frame intervals:` $\{n|\ n_1 - n_2|\ n_3 - n_4\ \ step\ \ s_1\}$ to save time consuming output of a contour after each step, specific frames can be selected. The step number is obtained from *cpfg*.

`verbose:` on/off switches on or off verbose mode (the default is off).

4.1.2 Communication

Turtle parameters sent to the field: position.

Communication symbol

`with 1 parameter` adds an object to the grid according to the module.

Parameters sent to the environment:

`1` radius of influence of the point.

Parameters set by the environment:

`1` the environment leaves the original value intact.

4.1.3 Algorithm

Program updates the grid of implicit field values after each point is processed by going through all voxels in the point's area of influence and modifying the field values in the centre points of all these voxel. When required the contour surface of a specific value is saved in a GLS format (either as triangles or polygons — see above).

Tips:

1. Instead of gradually increasing the radius of influence of a point, it is much more effective to set the resulting radius at the beginning. It saves time of updating the area of influence around such a point.

2. Use frame interval to save only those contours you need (e.g the last one).

4.1.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $implicit$ are located in node $environment$, subnode $implicit$.

4.2 Environmental program multiple

This program distributes incoming communication modules to several environmental programs specified in the specification file. The first parameter of each communication module defines which to environmental program the module and associated data will be transferred.

4.2.1 Execution

Name of the executable: multiple

Command line parameters:
multiple [-e environment_file] environment_argument_file

Commands in environment argument file

`field communication file:` $file.e$ this command specifies the communication specification file ($file.e$) that defines the communication link to an environmental program (the executable is defined in this file). Several commands `field communication file:` can be included. Note that their order in the environment argument file is important because the connections are indexed according to the order of these commands.

`verbose:` on/off switches on or off verbose mode (the default is off).

4.2.2 Communication

Turtle parameters sent to the field: Make sure that you include all parameters needed by the environmental programs connected to the program *multiple*.

Communication symbol

`with at least one parameter` Parameters sent to the environment:

`1` index of the environmental program (according to the order of commands `field communication file` in the environment argument file);

`2-` all subsequent parameters are shifted by one to the right (the second parameter becomes the first, *etc.*) and send to the particular environment.

Parameters set by the environment:

`1` unchanged (the index of the environmental program);

2- these parameters are set to the values of the parameters of the environmental module received from the given environment (shifted to the left — the first parameter becomes the second, *etc.*).

4.2.3 Algorithm

After the execution the program reads in the environment argument file and establishes connections to all given environmental programs. Afterwards, it distributes all environmental modules and any additional data associated with them to environmental programs. The first parameter of each environmental module specifies the index of the environmental program to which the module (without the first parameter) and all associated data are sent.

When a response is received from a given environment, it is send back to the plant simulator *cpfg* (after the first parameter is added back).

4.2.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $multiple$ are located in node $environment$, subnode $multiple$.

4.3 Environmental program terrain

This program determines the altitude and optionally the normal and a water content at a given point (specified by x and z coordinate). The terrain specification is read from a file using a routine developed by Matt Pharr ($http://graphics.stanford.edu/\sim mmp/eco$).

4.3.1 Execution

Name of the executable: terrain

Command line parameters:
terrain [-e environment_file] environment_argument_file

Commands in environment argument file

`terrain file:` $filename$ specifies the name of the terrain file. All currently used terrains are in $/home/jungle/mech/vlab/terrains$.

`verbose:` on/off switches on or off verbose mode (the default is off).

4.3.2 Communication

Turtle parameters sent to the field: position.

Communication symbol

`with 1-5 parameters` All parameters sent to the environment are ignored.

Parameters set by the environment:

`1` the latitude (y coordinate) at the point;

`2-4` if present, the normal at the point;

`5` if present, the water content at the point.

4.3.3 Algorithm

At its initialization, the program reads in the terrain specification from a given file. For each incoming communication module (representing a query), the program calls a routine provided by Matt Pharr (downloadable from $http://graphics.stanford.edu/mmp/eco/$) and returns the latitude, and possibly also the normal and water content at the point.

Note that this environment is the only one which uses the communication library function $MainLoop$, because the answer to each query can be processed immediately, independently of other queries. The source is in

$$/usr/u/vlab/src/ENVIRO/FIELDS/terrain.$$

4.3.4 Sample objects

Start hbrowser in /home/jungle/mech/vlab. The sample objects using the environmental program $terrain$ are located in node $environment$, subnode $terrain$.

5 Undocumented environmental programs

This section lists environmental programs which are not documented in this manual.

- `density` — a special purpose program used for a model of algae-like branching structures based on a paper by Cohen [8] (the model is presented in [25]). The program determines density in a given point as a sum of reciprocals of distances from end points of branches.
- `density3d` — used to reduce the density of branches for the plant climbing on a chestnut. The program uses a simple grid or an octree to count the number of leaves and/or apices appearing at a given location (voxel) and if a new apex grows into too dense area, its development is terminated.
- `dla` — used to simulate the diffusion of substances around a growing branching structure. The model is based on paper by Vaario [42] and it is described in detail in [25].
- `dla.old` — an older version of the program `dla`.
- `genetic1` — a special purpose program used to compute an effective leaf area for a model of branch tiers based on a paper by Fisher and Honda [11].
- `graphics_test` — used to illustrate the transfer of geometry between `cpfg` and environmental programs.
- `greene` — a special purpose program used for a model of trees generated by growing strand around a predefined skeleton. When growing the strand is tightly following the skeleton or an existing strands. Based on papers by Greene [17, 18].
- `molecule` — used to compute forces acting on a single atom in a molecule enabling the molecule to fold itself to an energy-minimized configuration.
- `ornament` — an extension of program `ulam` testing for a collision between end point of branches. In addition, the program `ornament` can input an image file and the growth is then allowed only in areas with black pixels.
- `parci` — a simple environmental program that processes the information coming from `cpfg` and runs program `parcinopy`. The program `parcinopy` was developed by Michael Chelle and it determines the amount of light reaching plant organs (considering also light reflected from and transmitted though surfaces, *e.g.* plant leaves). Since the program `parcinopy` can be executed only on suns, the simulation is distributed, *i.e.*`cpfg` runs on an SGI and `parcinopy` on a sun (see the sample models in $hofs/environment/parci/...$).
- `radia` — an attempt to interface `cpfg` with a radiosity program called `radiance`. The program was not suitable for our purposes and we do not have it any more.

