Non-Disruptive Development of a Next-Generation CAD Application Program

Bastiaan Veelo, SARC B.V., Trondheim/Norway, Bastiaan@SARC.nl
Herbert Koelman, SARC B.V., Bussum/Netherlands, H.J.Koelman@SARC.nl

Abstract

There is a limit to how far an application program can evolve in incremental steps. At some point in time, progression requires radical changes: a new generation that parts with the limitations of its legacy. In our case, due to richness in features, a complete rewrite of our hull design software would lead to an unattractively long down-time, which is why we have pursued a more efficient allocation of our programming resources. This is the report of an approach in which we keep both the production version and the development version fully functional within the same executable, providing a non-disruptive transition from one application generation to the next, while building on proven foundations.

1. Introduction

Almost two decades ago, SARC started research and development of a new computer method and application program for the design and fairing of ship hulls, called Fairway. Fairway would remedy the fundamental disadvantages and limitations of NURBS surfaces, a technique that is still at the core of most CAD software today, *Koelman (2003)*. Since its release on the maritime market in 1995, Fairway has been steadily improved and extended, so that it currently has a diverse feature set including

- User-defined geometric master-slave relations between curves
- Projection of spatial curves onto the defined model
- Hydrostatic analysis
- Target and model sectional area curves (SAC)
- Shape transformations
- Developable surfaces
- Shell plate expansion
- Multiple solids and model variants
- Boolean operations on solids
- Surface quality assessment methods.

Additionally, external modules have been developed that build on Fairway technology, such as

- A hull server for the interpolation of arbitrary hull lines upon the request of another process (RPC fashion)
- An import module that finds matching topological relations in a curve cloud for the conversion of loose ship lines into a solid model
- PhotoShip, an application for the reconstruction of 3D digital models of physical objects based on photographs.

At the turn of the millennium a new research project started in Trondheim to address another common problem in CAD: the fact that the extent, or scale of influence, of geometric manipulation decreases with increasing model detail. Because of this, the cost of producing variations on a design generally increases rapidly as the design progresses. The project produced an experimental version of Fairway demonstrating that the principles of spatial deformation can be used to achieve smooth shape variations of arbitrary extent, on models of any detail, *Veelo (2004a,b)*. However, we found that a commercial implementation imposed requirements on user interface, on graphical performance and on internal code structure that Fairway did not meet at that time. This extension, however useful,

exceeded the possibilities of incremental code evolution. We had hit the code barrier.

2. Breaking the code barrier

In 2006, the time was found right[1] for a radical modernization of Fairway, and we considered our options.
One possibility is to start from scratch and implement the next generation from the ground up. The advantage is that limiting legacy code can be eliminated and that you are free to use any modern programming language, library and technology. The disadvantage is that you will spend a long time implementing data structures, algorithms and features that already exist in the proven implementation of the previous generation. In addition comes time needed for testing and debugging. It is important to note that the new application is not production-ready until its feature set is on a par with the previous generation. The approach is known to work, even for large projects, as illustrated by the rewrite of CATIA V5 in 1999 by Dassault Systèmes, which could afford to put one thousand programmers to the job[2].
We don't have that kind of resources, and with the variety of functionality already present in the production version of Fairway, we would not be able the rewrite all of it from scratch without the result being outdated by the time we would be done. We needed a different option. Fortunately, we were not unprepared.

2.1. Anticipatory measures

In the past, strategic decisions had been made at SARC that were to offer flexibility and security regarding future technological developments and economical changes. These were

1. the use of a standardized programming language,
2. a generic menu library (“genmenu”), and
3. an abstracting Graphical User Interface (GUI) library (“Ywin”).

2.1.1. Language and compiler

Since 1980, Pascal has been the predominant programming language in use at SARC. Pascal has a number of advantages for engineering purposes over the contemporary alternatives Fortran and C:

- It enforces structured programming. This leads to software that is better structured, and hence may contain less bugs and is easier to maintain.
- It has a relatively small number of statements, yet is a very powerful language. So for people not 100% of the time engaged in programming it is easier to comprehend than one of the other languages. Moreover, the language does not invite to whizkiddery.
- The language is straightforward, also from a compiler point of view, which means it is quick to compile to efficient binaries.

