Programmed design of ship forms

A. Rodríguez, L. Fernández-Jambrina*

ABSTRACT

This paper describes a new category of CAD applications devoted to the definition and parameterization of hull forms, called programmed design. Programmed design relies on two prerequisites. The first one is a product model with a variety of types large enough to face the modeling of any type of ship. The second one is a design language dedicated to create the product model. The main purpose of the language is to publish the modeling algorithms of the application in the designer knowledge domain to let the designer create parametric model scripts. The programmed design is an evolution of the parametric design but it is not just parametric design. It is a tool to create parametric design tools. It provides a methodology to extract the design knowledge by abstracting a design experience in order to store and reuse it.

Programmed design is related with the organizational and architectural aspects of the CAD applications but not with the development of modeling algorithms. It is built on top and relies on existing algorithms provided by a comprehensive product model. Programmed design can be useful to develop new applications, to support the evolution of existing applications or even to integrate different types of application in a single one.

A three-level software architecture is proposed to make the implementation of the programmed design easier. These levels are the conceptual level based on the design language, the mathematical level based on the geometric formulation of the product model and the visual level based on the polyhedral representation of the model as required by the graphic card.

Finally, some scenarios of the use of programmed design are discussed. For instance, the development of specialized parametric hull form generators for a ship type or a family of ships or the creation of palettes of hull form components to be used as parametric design patterns. Also two new processes of reverse engineering which can considerably improve the application have been detected: the creation of the mathematical level from the visual level and the creation of the conceptual level from the mathematical level.

1. Introduction

The evolution of CAD applications is due to several causes. The improvements introduced by CAD developers in their products are inspired by users' feedback and by technical advances in most cases. Technical advances may be available to any development team willing to make the most of them. On the other hand, the most effective way to improve CAD applications is to take into account advanced users' suggestions in the development process. A CAD application can be enhanced by designers if they are provided with a tool to incorporate their knowledge to the application. The programmed design is a proposal to achieve this goal.

The following paragraphs are devoted to review the state of the art of ship hull form design applications in order to extract the best features of the existing product models and the most relevant parameterization methodologies. The result of this review is used to establish the foundations of the programmed design.

2. State of the art

Since the initial days of the IT era, applications devoted to the design of ship hull forms have been devised since they have been considered a key milestone within the ship design lifecycle. Most technical activities of the design process depend on this source of information, the most immediate of which are naval architecture calculations. The outcome of these applications can be considered the origin of the ship product model as conceived nowadays.

During this long period of existence, a myriad of different hull form applications have been developed, many of them before the inrush of general purpose CAD systems. A nice review can be found in [1]. With the advent of the CAD concept in other areas such as mechanical and architectural design, some of these hull form design applications were reformulated and others were developed under this paradigm, consolidating definitively the modern concept of ship product models. Nowadays there are many

different applications devoted to design of ship forms, and all of them are a very good tool for some specific scenarios within the design lifecycle.

For the purposes of the programmed design, the most relevant design applications are those developed under the principles of parametric design, since programmed design is an evolution of parametric design. Parameterization is a key feature which provides several wonderful capabilities for a ship model such as the possibility of performing multidisciplinary optimization and design reuse (see Refs. [2–5]). Programmed design can be conceived as a tool to create parametric design tools.

General purpose CAD systems like CATIA [6] or SolidWorks [7] are solid modelers using a parametric feature-based approach to create models and assemblies. But solid modelers are not especially suited to design ship forms. Surface modelers like Rhinoceros are more adequate to perform this task. While Rhino is not parametric in its conception, Grasshopper [8] enables the creation of flexible parametric designs with Rhino. Grasshopper is a graphical algorithm editor tightly integrated with Rhino's 3-D modeling tools.

However, in the context of ship forms design applications the parameterization methodologies which have emerged are quite different to those provided by general purpose CAD systems. There are at least three different methodologies to create parametric ship forms: global parameterization, geometric parameterization and parameterization by transformations. The following paragraphs are devoted to reviewing these methodologies of parameterization and the underlying product models.

2.1. Global parameterization

The variety of applications oriented to design of ship forms is surprisingly high, as has been already mentioned in the previous paragraphs. One of the most original approaches is the methodology provided by hull form generators, which are applications developed under the principles of parametric design. Refs. [9–11] provide a nice introduction to ship form parametric design concepts. The general approach to hull form generation requires bookkeeping on surface and volume domains. A solution to this problem is presented in [12].

Two applications are the most representative of this type. One is the FORAN hull form generator, based on a waterline formulation and the other is the FRIENDSHIP modeler, based on parametric design grounded on sections. To understand this type of application, let us consider the FORAN hull form generation, since it was the first application which implemented this concept.

The FORAN [13] hull form generator is based on a waterline formulation defined by parameters containing very relevant geometric, hydrostatic and hydrodynamic features. The amount of parameters available to define the waterline provides great flexibility. The waterline formulation has been developed to contain nice design characteristics if the parameters are maintained within some specified ranges. Any parameter of the waterline has a default value which is very convenient when information is scarce. One of the advantages of this approach is that the application contains a great amount of heuristic design information available for the designer.

To generate the hull surface, each of the parameters of the waterline is controlled by a parametric draft function which determines its vertical distribution. Then, the draft function parameters combined with the waterline formulation provide the degrees of freedom available for performing hull form variations. Due to the nature of the parameters of the waterline formulation, many of the draft functions are hydrostatic characteristics of the designed hull form, and this allows the designer direct control of these features.

In the case of the FRIENDSHIP modeler [14], a parametric design section is generated by means of a suitable set of longitudinal curves. The concept is very similar to the FORAN hull form generator except for the arrangement of curves, which is orthogonal to that of the FORAN lines. Longitudinal curves which control the parametric distribution of design sections are also very meaningful for the designer as it is for example the sectional area curve. Then, the designer is able to specify hull form characteristics within their semantic domain as in the FORAN system. See Ref. [15] for more detailed information.

FORAN waterline formulation allows us to nicely fit and distribute the hydrostatic characteristics of the hull. The FRIENDSHIP modeler can do something similar with longitudinal characteristics (some of them are also hydrostatics characteristics) and naturally supports Lackenby transformations [16]. A nice feature of the FRIENDSHIP modeler is that it generates geometry (curves and surfaces) based on the NURBS formulation. This makes easy its integration with any other CAD/CAE/CAM applications.

The parameters which define the control curves of the method are meaningful data in the knowledge domain of the designer such as main dimensions, hull and waterline coefficients, hydrostatics characteristics, underwater volume, center of buoyancy, geometric parameters, etc. With this method the designer can interact only with its semantic domain during the design process without taking care of the geometric details of the model. For this reason this type of parameterization can be called holistic or global parameterization.

