Software pertaining to the preparation of CAD data from IGES interface for mesh-free and mesh-based numerical solvers (IntegralCAD version 0.9.1)

Maharavo Randrianarivony

Graphische Datenverarbeitung und Visualisierung,
Fakultät für Informatik, Technische Universität Chemnitz,
GERMANY

February 27, 2007

Abstract: We focus on the programming aspect of the treatment of digitized geometries for subsequent use in mesh-free and mesh-based numerical solvers. That perspective includes the description of our C/C++ implementations which use OpenGL for the visualization and MFC classes for the user interface. We report on our experience about implementing with the IGES interface which serves as input for storage of geometric information. For mesh-free numerical solvers, it is helpful to decompose the boundary of a given solid into a set of four-sided surfaces. Additionally, we will describe the treatment of diffeomorphisms on four-sided domains by using transfinite interpolations. In particular, Coons and Gordon patches are appropriate for dealing with such mappings when the equations of the delineating curves are explicitly known. On the other hand, we show the implementation of the mesh generation algorithms which invoke the Laplace-Beltrami operator. We start from coarse meshes which one refine according to generalized Delaunay techniques. Our software is also featured by its ability of treating assembly of solids in B-Rep scheme.

1 Introduction

It happens very often in practice that a volumetric problem can be reduced to a surface problem by using Stokes or Green or similar formulas. That reduction of dimensionality from 3D to 2D has an advantage in numerical computation of some engineering problems: instead of discretizing the whole

3D domain, we can exclusively concentrate on its *boundary*. That is the main advantage of BEM (Boundary Element Method) over other numerical techniques such as FEM (Finite Element Method). In particular, that dimension decrementation is very important for *exterior* problems: the domain of interest is the exterior of a 3D domain Ω. That is, we search for a function which is defined at each point belonging to the complementary Ω^c. In practice, the standard examples are the simulation of flows past 3D obstacles or computation of electromagnetic fields over 3D objects. The traditional way of treating exterior problems is to consider a very large bounding box B which includes the model Ω and one discretizes the exterior domain $B \setminus \Omega$. That unnecessary complication can be avoided if we use Boundary Element techniques. According to our discussions with some specialists in numerics of integral equations, the drawback of BEM – in comparison to FEM – is that there are so far very few softwares (commercial or not) which implement it, so its scope is therefore almost only restricted to research purpose and its industrial adoption is thus somewhat limited.

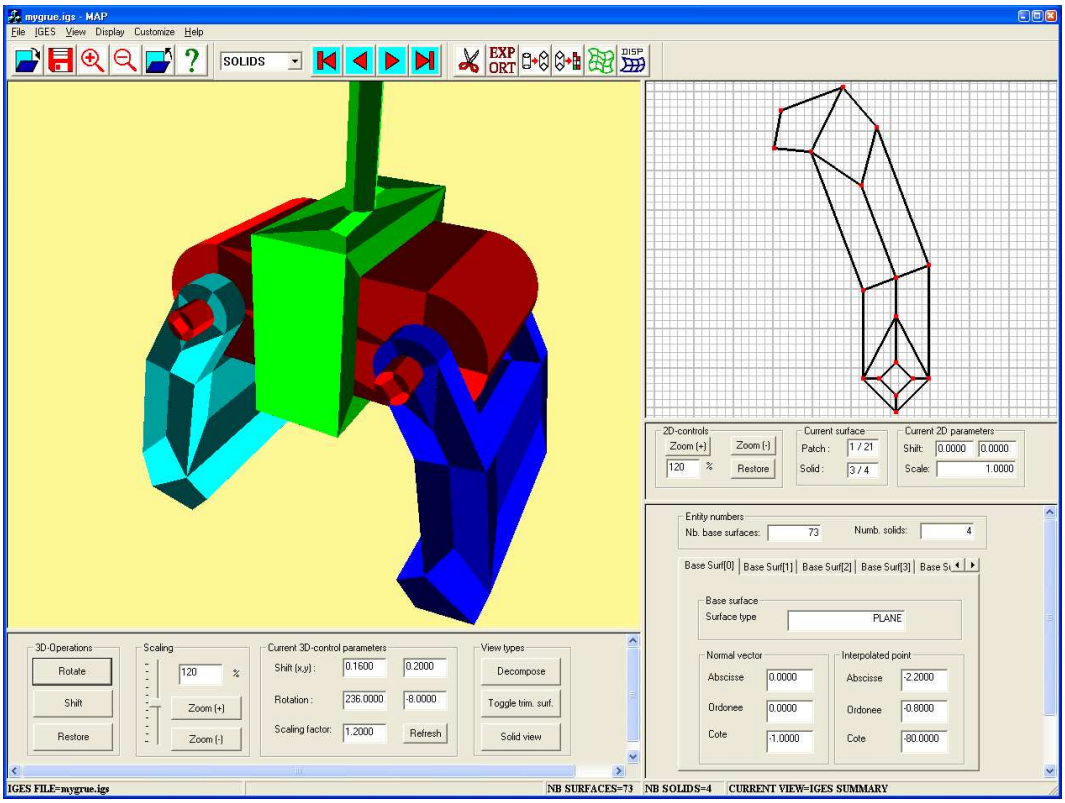


Figure 1: Main window

In this document, we describe the implementation of a nontrivial problem

in computational geometry which consists in preparing geometries to realize the reduction of dimensionality. Our process is for now applied to integral equations but we believe that it can be utilized for other engineering purposes. More precisely, we deal with the preparation of CAD models for subsequent use in integral equations with mesh-free or mesh-based structures. Let us note that we only focus on implementation part because the corresponding theoretical survey has already been completely documented in our former papers [17, 18]. The structure of this paper is as follows. We will give in the next two sections the exact formulation of treatment of CAD models together with some motivation from the numerics of integral equations. In section 4, we comment about our implementation tools including MFC [13], OpenGL and IGES. In particular, we will see MFC settings, some classes and typedefs which are used in implementing IGES entities. In order to simplify user interactions, we use menus and toolbars. We will provide in section 5 the realization and functionalities of each button of the toolbars. We will find there too the details about status bar which displays important information such as the number of solids in the model and the name of the current IGES file. We summarize the most important properties of the different trimmed surfaces in a property sheet. The parameters of each trimmed surface will be contained inside its property page which will be described in section 6. Furthermore, we detail in section 7, 8 the automatization of geometric input as well as its realization in MFC by using file dialogs. Since we need to extract the geometric entities from ascii files, we have to know the exact organization of the input files. That is why we give some detail about the structure of IGES interface. Afterwards, the required preparations so that MFC can be used together with OpenGL will be discussed. That combination procedures need some invocation of *wiggle* (Win32+OpenGL) functions. In section 10, we will show that it is possible to deal with several solids. The functionality of our software is now featured by B-Rep structures of solids. Thus, we apply the decomposition algorithm to each shell of the assembly. Toward the end of this paper, we summarize our former results [18] that describe the theoretical background which is used for the four-sided splitting, the diffeomorphism creation and mesh generation. In particular, some results using our adaptive algorithm and Gordon transfinite interpolation will be given.

2 Brief theoretical motivation

In the next discussion, we introduce some motivation about the importance of reducing a volumetric problem into a surface one. In particular, we recall briefly mesh-free and mesh-based methods for treating numerical integral equations. As an illustration, we consider the following Laplace problem with Dirichlet boundary condition. For a given function $g \in H^{1/2}(S)$, search

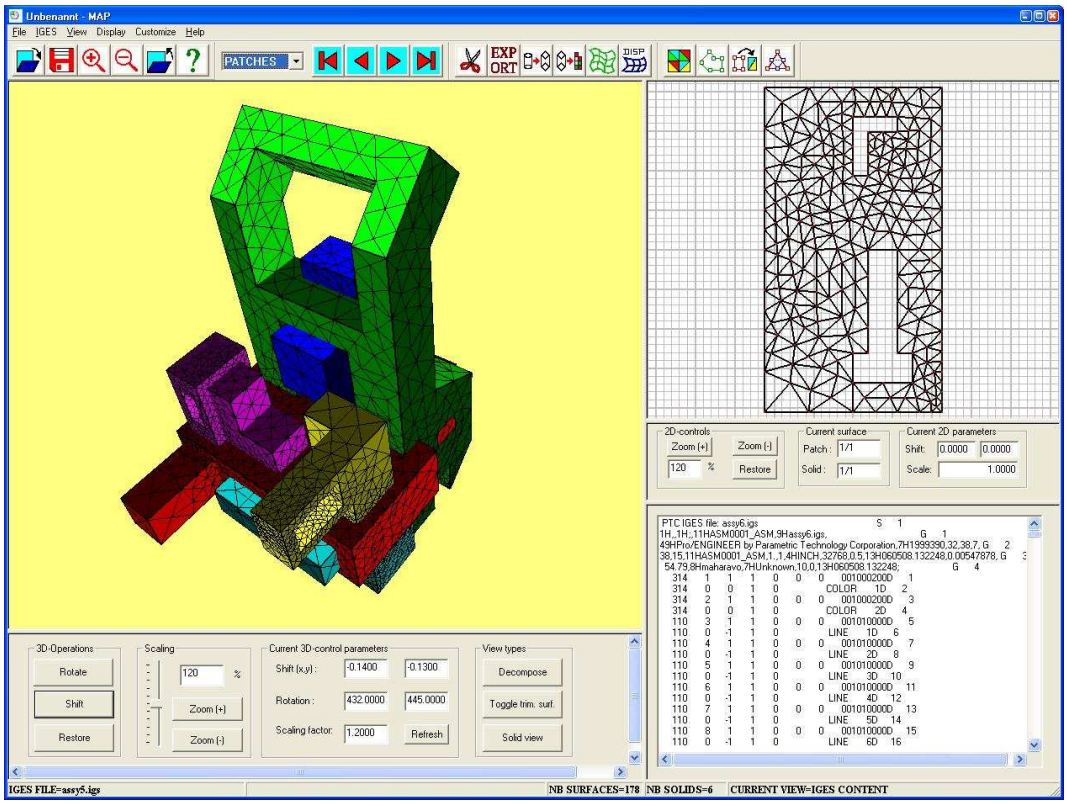


Figure 2: Main window

for some $\mathcal{U} \in H^1(\Omega)$ such that

$$\left\{ \begin{array}{rcll} \Delta \mathcal{U} & = 0 & \text{in} & \Omega \\ \mathcal{U} & = g & \text{on} & S, \end{array} \right. \tag{1}$$

where $\Omega \subset \mathbf{R}^3$ with boundary $S := \partial\Omega$.

Since we deal with a 3D problem, this can be reduced [9] to the solving of the following integral equation [9, 1] of the second kind on the 2D manifold S

$$u(\mathbf{x}) + \int_S \mathbf{k}(\mathbf{x}, \mathbf{t}) u(\mathbf{t}) d\mathbf{t} = -2g(\mathbf{x}) \qquad \forall\, \mathbf{x} \in S, \tag{2}$$

in which the unknown is $u \in H^{1/2}(S)$ and $\mathbf{k}$ is some bivariate kernel function defined on the 2D-manifold S. By introducing the double layer operator:

$$(\mathcal{K} f)(\mathbf{x}) := \int_S \mathbf{k}(\mathbf{x}, \mathbf{t}) f(\mathbf{t}) d\mathbf{t} \qquad \text{for } \mathbf{x} \in S, \tag{3}$$

the original unknown $\mathcal{U}$ of the Laplace problem (1) can be deduced from u

by applying

$$\mathcal{U} = \mathcal{K}u. \tag{4}$$

With the double layer operator $\mathcal{K}$ in place, we can introduce a second operator $\mathcal{A}$ defined by

$$(\mathcal{A}f)(\mathbf{x}) := f(\mathbf{x}) + (\mathcal{K}f)(\mathbf{x}) \qquad \forall\, \mathbf{x} \in S. \tag{5}$$

We obtain therefore the following variational formulation: search for $u \in H^{1/2}(S)$ such that

$$(\mathcal{A}u, v)_S = (f, v)_S \qquad \forall\, v \in H^{1/2}(S)\,. \tag{6}$$

In general, the approximation scheme consists in considering a finite dimensional space R_h where we approximate the function u by $u_h \in R_h$ in which we solve the following problem

$$(\mathcal{A}u_h, v_h)_S = (f, v_h)_S \qquad \forall\, v_h \in R_h. \tag{7}$$

The wavelet Galerkin method [19, 10, 3] is a special case of (7) in which the construction of the finite dimensional space R_h is done by means of multiresolution techniques. That method requires that one generates wavelets on the manifold S.

