CPSC589 - Modeling for Computer Graphics

Final Project

Sketch-Based Flower Modeller

Shannon Halbert
Beverley Rogers
Yanina Zholudz

Overview

We have created a 3D sketch-based flower modeller. Our application builds 3D models of flowers by creating and connecting floral components. Users can create and manipulate petals, leaves, stems and blooms and combine these to create single/multiple bloom structures.

Leaves and petals are created using ruled surfaces defined by two sketched curves inputted by the user. These surfaces are then deformed by sketching new cross-section curves.

A stem is created as a rotational blending surface based on a single sketched curve input by the user. This curve acts as the spine of the stem. The blend curves are created by offsetting the spine in two directions.

The overall flower structure is created by attaching leaves, petals, blooms and stems to one central root stem.

UI & Program Usage

UI Breakdown

Our user interface has three main sections: the sketch window, the display list, and the bloom structure windows. All sketching by the user is done in the large sketch window. When the user saves a 3D *FlowerPart* (either a *Surface* or a *Bloom*) for later use when constructing a flower, the *FlowerPart* is added to the display list at the top right corner where all previously-saved *FlowerParts* are displayed. Since the program works in creating one flower, *Stems* are not added to the display list. The two windows at the bottom right of the screen are the bloom structure windows. The middle window allows the user to manipulate the structure of the bloom, while the bottom window shows a preview of the bloom structure in 3D.