These tools are very useful for initial design and are unbeatable for preparing a contract design very quickly, as it is usually required for these short design processes. On the other hand, they are not flexible enough to fit any kind of hull form details, as it is required for fitting and fairing for production.

2.2. Parameterization by transformations

The holistic or global parameterization provided by hull form generators is a very powerful tool for the initial design stage and for conventional hull types. But there are other parametric approaches which are not as powerful but much more generic and its use can be extended to other design stages and for any type of hull forms. These other approaches are parameterization by transformations and local or geometric parameterization.

Parameterization by transformation consists in adding parametric features to an existing model (which could be parametric in origin but not necessarily) by means of a parametric transformation which produces a new hull form. For this reason this approach is also called parameterization "a posteriori" or partial parameterization.

Parameterization by transformation is aligned with one of the traditional methods of ship design, which consists in starting with a ship which is similar to the one required by design and performing affine transformations on it to reach the target dimensions. When there are more than one ship, the method described in Ref. [17] may be used to combine them.

The final step of parameterization by transformation is to apply Lackenby transformations [18] to fit hydrostatic features and hull coefficients. There are other types of transformations apart from affine and Lackenby transformations, such as local transformations which can be restricted to transform only some specific zones of the hull, but the first ones are the most useful and extended. For any world class application this functionality is a "must have" and consequently parameterization by transformation can be found in FORAN, NAPA, etc.

2.3. Geometric parameterization

Geometric parameterization is the most common methodology provided by general purpose CAD systems. Any of the input

arguments used during the design session can be considered as a parameter in order to produce variations of the design. The most common input arguments used for modeling are geometric conditions like dimensions and tangencies, providing the name to this type of parameterization. This functionality arises naturally in applications in which the whole design session is registered in a script. Applications with a user interface consisting of text commands typed by the user can provide this type of parameterization without too much development effort. An example of this type of parameterization can be found in the NAPA system.

Any scriptable product model can be parameterized with this methodology and consequently can be incorporated into the programmed design scope. The most common geometric product models used in the design of ship forms ought to be incorporated into the programmed design to implement geometric parameterization. The following paragraphs are devoted to describe the most relevant of them.

2.3.1. Wire models

One of the most successful and traditional approaches for developing a hull form design application is by means of a wire model. The wire model is an indirect way of defining a surface, which provides some advantages but also some disadvantages. Among the advantages, the most interesting is that the way of working mimics the traditional method of fitting and fairing by hand with batten and weights.

The main disadvantage of this model comes from the fact that defining a surface with a wire model is an indirect way of doing the work and the supplied information is incomplete. The surface is well defined just on the curves belonging to the wire model and has to be deduced on the holes between the defining curves. The algorithm used to fill the gaps is critical and it is what makes the difference between applications. For example, some systems, like NAPA, use a sort of bi-cubic patch while others, like PIAS-FAIRWAY, use transfinite interpolation [19].

As the fitting process can be performed with curves, it is very easy to carry out this work with a wire model. Since the fairing process should be based on the resulting surface, which is indirectly and incompletely controlled with this method, this process becomes cumbersome and requires experience, but it is straightforward and familiar for most designers.

A typical hull form which has undergone a fitting and fairing process for production with this method usually has many curves incorporated to control the resulting surface. A large number of curves in a wire model generates an even larger number of filling patches between gaps. The wire model application is capable of coping with this fact, but for other applications which need to import the resulting surface model it is usually a nightmare to deal with this fragmented set of small patches.

2.3.2. Surface models

The *B*-Spline/NURBS formulation is one the most common product model basis found among hull form design applications. This formulation provides not only the product model, but also a modeling tool which is very easy to use and requires little development effort [20]. The starting point of a design process with this type of applications is usually a simple surface, for example a rectangular plane plate or a cylinder. The dimensions and the number of control points of the initial patch are predefined by the user. From this starting point, the user can model the hull by moving and inserting control points (entire rows or columns) in the patch.

With this methodology the fairing process consists in getting a net of control points as uniformly and regularly distributed as possible. As in the NURBS formulation the derivatives of the surface are vectors which can be extracted from the net of control points, getting a uniform distribution of these points provides a smooth variation of the derivatives and consequently the same could be said about the curvatures. A more advanced approach to the generation of fair free-form surfaces can be found in [21]. The main problem that the designer finds with this type of application is the destructive interference between fitting and fairing processes. On moving the control points for the fitting process, the fairing is destroyed and the same happens the other way round. When the degrees of freedom are insufficient, additional rows or columns of control points should be added to increase the modeling capabilities. But increasing the amount of control points on the surface amplifies the destructive interference of both processes, making difficult convergence to an acceptable solution.

Fig. 1. Ship forms defined with FORAN FSURF.

Anyway, when the fitting requirements are not too strict or the hull forms are not very involved with constructive details, this method is very feasible and has many adepts. Examples of this type of application are MAXSURF [22] and FASTSHIP [23].

There is another type of hull form design application based on the NURBS formulation but with a very different modeling strategy. The advantages of a wire model for fitting tasks have already been pointed out. Additionally, the fairing process is better accomplished with a surface model [24]. The combination of both strategies can be implemented by means of sophisticated algorithms providing different constructive methods for creation, fitting and fairing of patches [25]. Algorithms for fitting curves to points (interpolation, approximation, etc.), construction of surface patches from curves (interpolation, approximation, skinning, etc.), automatic fairing of curves and patches, manual fitting and fairing of patches using curves as auxiliary tools, management of trimmed surfaces and management of continuity (in position and tangency) between adjacent patches are among the kind of algorithms required by this methodology. An example of this type of application is the FORAN FSURF module based on the NURBS formulation, shown in Fig. 1.

3. Programmed design

3.1. Concept

As it has been stated before, the programmed design goal is to provide a CAD environment in which advanced users may incorporate their knowledge to the CAD application by themselves. Consequently, the programmed design functionality is a component to be built on top of an existing CAD application. The CAD application has to provide the algorithms used to create the product model elements. On the other hand, the programmed design incorporates a design language to allow the advanced user the creation of modeling programs with those algorithms.