On the other hand, one needs to use numerical quadratures to evaluate the integrals involved in the entries of the stiffness matrix which results from the equation (7) when applied to the Wavelet Galerkin scheme. If the parametric function γ_i is a diffeomorphism, integrations over the manifold S can be transformed on the unit square:

$$\int_S u(\mathbf{x})v(\mathbf{x})\, d\sigma(\mathbf{x}) = \sum_{i=1}^{N} \int_{[0,1]^2} u(\gamma_i(\mathbf{s}))v(\gamma_i(\mathbf{s}))\sqrt{\det\left(K_i(\mathbf{s})\right)}\, d\mathbf{s}. \tag{8}$$

For mesh-based numerical solvers, the finite dimensional space R_h is usually chosen to be piecewise polynomials. That is, if we denote by $\mathcal{P}^k$ the set of polynomials of degree at most k, then

$$R_h \subset \{p : p|_T \in \mathcal{P}^k \quad \forall\, T \in \mathcal{M}_h\}. \tag{9}$$

There are basically two types of approximation methods which use a mesh: h-version and p-version. The degrees of the polynomials inside the triangles are kept constant if one uses the h-version approaches. That is, if we want to have more accurate approximation u_h of the solution u to the integral equation, then we have to refine the mesh $\mathcal{M}_h$. For that case, we need an a-posteriori error estimator $\varepsilon_T(u_h)$ which has the following property:

$$\lambda_1 \sum_{T\in\mathcal{M}_h} \varepsilon_T(u_h) \le \|u - u_h\| \le \lambda_2 \sum_{T\in\mathcal{M}_h} \varepsilon_T(u_h), \tag{10}$$

where λ_1 and λ_2 are positive constants independent of u and u_h. There are different methods of obtaining a-posteriori error estimators which can be used to identify the parts of the mesh that need to be refined. The special property of an a-posteriori error estimator $\varepsilon_T(u_h)$ is that it can be computed without knowing the function u. If the value of the a-posteriori error estimator $\varepsilon_T(u_h)$ with respect to the triangle T exceeds some prescribed value ε_0, then we subdivide the triangle T into several subtriangles. In the case of p-versions, the polynomial degrees are allowed to be variable. In order to obtain a better accuracy, the mesh $\mathcal{M}_h$ does not need to be refined. Instead, we increase polynomial degrees k of the piecewise polynomials in relation (9). A combination *hp*-version also exists in order to improve numerical performance.

3 Problem setting

Let us consider a closed surface $S \subset \mathbf{R}^3$ that is given as a collection of M trimmed parametric surfaces S_1, $\cdots$, S_M defined on the 2D-domains $\mathcal{D}_1$, $\cdots$, $\mathcal{D}_M$ which are multiply connected regions in $\mathbf{R}^2$. The external and internal (when relevant) boundary curves of each domain $\mathcal{D}_i$ are supposed to be composite curves. We suppose further that the parametric functions defining S_i

$$\psi_i : \mathcal{D}_i \longrightarrow S_i \tag{11}$$

are diffeomorphisms. Such a surface representation is usually met in the B-Rep settings of the shell which bounds a solid. In that case, the surface S practically represents the boundary of a solid.

Furthermore, we do not allow the existence of *cusps* at the boundaries of any $\mathcal{D}_i$. That is, if we suppose that the closed curve representing a boundary (exterior or interior) of $\mathcal{D}_i$ is given by the parametric curve κ, then we must have the following.

(B1) For all τ, we have $\dot{\kappa}(\tau) \neq \mathbf{0}$.

(B2) For all $\kappa(\tau)$ belonging to the boundary of $\mathcal{D}_i$,

$$\lim_{t \to \tau^-} \dot{\kappa}(t) \neq -\lambda \lim_{t \to \tau^+} \dot{\kappa}(t) \qquad \forall\, \lambda > 0. \tag{12}$$

Additionally, the curve κ is supposed to be bijective piecewise polynomial which can only have corners (discontinuity of tangents) at the segment separators [18].

In this document, we will discuss about the implementation of two *separate* geometric problems. First of all, we would like to tessellate S into m four-

sided subsurfaces F_i:

$$S = \bigcup_{i=1}^{m} F_i \,. \tag{13}$$

For that end, we will aim at having a splitting which is conforming. That is to say, every two different subregions F_i and F_j share a complete edge or they share a single corner or they are disjoint. Our main purpose is to keep the number m of surfaces F_i small. However, we do not intend to compute the globally optimal tessellation that minimizes m because the computational cost would be extremely high. Another requirement which is related to the first one is to generate a diffeomorphism γ_i from the unit square onto each four-sided surface F_i

$$\gamma_i : [0,1]^2 \longrightarrow F_i. \tag{14}$$

It is that additional requirement that makes the implementation and theoretical aspects of this problem very complicated. We will use transfinite interpolation techniques in order to deal with the mappings on patches having four curved sides. If we are unable to find a diffeomorphism, then we subdivide the four-sided surface F_i into a few four-sided subsurfaces.

The second problem that we treat in this paper is the generation of a mesh $\mathcal{M}_h$ on the surface S. There are some numerical solvers [1, 9] of integral equations which require the input geometric information to be represented as a mesh $\mathcal{M}_h$. The two standard assumptions that one expects from the triangular elements of the mesh $\mathcal{M}_h$ are the following:

(A1) The intersection of two triangles having nonempty intersection is either a node or a complete edge.

(A2) The smallest angle $\alpha_{\min}(T)$ inside each triangle T is larger than some prescribed threshold $\alpha_0 > 0$.

Note that if the lengths of the three edges in any triangle $T \in \mathcal{M}_h$ are proportional, then condition (A2) follows. That is, if we have

$$h_{\max}(T) \approx h_{\min}(T), \tag{15}$$

where $h_{\max}(T)$ and $h_{\min}(T)$ are the lengths of the longest and shortest edges of T respectively then (A2) holds. A mesh having triangles satisfying (15) is usually nicely shaped.

Before going any further, note that those above two problems do not have anything to do with one another. We have combined the software for treating them because both of them come from an IGES file and they are used for boundary integral equations.

4 Programming tools and MFC settings

The purpose of this section is to comment about the main points of our implementations whose main window can be found in Fig. 1 and Fig. 2. Generally speaking, our software has four major large components: instructions for geometric algorithms, visualization routines, IGES treatment procedures and user interface functions. The source codes of the software have been implemented with the help of a combination of C and C++. Since there are a lot of GUI (Graphical User Interface) applications, we use Win32 and MFC (Microsoft Foundation Classes) in order to implement the controls which facilitate user interactions. The display of graphical scenes is processed by means of OpenGL and GLUT [12, 14]. In fact, our window panes can exhibit 2D as well as 3D graphical scenes which can be manipulated by moving the mouse or by selecting some resource items such as formview button, menu entry, or toolbar button. As for the input geometric data, they are extracted from IGES files. Since we can only deal with ascii IGES files for now, the user has to convert any binary IGES files into ascii ones before loading them into our software. We use usual C routines like `fscanf` and string operations to extract the relevant data from IGES files.

Although we need various displays, we chose to use the MFC type SDI (Single Document Interface) instead of MDI (Multiple Document Interface). In order to enable the display of several sceneries simultaneously, the client window is split into several disproportional panes having different views. Some panes contain graphical scenes and others have formviews. Some splitters need T-shaped boundary because the panes do not have the form of a uniform array. As long as possible, we use the MFC ClassWizard with its message map handlers in order to assign new procedures to specified actions. The Document and Frame classes that are initially generated by the AppWizard have been modified in order that they fit with our particular purpose. For instance, some important part of the Frame class can be found below together with the generated message map functions.

```
class CMainFrame : public CFrameWnd
{protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)
// Attributes
public:
    BOOL         m_initSplitters; //Splitters were initialized
    CSplitterWnd m_mainSplitter;  //Splitter main window
    CSplitterWnd m_wndSpSub_left; //Splitter left subwindow
    CSplitterWnd m_wndSpSub_right;//Splitter right subwindow
    [...]
```

```
protected:
    CStatusBar       m_wndStatusBar;
    CToolBar         m_wndToolBar;    //main toolbar
    CAdvanceToolBar  m_advToolBar;    //navigation toolbar
    CToolBar         m_decomp_povray; //decomposition toolbar
    CFont            m_StatusBarFont; //font for status bar
    [...]
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnSize(UINT nType, int cx, int cy);
    [...]
    afx_msg void OnUpdateIndicatorVaryingView(CCmdUI* pCmdUI);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

In order that we can use the digitized CAD information in our geometric algorithms, several data structures and pertaining routines must be implemented. The problem about using an IGES file for storing the input geometric components is related to the loading, parsing and evaluating processes. During the treatments of the selected entities in Table 2, the following tasks have to be implemented.

(a) Automatic extraction of the size of the entities which require large storage: we need the different integer values which indicate the volume required in the data structures. That is a necessary step in order that accurate memory allocations and deallocations can be feasible. This step has to happen before the real entity extraction can take place.

(b) A large number of simple routines for transforming the IGES parameters to acceptable inputs in the code: those routines include location of the parameter delimiters and record delimiters so that appropriate records could be withdrawn from IGES files. This is necessary in order to access a specified record in the Parameter Data section which is pointed by an entity in the Directory Entry section.

(c) Extraction of relevant entities from the IGES file: one has to write a single routine for each entity because the IGES description of parameter sequences varies from one entity to another in the Parameter Data section.

(d) Loading of the parameters of the entities from the IGES file to the C-data structures: we need convertion of the IGES data formats into formats which are understood by the C programming language. We have generated a data structure for every entity. The following is for example the data structure that is used to store a NURBS surface [15]. The other entities can

be structured in a similar way.

