

The Potential of Free Software for Ship Design

Bastiaan N. Veelo, NTNU, Trondheim/Norway, Bastiaan.N.Veelo@ntnu.no

Abstract

This paper considers an unorthodox approach to software development and licensing, called Free Software, Libre Software or Open Source Software (recently abbreviated as FLOSS), with a focus on its potential in naval architecture. It is argued that this approach will increase productivity of programming efforts related to ship design, and increase the use/development ratio of the resulting software. Advantages and disadvantages of the approach are discussed, and an indication of how FLOSS for ship design can be initiated is given.

1 Software for a Special Purpose

Maritime industries put special demands on computer applications, as the existence of this conference illustrates. Advanced general purpose engineering design software (covered by the term “Product Life-cycle Management”, PLM) is successful in many other branches of engineering. But, despite their level of sophistication, they often fall short of *our* demands. Of course, computer applications exist that address some of our special needs, but because the user-base, or market, is relatively small, these products usually have one or more of the following disadvantages:

1. Limited functionality; especially functionality that is common in PLM applications may be lacking.
2. The user interface that we are presented with may not be as productive as the interfaces that most of our colleagues in general engineering enjoy.
3. High licence fees for the user.
4. Low profitability for the producer.

Because of these disadvantages and the shortcomings of mainstream CAD systems, it happens that custom software is developed in-house by an able naval architect, probably to address a small number of issues initially. An important advantage of such an effort, besides now being able to address particular issues, is *access to the source code*. Let us have a quick look at the advantages of source access.

1.1 Access to Source Code

Source code is of course the textual origin of software, that can be “understood” by both humans and machines. It is what designers and implementors of software (the programmers) produce, and what machines can compile into a binary format, which then can be run on computers to execute certain tasks. The binary format is machine-readable only, so the internal workings of the software are hidden from humans. Most software that you buy does not actually become your property, but you buy a license to use the software in its binary form. Any attempt to “reverse engineer” its inner workings is often explicitly prohibited. The knowledge that is represented by the source code of the program is a secret that is fiercely protected by its licensor, and as a licensee you cannot extend or evolve the software.

Access to source code, on the other hand, means you have access to the inner workings of the program. Besides the fact that this information is valuable in its own right, it also gives you various privileged abilities, provided you have the required technical skills (which you are likely to have if you produced the source):

To fix bugs. If you discover a bug in some software to which you only have access in binary format, the only thing you can do is report the bug to the vendor, wait for the next version to be released

and pay an upgrade license. This new version may or may not fix the bug, and almost certainly introduces new ones. With access to the source code you can track down the problem yourself and most probably fix it yourself, here and now.

To inspect it. With access to the source code, you can assess the correctness of its algorithms, and with it, the validity of the results that it produces.

To customise it. The software may have been written to assist in the solution of one specific problem. At a later point in time, you may encounter a problem that has similarities with the earlier one. With access to the source code, you can adapt the software to also assist in the solution of this new problem.

To extend it. With access to the source code of a program, you have the possibility to extend it so that it can perform more diverse tasks for you.

To evolve it. No non-trivial computer program is perfect. There is always room for improvement, e.g., in the user interface, in the efficiency of its algorithms, in the accuracy of its output, to support new hardware or to make use of performance features of new processors. Access to the source code gives you the opportunity to make general improvements, or even experiment with alternative approaches to the problem.

If you lack the technical skills to make use of these privileges, you can pay someone skilled, and still profit from the privileges. These privileges are so valuable that they sometimes are the main motivator for an in-house programming effort (*Mutu, 2003*).

1.2 Parallel Efforts

Raymond (1999) gives “empirical evidence that approximately 95% of [source] code is still written in-house”, and one may not be exaggerating much by saying that all ship design software available today started out as an in-house or even personal project. However, few of these initiatives grow out to produce a tool that others are willing to pay for. As software development usually does not belong to the core activities of a design office, there are not enough resources to raise the software to production quality¹, nor for support and maintenance that commercial distribution of the software would require.

Therefore, *there is reason to believe that, on a global basis, there is a considerable programming effort that is not exploited to its full potential*. Now, if you are responsible for some of this effort, what can you do to make it more worthwhile?

1.3 The Conventional View Reviewed

One way of seeing it, is that in-house developed software is an asset of very high value. Indeed, the investment in programming effort may be colossal in relation to your selling business. But since you are not licencing the software to others, it does not generate direct income. Obviously, the software is of great value to you in the work that you do. You may then conclude that, because of the investment and of its value to you, it is important to keep the software to yourself. In other words, you assume that the secrecy of your software increases the competitive position of your selling business. However, chances are that your competitors also use in-house developed software, and that the real advantage of mutual secret efforts is relatively limited.

So what are the alternatives?

Give the software away in binary form. As you are not making any money on the software, and its value for your competitive position is debatable, one option could be to give it away. You can try

¹“Production quality” in this context is a term to indicate a level of quality that makes software sell-able and involves stability, efficiency, ease of use, user manuals etc. In-house developed software can be used for production in-house, even if it is not of production quality.

to create a small revenue stream by making it “share-ware”, where people can pay a fee to make an advertisement notice go away. But, as the state of the software is below production quality, it is not likely to be of much use to anybody outside your department, not without the support that you are not interested to provide, and few will think it is worth paying the share-ware fee. The benefits for you are practically zero, unless you manage to turn it into a marketing stunt and attract attention to your selling business, and for others the benefits are close to zero. Mind that you should disclaim any warranty for the correctness of the software and the validity of its output, which you cannot afford without the software creating revenue, so serious application of the software is not attractive for third parties.

Publish the source code. This option includes the former one, because anyone will be able to compile the software into binary form. It is important to note that source code is automatically protected by international copyright law, as per the Berne Convention of 1886, meaning that no registration is required, nor is the inclusion of a copyright notice. No-one is allowed to make copies of source code or make changes without permission of its originator.

With the availability of the source code, third parties can now inspect it and assess the correctness of the software, and theoretically decide to use it to solve real-world problems. However, source code analysis requires a considerable effort, and copyright law prohibits third parties to make use of any of the other advantages mentioned in section 1.1, meaning that they will have to accept a low level of quality and lack of vendor support. The protected advantages may still be important enough for others to engage in parallel programming efforts. There are no real benefits for you.

Put the source code into the public domain. To unlock all advantages of source code access to everybody, you could explicitly give up on the copyright, and put your work in the public domain. This has the potential of greatly increasing the use of your programming effort, and eliminates the motivation for any parallel effort. Some people do this, simply because they like to see a wide adoption of their contribution. There may be some secondary economic advantages (*Raymond, 1999*), but the economic *disadvantage* can be far greater. Anybody or any corporation may legally take your code and build a proprietary computer programme on it, or integrate parts of it into existing proprietary software. By giving up on the copyright completely, you allow others to make direct profit off your effort for no compensation, and you might not even know.

Release the source code under relaxed copyright. Most commercial software licenses impose further restrictions, for which an agreement is necessary between licensee and licensor. But it is equally possible to formulate a license that states additional *freedoms*, relaxing the protection of the copyright, in exchange of specific obligations by the licensee. The license can state itself to be void in case any of the obligations are breached, which effectively turns the act into a violation of copyright law. For someone that does not accept the license and the obligations, the work is automatically protected by ordinary copyright. For people who do accept the license, no signed agreement needs to exist for the obligations to be binding.

This mechanism can be used to give others access to all advantages mentioned in section 1.1, and at the same time to prohibit corporate “hijacking” of your code. This can be achieved by requiring any derivative work or modification of your code to be released under the same license as you did. Suddenly, for you as the copyright holder, this implies a great advantage: any improvement that anybody makes to your work, you may incorporate. *This opens an enormous resource that has the potential to increase the quality of your software to production level and evolve it into a more powerful tool almost for free.* This is the mechanism that makes Free/Libre and Open Source Software successful.

2 Definition of Terms

The English language has difficulties to put a name on software that is licenced under relaxed copyright. The term “**Free Software**” has been used for many years now, and is often associated with a man called

Richard Stallman. In 1984, Stallman launched the GNU project (www.gnu.org) to develop a free operating system. GNU is still an important part of the now widely distributed free operating system that is informally called Linux². In 1985, Stallman founded the Free Software Foundation (FSF, www.fsf.org), to promote computer users' rights to use, study, copy, modify, and redistribute computer programs. Both Stallman and his projects still play an important role in the Free Software movement today.

However, there is an ambiguity in the word “free” because it has different meanings when used as in “free beer” and as in “free speech”. In practice, both meanings often apply to Free Software, but the price of Free Software is not really important. You may sell Free Software if you can. The latter meaning of the word “free” is about freedom and liberty, which really is the essence of Free Software.