In order to implement the programmed design environment a comprehensive product model is required to face the modeling of any type of ship. This product model will have to integrate most of the different modeling and parameterization methodologies which have been found in the review of the state of the art. To

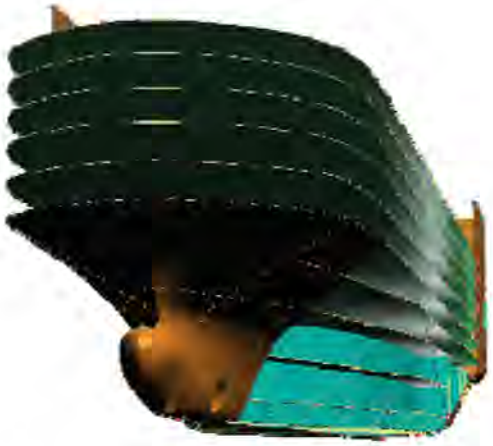

Fig. 2. Surface model for containing different forms: external hull, decks, bulkheads, etc. Some zones of the external hull form have been removed to show the inner forms.

facilitate the integration of very different components, the product model needs a structure specifically designed for this purpose. Additionally, the design language provides out of the box the geometric parameterization while other parametric approaches have to be implemented by the product model algorithms.

The following paragraphs are devoted to develop the product model structure that is required by the programmed design environment and the design language which is adequate to "program" such a product model.

3.2. Product model

In the context of this paper, the term product model refers to a computational representation of the ship, which is defined and exploited during the design process in order to support all technical activities. The product model should provide an unambiguous representation of the vessel, preferably based on neutral formats, in order to describe the ship as a product.

This representation contains information of geometric nature and other non-geometric data such as topology, parameters and technological attributes, but the most important information contained in the product model is the geometric representation of the ship.

Product model topology is nicely explained in Ref. [26] whereas this paper focuses on geometric representation. This representation is defined as a collection of types of entities that can be used to build the model and these are usually arranged in a hierarchical tree. This organization of the model is aimed to make the design process easier.

The product model required for the process of ship form design is a collection of independent surfaces which can be connected by topological relationships (see Fig. 2). These surfaces are hull(s), decks, bulkheads, superstructures, appendages, etc. They represent the first level of the hierarchical structure of the product model. Each of these forms may have a great complexity depending on the sort of ship. In conventional ships the most complex surface is the hull but modern ships require the same modeling capabilities more and more for any of their forms.

In order to manage the complexity of the surfaces, each of these forms can be divided into zones. The advantage of this subdivision is the possibility of applying a different modeling methodology to each of the zones in which the form has been split. Hence, the product model has to provide a different type of zone for each modeling methodology supported by the application and a subdivision method to split the form into zones, if more than one zone is required to model the form. Considering the different modeling methodologies found in the state of the art revision, it

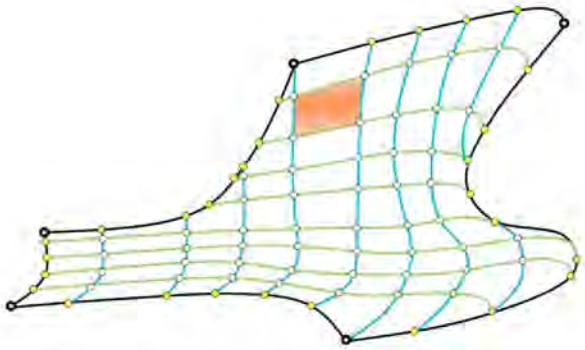

Fig. 3. Typical U–V arrangement of a fore body wire model.

is convenient to have three different types of zones: planar zones, constructive zones and generic zones.

The planar zones are required to make the definition of flat surfaces independent from the definition of curved surfaces when modeling the forms, since it is possible, but involved, managing both types of surfaces within the same zone. Planar zones are defined by means of 2D contour curves.

The constructive zone is aimed to supply the holistic parameterization and any other method to create a patch by means of geometric constructions like cones, cylinders, and swept surfaces or with more complex methods like skinning, blending, trimming and rounding surfaces. Constructive zones are basic constituents of the model which can be defined by means of specific and well defined methods. Depending on the complexity of the construction, the result may consist in one single patch, two patches or even three patches.

The generic zones implement the wire model methodology. The borders of the zone are defined with connected curves defining a 3D contour while the inner geometry is defined by two sets of intersecting curves called U curves and V curves. The U curves have the first and the last points in the borders of the zone and the inner points must lie on the V curves. The V curves are defined in the same way with respect to the borders and the U curves (see Fig. 3). The main advantage of this modeling methodology is that it does not need structured data with a rectangular arrangement as many of the constructive methods need. The key factor to implementing this approach is the filling algorithm which is used to create a surface from grid curves.

One way to provide an easy and robust method of filling gaps between grid curves is by means of Bézier patches of rectangular and triangular topology. This method may require additional work to prepare the grid in order to produce only rectangular and triangular patches, but the advantages overpass the effort spent in this preparation work. In any case, this work could be performed automatically by the application. Some applications create triangular patches by collapsing one border of a rectangular patch. This is not a good idea because in such patches the normal vector is undefined by the formulation despite there is enough geometric information to define it correctly. Using triangular Bézier patches this problem is avoided.

In order to help the coexistence of different types of zones within the same form, a subdivision procedure is required with the capability of keeping the continuity and optionally the tangency between adjacent zones. This requirement is easy to comply with if the zones are created following a specific sequence. The constructive zones must be defined first. Then, a map of contour curves enclosing generic and planar zones should be defined. This map can include the limiting curves of the constructive zones as contour curves for generic or planar zones when needed. Finally, each generic zone should be completed with the U and V curves, which should be extended up to the borders of the zone defined in the map. The map can be considered as an agreement between

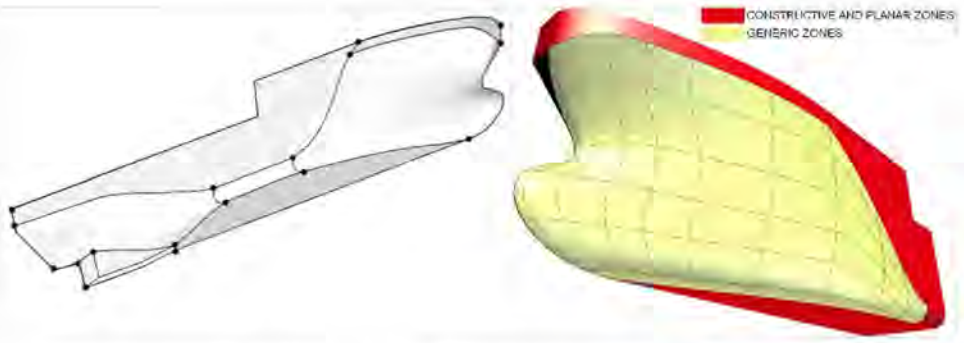

Fig. 4. A map of generic, constructive and planar zones.

adjacent zones with respect to the continuity and tangency of the form. It is possible to insert U and V curves inside a generic zone without modifying the geometry of the contour curves using the knot insertion algorithm of the NURBS formulation. Fig. 4 provides an example of a map.