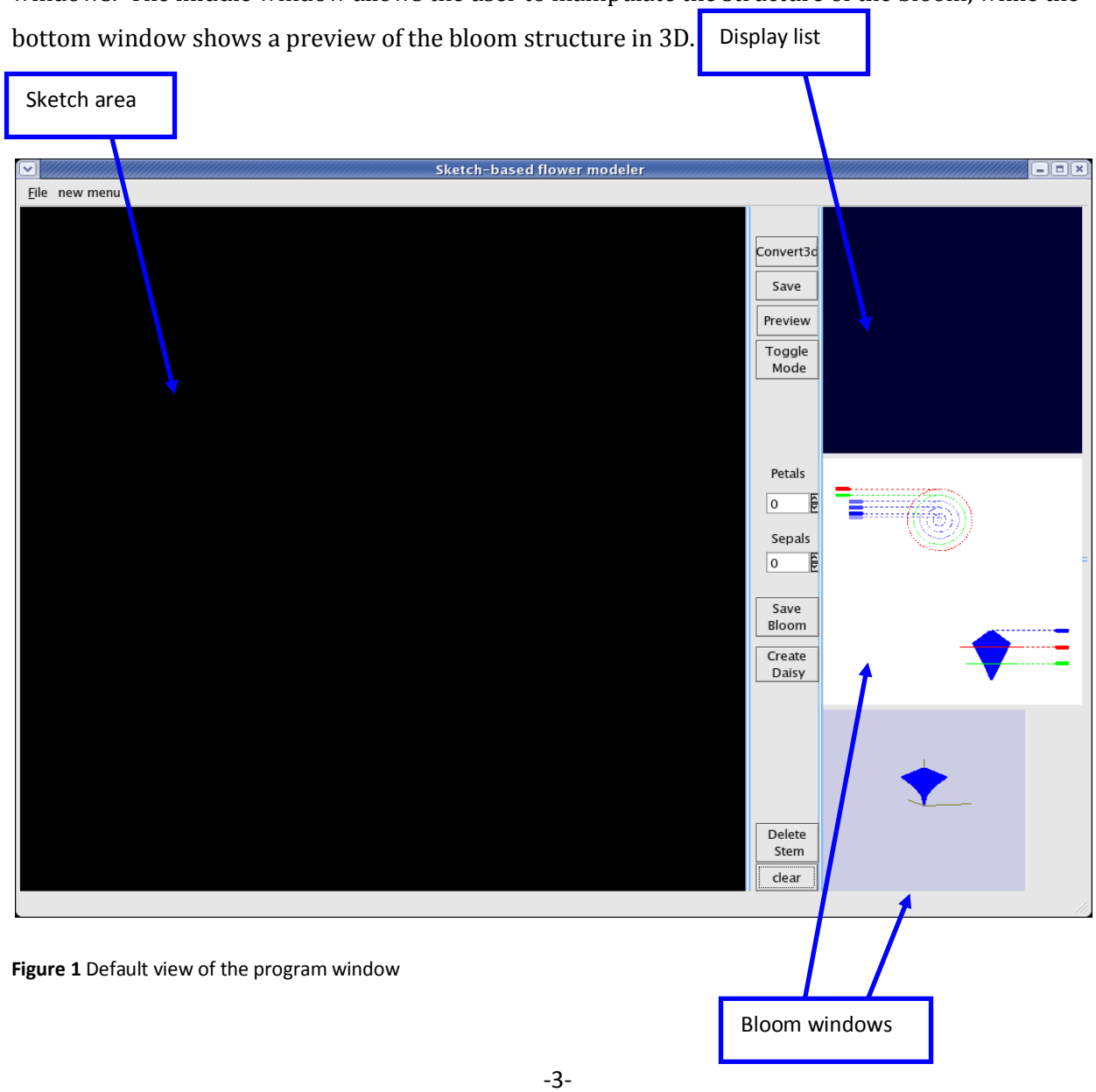


Figure 1 Default view of the program window

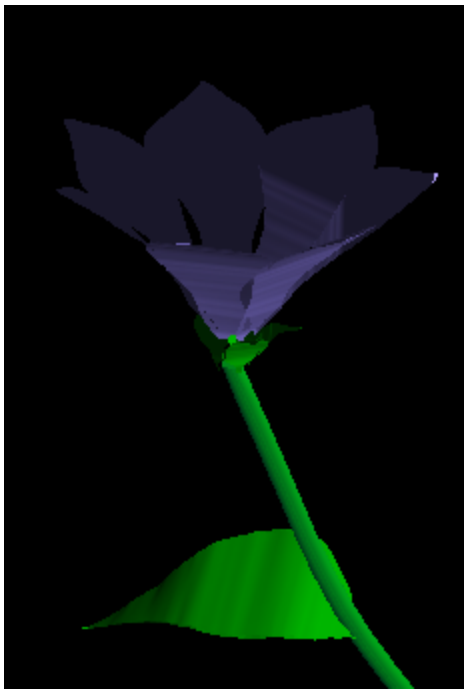
Using the Program

Drawing Surfaces

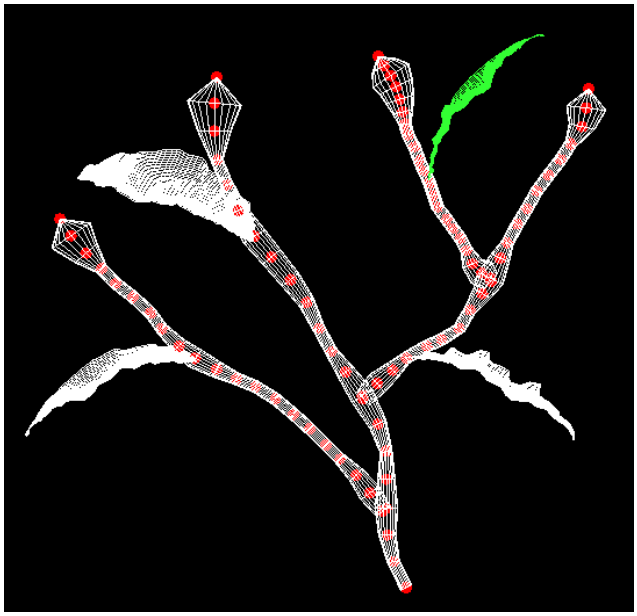
- The default mode of the sketch window is Surface drawing. The user needs to draw two curves on the screen by clicking the left mouse button and dragging the mouse to produce a surface. To produce proper normals (for rendering), each of the curves needs to be drawn from top to bottom.
- At any time, if the user does not like the curves he has drawn, he can click on the clear button and begin drawing again.
- Once the curves have been drawn, clicking on the "Convert 3D" button will convert the curves into a mesh.
- The user can now rotate the camera around the surface by right-clicking and dragging the mouse.
- At any time after the mesh has been created, the user can press the "Preview" button to view the rendered surface. Click the "Preview" button again to go back to wireframe mode.
- To update the cross-section of the curve, the mesh should be rotated by dragging the mouse in the upward direction. The user can then draw a new curve along the cross-section and the surface will automatically update.
- To update the length of the curve, the mesh should be rotated from the cross-sectional position by dragging the mouse to the left. If the mesh is rotated in the other direction, the mesh will respond in reverse to the manipulation curve.
- Both the cross-section and length can be updated multiple times by drawing new manipulation curves.
- When the user is happy with the surface, he can click on the "Save" button to add the Surface to the display list.
- The user can now draw new curves to add more surfaces by repeating the above steps.
- Please note that the first 3 Surfaces will be green and the next 3 will be red. This pattern will repeat itself for subsequent Surfaces.

Creating Blooms

- To create a bloom the user must first create one or more leaves, petals and sepals – see drawing a surface above for information on how to do that.
- The middle window on the right of the form shows the bloom structure and allows the user to change the default settings (more on these parameters below) of the bloom as well as add required petals and/or sepals.
- This window is split into two parts with the circular diagram representing a view of the bloom from above and the other diagram representing a side view of the bloom.
- To add petals (sepals), the user can use the petals (sepals) spin box to set the number of petals (sepals). The bloom diagram will show red (green) triangles where the petals (sepals) will be attached to the bloom. By clicking the appropriate triangle and then the desired leaf/petal in the display list, the petals (sepals) can be added to the bloom.
- At any point the user can preview the bloom in the 3D window.
- When the user is satisfied with the bloom he can click the "Save Bloom" button and a *Bloom* object will be saved to the display list. After a *Bloom* is saved, the circle diagram is reinitialized to the default values.



Creating Stems

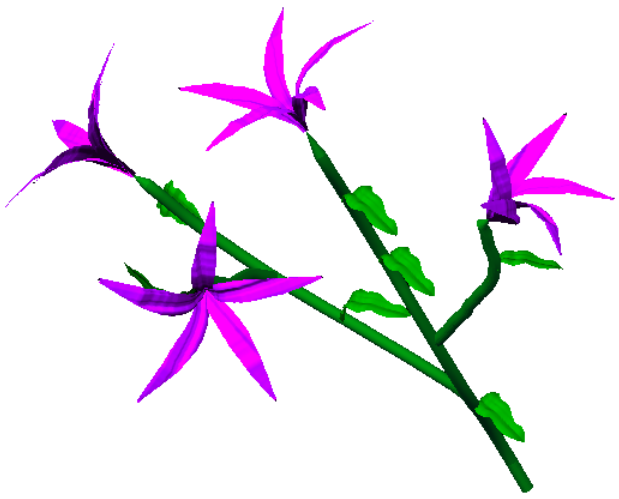


- Once the user has finished creating surface components (leaves and petals) for their flower, they can move into stem creation mode by clicking on the "toggle mode" button. In this mode; users draw stems and attach flower parts together to finalise their flower.
- Once in stem creation mode, users must draw an initial root stem. A stem is drawn by clicking and dragging with the left mouse button.
- Once the stem is drawn, it can be manipulated in several ways:
    - By clicking and dragging the right mouse button on the red circles within the stem (these represent the stem control points) users can move the stem around.
      Note: moving one point will also move every point on the stem that was drawn after that point and also any attached *FlowerParts*. This enables the user to move an entire branch of the stem without disturbing its shape by moving its root point.
    - Right clicking once on a control point will select that point. The stem containing the selected point will turn blue. Once a point is selected, users can change the radius of the stem at that point using the left and right arrow keys.
      Note: by default (if another point is not explicitly selected) the last point in the last drawn stem is selected.
    - A branch can be created from any of the red control points. To draw a branch, click and drag with the left mouse button starting on the control point that you want the branch to be connected to.
    - A stem can be deleted by first selecting any control point along it, then pressing the "delete stem" button. All *FlowerParts* branching off the deleted stem will also be deleted.
    - A flower part can be connected to any control point within the stem structure. (see assembling flowers)
- Thus far, all stem manipulation has been in2D. Once you are happy with your 2D stem structure click the "convert 3D" button to move into stem rotation mode. Once in rotation mode manipulation is done on entire branches rather than individual control points. Right click on a branch to select it (it will turn yellow when selected). You can now rotate the selected branch around its connection point using the up and down arrow keys.
- To view the stem structure from different angles, click and drag the right mouse button to rotate the camera.