Unfortunately, most people think of economics when they think of something free, and you always need to disambiguate the term when you speak of Free Software. It is possible that therefore “Free Software” does not appeal to business people. This has been considered to be a problem, which has motivated another movement, the Open Source Initiative (www.opensource.org), in 1998. The Open Source Initiative (OSI) is a marketing program for Free Software, and aims to advocate it on solid pragmatic grounds. It has registered the term “**Open Source**” as a trade mark, and popularised it as a synonym for “Free Software”. OSI certifies software as Open Source, and carries a list of approved licenses (amounting to almost 60 licenses at the time of this writing).

Although the term “Open Source” is probably the most widely used term today, it is not without its problems either. Some people feel that it does not cover the essence, i.e., freedom, well enough (*Stallman, 2002*). Indeed, sometimes software is claimed to be Open Source, whereas it really only is published source code, without the copyright having been relaxed. As a matter of fact, OSI clarifies that “Open Source doesn't just mean access to the source code” and gives a detailed definition of the term, consisting of ten criteria that software must comply with before it may be called Open Source. These criteria differ slightly from the ones that the FSF practices; some of the license restrictions that the OSI accepts for Open Source are too restrictive in the eyes of the FSF (*FSF, 2001*).

During a translation of the English text of the most important Free Software license, the GNU General Public License (GPL), into Spanish, it appeared that Roman languages do not suffer the ambiguity problem (*González-Barahona, 2004*). The word “free” with respect to price is translated into Spanish as “gratis”, and “free” with respect to liberty is translated as “libre”. Because of the problems with the terms “Free Software” and “Open Source”, some people have started to use the term “**Libre Software**”, even though this is a mix of languages. This term is gaining popularity, notably in official circles in Europe. For example, in 1999, the Information Society Directorate General of the European Commission initiated the European Working Group on Libre Software (eu.conecta.it).

In 2002, the acronym **FLOSS** has been introduced, in a survey and study commissioned by the European Commission (*Ghosh and Glott, 2002*), to cover all of Free/Libre and Open Source Software at once.

It is important to note that both Free Software and Open Source do not differentiate between the last two options of section 1.3. That is, both Free Software and Open Source software recognise source code that has been put in the public domain, because everyone is free to use such software in any way they like. Software that is guaranteed to be FLOSS in all its derivative variants, by means of a clause that makes derivative works inherit the licence of the original and the requirement that source code always must be made available, is often identified with the term “**Copyleft**” (as opposed to “Copyright”). In general, the advantages of FLOSS are considered to be highest under a Copyleft license.

2.1 FLOSS Licences

To give an exhaustive list of officially recognised Free/Libre and Open Source Software licenses is far beyond the scope of this paper. For this the reader is referred to the sites of the FSF and OSI, who maintain extensive lists of approved licenses (www.gnu.org/licenses/license-list.html and

²To be precise, Linux is actually only the kernel in this operating system.

www.opensource.org/licenses/). However, the important question is not whether any particular license is approved by either organisation. The important question is whether a piece of software that you are interested in is licensed under terms that you can accept, and for this you should always consult the actual license text that follows with the software. The other important question is which license to choose or construct when you consider to write FLOSS or consider to release existing work as FLOSS. To help answer the latter question, both organisations have put together guides and “HOWTOs”, see (www.fsf.org/licensing) and (*Raymond and Raymond, 2002*).

For completeness however, we briefly explain the licenses that are used for relevant FLOSS as listed in the appendix. Both the **MIT** and **BSD** licences allow almost everything, except removal of the license statement and copyright notice. They also disclaim warranty and liability, and the BSD licence adds a non-endorsement clause. The following licenses are all Copyleft. The most widely used license is the **GNU GPL** license, or General Public Licence. Proprietary software can not include or even link with GPLed code. The **QPL**, or Qt Public License, is rarely used, and is comparable but incompatible with the GPL. The **GNU LGPL**, or Lesser General Public Licence, is similar to the GPL but allows LGPLed libraries to be linked to by proprietary software. The **wxWindows** Library Licence is identical to the LGPL, however with an exception that states additional freedoms for binary distribution.

3 The Case for FLOSS

Due to space limitations, this paper is not going to do enough justice to the case for Free/Libre and Open Source Software. Many essays, articles and even books have been written on the subject by far better analysts and FLOSS advocates. For example, “The Cathedral and the Bazaar” by *Raymond (1999)* is very much worth a read, especially the online version that contains references to comments and replies by others. The information on the OSI website is a very good place to start, because it concisely discusses the case for FLOSS from the viewpoints of business management, users and of developers individually (www.opensource.org/advocacy/). The phenomenon of FLOSS is actively being studied, and MIT carries an impressive list of online articles from the FLOSS research community (opensource.mit.edu/online_papers.php). Large scale surveys of the FLOSS developer community have been performed by *Gosh and Glott (2002)* and *David et al. (2003)*, in which respectively 2784 and 1588 developers participated.

Especially the OSI helps you weigh the advantages and disadvantages between an open development model and a closed development model. Here it is often assumed that you are able to extract direct revenue out of your proprietary software, or in *Raymond’s (1999)* words “collect rent from your secret bits”. The OSI explains that even if this is the case, the pay-off of converting to FLOSS can be higher than the pay-off of remaining in proprietary mode. However, that consideration is outside the scope of this paper, and we will proceed with the simpler assumption that you are the only user of your in-house developed software.

There are three main advantages of writing FLOSS as opposed to writing software in-house. Firstly, you get valuable peer review. Secondly, you get to use free building blocks. And lastly, you work in an organisational mode that is arguably the most effective for the management of complex systems (*Raymond 1999*).

3.1 Peer Review

The peer review helps improving the quality and feature-richness of your software. Because the software is available freely, a community of “beta testers” (your users) will connect with you and help you find bugs. And because they have access to the code and are allowed to play with it, they will send you bug fixes if they can, in exchange for just gratification and recognition. For the same reasons, you may get sent patches that add new features to your software.

3.2 Free Building Blocks

Because you are writing FLOSS, you are free to build on existing FLOSS. You can do that by linking with one or more of the many excellent libraries that exist for different purposes, and thereby use the peer reviewed work of others instead of reinventing the wheel yourself. Also, instead of writing a new computer program from the ground up to help do your work, you may be able to find an existing project, either fully functional or in early stages of development, that you can extend to make it satisfy your requirements. In these ways you can have a flying start and reach your goal much sooner than when doing it all alone.

The risk of depending on third party code that exists in a proprietary setting, namely that the software may disappear because of a business situation with the vendor, is non-existent with FLOSS. If a piece of FLOSS gets dropped by its originators, you are free to pick it up and maintain and evolve it yourself. Therefore, a FLOSS project will thrive as long as there are people interested in it. The same advantage applies to the accessibility of data files. The information in old data files, written in a proprietary format with a proprietary program that is not available anymore, is inaccessible and effectively lost. But if you have the source code of the application that wrote the data file, you can understand the format and extract the information.

3.3 Your Competitive Position

If you are still reluctant to publish work from which your competitor may benefit, you may consider the following. Although maritime industries increasingly play on a global market, most of your business likely still relates to a certain geographic area. However, FLOSS development takes place on the Internet, where location is irrelevant. Whilst you compete more or less locally, you will collaborate globally in FLOSS development. Also, even when your nearest competitor starts using your software intensively, he will make improvements and he will have to contribute them back to you and make *your* software better. Therefore, it is important to be the first to embark in FLOSS mode, because it gives you more control of the situation. You still own the software, and it is you who decides where to take it. *Raymond (1999)* also says about this:

[T]here's a serious opportunity risk in waiting too long: you could get scooped by a competitor going open-source in the same market niche.

The reason this is a serious issue is that both the pool of users and the pool of talent available to be recruited into open-source cooperation for any given product category is limited, and recruitment tends to stick. If two producers are the first and second to open-source competing code of roughly equal function, the first is likely to attract the most users and the most and best-motivated co-developers; the second will have to take leavings.

So yes, your competitor may benefit from your actions, but you may benefit more.

3.4 Invitation to Take the Lead

If you are a scientist, commercial competition may not be so much of an issue, and with that the choice for FLOSS should be easy. FLOSS poses great opportunities for learning, and universities are encouraged to take the lead in developing FLOSS, also for maritime applications.

3.5 Play the Game Well

Before jumping on the FLOSS bandwagon in all enthusiasm, be sure to be prepared for the rules of the game. FLOSS development is a voluntary community effort, and people skills are essential when leading one. This is especially the case because the Internet is a poor medium for social communication. In order to make FLOSS development work for you, you need to understand the process very well, or have a natural feeling for it. Again, the writings of *Raymond (1999)* can be a source of insight.

4 Conclusion

This article has discussed how Free/Libre and Open Source Software can be an attractive alternative to in-house developed software for ship design. The freedom to use existing FLOSS building blocks is an important motivator to get involved in FLOSS development. Therefore, this article has an appendix with references to selected FLOSS projects of various categories that are relevant to ship design. The use of one or more of these building blocks makes a flying start possible in the initiation of FLOSS for ship design. The appendix also shows that it is already possible to use FLOSS for some aspects of ship design today, and that you will not have to pioneer the concept. If you decide to get actively involved, you will not be alone.