An early phenomenon regarding Pascal was that different compiler vendors extended the language in different ways, leading to a variety of Pascal dialects. In 1993 the International Organization for Standardization standardized Extended Pascal (ISO 10206), and rather than using a non-standard dialect of one specific vendor and thereby creating a dependency on its financial well-being, SARC opted for standard Extended Pascal. Prospero Software was the first to implement that standard, and its compiler still serves its purpose very well. Even so, Extended Pascal never enjoyed the hype and popularity we have seen in some other languages, which is why we have to turn to another language in our selection of third-party support-libraries, as you will see in a moment.

1 As a matter of fact, it happened during a comfortable evening chat at COMPIT'06 in Oegstgeest.
2 According to a CATIA instructor from IBM.

2.1.2. Abstraction

Ywin and genmenu are in-house developed libraries that increase our programming productivity and help to maintain uniformity of user interfaces. They are used across our entire product line, not just Fairway. Ywin is an abstraction to the Microsoft GUI and system libraries. One important role of Ywin is that it shields our programmers from the frustrations of having to deal with the Microsoft win32 API, which we consider to be of great relevance for productivity. Genmenu is a very flexible event-driven library that enables us to quickly create menu structures, input forms and spreadsheet-like alphanumerical interfaces. But the most important aspect in the context of this article is that Ywin and genmenu form a division between our own computer code and the underlying system libraries. In principle, we can migrate all our products to a different supporting system by changing the internals of Ywin and genmenu.

2.2. First approach

Our first approach for crossing the code barrier was to write a drop-in replacement for Ywin and genmenu, using a modern high level GUI library. This would open the road for more easily incorporating the power of a modern GUI into our abstracting libraries, which would benefit all our software at once. It would also help making Fairway ready for its leap forward.

2.2.1. GTK+ and GPC

We selected GTK+ as the GUI library to build on[3], mainly because it is written in C which is easily callable from Pascal, and because it is object-oriented nonetheless. GTK+ is also used as the basis for several other abstracting libraries, which is encouraging. As we were writing the library (now called YGTK) from the ground up to the specifications of Ywin and genmenu, our first Pascal test programs were small, using the currently available subset of functionality. Initially, we compiled these programs with the GNU Pascal Compiler (gpc), which is a free compiler that supports most of the Extended Pascal standard[4]. Because gpc produces object code of the same format as the C compiler gcc, this allowed us to link the Pascal parts and the C parts into the same executable, and use the same debugger to debug both parts. The project came along well, but when the time came to hook up the first real Pascal application to YGTK, things turned out to be more complicated than expected.

2.2.2. Complications

Firstly, the application pulled along a load of Extended Pascal modules that it links to, and gpc failed to compile that collection. Reasons were that gpc does not cover enough of the Extended Pascal standard, and our modules made use of some local Prospero extensions and platform-specific APIs. Instead of working out all the incompatibilities and the ones that were yet to be discovered, we decided to compile Pascal with Prospero and C with gcc. Due to different object formats these parts had to be separated into different DLLs from now on. This came with its own set of difficulties, including a compiler bug (or two) that had to be resolved and the writing of an ingenious piece of Assembly code to allow C code to call back into a nested Pascal procedure without messing up its call stack.
Also, while GTK+ originates from the GNU project and Linux world, it eventually became clear that it had not managed to flawlessly deal with the peculiarities of the MS Windows platform yet. Most notably were problems with the kerning of letters on screen and output to certain ink-jet printers consisting of just blocks of black. These problems got fixed, but not before we had long dropped the project.
Sadly, it was only after having implemented most of Ywin's functionality, that we uncovered a new set of inter-dependencies with low level code. In particular it was the hard-copy output that was problematic. Over the years, Ywin had grown beyond a clean abstraction of just GUI code. We had to

3 http://www.gtk.org/
4 http://www.gnu-pascal.de/

conclude that it was too difficult to separate plain GUI calls from the rest and that it would take too long to port all low level code. As it was not getting us much closer to our primary goal, being the next generation Fairway, we decided to stop there and consider a different approach.