- At any stage users can preview the rendered stem by clicking the “preview” button. Clicking ‘preview’ again will return the stem to wireframe mode.

Assembling Flowers

- First create all leaves and petals that you will need to create your flower. Save these in the display list in the top right corner of the ui. (You can move back to surface mode from stem creation mode by clicking ‘toggle mode’, but any stems you have created will be lost).
- Next move into stem creation mode and draw your root stem.
- From here you are free to connect components however you please.
- To connect a component to a stem; right click to select the control point to connect to, then left click on the component (in the display list window) to connect.
- If you connected a leaf or petal directly onto a stem it can now be manipulated in several ways:
    - To select the leaf/petal right click on it, it will turn green when selected.
      note: since selection depends on gluUnproject, it can be a little unpredictable
    - Once selected the leaf/petal can be rotated up and down with the up and down arrow keys or tilted with the left and right arrow keys.
    - The leaf/petal can also be turned into a *Daisy*. A *Daisy* is a basic flower created by duplicating and rotating the same leaf/petal around a stem control point. To create a daisy from a selected petal/leaf, choose the number of petals required using the scroll box in the bloom creation window and click the “create daisy” button.
- Once all desired components are connected, click the “convert 3D” button to move into rotation mode, from there you can rotate branches (as outlined in stem creation above) to 3-dimensionalize your flower.
- Click the “preview” button to see your rendered flower.
- Click and drag with the right mouse button to rotate your view.



Implementation Details

Introduction

This section explains the details of the backend of our system. It is comprised of three main classes: *Bloom*, *Stem*, and *Surface*. *Blooms* are composed of a collection of *Surfaces* and an ovary. A Stem is our high-level object that owns all *FlowerParts* attached to it. Surfaces represent both leaves and petals. Our controller classes, *GlSketch*, *Displaylist*, *CircleDiagram* and *StemSurface* handle all user input in the different widgets of the main program, delegating Qt commands to our class objects.

Surface

A Surface represents either a leaf or a petal. It is comprised of a mesh created through two user strokes. The mesh can be manipulated by user strokes along the cross-section or the length of the Surface.

Composition

Control curves

A Surface is initially created when the user draws two strokes in a downward motion, represented by quadratic B-splines of the Curve class. Once the user is happy with his curves, he can click on the "Convert to 3D" button which will create the mesh of points, surface. Although the curves are stored, they are not used after this point. When the surface is updated, the mesh itself is updated and not the initial curves. This is because the curves cannot represent the surface after the user has manipulated it with additional strokes because of the minute changes that can be made to the surface.

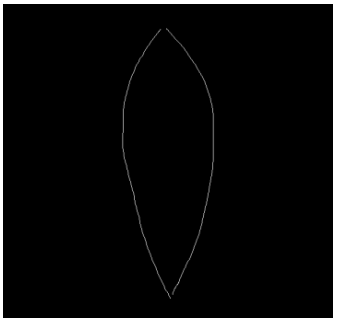


Figure 2: The user-drawn control curves.

Surface mesh

The surface mesh is created by taking the same number of equally-spaced points along each of the control curves and linearly interpolating between the points to make a ruled surface. These points are stored in a 2D Vector. Initially, we used reverse Chaikin subdivision on the curves' control points but found it did not guarantee the same number of points on each of the curves which made drawing the mesh more difficult and less smooth. Using the equally-spaced points also allows us to change the resolution of the mesh by controlling the number of points generated along the B-splines. We have found that the mesh edges still represent the control curves very closely.

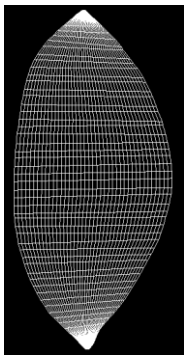


Figure 3: The initial surface mesh.

Secondary mesh

A secondary mesh, zDiff, is used to store the difference in the z-coordinates between a point on the mesh and the first Vector of points on the surface (surface[u][0]) as made by the cross-sectional manipulation stroke. This mesh is exactly the same size as the surface mesh but consists of only a 2D Vector of floats instead of Points as we only need to store the difference in the z direction. This was necessary so that manipulating surface along its length would not

overwrite any changes done to the cross-section of the mesh. The mesh is updated only when a cross-sectional stroke is drawn. The differences are then added to the z values of the mesh points to re-apply the cross-sectional stroke whenever a manipulation curve is drawn along the length of the surface.

Manipulation

A surface is manipulated when the user draws a stroke on the sketch area after converting the control curves into the surface mesh. The user must rotate his view of the surface so that he can draw a line along the length of the surface or the across the surface depending on how he wants the surface to deform. If the view is not rotated sufficiently, unfavourable results may occur. The mesh may update along the direction the user did not intend to update. The program looks at the end points of the curve and if the difference in the x-coordinates in more than the difference in the y-coordinates of the two points then the cross-section is updated, otherwise the length is updated.

Cross-Section

If the user rotates the view and draws a curve along the cross-section of the surface, the z-difference mesh described above will be updated as will the surface mesh. We chose not to scale the surface to the curve but instead to only use the part of the curve that intersects the mesh on the x plane. We found it more helpful to allow the user to have some leeway when drawing the curve to better draw the intended curve over the surface. For each row of the surface, the intersection of the curve with the boundary of the mesh is found by stepping through each control point of the manipulation curve until an intersection point is found. This is repeated from the end of the curve inward to find the ending intersection. Once the start and end points of the curve have been found, equally-spaced points are pulled from the manipulation curve using the algorithm described earlier when describing the surface mesh. The number of points used is the number of points on the row of the surface being updated (which is the same for each row of the mesh). This makes updating the z-difference mesh very simple as there is the same number of points on the curve as need to be updated in the mesh. Once the entire z-difference mesh has been updated, the z-values of the surface mesh are updated.

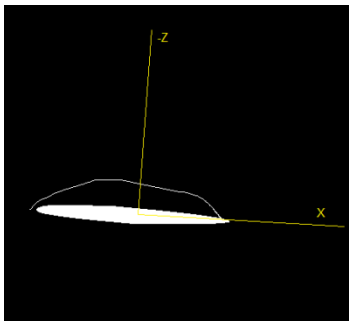


Figure 3: Manipulation curve along the surface cross-section.