- `skeleton` — contains the function calls necessary for the communication. It is very useful as a starting point for creating new environmental programs.

Examples of all programs mentioned above (except the `skeleton`) can be found in children of the node *environment* (you have to start the `hbrowser` in */home/jungle/mech/vlab*).

The source code of all programs listed in this manual can be found in */home/jungle/mech/src/ENVIRO/fields* or */usr/u/vlab/src/src/ENVIRO/FIELDS*.

References

[1] ARVO, J., AND KIRK, D. Modeling plants with environment-sensitive automata. *Proceedings of Ausgraph '88* (1988), 27–33.

[2] ARVO, J., AND KIRK, D. Particle transport and image synthesis. *Computer Graphics 24*, 4 (1990), 63–66.

[3] CHELLE, M. *Développement d'un modèle de radiosité mixte pour simuler la distribution du rayonnement dans les couverts végétaux*. PhD thesis, Université de Rennes I, 1997.

[4] CHELLE, M., MĚCH, R., AND PRUSINKIEWICZ, P. Comparison or two different radiative approaches, Monte Carlo ray tracing and radiosity, to compute the distribution of light in canopies. Manuscript, May 1997.

[5] CHIBA, N., OHKAWA, S., MURAOKA, K., AND MIURA, M. Visual simulation of botanical trees based on virtual heliotropism and dormancy break. *The Journal of Visualization and Computer Animation 5* (1994), 3–15.

[6] CIE Technical Committee 4.2: Standardization of Luminance Distribution on Clear Skies, 1973. Comission International de l'Eclairaze, Paris.

[7] CLAUSNITZER, V., AND HOPMANS, J. W. Simultaneous modeling of transient three-dimensional root growth and soild water flow. *Plant and Soil 164* (1994), 299–314.

[8] COHEN, D. Computer simulation of biological pattern generation processes. *Nature 216* (October 1967), 246–248.

[9] COHEN, M. F., CHEN, S. E., WALLACE, J. R., AND GREENBERG, J. R. A progressive refinement approach to fast radiosity image generation. *Computer Graphics (SIGGRAPH '88 Conference Proceedings) 22* (1988), 75–84.

[10] COHEN, M. F., AND GREENBERG, D. P. The hemi-cube: a radiosity solution for complex environments. *Computer Graphics 19*, 3 (1985), 31–40.

[11] FISHER, J. B., AND HONDA, H. Tree branch angle: Maximizing effective leaf area. *Science 199* (1977), 888–890.

[12] FOLEY, J. D., VAN DAM, A., FEINER, S., AND HUGHES, J. *Computer graphics: Principles and practice.* Addison-Wesley, Reading, 1990.

[13] GIERER, A. Directional cues for growing axons forming the retinotectal projection. *Development 101* (1987), 479–489.

[14] GLASSNER, A. S. *Principles of Digital Image Synthesis*, vol. 2. Morgan Kaufmann Publishers, San Francisco, California, 1995.

[15] GORAL, C. M., TORRANCE, K. E., AND GREENBERG, D. P. Modeling the interaction of light between diffuse surfaces. *Computer Graphics 18*, 3 (1984), 213–222.

[16] GOVAERTS, Y. M. *A model of light scattering in three-dimensional plant canopies: A Monte Carlo ray tracing approach*. PhD thesis, Université Catholique de Louvain, 1995.

[17] GREENE, N. Voxel space automata: modeling with stochastic growth processes in voxel space. *Computer Graphics 23*, 4 (1989), 175–184.

[18] GREENE, N. Detailing tree skeletons with voxel automata. *SIGGRAPH'91 course notes on photorealistic volume modeling and rendering techniques* (1991), 7:1–7:15.

[19] HALL, R. A., AND GREENBERG, D. P. A testbed for realistic image synthesis. *IEEE Computer Graphics and Applications 8*, 3 (1983).

[20] HANRAHAN, P. M., AND SALZMAN, D. A rapid hierarchical radiosity algorithm for unoccluded environments. *Eurographics Workshop on Photosimulation, Ralism and Physics in Computer Graphics* (June 1990).

[21] HANRAHAN, P. M., SALZMAN, D., AND AUPERLE, L. A rapid hierarchical radiosity algorithm. *Computer Graphics (SIGGRAPH '91 Conference Proceedings) 25* (1991), 197–206.

[22] KAANDORP, J. A. *Fractal Modelling. Growth and Form in Biology*. Sringer-Verlag, Berlin, 1994.

[23] KAJIYA, J. T. The rendering equation. *Computer Graphics (SIGGRAPH '86 Conference Proceedings) 20*, 4 (August 1986), 143–150.

[24] KALOS, M. H., AND WHITLOCK, P. A. *Monte Carlo Methods*. John Wiley & Sons, New York, 1986.

[25] MĚCH, R. *Modeling and Simulation of the Interaction of Plants with the Environment using L-systems and their Extensions*. PhD thesis, The University of Calgary, Calgary, Canada, November 1997.

[26] MOON, P., AND SPENCER, D. E. Illumination from a non-uniform sky. *Illumination Engineering 37* (1942), 707–726.

[27] MÜLLER, S., KRESSE, W., GATENBY, N., AND SCHÖFFEL, F. A radiosity approach for the simulation of daylight. In *Proceedings of Conference on Rendering Techniques* (1995), pp. 137–146.