2.3. Second approach

What we had been successful at was solving the DLL issue, and we were confident that we could tie together anything with a C interface. Our next approach was to shift the borderline between old and new: write a completely new interface for graphical presentation and interaction, using modern solutions, and only keep the proven data structures and routines that operate on them in Pascal. This approach has given us much of the same freedom as we would have had when writing a new application from the ground up, while not having to reinvent the wheel. It has almost been like writing a new application, but with a flying start. The result indeed looks and feels like a new application.

2.3.1. Qt and Coin3D

When shopping for support-libraries we soon landed on the Qt framework for the user interface[5] and on Coin3D for hardware accelerated graphics[6], both written in C++. Qt is very rich in features, has solid support for the MS Windows platform and generally makes programming in C++ more enjoyable. Coin3D is an implementation of Open Inventor, a retained mode high level object-oriented framework that abstracts OpenGL, with an interface to Qt.
Both Qt and Coin3D share a very important feature: they provide full access to their source code, with a license to change it. We have made use of that opportunity on several occasions, both for implementing extended functionality and for fixing problems faster that their support departments. This combines the advantages of building on other people's excellent work with the flexibility and security of having done everything ourselves.

3. Linking old and new

We started by isolating the core Fairway modules in a DLL, but due to entanglement with Ywin and other modules we were quickly heading for the same problems as earlier. The solution was both simple and advantageous: to embed the entire Fairway application in one DLL, including Ywin, genmenu and other modules that it links to. A small C++ main function provides the entry point for the executable and no changes were required to the original code.
The new C++ code lives in a second DLL, and both old and new code can call each other across DLL boundaries (Fig.1). The Qt event loop is easily stopped and started, which makes it possible to switch back and forth between user interfaces without having to leave the application or save the model file. This has the great advantage that every feature of the new interface is usable as soon as it is ready, because for everything that is not implemented yet one simply switches into the other interface. This has allowed us to start testing and using the second generation Fairway long before it is finished or feature-complete. This has produced valuable feedback and allowed early adjustments of the design. There is no down-time, the new generation just grows steadily alongside the preceding generation, until it is mature enough to replace it.
The advantage of shared core code between both the production version and the development version also works the other way. On several occasions we did improve the core Pascal code in support of the new generation, and as both generations are using the exact same core procedures, these improvements were directly available in the production version, without back porting of any kind.
The two user interfaces work quite flexibly in parallel, as can be seen from Fig. 2, which shows the new interface reusing an old genmenu for configuration of position sets, as an interim solution.

5 http://qt.nokia.com/
6 http://www.coin3d.org/

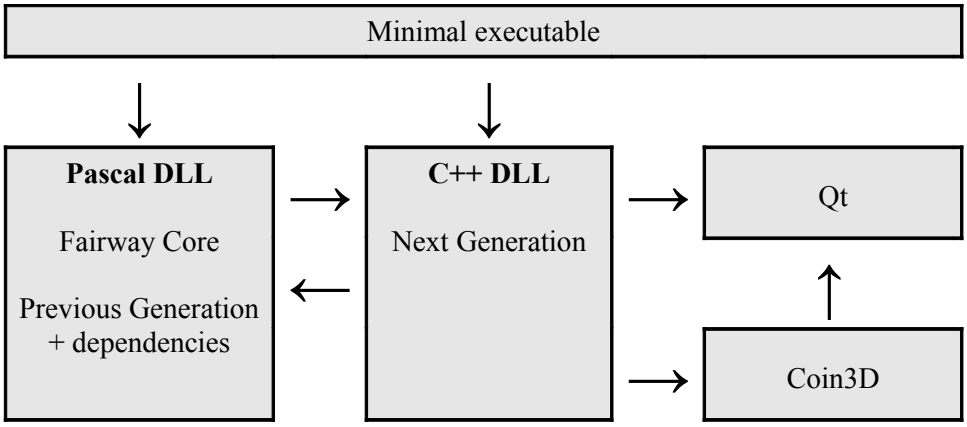

Fig. 1: Configuration of modules.

4. The next generation

Now that we had found the technical solution for the next generation Fairway, and had tested capabilities of Qt and Coin3D, we were in the fortunate position to gather around the table and make a preliminary design of the new user interface. These were the points on our wish list.