References

- DAVID, P.A.; WATERMAN, A.; ARORA, S. (2003), *FLOSS-US — the free/libre/open source software survey for 2003*, Stanford Institute for Economic Policy Research. www.stanford.edu/group/floss-us/
- FSF (2001), *Categories of Free and Non-Free Software*, Free Software Foundation. www.gnu.org/philosophy/categories.html
- GHOSH, R.A.; GLOTT, R. (2002), *Free/libre and open source software: survey and study*, International Institute of Infonomics, University of Maastricht, The Netherlands; and Berlecon Research GmbH, Berlin, Germany. www.infonomics.nl/FLOSS/
- GONZÁLEZ-BARAHONA, J.M. (2004), *Quo vadis, libre software?*, Universidad Rey Juan Carlos, Spain. sinetgy.org/jgb/articulos/libre-software-origin/
- MUTU, V.; IONAS, O.; GAVRILESCU, I. (2003), *SURF — an in-house development for a ship hull design software*, 2nd Int. Conf. Computer and IT Applic. Mar. Ind., COMPIT, Hamburg, pp.387–399
- RAYMOND, E.S.; RAYMOND, C.O. (2002), *Licensing HOWTO*, draft Open Source Initiative working paper. www.catb.org/~esr/Licensing-HOWTO.html
- RAYMOND, E.S. (1999), *The Cathedral & the Bazaar*, O'Reilly, ISBN 1-56592-724-9, www.catb.org/esr/writings/cathedral-bazaar/
- STALLMAN, R.M. (2002) *Why “Free Software” is Better Than “Open Source”*, in *Free Software, Free Society*, GNU Press, ISBN 1-882114-98-1. www.gnu.org/philosophy/free-software-for-freedom.html

APPENDIX

A FLOSS Software

Examples of existing Free/Libre and Open Source Software are given in six categories: geometric modelling, visualisation, numerics, finite element method, computational fluid dynamics and graphical user interfaces. The information about each project also includes the latest version and release date at the time of this writing, which is an indication of the degree of maturity and the developer activity respectively. Note that the list is by no means exhaustive.

A.1 Geometric Modelling

A.1.1 Blender

Blender is a 3D graphics creation suite with a focus on the entertainment industry. Its modelling capabilities include polygon meshes, NURBS surfaces, Bézier and B-spline curves, meta-balls, and “Smooth

proxy” style Catmull-Clark subdivision surfaces with optimal iso-lines display and sharpness editing. It supports Python scripting for the creation of custom tools.

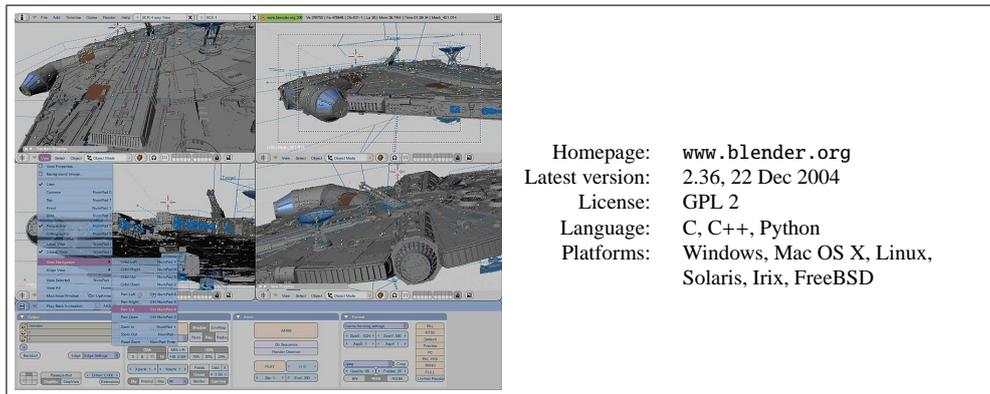


Fig.1: The Blender modeller

A.1.2 Wings 3D

Wings 3D is a polygon mesh modeller with a user interface that is easy to use for both beginners and advanced users. Wings 3D uses subdivision surfaces rather than NURBS surfaces.

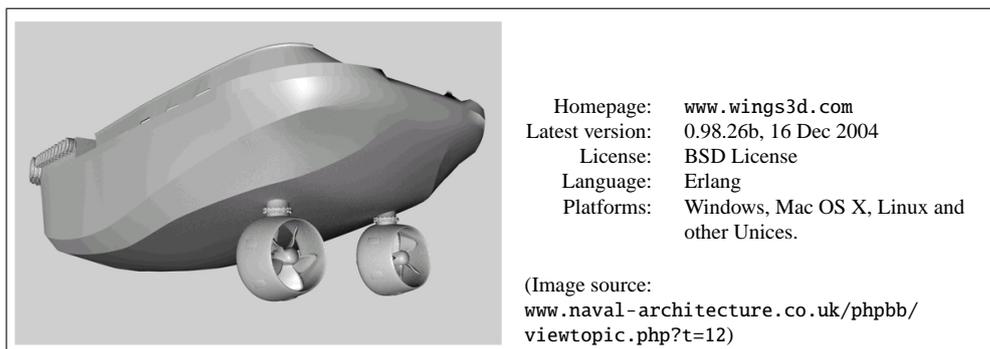


Fig.2: The Wings3D modeller

A.1.3 Coin3d

Coin3D is a set of libraries used for creating 3D graphics applications. Coin3D is fully compatible with SGI Open Inventor 2.1, the de facto standard for 3D visualisation and visual simulation software in the scientific and engineering community. Additional features in Coin3D include VRML97 support, 3D sound, 3D textures, and parallel rendering on multiple processors. See Fig.4.

A.1.4 Open CASCADE

Open CASCADE is an industrial Open Source alternative to proprietary 3D modelling kernels, a library that includes components for 3D surface and solid modelling, visualisation, data exchange and rapid application development. Open CASCADE Technology can be best applied in development of numerical simulation software including CAD/CAM/CAE, AEC and GIS, as well as PDM applications. The Technology exists from the mid 1990-s and has already been used by numerous commercial clients. See Fig.5.

A.1.5 FreeCAD

FreeCAD will be a general purpose 3D CAD system based on OpenCasCade. FreeCAD will aim directly at mechanical engineering, product design and related features (like CatiaV4 and V5, and SolidWorks). It will be a Feature-Based parametric modeller. Scripting is supported through Python.