[28] MYNENI, R. B., MARSHAK, A. K., KNYAZIKHIN, Y., AND ASRAR, G. Discrete ordinates method for photon transport in leaf canopies. In *Photon-Vegetation Interactions - Applications in Optical Remote Sensing and Plant Ecology*, R. Myneni and J. Ross, Eds. Springer-Verlag, Berlin, 1991, ch. 3, pp. 45–109.

[29] NISHITA, T., AND NAKAMAE, E. Continuous tone representation of three dimensional objects taking account of shadows and interreflection. *Computer Graphics 19*, 3 (1985), 61–67.

[30] NISHITA, T., AND NAKAMAE, E. Continuous tone representation of three dimensional objects illuminated by sky light. *Computer Graphics 20*, 4 (1986), 125–132.

[31] NOVOPLANSKY, A., COHEN, D., AND SACHS, T. How portulaca seedlings avoid their neighbours. *Oecologia 82* (1990), 490–493.

[32] PATTANAIK, S. N. *Computational Methods for Global Illumination and Visualisation of Complex 3D Environments*. PhD thesis, Birla Institute of Technology and Science, INDIA, 1993.

[33] PEREZ, R., SEALS, R., AND MICHALSKY, J. All-weather model for sky luminance distribution—preliminary configuration and validation. *Solar Energy 50*, 3 (1993), 235–245.

[34] SAMET, H. *Applications of Spatial Data structures*. Addison-Wesley, 1990.

[35] SHIRLEY, P. *Physically Based Lighting Calculations for Computer Graphics*. PhD thesis, University of Illinois at Urbana-Champaign, 1990.

[36] SHIRLEY, P. A ray tracing method for illumination calculation in diffuse-specular scenes. *Proceedings of Graphics Interface '90* (1990), 205–212.

[37] SHREIDER, Y. A. *The Monte Carlo Method*. Pergamon Press, New York, 1966.

[38] SILLION, F. X., AND PUECH, C. *Radiosity and global illumination*. Morgan Kaufmann Publishers, San Francisco, California, 1994.

[39] SOBOL', I. M. *The Monte Carlo method*. The University of Chicago Press, 1974.

[40] TAKAGI, A., TAKAOKA, H., OSHIMA, T., AND OGATA, Y. Accurate rendering technique based on colorimetric conception. *Computer Graphics 24*, 4 (1990), 263–272.

[41] TAKENAKA, A. A simulation model of tree architecture development based on growth response to local light environment. *Journal of Plant Research 107* (1994), 321–330.

[42] VAARIO, J., OGATA, N., AND SHIMOHARA, K. Synthesis of environment directed and genetic growth. To appear in the proceedings of the Artificial Life V conference, held in Nara, Japan, May 16–18, 1996. Included in *ALIFE V oral presentations* (preliminary version of the proceedings), pp. 207–214.

[43] VERHOEF, W. Light scattering by leaf layers with application to reflectance canopy modeling: the sail model. *Remote Sensing of Environment 16* (1984), 125–141.

[44] VERHOEF, W. Earth observation modeling based on layer scattering matrices. *Remote Sensing of Environment 17* (1985), 164–178.

A Program for computing light distribution

This appendix presents a method for computing the amount of light reaching plant organs. The environmental program described below computes both the direct and indirect light reflected or transmitted by other surfaces. The program is based on the Monte Carlo algorithm for estimating a rendering equation [23] whose solution determines the amount of light, both direct and indirect, reaching each object in the scene.

A.1 Background

The rendering equation defined by Kajiya [23] expresses the light intensity $I(x, x')$ passing from point x' of a surface to an arbitrary point x in space. The rendering equation is (from [23]):

$$I(x, x') = g(x, x') \left[\epsilon(x, x') + \int_S \rho(x, x', x'') I(x', x'') dx'' \right]. \quad (3)$$

The equation simply balances energy exchanges between surfaces. The intensity $I(x, x')$ is equal to the sum (integral) of the light intensity $\epsilon(x, x')$ emitted from a surface at point x' towards x and the total light intensity $I(x', x'')$ coming from all points x'' on all surfaces (S) to the point x' and scattered by the surface at point x' (as defined by the scattering coefficient $\rho(x, x', x'')$). The factor $g(x, x')$ is a "geometry" term expressing the visibility of point x from x'.

There are two commonly used techniques for solving the rendering equation. In the *radiosity* method [3, 10, 15, 29], surfaces are subdivided into smaller patches, which are assumed to diffusely emit and reflect light of a constant intensity throughout their surface. The rendering equation then expresses the light flux reaching a patch as a sum of the light emitted from all other patches, while considering occlusions. Solving the system of N equations for N patches yields light intensities reaching each patch.

The original radiosity method is very time consuming. First, it is necessary to compute a visibility factor (a *form factor*) for each pair of patches. This factor expresses what portion of one patch is visible from the other patch. An efficient method determines form factors for a patch by projecting all other patches onto a hemisphere or hemicube defined around the patch. It can be faster to project $N - 1$ patches onto N spheres or $5N$ hemicube faces than to compute directly the form factors for $N(N - 1)/2$ pairs of objects. Second, it is very time consuming to solve the system of N equations for very large values of N, even if the system is solved by iteration. To this end, the radiosity method has been extended by introducing a technique called *progressive refinement* [9], in which the form factors are determined only when they are needed for computing the energy passing from a surface to another surface, and surfaces are processed in order of importance with respect to the overall balance of energy. To reduce the total number of computed form factors, a hierarchical algorithm attempts to establish energy transfers between mesh elements of varying size, thus reducing the subdivision of surfaces [20, 21].

Many techniques that make the radiosity method more efficient are motivated by the use of the method for rendering of realistical looking scenes. In the scope of this thesis, however, we intend to find the solution of the rendering equation for the purpose of calculating the distribution of radiative energy in a canopy. Specifically, we are interested in light intensities reaching selected surfaces (*e.g.* leaves) as opposed to generating a view of the scene from a given point. Thus it is possible to take advantage of the special properties of the considered scenes and to improve the efficiency of the radiosity method by making a few assumptions.