```
typedef struct nurbs_surface{
      int nu;          //order in u-variable
      int nv;          //order in v-variable
      int ku;          //smoothness in u-variable
      int kv;          //smoothness in v-variable
      point **d;       //3D coordinates of control points
      double **w;      //weights
      double *tau_u;   //knot sequence along u-direction
      double *tau_v;   //knot sequence along v-direction
      double u0;       //[u0,u1]=interval of def along u-dir
      double u1;       //[u0,u1]=interval of def along u-dir
      double v0;       //[v0,v1]=interval of def along v-dir
      double v1;       //[v0,v1]=interval of def along v-dir
      int prop1;       //open or closed in u-direction
      int prop2;       //open or closed in v-direction
      int prop3;       //rational or polynomial
      int prop4;       //periodic or not in u-direction
      int prop5;       //periodic or not in v-direction
}nurbs_surface;
```

In order to quickly understand the above structure, we recall that a NURBS surface is mathematically [16] represented as

$$\mathbf{x}(u,v) = \frac{\sum_{i=0}^{n_u}\sum_{j=0}^{n_v}\omega_{ij}\mathbf{d}_{ij}N_i^{k_u}(u)N_j^{k_v}(v)}{\sum_{i=0}^{n_u}\sum_{j=0}^{n_v}\omega_{ij}N_i^{k_u}(u)N_j^{k_v}(v)} \qquad (u,v)\in[u_0,u_1]\times[v_0,v_1] \tag{16}$$

where the bases functions are defined as

$$N_i^k(t) := \frac{t-\theta_i}{\theta_{i+k-1}-\theta_i}N_i^{k-1}(t) + \frac{\theta_{i+k}-t}{\theta_{i+k}-\theta_{i+1}}N_{i+1}^{k-1}(t) \tag{17}$$

with N_i^1 being the characteristic function of $[\theta_i, \theta_{i+1})$. Consequently, the knowledge of the weights ω_{ij}, control points $\mathbf{d}_{ij}$ as well as the knot sequences is enough to completely describe a NURBS surface. When the weights ω_{ij} have unit values, $\mathbf{x}(u,v)$ becomes a polynomial B-Spline.

(f) Evaluation functions: one step that cannot be neglected is the entity evaluations because one has to evaluate the loaded entities repeatedly in our subsequent geometric algorithms. We have to implement an evaluation routine for each and every given data structure. For instance, we need a de-Boor algorithm to evaluate a B-spline surface.

Entities	Number
source files (*.c and *.cpp)	167
header files	34
directories	8
classes	20
typedef's	95
resource files	7

Table 1: Four fields in terminate section

At the beginning, the size of this project was not large but as more codes were implemented in order to make different algorithms efficient, its dimension has also grown. Additionally, making everything user-friendly needs lots of code. The current respective number of entities is enlisted on Table 1. Like many softwares, this one is constantly developed and improved. As a consequence, those numbers are subject to change in the future.

5 Toolbars and status bar

We use toolbars in order to execute specified actions which could be file operations, geometric routines or visualization tasks. Each button on a toolbar has generally an equivalent entry in the menu. We have rescaled the toolbar buttons to have size 32×32 instead of the default 16×16. In fact, we have four toolbars (see Fig. 3) having the following identifiers

IDR_MAINFRAME

IDR_TOOLBAR_ADVANCE

IDR_DECOMP_POVRAY

IDR_MESH_GENERATION.

Now, we would like to describe the functionalities and the identifiers of the different toolbar buttons. Some of them are immediately clear when you see the descriptive graphs on the icons but other ones really need some explanations. For the main toolbar that is displayed in Fig. 3(a), the identifier and purpose of each member button are as follows.

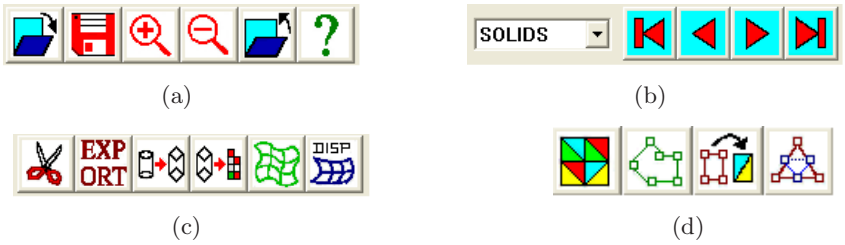
ID_FILE_OPEN: opening an IGES file

ID_FILE_SAVE: storing the results in IGES or *.dat files

ID_BUTTON_ZOOM_PLUS: magnification of a 2D/3D scene

ID_BUTTON_ZOOM_MINUS: miniature view of a 2D/3D scene

ID_FILE_CLOSE: closing an IGES file



(a) (b) (c) (d)

Figure 3: (a)Main toolbar (b)Navigation toolbar with a combo-box (c)Decomposition toolbar (d)Mesh generation toolbar

`ID_BUTTON_HELP`: getting a simple help

Now we would like to comment about the navigation toolbar (Fig. 3(b)) which contains a combobox. The three possible entries of the combo box are `PATCHES`, `SOLIDS`, `CELLS`. They specify the roles of the arrow buttons of the navigation toolbar. If `SOLID` is selected, then we navigate through the solids of an assembly. If `PATCHES` is chosen, the arrow buttons modify the indices of the trimmed surfaces or patches in the current solid. `CELLS` is selected if one wants to navigate through the list of quadrilaterals or four-sided domains on the current trimmed surface. The functionalities of the members of the navigation toolbar are as follows (object is what the combo-box specifies):

`ID_START`: going to the starting object

`ID_BACKWARD`: going backward once

`ID_FORWARD`: going forward once

`ID_END`: going to the final object

The third toolbar which is shown in Fig. 3(c) deals with the decomposition of the model in the loaded IGES file into a set of simpler structures. Additionally, it is used to generate a file script which can be incorporated in the POV-RAY program for rendering tasks as post-processing. More specifically, the functionalities of the buttons of the decomposition toolbar are as follows:

`ID_TOOL_DECOMPOSE`: decomposing into foursided patches

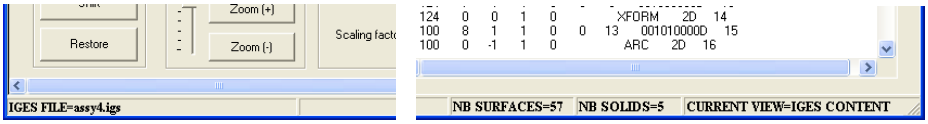
`ID_TOOL_POVRAY`: exporting for POVRAY rendering

`ID_TOOL_MODSTRUCT`: generating a polyhedral approximation

`ID_TOOL_GRID`: decomposition from a polyhedral approximation

`ID_BUTTON_TRANSFINITE`: generating the diffeomorphisms

`ID_DISPLAY_TRANSFINITE`: display the diffeomorphisms



(a) (b)

Figure 4: Status bar

The fourth toolbar which is shown in Fig. 3(d) is for the generation of meshes. The purpose of its toolbar buttons is as follows.

`ID_GENERATE_MESH`: generating a mesh directly from the IGES file

`ID_BOUND_GRAPH`: polygonal boundary approximation

`ID_TRIANGULATE`: mesh from polygonal approximation

`ID_REFINE_MESH`: refining the current mesh

Note that not all buttons are always functional. For example, you cannot use the button for the decomposition from a polyhedral approximation unless the model boundaries have already been split into polygonal approximations.

In the remaining of this section, we will discuss about the settings of the status bar. We do not use the standard status bar of MFC which indicates the currently pressed special keys on the keyboard. Rather, we display on the status bar three items as illustrated in Fig 4: the name of the currently loaded IGES file, the number of all trimmed surfaces, the number of solids in the current assembly, and a parameter which identifies the type of view that is displayed at the right-bottom window pane. Instead of using the standard font, we use another font type and font size which are specified by the following logical font structure.

```
LOGFONT lf;
[...]
lf.lfWeight             = FW_BOLD;
lf.lfItalic             = FALSE;
lf.lfUnderline          = FALSE;
lf.lfStrikeOut          = FALSE;
lf.lfCharSet            = DEFAULT_CHARSET;
lf.lfOutPrecision       = OUT_DEFAULT_PRECIS;
lf.lfClipPrecision      = CLIP_DEFAULT_PRECIS;
lf.lfQuality            = DEFAULT_QUALITY;
lf.lfPitchAndFamily     = VARIABLE_PITCH | FF_ROMAN;
[...]
```

For the realization, we generate the following indicator table:

```
static UINT indicators[] =
   {ID_INDICATOR_IGESFILE,
    ID_SEPARATOR,
    ID_INDICATOR_NB_SURFACES,
    ID_INDICATOR_NB_SOLIDS,
    ID_INDICATOR_BOTVIEW
   };
```

6 Formviews and property sheet

The main window contains several formviews which are embedded in different panes. In this section, we will mainly comment about the formview located at the left-bottom window pane. Its member which have been designed by means of the resource editor are displayed in Fig. 5. They are used to control the view on the top-left window pane which shows OpenGL 3D scenes. The leftmost buttons of the formview are used to change the role of the mouse button. When "rotate" button is selected, the mouse is used to turn the 3D scenes. The "shift" button is used to make the mouse button shift the graphical scene. When the "restore" button is hit, the 3D scene takes its original position, size and posture. On the other hand, the manipulation parameters are shown in the corresponding edit boxes. You find there the current shifting and scaling parameters together with the two current spherical coordinates for the rotation. The refresh button is used to apply the data in the edit box to the 3D scene.

In the right-bottom window pane, we can display the summary of the geometric data which are stored in the IGES file in form of a property sheet. It is contained in a formview which contains the number of base surfaces and the number of solids besides the property pages. The property sheet summarizes the important property of the base surfaces of the trimmed surfaces. For example, the property page corresponding to a surface of revolution displays the parameters of the axis of revolution, the type of generatrix and the angle range (starting and terminating angles).

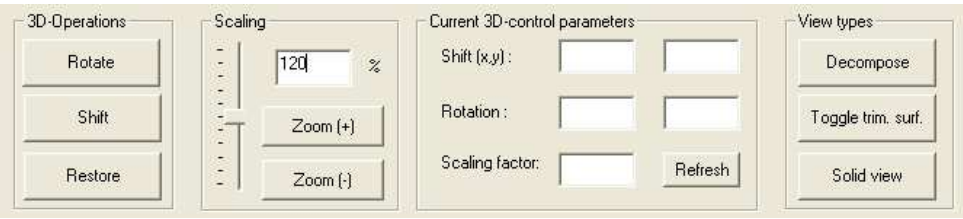
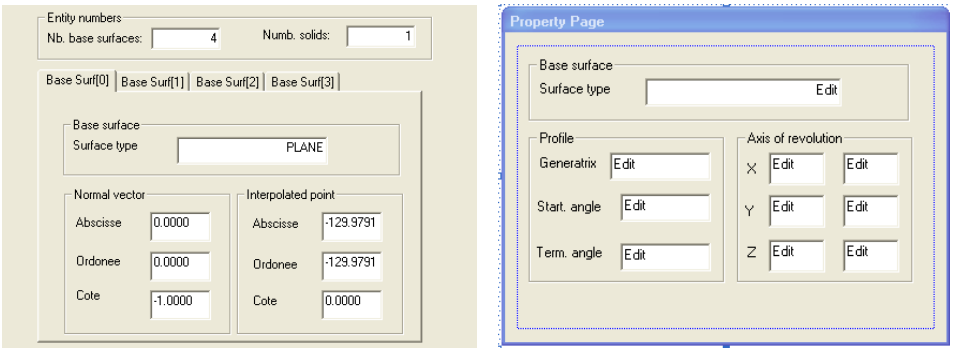


Figure 5: Formview for the manipulation of 3D scene



(a) (b)

Figure 6: (a)Property sheet (b)Property page of the summary of a surface of revolution

For each property page, we have to generate two subclasses which are derived respectively from the MFC classes `CPropertySheet` and `CPropertyPage` such as:

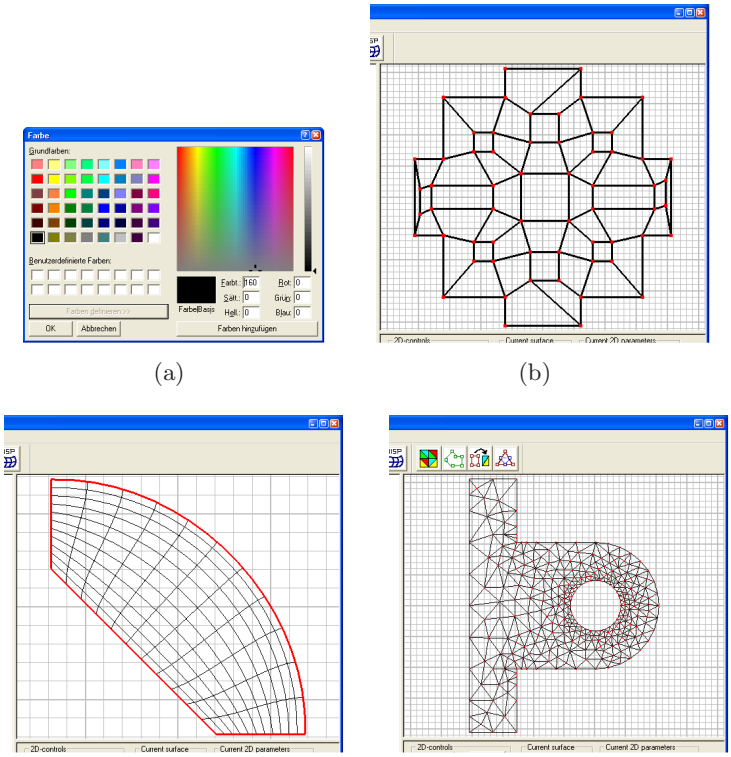
```
class CSummaryPSheet : public CPropertySheet
class CSplinePPage   : public CPropertyPage
```

The corresponding controls for the property pages are drawn as usual with the help of the resource editor. In the formview which contains the property sheet, we put a picture control in place of the future property sheet. The placement of the property sheet is then done programmatically by overriding the `OnInitialUpdate()` member function of the corresponding class.

In the top-right window pane, we can display three different 2D-graphical scenes. First, one can show there the quadrangulations (Fig. 7(b)) corresponding to the four-sided splitting. Second, that pane illustrates the transfinite interpolation on each four-sided domain (Fig. 7(c)). We show there the images of the u-constant and v-constant isolines. The third possibility (Fig. 7(d)) is to display planar meshes on the parameter domains of the trimmed surfaces. In the background, there is grid which facilitates the 2D-views and which can be refined or coarsened by choosing the relevant menu entry.

7 File I/O

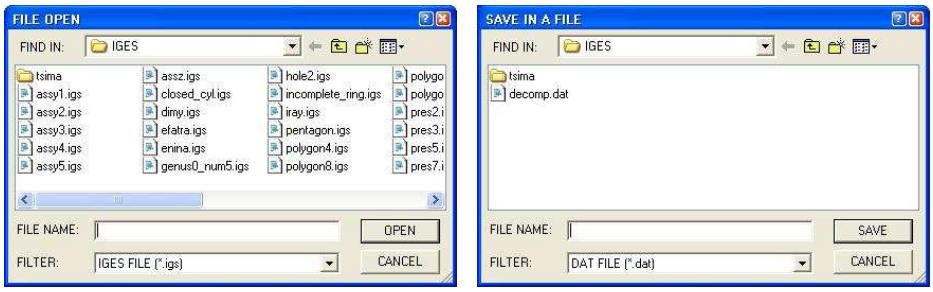
The geometric data which are input in our program are stored in IGES files which one generates with the help of some CAD system such as Pro-Engineer. When this software has been implemented for the first time, it



(a) (b)

(c) (d)

Figure 7: (a)Dialog for color customization; OpenGL 2D display: (b)Quadrangulation (c)Diffeomorphism (d) Planar mesh



(a) (b)

Figure 8: File dialogs (a) IGES input, (b) File export

was a console application. The names of the IGES files was hardcoded inside the program. Since that was very inconvenient, we use dialog application as in Fig. 8(a) in order to easily specify the file names and to locate the containing directory. We use the filter `*.igs` in order that the user can only load IGES files. That means, the `OnFileOpen()` function was overridden so as it becomes as follows.