4.1. Wish list

1. **Integration of interfaces.** Fairway started with a genmenu-based alphanumerical interface during its early development, and was then extended with a custom graphical modelling interface. Once hardware-accelerated graphics became available to the consumer market, Fairway was given an OpenGL render mode. This mode remained limited to presenting static geometry, because . The three interfaces complement each other but cannot be active at the same time. We wanted the new modelling interface to be hardware-accelerated, dynamic, and provide simultaneous graphical and alphanumerical input and feedback in a unified way.
2. **Redesign modelling actions.** In the old modelling interface the methods for interaction consist of assorted functions, distributed among different menus. These are to be activated in sequence to perform a modelling action, such as connecting network points with a curve. If a function is activated out of sequence, e.g., choosing “Connect Point” without first having activated a curve and requested the display of points, the system complains with a pop-up message. With well over a decade of experience in working with Fairway we wished to redesign the way modelling actions are performed by the user.
3. **Undo and redo.** Undo-functionality is easily taken for granted if it is there, but implementing it in a CAD system is a nontrivial undertaking due to the amount and complexity of information that needs to be kept track of. Fairway has had an implementation of undo for a long time, which it uses internally in the process of saving models to file. This system, however, is limited to Euler operators and only covers the topology of the model. For user-level undo functionality, geometrical changes need to be kept track of as well.
4. **Minimize mouse centimeters and reduce pop-ups.** As we are users too, we want working with Fairway to be a smooth process and non-straining experience. Hence we have the general design objective to keep mouse movements and clicks to a minimum. Pop-up boxes, although effective, disrupt the designer in his work and are often associated with a problem. The old interface pops up a message when an action cannot be performed, like when a curve is locked and therefore cannot be removed. We want the new design to minimize the need for pop-ups.

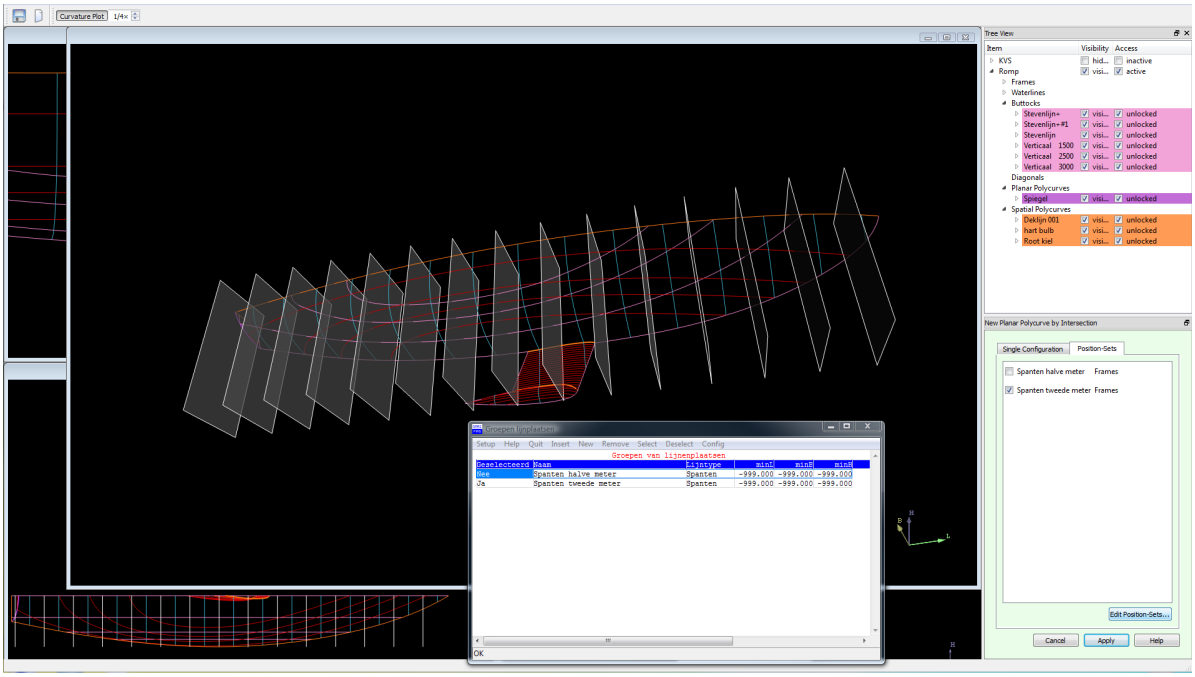

Fig. 2: Example of mixing user interfaces. Here the new interface forwards the configuration of position sets to an old genmenu dialog.

