Extensions to the graphical interpretation of L-systems based on turtle geometry

R. Měch, P. Prusinkiewicz,
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada T2N 1N4
e-mail: mech|pwp@cpsc.ucalgary.ca

and J. Hanan
CSIRO - Cooperative Research Centre
for Tropical Pest Management
Brisbane, Australia
e-mail: jim@ctpm.uq.oz.au

Abstract

This material introduces extensions to the graphical interpretation of L-systems based on turtle geometry, resulting in a higher degree of realism of visualized models.

Contents

1 Introduction

The graphical interpretation of L-systems introduced in [11, 12] and extended to parametric L-systems in [9, 16] has been extended by incorporating several features aimed at improving the realism of generated plant structures. This report is intended as a user's guide to these features.

The graphical interpretation is based on a Logo-style turtle which is controlled by commands associated with selected modules. The turtle is characterized by the following parameters (see Figure 1):

position position in a three-dimensional coordinate system;

heading heading vector specifying turtle's orientation (see Figure 1);

left left vector;

up up vector;

line width current width of a line or cylinder drawn by the turtle;

color index index of the current color or material;

texture index index of the current texture;

contour id of the current contour for generalized cylinders;

elasticities a set of parameters specifying the susceptibility of the direction adjustments due to tropisms.

Initial values of the turtle parameters are defined by global viewing parameters. During the interpretation, turtle parameters can be modified by various modules (see Appendix C).

Following sections introduce new features in graphical interpretation of L-systems and illustrate their use with several examples.

2 Circles and Spheres

Circles and spheres are useful primitives, and have been added to the list of interpreted primitives.

Circles and spheres are defined by following modules:

@o draw a disk in plane $z = 0$. Turtle position determines x and y coordinates of the disk center. The disk diameter is defined by turtle parameter *line width*.

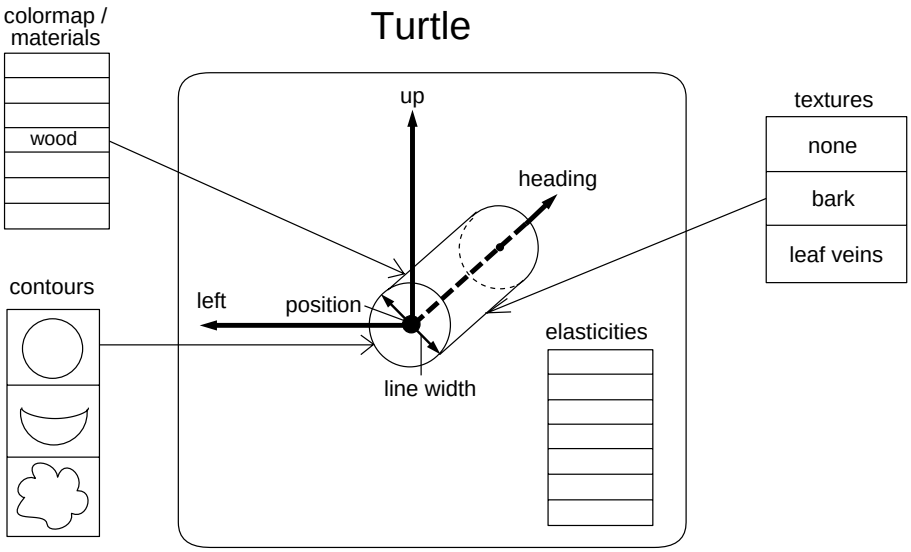


Figure 1: Turtle parameters.

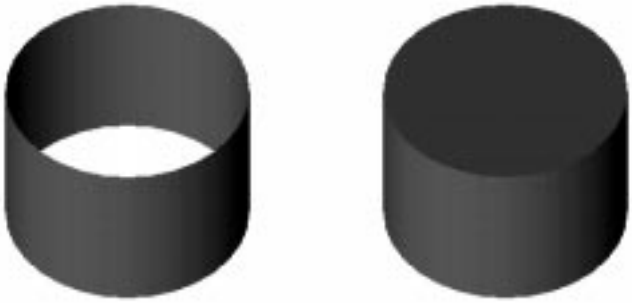


Figure 2: A cylinder without and with caps.

`@c` draw a disk in the plane of turtle heading and left vector, with the center at the turtle position and diameter equal to turtle parameter *line width*.

`@O` defines a sphere at the turtle position with diameter equal to *line width*.

If a parameter is added to any of the three modules, it specifies the diameter of the disk or sphere.

Module `@o` can be used any time the simulated plant is two-dimensional, for example, to visualize a signal traveling through the plant. In the case of a three-dimensional structure, a module `@O` would be used for the same purpose. For example, thefollowing homomorphism makes use of module `@c` to terminate cylinders with circular caps (see Figure 2).

`F(len)` $\rightarrow$`[&(90)@c]F(len)[&(90)@c]`

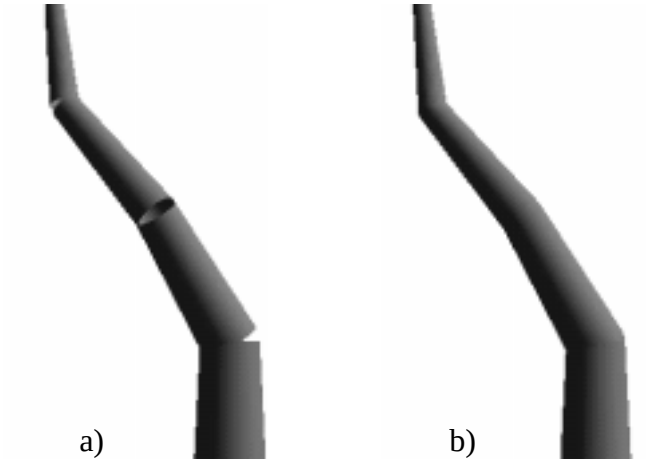


Figure 3: A branch rendered: a) with visible discontinuities, b) with sphere at the end of each segment.

Each symbol `F` is preceded and followed by module `@c` rotated by 90° in square brackets to localize the effect of the rotation.

The next few examples illustrate the use of spheres.

When two subsequent branch segments are not in the same line, a visible crack appears (Figure 3a). A simple solution is to draw a sphere at the top of each segment. If segments are not tapered, i.e. the widths at the base and the top of the segment are the same, a sphere can be inserted after each module `F` in the successor of all L-system productions, or the following homomorphism production can be used:

`F(len) →F(len)@O`

If segments are tapered, the sphere must be inserted after the module `!` or # modifying the width at the top, otherwise segments would not be tapered and the diameter of the sphere would be equal to the width of the segment at its base. The branch in Figure 3a was generated by the following L-system in 3 steps:

ω: `F!A`

p_1: `A →/(90)-(32)F!A`

The smoother branch in Figure 3b was generated by the same L-system with the following homomorphism

`homomorphism`

`F! →F!@O`

A better solution to creating curved branchess is presented in Section 3.

An example of spheres used as an important part of the model is illustrated in Figure 4 where a molecule of butane is rendered using spheres as atoms of carbon and hydrogen

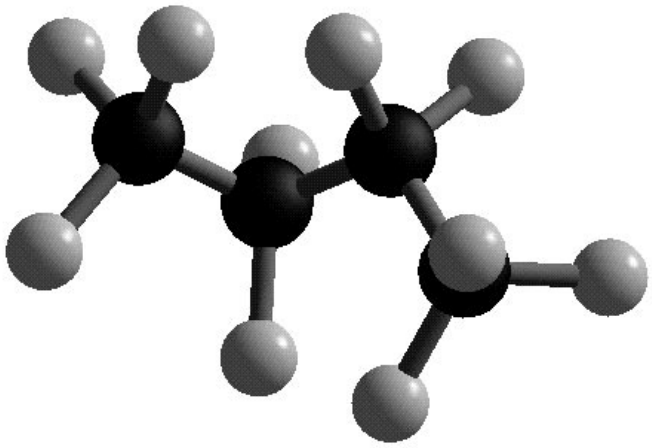


Figure 4: Molecule of butane generated using an L-system.

and segments represent their chemical bounds. The molecule was created by the following L-system which allows one to visualize hydrocarbon molecules of various lengths (i.e. with different number of carbon atoms):

```
#define Len 4      /* length of the hydrocarbon chain */
#define θ 70.5     /* divergence angle */
#define Csize 0.6  /* diameter of carbon atom */
#define Hsize 0.5  /* diameter of hydrogen atom */
ω:  -(90)#(0.15)|HX(Len)

p1:  X(n) :  n > 0 →FC[^(θ)FH]/(120)[^(θ)FH]/(120)[^(θ)X(n-1)]
p2:  X(l) :  l == 0 →FH
p3:  C →[;@O(Csize)]
p4:  H →[;;@O(Hsize)]
```

The simulation is run for $Len + 2$ steps to obtain a molecule with Len carbon atoms.

3 Generalized cylinders

It is often desirable to create smooth curvatures, especially when modeling plants which contain many curved segments. One approach is to model a curved branch segment with several short straight segments approximating the curvature. The advantage of this method is the possibility of the internal control of the curvature by L-system productions. On the other hand, this approach makes L-system productions rather complicated and elongates the generated string.

In another approach, curved branches are modeled as generalized cylinders. A set of

control points defines the axis of a generalized cylinder as a parametric curve consisting of a sequence of cubic curve segments. The cross-section of a branch segment, for example a disc, is then swept along the cylinder axis creating a three-dimensional object (see also [2]). In our case, segments of the parametric curve are defined as Hermite curves [7]. A Hermite curve is a cubic polynomial curve specified by two control points and tangent vectors in these points.

3.1 L-system defined parametric curve

During the interpretation of an L-system generated string, control points specifying the axis of a generalized cylinder are created when following special modules are encountered:

`@Gs` start a generalized cylinder — defines the first control point.

`@Gc(n)` continue a generalized cylinder — defines a control point. The cylinder has to be started with module `@Gs`. The optional parameter `n` specifies the number of cylindrical mesh strips drawn between this and previous control point.

`@Ge(n)` end a generalized cylinder — defines the last control point. The optional parameter `n` has the same function as for module `@Gc`.

Each pair of consequent points specifies a Hermite curve.

The location of a control point is equal to the actual turtle position when the module `@Gx` is interpreted. The direction of tangents of Hermite curves originating or terminating at the control point is equal to the turtle heading vector. The length of the two tangent vectors of a single Hermite curve is computed as the Euclidean distance between the two control points multiplied by a tangent coefficient. This allows the curve to be scaled up and down without changing the curvature. The tangent coefficient defaults to 1.2 (an empirical value yielding a smooth curvature along several connected curves).

For example, the following string defines a generalized cylinder around a single Hermite curve:

```
@Gsf-(45)f@Ge(6).
```

Figure 5a shows the turtle path in grey and the two control points with the resulting curve in black. If a disk is swept along the curve, a generalized cylinder is created (Figure 5b and 5c).

The resulting generalized cylinder has to be polygonized to be visualized on the screen. For this purpose, it is split into several cylindrical mesh strips. The first parameter of modules `@Gc` and `@Ge` specifies how many mesh strips are created between two control points (the default value is 1). A cylindrical mesh strip is defined by two points on the generalized cylinder axis. These points can be determined from the parametric equation of the Hermite

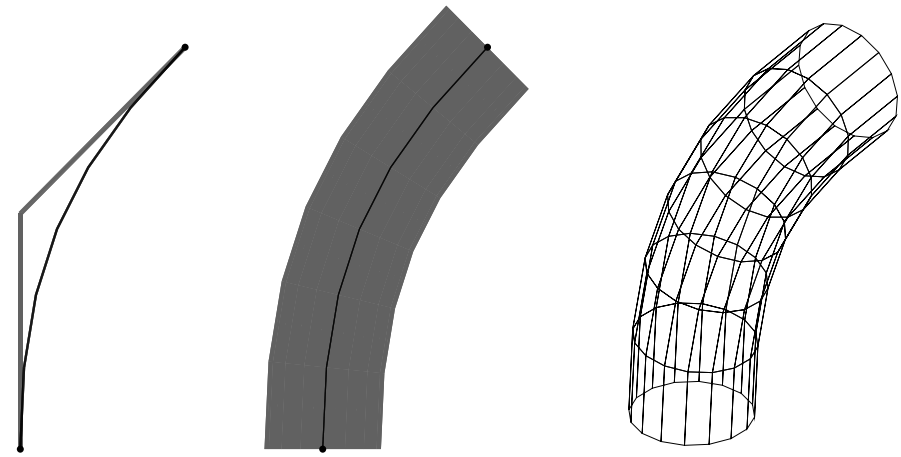


Figure 5: A simple generalized cylinder: a) the original Hermite curve, b) the longitudinal section, c) a wireframe model.

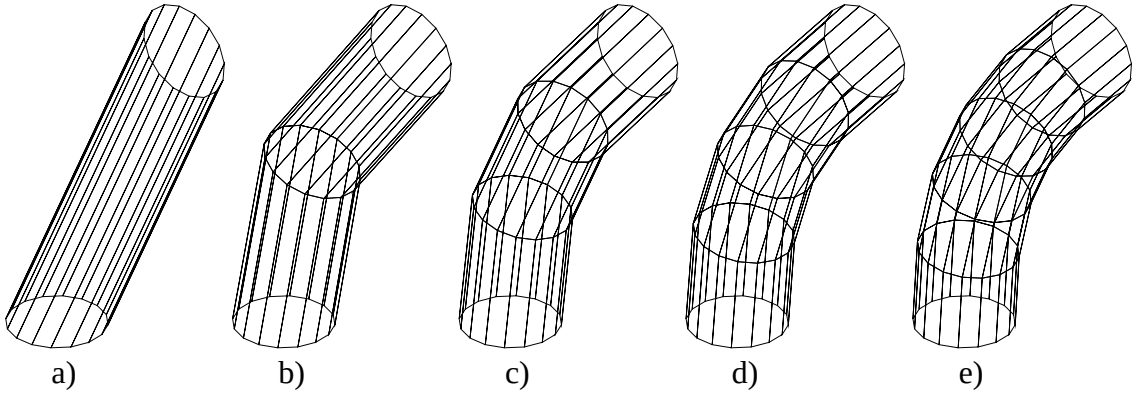


Figure 6: The same generalized cylinder visualized with 1 to 5 cylindrical mesh strips.

curve segment. If the parametric equation of the curve segment is $F(u)$ for u going from 0 to 1, the i-th strip out of n is between points $F((i-1)/n)$ and $F(i/n)$. The disk swept along the axis is always perpendicular to the axis and the axis tangent in a point $F((i-1)/n)$ or $F(i/n)$ can be determined from the derivation $F'(u)$ of the curve cubic function. Sections 3.2.3 and 3.3 describe in more detail the method of connecting two subsequent disks.

In the previous example, there are 6 cylindrical mesh strips along the generalized cylinder. The same generalized cylinder with different number of mesh strips (1 to 5) is visualized in Figure 6.

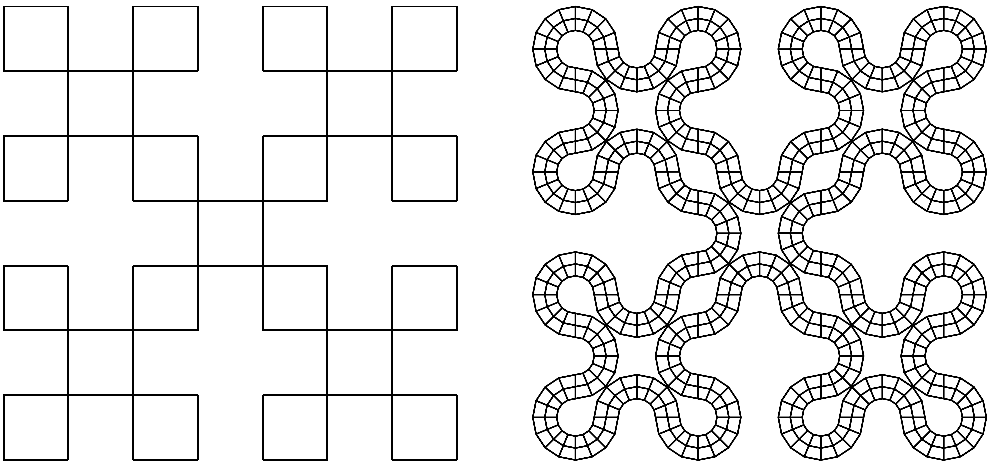


Figure 7: A fractal curve drawn using lines (a) and generalized cylinders (b).