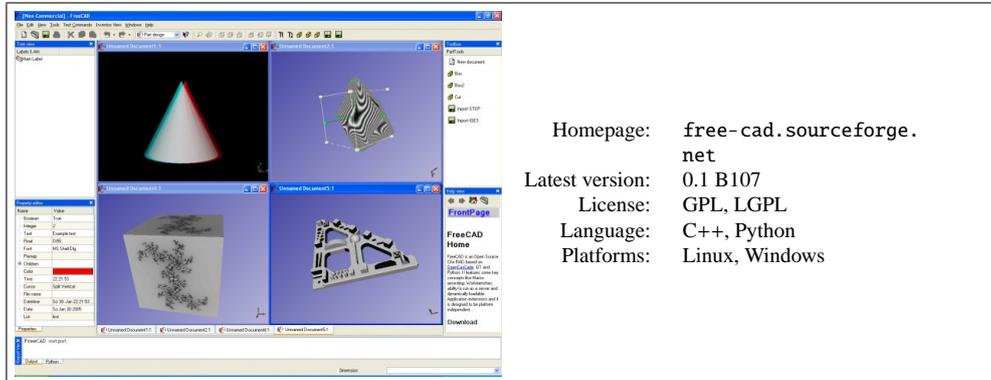


Fig.3: The FreeCAD system

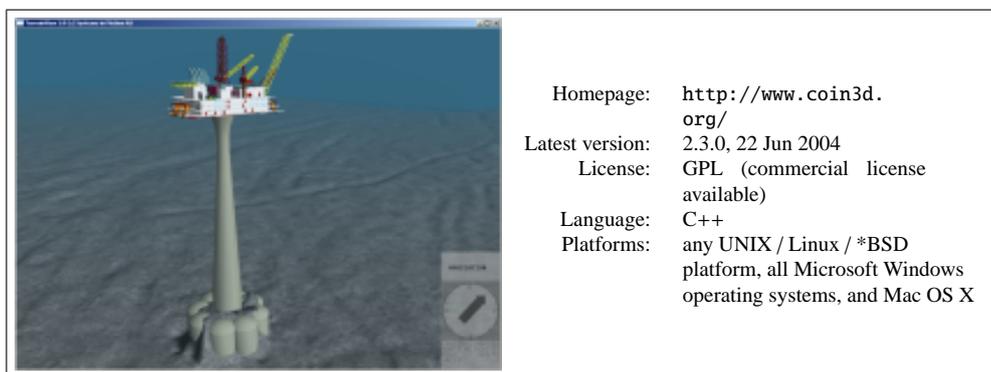


Fig.4: The Coin3D graphics library

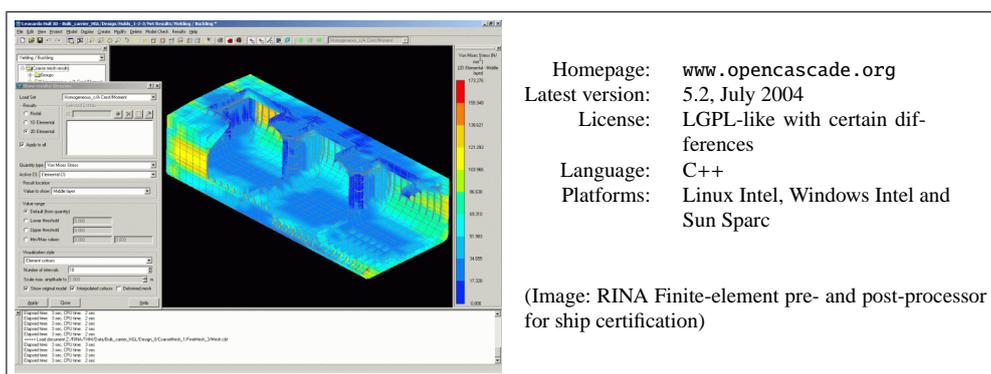


Fig.5: The OpenCASCADE surface and solid modelling kernel

A.1.6 BRL-CAD

A powerful constructive solid geometry (CSG) solid modelling system that includes an interactive geometry editor, ray tracing support for rendering and geometric analysis, network distributed frame-buffer support, and image and signal-processing tools.

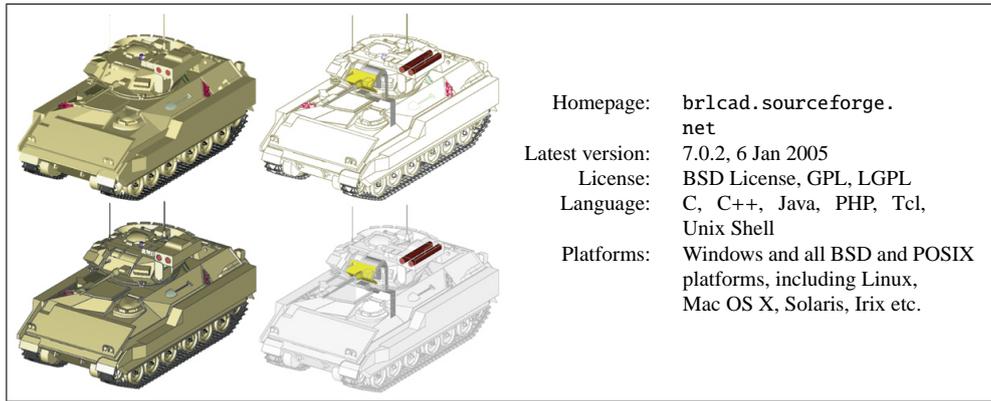


Fig.6: The BRL-CAD solid modelling system

A.1.7 Varkon

VARKON can be used as a traditional CAD-system with drafting, modelling and visualisation if you want to, but the real power of VARKON is in parametric modelling and CAD applications development. VARKON includes interactive parametric modelling in 2D or 3D but also the unique MBS programming language integrated in the graphical environment. VARKON is designed to be modified and extended with knowledge and functionality specific to a certain product or problem. It is not a true solid modeller.

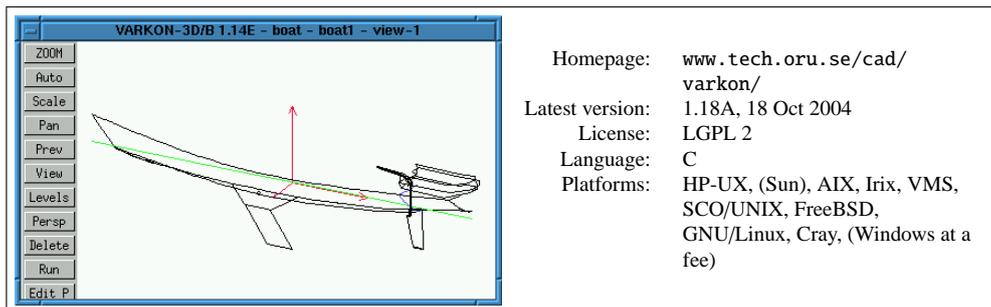


Fig.7: The Varkon CAD system

A.1.8 Sailcut CAD

Sailcut CAD is a sail design and plotting software. It allows you to design and visualise your own sail and compute the accurate development of all panels in flat sheets.

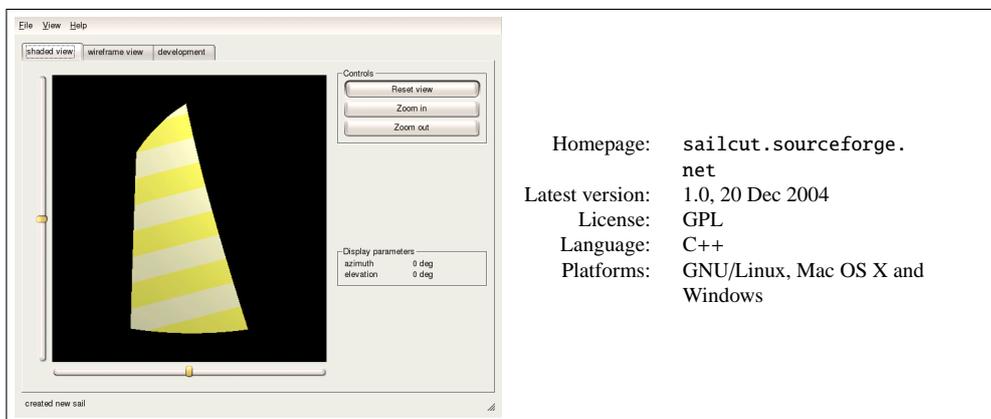


Fig.8: The Sailcut CAD sail design and plotting software

A.1.9 SketchBoard

SketchBoard is a sketch oriented CAD software. It is a tool for designers to develop their design by sketching and modelling in 3D. It has a system that helps converting the sketch into a polygon.



Fig.9: The SketchBoard CAD software

A.2 Visualisation

A.2.1 ADMesh

Processor for repairing, transforming and merging triangulated solid meshes. Reads STL file format (used for rapid prototyping applications) and writes STL, VRML, OFF, and DXF files.

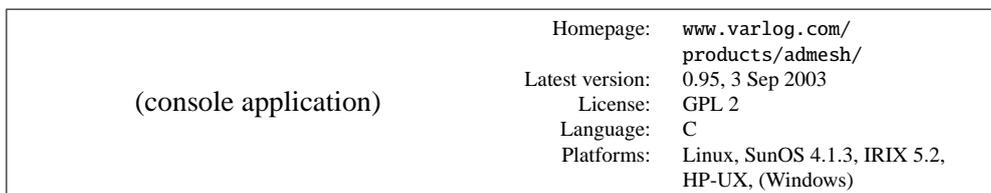


Fig.10: The ADMesh processor

A.2.2 Pixie

Pixie is a photo-realistic renderer that uses a RenderMan-like interface. Features include programmable shading, motion blur, depth of field, ray-tracing, scan-line rendering, occlusion culling, global illumination, caustics, etc.

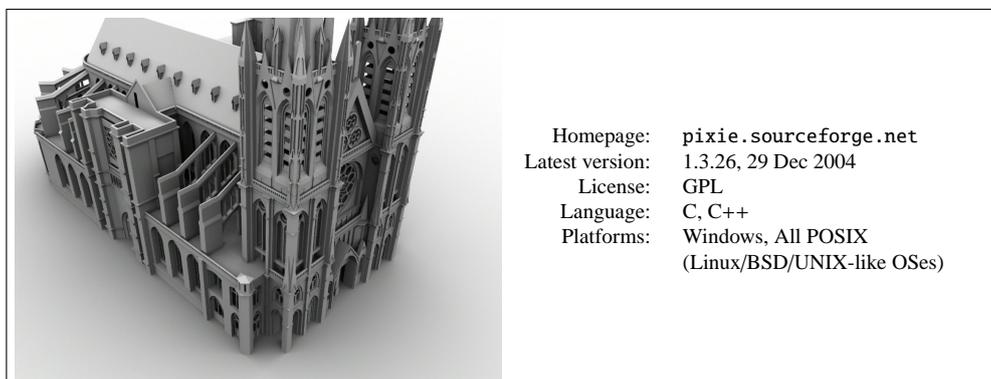


Fig.11: The Pixie renderer

A.2.3 YafRay

YafRay is a powerful ray-tracer. It enables you to create fantastic images and animations of a photo realistic quality. Thanks to its API (Application Programming Interface) and its modular structure, it is possible to develop rendering plug-ins, making it possible to use YafRay from any program or 3D suite. At the time of this writing, suites as Blender, Wing3D or Aztec take advantage of this feature.