3.3. Design language

A language which is suitable for supporting modeling tasks performed by a form designer without programming skills must comply with some specific requirements (see Ref. [27]).

First at all, the language has to be read by the design application and consequently the latter plays the role of interpreter of the language. Hence, the role of the designer is to specify what the program should accomplish, rather than describing how to go about accomplishing it, as the designer is not supposed to have special skills to write complex algorithms. In order to make the use of the language easier to the designer, the language should be specifically oriented to the ship form modeling domain. Consequently, the design language should be a declarative interpreted domain specific language. Additionally, the language should provide some control sentences such as loops and conditional branches to facilitate executing repetitive tasks and automation of processes.

A declarative language script consists of a sequence of declarations which aim in this case to build product model elements. Each of these sentences creates a geometric construction which can be either a product model component or an auxiliary entity to be used later to build a more complex component of the product model. These declarative sentences can be considered "constructors" of primitives (auxiliary elements) or components (constitutive product model elements).

Hence, a design language script is a set of constructors which can be combined with control sentences to create loops, branches and procedures. The outcome of executing this script should be a complete ship product model or a part of it.

3.3.1. Types and constructors

The design language has to provide a complete set of element types in order to be capable of creating any primitive or component which may be required to build the ship product model with any of the methodologies which have been selected from the state of the art.

According to the language premises, each type is provided with a set of constructors and each constructor implements a specific algorithm or method to create an element of such type. Each constructor is identified by a unique name within the type and has associated a list of arguments which collects all the input data required by the algorithm. Each element of the list of arguments is a primitive of the product model. The result of executing the constructor is a new primitive or a new component of the product model with a unique name in the whole product model.

In order to create a primitive or a component of an explicit type with a specific constructor, all of the primitives required by the list of arguments must be created before invoking the constructor. This may produce very heavy scripts. To avoid this problem the language must provide the possibility of specifying anonymous constructors. An anonymous constructor creates on the fly an element of the required type within the list of arguments to be consumed immediately and for this reason no name is required.

Most common primitive types are points, polylines, curves, patches, planes, vectors, lines, segments, integers, floats, strings, Booleans and lists of primitives. Component types are constructive zones, generic zones, planar zones, maps and forms.

The syntax of a constructor invocation is: ***TYPE elemID ConstructorID (primID1, ..., primIDn)***.

An anonymous constructor contains only the constructor identification with the list of arguments, as the type is inferred from the context: ***ConstructorID (primID1, ..., primIDn)***.

Using an anonymous constructor consists in substituting a primitive identification in the list of arguments of another constructor with the anonymous constructor invocation:

TYPE elemID ConstructorID1 (***primID1***, ***primID2***, . . . , ***primIDn***);

↓

TYPE elemID ConstructorID1 (***primID1***, ***ConstructorID2*** (***p1***, . . . , ***pn***), . . . , ***primIDn***).

Anonymous constructors can be nested ad infinitum.

3.3.2. Control sentences

The designer must have the possibility of writing loops and conditional jumps to implement more advanced procedures. The following schemas are required:

Conditional branching: ***if*** (B_1)***;...;else if*** (B_2)***;...;else;...;end if***.

Conditional loop: ***while(B);...;end while***.

In the above expressions, B, B_1 and B_2 are identifications of Boolean elements or anonymous constructors of such type of primitive.

List scanning loop: ***for(listID, listIndex, listElement);...; end for;*** In order to take full advantage of the list scanning loop, the language has to include a list type for each type of primitive (float list, points list, curves list, etc.) and some list of lists types (list of lists of floats, etc.).

Element mutator: ***set elemID ConstructorID (primID1,..., primIDn);*** Mutator syntax is required to modify existing elements by means of any of its constructors.

In order to organize, encapsulate and reuse design language scripts, the language must provide the possibility of writing procedures and user constructors:

Procedure encapsulation: ***proc(inputID1,.., inputIDn);...;end proc(outputID1,.., outputIDm)***.

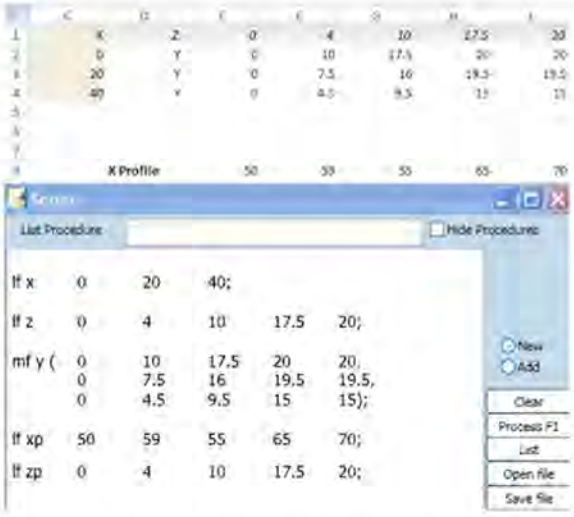

Fig. 5. Simplified offset table, three frames by five waterlines with the fore profile and the appropriate data structures to manage this information.

Procedure invocation; ***call proc((inputID1,.., inputIDn), (outputID1,.., outputIDm))***.

User constructor definition: ***cons TYPE ConstructorID(TYPE1 primID1,.., TYPEn primIDn);...;ret retID***.

User constructor invocation: ***TYPE elemID ConstructorID (primID1, ..., primIDn)***.

A complete language specification is out of scope of this paper, but in order to explore the programmed design concept the authors have developed a program for demonstration purposes. This prototype implements some geometric parameterization constructors. In the next paragraph a simplified case of use of programmed design is shown, which has been created with the demonstration program just to illustrate this paper.

Example. The starting point of this example is a very simplified offset table for a fore body, as shown in Fig. 5. The offset table is transferred to the design language primitives devoted to contain lists of floats ***lf*** and matrices of floats ***mf***, which in fact are lists of lists of floats. This transfer has been performed by means of the clipboard copy and paste functions between the worksheet and the script editor of the demonstration program.

The heights of the waterlines are stored in a list of floats named ***z*** with the following sentence: ***lf z* 0 4 10 17.5 20;** This sentence is using a simplification of the most formal syntax which should be: ***lf z* (0, 4, 10, 17.5, 20);** This simplification is allowed for plain lists of arguments, as those without anonymous constructors are. Other lists of floats like ***x, xp*** and ***zp*** are defined in a similar way, while ***y*** is a list of lists of floats of type ***mf***.