4.2. Design and development

In bullet form, these are some of the characteristics of the new generation Fairway.

- Actions have become a central entity in the user interface. We have gone from a large set of functions to a small set of actions. They have the following properties.
  - Actions have their own user interface panel on which relevant information and tools specific to the context are presented (Fig.3).
  - Only one action can be active at the same time.
  - The action panel appears when an action is started and moves out of the way when it is closed.
  - Actions typically have three stages:
    - Starting stage: Prerequisites such as a selection are checked, possibly with a short note of what the user is expected to do.
    - Configuration stage: The user changes values in the action panel or uses tools for manipulation presented there. Changes are easily visible because the original state of the model is displayed transparently.
    - Apply or Cancel stage: When Cancel is pressed the action is terminated and changes are reverted. When Apply is pressed, changes are committed to the model and an undo-moment is created. This helps reduce the amount of geometrical changes that need to be tracked, and is easily comprehensible by the user.
  - Some actions can be applied to a selection set, e.g., changing properties of several curves in one go.
  - Action panels, as well as other UI elements, can be detached from the main window and put floating on top of it or even onto another monitor (Fig.5). This allows the user to organise the interface to personal preferences and maximise the modelling area.
- Graphical manipulation of positions are performed through specially crafted draggers (Fig.3).
  - Draggers are geometrical objects that can be moved in selectable main planes, or constrained in specific directions, independent from viewing direction. (No more translation parallel to the screen.)