As a more complex example, the following L-system defines a fractal curve known as the snake Kolam pattern [14] (Figure 7a); it is similar to Sierpinski space-filling curve [17]:

ω: `FX+F+FX+F`
p_1: `X`$\rightarrow$`X-F-F+FX+F+FX-F-F+FX`

To visualize the curve with a generalized cylinder, it is convenient to define a homomorphism that replaces each module `F` representing a straight line segment with a substring `f@Gc(4)f` defining one control point in the middle of an invisible line segment `ff`. The axiom is modified to differentiate between the first segment and any other, because the generalized cylinder must be started using module `@Gs`. Also an additional modules `+F` are added at the end of the axiom to define another control point which would coincide with the first one closing the generalized cylinder:

ω: `G X+F+FX+F +F`
p_1: `X`$\rightarrow$`X-F-F+FX+F+FX-F-F+FX`
`homomorphism`
`G`$\rightarrow$`@Gsf`
`F`$\rightarrow$`f@Gc(4)f`

The resulting curve after 2 derivation steps is shown in Figure 7b.

3.2 Modifying the shape of generalized cylinders

The shape of a generalized cylinder segment between two consecutive control points can be modified by modules:

`@Gt(start,end)` controls the length of tangents of a Hermite curve between two consecutive control points by modifying the default value 1.2 of tangent coefficients (see

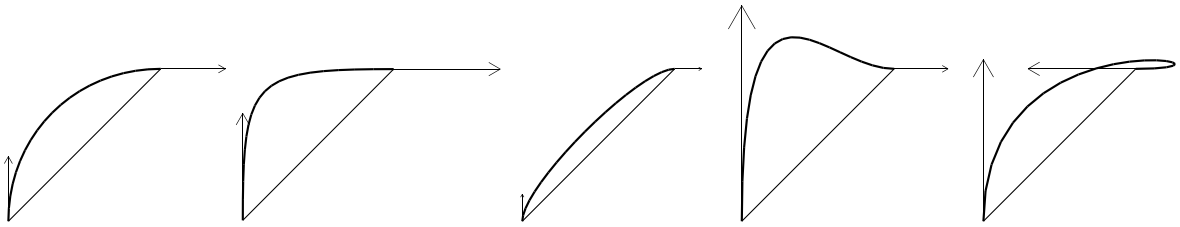


Figure 8: Hermite curves with different values of tangent coefficients: a) default 1.2,1.2; b) 2,2; c) 0.5,0.5; d) 4,1; e)3,-2.

Section 3.2.1).

`@Gr(angle1,length1,angle1,length2)` specifies the slope and length of two tangents of a Hermite curve defining the radius change as a longitudinal section between two consecutive control points of a generalized cylinder axis (see Section 3.2.2). As a default, the radii at the two control points are linearly interpolated along the segment.

`@Gr(flag)` switches on (`flag`=1) or off (`flag`=0) an automatic adjustment of tangents of a longitudinal section for segments of non-unit length. The longitudinal section is always defined for a segment of a unit length and then stretched onto the segment of a non-unit length. As a default, tangents are not adjusted after the stretching (see Section 3.2.2 for more details).

`@#(id)` replaces the default disk cross-section with a user-defined cross-section with an index `id` (see Section 3.2.2 and Appendix B.1).

3.2.1 Specifying tangents of the axis curve

Module `@Gt(start,end)` modifies the tangent coefficients for tangents at the start point and at the end point of a Hermite curve specifying the generalized cylinder axis. The module `@Gt` has to be inserted before the module defining the second control point of the Hermite curve segment. Figure 8 illustrates the use of the module `@Gt`. String:

```
@Gs-(45)f-(45)@Gt(start,end)@Ge(20)
```

is interpreted with different values of tangent coefficients `start` and `end`. Visualized tangent vectors are scaled by 1/4 to fit in the figure.

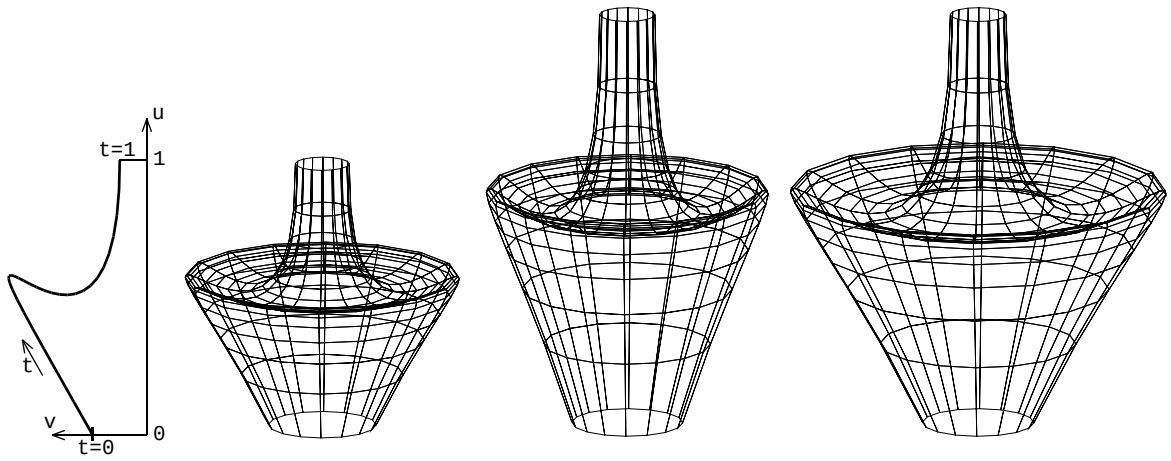


Figure 9: Radius interpolation: desired Hermite curve (a) applied to a generalized cylinder of length 1.0 (b) and 1.5 (c,d) without (c) and with (d) adjustment of tangents of the longitudinal section.

3.2.2 Modifying the longitudinal section

If the radii of the line cross-section associated with consecutive control points are different from each other, the radius of cross-sections between the two control points is interpolated. As a default, the radius is interpolated linearly. If the radius at the first control point is r_b and the radius at the second control point is r_t, the radii r_i and r_{i+1} of the i-th mesh strip out of n are:

$$\begin{aligned} r_i &= r_b + \tfrac{i-1}{n}(r_t - r_b) \\ r_{i+1} &= r_b + \tfrac{i}{n}(r_t - r_b) \qquad i = 1, 2, ..., n. \end{aligned}$$

It is also possible to define a two-dimensional Hermite curve $R(t) = (R_u(t), R_v(t))$ (for $t \in \langle 0, 1 \rangle$) capturing the change of the "longitudinal section" of a straight generalized cylinder with length 1 (Figure 9a and 9b). The first coordinate, $R_u(t)$, specifies the position along the axis of the generalized cylinder (Figure 9a). The coordinate $R_v(t)$ defines the radius at the computed point $R_u(t)$ on the cylinder axis.

The curve $R(t)$ is defined by the radius r_b at the bottom and radius r_t at the top of a generalized cylinder segment of a unit length, specifying two control points $(0, r_b)$ and $(1, r_t)$, and two tangents at these control points. In the case of the default linear interpolation, mentioned above, both tangents are $(1, r_t - r_b)$.

Tangents of the curve $R(t)$ can be modified by a module `@Gr` with two or four parameters. The first two parameters specify the angle of the first tangent with the u axis and

the tangent's length. Similarly, the third and fourth parameter define the second tangent. If only two parameters are included with module `@Gr` the second tangent is equal to the first one. If the length of a tangent is 0, the default tangent $(1, r_t - r_b)$ is used. As in the case of the module `@Gt`, the module `@Gr` has to be placed before the module specifying the second control point of a single Hermite curve segment.

Figure 9b illustrates the use of the module `@Gt` in a generalized cylinder of length 1, base radius 0.2, and top radius 0.1:

```
!(0.4)@Gsf(1)!(0.2)@Gr(30,4.5,0,3.5)@Ge(20).
```

To specify tangents of the longitudinal section, a module `@Gt` with parameters $30°$, 4.5, $0°$, and 3.5 was inserted before the module `@Ge` which defines the second control point.

If the line segment has a non-unit length len the longitudinal section is defined for a segment of a unit length, using values $R_u(t)$ going from 0 to 1, as in Figure 9a and then it is stretched along the axis of the segment. If the axis is represented as a line $F(u) = \vec{P} + u \cdot len \cdot \vec{H}$, $u \in \langle 0, 1 \rangle$, where $\vec{P}$ is a point at the bottom of the segment and $\vec{H}$ is the unit direction of the segment, then the i-th mesh strip is between points $F(R_u(\frac{i-1}{n}))$ and $F(R_u(\frac{i}{n}))$ with radii $R_v(\frac{i-1}{n})$ and $R_v(\frac{i}{n})$.

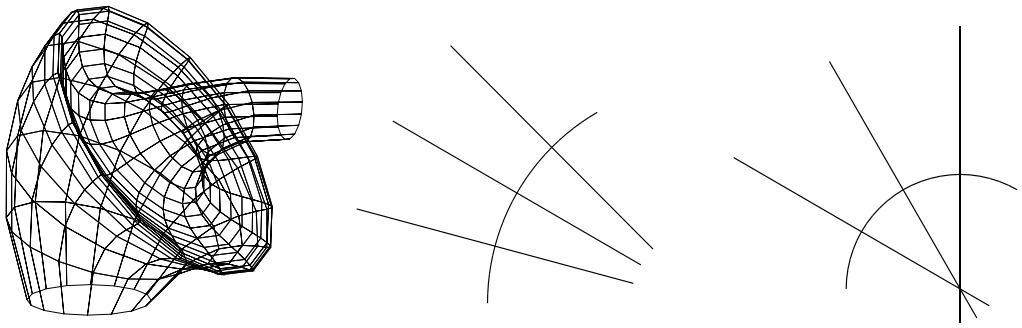
The generalized cylinder of length 1.5 in Figure 9c uses the same parameters for `@Gr` as in the previous example:

```
!(0.4)@Gsf(1.5)!(0.2)@Gr(30,4.5,0,3.5)@Ge(20).
```

Note that in this case the specified angle $30°$ of the first tangent of the radius curve $R(t)$ does not correspond to the real angle of the longitudinal section curve with the generalized cylinder axis. This is due to the scaling of the curve along just one axis (u). It is possible to adjust the tangents after the mapping of the curve $R(t)$ onto a segment of length len to keep the angle with the cylinder axis the same as specified by the module `@Gr`. The tangent adjustment, initially disabled, can be switched on and off by a module `@Gr(1)` and `@Gr(0)`, respectively. The cylinder in Figure 9d was generated using the same string as in the previous example, with the tangent adjustment switched on by a module `@Gr(1)`.

If the generalized cylinder segment is not straight, the curve $R(t)$ defines the longitudinal section of a straight segment of a unit length with radii at the base and at the top equal to radii of the curved segment. The longitudinal section is then mapped onto the axis of a curved genralized cylinder similarly as in the case of a straight segment of a non-unit length: if the axis is represented as a Hermite curve $F(u)$, $u \in \langle 0, 1 \rangle$, then the i-th mesh strip is between points $F(R_u(\frac{i-1}{n}))$ and $F(R_u(\frac{i}{n}))$ with radii $R_v(\frac{i-1}{n})$ and $R_v(\frac{i}{n})$. In the case of a curved segment, it is not easy to determine the length of the generalized cylinder axis. The adjustment of tangents of the longitudinal section, when required, is thus made using the Euclidean distance between control points of the cylinder axis.

An example of a curved generalized cylinder with the same longitudinal section as in the previous example is in Figure 10a. The segment is specified by the following string:



a) b) c)

Figure 10: Curved generalized cylinders: a) with radius interpolation; b) sweeping of a cross-section along the axis; c) crossed cross-sections when curvature radius is high.

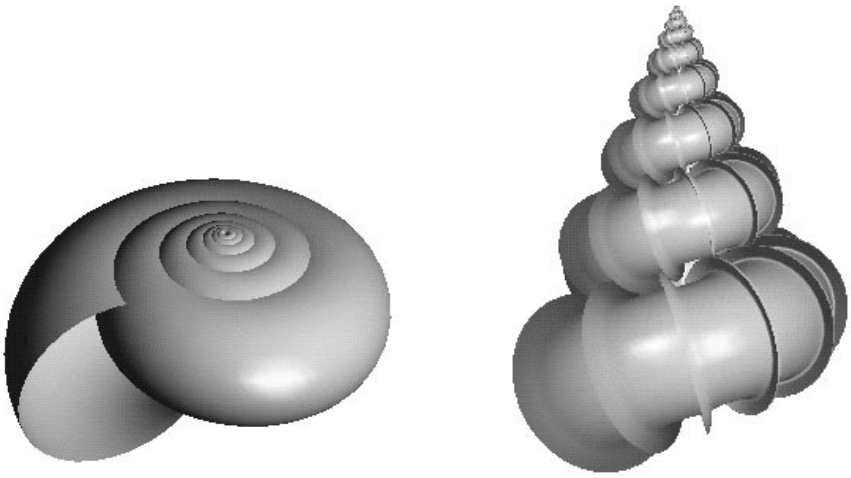
```
!(0.4)@Gs-(45)f-(45)!(0.2)@Gr(30,4.5,0,3.5)@Ge(20).
```

The cross-section of a generalized cylinder is perpendicular to the cylinder axis at each point $F(R_u(\frac{i}{n}))$. Thus it can happen that if the radius of the axis curvature is smaller than the radius of the generalized cylinder segment, two subsequent cross-section are crossed (Figure 10b and 10c). Since the occurrence of these artifacts on generalized cylinders representing parts of a plant is rare, there is no mechanism for avoiding these artifacts implemented in the modeling program `cpfg`.

The shell in Figure 11a illustrates the default change in radius of a tapering generalized cylinder that follows a helico-spiral [4]. The shell was generated in 117 steps using the following L-systems:

```
#define R    1.03  /* scaling between subsequent shell segments */
#define Ang1 20    /* angle of rotation between segments */
#define Ang2 2.1   /* angle of twist between segments */
#define Wid  5     /* width scaling */
ω: #(Wid)@Gt(1.1)@GsA(1)
p1: A(s) → +(Ang1)/(Ang2)F(s)A(s*R)
homomorphism
h1: F(s) → f(s)#(s*Wid)+(Ang1/2)@Gc(4)-(Ang1/2)
```

The axiom ω defines the first control point of the generalized cylinder axis and a module A. The production p_1 replaces the module A by a straight line segment and a new module A. The segment F is rotated in such a way that the sequence of segments forms a helico-spiral. The homomorphism production h_1 replaces the segment F by an invisible segment f of the same length with a control point at the end. The rotation before and after the control point makes the turtle heading vector at the control point approximately equal to the tangent of the helico-spiral.



a) b)

Figure 11: Shells generated with the default radius interpolation (a) and with modified radius tangents (b).