```
void CMAPApp::OnFileOpen()
{static char szFilter[] = "IGES FILE (*.igs)|*.igs||";
CNewFileDialog fileName(TRUE,"*.igs",NULL,OFN_HIDEREADONLY |
OFN_OVERWRITEPROMPT,szFilter);
fileName.m_ofn.lpstrTitle="FILE OPEN";
if(fileName.DoModal() != IDOK)
     return;
CWinApp::OpenDocumentFile(fileName.GetPathName());
}
```

The same remark holds for data storage inside a file by using the dialog in Fig. 8(b). Our software supports two formats for the output of the four-sided decomposition. First, we implemented an IGES export which calls NURBS/B-Spline subdivision routines many times. Another format of the output files has the ending `*.dat`. That second format has the following structure.

```
NUMBER OF NODES=nnd
NUMBER OF ELEMENTS=nel
NUMBER OF EDGES=ned
NODE[0]=[x0,y0]  FLAG=fl0   REAL PARAMETER=t0
NODE[1]=[x1,y1]  FLAG=fl1   REAL PARAMETER=t1
....
```

```
ELEMENT[0]=[a0,b0,c0,d0]  [e0,f0,d0,k0]
ELEMENT[1]=[a0,b0,c0,d0]  [e0,f0,d0,k0]
....

EDGE[0]=[n0,m0]   INCIDENT=[j0,i0]
EDGE[1]=[n1,m1]   INCIDENT=[j1,i1]
...
```

The comments which are in upper cases can be enabled or disabled. In the above structure, the flag information has the following values.

Flag(x)=-1 if the node x is located on the exterior boundary

Flag(x)=i if the node x is located on the i-th interior boundary

Flag(x)=-2 otherwise (i.e. x is a nonboundary node)

As for the mesh generation, the format of the output is the usual list of vertices and the list of triangles.

8 IGES interface as geometric storage

Since we extract the geometric information from an IGES file, it is important to understand the structure of an IGES file. In order that we can load an IGES file into our program, many routines which require knowledge of IGES sections had to be implemented. As a consequence, we describe here the IGES format briefly. While reading this short description, we recommend that the readers compare the explanation with Fig. 9 which shows a simple example of an IGES file.

8.1 Entity classification and data types

We distinguish two main entity categories in an IGES file: geometric and nongeometric entities. The first group contains information which is required to describe shapes such as curves, surfaces, solids and relationships between them. The second group is needed for other graphical or computational purposes that include color properties in RGB, CMY, or HLS format, or physical units such as measures of mass, time, temperature, luminous intensity which are needed in physical simulations or numerical applications. Descriptive properties such as text fonts are also classified in the non-geometric entities. Since we use IGES to store model for four-sided decomposition, we are only interested in geometric entities.

IGES standard supports several data types which include strings, integers,

```
 PTC IGES file: H:\IGES_files\open_cylinder.igs                          S      1
1H,,1H;,7HPRT0001,31HH:\IGES_files\open_cylinder.igs,                    G      1
49HPro/ENGINEER by Parametric Technology Corporation,7H2001150,32,38,7, G      2
38,15,7HPRT0001,1.,1,4HINCH,32768,0.5,13H021014.152641,0.00865991,       G      3
86.6025,8Hmaharavo,7HUnknown,10,0,13H021014.152641;                      G      4
     124       1       1       1       0       0       0       001000000D      1
     124       0       0       1       0                     XFORM       1D      2
     100       2       1       1       0       0       1       001010000D      3
     100       0       0       1       0                       ARC       1D      4
     124       3       1       1       0       0       0       001000000D      5
     124       0       0       1       0                     XFORM       2D      6
     100       4       1       1       0       0       5       001010000D      7
     100       0       0       1       0                       ARC       2D      8
     110       5       1       1       0       0       0       001010000D      9
     110       0       0       1       0                      LINE       1D     10
     124       6       1       1       0       0       0       001000000D     11
     124       0       0       1       0                     XFORM       3D     12
     100       7       1       1       0       0      11       001010000D     13
     100       0       0       1       0                       ARC       3D     14
     124       8       1       1       0       0       0       001000000D     15
     124       0       0       1       0                     XFORM       4D     16
     100       9       1       1       0       0      15       001010000D     17
     100       0       0       1       0                       ARC       4D     18
     110      10       1       1       0       0       0       001010000D     19
     110       0       0       1       0                      LINE       2D     20
     110      11       1       1       0       0       0       001010000D     21
     110       0       0       1       0                      LINE       3D     22
     110      12       1       1       0       0       0       001010000D     23
     110       0       0       1       0                      LINE       4D     24
     120      13       1       1       0       0       0       001010000D     25
     120       0       0       1       0                      SREV       1D     26
     110      14       1       1       0       0       0       001010000D     27
     110       0       0       1       0                      LINE       5D     28
     102      15       1       1       0       0       0       001010000D     29
     102       0       0       1       0                    CCURVE       1D     30
     110      16       1       1       0       0       0       001010500D     31
     110       0       0       2       0                      LINE       6D     32
     110      18       1       1       0       0       0       001010500D     33
     110       0       0       2       0                      LINE       7D     34
     110      20       1       1       0       0       0       001010500D     35
     110       0       0       2       0                      LINE       8D     36
     110      22       1       1       0       0       0       001010500D     37
     110       0       0       2       0                      LINE       9D     38
     102      24       1       1       0       0       0       001010500D     39
     102       0       0       1       0                    CCURVE       2D     40
     142      25       1       1       0       0       0       001010500D     41
     142       0       0       1       0                    UV_BND       1D     42
     144      26       1       1       0       0       0       000000000D     43
     144       0       0       1       0                    TRM_SRF       1D     44
     110      27       1       1       0       0       0       001010000D     45
     110       0       0       1       0                      LINE      10D     46
     110      28       1       1       0       0       0       001010000D     47
     110       0       0       1       0                      LINE      11D     48
     120      29       1       1       0       0       0       001010000D     49
     120       0       0       1       0                      SREV       2D     50
     110      30       1       1       0       0       0       001010000D     51
     110       0       0       1       0                      LINE      12D     52
     102      31       1       1       0       0       0       001010000D     53
     102       0       0       1       0                    CCURVE       3D     54
     110      32       1       1       0       0       0       001010500D     55
     110       0       0       2       0                      LINE      13D     56
     110      34       1       1       0       0       0       001010500D     57
```