Figure 4: Above surface after the cross-section has been updated.

Length

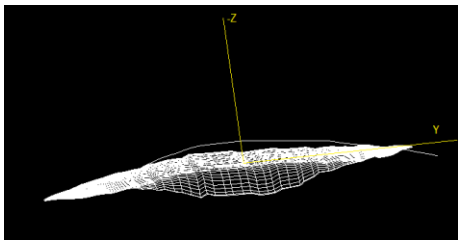


Figure 4: Manipulation curve along the length of the surface.

If the user rotates the view and draws a curve along the length of the surface, the first column of points (surface[u][0]) will be updated. The rest of the surface is updated by iterating over the surface and adding the value at surface[u][0] to zdiff[u][v]. This is done so the manipulation curve drawn along the length of the surface does not overwrite changes made to the cross-section. Again, the mesh is not scaled to the curve. Instead the intersection, in the case in the y-direction, is found for the first column of the mesh. If the surface is not rotated as specified in the "Program Usage" section, unpredictable results will occur. In hind sight, it may have been more intuitive for the user if we had instead used the manipulation curve to update the centre column of the mesh since it is roughly a straight line between the bottom and top points of the surface.

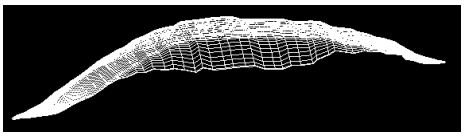
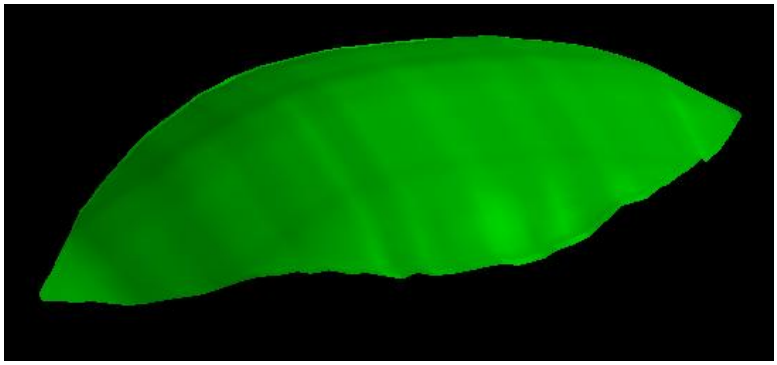
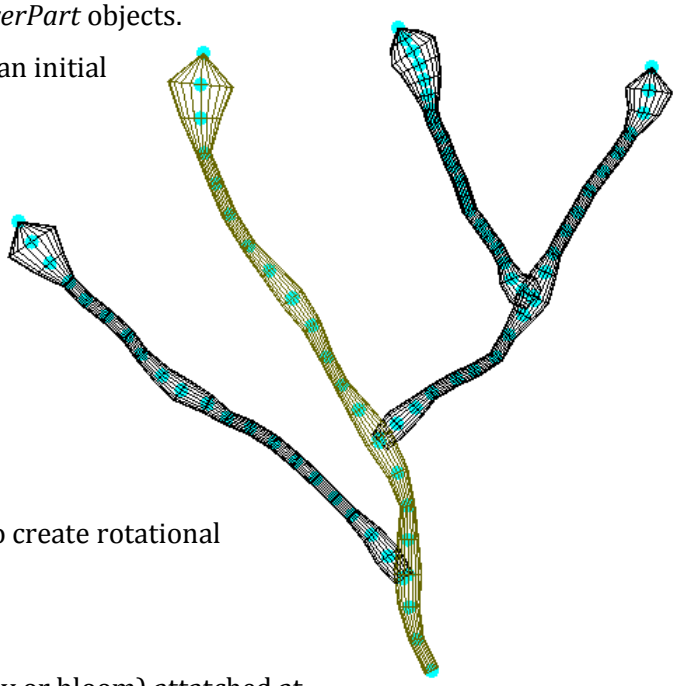


Figure 5: Above surface after the length has been updated.



Stem

A stem is stored as a recursive tree structure of *FlowerPart* objects. These can be *Stems, Leaves*, *Daisies* or *Blooms.* It has an initial base *Stem* containing 20 *Points* each of which can be manipulated to change the shape of the stem and can have a *FlowerPart* attached.



A stemPoint is defined as:

Point pos

- The point's position in 3D space

float radius

- The radius of the stem at that point (used to create rotational blended surface)

FlowerPart *branch

- A pointer to the FlowerPart (stem, leaf, daisy or bloom) attatched at that point

Stem creation and manipulation

The stems' surfaces are created as rotational blending surfaces. In order to create these rotational blending surfaces, all construction curves must be co-planar. We do however want to be able to create fully 3 dimensional flowers. To facilitate this, stem creation has two modes, ***draw mode*** and ***manip mode.***

2D – Draw Mode

In draw mode, users create the basic stem layout by sketching curves with the mouse. Mouse co-ordinates are projected onto the x-y plane and are used as control points to calculate a 3rd order B-Spline curve from which 20 evenly spaced ***stemPoints*** are created. Left and right curves are created by offsetting each stemPoint to the left and right by its current stored radius. The left and right curves are then used to create a rotational blending surface using the sketched curve as the central rotational axis.

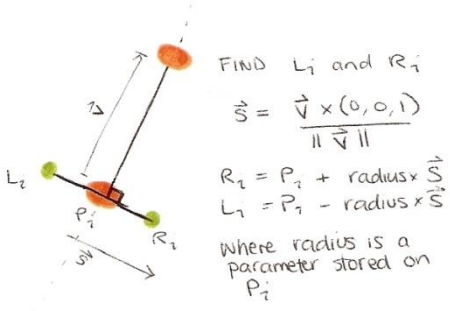


Figure 6: Finding left and right curves

Calculating the rotational blending surface

The stem surface is stored as a 2D vector of points created as follows:

For each stemPoint (S_i) in the stem, a circle of the desired radius is created centered on the origin. The frame transformation is then applied to the circle to orient it correctly with the stem. The new frame is:

((0, 0, 0), x, y, z) where

$$x = \frac{|L_i \, X \, R_i|}{|L_i||R_i|}$$

$$y = i \times (0, 0, 1)$$

$$z = (0, 0, 1)$$

8 points are taken around each circle and each is then translated by the vector

$S_i - (0, 0, 0)$

to bring it back to the correct height on the stem.