A recent work of Chelle addresses this issue by extending the radiosity method specifically for calculating the distribution of light in a canopy [3]. In his method, the number of unknowns in the rendering equations can be reduced in the case of more or less uniform canopies, *e.g.* of a corn field. First, the canopy is divided into horizontal layers of constant light characteristics and a *turbid medium* model [43, 44] is applied to efficiently compute the light fluxes between the layers. In the second step, the radiosity equation is solved for primitives inside a sphere of a certain radius, while the light coming from the outside of the sphere is determined from the fluxes at a given layer (computed using the turbid medium model). This approach significantly improves the performance of the radiosity algorithm while only slightly reducing the accuracy of the results [3].

The other common algorithm solves the rendering equation (3) using *Monte Carlo* methods — techniques relying on random processes [24, 35, 37, 39]. In the context of the rendering equation, they can be applied in two ways [38]:

- by evaluating the integral in the rendering equation using stochastic approximation techniques in a purely mathematic way with no special adjustment for the purpose of calculating the light distribution in the scene.
- by following the path of light, originating from a light source. The light is traced using the approach of a "random walk", *i.e.*every time the light reaches a surface, its fate is decided randomly (see below for more details). This method, called *path tracing*, was first introduced by Kajiya [23] (see also [14, 35]). Path tracing has been effectively applied to calculating the distribution of light in a canopy [3, 16].

In the path tracing algorithm, light particles (or a group of particles — "photon bundles" [38]) are emitted from light sources in different directions, chosen at random. The paths of the particles are traced individually as rays. If a ray hits an object, the light associated with it (expressed as a radiant power [38]) can be completely absorbed, reflected from or transmitted through the object. The selection between these three options is made randomly according to the surface parameters (specifying what portion of the incoming light is absorbed, reflected, or transmitted). The precision of the algorithm depends on the number of traced rays. For a desired variance of error, it is possible to compute the minimum number of required rays [14, Section 18.8]). After

all rays have been traced, the amount of light absorbed by each object is calculated as the sum of radiant powers of all rays that were terminated at this object.

A comparison of the implementation of Chelle's extended radiosity algorithm and a Monte Carlo algorithm, made by Chelle, Prusinkiewicz, and myself [4], indicates that these approaches generate similar results in a similar time. Consequently, I have chosen the Monte Carlo algorithm as a base for the environmental program described below.

A.2 Operation of the model of the environment

The environmental program *MonteCarlo* is based on my implementation of the Monte Carlo algorithm, operating under the following assumptions. The program deals only with polygons, because it is used mainly for computing the light distribution in canopies and it is possible to represent all parts of a plant, such as leaves or stems, objects around the plant, or the ground as a set of polygons. In addition, the leaves are assumed to be thin enough that the refraction of light transmitted through the leaves can be ignored.

A.2.1 Preprocessing

In the preprocessing phase of the algorithm, the scene polygons are stored in a regular grid and a bounding sphere is computed around all polygons. The use of a regular grid is a standard technique that accelerates ray-tracing by reducing the number of intersection tests between a ray and polygons in the scene [12, 34]. The bounding sphere is used to determine the boundaries for rays entering the scene (see Section A.2.2).

The following heuristic computes the sphere incrementally, as vertices $V_{i,j}$ of incoming polygons, indexed $j = 1, \ldots$, are processed. The first value of the sphere center C_1 is set to the first vertex $V_{1,1}$ of the first polygon and radius r_1 is set to 0. If the $(k+1)$-st processed vertex $V_{i,j}$ is outside the sphere (C_k, r_k), the center C_k is moved towards $V_{i,j}$ by half the difference between the distance $|C_k - V_{i,j}|$ and radius r_k:

$$C_{k+1} = C_k + \frac{V_{i,j} - C_k}{|C_k - V_{i,j}|} \cdot \frac{|C_k - V_{i,j}| - r_k}{2}.$$

The radius r_k is then increased by the same difference:

$$r_{k+1} = r_k + \frac{|C_k - V_{i,j}| - r_k}{2}.$$

Thus the new sphere includes all points in the sphere from the previous iteration plus the vertex $V_{i,j}$.

It is important to have a tight bounding sphere, because the number of rays traced depends on the size of the sphere (Section A.2.2). This algorithm does not compute the smallest bounding sphere, but in the case of a set of leaves distributed in a more or less spherical crown, the computed sphere is very close to the minimal one. Since the algorithm is not limited to computing the distribution of light in a tree canopy, the resulting bounding sphere is compared with the bounding sphere of the bounding box enclosing all objects and the sphere with the smaller radius is chosen.

A.2.2 Generating initial rays

The computation of the light distribution starts with generating initial rays representing photon bundles of a certain radiant power Φ emitted from the light sources [36]. The rays are chosen randomly so they sample the specified light sources. For each initial ray, it is necessary to choose its direction and point of departure.

In the case of plant models, the light source is usually the sky. The sky can be approximated by defining a set of directional light sources or represented by a continuous function, specifying the radiant power coming from an arbitrary direction (*e.g.* a CIE standard overcast sky luminance function [6]). In both cases, the direction of initial rays is generated randomly so that the ray distribution reflects different intensities of light coming from different directions, *i.e.*more rays are coming from the direction of bright light sources and the initial radiant power of all rays is 1.

If a list of m directions v_i with intensities I_i, $i = 1, ..m$, is specified, the direction of an initial ray is chosen randomly among directions v_i with probability $I_i/(\sum_{j=1}^{m} I_j)$ for direction v_i. The number N of initial rays depends on the size of the bounding sphere enclosing all objects in the scene. If r is the radius of the bounding sphere, d is the ray density (specified by the user and defining the number of rays of unit radiant power generated per unit area), and I_i are intensities of light coming from m specified directions, the total number of initial rays with radiant power 1 is:

$$N = \pi r^2 d \sum_{j=0}^{m} I_j. \tag{4}$$

If the sky is represented by the continuous luminance function $L(\theta, \varphi)$ [6], the direction of an initial ray is specified as (θ, φ), where θ is the angle with respect to zenith and φ is the angle with respect to north (the zenith is equal to y axis, and north is in the direction of the positive z axis). The direction is generated according to a probability density function based on the luminance function L so that more rays are generated in directions with higher intensity [38].