The possibility to control tangents of the longitudinal section of a generalized cylinder allows the user to create complex shapes, such as the Precious Wentletrap shell in Figure 11b. To generate the shell, the previous L-system was modified to distinguish odd and even segments F:

```
 #define R    1.02  /* scaling between subsequent shell segments */
 #define Ang1 -15   /* angle of rotation between segments */
 #define Ang2 -4    /* angle of twist between segments */
 #define Wid  6.4   /* width scaling */
ω: #(Wid)@Gt(1.1)@Gr(1)@GsA(0.9,0)
p1: A(s,n) → +(Ang1)/(Ang2)F(s,n)A(s*R,n+1)
homomorphism
h1: F(s,n): n%2!=0 → f(s)#(s*Wid*0.75)-(Ang1/2)@Gr(-100,2.5,0,4)
        @Gc(12)+(Ang1/2)
h2: F(s,n): n%2==0 → f(s*0.9)#(s*Wid)-(Ang1/2)@Gr(0,3,60,2.5)
        @Gc(12)@Gr(60,1,-100,1) f(s*0.1)@Gc(12)+(Ang1/2)
```

The homomorphism production h_1 replaces odd segments F by and invisible segment f with a control point P at the end. This control point is in the middle between two protruding ridges on the shell. Thus the diameter is reduced by one quarter. The homomorphism production h_2 defines two control points R and S close to each other. Both points are located on the ridge. The tangents between points P and R, 0° of length 3 and 60° of length 2.5, define the first slope of the ridge. Tangents (60°,1) and (-100°,1) between points R and S specify the tip of the ridge and tangents (-100°,2.5) and (0°,4) control the second,

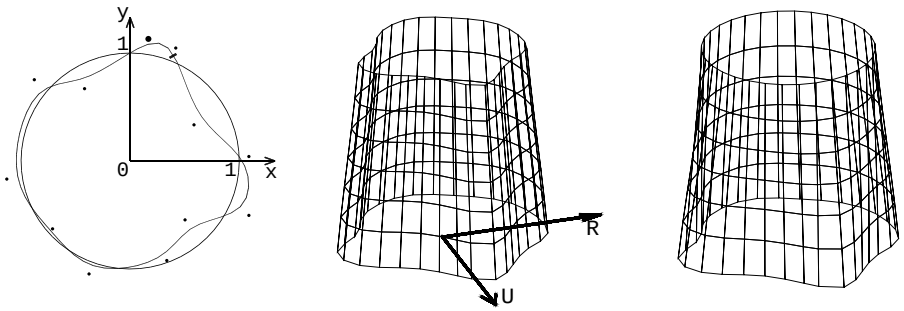


Figure 12: A 2D contour defined as a closed spline (a) is applied to both ends (b) or just one end (c) of a generalized cylinder segment.

concave slope of the ridge. Since the distance between control points is increasing towards the shell opening, the scaling of tangents is switched on by module `@Gr(1)` in the axiom.

3.2.3 Defining the cross-section

As a default, the cross-section (contour) of a generalized cylinder is a disk. It is possible to use an arbitrary contour defined as a closed three dimensional parametric curve consisting of several B-spline segments. The contour curve is specified by a set of control points which are read from a text file (see Appendix B.1). Control points are defined by two coordinates, in which case the third coordinates is assigned to be 0 (Figure 12a), or by three coordinates (Figure 13a). The dots in Figure 12a represent the specified control points with the first control point (close to the y axis) a little bigger. The circle with radius 1 represents the default circular contour for comparison.

If there are n control points P_i ($i = 0, ..., n-1$) specifying the contour, the contour consists of n B-spline segments. Each segment is computed using a parametric B-spline function $F_i(t)$, $i = 0, ..., n-1$ for $t \in \langle 0, 1 \rangle$, based on four control points P_i, $P_{(i+1)\%n}$, $P_{(i+2)\%n}$, and $P_{(i+3)\%n}$ (where $x\%y$ denotes the remainder from the division of x by y). Thus, segments $F_{n-3}(t)$, $F_{n-2}(t)$, and $F_{n-1}(t)$ are formed using control points at the beginning of the sequence of control points, forming a closed curve.

Generalized cylinders are visualized using cylindrical mesh strips with a certain number of polygons around the mesh. The number p is set at the beginning of the visualization (see Appendix B.1). The value of p is 32 in Figure 12b and 12c. When a non default contour is used, it is necessary to find the required number p of vertices on the contour. These vertices are computed in such a way that the length of the contour between them is approximately constant along the contour. For this purpose, $10n$ points on the contour (n is the number of control points specifying the contour) are computed with a constant step of the curve

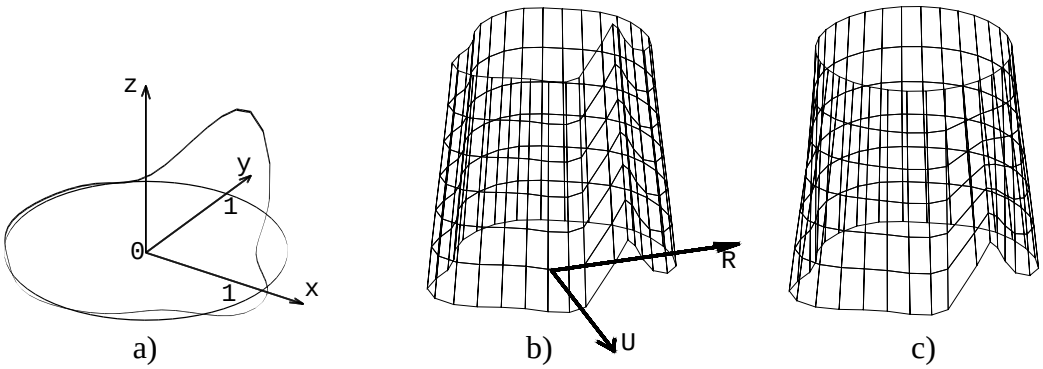


Figure 13: A 3D contour defined as a closed spline (a) is applied to one end of a generalized cylinder segment (b,c).

parameter t. Distances between the points (in the Euclidean space) are used to determine the equally spaced vertices V_i, $i = 0, .., p-1$, around the contour. To be able to connect vertices of different contours, all vertices are rotated around the z axis so that the first vertex lies on the x axis (in the contour coordinate space). In the Figure 12a, the first vertex is marked by a thick line, thus in this case all vertices are rotated clock-wise by about 70 degrees.

During visualization of a generalized cylinder, contour vertices V_i are used to compute vertices C_i of the contour of a cylindrical mesh along the generalized cylinder:

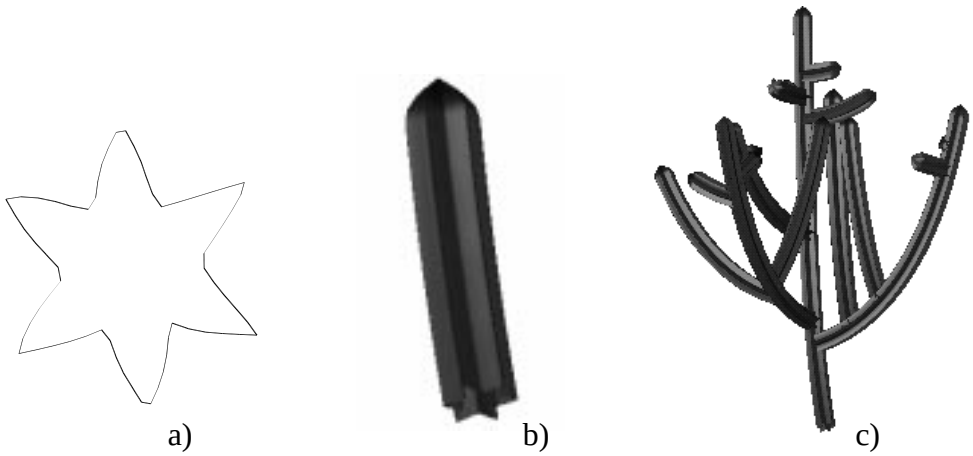
$$C_i = P + radius(V_{i,x}\vec{U} + V_{i,y}\vec{R} + V_{i,z}\vec{H})$$

using turtle's position as the origin and the up, the right (opposite to the left turtle vector), and the heading vector scaled by the turtle radius parameter as axes of a coordinate system into which coordinates of vertices are transformed. Thus point (0,0,0) in contour's coordinates corresponds to the point P on the axis of a generalized cylinder and vertex V_0, in our example rotated by 70° to lie on the x axis, is in the direction of the turtle's up vector $\vec{U}$ (see Figure 12b).

The default circle contour with index 0 can be changed by module @#(id) where id specifies the contour's index (see Appendix B.1). It is possible to use different contours for subsequent control points of a generalized cylinder. Let $F(u) = (F_x(u), F_y(u), F_z(u))$, $u \in \langle 0, 1\rangle$, be the Hermite curve of the cylinder axis, $R(t) = (R_u(t), R_v(t))$, $t \in \langle 0, 1\rangle$, the curve representing the longitudinal section. If V_i are vertices on the contour at the bottom of the generalized cylinder and V_i' vertices on the contour at the top, the contour vertices U_i for a point $P(t) = F(R_u(t))$, $t \in \langle 0, 1\rangle$, on the axis of the generalized cylinder are:

$$U_i(t) = V_i + t(V_i' - V_i)$$

and the radius of the contour is $R_v(t)$ (see examples in Figures 12c and 13c).



a) b) c)

Figure 14: A 2D contour (a) applied to a generalized cylinder (b) that was used as a part of a cactus model (c).

Generalized cylinders in Figure 12b and 12c were created by interpretation of the string

```
@#(2)@Gsf(0.5)!@Ge(6)&(90)@c
```

and

```
@#(2)@Gsf(0.5)@#(0)!@Ge(6)&(90)@c
```

respectively. Both strings specify two control points with one or two different contours (with index 2 and a default 0)[1]. The number of polygons along the contour was set to 32.

Figure 13a illustrates an example of a three-dimensional contour. The contour file from the previous example was modified by adding a third coordinate to each contour control point. This coordinate is equal to 0 in all but the first control point where it is 1. The contour is applied to segments generated using the same strings as in Figure 12b and 12c. The resulting generalized cylinders are in Figure 13b and 13c.

More realistic examples of the use of closed contours are shown in Figure 14 and Figure 15. The contour in Figure 14a was applied to branch segments (Figure 14b) of a cactus *Lemaireocereus chende* in Figure 14c. The cactus was modeled by the following L-system:

[1] The two-dimensional contour was created in the drawing program `xfig` and converted to a text file with coordinates of control points.

```
ω: @#(2)@Ts(1,0.07)!(0.9)S FA(0)?H(0,0,0)E
p1: A(ord) > ?H(x,y,z) : y<0 → %
p2: A(ord) → FB(ord)/A(ord)
p3: B(ord) :  ran(1) < 0.2-ord*0.1
        → [-(90)/(nran(90,30))SA(ord+1)?H(0,0,0)E]
          [+(90)/(nran(90,30))SA(ord+1)?H(0,0,0)E]
p4: B(ord) :  ran(1) < 0.2-ord*0.1
        → [-(90)/(nran(90,30))SA(ord+1)?H(0,0,0)E]
p5: B(ord) :  ran(1) < 0.2-ord*0.1
        → [+(90)/(nran(90,30))SA(ord+1)?H(0,0,0)E]
p6: B(ord) --> ε
homomorphism
h1: S → @Gs
h2: E → @Gr(0,1,-45,1)!(0.01)@Ge(4)
h3: F → f(0.5)@Gc(2)f(0.5)
```

The simulation starts with a single line segment F followed by an apex A defined in the axiom ω. The production p_2 replaces the apex with a segment F, a branch marker B, and a new apex A. Productions p_3, p_4, and p_5 create either two branches or just one to the left or one to the right with a probability decreasing with an increasing branch order (specified by the parameter of modules A and B). If none of them is applied, production p_6 removes the branch marker B. Each apex is followed by a module $?H$ for querying the turtle heading vector. If an apex is oriented downwards, it is removed by the production p_1.

Generalized cylinders are introduced by homomorphism productions. The production h_1 replaces a module S appearing at the base of each branch by the first control point of a generalized cylinder. A module E terminates a generalized cylinder with a pointed tip by changing the tangent of the longitudinal section at the tip to ($-45°$,1). The production h_3 replaces a straight line segment F with an invisible line segment with a control point in the middle.

The plant (*Aloe variegata*) in Figure 15c was generated using the following L-system:

```
ω: @Gr(1)A(25)!(0.8),(3)F(12)!(0.4)P(30)
p1: A(n): n>1 → f(0.1)/(137.5)[-(15+2*n)L(10)]A(n-1)
p2: P(n): n>1 → F(0.05+n/80)/(137)
                [-(n*4)G(0.7+n/15)]P(n-1)
homomorphism
h1: L(s) → [@#(2)@Tx(1),(1)!(s/4)@Gs-(10)f(s)-(15)
           @Gr(5,1,-20,1)!(s/100)@Ge(10)]
h2: G(s) → !(0.1)F(0.7)[]!(0.4),(2)
           @#(3)!(2.7/4)@Gs-(10)f(s)-(15)
           @Gr(0,4,-50,0.3)!(2/100),(2)@Ge(10)
```

The axiom defines the plant structure which consists of a whorl of thick leaves at the bottom

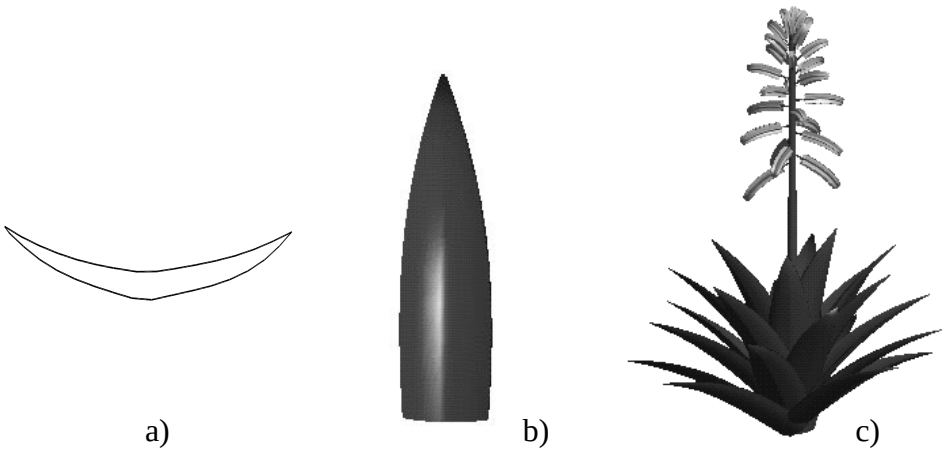


Figure 15: A 2D contour (a) applied to a leaf (b). The whole plant with the whorl of leaves (c).

(the module A) a tapered stem F and a phyllotactic pattern of flowers at the top (the module P). The production p_1 produces a whorl of leaves L with a decreasing initial angle. Each leaf is then visualized using the homomorphism production h_1 specifying two control points of a generalized cylinder axis with a user-defined contour (Figure 15a). The shape of the leaf is also controlled by the module $@Gr$ modifying the tangents of the leaf longitudinal section. The production p_2 produces flowers G visualized using the production h_2. The flower contour is similar to the contour from the previous example (Figure 14a) and again only two points of a generalized cylinder are defined.

Open contours

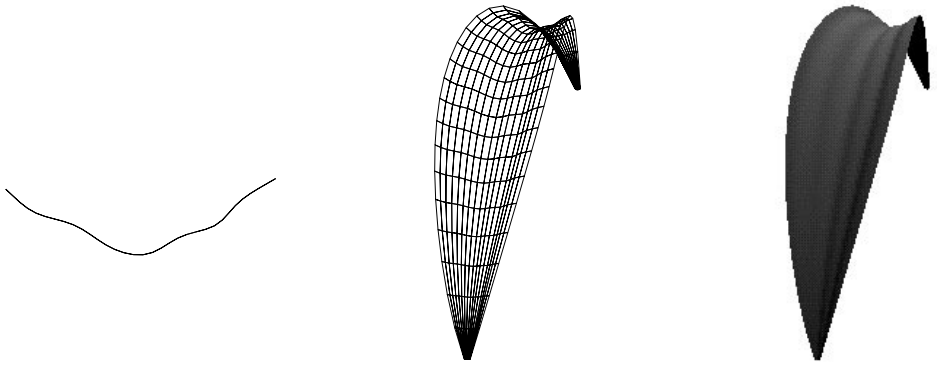
The previous section describes the use of closed contours. It may be useful to consider also open contours which can be used to define the cross-section of a long thin leaf blade, for example.

Open contours are defined in the same way as closed contours. The only difference is that n control points P_i define $(n-3)$ B-spline segments specified by parametric B-spline functions $F_i(t)$, $i = 0, ..., n-1$ for $t \in \langle 0, 1 \rangle$, based on four control points P_i, P_{i+1}, P_{i+2}, and P_{i+3}. Since the same number p of polygons is drawn along the open contour as along the closed contour, the same number of contour vertices V_i, $i = 0, ..., p-1$ is computed along the contour. This allows to use the same interpolation techniques between two different open contours or an open contour and a closed contour as described above.