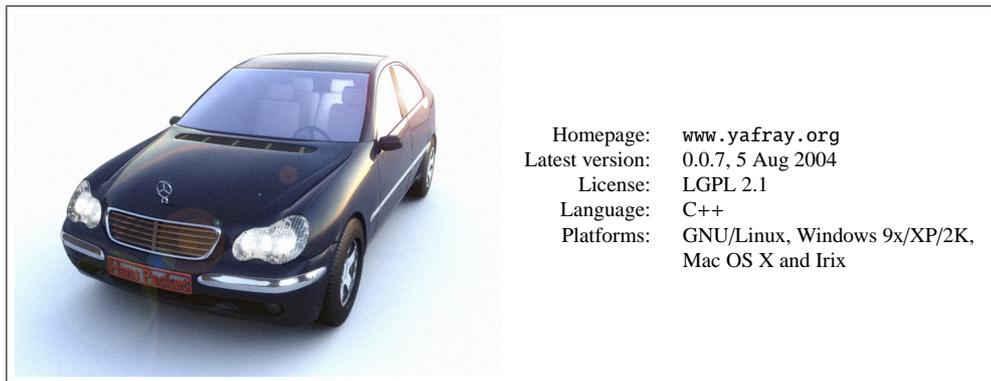


Fig.12: The YafRay ray-tracer

A.2.4 GeomView

GeomView is an interactive 3D viewing program for Unix. It can be used as a stand-alone viewer for static objects or as a display engine for other programs which produce dynamically changing geometry. It can display objects described in a variety of file formats.

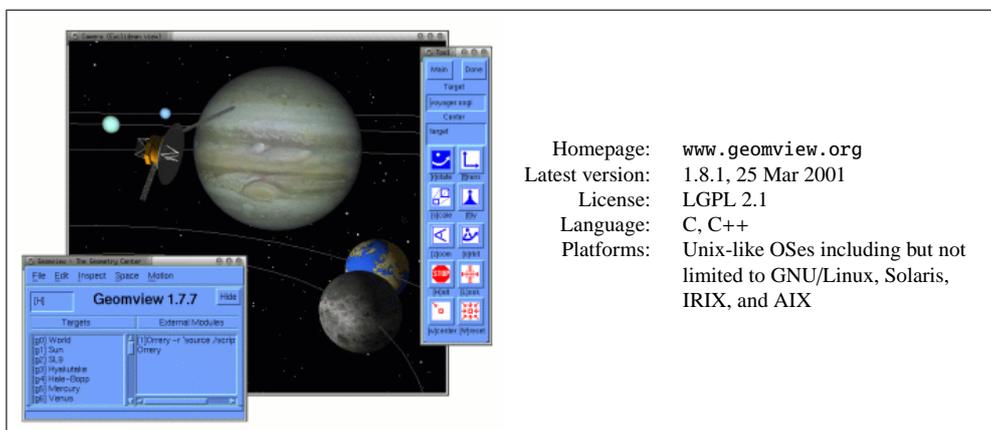


Fig.13: The GeomView viewer

A.2.5 MayaVi

MayaVi is a free, easy to use scientific data visualiser. See Fig.15.

A.3 Numerics

Beyond the software referenced below, there is a host of other powerful numerics libraries. Many of them are listed at www.oonumerics.org/oon.

A.3.1 Octave

GNU Octave is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with Matlab. It may also be used as a batch-oriented language.

Octave has extensive tools for solving common numerical linear algebra problems, finding the roots of nonlinear equations, integrating ordinary functions, manipulating polynomials, and integrating ordinary differential and differential-algebraic equations. It is easily extensible and customisable via user-defined functions written in Octave's own language, or using dynamically loaded modules written in C++, C, Fortran, or other languages.

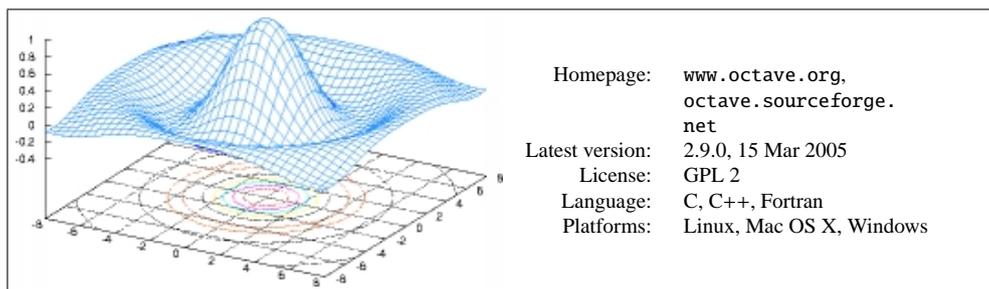


Fig.14: The Octave language (mostly Matlab compatible)

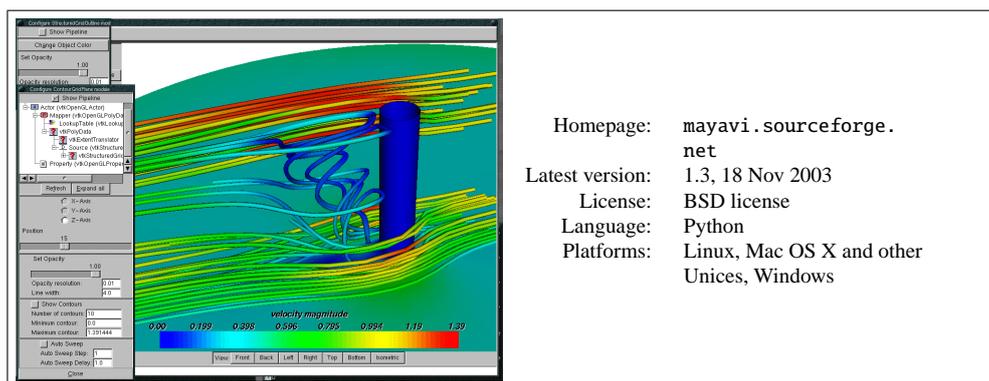


Fig.15: The MayaVi data visualiser

A.3.2 FreeMat

FreeMat is a free environment for rapid engineering and scientific prototyping and data processing. It is similar to proprietary systems such as MATLAB from Mathworks, and IDL from Research Systems. FreeMat includes several novel features such as a code-less interface to external C/C++/FORTRAN code, parallel/distributed algorithm development (via MPI), and plotting and visualisation capabilities. See Fig.17.

A.3.3 Maxima

Maxima is a fairly complete computer algebra system written in lisp with an emphasis on symbolic computation. Its abilities include symbolic integration, 3D plotting, and an ODE solver. See Fig.18.

A.3.4 ATLAS

The ATLAS (Automatically Tuned Linear Algebra Software) project is an ongoing research effort focusing on applying empirical techniques in order to provide portable performance. At present, it provides

C and Fortran77 interfaces to a portably efficient BLAS implementation, as well as a few routines from LAPACK.