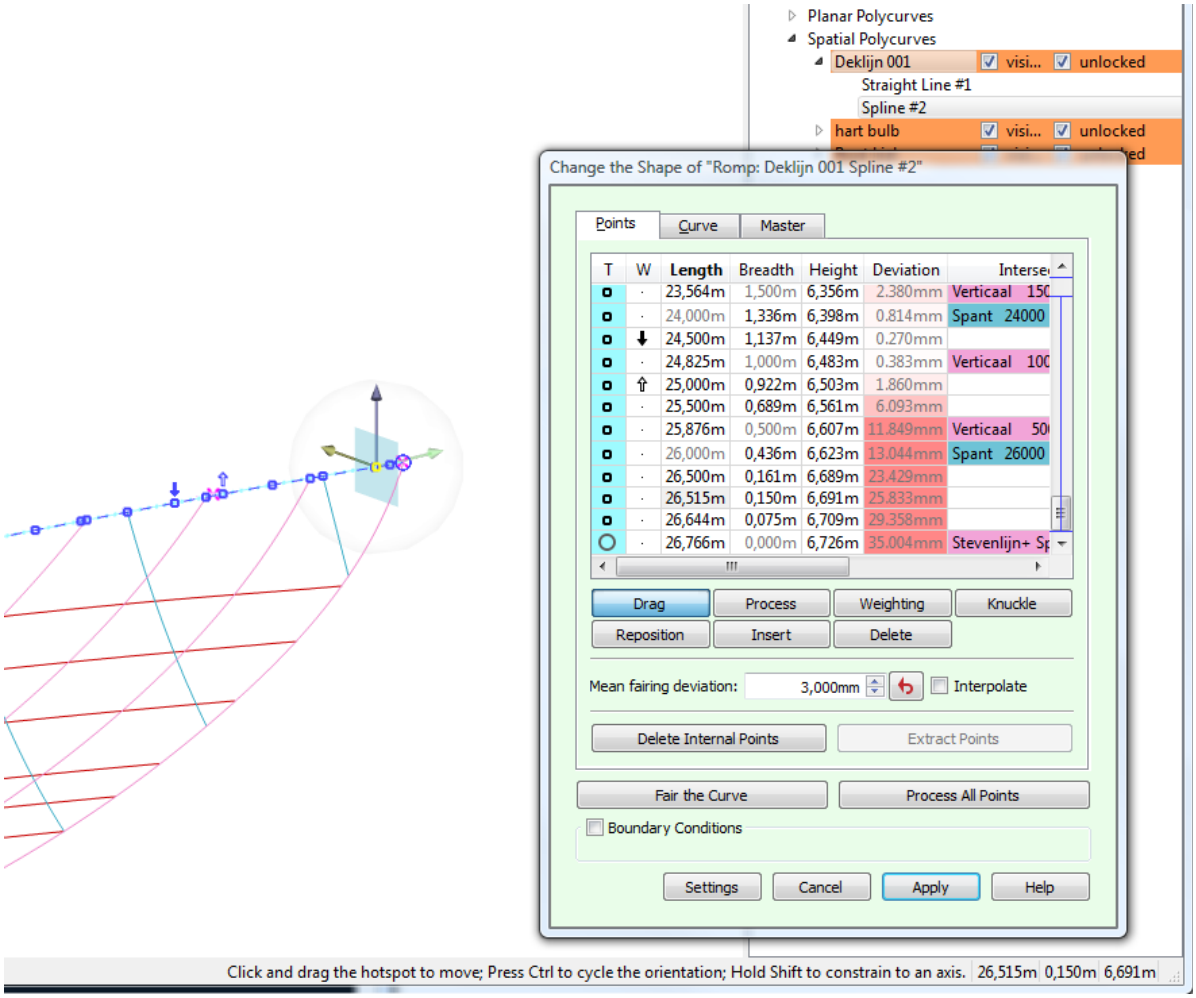

Fig. 3: Closeup of the floating action panel for curve manipulation and a translation dragger. Points are marked to indicate their weight factor (arrow up and down) both in the table and on screen. Next to the editable coordinates is their colour-coded deviation from the curve, followed by the (click-through) intersecting curves. On the status bar the dragger displays operation instructions for its current context, followed by its position.

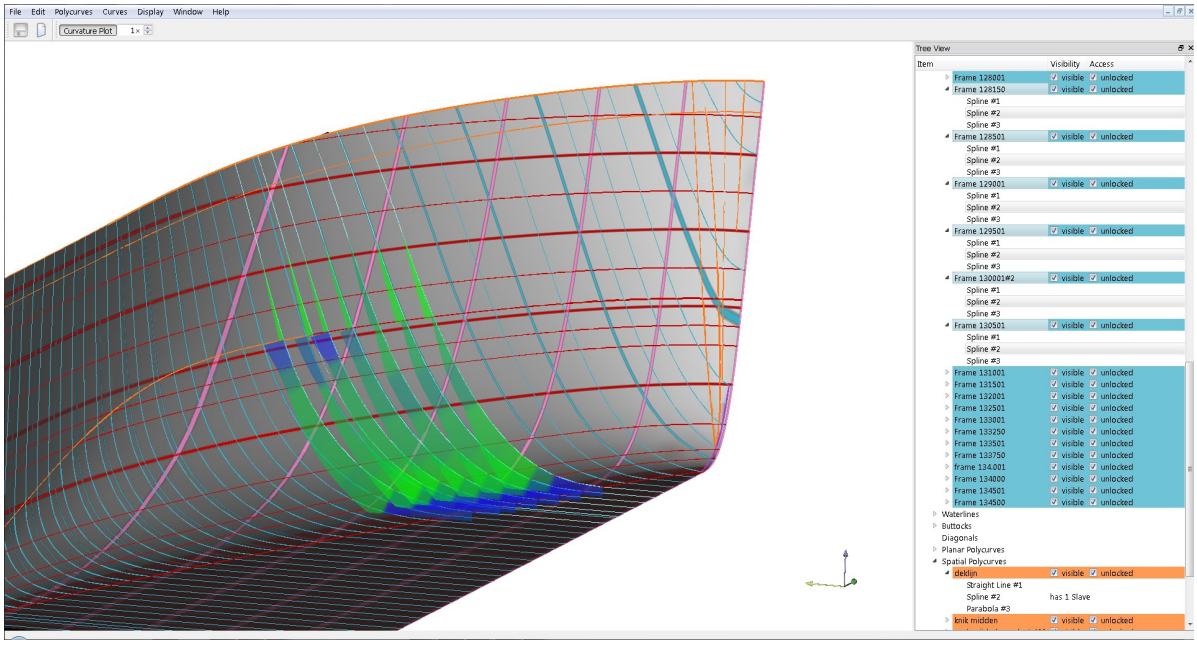

Fig. 4: Rendered shell and a sequence of selected frames with curvature plots.

