Abstract

Visualizing and manipulating large feature diagrams is still an open issue for the SPL community. Few effort has been made on improving the techniques to get the most out of drawing space and current feature modeling tools either use file-system-like trees or 2D graphs that must be scrolled to locate features. The aim of this paper is presenting a new method to draw large feature models based on cone trees, a three-dimensional visualization technique to represent hierarchical information. In order to evaluate our proposal, we develop a prototype that generates standard 3D files so it can be easily integrated into existing tools. Finally, we present a roadmap for a future extension of our proposal with dynamic behaviour so large feature models handling might be improved.

1. Motivation

As software product lines (SPL) have become a reality in software development and tools are appearing to deal with the different activities in SPL, the size of the systems impose a limitation of the amount of information that users are able to manage at the same time. Last edition of ViSPLE workshop [15] was held with the intention of finding solutions for complex SPL visualization. It remarks the community interest in finding solutions to problems such as the understanding of large SPL and variability representation. In the workshop conclusions report, it was emphasized the importance of visualization techniques when the user wants to look at a lot of informations at the same time and was recommended to find for solutions in the visualization theory and other applications of visualization techniques. That is the intention of this work, which is based on works in 3D visualization such as [5, 10] that propose using three-dimensional rooms to visualize information.

We focus on feature diagrams in all their possible forms [14] are affected by the increasing size of SPL. Most of existing tools either use browser-like viewers to represent feature diagrams or traditional graph layout algorithms to provide graphical interfaces for feature modeling. As an advance towards finding better techniques for SPL visualization, Nestor et al.[12] propose some visualization techniques to improve some common tasks in SPL engineering. This paper intends to propose an alternative to existing works by introducing three-dimensional representation techniques to the visualization techniques used to manipulate feature diagrams.

Specifically, cone trees [6, 7, 13] are proposed as an alternative to represent large hierarchies in the three-dimensional space. Cone trees distribute the elements of tree-like structures in a virtual 3D room where roots of sub-trees are placed in the apex of cones that link them to their child nodes which are placed around their cone bases. Some experiments have evaluated how they have improved some tasks when users work with large hierarchies, which is our premise to propose them to manipulate large feature models.

In this paper our contribution is fourfold: i) we propose a roadmap to produce and evaluate the use of cone trees to represent feature diagrams in what we call feature cone trees (FCT); ii) We provide a detailed description of the process to obtain static FCT; iii) We propose using several techniques to incorporate dynamic behaviour into FCT to improve the user experience; iv) A prototype to produce static FCT is provided so a first evaluation of our proposal can be given.

This paper is structured as follows: Section 2 briefly describes cone trees and comments the benefits of using them compared to traditional 2D layout algorithms; In Section 3, we propose a process to produce FCT from feature diagrams, remarking how the original proposals on cone trees are adapted to fit into feature diagrams structure; Some details about a prototype implementation of our proposal that integrates into our FAMA tool[4] are given in Section 4; In Section 5, we discuss about the applicability our proposal based on previous evaluations of cone trees in other contexts; Lastly, some conclusions and a roadmap for our future work in feature diagrams visualization are given in...
Section 6.

2. Cone Trees

Robertson et al. [13] proposed the visualization of hierarchical information in the three-dimensional space. The root of the hierarchy is placed at the ceiling of a virtual 3D room and its children are evenly placed along the base of a cone whose apex is the parent node. Next layers are drawn following the same process, where child nodes will be considered the root of their subtrees.

The interaction with a cone tree is eased by rotating all the subcones whenever a node is selected. This way, selected and parent nodes are the nearest from the user viewpoint. Animations are introduced so the user is aware of the rotation process and the perception of the whole structure is never lost.

The main goal of cone trees is an effective use of screen space to visualize the whole structure. While 2D graph layout algorithms [11] are unable to fit large hierarchies on the screen, cone trees are able to use depth to make a better use of space. When broad hierarchies are drawn using traditional 2D algorithms, the user must scroll to perceive the whole structure and understanding the hierarchy topology is a hard task. Cone trees prioritize the understanding of the topology and emphasizes the information relevant to the user bringing it to the front of the three-dimensional space.

The benefits of using cone trees to represent large hierarchies have been considered for several case studies such as an organizational structure browser [13] or Unix file-system browsers [6]. For this last case, the authors present a small experiment to measure how cone trees improve some common tasks with file browsers. The goal of this paper is describing a process to transform feature diagrams into cone trees so it could improve some interacting tasks on large diagrams, such as they do in other contexts.