Using the above data, the following declaration creates the profile curve: ***c prof.xz(0,.(.lf xp zp));*** where ***c*** indicates that the type is a curve and ***prof*** is the curve identification. The curve constructor for a profile is.***xz***, which has as arguments a float for the *Y* coordinate and a 2D curve. The first argument of the curve constructor is the *Y* coordinate **0**, as the profile is in the central plane. The second argument is.***(.lf xp zp)*** where. (indicates the default constructor for 2D curves, that has as argument a list of 2D points). In this case, the list of 2D points is created by an anonymous constructor which takes two list of floats:.***lf xp zp***. This list of arguments can be typed in the shortened way as it is simple, instead of the formal way.***lf(xp, zp)***. The result of executing this constructor is shown in Fig. 6.

In order to provide tangent conditions at points of the profile there are other constructors for the 2D curve in which the initial and/or final tangents or even a tangent for each point of the curve can be defined. The following declaration uses the last one, which requires a list of 2D points and a list of 2D tangencies as input arguments:

ltc2 tprof* 0 90 90 30.*n*;** List of 2D tangencies (2D vectors or angles) with name ***tprof and where ***.n*** means free tangency.

.pt(.lf xp zp, tprof) Anonymous 2D curve constructor with two arguments. The first is the list of 2D points given by an anonymous constructor from 2 lists of floats and the second is the previously defined list of 2D tangencies ***tprof***.

c prof xz*(0,.*pt(.lf xp zp, tprof))**. Profile curve constructor ***.xz with a different second argument indicating a 2D curve with tangencies defined by the anonymous constructor explained before. The resulting 3D curve name is ***prof***. The results can be appreciated in Fig. 7.

The same technique is used to create each section. In this case, first of all a tangency pattern is created with the 2D tangency constructor:

ltc2 tfr* 0 .*n* .*n* 90 90**. List of 2D tangencies (2D vectors or angles) with name ***tfr and where.***n*** means free tangency.

.lis x* 0** Anonymous constructor returning the element of list ***x with index **0**. This is the first element of the list of floats x, and so it is a float constructor returning the float value **0.0**.

.lis y* 0** Anonymous constructor returning the element of list ***y with index **0**. This is the first element of the list of lists of floats ***y***, and so it is a list of float constructor returning the list of floats **(0.0, 10.0, 17.5, 20.0, 20.0)**.

.lf(.lis y 0, z) Anonymous constructor of a list of 2D points with two arguments which are lists of floats. The first one is constructed by the anonymous constructed explained before and the second one is the previously created list named ***z***.

.pt(.lf(.lis y 0, z), tfr) Anonymous 2D curve constructor with two arguments. The first one is the list of 2D points given by an

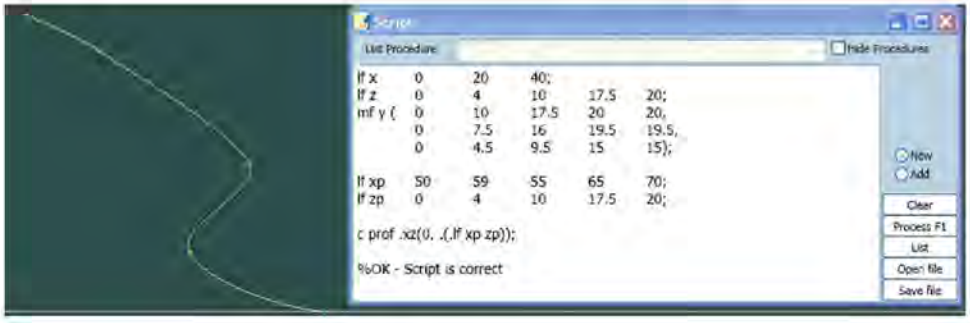

Fig. 6. Profile construction from two lists of *X* and *Z* coordinates.

Fig. 7. Profile curve with tangent conditions.

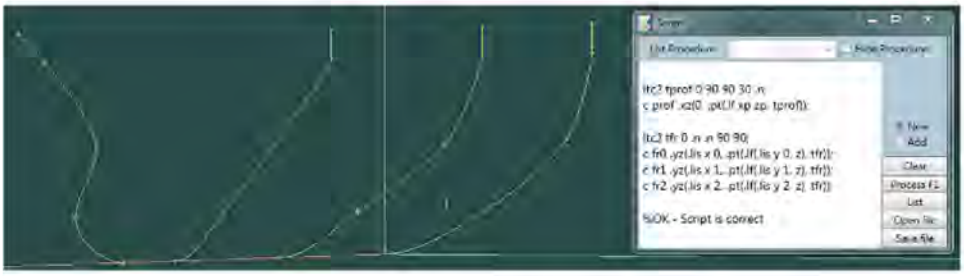

Fig. 8. Profile and frame curves.

anonymous constructor from 2 lists of floats explained before and the second one is the previously defined list of 2D tangencies ***tfr***.

cfr0.yz(.***lisx0***, .***pt***(.***lf*** (.***lisy0***, *z*), ***tfr***));

cfr1.yz(.***lisx1***, .***pt***(.***lf*** (.***lisy1***, *z*), ***tfr***));

cfr2.yz(.***lisx2***, .***pt***(.***lf*** (.***lisy2***, *z*), ***tfr***)).

These three constructors define named 3D curves in a *YZ* plane given an *X* coordinate (a float value) and a 2D curve defined in *U*, *V* coordinates corresponding in this constructor with *Y* and *Z* coordinates. The arguments are given by means of already explained anonymous constructors. The result is shown in Fig. 8.

Hence, the body is created with a patch constructor from the previous curves and a new curve that is used to indicate the tangency condition at the fore border. This is shown in Fig. 9 with some improvements as a "for loop" to create the frames using a list of curves:

The loop to create the curves uses a list of curves created outside the loop and used within the loop to load the curves: ***lc lfr.ini;*** where ***lc*** stands for a list of curves, ***lfr*** is the name of the list of curves and.***ini*** is the constructor to create an empty list.

The loop is invoked by the sentence ***for x i xc;*** where ***x*** is the list to scan (a list of floats which contain the abscissas of the frame sections), ***i*** is the index of the current element of the list (of integer type) and ***xc*** is the current element of the list (of the type corresponding to the elements of the list, in this case a float).

Within the loop the curves are created anonymously as they are added to the list. Since the list is not created but modified, a special syntax for mutators is required: ***set lfr.add(.yz(xc,.pt(.lf(.lis y i, z), tfr)));*** which uses the generic list constructor.add to create a new list adding the argument element to the previous list ***lfr***.