To determine the point of departure of a ray with the given direction (both in the case of a continuous function or m directions), a random point P on a disk with center C and radius r (of the bounding sphere) perpendicular to the ray direction is selected. The point P (moved along the ray direction so that the ray starts outside the scene) with the generated direction then specifies the initial ray.

A.2.3 Tracing of rays

Each initial ray with the radiant power 1 is traced in the grid using standard ray-tracing techniques [12]. If the intersection with an object is detected, the local light model described below is applied. First, it is decided whether the ray is absorbed, reflected, or transmitted. Then, if the ray is transmitted or reflected, the direction of the new ray is determined according to the directional distribution of reflected or transmitted light.

If a ray with radiant power Φ hits a surface at point P, a portion of the power is absorbed, a portion is reflected, and a portion is transmitted. If the value of the surface

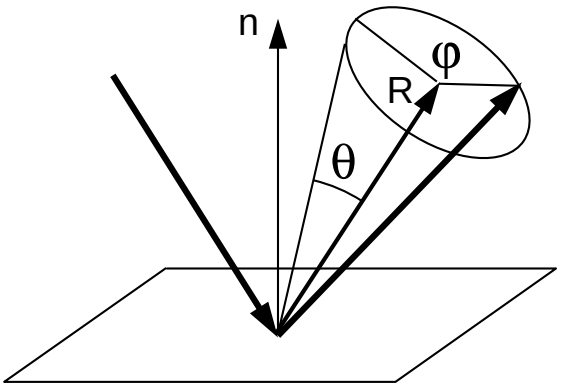

Figure 1: Definition of angles θ and φ with respect to the "ideal" reflected ray $\vec{R}$

absorbance is equal to a, the reflectance is equal to r, and the transmittance is equal to t ($a + r + t = 1$), the radiant power absorbed by the surface is equal to Φa, reflected Φt, and transmitted Φt. Generally, the values of a, r, and t depend on the wavelength of the incoming photons and the incidence angle (the angle of the ray with the surface tangent at the intersection point).

Usually, only one possibility is chosen, and the incoming ray is either absorbed, reflected, or transmitted, based on probabilities a, r, and t [38]. In the program *MonteCarlo*, a part of the ray's radiant power Φ is always absorbed by the surface (the absorbed power Φa is added to the value of a parameter associated with the surface). The remaining power $\Phi(r + t)$ is either reflected or transmitted. In this case, the reflected or transmitted ray is chosen randomly with probability $r/(r + t)$ and $t/(r + t)$, respectively.

The direction of the new ray is chosen as follows. Generally, the radiant power reaching the surface at a certain point is not reflected equally in all directions. The directional distribution of the reflected light is expressed by *bidirectional reflectance distribution functions* (BRDFs) [14, 38]. Similarly, a *BTDF* describes the directional distribution of transmitted light. There are different functions for different incidence angles.

The program *MonteCarlo* approximates the BRDFs by the Blinn-Phong reflectance model [35] and assumes that the values of a, r, and t do not depend on the incidence angle. If the scene contains many plant organs and the ray represents only a single wavelength (not an average radiant power of all wavelengths), it is a common practice to assume constant values of these parameters [28].

In the Blinn-Phong model, the direction of the reflected ray is expressed by angles θ and φ, where θ (a zenith angle) is an angle between the ray and the "ideal" reflected ray $\vec{R}$ and angle φ is an azimuth angle with respect to ray $\vec{R}$ (Figure 1). If the direction of all reflected rays is chosen randomly using a uniform distribution, the radiant power

of a reflected ray for all incidence angles is:

$$\Phi(\theta, \varphi) = \Phi_r \frac{n+2}{8\pi} \cos^n \frac{\theta}{2}, \tag{5}$$

where Φ_r is the total reflected radiant power (in our case, $(r+t)\Phi$), n is the reflectance scattering exponent, and angles θ and φ range from 0 to π[3] and from 0 to 2π, respectively [35]. High values of n represent smooth surfaces for which most of the light is reflected close to the ray $\vec{R}$. The Blinn-Phong model is used instead of the simple Phong model, because in the Phong model, the angle θ ranges only from 0 to $\pi/2$, which does not cover all possible ray directions in case the "ideal" (reference) reflected ray $\vec{R}$ is not equal to the surface normal $\vec{n}$ [35].

Following Hall and Greenberg [19], the same formula (5) is used to determine the radiant power of rays transmitted in a given direction around the reference transmitted ray (equivalent to ray $\vec{R}$ in case of reflection). In our case, the polygons represent either the boundary of opaque objects or thin surfaces, such as leaves, in which case it is possible to ignore refraction and the reference transmitted ray is equal to the original ray.

If the direction of a reflected or transmitted ray was chosen at random with the uniform distribution, there would be a lot of rays with a low value of radiant power not contributing much to the light absorbed by an object a ray may hit. That is why the direction of the reflected or transmitted ray is generated randomly according to the radiant power distribution function (equation (5)) and the ray radiant power stays equal to $\Phi_r = \Phi(r+t)$. In case of the Blinn-Phong model, the ray direction is computed using formula [35]:

$$(\theta, \varphi) = (2 \arccos((1-r_1)^{\frac{1}{n+2}}), 2\pi r_2),$$

where r_1 and r_2 are uniformly distributed random numbers in the interval $\langle 0, 1)$.

Afterwards, the reflected or transmitted ray is generated and the ray is traced until another hit occurs.