    - Draggers indicate available degrees of freedom.
    - They have a protection against uncontrolled translation in or out of the screen, when one of its axes or planes gets close to parallel with the viewing direction.
    - They are easy to grab with the mouse, as they are contained within something that resembles a soap bubble, which reacts to mouse clicks without obstructing the model.
- When no action is active, the left mouse button is in selection mode. Points and vertices become available on selected curves, and a click on them restarts the tool that was last used for manipulation. (Previously one needed several functions to accomplish this.)
- The middle button, combined with modifier keys, provides all rotation and panning functionality, which is available at any time. (Which was not the case previously.)
- Curves light up when the mouse pointer is moved over them and show the following additional information instantly:
    - The name of the part it belongs to in the status bar
    - A curvature plot with colour-coded gradient
    - The position of spline vertices
    - Knuckles
    - The position and weight of fairing points
    - The direction of specified tangents

  For all these things one previously had to use one or several functions.
    - This *pre-lighting* indicates what will be selected upon a mouse click. Because the mouse pointer does not need to be exactly over the element, this visual feedback reduces the precision-requirement of hand movements.
    - The active action can control which items are selectable, so that illegal selections cannot be made. A different *pre-light* colour provides early warning to the user and a status message may explain the situation. This saves many a pop-up box.
- A tree view presents the structure of the model, showing solids, polycurves and curves, and some of their properties like visibility, lock status and master-slave connections. Selections may also be made here.
- All open modelling windows are updated simultaneously, displaying selections, curvature plots and draggers. All windows are active at the same time, the user may start in one and continue in another. (This may seem obvious, but is a real improvement over the previous generation.)
- We have support for SpaceNavigator devices from 3Dconnexion[7] (Fig.5).
- There is no command line. We believe that our action panels are so powerful and intuitive that you can do anything you would be able to do on a command line, only better.

5. Future

Even though there is no pressing need, once the new generation is feature-complete and stable, we can start tripping away the unused Pascal code of the previous generation. This will be much easier then than when we started, because it is clear what code is still in use and what code has become redundant. The cause for the earlier entanglement with other code, like pop-up dialogues initiated by computational procedures, will have been removed by the new interface, and therefore the presence of remote code in the DLL, like Ywin, will be obsolete.
As both Qt and Coin3D support multiple computer platforms, it is only the Prospero-compiled DLL that keeps us tied to MS Windows. With a reduced Pascal core we can consider switching to another compiler that does support multiple platforms, like gpc, as the number of incompatibilities that require porting will be much easier to handle. Then the road will be open to both 64bit binaries and platforms like Mac and Linux, should the customer desire.
But our biggest wish is to extend the fresh Fairway with new exciting features, like spatial deformation mentioned earlier, and other ideas that we like to reveal at a later opportunity.