Figure 16 illustrates the use of an open contour (a) in a long leaf blade visualized using a generalized cylinder (Figure 16b and 16c). The leaf blade is defined by the string:

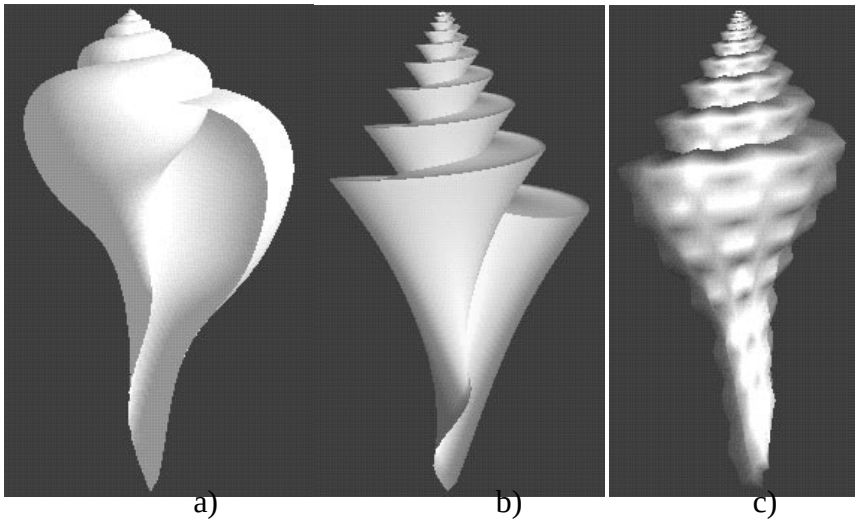
```
-(40)@#(2)@Gs-(90)f(10)@Gr(1)@Gr(20,1.5,-20,1.5)@Gt(2,1)-(50)@Ge(20).
```

The first and the last contour control point is repeated three times to have the contour curve



a) b) c)

Figure 16: A contour defined as an open spline (a) is applied to a generalized cylinder (b,c).



a) b) c)

Figure 17: Shells generated by sweeping an open contour along a helico-spiral. Two contours are interpolated in the third shell.

starting and terminating at the first and last contour control point, respectively.

Another example illustrates the application of an open contour to the shell model from Section 3.2.2. Shells in Figures 17a and 17b use a conical and a triangular contour. The L-system from Section 3.2.2 was modified by adding a rotation around the turtle heading vector to properly orient the contour:

```
#define R    1.04  /* scaling between subsequent shell segment */
#define Ang1 -20   /* angle of rotation between segments */
#define Ang2 -3.9  /* angle of twist between segments */
#define Ang3 78    /* initial rotation of the contour */
#define Wid  5     /* width scaling */
ω: #(5)@Gt(1.0)@#(2)\(Ang3)@Gs/(Ang3)A(1)
p1: A(s) → +(Ang1)/(Ang2)F(s)A(s*R)
homomorphism
h1: F(s): → f(s)#(s*Wid)-(Ang1/2) \(Ang3)@Gc(4)/(Ang3)+(Ang1/2)
```

In the case of the shell with the triangular contour, the L-system parameters are as follows: $R = 1.02$, $Ang1 = -20$, $Ang2 = -3$, and the contour rotation angle is $Ang3 = 80$.

The shell in Figure 17c was generated using the L-system:

```
#define R    1.02  /* scaling between subsequent shell segment */
#define Ang1 -20   /* angle of rotation between segments */
#define Ang2 -3    /* angle of twist between segments */
#define Ang3 86    /* initial rotation of the contour */
#define Wid  5     /* width scaling */
ω: #(5)@Gt(1.0)@#(2)\(Ang3)@Gs/(Ang3)A(1,0)
p1: A(s,n) → +(Ang1)/(Ang2)F(s,n)A(s*R,n+1)
homomorphism
h1: F(s,n): n%2==0 → f(s)#(s*Wid)-(Ang1/2)\(Ang3)
            @#(3)@Gc(6)/(Ang3)+(Ang1/2)
h2: F(s,n): n%2!=0 → f(s)#(s*Wid*1.04)-(Ang1/2)\(Ang3)
            @#(2)@Gc(6)/(Ang3)+(Ang1/2)
```

in which the odd and even shell segments have different contours.

The Pelican shell in Figure 18 uses two contours as in the previous example plus the third contour for the opening. To generate the shell, the previous L-system was extended by the following homomorphism production:

```
h1: F(s,n): n==Steps-1 → +(ANG1/2)f(2*s)#(s*Wid)-(Ang1/2)\(Ang3)
            @#(4)@Gc(6)/(Ang3)+(Ang1/2)
```

changing the contour for the very last shell segment (the index of F is equal to the number of simulation steps minus one). The L-system parameters are: $R = 1.025$, $Ang1 = -20$, $Ang2 = 3.6$, and $Ang3 = 84$.

3.3 Twist of generalized cylinders

As mentioned in Section 3.2.3, coordinates of contour vertices are rotated around the z axis in the contour coordinate space so that the first vertex is on axis x. In the world coordinate space it means that the first contour vertex lies in the direction of the turtle's left vector. It may happen, though, that the turtle left and up vectors are rotated around the heading vector

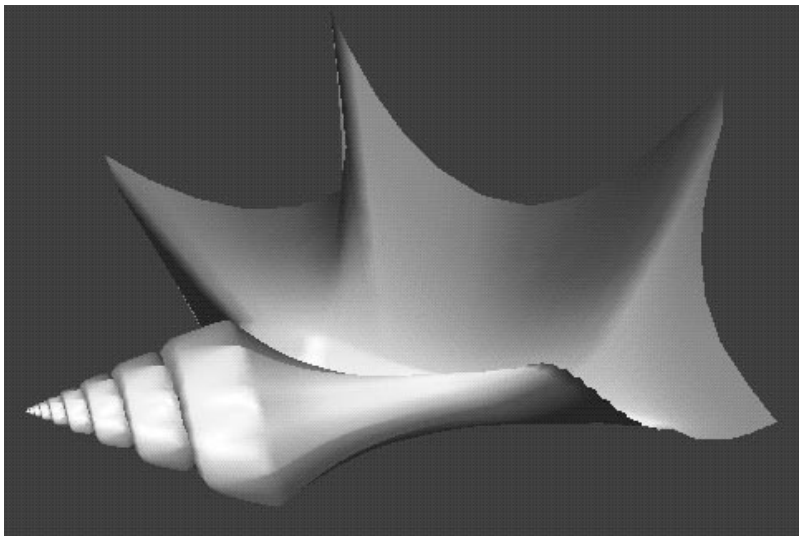
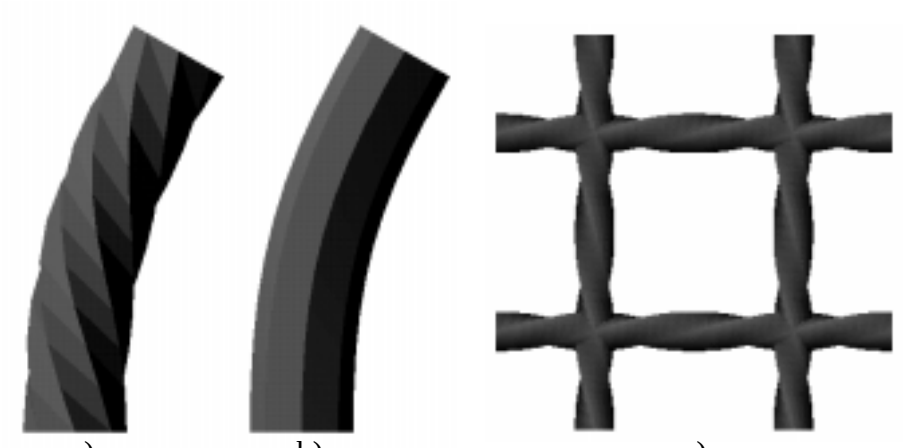


Figure 18: Pelican shell.



a) b) c)

Figure 19: A branch segment rendered with the original twist (a) and minimized twist (b). Sometimes the twist is desired (c).

(using symbols / or \) between two control points. If contour vertices are always aligned with the turtle orientation and the first vertex on the first contour is connected with the first vertex on the second contour, the resulting generalized cylinder is twisted. An example of twisted generalized cylinder is shown in Figure 19a. It results from the interpretation of the string

```
@Gsf-(30)f\(180)@Ge(10).
```

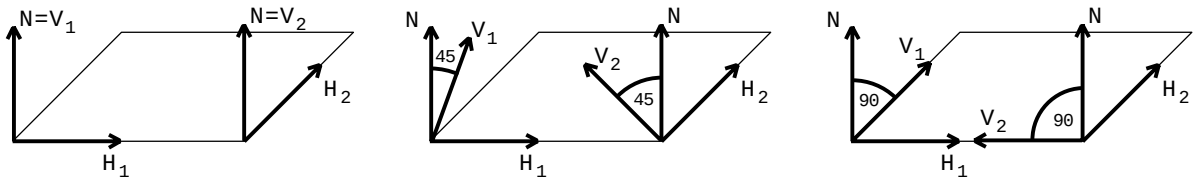


Figure 20: Two consecutive frames with different angles between vector $\vec{V_i}$ and the normal $\vec{N}$ of the plane of rotation.

To avoid the twist, the cross-section (contour) of a cylindrical mesh strip is aligned with respect to the contour of the previous strip. Instead of rotating the first contour vertex to lie on the turtle left vector, a new reference vector $\vec{V}$ is computed and the first contour vertex is placed in its direction from the turtle position. Thus, the turtle position P, the heading vector $\vec{H}$, the vector $\vec{V}$, and the vector $\vec{H} \times \vec{V}$ form a reference frame defining position and orientation of the contour at a given point along the axis of the generalized cylinder.

There are several methods for constructing reference frames along a parametric curve. A brief overview is given in [3]. For example, it is possible to determine three orthonormal frame vectors from the parametric equation of the curve, as in the case of the Frenet frame [5]. Unfortunately, the Frenet frame is undefined along straight line segments or can be suddenly reversed on either side of an inflection point.

Other methods for constructing frames applicable to our problem include the rotation-minimizing method used by Bloomenthal [2] to compute frames of tree limbs and the double-cross method proposed by Sloan (mentioned in [3]).

The Bloomenthal's rotation-minimizing method was selected. Having a previous set of vectors ($\vec{H}_1$, $\vec{V}_1$, $\vec{H}_1 \times \vec{V}_1$), the curent frame ($\vec{H}_2$, $\vec{V}_2$, $\vec{H}_2 \times \vec{V}_2$) for a given vector $\vec{H}_2$ is determined by rotating the frame ($\vec{H}_1$, $\vec{V}_1$, $\vec{H}_1 \times \vec{V}_1$) around $\vec{H}_1 \times \vec{H}_2$ so that rotated $\vec{H}_1$ matches with $\vec{H}_2$. Let us consider two adjacent stem segments. If they lie in one line ($\vec{H}_1 = \vec{H}_2$), the vector $\vec{V}_2$ of the second one is the same as vector $\vec{V}_1$ of the first one. If they define a plane (with normal vector $\vec{H}_1 \times \vec{H}_2$), vectors $\vec{V}_i$ of both segments have the same angle with this plane and the same angle with the plane's normal $\vec{H}_1 \times \vec{H}_2$ because the active frame 2 was constructed by rotating frame 1 around the vector $\vec{H}_1 \times \vec{H}_2$. Figure 20 shows three examples in which the angle of vectors $\vec{V}_i$ with the two-segment plane normal $\vec{N} = \vec{H}_1 \times \vec{H}_2$ is (from left to right) 0, 45, and 90 degrees.

By default, generalized cylinders are drawn in such a way that their twist is minimized to obtain smooth connections (see Figure 19b). To be able to create twisted segments (e.g. ornamental structures such as the one in Figure 19c) it is possible to switch off the minimization of the twist (see Appendix B).

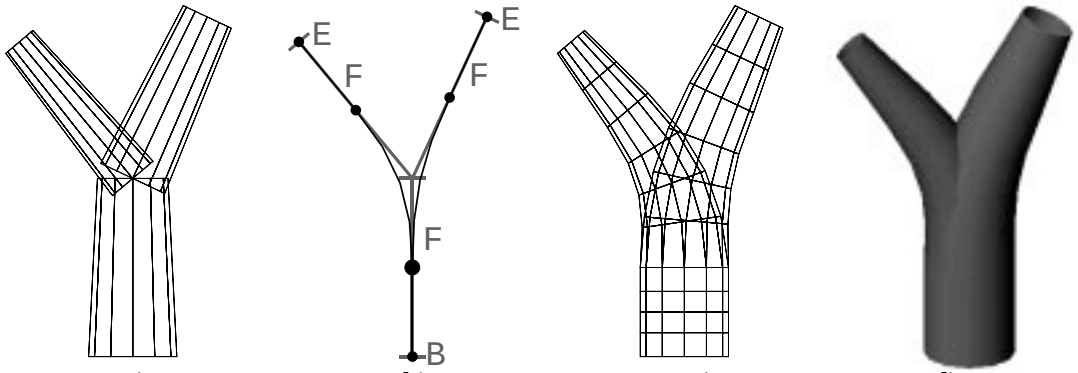


Figure 21: A branch fork rendered as: a) straight segments, b) parametric curves (dots denote control points), c,d) generalized cylinders (wireframe and shaded).

3.4 Branching of generalized cylinders

Branches in an L-system generated string are delimited by modules `[` and `]`. When module `[` is interpreted, actual turtle parameters are pushed onto a stack. When a branch is finished and module `]` is encountered, the turtle parameters are retrieved from the stack and another branch can be interpreted. For example, the string:

```
F![+(40)!F!][-(25)F!]
```

is visualized in Figure 21a. The first segment `F` supports two branches: to the left with an angle of 40 degrees from the original direction and to the right with an angle 25 degrees. Module `!` reduces the segment width by 0.1.

The branches can be visualized using generalized cylinders. A module `B` is placed at the beginning of the branch and a module `E` at the end of each terminal branch segment to start and terminate generalized cylinders:

```
BF![+(40)!F!E][-(25)F!E]
```

The following homomorphism productions replace each straight segment `F` of length 1 with an invisible segment of the same length with one control point in the middle (Figure 21b):

```
homomorphism
p1:  B → @Gb
p2:  E → @Ge(2)
p3:  F → f(0.5)@Gc(4)f(0.5)
```

creating a generalized cylinder shown in Figure 21c and 21d. Placing a control point in the middle of each straight segment results in a generalized cylinder that closely follows the original branching structure in Figure 21a.

Information about the active control point (the last one specified) is associated with the turtle as one of the turtle parameters. Thus when the first branch is finished and turtle

parameters are retrieved from the stack, the second branch connects to the same control point (in Figure 21a represented by a bigger dot) as the first branch.

Note that the branch visualized using generalized cylinders is very similar to the original one, only the connection of branches is smoother. Since it is slower to draw generalized cylinders than straight line segments, plant models are often developed using straight lines to represent branches. If one module, for example B, is placed at the beginning of the string and another symbol, e.g. E, at the end of each branch, the final structure can be visualized using generalized cylinders with control points in the middle of each line segment by applying the above homomorphism.

4 Surfaces

The material presented in this section has originally appeared in Jim Hanan's Ph.D. dissertation [9].

4.1 Predefined surfaces

A standard computer graphics method for defining surfaces makes use of *parametric bicubic patches* [1, 6]. This technique is well suited for interactive design of arbitrary surface shapes. The control points that define an individual patch can be modified using a graphical interface [8, Section 4.2], and several patches can be combined to create a more complex surface [8, Section 3.5]. The resulting surface definition can then be stored in a file for use during turtle interpretation.

Predefined surfaces are incorporated into a plant model by extending the L-system alphabet. When the turtle encounters a symbol representing a surface preceded by a tilde ($\sim$), the corresponding surface is drawn. The exact position and orientation of a predefined surface S is determined using the user-defined *contact point* P_S, *heading vector* $\vec{H}_S$, and *up vector* $\vec{U}_S$ as references. The surface is translated in such a way that its contact point matches the current position of the turtle, and is rotated to align its heading and up vectors with the corresponding vectors of the turtle. If a surface represents an internal part of a plant's structure, the turtle is positioned at a user-defined *end point* once the surface has been drawn.