This disc of points is then added to the stem surface vector of points.

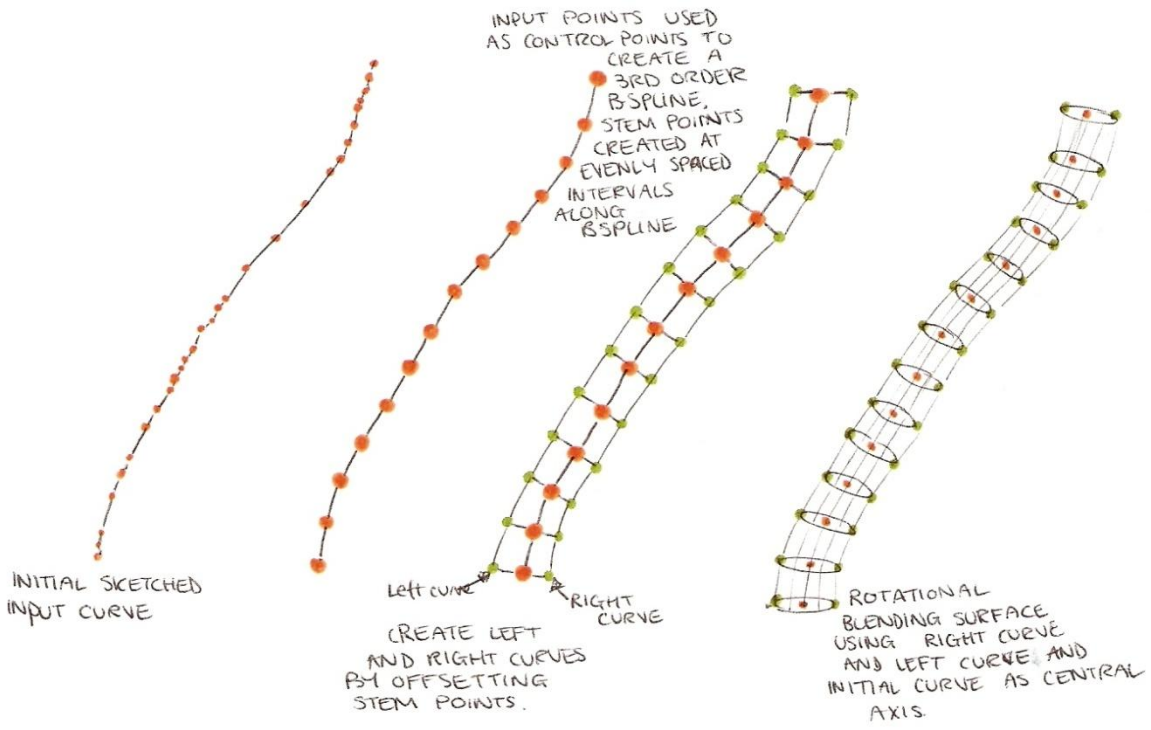


Figure 7: The stem creation process, from sketched points to stem surface

The radius of the stem at any given control point can be changed by selecting it and using the left and right arrow keys. The radii of neighbouring points along the stem are also changed slightly keep the stem relatively smooth. Whenever a radius is changed, the entire surface for the effected branch of the stem is recalculated.

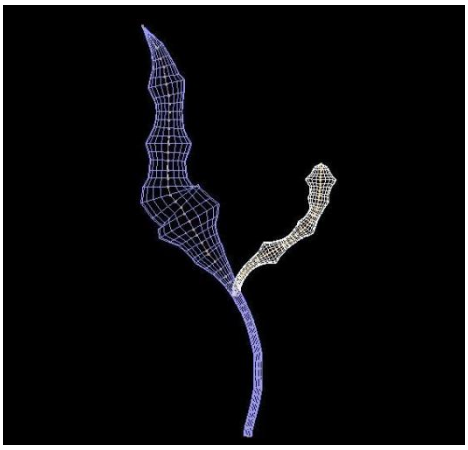


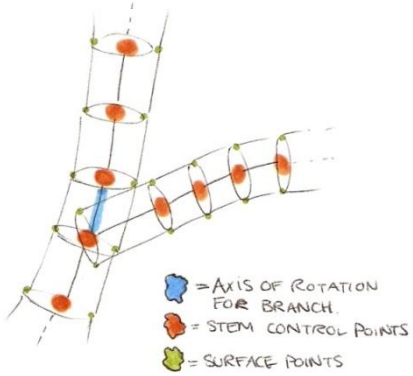
Figure 8: An example of a two branch stem

Once a root stem is created, branches can be added by drawing new stems starting at control points on the existing stem. When the user clicks the left mouse button, the control point clicked on is found and a new Stem object is created and stored as its branch pointer.

Branches can be moved around by right clicking and dragging on their control points. When a control point is moved, every point in the stem after that point is also moved (this includes all branches and flower parts attached). This is implemented by calling 'translate' on each point and on any non-NULL branch pointers in the selected branch.

3D – Stem rotations

To manipulate the stem in 3D, we move into manip mode by clicking the "convert 3D" button. From here manipulation involves moving entire branches rather than individual control points. A branch is selected by right clicking on it (it will turn yellow when selected). Selected branches can then be rotated around their parent stem using the up and down arrow keys.



The axis of rotation is taken to be the vector between the point on the parent stem at which the branch is connected, and the next stemPoint in the parent stem. All branches and flower parts hanging off the selected stem are also rotated.

> Note: Once you have moved into 3D manipulation mode, you cannot go back to draw mode.

Initially we had planned to have a set of pre-built stem structures for the user to choose from. They would then be able to perform some basic manipulations on the prebuilt forms in order to personalise their designs. We decided, however, that sketching stems was not only more fun but also more in keeping with the sketch based nature of our application. It also gave us the opportunity to experiment with rotational blending surfaces.