7 http://www.3dconnexion.com/

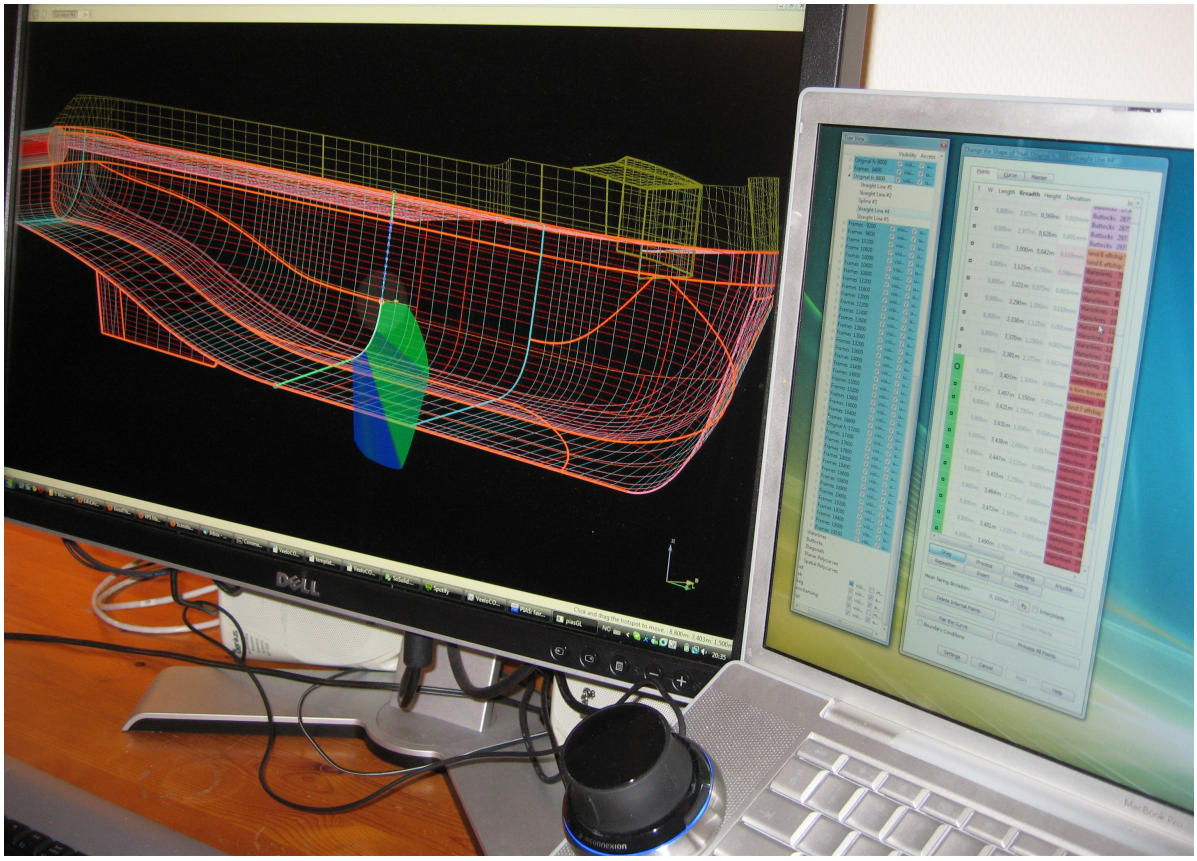

Fig. 5: Multi-monitor setup with detached windows for maximal work area.
A 6-degrees-of-freedom navigation device in the foreground.

6. Conclusion

One might suspect that by keeping the computational core and replacing "just" the user interface one would produce an application that resembles much its predecessor in how it functions and is operated, only with a shiny appearance and probably with some visual effects. Instead, the change has enabled us to redesign the way we work with the program, providing a completely new user experience. Much of the empowerment is due to the third-party frameworks that we are using. It has been, and still is, a fun project to work on, albeit not without some hard moments. These are the lessons that we have learned:

1. Be strict in separating algorithms and interfaces. This counts for interfaces to devices (printers, plotters, sensors), to program applications (dedicated: fairAPI; generic: agent-based), and to humans (GUI). No pop-ups, ideally not even strings. Just codes, if you have to.
2. Implementation takes twice as long as you planned; even though you knew that and planned for double the time you thought the job would take.
3. Favour third-party libraries that provide access to its source code above the ones that don't.
4. Anticipate changes in technology and design for flexibility. We are not good at predicting the future and decisions are likely to be suboptimal. Doing the best you can is still better than programming for the present only.
5. Develop in iterations, covering a broad set of systems progressively implementing details, in order to recognize problem areas early.
6. Give up in time. Start over.
7. Value the interaction between the user and the developer. You can't design a system like this the way they teach Programming Methodology in university: Analysis → Design → Implementation → Validation. It is more a process of investigating what is wanted and investigating what is possible and then have the two grow closer in rounds of iteration. What

is wanted is affected by the possibilities and new possibilities are discovered in response to what is wanted. In the end you get more than you thought you wanted and thought was possible.

References

KOELMAN, H.J. (2003), *Application of the H-rep Ship Hull Modelling Concept,* Ship Technology Research 50/4, pp.172–181

VEELO, B.N. (2004a), *Shape Modification of Ship Hulls in H-rep,* Ship Technology Research 51, pp.162–172

VEELO, B.N. (2004b), *Variations of Shape in Industrial Geometric Models,* doctoral thesis Norwegian University of Science and Technology, NTNU.