The following L-system produces the apple blossom shown on the left side of Figure 22 in two derivation steps, given an angle increment of $18°$.

$$\begin{array}{lll} \omega: & \texttt{FFFFFB} & \\ p_1: & \texttt{B} & \rightarrow \quad [\texttt{S////S////S////S////S}] \\ p_2: & \texttt{S} & \rightarrow \quad [\sim\texttt{C}][\sim\texttt{P}][\wedge\wedge\,\texttt{F}[-\texttt{F}][+\texttt{F}]] \end{array}$$

The F's in the axiom represent the blossom's stem, while the B represents a bud. In the first derivation step, production p_1 replaces the symbol B by five segments S separated by

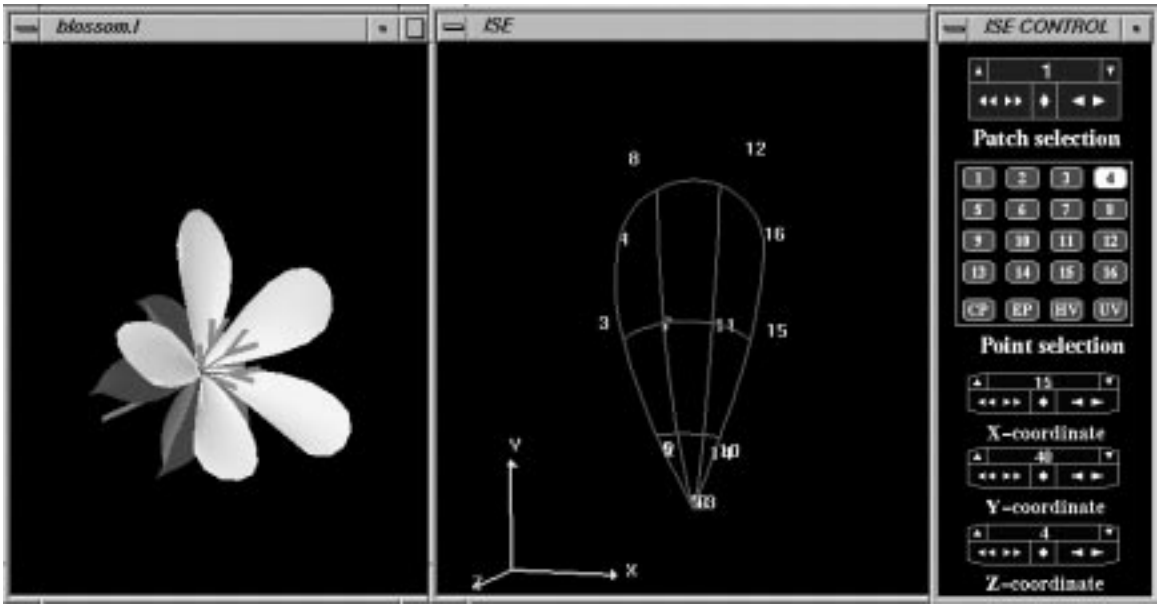


Figure 22: Apple blossom and interactive surface editor

/ symbols. In the second derivation step, production p_2 creates the three components of each segment, a calyx leaf [∼C], a petal [∼P], and a stamen [∧ ∧ F[−F][+F]]. During turtle interpretation, the predefined surfaces C, representing the leaf, and P, representing the petal, will be incorporated into the image. These surfaces were designed using the interactive surface editor shown on the right in Figure 22.

4.2 Developmental surfaces

Predefined surfaces do not "grow"; if a developmental sequence is required, surfaces representing individual stages of surface growth must be separately defined and incorporated into the model. An alternate approach is to allow the turtle to create polygons directly. The opening brace "{" and the closing brace "}" are introduced as commands that delimit the substring which determines the boundary of a polygon to be filled. When an opening brace is encountered during interpretation, an empty *list of vertices* representing the current polygon is created. Subsequently, whenever an F or f is interpreted, the resulting turtle position is appended as a vertex on the list. Interpretation of the closing brace causes the current polygon to be filled. Using this approach, L-system productions can be employed in a number of different ways to change the size and shape of a polygon over time.

The first possibility is to trace surface boundaries using the turtle and fill the resulting polygons, as in the L-system given below:

Figure 23: A model of a fern frond with polygonal leaflets

$$\begin{array}{ll} \omega: & \mathtt{L} \\ p_1: & \mathtt{L} \rightarrow \{-\mathtt{FX} + \mathtt{X} - \mathtt{FX} - \mid -\mathtt{FX} + \mathtt{X} + \mathtt{FX}\} \\ p_2: & \mathtt{X} \rightarrow \mathtt{FX} \end{array}$$

Production p_1 defines leaf L as a closed planar polygon. Production p_2 increases the lengths of its edges linearly. This technique was used to model the leaflets on the fern branch in Figure 23. Leaflets appear in order of age with the youngest at the top.

In practice, the tracing of polygon boundaries only produces acceptable effects for small, flat surfaces. In other cases it is more convenient to use a tree structure as a framework for a polygon. Vertices are specified by a sequence of turtle positions marked by the dot symbol (`.`). An example is given in Figure 24. The letter `G` has been used instead of `F` to indicate that the segments enclosed between the braces should not be interpreted as the edges of the constructed polygon. The numbers correspond to the order in which the turtle specifies the vertices.

In the techniques discussed so far, the turtle specifies the vertices of one polygon, then moves on to the next. Further flexibility in surface definition can be achieved by interleaving vertex specifications for different polygons. In order to accomplish this, the interpretation of braces is redefined as follows. A string containing nested braces is evaluated using two data structures, the list of vertices representing the current polygon and a *polygon stack*. At the beginning of string interpretation, both structures are empty. The interpretation of an opening brace "{" initializes a new polygon list and pushes it onto the polygon stack. When the turtle encounters a closing brace "}" it pops the current polygon from the top of the stack

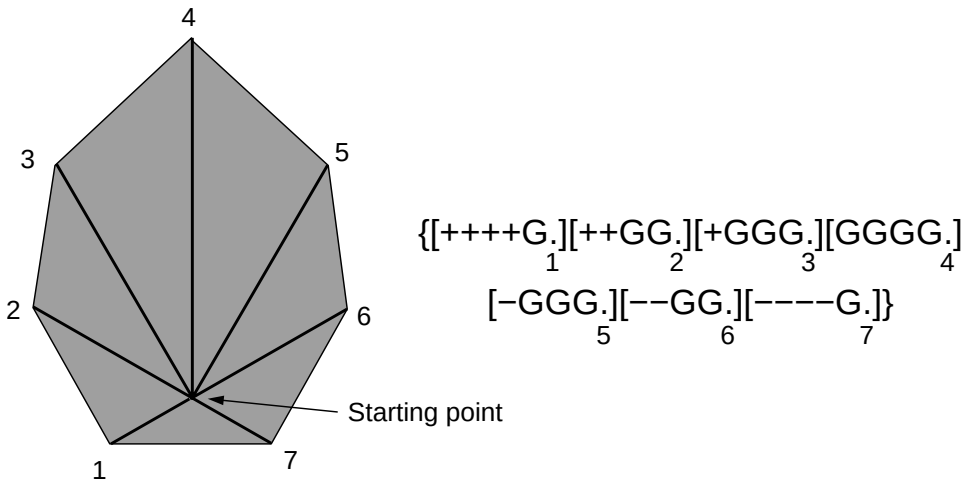


Figure 24: Surface specification using a branching structure as a framework. The numbers correspond to the order of vertex specification by the turtle.

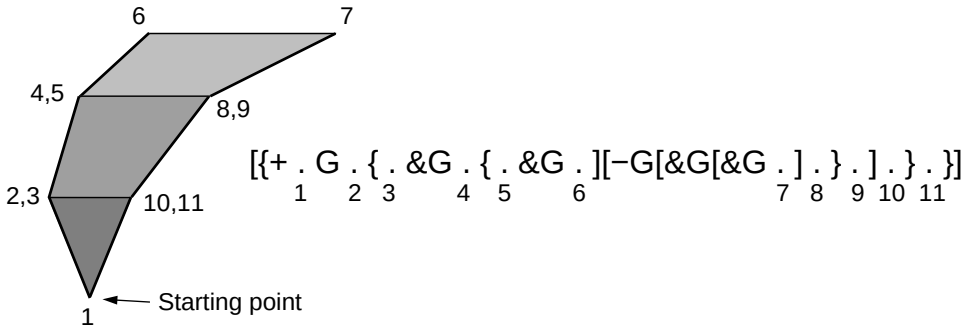


Figure 25: Surface specification using stacked polygons. The numbers correspond to the order of vertex specification by the turtle.

and draws the polygon specified by its list of vertices. An example of string interpretation involving nested braces is given in Figure 25. This surface cannot be described using a single pair of braces, since methods for filling non-planar polygons are not well defined. Therefore, the figure is decomposed into three polygons connecting the following sets of vertices: $\{1, 2, 11\}$, $\{3, 4, 9, 10\}$, and $\{5, 6, 7, 8\}$. Note that it is necessary to have separate stacks for polygons and branches, as they operate independently. In this case, all three polygons start in one branch and are completed in another.

4.3 Developmental bicubic surfaces

As described in previous Section, L-systems can be used to model the development of plant organs, such as leaves and petals, using polygons which are modified over time. However, bicubic surfaces provide a more convenient method for modelling smooth curved surfaces; a very complex L-system would be required to produce a polygonal surface as smooth as a bicubic patch. Developmental bicubic surfaces can be incorporated into a model using the following set of black-box routines, which allow the specification of a Bezier-form bicubic surface [1, 6, 8].

- @PS(i) initializes the four rows and columns of control points for surface i to $(0, 0, 0)$.
- @PC(i, r, c) assigns the current position of the turtle to the control point of surface i in row r and column c.
- @PD(i, s, t) draws the surface defined by the control points of surface i using s lines along the rows and t along the columns.

The first step in creating a developmental model of a plant organ is to define the initial and final shapes in the sequence. When using an interactive surface editor, the user works with 16 control points for each surface patch. The manipulation of a three-dimensional control point using a two-dimensional input device, such as a mouse, is not necessarily straightforward. In addition, the creation of the symmetric shapes common in plant components often requires the concerted readjustment of several control points, which can be a tedious task using a standard interactive editor. Parametric L-systems can be used to implement a more intuitive set of parameters defining a particular class of surface shapes. The following L-system allows the user to manipulate parameters for petal width, length, and bending angles in order to model members of a family of petals. It is a simple hierarchical

model of one possible control point layout.

L-system 1: Bicubic surface petals

```
#define CL 100          /* Central length */
#define BW 35           /* Base width */
#define TW 35           /* Tip width */
#define BA 0            /* Base angle */
#define TA 0            /* Tip angle */
```

ω : P

p_1 : P $\rightarrow$ [S[l][r]B[L][R]D]

p_2 : S $\rightarrow$ @PS(0)f(30)

p_3 : B $\rightarrow$ $\wedge$(BA)f(CL) $\wedge$ (TA)

p_4 : D $\rightarrow$;(100)@PD(0, 4, 4)

p_5 : l $\rightarrow$ +(90)f(BW)@PC(0, 0, 0) + (90 + atan(CL/BW))
[|f(CL/3)@PC(0, 1, 0) − (90) $\wedge$ (BA)f(BW ∗ 2/3)@PC(0, 1, 1)]
[f(50)@PC(0, 0, 1)]

p_6 : r $\rightarrow$ −(90)f(BW)@PC(0, 0, 3) − (90 + atan(CL/BW))
[|f(CL/3)@PC(0, 1, 3) + (90) $\wedge$ (BA)f(BW ∗ 2/3)@PC(0, 1, 2)]
[f(50)@PC(0, 0, 2)]

p_7 : L $\rightarrow$ +(90)f(TW)@PC(0, 3, 0) + (90 − atan(50/TW))
[f(CL/3))@PC(0, 2, 0) + (90) $\wedge$ (TA)f(TW ∗ 2/3)@PC(0, 2, 1)]
[|f(30)@PC(0, 3, 1)]

p_8 : R $\rightarrow$ −(90)f(TW)@PC(0, 3, 3) − (90 − atan(50/TW))
[f(CL/3)@PC(0, 2, 3) − (90) $\wedge$ (TA)f(TW ∗ 2/3)@PC(0, 2, 2)]
[|f(30)@PC(0, 3, 2)]

According to production p_1 a petal is composed of the start segment S, left and right halves of the leaf base l and r, the body B, left and right halves of the leaf tip L and R, and the drawing segment D. Production p_2 issues the patch initialization command @PS(0). The f(30) module moves the turtle so that the edge of the surface will go through the turtle's initial position. The petal is modelled as two laterally symmetric halves, each consisting of a base and tip portion. Productions p_5 and p_6 define the leaf base by producing mirror-image responses in the turtle with respect to the central axis. Productions p_7 and p_8 do the same for the leaf tip. Production p_3 defines the central length and relative angles of the base and tip. Production p_4 specifies the colour command ;(100) and the patch drawing command @PD(0, 4, 4). As illustrated in Figure 26, the base of the leaf is defined by the first two rows of control points in the bicubic patch, while the tip is defined by the last two rows. This L-system allows the user to control a petal's shape in terms of its central length CL, its tip and base width, TW and BW, and the angles between base and center line, BA, and between center line and tip, TA. The remainder of the angles and lengths are defined by the family of surfaces to be modelled and the geometry of a Bezier patch. For instance, in order to

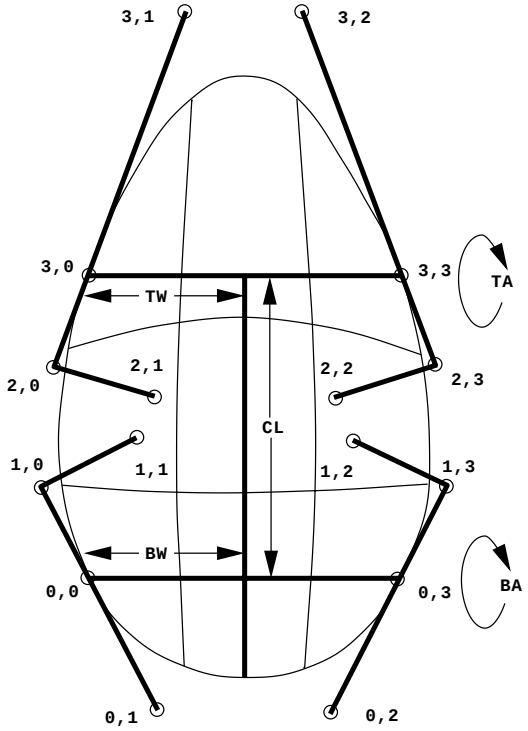


Figure 26: Petal control structure. Control points are labelled by row and column.

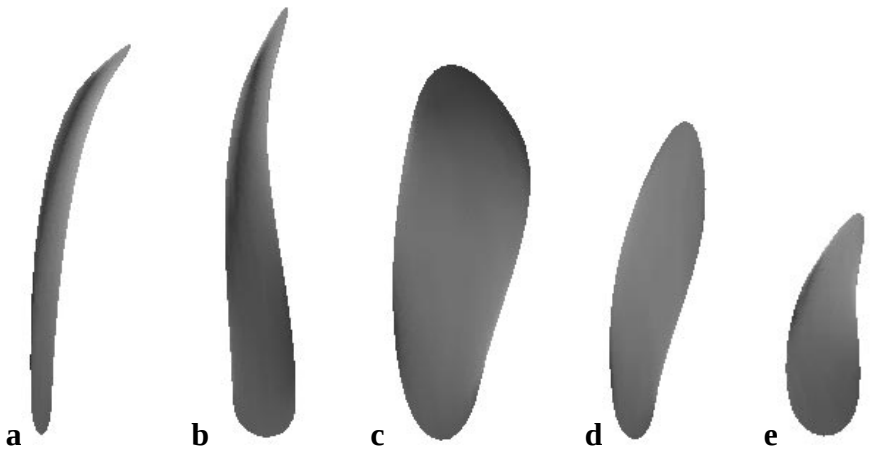


Figure 27: Petal shapes

maintain first order continuity of the edge passing through a control point at a corner of the patch, the control point and its neighbours in the outside row and column must be collinear.

Interactive manipulation of the parameters in the `#define` statements produced the petal shapes in Figure 27, which correspond to the values in the following table.