A.2.4 Terminating the ray

There are two mechanisms that can be applied to determine the termination of a ray. In the first approach, tracing of a ray continues until the ray depth (equal to the number of hits on the ray's path) reaches the maximum user-specified value. The optional method for the terminating a ray is called *Russian roulette* [14, 38]. In this method, each ray has associated with it a probability parameter. This parameter specifies the probability with which the particular path was chosen. Initially this probability is one. After each hit, it is multiplied by parameters specifying the probability of generating reflected or transmitted ray ($r/(r+t)$ or $t/(r+t)$). Before generating a new ray, the ray's radiant power Φ multiplied by the probability parameter $prob$ is compared with a threshold value Φ_t. If $\Phi \cdot prob \geq \Phi_t$ the ray continues. Otherwise a random

[3] Rays going into the surface are ignored.

number between 0 and 1 is generated and if it is below a given probability threshold p, the ray is terminated. If the ray continues, its radiant power Φ is divided by $(1-p)$. This amounts to an increase of the radiant power to account for rays that have been terminated [2, 32]. In our simulations, we use values of $\Phi_t = 0.1$ and $p = 0.7$.

A.2.5 Interfacing with the plant simulator

Generally, the environmental program receives all polygons representing the plant's leaves and possibly objects around the plant. Then the program computes the radiant powers reaching each surface. After all rays have been traced, parameters storing the radiant power absorbed by an object are sent to the plant model. If the object consists of more than one polygon the contribution of all polygons is summed.

Input of the plant geometry

In each simulation step, the environmental process receives information about all communication modules and the selected plant modules. The plant modules following communication modules represent plant organs.

In the program *MonteCarlo*, the geometric information about a plant organ is transferred in two ways. The difference in the two approaches is in the way the module following the communication module $?E$ is represented.

1. For a module P, the environmental process constructs a parallelogram of a specified orientation (based on turtle vectors). The lengths of diagonals of the parallelogram are controlled by the parameters of the module P.

2. For any other module, the environment expects that the plant simulator sends a set of polygons representing the module.

Specifically, if the communication module $?E$ is followed by module $P(a, b, ...)$ with two or more parameters, the module P defines a polygon constructed in the following way. The polygon is a parallelogram with one diagonal in the direction of the turtle heading vector $\vec{H}$ and length $2a$, and the second diagonal in the direction of the left vector $\vec{L}$ and length b. The turtle position $\vec{P}$ defines one vertex of the polygon. Thus, the message sent to the environment has to also include the turtle position $\vec{P}$, the heading vector $\vec{H}$, and the left vector $\vec{L}$. The third and fourth parameter of the module P, if present, specify the index of the material for the front and back sides of the polygon. The material is defined in the specification file of the environmental program (see Section A.2.5).

The definition of the polygon shape is fixed and can be changed only by varying values of a and b. Thus in the second approach, the environment can receive a set of polygons representing the module following $?E$. Thus for any module other than P, the environment expects the geometry in the form of polygons sent by the plant model. To define a complex organ, homomorphism productions can specify the geometry of the module, and all the polygons representing the module are transferred to the environment, which considers them as part of a single object. The material indexes are then specified by the first two parameters of the module following the module $?E$.

Materials

Parameters of surface materials are defined in a specification file processed by the environment at the beginning of the simulation. Each material is defined by 4 parameters:

1. **reflectance** - controls the percentage of the incoming radiant power that is reflected from the surface. A value between 0 (a black body) to 1 (a perfect reflector) can be used. The value does not depend on the incidence angle of the light direction reaching the surface.

2. **reflectance scattering exponent** - specifies the roughness of a surface. Value of 0 represents a perfectly diffuse surface, for which the radiant power of the reflected ray does not depend on its direction — the higher the value the smoother the surface.

3. **transmittance** - controls the percentage of the radiant power being transmitted through the surface.

4. **transmittance scattering exponent** - similarly as the reflectance exponent, the transmittance scattering exponent specifies the degree of scattering of the transmitted light. For a value of 0, the light is scattered equally in all directions, while for higher values, more rays are concentrated around the "ideal" direction of the transmitted ray.

The sum of reflectance and transmittance must be in the interval $\langle 0, 1 \rangle$. Since the surfaces are thin, the index of refraction does not have to be specified.

As a default, surfaces use the first material defined in the specification file. This default material can be changed by including the index of a desired material (corresponding to the order of its definition in the specification file) in the module following the communication module. It is possible to specify different materials for the two sides of a surface (by including two material indexes in the module).

A.3 Computing the ratio of different wavelengths

In this section, the algorithm presented above is extended by making it possible to consider different surface parameters for different wavelengths and to efficiently compute the ratio of light fluxes for two selected wavelengths reaching a surface.

A.3.1 Background

In the case of the Monte Carlo algorithm, it is a common approach to specify a set of surface parameters for each considered wavelength and then to run the algorithm several times, separately for each wavelength [38].

If we were interested only in the flux of red light R and the flux (radiant power) of the far red light FR reaching a surface, this method would be sufficient, because it is possible to keep the variance of the results below a given threshold by tracing a certain

amount of initial rays (by modifying the ray density d). Specifically, it can be shown that to reduce the variance of the energy absorbed by the objects below a threshold V_0, it is sufficient to trace C/V_0 initial rays, where C is a constant [14, Section 18.8].

The problem in our case is that even if values of R and FR for a single object have a variance below V_0, their ratio could have a much higher variance. Consequently, many more initial rays are required to reduce the variance of the ratio.

The following sections present a modification of the Monte Carlo algorithm, which makes it possible to compute the ratio more efficiently than running the method separately for each wavelength.

A.3.2 Generating initial rays

To reduce the number of initial rays, and consequently the running time of the Monte Carlo algorithm, I have modified the algorithm by tracing a single ray for all wavelengths at once[4]. Each ray carries compound information about different wavelengths in the form of the radiant power for each wavelength. The local light model is then modified to be able to consider different probabilities of generating a reflected or transmitted ray and different scattering exponents for each wavelength in selecting a single reflected or transmitted ray.

If there are s wavelength samples considered, each ray has associated radiant powers Φ_i, $i = 1, .., s$, for each wavelength. Initially, the radiant powers Φ_i are set to the source intensities I_i for given wavelengths λ_i. The number of initial rays is then computed using the formula 4, with all intensities I_j set to 1, *i.e.*there are $\pi r^2 d$ initial rays for each light source.