The tangency condition at the fore end if defined by the curve ***c tbody.pxz(.xy*(50 0, 20),.*pt(.lf xp zp, tprof));***

The curve name is ***tbody*** and the constructor.***pxz*** takes two arguments. The first one is a plane containing the curve and the second one is a 2D curve that represents the 3D curve projected on the *XZ* plane. In this case the plane is parallel to the *Z* axis passing through the point (50, 0, 0) forming 20° with plane *XZ*, as it is indicated by the anonymous constructor.***xy*(50 0, 20)**. The 2D curve is the same as it has been used to create the profile.***pt(.lf xp zp, tprof)***. This tangency curve combined with the fore profile defines tangency vectors at the fore end perpendicular to the center plane.

Then the patch named ***body*** is created with the constructor.***ti***: ***pat body.ti(1 tbody,.rev lfr);***

The.***ti*** constructor has two arguments, a patch tangency condition for the initial tangency (the final tangency is free with this constructor) and a list of curves representing patch sections. A patch tangency condition is given by a float factor and a curve. Tangent vectors are taken from the affected patch curve (in this case the first element of the list of curves) to the given tangency curve and multiplied by the given factor. In this case the factor is 1.0 and the curve is ***tbody*** which has been previously defined in a convenient way to get tangencies at the fore end perpendicular to the center plane.

The list of curves is defined with the anonymous constrictor.***rev lfr***, which returns the list ***lfr*** in reversed order. It is defined this way to get the patch normal vectors pointing outside. This is the reason for using the.***ti*** constructor for the patch, since the profile becomes the first curve of the list.

Finally, a new user constructor can be defined (Fig. 10) and used parametrically (Fig. 11).

Note that the selected text in the Fig. 10 is a complete definition of a new user constructor for patches. Once it is defined, it can be invoked as any other "built in" constructor of the underlying CAD application. Additionally, the list of arguments of the new constructor are parameters on which it is possible to perform parametric variations as shown in Fig. 11, where changed values are highlighted. Any input argument can be modified providing geometric parameterization. The user constructor can be applied to any data structure similar to the simplified offset table of the example. With some additional programming effort, the constructor can be abstracted for more generic offset tables.

The complete script can be wrapped with a new user constructor exposing just the parameters which are required for an specific analysis, as for example, length and height of the bulbous bow.

This example is necessarily oversimplified in order to keep the paper in a reasonable length, but it illustrates the idea of a tool used

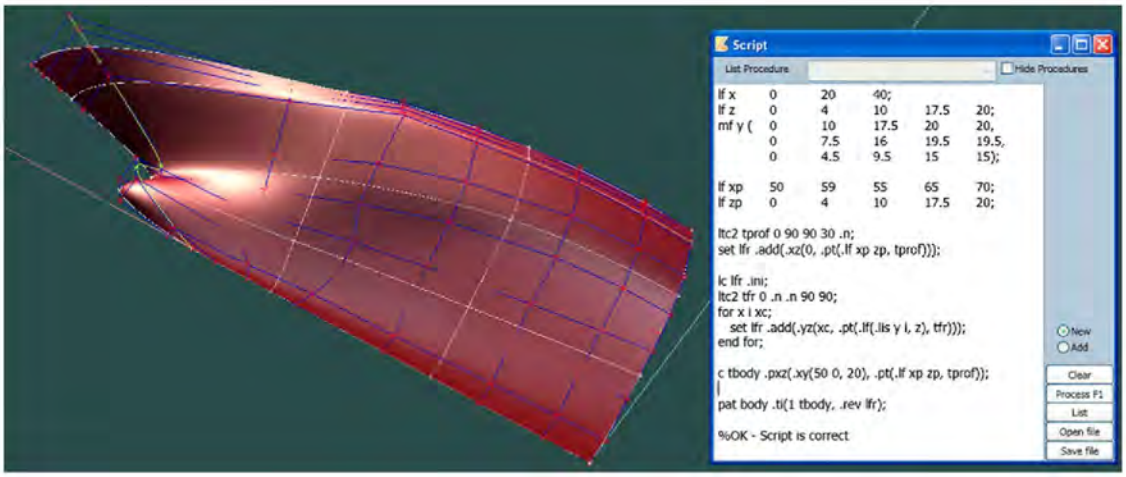

Fig. 9. Fore body with bow tangency.

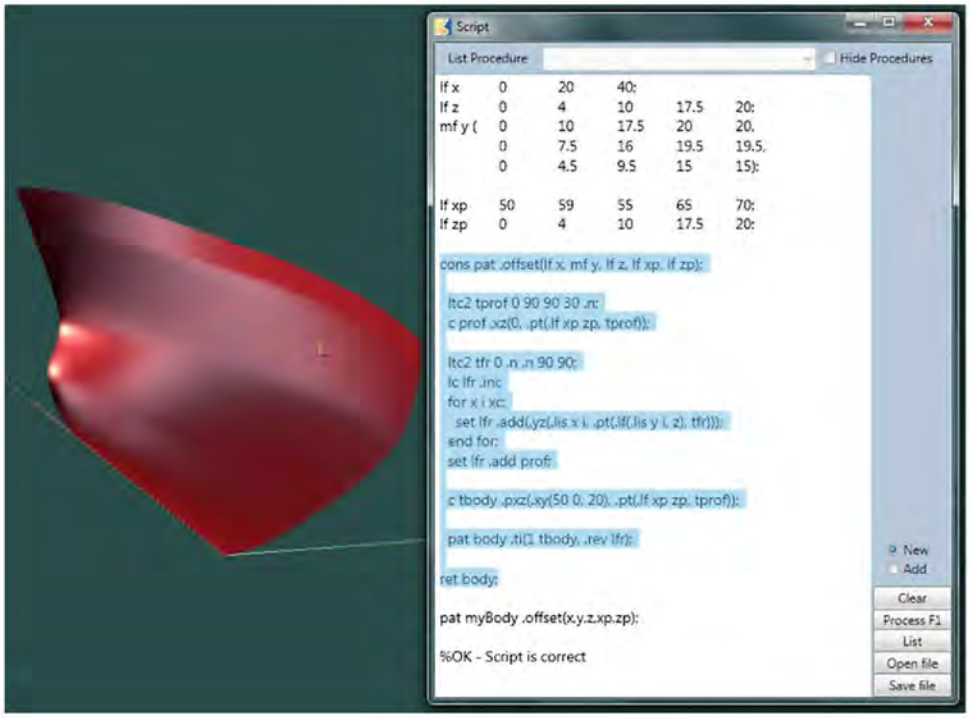

Fig. 10. Fore body constructor.