Figure	CL	BW	TW	BA	TA
a	150	5	5	25	50
b	150	15	5	0	50
c	120	20	25	12	-40
d	100	10	15	25	0
e	50	15	10	12	40

Once the initial and final shapes have been chosen, an L-system must be designed to interpolate between the two shapes. For example, the following L-system interpolates between shapes e and c in Figure 27.

L-system 2: `Developmental bicubic surface petal`

```
#define N    10  /* Number of steps */
#define ICL  50  /* Initial central length */
#define FCL  150 /* Final central length */
#define IBW  15  /* Initial base width */
#define FBW  15  /* Final base width */
#define ITW  10  /* Initial tip width */
#define FTW  5   /* Final tip width */
#define IBA  15  /* Initial base angle */
#define FBA  0   /* Final base angle */
#define ITA  35  /* Initial tip angle */
#define FTA  50  /* Final tip angle */
```

L-system 2: Developmental bicubic surface petal - continued

ω : P

p_1 : P $\rightarrow$ [S[l][r]B[L][R]D]

p_2 : S $\rightarrow$ @PS(0)f(30)

p_3 : B $\rightarrow$ $\wedge$(IBA, FBA, (FBA − IBA)/N)f(ICL, FCL, (FCL − ICL)/N)
$\wedge$(ITA, FTA, (FTA − ITA)/N)

p_4 : D $\rightarrow$;(100)@PD(0, 4, 4)

p_5 : l $\rightarrow$ +(90)f(IBW, FBW, (FBW − IBW)/N)@PC(0, 0, 0) + (90 + atan(ICL/IBW),
90 + atan(FCL/FBW), (atan(FCL/FBW) − atan(ICL/IBW))/N)
[|f(ICL/3, FCL/3, (FCL − ICL)/3/N)@PC(0, 1, 0) − (90)
$\wedge$(IBA, FBA, (FBA − IBA)/N)f(IBW ∗ 2/3, FBW ∗ 2/3, 2/3 ∗ (FBW − IBW)/N)
@PC(0, 1, 1)][f(50)@PC(0, 0, 1)]

p_6 : r $\rightarrow$ −(90)f(IBW, FBW, (FBW − IBW)/N)@PC(0, 0, 3) − (90 + atan(ICL/IBW),
90 + atan(FCL/FBW), (atan(FCL/FBW) − atan(ICL/IBW))/N)
[|f(ICL/3, FCL/3, (FCL − ICL)/3/N)@PC(0, 1, 3) + (90)
$\wedge$(IBA, FBA, (FBA − IBA)/N)f(IBW ∗ 2/3, FBW ∗ 2/3, 2/3 ∗ (FBW − IBW)/N)
@PC(0, 1, 2)][f(50)@PC(0, 0, 2)]

p_7 : L $\rightarrow$ +(90)f(TW)@PC(0, 3, 0) + (90 − atan(50/TW))
[f(ICL/3), FCL/3, (FCL − ICL)/3/N)@PC(0, 2, 0) + (90)
$\wedge$(ITA, FTA, (FTA − ITA)/N)f(ITW ∗ 2/3, FTW ∗ 2/3,
2/3 ∗ (FTW − ITW)/N)@PC(0, 2, 1)][|f(30)@PC(0, 3, 1)]

p_8 : R $\rightarrow$ −(90)f(TW)@PC(0, 3, 3) − (90 − atan(50/TW))
[f(ICL/3, FCL/3, (FCL − ICL)/3/N)@PC(0, 2, 3) − (90)
$\wedge$(ITA, FTA, (FTA − ITA)/N)f(ITW ∗ 2/3, FTW ∗ 2/3,
2/3 ∗ (FTW − ITW)/N)@PC(0, 2, 2)][|f(30)@PC(0, 3, 2)]

p_9 : f(v, V, i) : v < V $\rightarrow$ f(v + i, V, i)

p_{10} : +(v, V, i) : v < V $\rightarrow$ +(v + i, V, i)

p_{11} : −(v, V, i) : v < V $\rightarrow$ −(v + i, V, i)

p_{12} : $\wedge$(v, V, i) : v < V $\rightarrow$ $\wedge$(v + i, V, i)

The turtle interpretation commands with values to be interpolated have three parameters: v representing the current value, V representing the limit or final value, and i representing the increment to be applied in each step. Productions p_1 to p_8 are the same as before, except that modules representing commands with parameters to be interpolated have the appropriate initial values included. Productions p_9 to p_{12} control the linear interpolation of lengths and angles. This L-system produces the sequence of images presented in Figure 28. The sequence of flower heads shown in Figure 29 comes from an animation of rose campion development produced by Prusinkiewicz and Hammel [13] using a similar technique.

The presence of parameters allows the specification of control points by row and column number in the black-box routines. A less intuitive symbolic identification of the black-box routines would have been required for standard L-systems.

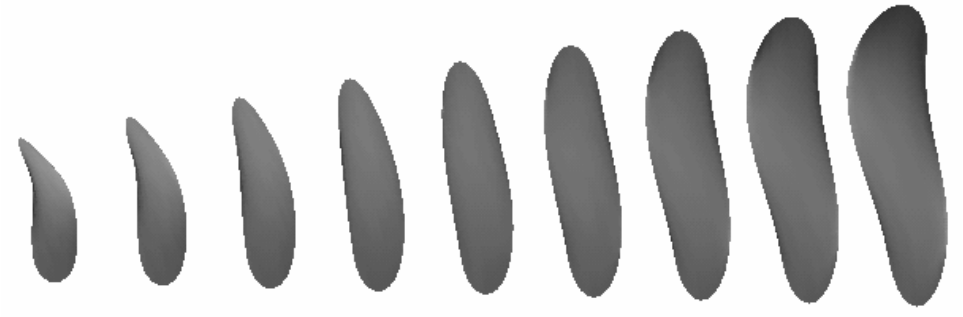


Figure 28: Development of a petal

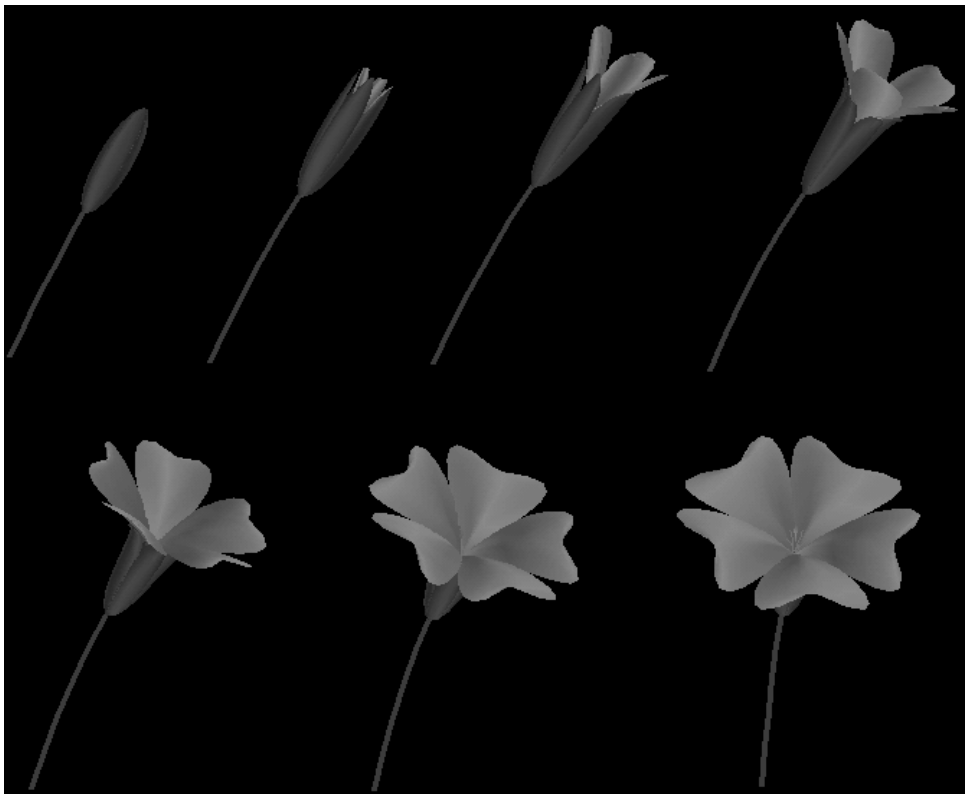


Figure 29: Development of a rose campion flower ©1991 P. Prusinkiewicz and M. Hammel

5 Textures

An important aspect of graphical modeling is the use of textures. In the existing implementation of `cpfg`, textures can be defined as two-dimensional images which are mapped on a given surface. One possibility is to use the image color directly on the surface. In that case,

though, the surface cannot be shaded. The other option is to multiply the surface diffuse color by the intensity at a given pixel of the texture image. The intensity, ranging from 0 to 1, is either computed from the red, blue and green components of the image color using the following formula [7]:

$$I = R \cdot 0.299 + G \cdot 0.587 + B \cdot 0.114$$

or obtained directly from a single-channel image. Appendix B.2 explains how to switch between these two modes.

5.1 Textured bicubic surfaces

The texture of a predefined bicubic surface can be specified either as a viewing attribute, in which case all instances of the surface have the same texture, or can be set during the string interpretation by module `@Tx(index)` where parameter `index` specifies the texture index. The value of index can be equal to 0 (texturing is switched off) or to a number 1,2,3,... corresponding to the first, second, third, etc., texture as specified in the view file (see Appendix B.2). The first option can be useful in case when a given surface, for example a leaf, has one fixed texture associated with it. The second option is used when a surface can have more than one texture. For example, there is a set of textures defined for a leaf surface and for each instance of the surface, the texture is randomly selected from the set.

Another issue is how to map the texture on the surface, or how the object coordinates (x, y, z) are transformed into the (u, v) coordinates of texels, pixels of the texture image. For predefined bicubic surfaces, there are two ways texel coordinates are computed:

1. from s and t coordinates of each Bézier patch representing the surface (both s, and t varies from 0 to 1) (Figures 30b and 31a);

2. from x and y coordinates of the entire surface scaled to the interval $\langle 0, 1 \rangle$ (Figure 30c and 31b).

The texture pattern is often distorted as in Figure 31a or does not match with the surface shape (Figure 32b) and the texture image has to be warped. An example of an image with a venation pattern warped to fit a leaf surface is shown in Figure 32. The middle vein is moved to the left and the first two branched veins are adjusted to start from the leaf base.

Textures on developmental surfaces (defined within the L-system) can be set only during the string interpretation.

5.2 Textured cylinders and generalized cylinders

Textures on line segments rendered as cylinders and generalized cylinders are set during the string interpretation by symbol `@Tx(index)`. Parameter `index` specifies the texture index.

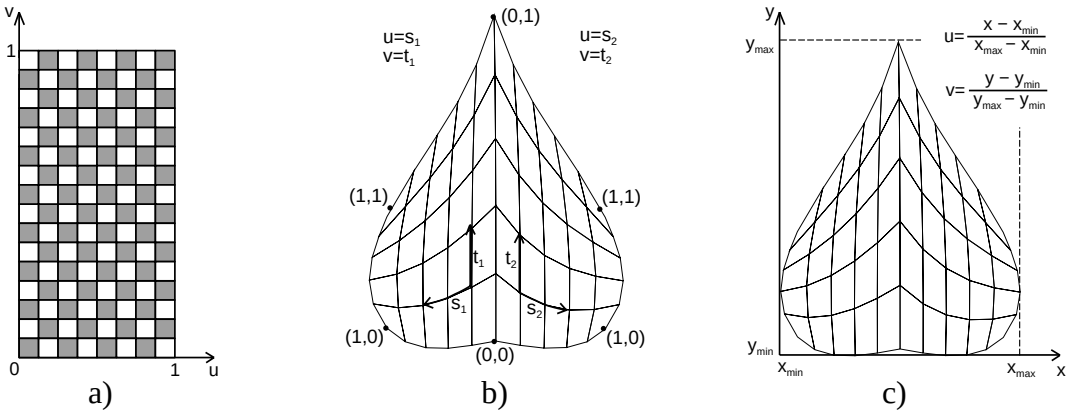


Figure 30: Transformation from object coordinates to texture coordinates (a): b) using parameters s and t of the Bézier formula, c) using point coordinates x and y.

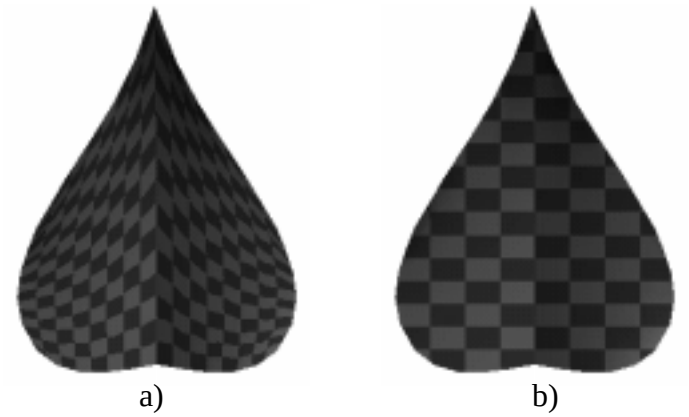


Figure 31: Chessboard texture on a surface: a) texture per patch, b) texture per surface.

Texel coordinate u ranges from 0 to 1 going around the segment circumference. The v coordinate increases along the segment in such a way that the aspect ratio of image pixels mapped on the surface is always 1. If the circumference of the base disk of a cylindrical segment is c and the segment length is l, the texel v coordinate is computed as:

$$v = \lfloor v_0 + lR_x/cR_y \rfloor,$$

where R_x and R_y are sizes of the texture image, and v_0 is the final texel coordinate from the previous line segments (Figure 33a). The very first line segments starts with $v_0 = 0$ and the last value of texel v coordinate for a segment is used as the starting value for the subsequent segment to keep the continuity of the texture mapping.

The left and right side of the texture image are aligned to each other and the image is repeated over and over along the stem. If the texture image is dark on the bottom and light

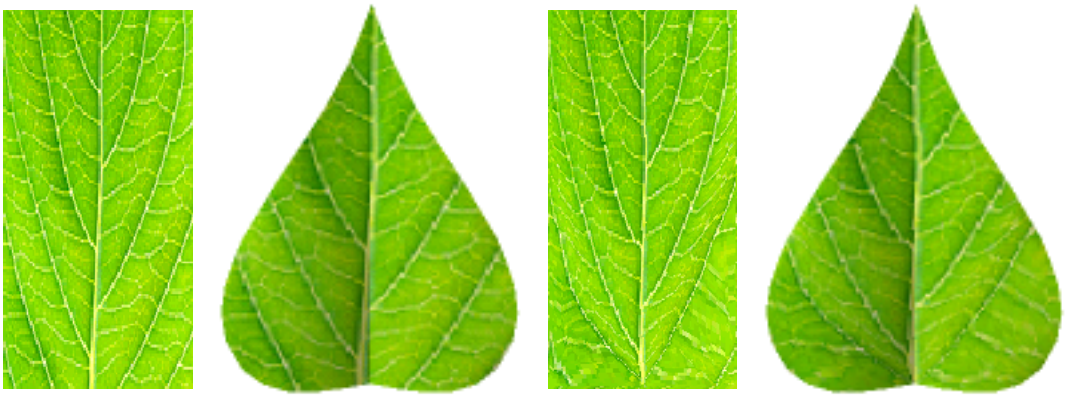


Figure 32: Original texture (a) on a leaf (b). Warped texture (b) on the same leaf (d).