3. Feature Diagrams Visualization with Cone Trees

We aim to use cone trees to represent feature diagrams in order to give a solution to large feature diagrams visualization. Our first step towards this objective consists of defining the static layout of feature cone trees (FCT), that we describe in this paper. As a second step, we approach the guidelines to deal with dynamic handling of cone trees that will be a goal of our future research (Figure 1).

3.1 Building Feature Cone Trees

Cone trees depict hierarchical information by using nodes and cones or lines that link parent and child nodes.
Analogously, child features in the feature diagram are represented by a semitransparent sphere with radius \( r \). In our particular case it means that they are recursively built from the leaves of the feature trees are built bottom-up. In our particular case it means we cope with building them from feature diagrams. Cone trees are built bottom-up, because we need to know the radii of their respective child features. This is the reason why the trees are built bottom-up, because we need to know the radii of the subcones for each child node to build an upper level. To define a cone, its height and base radius are used. The height of the virtual cone \( h \) is also a constant of the cone tree. The radius of the cone base \( R \) is obtained by a formula that approximately estimates the perimeter of the base by adding the diameter (twice the radii) of the child features (see Fig. 3).

\[
R = \sum_i \frac{2 \cdot r_{c_i}}{\pi} = \sum_i \frac{r_{c_i}}{\pi}
\]

Note that the result is not precise as we are really calculating the perimeter of the circumscribe polygon to the cone base which is less than the perimeter of the cone base. If we used this imprecise radius, the spheres may overlap with a high number of child features so the radius must be increased. We correct this deviation adding the radius of the largest \( r_{c_i} \) to \( R \) as proposed in [6]:

\[
R = \sum_i \frac{r_{c_i}}{\pi} + \max_i r_{c_i}
\]

Once the cone proportions are obtained, each sphere must be placed around the base. We must calculate the angle \( \theta_i \) between two adjacent child features \( f_{i-1} \) and \( f_i \) whose radii are \( r_{c_{i-1}} \) and \( r_{c_i} \) respectively. First child feature \( f_1 \) is placed at zero-radians position. Remaining child features are placed around the base calculating its \( \theta_i \) angle by this formula (in radians):

\[
\theta_i = \frac{r_{c_i} + r_{c_{i-1}}}{R} \theta_{i-1}
\]

It is also an approximate calculus, however its visual effect is unnoticed. Finally, we have to calculate the position of each sphere in the three dimensional space from the angles. Let us consider that the center of \( F_p \) sphere is at \((0,0,0)\). We may set the position of each child feature by calculating its spherical coordinates \([8]\). We only need the distance between \( F_p \) and \( F_c \) sphere centers \((l)\); the angle \( \alpha \) forms the y-axis \((\alpha)\) and the angle \( \theta \) from the z-axis \((\theta)\). Cartesian coordinates can be easily calculated from these data in case you prefer them to calculate the position. Notice that \( l \) and \( \alpha \) are fixed parameters of the cone so they must not be calculated for all the child features.

3.3 Building the relationships

Once the spheres are placed at the tree, relationships between parent and child nodes must be drawn. As spheres are semitransparent, visual cutter could arise in case we drew lines from the parent sphere center to the child sphere center. We do not want the lines to be within the spheres but going from the parent sphere surface to the child sphere surface. In Figure 4 a relationship is drawn with length \( l \) starting from a point \((o_x, o_y)\) in the surface of the parent sphere. Notice that two dimensions are used in the example to simplify the procedure and a third dimension can be easily added. A line can be describe by its origin point \((o_x, o_y)\) and destination point \((d_x, d_y)\) which is also on the...
child sphere surface. These parameters are calculated by the following formulae:

\[
\begin{align*}
(o_x, o_y) &= (r_p \sin \alpha, r_p \cos \alpha) + (p_x, p_y) \\
(d_x, d_y) &= (c_x, c_y) - (r_c \sin \alpha, r_c \cos \alpha)
\end{align*}
\]

where \((c_x, c_y)\) is the parent sphere center, \((c_x, c_y)\) is the child sphere center and \(\alpha\) is calculated as follows:

\[
\alpha = \arctan \frac{R}{h}
\]