```
     110       0       0       2       0                    LINE      14D     58
     110      36       1       1       0       0       0       001010500D     59
     110       0       0       2       0                    LINE      15D     60
     110      38       1       1       0       0       0       001010500D     61
     110       0       0       1       0                    LINE      16D     62
     102      39       1       1       0       0       0       001010500D     63
     102       0       0       1       0                   CCURVE       4D     64
     142      40       1       1       0       0       0       001010500D     65
     142       0       0       1       0                  UV_BND       2D     66
     144      41       1       1       0       0       0       000000000D     67
     144       0       0       1       0                 TRM_SRF       2D     68
     406      42       1       1       0       0       0       001000000D     69
     406       0       0       1      15                    PROP       1D     70
     402      43       1       1       0       0       0       000000300D     71
     402       0       0       2       7                   LAYER       1D     72
     406      45       1       1       0       0       0       001000000D     73
     406       0       0       1      15                    PROP       2D     74
     402      46       1       1       0       0       0       000000300D     75
     402       0       0       2       7                   LAYER       2D     76
124,-1D0,0D0,0D0,7.5D1,0D0,-1D0,0D0,1.5D2,0D0,0D0,1D0,0D0;             1P      1
100,0D0,0D0,0D0,2.5D1,0D0,-2.5D1,0D0;                                  3P      2
124,1D0,0D0,0D0,7.5D1,0D0,-1D0,0D0,1.5D2,0D0,0D0,-1D0,-5D1;            5P      3
100,0D0,0D0,0D0,2.5D1,0D0,-2.5D1,0D0;                                  7P      4
110,5D1,1.5D2,0D0,5D1,1.5D2,-5D1;                                      9P      5
124,1D0,0D0,0D0,7.5D1,0D0,1D0,0D0,1.5D2,0D0,0D0,1D0,0D0;              11P      6
100,0D0,0D0,0D0,2.5D1,0D0,-2.5D1,0D0;                                 13P      7
124,-1D0,0D0,0D0,7.5D1,0D0,1D0,0D0,1.5D2,0D0,0D0,-1D0,-5D1;           15P      8
100,0D0,0D0,0D0,2.5D1,0D0,-2.5D1,0D0;                                 17P      9
110,1D2,1.5D2,0D0,1D2,1.5D2,-5D1;                                     19P     10
110,7.5D1,1.5D2,0D0,7.5D1,1.5D2,1D0;                                  21P     11
110,1D2,1.5D2,-5.1D1,1D2,1.5D2,1D0;                                   23P     12
120,21,23,3.078760800518D0,6.346017160251D0;                          25P     13
110,5D1,1.5D2,-5D1,5D1,1.5D2,0D0;                                     27P     14
102,4,3,19,7,27;                                                      29P     15
110,9.807692307692D-1,3.141592653590D0,0D0,9.807692307692D-1,         31P     16
6.283185307180D0,0D0;                                                 31P     17
110,9.807692307692D-1,6.283185307180D0,0D0,1.923076923077D-2,         33P     18
6.283185307180D0,0D0;                                                 33P     19
110,1.923076923077D-2,6.283185307180D0,0D0,1.923076923077D-2,         35P     20
3.141592653590D0,0D0;                                                 35P     21
110,1.923076923077D-2,3.141592653590D0,0D0,9.807692307692D-1,         37P     22
3.141592653590D0,0D0;                                                 37P     23
102,4,31,33,35,37;                                                    39P     24
142,0,25,39,29,1;                                                     41P     25
144,25,1,0,41;                                                        43P     26
110,7.5D1,1.5D2,0D0,7.5D1,1.5D2,1D0;                                  45P     27
110,1D2,1.5D2,-5.1D1,1D2,1.5D2,1D0;                                   47P     28
120,45,47,-6.283185307180D-2,3.204424506662D0;                        49P     29
110,1D2,1.5D2,-5D1,1D2,1.5D2,0D0;                                     51P     30
102,4,13,9,17,51;                                                     53P     31
110,9.807692307692D-1,0D0,0D0,9.807692307692D-1,                      55P     32
3.141592653590D0,0D0;                                                 55P     33
110,9.807692307692D-1,3.141592653590D0,0D0,1.923076923077D-2,         57P     34
3.141592653590D0,0D0;                                                 57P     35
110,1.923076923077D-2,3.141592653590D0,0D0,1.923076923077D-2,         59P     36
0D0,0D0;                                                              59P     37
110,1.923076923077D-2,0D0,0D0,9.807692307692D-1,0D0,0D0;              61P     38
102,4,55,57,59,61;                                                    63P     39
142,0,49,63,53,1;                                                     65P     40
144,49,1,0,65;                                                        67P     41
406,1,17H02___PRT_ALL_AXES;                                           69P     42
402,18,3,7,9,13,17,19,21,23,25,29,39,43,45,47,49,53,63,67,0,1,        71P     43
69;                                                                   71P     44
406,1,18H06___PRT_ALL_SURFS;                                          73P     45
402,18,3,7,9,13,17,19,21,23,25,29,39,43,45,47,49,53,63,67,0,1,        75P     46
73;                                                                   75P     47
S      1G      4D     76P     47                                      T      1
```

Figure 9: Sample of an IGES file

Entity	ID number	IGES-code
Line	110	`LINE`
Circular arc	100	`ARC`
Polynomial/rational B-spline curve	126	`B_SPLINE`
Composite curve	102	`CCURVE`
Surface of revolution	120	`SREV`
Tabulated cylinder	122	`TCYL`
Polynomial/rational B-spline surface	128	`SPLSURF`
Trimmed parametric surface	144	`TRM_SRF`
Transformation matrix	124	`XFORM`

Table 2: Implemented curve, surface and transformation entities

and real numbers. Real numbers are usually represented in the form `rDs`, a floating point number r followed by the character 'D' followed by a signed integer s as `9.807693D-1` or `1.5D2`. This represents a real number r multiplied by ten to the power of s. A string has the format `lHf` in which the integer `l` determines the length of the string to be described. The real string is shown after the letter `H` as `11HProEngineer`.

In the IGES format, a *parameter* is a value which can be integer, string, or floating point and which describes some information in the global section or parameter data. Those parameters are separated by a symbol known as *parameter delimiter* which can be specified or defined in the global section and which has comma (,) as default value. Every entity which is found in the directory entry is detailed by one *record* which is a sequence of parameters. The symbol separating two records is the *record delimiter* whose default value is a semicolon (;).

8.2 IGES file organization

The number of lines of an IGES file depends on the geometric information to be stored. The lines are generally partitioned into five main sections:

(1) Start section,
(2) Global section,
(3) Directory entry,
(4) Parameter data,
(5) Terminate section.

The structure of an IGES file is always organized in 80 columns whose corresponding specific roles are split into three parts. First, columns 1 till 72 contain the most valuable parameter and record values pertaining to the digitized geometry. Those columns include information which varies

according to the section to be described. Second, column 73 contains one letter which specifies the current section. Thus, the five IGES-sections that we have described above are identified respectively by the letters '**S**', '**G**', '**D**', '**P**', and '**T**'. Finally, columns 74 till 80 contain integer data which are right-justified and which indicate the line numbers of every section. In simple words, an IGES file has the structure displayed in Table 3.

1 ... 72	73	74 ... 80
START SECTION	S	1
	S	2
	S	...
GLOBAL SECTION	G	1
	G	2
	G	...
DIRECTORY ENTRY	D	1
	D	2
	D	...
PARAMETER DATA	P	1
	P	2
	P	...
TERMINATE SECTION	T	1

Table 3: IGES file structure

Now, we would like to summarize the purpose of each section of an IGES file. The start section which should have at least one line is a human-readable section in which you can write anything like the directory location of the file. This section usually contains the comments of the sender to the receiver.

In the global section, we find general information such as how to read the current file, where, when and by whom was the file generated. It can also specify the parameter delimiter as well as the record delimiter.

The actual description of the entities takes place in the directory entry and parameter data sections. The directory entry gives in general an overview of the different components of the stored geometry and it points to records in the Parameter Data section which contains the complete information about all parameters. For instance, in the directory entry we can see that we have to deal with NURBS surfaces while the information about the control points, knot sequence and weights cannot be found in the Directory Entry. Every entity has its own identification number which can range from 0 to as many as 514. In Table 2, we summarize the entity numbers of a few important geometries. It is in the Parameter Data section that we can find the actual values of all parameters. The content of the parameter data section varies

Field	Columns	Section
1	1-8	Start
2	9-16	Global
3	17-24	Directory entry
4	25-32	Parameter data

Table 4: Four fields in terminate section

from one entity to another but it has in general the following structure

```
entity number,parameter1,parameter2,...,parameterN;
```

The terminate section consists only of a single line which does not use columns 33-72 and which describes the lengths of the former four sections as described in Table 4.

9 OpenGL encapsulation

All our 2D and 3D graphical scenes are implemented with the help of OpenGL. In order that we can link OpenGL with MFC windows, a Rendering Context (RC) must be created. Otherwise, the OpenGL instructions do not display anything on the screen. We use *wiggle* functions which are usually recognizable by their prefix *wgl* for the preparation of OpenGL setting in windows. Note that *wiggle* is not part of MFC but a common way to merge Win32 with OpenGL.

Before drawing under MFC with OpenGL, the following introductory instructions must be executed. First, one gets a DC (Device Context) which comes from the client area of the main window. Second, one selects a pixel format for the DC by invoking a function `SetupPixelFormat` which does the following: it initializes the Pixel Format Descriptor with the data structure PIXELFORMATDESCRIPTOR. Then, it calls `ChoosePixelFormat` which accepts a handle of the DC and which searches for a pixel format that best approximates the former pixel format descriptor supported by the DC. Afterwards, it invokes `SetPixelFormat` which sets the chosen pixel format. As third step, one creates an RC from that DC by using `wglCreateContext` that takes a handle of a DC as argument and that returns a handle of an RC. Finally, one makes the newly created RC current by using `wglMakeCurrent`. After those preparations, any OpenGL commands can be invoked to draw graphical scenes.

After using OpenGL routines, the DC and RC should be released. In order to release the RC, we first make the RC noncurrent by invoking

`wglMakeCurrent(0,0)`. Then, we delete it with `wglDeleteContext` which takes the handle of the RC as argument. In order to release the DC, one simply needs to invoke `delete` followed by the handle of the DC.

10 Assembly of several solids

The software is featured by its ability to handle a geometric model which is composed of several solids such as those in Fig. 10. The algorithms are then applied to each geometric shell. For the case of assemblies, we cannot yet accept IGES files which do not contain color information. Therefore, the designer has to make sure that while he is preparing the geometric information, each solid has its own color. That is important because it is the simplest way to sort the surface components of each shell.

For the graphical scene we use graph coloring algorithm from discrete mathematics in order that each solid has its own color while two adjacent four-sided patches have distinct colors.

Observing the whole assembly at once usually obscurs some special features of a particular solid as illustrated in Fig. 10(a). It is possible for the user to inspect the four-sided decomposition of each individual solid without considering the assembly by choosing the button "Solid View" and by navigating with the help of the top arrow buttons. That way, it is possible to look every side of each individual solid as in Fig. 10(b).

For the realization, we use the alpha-Blending facility of OpenGL. That is, we include the color definitions between the following instructions:

```
glEnable(GL_BLEND);
```

```
glDisable(GL_BLEND);
```

So, the fourth argument of the color definition in OpenGL is considered:

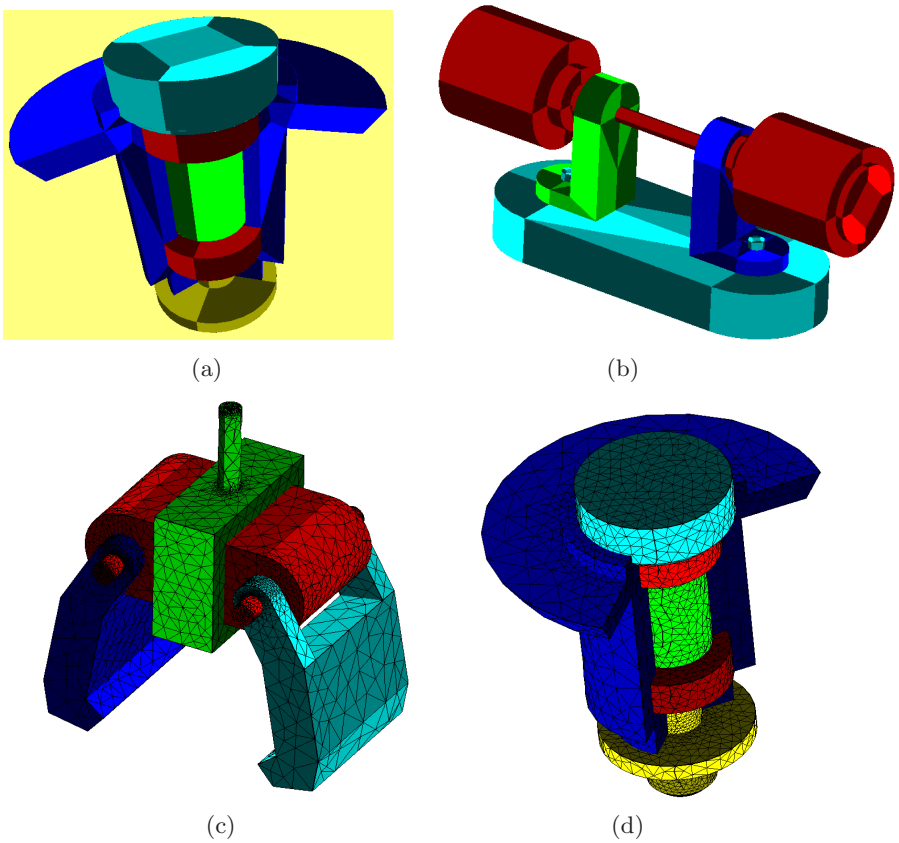
```
glColor4f(...,...,...,alpha);
```

In order to have a good emphasis, the transparent solid has gray basic color. In order that the opaque solid is always visible, it has to be drawn before any other transparent solids are drawn. We choose the OpenGL blending function to have the following arguments:

```
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
```

11 Four-sided decomposition

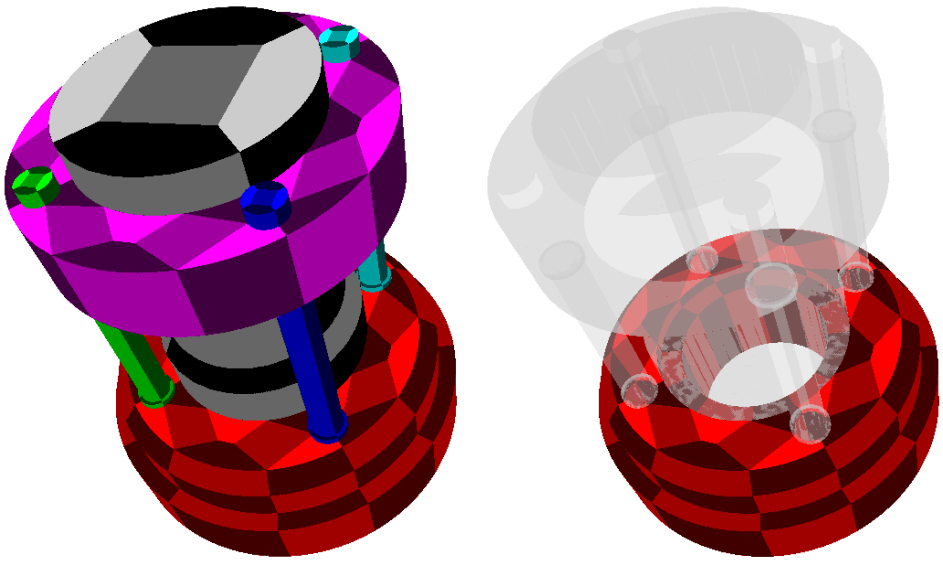
This section will be occupied by the short description of the features of our approach for geometric tessellation. We will consider the decomposition of



(a) (b)

(c) (d)

Figure 10: Some instances of assemblies



(a) (b)

Figure 11: (a)Four-sided splitting of the complete assembly (b)One shell is opaque, the other ones are transparent

a closed surface S given as a set of surfaces $\{S_i\}_{i\in\Lambda}$ into a collection of four-sided subsurfaces:

$$S = \bigcup_{i=1}^{m} F_i \,. \tag{18}$$

We suppose that each S_i is the image by a function ψ_i of a multiply connected 2D-region $\mathcal{D}_i$ having possibly curved boundaries.

Our main approach to achieve (18) consists in splitting the 2D regions $\mathcal{D}_i$ into four-sided regions $Q_{k,i}$:

$$\mathcal{D}_i = \bigcup_{k} Q_{k,i}. \tag{19}$$

Thereto for each $\mathcal{D}_i$, we create an even polygonal approximation $P^{(i)}$ which we decompose into a set of quadrilaterals $q_{k,i}$. In order to obtain the four-sided domains $Q_{k,i}$ from $q_{k,i}$, we replace the straight boundary edges of $q_{k,i}$ by the corresponding curve portion of $\mathcal{D}_i$. That process could generate boundary interferences which need to be detected and repaired.

So as to have a decomposition which is conforming everywhere, we proceed as follows. We approximate the curved boundaries of $\{S_i\}$ by straight line segments separated by nodes $\{X_k\} \subset \mathbf{R}^3$. Then, we make the local splitting (19) in such a way that it is conforming inside $\mathcal{D}_i$ and that it uses only the preimages $\psi_i^{-1}(X_k)$ of the nodes $\{X_k\}$ as boundary vertices. That is, we do not use any additional boundary nodes.

The method that we propose for the local split (19) tessellates a polygon with n boundary vertices into $\mathcal{O}(n)$ convex quadrilaterals. Therefore, if the number of its boundary vertices n_i for all polygons $P^{(i)}$ is smaller than n, then the total number of quadrilaterals is of order $\mathcal{O}(M \cdot n)$. For all examples that we considered, we found that the total number of quadrilaterals is quite small. However, we do not examine how close our local approach comes in average to the globally optimal solution.

Since quadrangulation is an important step in the process of decomposing a trimmed surface into a set of four-sided patches, we would like to recall the main results about algorithmic quadrangulations. In order to realize the above approach, we should solve the problem about quadrilating multiply connected polygons. We show that for a simply connected polygon, either one can chop off a quadrilateral which is not necessarily convex by inserting a cut or one can introduce an internal Steiner point to remove a convex quadrilateral. More precisely, we proved the following result.

Theorem 1 Consider a simple polygon P having at least four vertices, one of the following two statements must hold true:

(Op1) One can remove a quadrilateral which is not necessarily convex by inserting a single cut.

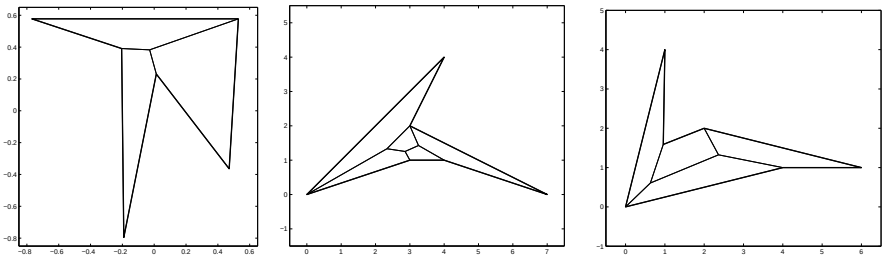
(Op2) There exists a point ω located strictly within P such that one can remove a *convex* quadrilateral by inserting two line segments emanating from ω to two vertices of P.

In order to generalize this result about simply connected polygons to multiply connected ones, the notion of double-edged polygons is introduced. Such a polygon may contain different nodes having the same coordinates but its interior is connected. We prove that the above result about the decomposition of a simply connected polygon holds true for double-edged ones. In its proof, the following generalization of the two-ear theorem for double-edged polygons is used.

Theorem 2 From every double-edged polygon P having more than four vertices, one may chop off two triangles.

In order to generate a double-edged polygon from a multiply connected one, a cut per internal boundary is inserted. Each cut is afterwards replaced by double edges which are traversed in opposite directions.

By applying the above theoretical results, a quadrangulation technique is obtained that repeatedly removes quadrilaterals. Since the process of removing quadrilaterals by inserting cuts might generate a quadrangulation



(a) (b) (c)

Figure 12: Quadrangulation of hexagons with different number of internal Steiner points. No boundary Steiner points are required.

$\mathcal{Q}$ having non-convex quadrilaterals, the resulting quadrangulation need to be converted into another one which contains only convex quadrilaterals as follows. First, we merge every pair of nonconvex quadrilaterals whose union is a quadrilateral. The second step consists in forming the hexagon which results from the union of any nonconvex quadrilateral Q and a neighboring quadrilateral P. After generating a convex quadrangulation $\mathcal{Q}_{\text{loc}}$ from the hexagon, the union $Q \cup P$ of $\mathcal{Q}$ is replaced by $\mathcal{Q}_{\text{loc}}$. The quadrangulation of a hexagon which is illustrated in Fig. 12 is stated in the following result.

Theorem 3 Every hexagon (which may include reflex vertices) can be decomposed into a set of convex quadrilaterals by using at most three internal Steiner points.

Furthermore, we do not want any four-sided domain which has G^1-vertices (i.e. smooth corners). Therefore, a repairing process is used in order to ensure that some internal edges emanate from such vertices.

12 Transfinite interpolation

The generation of the diffeomorphisms on foursided domains is done with the help of transfinite interpolations. In most cases for mechanical objects, the use of Coons patches is sufficient to create a diffeomorphism from the unit square to a domain having four curved sides. For more complicated cases, we need to use Gordon patches. The description of the theoretical analyses have been done in a former paper. Let us recall briefly the fundamental definitions of those two concepts. Let α, β, γ, δ be four parametric curves that fulfill the compatibility conditions at the corners:

$$\alpha(0) = \delta(0)\,, \quad \alpha(1) = \beta(0)\,, \quad \gamma(0) = \delta(1)\,, \quad \gamma(1) = \beta(1). \qquad (20)$$

The Coons patch which corresponds to the blending functions F_0 and F_1 is defined by:

$$\begin{aligned}\mathbf{x}(u,v) \;:=\; & \begin{bmatrix} F_0(u) & F_1(u) \end{bmatrix} \begin{bmatrix} \delta(v) \\ \beta(v) \end{bmatrix} + \\ & \begin{bmatrix} \alpha(u) & \gamma(u) \end{bmatrix} \begin{bmatrix} F_0(v) \\ F_1(v) \end{bmatrix} - \\ & \begin{bmatrix} F_0(u) & F_1(u) \end{bmatrix} \begin{bmatrix} \alpha(0) & \gamma(0) \\ \alpha(1) & \gamma(1) \end{bmatrix} \begin{bmatrix} F_0(v) \\ F_1(v) \end{bmatrix}.\end{aligned} \tag{21}$$

We state the following theorems without proof. The complete theoretical proofs and missing or obscure notations can be found in our earlier works.

Theorem 4 Let $\tilde{M}$ be a constant that verifies the following upper bounds for all $i = 0, \cdots, n-1$ and $j = 0, \cdots, n$

$$\begin{aligned} n\|\phi_j(\gamma_{i+1} - \gamma_i + \alpha_i - \alpha_{i+1}) + (\alpha_{i+1} - \alpha_i)\| &\leq \tilde{M} \\ n\|\phi_j(\beta_{i+1} - \beta_i + \delta_i - \delta_{i+1}) + (\delta_{i+1} - \delta_i)\| &\leq \tilde{M}. \end{aligned} \tag{22}$$