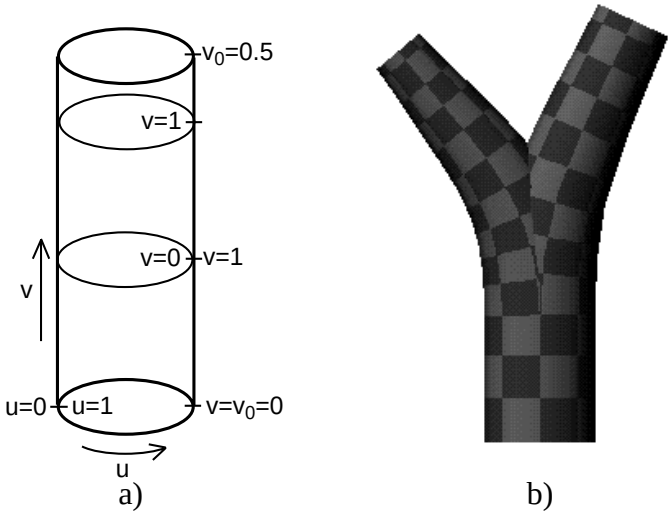


Figure 33: Mapping of a texture on a cylinder (a); textured branching generalized cylinder (b).

on the top, for example, this will cause a visible discontinuity in the color of the surface along the segment. Similarly, if the image is much darker on the left border than on the right one, a visible stripe along the segment can be created. It is often necessary to modify the texture image in such a way that the left border of the image matches the right border and the top matches the bottom.

An example of a texture mapped to a branching generalized cylinder is shown in Figure 33b. The discontinuity of the texture in the branching point is caused by the fact that branching generalized cylinders are created by overlapping two generalized cylinders fol-

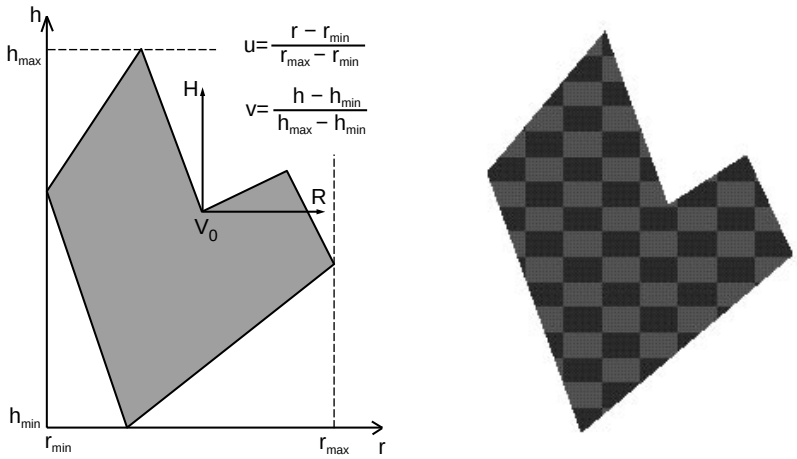


Figure 34: Mapping of a texture on a polygon (a); a textured polygon (b).

lowing each daughter branch.

5.3 Textured polygons

Textures on polygons are set in the string by the symbol `@Tx(index)` the same way as for surfaces, cylinders, or generalized cylinders.

Texture coordinates u, v are determined from coordinates h and r of polygon vertices. The axes h and r are defined by the turtle heading vector $\vec{H}$ and the right vector $\vec{R}$ (equal to the negative left vector $\vec{L}$) of the first polygon vertex V_0. Similarly as in the case of bicubic surfaces (Figure 30c), h and r coordinates are scaled to the interval $\langle 0, 1 \rangle$ to obtain texture coordinates u, v (see Figure 34a). An example of a textured polygon is shown in Figure 34b.

6 Response to directional stimuli

There are two categories of response to a directional stimulus. First, it is a mechanical response to a force pulling the plant organs, e.g. wind or the gravity force. Second, the plant can react actively by bending its organs away or towards the stimulus direction as in case of *tropisms*.

A tropism is a plant movement during which the differential growth on the opposite sides of a plant organ causes the organ to bend [18]. A tropism response is usually triggered by a directional influence of gravity (*gravitropism*) or light (called *heliotropism* or *phytotropism*). Imagine, for example, a root that tries to grow downwards from the horizontal position. The desired direction is achieved by faster growth of the upper side of the

root.

To simulate tropisms and the plant's response to an external force, Prusinkiewicz in [15] introduced a simple mechanism, which modifies the orientation of each line segment towards or away from the stimulus direction. The angle between the original and the new orientation is based on the angle between the line segment and the stimulus direction and a parameter controlling the susceptibility of the segment to bending. Based on the computation of torque acceleration, the angle β by which the segment with orientation $\vec{H}$ is rotated towards the stimulus direction $\vec{T}$ is:

$$\beta = e|\vec{H} \times \vec{T}|, \tag{1}$$

where e is the parameter of segment's susceptibility to the stimulus. If the parameter e is equal to 0, no adjustment is made. On the other hand, if it is too high (usually above 1, the adjusted segment may "overshoot" and bend too much). Negative values cause stems to be bent away from the stimulus direction. The response is bigger when the angle of the stem and the vector $\vec{T}$ is close to 90° and lower when the stem is almost aligned with the stimulus vector $\vec{T}$.

In case of tropisms, formula (1) can be extended to include more complex tropisms, such as a plagiotropism when branches try to be perpendicular to the vector of gravity or the general case, diatropism, when stems try to achieve an angle γ (not necessarily 90 degrees) with the tropism vector. If a segment with orientation $\vec{H}$ tries to reach angle γ with the tropism vector $\vec{T}$, the bending angle is (the full derivation is given in [10]):

$$\alpha = e\left(\cos(\gamma) - \sin(\gamma)\frac{\vec{H}\cdot\vec{T}}{|\vec{H}\times\vec{T}|}\right)(\vec{H}\times\vec{T}) \tag{2}$$

The adjustment of the segment's orientation takes place during the interpretation, when the L-system generated string is visualized, and the position and orientation of each interpreted module is known. During the simulation, the L-system model can only modify the elasticity parameter.

Each tropism is defined in the *view* file (see Appendix B) by its vector, an angle a shoot is trying to achieve with respect to the tropism vector, and two parameters: `initial elasticity` and `elasticity increment`. Initial elasticity specifies the susceptibility of a segment to the tropism. Its initial value can be modified in productions using special control symbols mentioned bellow:

`@Ts(`$index, value$`)` sets the elasticity of tropism with index $index$ to $value$. The parameter $index$ specifies the tropism according to the order of its specification in the *view* file.

`@Ti(`$index, value$`)` increments the elasticity of tropism with index $index$ by $value$.

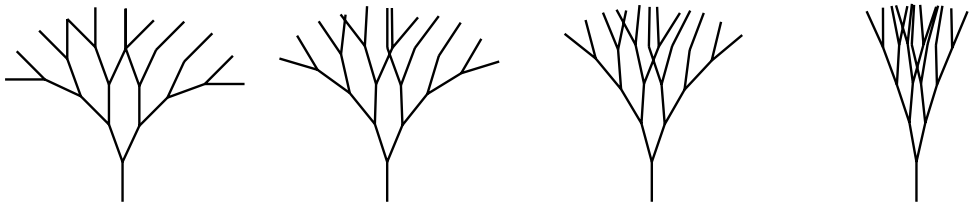


Figure 35: Orthotropism effect for elasticity values 0, 0.1, 0.25, and 0.5.

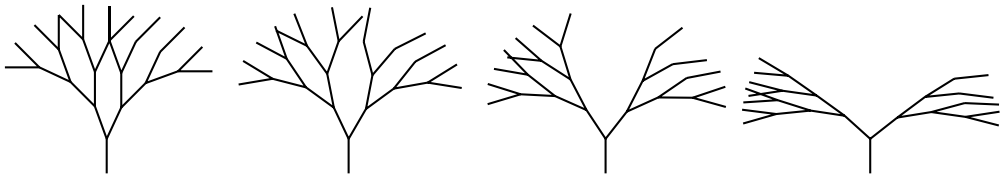


Figure 36: Plagiotropism effect for elasticity values 0, 0.1, 0.25, and 0.5.

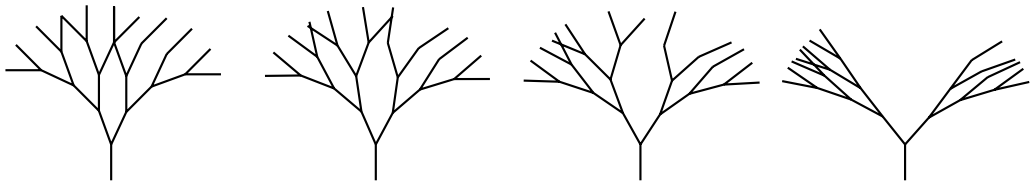


Figure 37: Diatropism effect for elasticity values 0, 0.1, 0.25, and 0.5.

`@Td(`$index, value$`)` decrements the elasticity of tropism with index $index$ by $value$.

Without a specified value, modules `@Ti` and `@Td` modify the elasticity using the predefined elasticity increment.

Figures 35, 36, and 37 illustrate the effect of an orthotropism ($\gamma = 0$), plagiotropism ($\gamma = 90°$), and diatropism ($\gamma = 60°$) on a simple tree-like structure. The tropism vector is (0,1,0) and the elasticity parameter varies from 0 (no adjustment) to 0.5.

Figure 38 illustrates the effects of gravity on a simple tree. The gravity is simulated by defining an orthotropism with direction $(0, -1, 0)$.

When a segment direction is changed, it is necessary to correctly compute the stem's up vector controlling the orientation of the unit with respect to its axis. The assumption made is that the sequence of segments behaves as a rubber tube; in other words, it tries to minimize any possible twist (see Section 3.3).

The twist minimizing adjustment of the turtle's up vector can eliminate rotation around the heading vector specified by symbols `/` and \. To force such a rotation a symbol `@Tf`

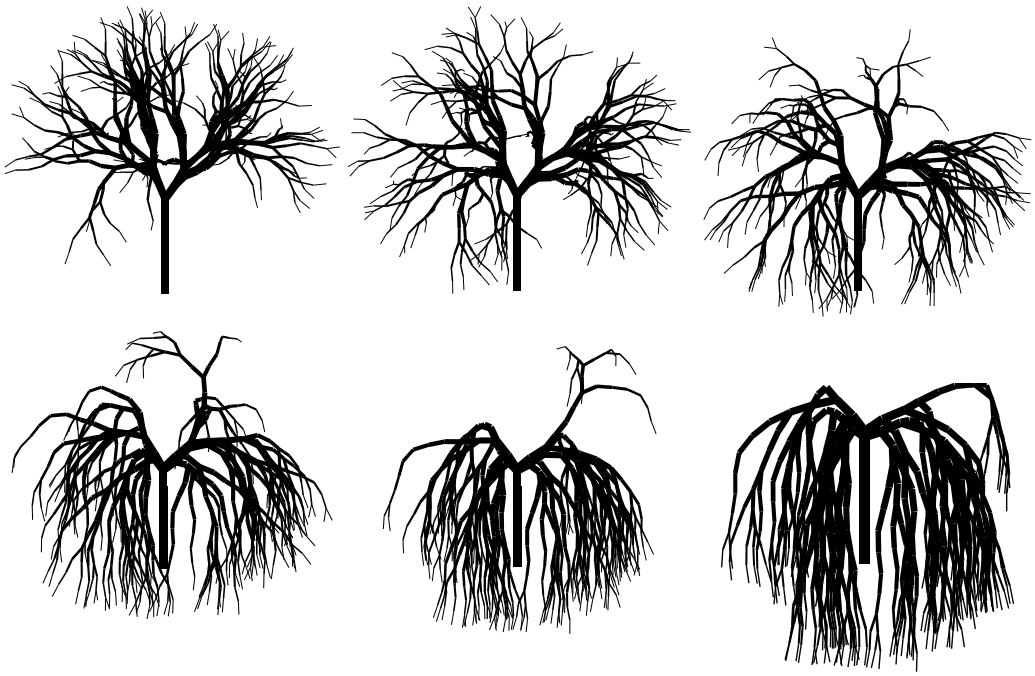


Figure 38: Effect of gravity for elasticity values 0, 0.1, 0.2, 0.3, 0.4, and 0.5

should be inserted after each rotation symbol. The orientation of the up vector after the forced rotation will then be considered as the base for a subsequent minimizing of the twist.

The mechanism of directional response allows the model to simulate basic plant responses to directional influences, such as stems growing towards or away from the light, or the main stem growing upwards and main root growing downwards. The next section introduces another new built-in mechanism twisting a plant segment as the response to a directional stimulus. This mechanism combined with tropism can simulate the proper orientation of leaves.

6.1 Response by twist

During the twist movement, the turtle up vector is rotated around the stem direction (turtle heading $\vec{H}$). The up vector is rotated towards a vector $\vec{V}$ specified as the projection of the tropism vector $\vec{T}$ onto the rotation plane (with normal $\vec{H}$):

$$\vec{V} = (\vec{H} \times \vec{T}) \times \vec{H}.$$

To obtain the rotation angle α the same formula as in the case of tropisms is applied with $\vec{T} = \vec{V}$:

$$\alpha = e|\vec{H} \times \vec{V}|$$

Figure 39: Twisting of leaf stalks using elasticity values 0, 0.1, 0.3, and 0.6.

After substituting for $\vec{V}$, it can be simplified to:

$$\alpha = e|\vec{T} \times \vec{H}|.$$

A twist is defined similarly as a tropism with only one difference: the angle parameters is not used (see Appendix B).

Twists and tropisms belong to the same group of environmental effects. Thus the value of the elasticity parameter can be changed using the same symbols $@Ts$, $@Ti$, and $@Td$ as for a tropism. The index specifying a tropism or twist is then the order of the tropism or twist specification in the view file.

Figure 39 illustrates the use of twist response on a twig with few leaves. A leaf stalk consists of three short line segments which are twisted so the turtle up vector at the end of the stalk points upwards making the leaf blade more exposed to the incoming light.

6.2 Combination of directional responses

The leaf twig from the previous example still does not look realistic, because the response is more complex than just the twist of the stalk. Generally, a leaf is trying to orient its blade perpendicularly to the direction of the incoming light. This can be achieved by combining the twist from the previous example and a plagiotropism trying to orient the stalk perpendicularly to the direction of light. In addition, the gravity force is pulling the leaf down.

Following L-system incorporates all three directional responses to obtain a a better orientation of leaves.

```
#define PE 0.0   /* perpendicular elasticity */
#define TE 0.6   /* twist elasticity */
#define GE 0.0   /* gravity elasticity */
#define LA 35    /* leaf angle */
#define Leaf   [@Ti(1,GE)@Ti(2,PE)@Ti(3,TE)\
                !(0.02)F(.12)F(.12)F(.12) l]
```

Figure 40: Adjusting leaf stalks using plagiotropism with elasticity values 0, 0.1, 0.3, and 0.6.

Figure 41: Adjusting leaf stalks using gravitropism with elasticity values 0, 0.1, 0.3, and 0.6.

$\omega:\quad -(25)/(90)FLA$
$p_1:\quad A \rightarrow /(90) - (20)FLA$
homomorphism
$h_1:\quad L \rightarrow [,(8)[\&(LA)Leaf]/(180)\&(LA)Leaf]$
The L-system creates three branch internodes with a pair of leaves at the end of each internode (using production p_1). The pair of leaves is visualized in homomorphism production h_1. Each leaf consists of a three-segment stalk and a bicubic surface defining the leaf blade. The stalk orientation is adjusted using three mechanism of directional response:

1. a plagiotropism with direction $(0, 1, 0)$ and angle $90°$;
2. a twist with direction $(0, 1, 0)$;
3. a gravitropism with direction $(0, -1, 0)$;

simulating effects of the light coming from the top and effects of the gravity force.

Figure 40 shows an increasing sensitivity of leaf stalks to the plagiotropism trying to make the blade axis perpendicular to the light direction $(0, 1, 0)$. Similarly, twigs in Figure 41 experience the effect of gravity pulling leaves down.

As can be seen in Figure 39 from the previous section and Figures 40 and 41, each of the three effects simulated separately does not result in a proper orientation of leaves. If all three mechanism are combined, as in Figure 42, the leaf orientation looks more realistic.

Figure 42: Adjusting leaf stalks using all three mechanisms with the same elasticity values 0, 0.1, 0.3, and 0.6.

A Interpreted symbols

During the visualization, the string of symbols is parsed from left to right and every time a special symbol controlling the turtle is encountered the function associated with the symbol is performed. Symbols with predefined interpretations are listed below.

Symbols with no parameters use default values specified at the beginning of the simulation. If a symbol has more parameters than those specified bellow, the additional parameters are ignored.

Turtle rotations
The turtle can be rotated only around its heading, left, or, up vector (Figure 43):

`+(`θ`)` Turn left by angle θ° around the U axis.