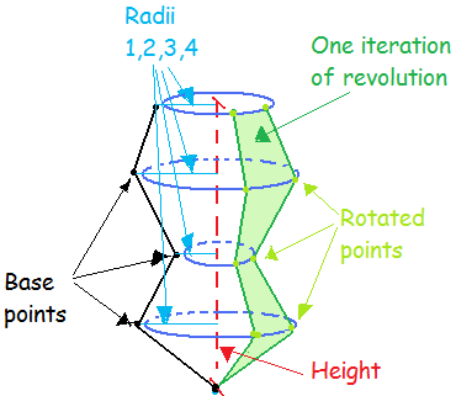
Bloom

A bloom consists of a surface of revolution representing an ovary and a collection of surfaces representing petals/sepals.

The manipulation of the bloom is done according to the paper by Takashi Ijiri, Makoto Okabe, Shigeru Owada, and Takeo Igarashi "Floral diagrams and inflorescences: Interactive flower modeling using botanical structural constraints" ACM Transactions on Computer Graphics, Vol.24, No.3, ACM SIGGRAPH 2005, Los Angeles, USA, 2005 (chapter 4.1 Floral diagram editor)

Composition

The ovary is constructed from five base points that are rotated around y-axis.

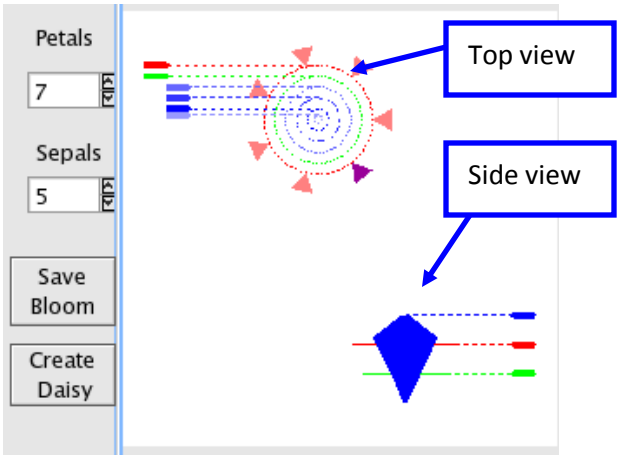


The base points are defined as follows:

- the first point is set to (0,0,0) in bloom coordinates
- the second point is (radius1, ¼ height, 0)
- third point – (radius2, ½ height, 0)
- fourth point – (radius3, ¾ height, 0)
- fifth point – (radius4, height, 0)

The radii and the height can be manipulated by the user though the bloom diagram.

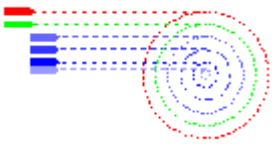
This editor's purpose is to manipulate the geometry of ovary and positions of petals/sepals. Thus the flower parts are represented by triangles (the specific shapes of these are defined during sketching surfaces to create leaves, petals and sepals). Any changes made by the user are immediately reflected in 3D view of the bloom diagram (a window directly below the diagram).



The petals/sepals are added by clicking on a triangle on the diagram and then on a corresponding leaf/petal in the display list.

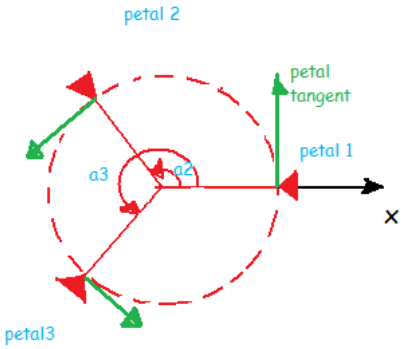
The petals/sepals are arranged symmetrically around the center axis of the bloom (going through the ovary). One of the directions for future work is to add functionality to allow differentiated positioning of petals and sepals as well as adding other bloom parts, such as pistil and stamens.

The bloom has the following parameters that a user can modify:



- Four radii of the ovary (represented in blue). These specify how wide the ovary will be at quarter, half, three quarters and full height of the ovary. To manipulate these – drag the blue handles on the circular diagram.

- **Ovary height** – represents how tall the ovary will be. To manipulate it – drag the blue handle at the side view.
- **Petal radius** – represents how far from the center of the bloom the petals will be (red handle on circular diagram).
- **Petal height** – represents how high with respect to the ovary the petals will be placed (red handle on the side view).
- **Sepal radius** – represents how far from the center of the bloom the sepals will be (green handle on circular diagram).
- **Sepal height** – represents how high with respect to the ovary the sepals will be placed (green handle on the side view).



The angle between the positive x-axis and vector from the center of the bloom to the petal position is saved for each petal and later used in drawing the bloom.

Each petal/sepal is positioned at the point (the coordinates of that point are calculated from the corresponding triangle on the diagram) and rotated around petal/sepal tangent -45 degrees and around petal vein for a corresponding angle plus 90 degrees (for example, on this diagram petal3 would be rotated – (a3+PI/2) radians). Such manipulations ensure that all the petals/sepals are positioned uniformly around the bloom and facing upward and toward the bloom center.

After all the required manipulations of the bloom are done, it can be saved to display list. At that point a bloom can be added to a stem at any control point.

Controller Classes

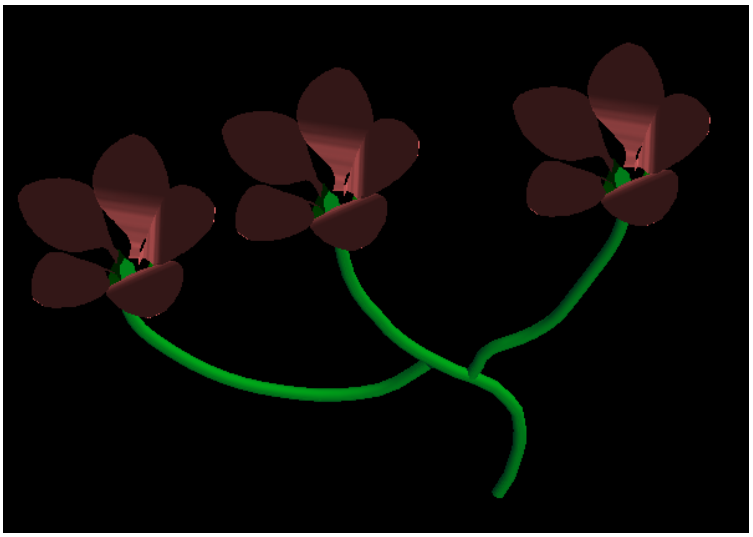
We created four controller classes to handle all Window and Qt events. The *GLSketch* class is our main controller class. It owns the *StemSurface* and handles the user sketch input. It is also responsible for sending *FlowerParts* to the *DisplayList* to be saved.