Suppose that for all $i, j = 0, \cdots, n-1$

$$\begin{aligned} n^2 \det[\alpha_{i+1} - \alpha_i, \delta_{j+1} - \delta_j] &> 0\,, \\ n^2 \det[\alpha_{i+1} - \alpha_i, \beta_{j+1} - \beta_j] &> 0\,, \\ n^2 \det[\gamma_{i+1} - \gamma_i, \delta_{j+1} - \delta_j] &> 0\,, \\ n^2 \det[\gamma_{i+1} - \gamma_i, \beta_{j+1} - \beta_j] &> 0\,. \end{aligned} \tag{23}$$

If $\tilde{\tau} > 0$ is defined to be the minimum of the expressions in (23), then $2\tilde{M}\tilde{F} + \tilde{F}^2 < \tilde{\tau}$ is sufficient for $\mathbf{x}$ to be a diffeomorphism.

Theorem 5 Consider the assumption above and define

$$D(i,j,k,l) := n^2 \det[\mathbf{E}_{i+1,j} - \mathbf{E}_{ij}, \mathbf{E}_{k,l+1} - \mathbf{E}_{kl}], \tag{24}$$

$$\begin{aligned} C(i,j,k,l) \;:=\; & \tfrac{l}{n}\left[\tfrac{i}{n} D(i-1,j,k,l-1) + (1-\tfrac{i}{n}) D(i,j,k,l-1)\right] + \\ & (1-\tfrac{l}{n})\left[\tfrac{i}{n} D(i-1,j,k,l) + (1-\tfrac{i}{n}) D(i,j,k,l)\right]. \end{aligned} \tag{25}$$

If for all $p, q = 0, \cdots, 2n$

$$J_{pq} := \sum_{\substack{i+k=p \\ j+l=q}} C(i,j,k,l) \frac{\binom{n}{i}\binom{n}{k}}{\binom{2n}{i+k}} \frac{\binom{n}{j}\binom{n}{l}}{\binom{2n}{j+l}} > 0, \tag{26}$$

then the Coons patch is a diffeomorphism.

From those two theorems, we could deduce two algorithms for checking the regularity of a Coons map. According to our former analyses, the first algorithm executes very fast but it does not give a satisfactory result when the bounding curves α, β, γ, δ become complicated. In the opposite, the second algorithm is much more stable but it is very computationally intensive. In order to obtain stable algorithm while keeping low computational cost, we have proved the following theorem based on subdivision.

Theorem 6 Let us adopt the same notations as in the previous statement. Suppose that the Coons patch $\mathbf{x}$ defined with α, β, γ, δ is a diffeomorphism. Suppose further that we have subdivided J into σ^2 Bézier surfaces J^{ij} which are defined on

$$I^{ij} := [(i-1)/\sigma, i/\sigma] \times [(j-1)/\sigma, j/\sigma] \quad i,j = 1, \cdots, \sigma. \tag{27}$$

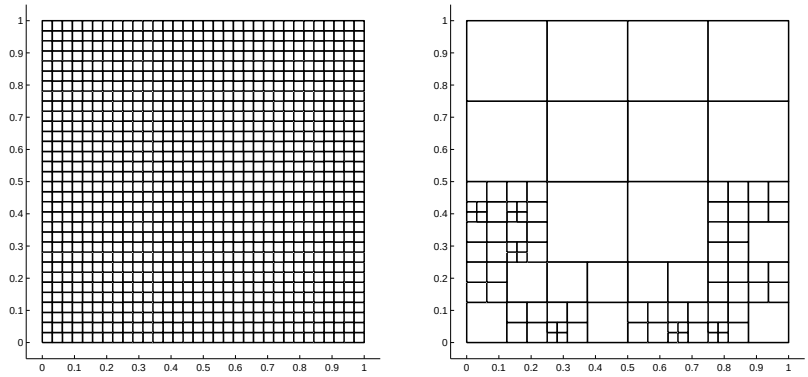
Denote by J^{ij}_{pq}, $p,q = 0, \cdots, 2n$ the control points of the Bézier surface J^{ij}. We claim that if σ is sufficiently large then J^{ij}_{pq} is of constant sign uniformly on $i,j = 1, \cdots, \sigma$ and on $p,q = 0, \cdots, 2n$.

From this theorem, one can deduce the following adaptive algorithm which is both robust and efficient in practice. In Fig. 13, we compare a uniform subdivision and an adaptive one which comes from our algorithm. It consists in subdividing the unit square into a grid which contains several rectangular domains called cells on which Bézier functions are defined.

Algorithm: Adaptive regularity

step 0 : Initialize the grid G to have only one cell $[0,1] \times [0,1]$ and compute the Bézier coefficients J_{pq} according to (26).

step 1 : Traverse the cells $I = [a,b] \times [c,d]$ of the grid G:



(a) (b)

Figure 13: Multiple subdivisions: (a)uniform (b)adaptive

- Check if all coefficients J^I_{pq} have constant sign irrespective of the indices $p, q = 0, \cdots, 2n$.
- If not, split I into four cells $I1$, $I2$, $I3$, $I4$ and subdivide the Bézier surface J^I into four Bézier surfaces J^{I1}, J^{I2}, J^{I3}, J^{I4}.
- If there was some cell $\tilde{I} \neq I$ for which $J^{\tilde{I}}_{pq}$ was always positive (resp. negative) for all $p, q = 0, \cdots, 2n$ and the current J^I_{pq} is always negative (resp. positive), then abort the whole algorithm and conclude that the Coons map is NOT a diffeomorphism.

step 2 : If in step 1, all J^I_{pq} have constant sign irrespective of the cell I and the indices $p, q = 0, 1, \cdots, 2n$ then terminate the algorithm and conclude that the Coons map is a diffeomorphism otherwise go to step 1.

Now let us turn our attention to the case of Gordon patches which are used to generate diffeomorphisms for more complicated four-sided domains. Suppose that we have two families of curves $\mathbf{f}_j$, $\mathbf{g}_i$, $j = 0, \cdots, N$, $i = 0, \cdots, M$ satisfying the *compatibility condition*:

$$\mathbf{x}_{ij} := \mathbf{g}_i(v_j) = \mathbf{f}_j(u_i) \qquad \forall (i, j) \in \{0, \cdots, M\} \times \{0, \cdots, N\}. \tag{28}$$

The automatic determination of the internal curves is used by using Floater parametrization and cubic spline interpolations.

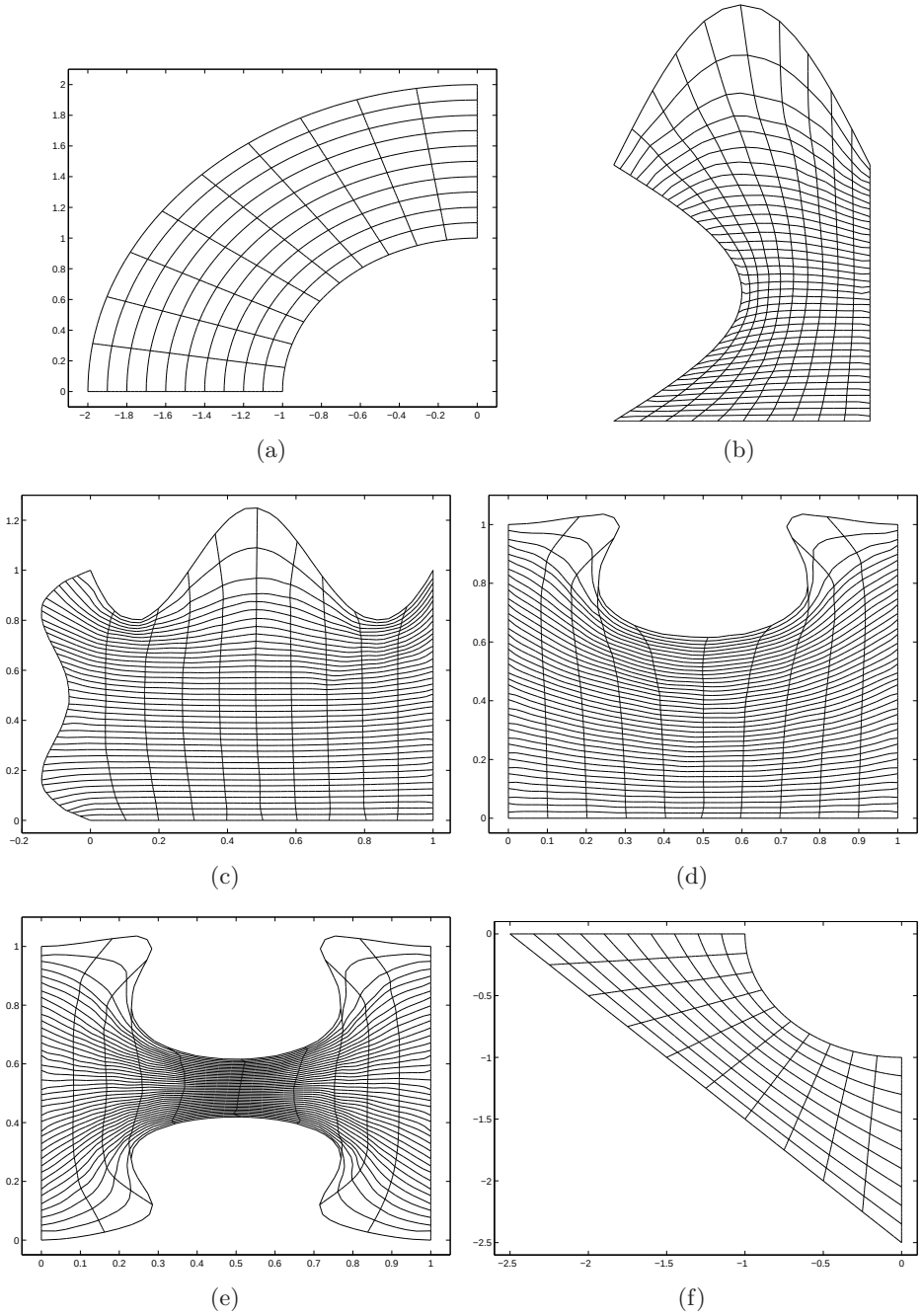


Figure 14: Mappings on four-sided patches: images of horizontal and vertical lines from the unit square.

13 Mesh generation

We would like to summarize in this section the theoretical methods for the generation of a mesh from CAD models. The initial step in generating a mesh consists in taking boundary polygonal approximations. Afterwards, one generates a mesh for each face separately as we will see below. We suppose that a single parametric function S is provided by giving a smooth parametric function $\mathbf{x}$ defined on a parameter domain $\mathcal{D} \subset \mathbf{R}^2$. The approach in triangulating S is processed in two steps. First, we generate a 2D mesh on the parameter domain $\mathcal{D}$ according to the first fundamental form. Afterwards, we lift the resulting 2D mesh to the parametric surface S by computing its image by $\mathbf{x}$. For that purpose, we start from a coarse 2D mesh of $\mathcal{D}$ and we use a generalized two dimensional Delaunay refinement which we want to sketch below. The initial coarse mesh is obtained by recursively applying the two-ear theorem which yields a triangulation having only boundary nodes.

We introduce an edge size function ρ which is defined on the parametric surface $\rho : S \longrightarrow \mathbf{R}^+$. By composing ρ with the parameterization $\mathbf{x}$ of S, we have another function $\tilde{\rho} := \rho \circ \mathbf{x}$ which we will call henceforth "parameter edge size function" because it is defined for all $\mathbf{u} = (u, v)$ in the parameter domain. Let us consider a 2D edge $[\mathbf{a}, \mathbf{b}] \subset \mathcal{D}$ and let us denote the first fundamental forms at $\mathbf{a}$ and $\mathbf{b}$ by $I_{\mathbf{a}}$ and $I_{\mathbf{b}}$ respectively. Further, we introduce the following average distance between $\mathbf{a}$ and $\mathbf{b}$

$$d_{\mathrm{Riem}}(\mathbf{a}, \mathbf{b}) := \sqrt{\overrightarrow{\mathbf{ab}}^T T \overrightarrow{\mathbf{ab}}} \qquad T := 0.5(I_{\mathbf{a}} + I_{\mathbf{b}}). \tag{29}$$

We split the 2D edge $[\mathbf{a}, \mathbf{b}]$ if this average distance exceeds the value of the parameter edge size function $\tilde{\rho}$ at the midnode of $[\mathbf{a}, \mathbf{b}]$. Note that only internal edges are allowed to be split. As a consequence, no new boundary nodes are introduced during the refinement process.

Let us consider the situation where the 2D edge $[\mathbf{a}, \mathbf{c}]$ is shared by two triangles which form a convex quadrilateral. Denote by $I_{\mathbf{a}}$, $I_{\mathbf{b}}$, $I_{\mathbf{c}}$ and $I_{\mathbf{d}}$ the values of the first fundamental form when applied to the nodes $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$ respectively. By defining T to be the average of these $I_{\mathbf{a}}$, $I_{\mathbf{b}}$, $I_{\mathbf{c}}$ and $I_{\mathbf{d}}$, we perform the generalized Delaunay edge flipping if the following generalized Delaunay angle criterion is met