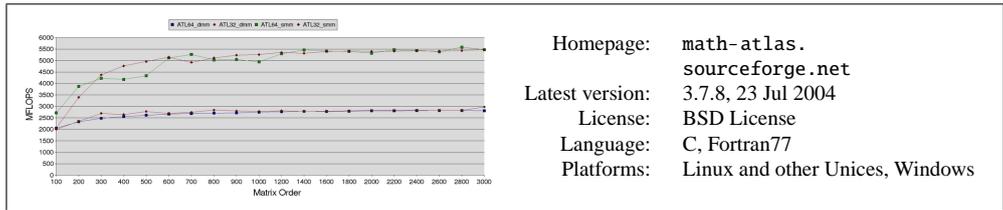


Fig.16: The Automatically Tuned Linear Algebra Software

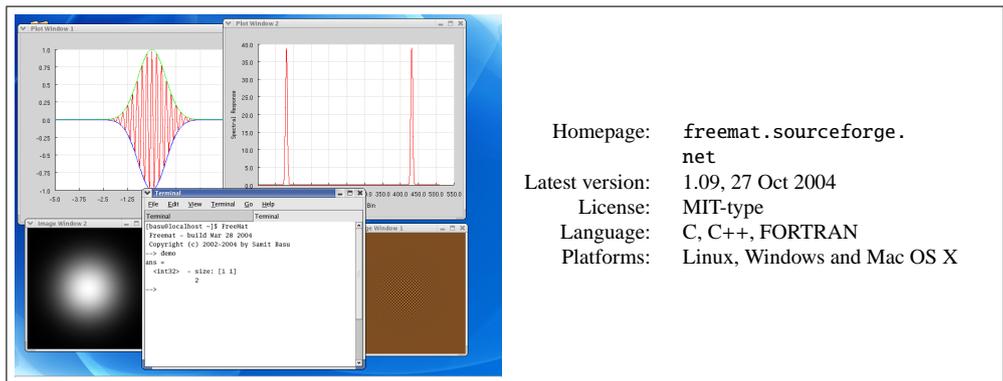


Fig.17: FreeMat, an other alternative to Matlab

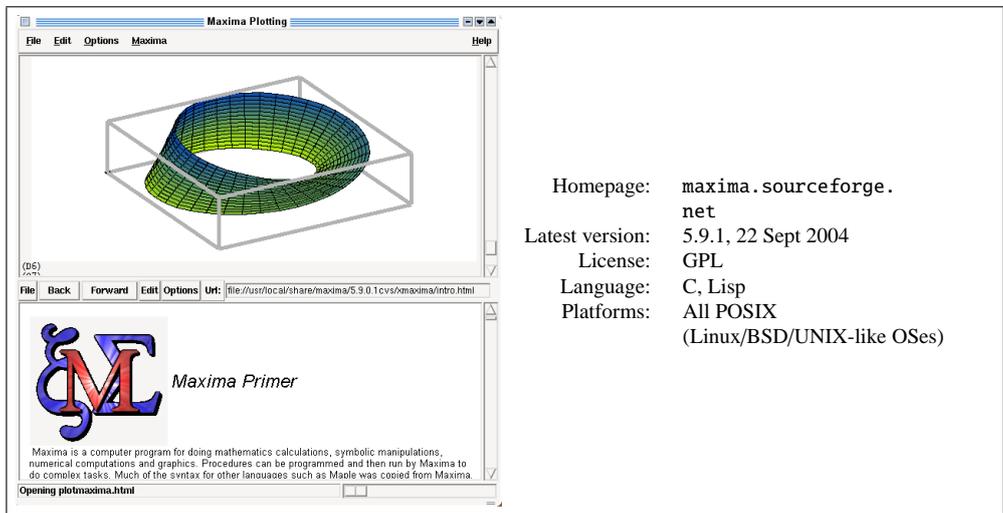


Fig.18: The Maxima computer algebra system

A.4 FEM

A.4.1 Gmsh (FEM Mesh Generator)

Gmsh is an automatic 3D finite element grid generator (primarily Delaunay) with a build-in CAD engine and post-processor. Its design goal is to provide a simple meshing tool for academic problems with parametric input and advanced visualisation capabilities.

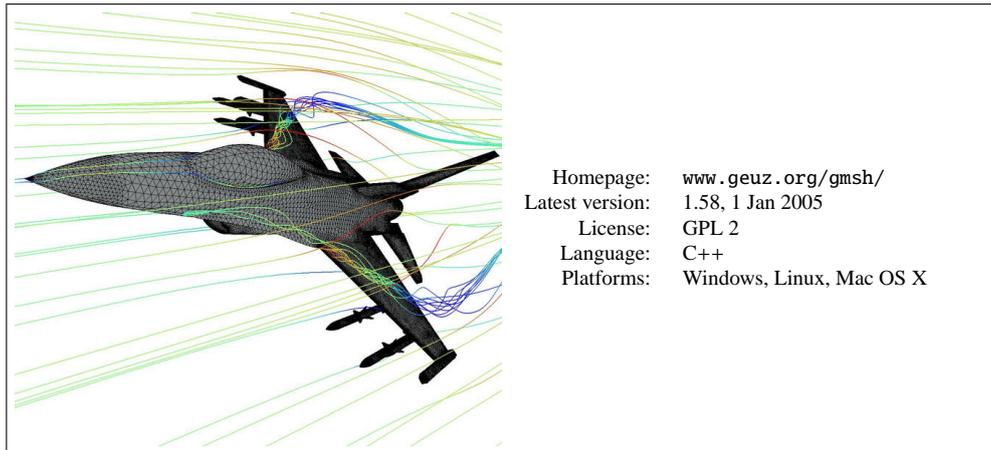


Fig.19: The Gmsh mesh generator

A.4.2 NETGEN — Automatic Mesh Generator

NETGEN is an automatic 3D tetrahedral mesh generator. It accepts input from constructive solid geometry (CSG) or boundary representation (BRep) from STL file format. The connection to a geometry kernel allows the handling of IGES and STEP files. NETGEN contains modules for mesh optimisation and hierarchical mesh refinement. Works together with NGSolve (see Section A.4.3).

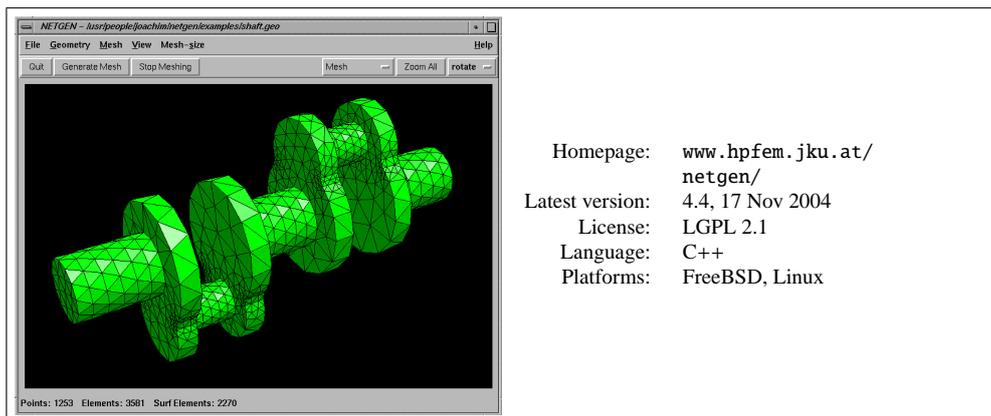


Fig.20: The NETGEN mesh generator

A.4.3 NGSolve

NGSolve is a general purpose 3D finite element solver, supporting boundary value problems, initial-boundary value problems and Eigenvalue problems for the available types of equations, namely scalar (heat flow), elasticity, and magnetic field. NGSolve performs adaptive mesh refinement, the matrix equations are solved by optimal order multi-grid methods. Works together with NETGEN (see Section A.4.2). See also Fig.22.

A.4.4 CalculiX

A Three-Dimensional Structural Finite Element Program designed to solve field problems. With CalculiX Finite Element Models can be build, calculated and post-processed. The pre- and post-processor is an interactive 3D-tool using the OpenGL API. The solver is able to do linear and non-linear calculations. Static, dynamic and thermal solutions are available. Both programs can be used independently. Because the solver makes use of the Abaqus input format, it is possible to use proprietary pre-processors as well.

In turn, the pre-processor is able to write mesh related data for Nastran, Abaqus and Ansys and for the free cfd code duns. A VDA CAD interface is available.

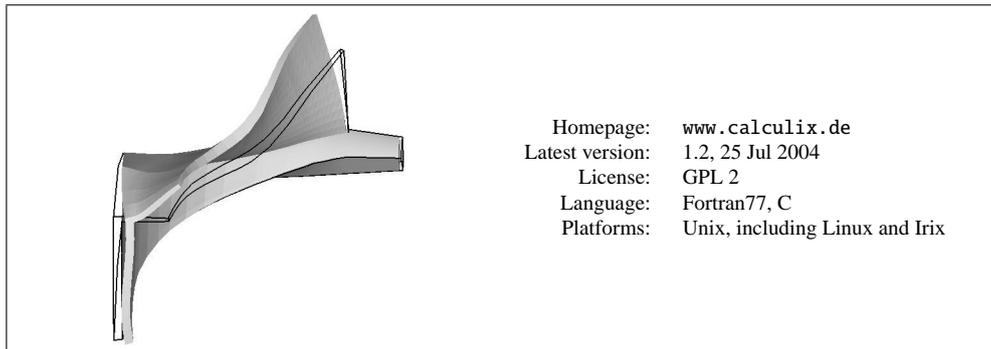


Fig.21: The CalculiX solver

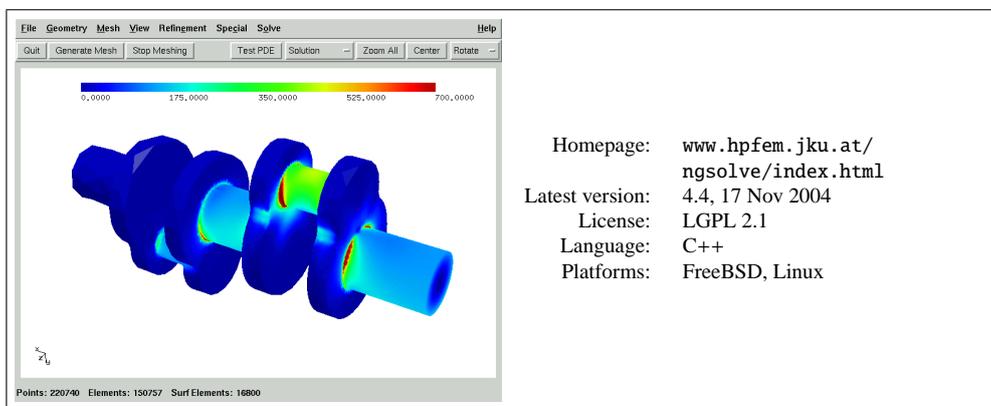


Fig.22: The NGSolve solver

A.4.5 SLFFEA

SLFFEA stands for San Le's Free Finite Element Analysis. It is a package of scientific software and graphical user interfaces for use in (non-linear) finite element analysis.