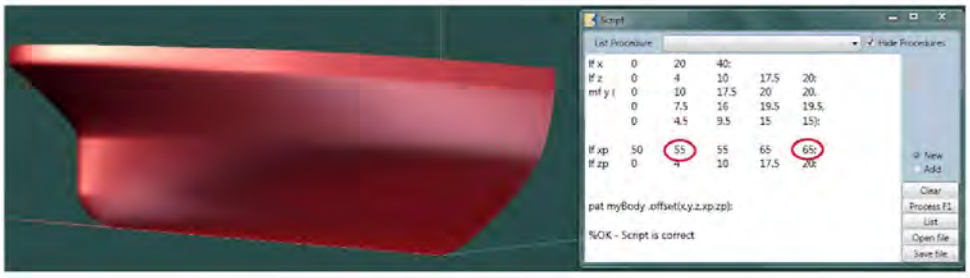

Fig. 11. Fore body parametrized.

by advanced designers to write their design experiences with a dedicated language and which can be stored and reused whenever required.

4. Architecture of the application

Ref. [28] provides a very detailed explanation of the most relevant architectural aspects of shipbuilding CAD applications.

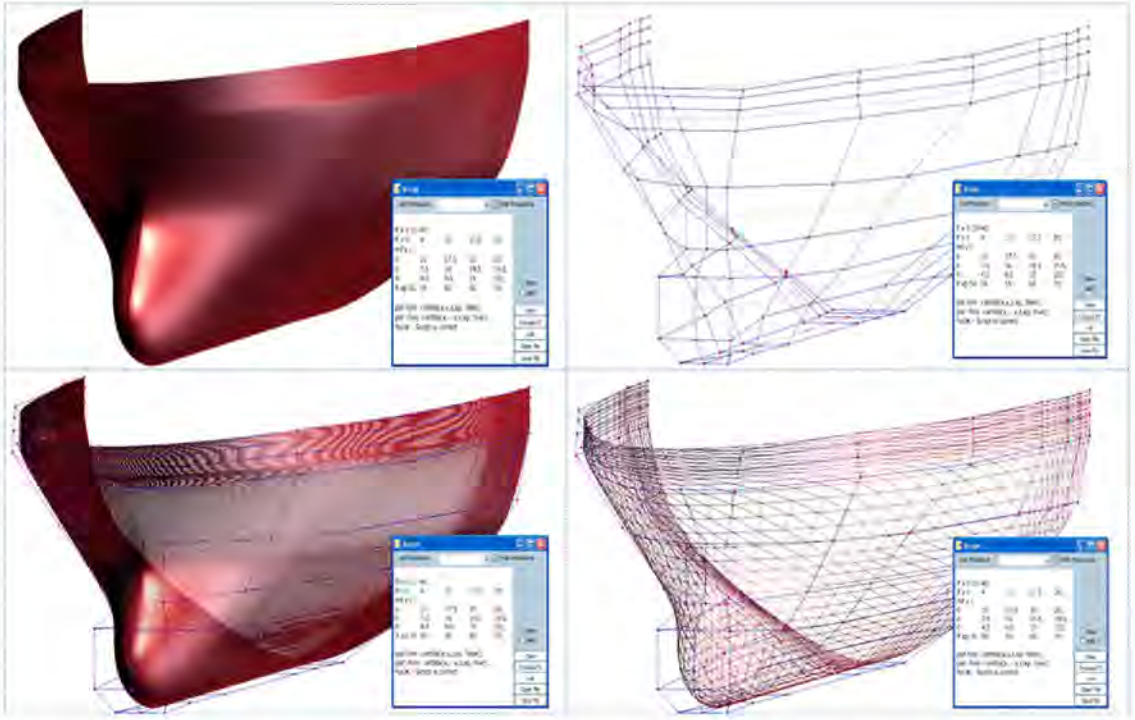

Fig. 12. Top left, conceptual level (the design program and its result), top right, mathematical level (the net of control points and the isoparametric curves of the NURBS formulation) and 2 different visual levels (bottom left generated with higher precision and bottom right generated with lower precision).

The functionality of a design application comprises the tools for creating, viewing, editing and interrogating the product model. The implementation of this functionality requires data structures which support these functions and all the algorithms associated with these tasks. Under the paradigm of object orientation, both data structures and algorithms are encapsulated in classes. The main goal of the process of software design is the assignment of responsibilities to the objects (a mantra for developers).

The result of the software design is a class diagram which contains all the data structures and algorithms required by the application. The diagram also represents the relations between objects. A relevant subset of this diagram should be devoted to implement the product model. The product model class diagram has to supply the algorithms required by other objects devoted to provide services such as visualization, selection, edition, interrogation, etc.

In order to make easier the interaction between these service objects and the product model implementation, it is very convenient to split the product model into three levels. These levels are especially useful for arranging the different types of algorithms which are implemented by the product model and can be considered levels of abstraction. The most abstract of them can be called the conceptual level and it is devoted to implement the design language, the organization of the model, the topology and any other kind of relational or organizational aspect of the product model. The next level of abstraction provides a mathematical formulation for each of the entities created by the conceptual model. In this example the mathematical level is based on NURBS formulation and Bézier triangles and rectangles. Finally, the model has to interact with the graphic card and requires a visual representation of the model based on faceted or polyhedral surfaces and polylines. This visual model is extracted from the mathematical level by means of some algorithms which require the pre-selection of certain precision parameters. Fig. 12 provides a graphic explanation of the previous concepts.

These three levels can be interpreted as a cause–effect chain or a three stage projection process. The concept is implemented or described mathematically and these mathematical entities are visualized or represented as low level visual entities. It is also possible to consider that concept space is projected to a mathematical space for description and implementation and mathematical space is projected to a visual space for machine/world representation (visualization, selection and any kind of exploitation).

The projection from concept to mathematical level determines which algorithms are possible for creation and edition of each element or which constructors are available for each type. The projection from mathematical to visual level is determined by the level of precision which is required by the exploitation that the visual model will undergo. For different uses, different visual models with different precisions will be generated. Considering an ideal scenario, the whole definition of the product model should be performed at the conceptual level while the whole product model exploitation will take place at the visual level.

Taking into account the previous ideas, there is an optimal level for each product model algorithm to be implemented within the application, but sometimes there is space for selection. For example, an intersection algorithm can be implemented in the mathematical level (very complex) or in the visual level (easier but the outcome depends on the precision with which it has been generated).

In order to enforce the programmed design concept and to implement the proposed architecture a design application based on these premises has been developed. This prototype provides some geometric parameterization constructors. Developing holistic constructors is much more complex and it is let to a subsequent phase. The application has been developed in .NET C# language with Visual Studio 2010 Express Edition (.NET Framework 4.0). The application is configured as a Windows Presentation Foundation (WPF) application but it also uses Windows Forms for some specific tasks. The 3D engine is based on XNA Game Studio 4.0 for Windows (Xbox is out of scope). XNA requires the use of Windows Forms to be integrated within a WPF application. All of the examples provided in this paper have been generated with such application.