A.3.3 Modifications of the local light model

Similarly to the original method, surfaces are characterized by four parameters specifying reflectance r_i, reflectance exponent n_{r_i}, transmittance t_i, and transmittance exponent n_{t_i} (for each wavelength separately). Each surface contains a set of s parameters A_i, $i = 1, .., s$, that store the amount of light absorbed by the surface for each wavelength.

When a ray hits a surface, each radiant power Φ_i is reduced to account for the absorbed light. Thus parameters A_i are increased by $\Phi_i(1 - r_i - t_i)$ (as in the original method, see Section A.2.3). The remaining radiant powers:

$$\Phi_{a_i} = \Phi_i(r_i + t_i)$$

are used as the radiant powers of a reflected or transmitted ray.

In the Monte Carlo method described in Section A.2, the selection between a reflected and transmitted ray was done based on the surface reflectance r and transmittance t. In the modified method, the values of these parameters may significantly vary for different wavelengths and it is necessary to balance the probabilities of choosing

[4] To the best of our knowledge this method has not been published before.

the reflected or transmitted ray for all wavelengths. Instead of simply averaging the values of $r_i/(r_i + t_i)$ and $t_i/(r_i + t_i)$, each value is multiplied by the actual ray radiant power Φ_{a_i} to increase the influence of wavelengths with higher radiant power.

The probability of generating a reflected or transmitted ray $\vec{R}$ is computed from the sums of probabilities for each wavelength. The probability of generating a reflected ray is:

$$p_r = \frac{\sum_{i=1}^{n} r_i/(r_i + t_i)\Phi_{a_i}}{\sum_{i=1}^{n} \Phi_{a_i}} = \frac{\sum_{i=1}^{n} r_i \Phi_i}{\sum_{i=1}^{n} \Phi_{a_i}}.$$

Similarly, the probability of generating a transmitted ray is:

$$p_t = \frac{\sum_{i=1}^{n} t_i/(r_i + t_i)\Phi_{a_i}}{\sum_{i=1}^{n} \Phi_{a_i}} = \frac{\sum_{i=1}^{n} t_i \Phi_i}{\sum_{i=1}^{n} \Phi_{a_i}}.$$

In the local model presented in Section A.2.3, the radiant power of a reflected or transmitted ray was equal to the ray radiant power before the hit (Φ), because the reflected or transmitted ray was chosen in such a way that on average, the reflected radiant power is equal to Φr and the transmitted radiant power is Φt. Specifically, out of N rays with the same power Φ reaching the surface, Nr rays are reflected and Nt are transmitted (if N is sufficiently large). Thus out of the total incoming radiant power $N\Phi$, the power $Nr\Phi$ is reflected and $Nt\Phi$ is transmitted.

Since the ray is now chosen based on the overall reflected or transmitted radiant power using values p_r and p_t, out of N incoming rays (with the same radiant powers Φ_i, $i = 1, 2, ..., s$), there are Np_r reflected and Np_t transmitted rays. To keep the total amount of reflected radiant powers equal to $N\Phi_i r_i$, the radiant powers associated with a reflected ray are multiplied by a factor r_i/p_r. Thus radiant powers Φ_{R_i} of the reflected ray are:

$$\Phi_{R_i} = \Phi_i \frac{r_i}{p_r}.$$

Similarly radiant powers Φ_{T_i} of the transmitted ray are:

$$\Phi_{T_i} = \Phi_i \frac{t_i}{p_t}.$$

For example, if a reflected ray is traced and a reflectance for some wavelength is 0, the corresponding radiant power is also 0.

Once the reflected or transmitted ray is chosen according to probabilities p_r and p_t, it is necessary to determine the direction of the ray. In Section A.2, the direction of both the reflected and transmitted ray is generated using the Phong-Blinn reflectance model. Since generally, the scattering exponents may be different for each wavelength, the program first choses one of the exponents using a similar approach as in the case of reflectances and transmittances, and then adjusts the radiant power Φ_i of each wavelength according to the difference between the exponent n_i and chosen exponent n.

The direction (θ, φ) of the new ray is generated randomly according to the distribution function (5). The exponent n is randomly chosen between exponents n_{r_i} for

reflected or n_{t_i} for transmitted ray with probabilities

$$p(n = n_{r_i}) = \frac{I_{R_i}}{\sum_{i=1}^{n} I_{R_i}},$$

and

$$p(n = n_{t_i}) = \frac{I_{T_i}}{\sum_{i=1}^{n} I_{T_i}}.$$

The direction of the new ray reflects the proper distribution of radiant powers only for exponents equal to n. It is thus necessary to adjust the ray radiant powers for all wavelengths for which n_{r_i} or n_{t_i} is not equal to the selected n.

Imagine that s different rays (for each wavelength) would be generated. The ray direction is generated according to the radiant power distribution function p:

$$p(n_i, \theta, \varphi) = \frac{n_i + 2}{8\pi} \cos^{n_i} \frac{\theta}{2}. \tag{6}$$

If Φ_i is the total radiant power reflected from or transmitted through the surface (Φ_{R_i} or Φ_{T_i}) the radiant power leaving the surface in the direction (θ, φ) is equal to $\Phi_i p(n_i, \theta, \varphi)$, where n_i is equal to either n_{r_i} or n_{t_i}.

Thus if the direction is generated using the radiant power distribution function $p(n, \theta, \varphi)$ instead of $p(n_i, \theta, \varphi)$, the radiant power Φ_i associated with the ray is multiplied by a factor

$$\frac{p(n_i, \theta, \varphi)}{p(n, \theta, \varphi)} = \frac{n_i + 2}{n + 2} \cos^{n_i - n} \frac{\theta}{2}.$$

Consequently, all intensities of the reflected or transmitted ray appropriately reflect the corresponding properties of the surface.