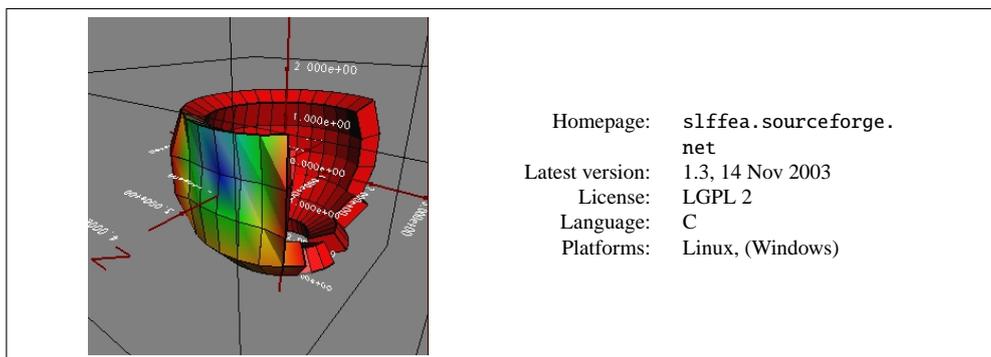


Fig.23: San Le's Free Finite Element Analysis

A.4.6 TOCHNOG

TOCHNOG is a feature-rich finite element program. Among the FE models supported are: differential equations (materials), convection-diffusion equations, Stokes and Navier-Stokes (fluids), elasticity (isotropy and transverse isotropy), plasticity (Von-Mises, Mohr-Coulomb, etc.; plastic surfaces can be

arbitrarily combined). Residues in equations and error estimates for all data can be printed or plotted using gnuplot/plotmtv, CalculiX or gmsh.

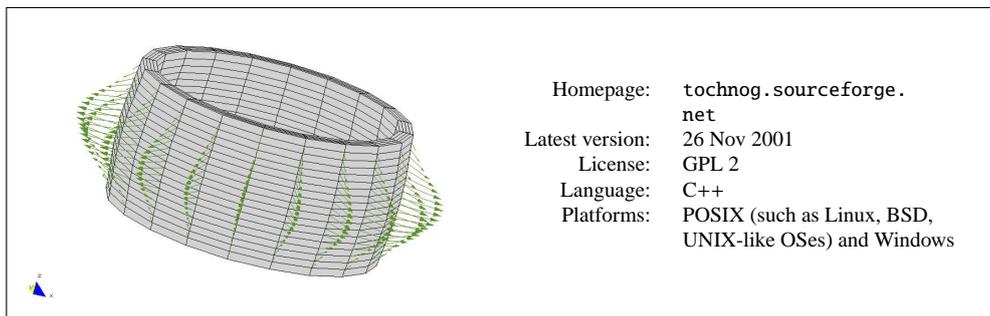


Fig.24: The TOCHNOG finite element programme

A.4.7 OFELI

OFELI (Object Finite Element Library) is a library of finite element C++ classes for multipurpose development of finite element software. It is intended for teaching, research and industrial developments as well.

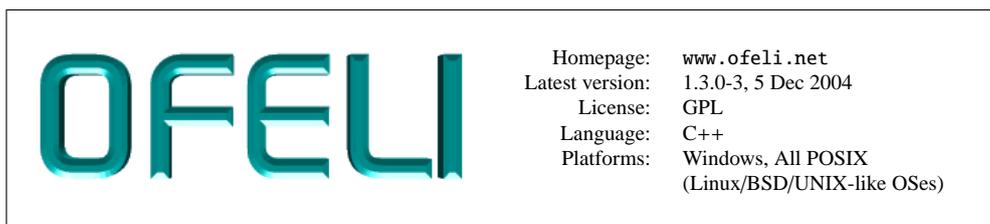


Fig.25: The Object Finite Element Library

A.5 CFD

A.5.1 FEATFLOW

FEATFLOW is both a user oriented as well as a general purpose subroutine system for the numerical solution of the incompressible Navier-Stokes equations in two and three space dimensions. It supports stationary and non-stationary problems.

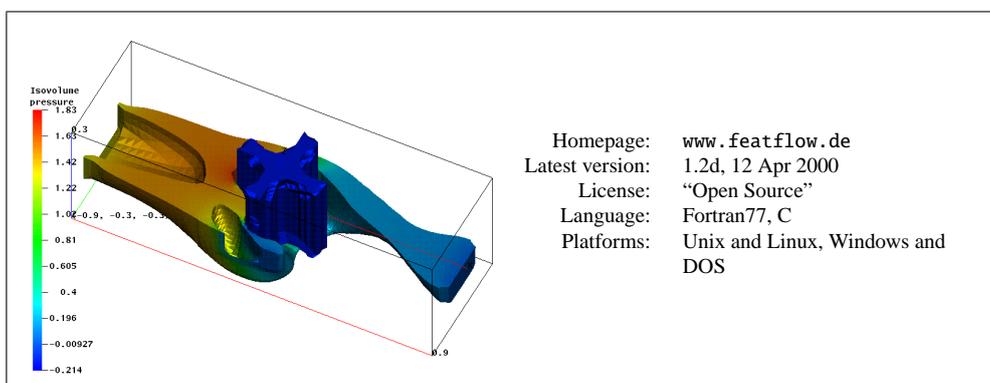


Fig.26: The FEATFLOW Navier-Stokes solver program and library

A.5.2 DUNS

The DUNS (Diagonalised Upwind Navier-Stokes) code is a 2D/3D, structured, multi-block, multi-species, reacting, steady/unsteady, Navier Stokes fluid dynamics code with q-omega turbulence model.

Homepage:	duns.sourceforge.net
Latest version:	2.7.1, 25 Sept 2003
License:	GPL 2
Language:	C, Fortran77
Platforms:	Linux and other POSIX compliant platforms

Fig.27: The DUNS code

A.5.3 OpenFlower

OpenFlower is an open source CFD software (FLOW solvER, literally) written in C++. It is mainly devoted to the resolution of the turbulent unsteady incompressible Navier-Stokes equations with a LES approach. It can deal with arbitrary complex 3D geometries with its finite volume approach.

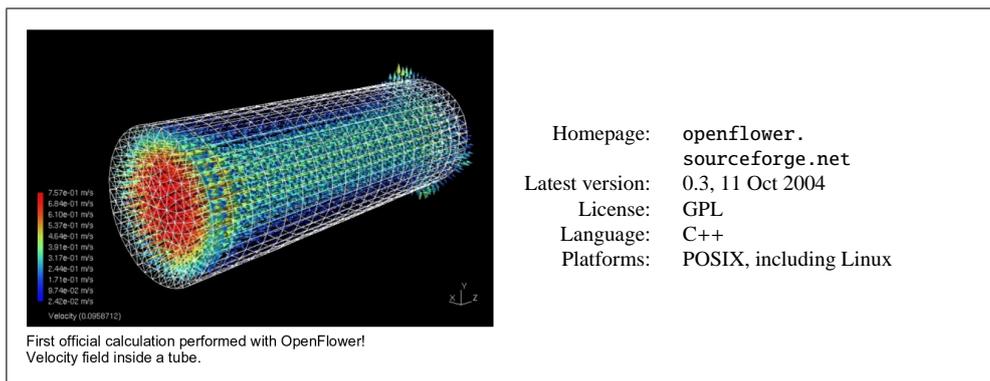


Fig.28: The OpenFlower CFD software

A.5.4 Gerris Flow Solver

Gerris is a tool for generic numerical simulations of flows, possibly in geometrically complex geometries and including adaptive, multi phase and interfacial flows capabilities.

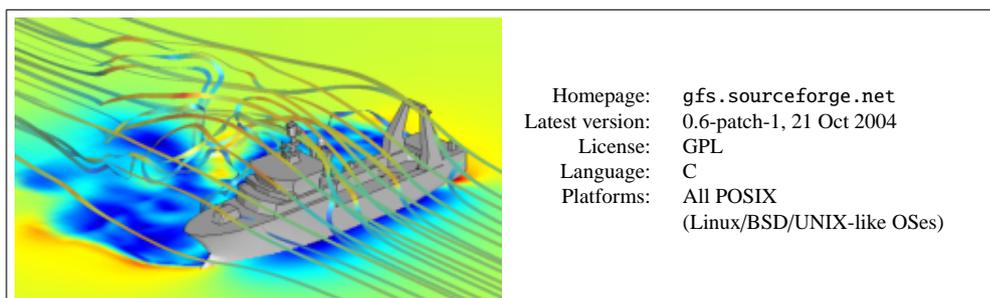


Fig.29: The Gerris Flow Solver

A.6 GUI

A.6.1 Qt

One of the key design goals behind Qt is to make cross-platform application programming intuitive, easy and fun. Qt achieves this goal by abstracting low-level infrastructure functionality in the underlying window and operating systems, providing a coherent and logical interface that makes sense to programmers. The Qt API and tools are consistent across all supported platforms, enabling platform independent application development and deployment. Qt applications run natively, compiled from the same source code, on all supported platforms. Qt is the basis of the “K” Desktop Environment for Linux (KDE).