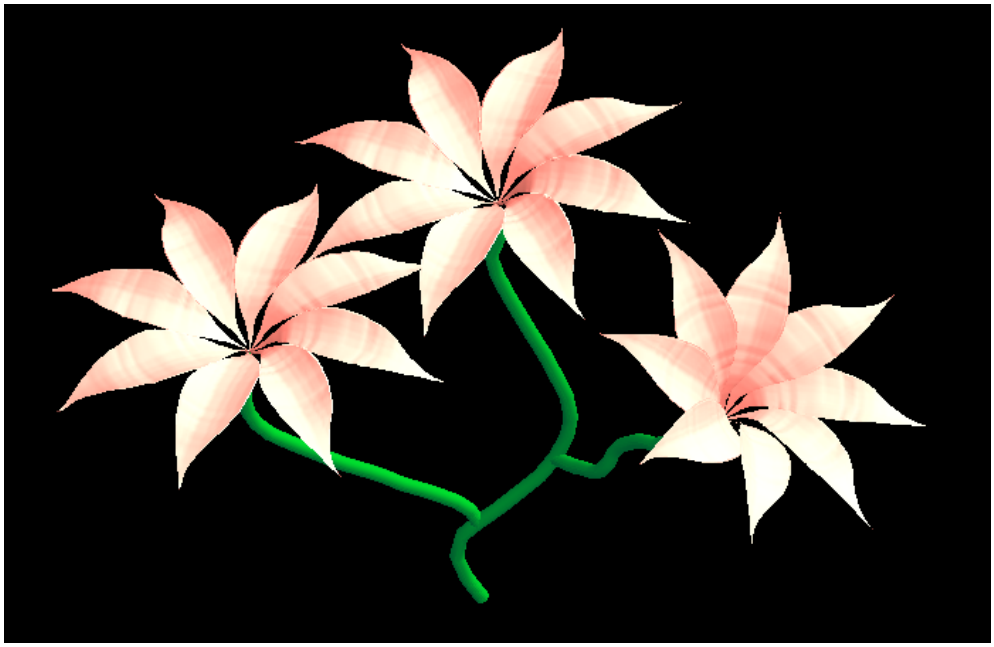
The *DisplayList* class is quite simple. It just displays a copy of each saved *FlowerPart*. When a *FlowerPart* is clicked on, *DisplayList* emits a signal for either the Stem in the *GLSketch* window or *CircleDiagram* to catch to add that *FlowerPart* to the stem or bloom.

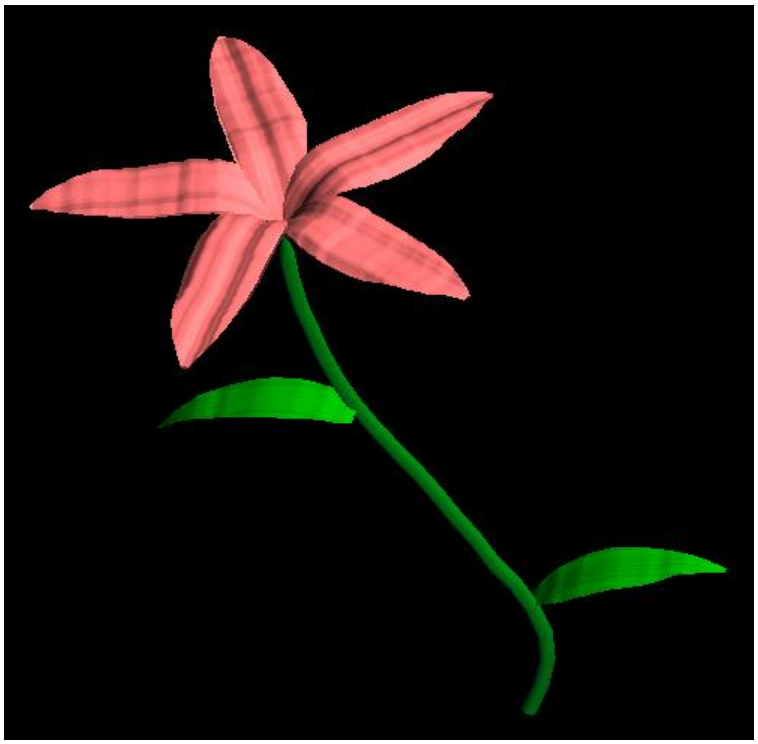
The *StemSurface* class keeps track of selected parts of the flower model while in *Stem* mode. It owns a *Stem*, representing the root stem of the flower, a branch representing the current selected branch and a vector of *Surfaces* representing the leaves currently attached to the stem. It handles all mouse events, creating and manipulating stems and connecting *FlowerParts.*

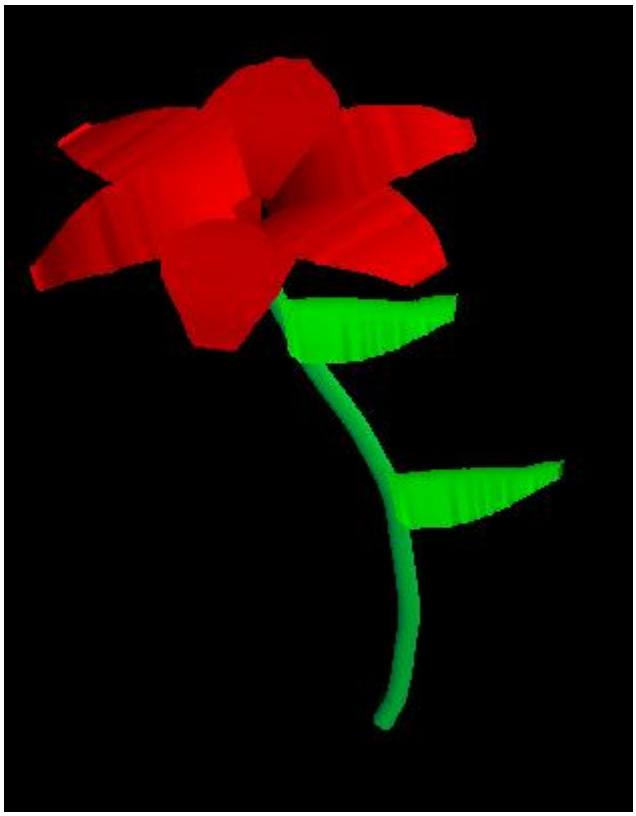
Lastly, *CircleDiagram* and *CircleDiagram3D* are responsible for creating *Blooms*. *CircleDiagram* provides a 2D view of the bloom structure while *CircleDiagram3d* displays a 3D preview of the ovary with petal and sepal placement. *CircleDiagram* translates the user input to create a *Bloom* once the "Create Bloom" button has been pressed. Details of the bloom structure can be found in the Bloom section of details.