In case we want to use spherical coordinates (polar coordinates in the bi-dimensional space) origin and destination points are defined in terms of an angle and a distance from the origin, which is the parent sphere center. The angle for both points will be \(\alpha\); the distance for the origin point is equal to the radius of the parent sphere \(r_p\); the distance for the destination point is equal to \(r_p + l\) where \(l\) is the length of the line that can be easily calculated from the available parameters:

\[
l = \sqrt{R^2 + h^2} - r_c - r_p
\]

### Figure 4. Relationships rendering

Another factor to be taken into account are the set relationships, that link an unique parent feature to a set of child features with a common cardinality. As the cardinality of a set relationship must be drawn, if we used lines to link child features in a set relationship to their parent feature, and placed text in any of the lines, it will be impossible to figure out the boundaries of the set relationship. As a solution to this problem, we propose creating an intermediate node (sphere) containing the cardinality of the set relationship that links to the upper parent node and the lower child nodes. This way we avoid the misunderstanding a floating cardinality could cause in a complex feature diagram.

### 3.4 Aesthetics Aspects

Feature Diagrams mainly represent three kinds of information, namely: i) features, ii) which are the related features and iii) how they are linked (mandatory, optional, set relationships, ...). By now, we propose a method to represent the features (spheres) and how they are related (lines). Although we have already proposed a representation of set-relationships, we must also provide information regarding the kind of relationship of binary relationships, mainly optional and mandatory ones.

Carriere and Kazman [6] propose using colours, shapes or text to provide extra information to cone trees. In our case we are already using text within the spheres to represent the feature names. We have opted for using colours to distinguish between optional and mandatory relationships. However, instead of changing the colour of the line that represents the relationship, we change the colour of the sphere of the child feature that is affected by the relationship.

Colour selection is critical, because the selected colour must be easily discriminated and must induce to a fast association to the represented information, in our case, the kind of relationship. We have used red for the root and any mandatory feature; white for optional features; and yellow to fill a set-relationship cardinality sphere and white for its child features. This is our first approach and testing other colour palettes that could help on easily discriminating the relationships is an important issue on our future work. Even colouring the relationship lines may be useful when it is wanted to remark information relevant such as an erroneous relationship [16].

We have proposed the usage of semitransparent spheres which is different from the original proposals on cone trees [6, 13]. It allows to place a text with the name of a feature within an sphere, instead of the side of the node, getting the most out of the drawing space. We use billboardning [2] to draw text in the three-dimensional space. It moves the text everytime the camera or the tree rotates so it will be always staring at the users point of view.

### 3.5 Introducing Dynamic Behaviour into Feature Cone Trees

The real benefit of using cone trees to represent large hierarchical structures comes when techniques to interact with them are introduced to get the best user experience. Although the main goal of this paper is presenting a process to transform feature diagrams into cone trees, we want to remark some of the existing techniques to interact with cone trees.

Visualizing the whole FCT gives a first understanding of the underlying topology of the feature diagram. For a complex or large hierarchy, users focus on the part of the FCT
that is relevant for their current work. While some information becomes relevant, others become irrelevant. Gardening operations such as pruning and growing the tree are proposed in the literature [13] to reduce the complexity and increase the usability of large hierarchies. Carriere and Kazman[6] propose using fish-eye viewing to display only those nodes whose degree of interest (DOI) is greater than a threshold. A node DOI is calculated from the positions in the tree where the user focus is and that node. Filtering only takes into consideration the structure of the tree but ignore any semantics within the model. For our purpose, we could use DOI to draw the cross–tree constraints, which have not been considered in this paper, just when the focus is on a node that is affected by one of them. This way, the visual cutter derived from drawing all the cross-tree constraints in a FCT may be reduced and could be drawn just on demand.

FCTs are subject to change as the user may modify the feature diagram and execute queries to filter the information within the diagram[3]. Anytime a FCT changes, the layout must be rebuilt (transforming the new feature diagram into another FCT) or rearranged. To avoid the user loosing the scope of these changing operations, movement through the cone tree must be smooth enough. As an example, Mackinlay et al.[9] propose a technique to move a viewpoint to a point of interest smoothly.

4. Implementation

A prototype of the described mapping from feature diagrams to cone trees has been implemented and integrated into our FAMA tool [4]. As we wanted the implementation to be easily reproduced and integrated into third-party tools, we opted for developing a prototype that used X3D [1] to represent an FCT. X3D is an ISO open standard file format to represent 3D scenes and objects. They can be reproduced by a set of tools which are frequently integrated into common web browsers. X3D is an XML-inspired format that is generated from FAMA with few effort.