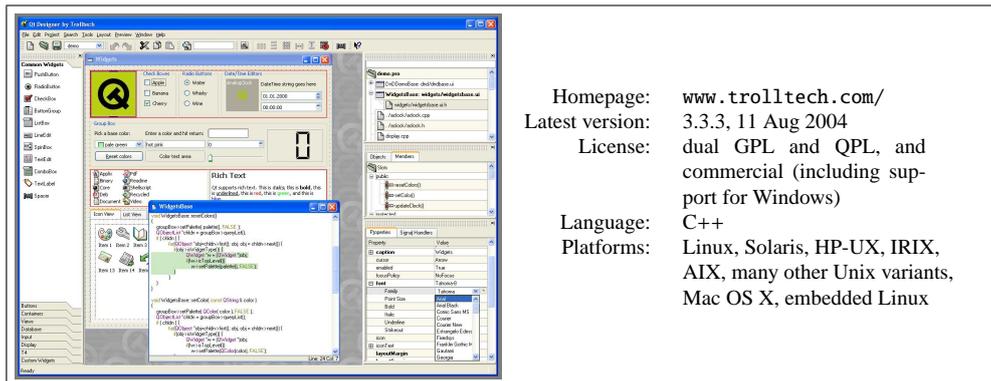


Fig.30: The Qt application framework

A.6.2 GTK+

GTK+ is a multi-platform toolkit for creating graphical user interfaces. Offering a complete set of widgets, GTK+ is suitable for projects ranging from small one-off projects to complete application suites. GTK+ has been designed from the ground up to support a range of languages, not only C/C++. Using GTK+ from languages such as Perl and Python (especially in combination with the Glade GUI builder) provides an effective method of rapid application development. GTK+ is the basis of the “Gnome” desktop environment on Linux.

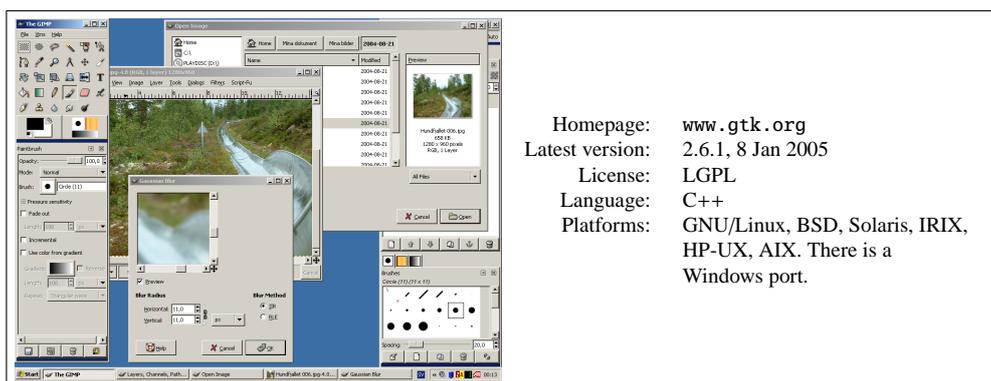
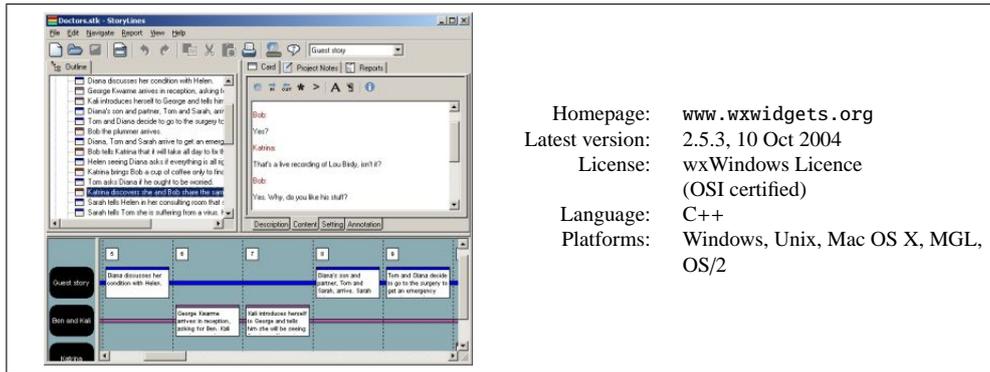


Fig.31: The GTK+ GUI library

A.6.3 wxWidgets

wxWidgets gives you a single, easy-to-use API for writing GUI applications on multiple platforms, with native look and feel. On top of great GUI functionality, wxWidgets gives you: online help, network programming, streams, clipboard and drag and drop, multi-threading, image loading and saving in a variety of popular formats, database support, HTML viewing and printing etc.

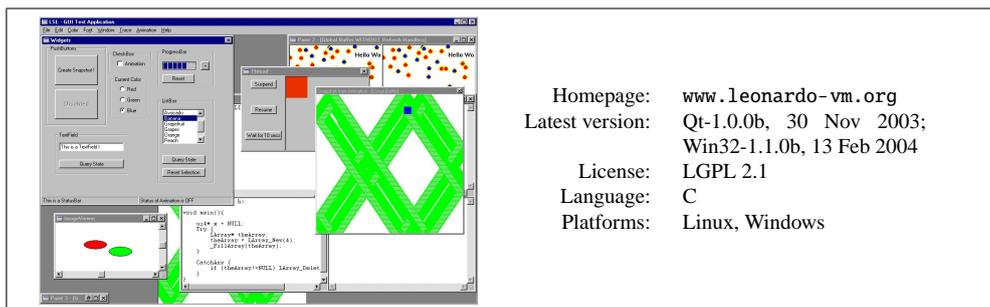


Homepage: www.wxwidgets.org
 Latest version: 2.5.3, 10 Oct 2004
 License: wxWindows Licence (OSI certified)
 Language: C++
 Platforms: Windows, Unix, Mac OS X, MGL, OS/2

Fig.32: The wxWidgets GUI library

A.6.4 Leonardo

The Leonardo Library (LL) is a cross-platform, open-source C toolkit for program development. The library is light, well-documented, easy to learn, and covers a large number of functionalities, including graphic user interface, thread, I/O and memory management. Differently from other programming toolkits, the LL also includes components with fundamental algorithms and data structures. Graphic user interface components defined by the library preserve the look and feel of the target platform.

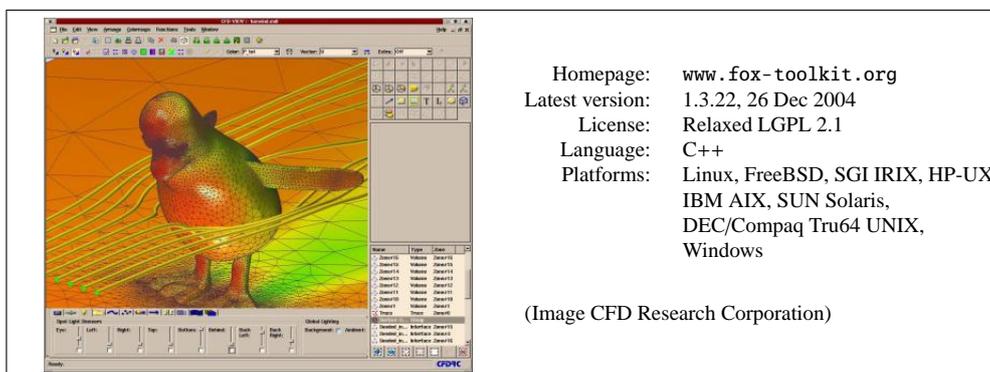


Homepage: www.leonardo-vm.org
 Latest version: Qt-1.0.0b, 30 Nov 2003;
 Win32-1.1.0b, 13 Feb 2004
 License: LGPL 2.1
 Language: C
 Platforms: Linux, Windows

Fig.33: The Leonardo Library

A.6.5 FOX Toolkit

FOX is a C++ toolkit for easy and effective development of Graphical User Interfaces. It offers a wide collection of controls, and provides state of the art facilities such as drag and drop, selection, as well as OpenGL widgets for 3D graphical manipulation. FOX also implements icons, images, and user-convenience features such as status line help, and tool-tips. Tool-tips may even be used for 3D objects.



Homepage: www.fox-toolkit.org
 Latest version: 1.3.22, 26 Dec 2004
 License: Relaxed LGPL 2.1
 Language: C++
 Platforms: Linux, FreeBSD, SGI IRIX, HP-UX, IBM AIX, SUN Solaris, DEC/Compaq Tru64 UNIX, Windows

(Image CFD Research Corporation)

Fig.34: The FOX GUI toolkit