`-(`θ`)` Turn right by angle θ° around the U axis.

`&(`θ`)` Pitch down by angle θ° around the L axis.

$\wedge$`(`θ`)` Pitch up by angle θ° around the L axis.

`\(`θ`)` Roll left by angle θ° around the H axis.

`/(`θ`)` Roll right by angle θ° around the H axis.

`|` Turn around 180° around the U axis. This is equivalent to `+(180)` or `-(180)`. It does not roll or pitch the turtle.

`@v` roll the turtle around the H axis so that H and U lie in a common vertical plane with U closest to up.

If no parameter is given for the symbols `+`, `-`, `&`, $\wedge$, `\`, and `/`, the value of the global parameter `angle increment` is used.

Changing turtle parameters
The following symbols change turtle parameters:

`;(`n`,`$n2$`)` Increase the value of the current color index or material index by the `color increment`, or set to n if a parameter is given. If two-sided materials are used – the initial color index in the *view* file has two parameters (only in material mode) – both indexes of the front and back material are increased or set to n. If an optional second parameter is present, the index of the back material is set to $n2$.

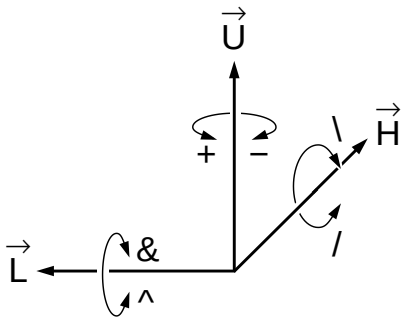


Figure 43: Controlling the turtle in three dimensions

`,(`n`,`$n2$`)` Decrease the value of the current color index or material by the `color increment`, or set to n if a parameter is given. In case of two-sided materials, the function is the same as for module ;.

`#(`n`)` Increase the value of the current line width by the global parameter `line width increment`, or set to n if a parameter is given.

`!(`n`)` Decrease the value of the current line width by the global parameter `line width increment`, or set to n if a parameter is given.

`@Tx(`$index$`)` Sets texture with index $index$ (the order of the texture specification in the view file). Index 0 switches off texturing. If a predefined bicubic surface has associated a texture index in the view file, its texture is fixed and cannot be changed by module $@Tx$.

Changing position and drawing

`F(`d`)` Move forward a step of length d and draw a line segment from the original position to the new position of the turtle. If the polygon flag is on (see the symbols { and }), the final position is recorded as a vertex of the current polygon. If no parameter is given, the default step size 1 is used.

`f(`d`)` Move forward a step of length d without drawing a line. If the polygon flag is on, the final position is recorded as a vertex of the current polygon. If no parameter is given, the default step size 1 is used.

`G(`d`)` Move forward a step of length d and draw a line. If no parameter is given, the default step size 1 is used.

`g(`d`)` Move forward a step of length d without drawing a line. If no parameter is given, the default step size 1 is used.

`@o(`d`)` draw a circle of diameter d in the plane of the screen. If no parameter is given, the current line width will be used.

`@c(`d`)` draw a circle of diameter d in the HL plane. If no parameter is given, the current line width will be used.

`@O(`d`)` draw a sphere of diameter d. If no parameter is given, the current line width will be used. The spheres produced can be shaded even in the colormap mode, since a set of polygons approximating a sphere is generated using code from the widely available `sphere.c` file by Jon Leech (`leech@cs.unc.edu`).

The global parameter `line style` specifies whether the line is drawn as a line, polygon, or a cylinder.

Modeling of structures with branches

`[` Push the current state of the turtle (all its parameters) onto a pushdown stack.

`]` Pop a state from the stack and make it the current state of the turtle.

`%` The symbol `%` cuts the remainder of a branch. Whenever it is detected in the string during the generation process, it and all following symbols up to the closest unmatched right bracket `]` are ignored for derivation purposes, and will therefore disappear from the generated string. If an unmatched right bracket is not found, symbols are ignored until the end of the string.

Symbols used to create polygons along with `F` and `f`

{ start a new polygon by pushing the current turtle position onto the polygon stack and set the polygon flag on.

} Pop a polygon from the stack and render it. If no more polygons are on the stack, turn the polygon flag off.

`.` Place the current state of the turtle on the polygon stack if the polygon flag is on.

Drawing parametric bicubic surfaces

$\sim$ Draw the predefined surface identified by the symbol immediately following the $\sim$ at the turtle's current location and orientation. The control points, geometry and neighborhood information for surfaces are read from surface specification files at the beginning of the simulation.

`@PS(`*i,basis*`)` initializes the four rows and columns of control points for an L-system defined surface i to $(0, 0, 0)$. The optional parameter *basis* specifies the type of patch as:

1. Bézier,
2. B-spline,
3. Cardinal spline.

If no basis is given, the default, Bézier, is used.

`@PC(`*i,r,c*`)` assigns the current position of the turtle to the control point of the L-system defined surface i in row r and column c.

`@PD(`*i,s,t*`)` draws the surface defined by the control points of surface i using s lines along the rows and t lines along the columns.

Drawing generalized cylinders

`@Gs` Start a generalized cylinder in the current turtle position.

`@Gc(`$strips$`)` specifies a control point on the central line of the generalized cylinder. The value of $strips$ specifies how many mesh strips are drawn between the control point and the previous one. The more strips are drawn the smoother the generalized cylinder looks. If no parameter is given, one strip is drawn.

`@Ge(`$rings$`)` End a generalized cylinder. The parameter $strips$ controls the number of strips as for symbol `@Gc`.

`@Gr(angle1,length1,angle1,length2)` specifies the slope and length of two tangents of a Hermite curve defining the radius change as a longitudinal section between two consecutive control points of a generalized cylinder axis (see Section 3.2.2). As a default, the radii at the two control points are linearly interpolated along the segment.

`@Gr(flag)` switches on (`flag`=1) or off (`flag`=0) an automatic adjustment of tangents of a longitudinal section for segments of non-unit length. The longitudinal section is always defined for a segment of a unit length and then stretched onto the segment of a non-unit length. As a default, tangents are not adjusted after the stretching (see Section 3.2.2 for more details).

`@#(`$contour\_id$`)` sets a different contour for the generalized cylinder. Contours are specified in the view file. A contour with `id` 0 is the default circle.

Changing tropisms parameters

`@Ts(`$index$`,`$value$`)` Set elasticity parameter of tropism with index $index$ to $value$. Index is given by the order of the tropism specification in the view file (starting with 1).

`@Td(`$index$`[,`$value$`])` Decrease the elasticity parameter by the default elasticity increment specified in the view file or by the given value $value$.

`@Ti(`$index$`[,`$value$`])` Increase the elasticity parameter by the default elasticity increment specified in the view file or by the given value $value$.

`@Tp` Prevent twist. This command adjusts the turtle's up and left vector to minimize the twist (see Section 3.3).

`@Tf` Force the twist. Since tropisms automatically force twist prevention, the effect of symbols `/` or \ can be nullified. It is necessary then to add the symbol `@Tf` to force the twist.

Symbols for Sub-L-systems

`?(`*id,scale*`)` Causes the generator to save a reference to the current L-system on a stack and to use the list of productions from the sub-L-system identified by *id* during subsequent production matching and application. During interpretation, the current scale is saved on a stack and the structure resulting from interpretation of the generated substring is scaled by *scale*.

`$` End the sub-L-system and return to the previous set of productions and scale.

Miscelaneous commands

`@L("Label")` prints the "label" in the drawing window at the current turtle location using the font specified in the view file.

`@S("any system call")` will make the system call when interpreted.

B View file commands

The graphical extensions described in the text include not only new modules with a specific interpretation but also several view file commands setting initial parameters of the graphical interpretation:

`contour:` allows the user to specify the cross-section of a generalized cylinder as a open or closed Bézier spline (see below for more details);

`contour sides:` defines the number of polygons drawn around a contour;

`twist of cylinders:` switches on or off the twist minimizing method applied during the visualization of generalized cylinders;

`texture:` specifies the texture and the way it is mapped on a surface.

The details of specification of contours and textures are given in the following sections.

B.1 Specification of contours

Contours are specified in the view file by the command:

```
contour: id contour_file
```

where `id` is a unique positive number identifying the contour and `contour_file` is the name of a text file containing a list of coordinates of control points.

The number of polygons around a contour can be also set in the view file by command:

```
contour sides: n
```

where `n` specifies the number of polygons. Currently, this number is constant during the interpretation.

The contour file has the following syntax: the first line contains a number of control points, the dimension of the contour (2 or 3), and an identifier of an open or closed contour (the word `open` or `closed`). Subsequent lines contain two or three coordinates of control points, one point per line.

The contour in Figure 12a is specified by file:

```
12 2 closed
0.166482 -1.123751
0.416204 -1.040511
0.582686 -0.332963
1.082131 -0.041620
```

```
1.082131 0.499445
0.499445 0.541065
0.332963 0.915649
-0.374584 1.040511
-0.707547 0.624306
-1.123751 0.166482
-0.874029 -0.749168
-0.416204 -0.665927
```

The contour in Figure 13a is a three-dimensional extension of the previous contour:

```
12 3 closed
0.166482 -1.123751 1.0
0.416204 -1.040511 0.0
0.582686 -0.332963 0.0
1.082131 -0.041620 0.0
1.082131 0.499445 0.0
0.499445 0.541065 0.0
0.332963 0.915649 0.0
-0.374584 1.040511 0.0
-0.707547 0.624306 0.0
-1.123751 0.166482 0.0
-0.874029 -0.749168 0.0
-0.416204 -0.665927 0.0
```

The open contour in Figure 16a is defined by file:

```
15 2 open
-0.914286 -0.400000
-0.914286 -0.400000
-0.914286 -0.400000
-0.871429 -0.342857
-0.742857 -0.171429
-0.457143 -0.085714
-0.285714 0.142857
0.000000 0.271429
0.228571 0.114286
0.514286 0.142857
0.657143 -0.028571
0.892857 -0.114286
0.971429 -0.142857
```

```
0.971429 -0.142857
0.971429 -0.142857
```

Note that the first and the last control point is repeated three times to make sure the contour starts and finishes in the given points.

It is possible to create a two-dimensional contour in a drawing program `xfig` and convert it to the contour text file using a conversion utility `fig2con`. It is necessary to use either open or closed Bézier spline for the contour and to define a reference circle specifying the origin and scale of the contour coordinates. The created *xfig* file can be converted using the command:

```
fig2con <contour.xfig >contour.spec
```

Currently, no utility for designing three-dimensional contours is provided. One possibility is to create a two-dimensional contour in `xfig` and add the third coordinates by editing the contour specification file.

B.2 Definition of textures

Textures are specified in the view file using a command `texture`:

texture: F: image_name H: mag_filter L: min_filter E: env_mode S:

where

- the parameter `image_name` specifies the texture image. Currently, it is possible to specify iris *rgb*, Utah raster toolkit *rle*, and targa *tga* images (distinguished by the extension). The image resolution can be arbitrary.
- commands `H`, `L`, `S`, and `E` are optional.
- command `H: mag_filter` is used in case texture pixels viewed on the screen are bigger than one window pixel. The flag `mag_filter` is equal to:
  - *linear* (or *l* only) — the texture image is smoothened when mapped onto the surface;
  - *near* (or *n* only) — the texture image is not smoothened, thus texture pixels can appear as big squares.

  The default is *near*.
- command `L: min_filter` is used in case texture pixels viewed on the screen are smaller than one window pixel. The flag `min_filter` is equal to:

  - *linear* (or *l* only) — more texture pixels are used to compute the color of a given pixel on the screen;
  - *near* (or *n* only) — just one texture pixel is used to compute the color of a given pixel on the screen. This mode is faster but can produce some aliasing effects.

  The default is *near*.

- command `E: env_mode` controls the way the texture is combined with the surface color (see The OpenGL Programming Guide, chapter 9, section Modulating and Blending). The flag `env_modes` is equal to:

  - *modulate* (or *m* only) — the diffuse color of the surface material is multiplied by the color of the texture pixel;
  - *decal* (or *d* only) — the color of the texture pixel is used as the color of the surface;

  The default is *modulate*.

- the command `S:` is recognized only if the texture is used on bicubic surfaces predefined in the view file. If the command is present in the texture specification, the texture image is mapped onto the whole surface. Otherwise, the texture image is mapped onto each surface patch separately.

  As a default, the texture is mapped onto each patch. In this case, texture coordinates are derived from s and t coordinates of the parametric equation of the Bézier patch representing the surface (both s, and t goes from 0 to 1). In the case of mapping onto the whole surface, the surface boundaries along x and y axes are found and the texture is mapped into z-plane so that it fits the surface bounding box in x and y coordinates (see Section 5.1).

B.3 Definition of tropisms and twists

Tropisms and twists are specified in the view file using commands `tropism` or `torque`:

tropism: T: vector A: angle E: initial_elasticity S: elasticity_increment
torque: T: vector E: initial_elasticity S: elasticity_increment

where

- parameter `vector` specifies x, y, and z coordinates of the tropism vector. This parameter has to be present.
- parameter `angle` specifies the diatropism angle. The default value is 0.

- parameter `initial elasticity` defines the initial value of the elasticity parameter (if not present, the initial value is 0).
- parameter `elasticity increment` defines the value of elasticity increment used by modules `@Ti` and `@Td`.

References

[1] R. Bartels, J. Beatty, and B. Barsky, editors. *An introduction to splines for use in computer graphics and geometric modeling*. Morgan Kaufman, Los Altos, California, 1987.

[2] J. Bloomenthal. Modeling the mighty maple. *Computer Graphics (SIGGRAPH '94 Conference Proceedings)*, 19(3):305–311, July 1985.

[3] J. Bloomenthal. *Skeletal Design of Natural Forms*. PhD thesis, University of Calgary, Calgary, Alberta, Canada, Jan 1995.

[4] H. S. M. Coxeter. *Introduction to Geometry*. J. Wiley & Sons, New York, 1961.

[5] I. D. Faux and M. J. Pratt. *Computational Geometry for Design and Manufacture*. Ellis Horwood, Chichester, 1979.

[6] J. D. Foley and A. Van Dam. *Fundamentals of interactive computer graphics*. Addison-Wesley, Reading, 1982.

[7] J. D. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer graphics: Principles and practice*. Addison-Wesley, Reading, 1990.

[8] J. S. Hanan. PLANTWORKS: A software system for realistic plant modelling. Master's thesis, University of Regina, 1988.

[9] J. S. Hanan. *Parametric L-systems and their application to the modelling and visualization of plants*. PhD thesis, University of Regina, June 1992.

[10] R. Měch. Tropisms in Lindenmayer systems. *Unpublished manuscript*.

[11] P. Prusinkiewicz. Graphical applications of L-systems. In *Proceedings of Graphics Interface '86 — Vision Interface '86*, pages 247–253, 1986.

[12] P. Prusinkiewicz. Applications of L-systems to computer imagery. In H. Ehrig, M. Nagl, A. Rosenfeld, and G. Rozenberg, editors, *Graph grammars and their application to computer science; Third International Workshop*, pages 534–548. Springer-Verlag, Berlin, 1987. Lecture Notes in Computer Science 291.

[13] P. Prusinkiewicz and M. S. Hammel. Continuous Animation Using L-systems, Video Tape, University of Calgary, 1991.

[14] P. Prusinkiewicz and J. Hanan. *Lindenmayer systems, fractals, and plants*, volume 79 of *Lecture Notes in Biomathematics*. Springer-Verlag, Berlin, 1989 (second printing 1992).

[15] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990.

[16] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990 (second printing 1996). With J. S. Hanan, F. D. Fracchia, D. R. Fowler, M. J. M. de Boer, and L. Mercer.

[17] W Sierpiński. Sur une nouvelle courbe qui remplit tout une aire plaine. *Bull. Acad. Sci. Cracovie, Série A*, pages 462–478, 1912. Reprinted in W. Sierpiński, *Oeuvres choisies*, S. Hartman et al., editors, pages 52–66, PWN – Éditions Scientifiques de Pologne, Warsaw, 1975.

[18] E. Strasburger, F. Noll, H. Schenck, and G. Karsten. *A Text-Book of Botany*. MacMillan and Co. Limited, London, 1908.