FCTs in Figures 5(a),6(a) and 8 represent three feature diagrams. They have different sizes so the effects of using FCT can be evaluated by the reader. Figure 5(a) represents a Home-Integration System(HIS) from [3]; Figure 6(a) is a top view from SAUCE feature diagram [16] that describes an ERP software product line; Figure 8 depicts a random feature model with 1000 features where we can appreciate its topology at first glance. These and more examples of the models obtained from FAMA can be found at http://www.lsi.us.es/~trinidad/fct.

5. Discussion

We have presented a proposal to draw FCT as an alternative to traditional 2D feature diagrams representation. As an evaluation of our proposal depends on building a complete tool that also integrates dynamic behaviour, we rely on other studies where cone trees have been applied to other contexts. In [6], an empirical study on cone trees performance tries to find their limitations. Although there are some factors such as drawing speed, hardware acceleration and screen space and resolution that may affect the obtained results, they suggest that with cone trees containing more than 1000 nodes or 10 layers, distant nodes cannot be identifiable as can be appreciated in Fig. 8. However, dynamic behaviour techniques and gardening techniques considerably increase the limit to up to 5000. A small user test is presented to show how cone trees helped users to find some information in a large file system faster than using traditional shell utilities.

Although the results were very promising, Cockburn and Mckenzie [7] show an sceptical point of view of this improvement due to the results of another case study. In this study, cone trees are demonstrated to be faster for some tasks while traditional browsers performed better for other tasks. However, users were already familiar to traditional browsers and the lack of training in using cone trees affected the experiment. These contradictory results suggest
Figure 6. SAUCE ERP Representations
us to adopt an eclectic position in the evaluation of the best choice. As cone trees may improve some tasks with large hierarchies, other tasks may be performed faster using traditional 2D representations.

FCT seem to be an alternative to traditional 2D techniques as they are able to rearrange the structure to bring the relevant features nearer the focus using gardening and fisheye techniques when users work with large feature models. Existing feature diagram tools commonly deal with file-system-like editors where scrollbars are used to navigate on a structure that needs several screens to be completely drawn. As an example, in our FAMA tool we needed scrolling through 11 screens (Figure 7) to visualize the whole SAUCE feature diagram in Figure 6(b). This situation is similar to the one in the experiment presented by Carrier and Kazman[6]. In their study, they validate cone trees against traditional file-system editors by proposing some inexperienced users performing some traditional file-system tasks using both systems. Therefore we expect obtaining similar results in our future work, where we will replicate this experiment to obtain results specifically for FCTs but considering different feature diagram sizes.

FCTs don’t intend to overcome 2D techniques but to propose an alternative to large feature diagrams. Traditional 2D techniques can be the best choice for small feature diagrams such as the HIS in Figures 5(a) and 5(b). Furthermore, Robertson et al.[5] argue about the effectiveness of cone trees for balanced hierarchies, where it is difficult to distinguish the subtrees by their topology. In these cases, 2D techniques exploit the space in a better manner that FCT. However, handling large set relationships with many child features is not trivial in 2D layouts. In these cases, FCTs reduce the needed space in Figure 6(b) as they surround the
parent node space an can be a better choice.

6. Conclusions and Future Work

This is the first step (Fig. 1) towards a full integration of 3D interaction to feature modeling. Our intention is presenting an alternative in a field that we think that has not been sufficiently explored such as feature diagrams visualization. We want to highlight that our contribution brings several solutions that have succeeded in other research areas to the software product lines context.

This paper cannot be understood without taking into account any further work. As a final goal, we want to evaluate the user interaction with FCT so we can compare current feature modeling tools visualization to our proposal. To achieve it, we will develop a tool that extends our prototype with a dynamic behaviour, as already approached in this paper. Techniques such as fish-eye view, different DOI formulae, colouring and rotations may increase the usability of FCT. However, they will not only improve its usability but also increase our tool functionality as we might incorporate cross-tree constraints to FCT. As they break the tree-like structure, some techniques should be developed to avoid constraint relationship lines crossing and passing node spheres.

We are aware that our proposal is by no means exhaustive while no evaluation is provided. In this paper we have evaluated our proposal relying on previous case studies in other contexts. Whenever we finish our tool implementation, we want to evaluate our proposal by making a full user experience study similar to those commented in this paper.

Acknowledgements

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472) and by the Andalusian Government under project ISABEL (TIC-2533).

References