5. Conclusions

Programmed design can be considered as a new design methodology based on a design language with the semantics and syntax which has been explained before. As such, it can be considered another type of design tool with its own advantages and disadvantages. For a single or sporadic design, programmed design may not be the preferred tool. The user interface based on a design language is not the most adequate for the occasional designer. The best scenario where programmed design is without any doubt full of advantages is design reuse and knowledge management. In this way, programmed design can be considered as a tool for creating design tools or as a method to store and reuse design experiences. The tools developed with programmed design scripts should be wrapped within advanced user interface widgets to facilitate their usage by less experienced designers.

One type of design tool which can be created with programmed design is a parametric family of ship forms. Holistic parameterization is able to prefix the integral properties of the hull (volumetric and hydrostatic) within the product model, but paying a high cost with respect to the generality of the solution. With geometric parameterization, the integral properties should be evaluated after defining the geometry and this geometry should be modified to reach the required results. This retrofitted process can be "programmed" in a single script, combining geometric parameterization with parameterization by transformations. But combining holistic and geometric parameterizations with transformations can define extremely powerful hull form generators. In order to make the combination of local and global parameterization easier, it is necessary to develop functions to facilitate the distribution of those parameters which are automatically managed by the holistic zones. If the contribution of the geometrical zones to these parameters is evaluated, the contribution of the holistic zones can be easily adjusted to produce the final result of the parameter as required by the complete model. Typically, holistic zones are defined between aft and fore perpendiculars and geometric zones are used for appendages, bulbs, aft and fore endings, etc.

A more generic design tool can be based on the use of a palette of form components that are used like design patterns. These components should be strongly typified and its interfaces with other components should have to be also very well specified. These components must be defined at a level of granularity in which most of the "normal" hull forms are composed of the same types of components, following a similar strategy as the interim product technology.

Finally, one way to improve the use of programmed design is to feedback the normal flow from conceptual level to mathematical level and from this to visual level. Hence, this feedback will require the possibility of transferring a visual model to a mathematical model and a mathematical model to a conceptual model. Of these two processes, the last one is the most useful as most surface models defined with design applications can be imported at the mathematical level. The process could require the selection of the types of entities and constructors which are going to receive (generate) the mathematical model but it should be automated as much as possible in order to take full advantage of the process. This functionality will allow the application to incorporate external designs with the same capabilities as native or proprietary designs and consequently produce programmed designs with external information in a very easy way.

Nowacki H. Five decades of computer-aided ship design. Computer-Aided Design 2010;42(11):956–69.

Abt C, Harries S. A new approach to integration of CAD and CFD for Naval architects. In: 6th international conference on computer applications and information technology in the Maritime industries, COMPIT2007, Cortona, April 2007.

Harries S. Serious play in ship design, tradition and future of ship design in Berlin, Colloquium, Technical University Berlin, Abridged version, February; 2008.

Harries S, Abt C. Parametric curve design applying fairness criteria. In: International workshop on creating fair and shape-preserving curves and surfaces. Berlin/Potsdam: Teubner Verlag; 1998.

Harries S, Nowacki H. Form parameter approach to the design of fair hull shapes. In: 10th international conference on computer applications in shipbuilding. ICCAS '99, Massachusetts Institute of Technology, Cambridge, MA, USA; June 1999.

www.3ds.com/products/catia.

www.solidworks.com/.

www.grasshopper3d.com/.

Abt C, Bade SD, Birk L, Harries S. Parametric hull form design – a step towards one week ship design. In: 8th international symposium on practical design of ships and other floating structures, PRADS 2001, Shanghai; September 2001.

Abt C, Harries S, Hochkirch K. Constraint management for marine design applications. In: international symposium on practical design of ships and other floating structures, PRADS 2004, Lübeck–Travemünde; September 2004.

Kuiper G. Preliminary design of ship lines by mathematical methods. Journal of Ship Research 1970;14.

Nowacki H, Kim H. Form parameter based design of hull shapes as volume and surface objects. In: Proc. 12th international conference on computer applications in shipbuilding ICCAS'2005, Busan, Korea; August 2005.

www.senermar.es/NAVAL/foran/en.

www.friendship-systems.com.

Abt C, Birk L, Harries S. parametric hull design: the FRIENDSHIP-modeler. In: International conference on ship and shipping research, NAV 2003, Palermo; June 2003.

Abt C, Harries S. Hull variation and improvement using the generalized Lackenby method of the FRIENDSHIP-framework. In: The naval architect, RINA, September 2007.

Kang JY, Lee BS. Mesh-based morphing method for rapid hull form generation. Computer-Aided Design 2010;42(11):970–6.

Lackenby H. On the systematic variation of ship forms. RINA-Transactions 1950;92.

Veelo B. Shape modification of Hull models in H-REP. In: Conference on computer app. and IT Maritime ind., COMPIT'04, 2004.

Perez-Arribas F, Suarez JA, Fernandez-Jambrina L. Automatic surface modelling of a ship hull. Computer-Aided Design 2006;38:584–94.

Nowacki H, Jin F, Ye X. A synthesis process for fair free-form surfaces. In: Strasser W, Klein R, Rau R, editors. Geometric Modeling: Theory and Practice. Berlin, Heidelberg, New York: Springer-Publ.; 1997.

MAXSURF user's manual, formation design systems Pty Ltd. 1984–2009.

FASTSHIP Proteus Engineering.

http://www.proteusengineering.com/fastship.htm.

Brunet P, Vinacua A, Vivo M, Rodriguez A. Surface fairing for ship hull design application. Mathematical Engineering in Industry 1998;7(2):79–193.

Rodriguez A, Vivo M, Vinacua A. New tools for hull surface modeling. In: 1st int. conference on computer app. and IT Maritime Ind., COMPIT'00, 2000.

Solano L, Gurrea I, Brunet P. Topological constraints in ship design, IFIP conference proceedings. In: Proceedings of the IFIP TC5 WG5.2 fourth workshop on knowledge intensive cad to knowledge intensive engineering, Vol. 207. 2000. pp. 173–182.

Reidar T, Rodriguez A. Automation tools in the design process. In: 3rd int. conference on computer app. and it Maritime ind., COMPIT'04, 2004.

Rodriguez A, Gonzalez C, Gurrea I, Solano L. Kernel architecture for the development of cad/cam applications in shipbuilding environments. In: 2nd int. conference on computer app. and it Maritime ind., COMPIT'03, 2003.