$$\|\overrightarrow{\mathbf{bc}} \times \overrightarrow{\mathbf{ba}}\|(\overrightarrow{\mathbf{da}}^T T \overrightarrow{\mathbf{dc}}) < \|\overrightarrow{\mathbf{da}} \times \overrightarrow{\mathbf{dc}}\|(\overrightarrow{\mathbf{cb}}^T T \overrightarrow{\mathbf{ba}}). \tag{30}$$

That is, we replace the two triangles $[\mathbf{a}, \mathbf{c}, \mathbf{d}]$ and $[\mathbf{a}, \mathbf{b}, \mathbf{c}]$ by $[\mathbf{a}, \mathbf{b}, \mathbf{d}]$ and $[\mathbf{b}, \mathbf{c}, \mathbf{d}]$ if we have the above inequality.

For a real-valued differential function F defined on S, the Laplace-Beltrami

operator is defined by

$$\Delta_S F = -\frac{1}{\sqrt{g}} \frac{\partial}{\partial u_j} \left(\sqrt{g} g_{ij} \frac{\partial F}{\partial u_i} \right) \tag{31}$$

in which we use Einstein notation in indexing and g_{ij} are obtained from the first fundamental form. The function F is said to be harmonic if

$$\Delta_S F = 0. \tag{32}$$

In order to obtain an edge size function ρ which varies smoothly, it should be harmonic. We have therefore the following problem

$$\left\{ \begin{array}{rcll} -\Delta_S \rho & = & 0 & \text{in} \quad S \\ \rho & = & \rho_{\text{bound}} & \text{on} \quad \partial S \,. \end{array} \right. \tag{33}$$

We would like to sketch now how to numerically solve the boundary value problem (33) involving the Laplace-Beltrami equation. We approximate the function ρ by a function ρ_h by means of the finite element method. For that end, we take a temporary mesh $\mathbf{M}_h$ on S and we denote its boundary by $\partial \mathbf{M}_h$.

The values of ρ at the boundary which are denoted by ρ_{bound} are known because the piecewise linear boundary has already been determined in the polygonal boundary approximation. Since the function ρ represents the edge size, we can define its value at a boundary node A to be average of the lengths of the two incident boundary edges of A.

For a smooth function ϕ which takes value zero at the boundary we have

$$-\int_S \Delta_S \rho \phi = \int_S < \nabla_S \rho, \nabla_S \phi > =: a(\rho, \phi) \,. \tag{34}$$

Let us define the following set of approximating linear space

$$V_h := \{ f \in \mathbf{C}^0(\mathbf{M}_h) \,:\, f_{|T} \in \mathbf{P}_1 \; \forall \, T \in \mathbf{M}_h \} \,, \tag{35}$$

where $\mathbf{C}^0(\mathbf{M}_h)$ denotes the space of functions which are globally continuous on $\mathbf{M}_h$ and $\mathbf{P}_1$ the space of linear polynomials. For a function g we define the set

$$V_h^g := \{ f \in V_h : \; f = g \quad \text{on} \quad \partial \mathbf{M}_h \} \tag{36}$$

which is not in general a linear space.

The approximated solution ρ_h will reside in the set $V_h^{\rho_{\text{bound}}}$. In order to find ρ_h, we pick an arbitrary element $\tilde{\rho}$ of $V_h^{\rho_{\text{bound}}}$ and define μ_h by setting

$$\rho_h = \tilde{\rho} + \mu_h \,. \tag{37}$$

The function ρ_h is therefore completely determined if we know the new unknown function μ_h belonging to V_h^0. Observe that V_h^0 is a linear space in which we choose a basis $\{\phi_i\}_{i\in I}$ in which we have:

$$\mu_h = \sum_{i\in I} \mu_i \phi_i \,. \tag{38}$$

By introducing the following bilinear form $a_h(\cdot,\cdot)$ which approximates $a(\cdot,\cdot)$ of (34)

$$a_h(\psi,\phi) := \sum_{T\in\mathbf{M}_h} a_T(\psi,\phi) \qquad \text{with} \qquad a_T(\psi,\phi) :=< \nabla_T\psi, \nabla_T\phi >, \tag{39}$$

we have

$$a_h(\rho_h,\phi) = 0 \qquad \forall\phi \in V_h^0 \tag{40}$$

or equivalently

$$a_h(\mu_h,\phi) = -a_h(\tilde{\rho},\phi) \qquad \forall\phi \in V_h^0 \,. \tag{41}$$

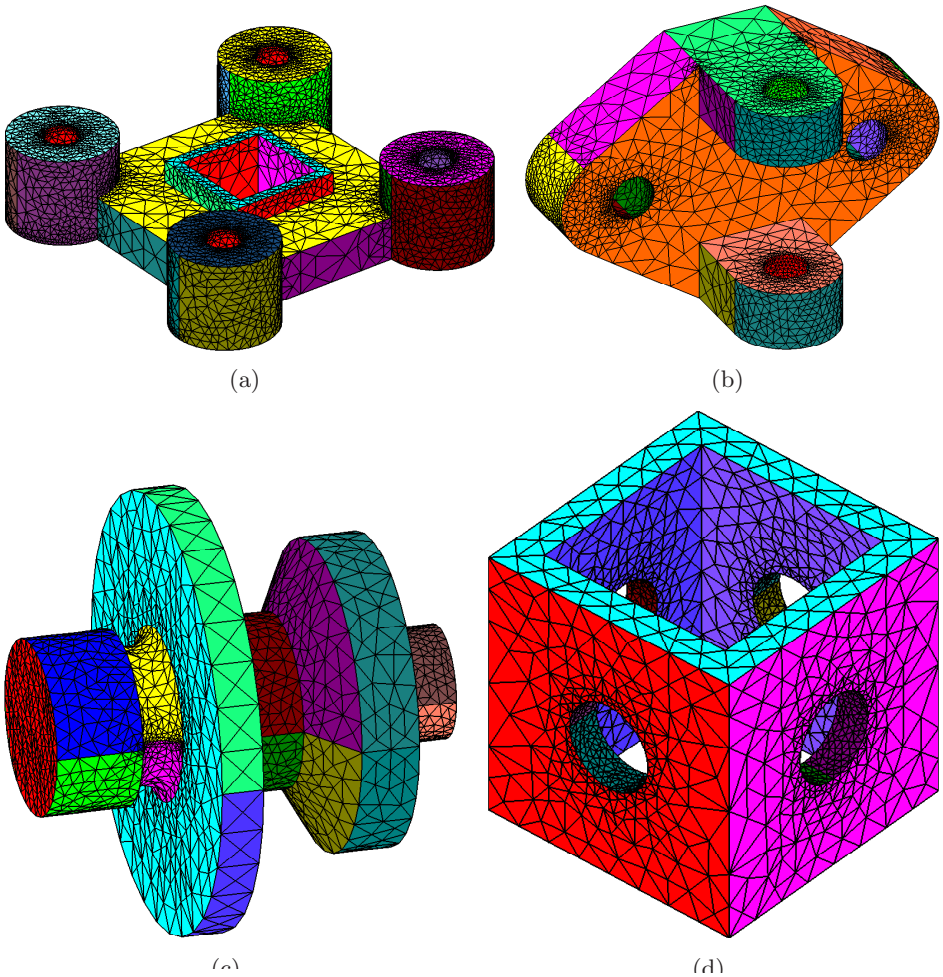
Since ϕ_i builds a basis for V_h^0, this leads to a linear equation

$$\sum_{i\in I} a_h(\phi_i,\phi_j)\mu_i = -a_h(\tilde{\rho},\phi_j) \qquad \forall j \in I \,. \tag{42}$$

One can assemble the stiffness matrix $M_{ij} := a_h(\phi_i,\phi_j)$ and solve (42) for μ_i which yields the value of μ_h by using equation (38).

For every triangle T in $\mathbf{M}_h$ with internal angles α_1, α_2 and α_3, its contribution to the stiffness matrix M is

$$M_T = 0.5 \begin{bmatrix} \cot\alpha_2 + \cot\alpha_3 & -\cot\alpha_3 & -\cot\alpha_2 \\ -\cot\alpha_3 & \cot\alpha_1 + \cot\alpha_3 & -\cot\alpha_1 \\ -\cot\alpha_2 & -\cot\alpha_1 & \cot\alpha_1 + \cot\alpha_2 \end{bmatrix} . \tag{43}$$



(a) (b)

(c) (d)

(e) (f)

Figure 15: Results from CAD data in IGES files

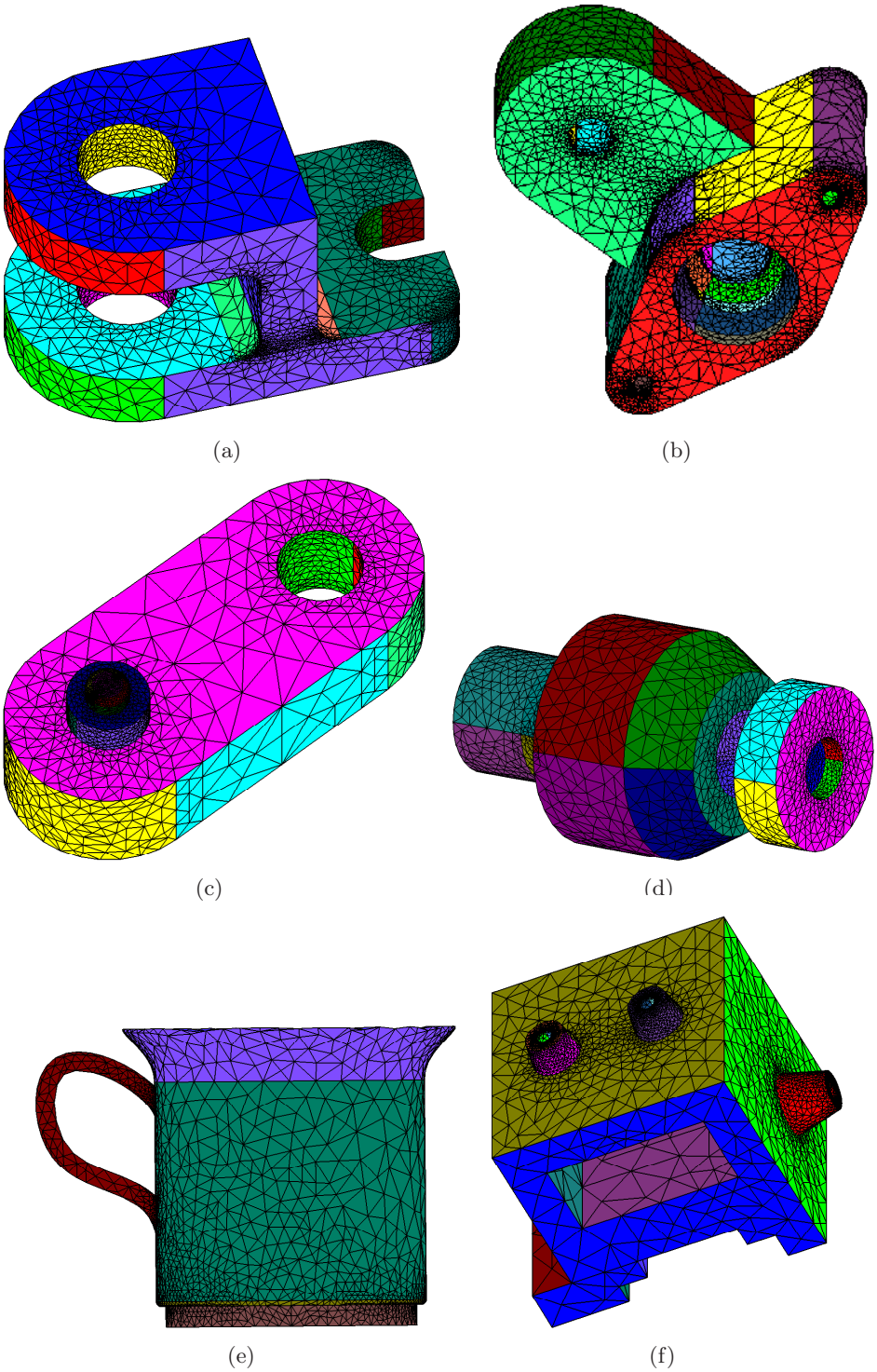


Figure 16: Results from CAD data in IGES files

References

[1] K. Atkinson, The numerical solution of integral equations of the second kind, Cambridge university press, Cambridge, 1997.

[2] D. Bremner, F. Hurtado, S. Ramaswami, V. Sacristan, Small convex quadrangulations of point sets, in: Proc. 12th international symposium, ISAAC 2001, Christchurch, New Zealand, 2001, pp. 623–635.

[3] W. Dahmen, Wavelet and multiscale methods for operator equations, Acta Numerica **6** (1997) 55–228.

[4] W. Dahmen, R. Schneider, Wavelets on manifolds I: Construction and domain decomposition, SIAM J. Math. Anal. **31**, No. 1 (1999) 184–230.

[5] W. Dahmen, A. Kunoth, K. Urban, Biorthogonal spline wavelets on the interval: stability and moment conditions, Appl. Comput. Harmon. Anal. **6**, No. 2 (1999) 132–196.

[6] A. Forrest, On Coons and other methods for the representation of curved surfaces, Comput. Graph. Img. Process. **1** (1972) 341–359.

[7] W. Gordon, C. Hall, Construction of curvilinear co-ordinate systems and applications to mesh generation, Int. J. Numer. Methods Eng. **7** (1973) 461–477.

[8] W. Gordon, C. Hall, Transfinite element methods: blending-function interpolation over arbitrary curved element domains, Numer. Math. **21** (1973) 109–129.

[9] W. Hackbusch, Integral equations: theory and numerical treatment, International series of numerical mathematics 120, Basel: Birkhaüser, 1995.

[10] H. Harbrecht, R. Schneider, Biorthogonal wavelet bases for the boundary element method, Preprint SFB393/03-10, Technische Universität Chemnitz, Sonderforschungsbereich 393, 2003.

[11] H. Harbrecht, R. Schneider, Wavelets for the fast solution of boundary integral equations, Preprint SFB393/02-19, Technische Universität Chemnitz, Sonderforschungsbereich 393, 2002.

[12] F. Hill, Computer graphics using OpenGL, Prentice Hall, London, 2001.

[13] G. Shepherd, S. Wingo, MFC internals, Addison-Wesley Developers Press, Reading, 1997.

[14] M. Kilgard, The OpenGL Utility Toolkit (GLUT), programming interface API version 3, Silicon Graphics Inc., 1994.

[15] L. Piegl, A menagerie of rational B-spline circles, IEEE Computer Graphics and Applications **9** (Sept. 1989) 48–56.

[16] L. Piegl, Infinite control points - a method for representing surfaces of revolution using boundary data, IEEE Computer Graphics and Applications **7** (March 1987) 45–55.

[17] M. Randrianarivony, G. Brunnett, Sufficient and necessary conditions for the regularity of planar Coons map, Sonderforschungsbereich 393, Preprint SFB393/04-07, 2004.

[18] M. Randrianarivony, Geometric processing of CAD data and meshes as input of integral equation solvers, PhD. thesis, TU Chemnitz.

[19] R. Schneider, Multiskalen- und Wavelet-Matrixkompression: Analysisbasierte Methoden zur Lösung grosser vollbesetzter Gleichungssysteme, Teubner, Stuttgart, 1998.

[20] U. S. Product Data Association, Initial Graphics Exchange Specification. IGES 5.3, Trident Research Center, SC, 1996.