A.3.4 Tracing rays

The rays are traced the same way as in the original method (Section A.2), except that the Russian roulette method of terminating rays is modified to account for several wavelengths. After each hit, the radiant powers multiplied by the probabilities of chosing a given path (for each wavelength) are added and divided by the number of wavelengths. The resulting value is compared with the threshold Φ_t as in the original Russian roulette method. The ray is terminated with probability p or all its radiant powers are increased by factor $1/(1 - p)$.

As in the original method, after all initial rays have been traced, parameters accumulating the light radiant power absorbed by an object are sent to the plant model.

A.3.5 Comparison with the standard method

To compare the method described in this section with the commonly used approach [38] in which the rays are traced separately for each wavelength, the following experiment has been designed (for a similar approach, see [3]).

An L-system model was used to generate a set of N triangles, which are placed and oriented at random in a cube of size $10 \times 10 \times 10$ units. The triangles are then transferred to the environment, which performs a given number of simulations S. Thus instead of computing the values of the R/FR ratio just once, the program repeats the process S times and outputs a mean value of the ratio and its standard deviation for each triangle. To be able to compare the results for the two algorithms, the resulting standard deviation for every triangle is divided by the corresponding mean to obtain a relative standard deviation for the triangle (to account for differences in the mean values for different triangles). The relative deviations of all triangles are averaged resulting in an average relative standard deviation of the experiment $rel\ \bar{\sigma}_{FR/R}$. In addition, values of $rel\ \bar{\sigma}_R$ and $rel\ \bar{\sigma}_{FR}$, corresponding to the average relative standard deviations for fluxes of red light or far red light reaching the triangles, are also computed. Table 1 summarizes the results of the simulations.

Simu-lation	Triangles, Area	Rays separate/together	$rel\ \bar{\sigma}_R$,$\bar{\sigma}_{FR}$; $\bar{\sigma}_{R/FR}$ separate/together	Time[a] (min) separate/together
1	100, 0.25	21,500	0.18, 0.21; **0.31**	3:58
		10,000	0.21, 0.20; **0.12**	1:49
2	100, 0.25	107,500	0.061, 0.080; **0.10**	19:35
		50,000	0.071, 0.078; **0.050**	8:37
3	150, 1.0	21,500	0.12,0.11; **0.17**	6:12
		10,000	0.14, 0.12; **0.14**	3:03
4	150, 1.0	107,500	0.042, 0.49; **0.066**	32:58
		50,000	0.051, 0.053; **0.059**	14:54
5	200, 2.0	21,500	0.28, 0.18; **0.37**	11:45
		10,000	0.38, 0.20; **0.38**	6:44
6	200, 2.0	107,500	0.12, 0.080; **0.15**	60:24
		50,000	0.16, 0.088; **0.16**	33:03

[a] 100 consecutive simulations on a 150MHz/32MB R5000.

Table 1: Comparing the precision of results and simulation times of the standard Monte Carlo method with the new approach of tracing one ray for all wavelengths

In the first simulation, 100 triangles of the area of 0.25 square units are considered. The four surface parameters (r, n_r, t, n_t) were equal to 0.1, 0, 0.0, and 10 for red light and 0.35, 0, 0.55, and 10 for far red light. The ray density was set to 100 rays per square unit of the light source. Consequently, 21,500 rays (10,000 for red and 11,500 for far red light) were traced using the standard approach and only 10,000 rays using the new method. The simulations were run 100 times to obtain the variances of the fluxes reaching each triangle. The corresponding running times were 3:58 minutes and 1:49 minutes for the original and new method, respectively. Although the running time

of the new method was less than half of the time for the standard method, the average standard deviation of the new method (0.91) was almost three times lower that the one of the standard method (2.83). Interestingly, the average relative standard deviations of red and far red fluxes reaching the triangles are comparable for both methods (only slightly higher for the red light), suggesting that even if less rays are traced using the new method, the results for separate wavelengths are basically the same as for the original method.

In the second simulation, the number of rays has been increased 5 times. The corresponding results are in Table 1. The average variances are about three times lower as compared to the simulation with less rays. The new method is about twice as fast and twice as precise in computing the R/FR ratio than the original method.

The significant reduction of the average variance of the ratio may be caused by the fact that triangles are small and they do not overlap each other too much. In that case, the direct light reaching each triangle constitutes the main influence and the results are biased, because the R/FR ratio of rays before the first hit is exact. If we want to obtain a better comparison of the methods, we have to make the set of triangles denser.

Thus in the following two simulations, the previous two experiments were repeated, with the number of triangles increased to 150 and their area set to 1. The results, shown in Table 1, support the observation about the effect of the direct light, because the difference in $rel\ \bar{\sigma}_{R/FR}$ (for the two methods) is reduced. Nevertheless, the value $rel\ \bar{\sigma}_{R/FR}$ for the new method is still below the one produced by the original method and the values of deviations for red and far red light are again comparable. Note that the running time is again half the time of the original method.

In the last two simulations, the number of triangles was further increased to 200, and their area to 2. As illustrated in Table 1, the difference in $rel\ \bar{\sigma}_{R/FR}$ is further reduced and the values of $rel\ \bar{\sigma}_R$ and $rel\ \bar{\sigma}_{FR}$ for the new method are higher that the ones for the old method, especially for red light where the difference is about 35%. These results are caused by the fact that the size and the number of triangles is very high considering the size of the cube in which they are placed ($10 \times 10 \times 10$) and most triangles receive only reflected or transmitted light. In a real plant canopy, though, the density of leaves would be lower (in the order of the density in the second set of simulations, using 150 triangles).

As illustrated by the above experiments, the new algorithm performs much better as compared to the common approach of tracing separate sets of rays for each wavelength. The new method seems to be slightly more precise than the original method while reducing the running time by half. In addition, the resulting values of red and far red fluxes reaching the objects were comparable for both methods. Consequently, the new method is very useful for computing the quality of light reaching the plant and can be used effectively in the simulation of plants interacting with the light environment.