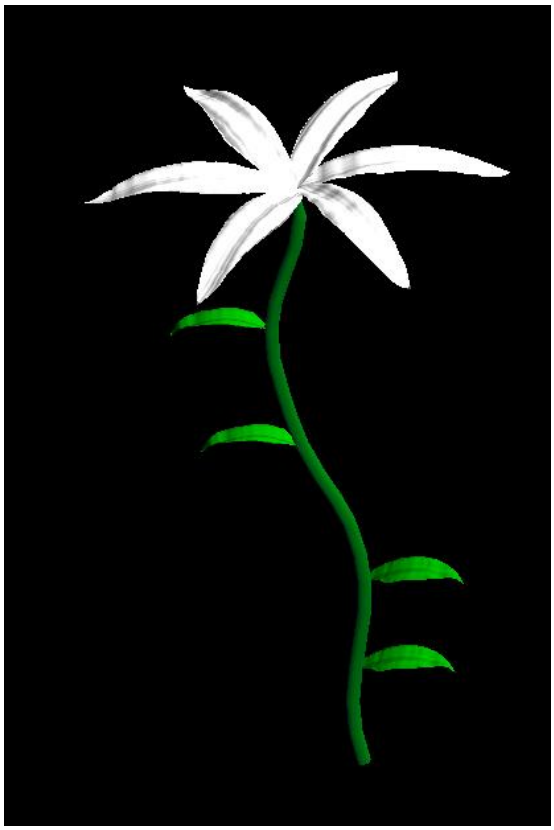
Sample Results





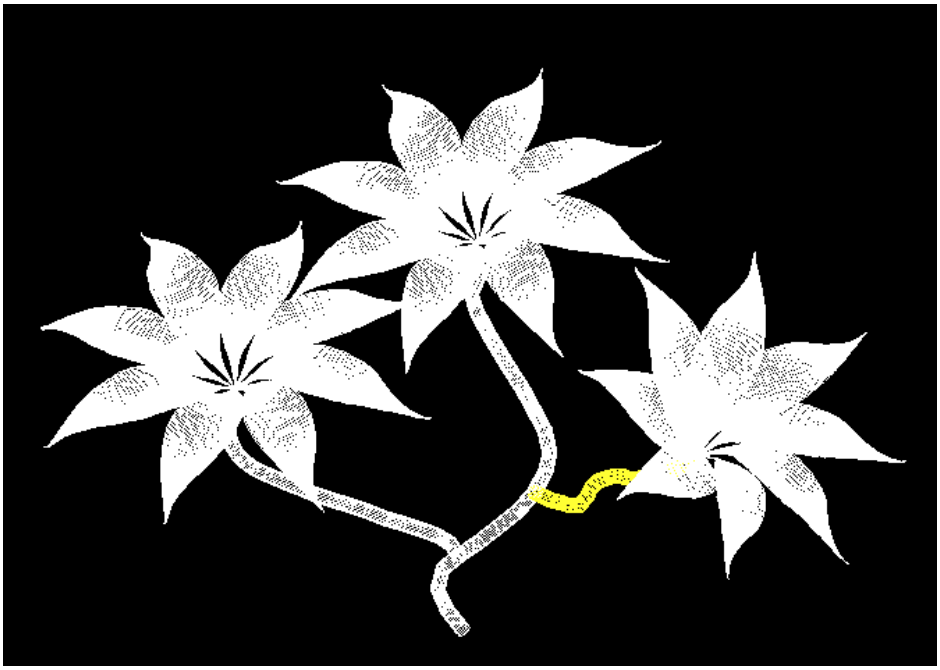






Future Work

Our system creates complete models of flowers. Unfortunately, we did not have the time to implement the creation of stamens, which would make the models much more realistic. Also, we were unable to add load and save functions, however we intend to add this functionality after the final submission of the project.



Conclusion

We succeeded in our main goal of creating a sketch-based flower modeller. We believe our interface is intuitive and easy to use. Because of our class hierarchy, the program could easily be extended to make scenes of flowers instead of individual ones by adding save functionality for the Stem class. We learned much about topics in the course, including B-splines, ruled-surfaces, surface manipulation, sketch-based modelling, and surfaces of revolution and are very pleased with the our results.

References

Local B-spline Multiresolution with Examples in Iris Synthesis and Volumetric Rendering, *Faramarz F. Samavati, Richard .H. Bartels and Luke Olsen, Image Pattern Recognition: Synthesis and Analysis in Biometrics, Series in Machine Perception and Artificial Intelligence , Vol. 67, World Scientific Publishing, 2007.*

Floral Diagrams and Inflorescences: Interactive Flower Modeling Using Botanical Structural Constraints, *Takashi Ijiri, Shigeru Owada, Makoto Okabe, Takeo Igarashi, ACM Transactions on Graphics, 2005 (Proc. SIGGRAPH 2005), Volume 24, No.3, pp. 720-726, SIGGRAPH 2007.*

Seamless Integration of Initial Sketching and Subsequent Detail Editing in Flower Modeling. *T. Ijiri, S. Owada, and T. Igarashi. The University of Tokyo, Eurographics Association and Blackwell Publishing 2005.*

Sketch-based Modeling with Few Strokes, *Joseph J. Cherlin, Faramarz Samavati, Mario Costa Sousa, Joaquim A.Jorge.*

http://en.wikipedia.org/wiki/Inflorescence for definition and types of inflorescence patterns.