Computation of Components' Interfaces in Highly Complex Assemblies

François Jourdes, Georges-Pierre Bonneau, Stefanie Hahmann, Jean-Claude Léon, François Faure

Computation of components' interfaces in highly complex assemblies

François Jourdes, Georges-Pierre Bonneau, Stefanie Hahmann, Jean-Claude Léon, François Faure [a]

[a] *University of Grenoble, INRIA and Laboratoire Jean Kuntzmann*

Abstract

The preparation of CAD models from complex assemblies for simulation purposes is a very time-consuming and tedious process, since many tasks such as meshing and idealization are still completed manually. Herein, the detection and extraction of geometric interfaces between components of the assembly is of central importance not only for the simulation objectives but also for all necessary shape transformations such as idealizations or detail removals. It is a repetitive task in particular when complex assemblies have to be dealt with. This paper proposes a method to rapidly and fully automatically generate a precise geometric description of interfaces in generic B-Rep CAD models. The approach combines an efficient GPU ray-casting technique commonly used in computer graphics with a graph-based curve extraction algorithm. Not only is it able to detect a large number of interfaces efficiently, but it also provides an accurate Nurbs geometry of the interfaces, that can be stored in a plain STEP file [1] for further downstream treatment. We demonstrate our approach on examples from aeronautics and automotive industry.

Key words: Interface, Imprint, Assembly, CAD/CAM

1. *Introduction*

The widespread use of CAD software in industries over decades has led to common representations of products as assemblies, also designated as digital mock ups (DMUs). The extensive use of CAD has led to increasingly complex DMUs, ranging from tens of components for simple products to thousands of components for a car to hundreds of thousands of components for an aircraft. More recently, commercial software platforms have expanded to cover the product lifecycle, addressing, among various steps, the design stage and the numerical behavior simulations using CAE software. While finite element (FE) analyses of single components have been under focus for long and are still requiring new developments [20], companies are now heading toward behavior simulations at the assembly level [18]. Indeed, the design of complex products containing numerous components remains tedious since many tasks require a large amount of user interaction that can be only conducted by experts. First, generating FE models at rather large assembly level, i.e. incorporating tens of components, requires the simplification and/or idealization of component shapes, consistently with the simulation objectives, e.g. replacing volume sub domains by either planar surfaces or lines when their mechanical behavior resembles plate's or beam's one, respectively. In addition to the modeling of each component individually, mechanical simulation requires to model the interfaces between components [4,5,18]. While the kinematic simulation of assemblies only requires knowledge about the type and spatial position of the contact surfaces, stress computations using FE simulations require the precise modeling of the imprint of one component on each other within each interface. Given two partially overlapping surfaces, imprinting consists in the process of inserting curves and vertices in each surface such that they share a topologically well defined interface.

While the automatic meshing of components based on their B-Rep reprensentations is available in CAE sofware, remeshing components to conform to their interfaces may involve man-months of engineering. Similar observations can be transposed to other stages of the product lifecycle like assembly simulations using virtual reality techniques [9,23] where component interfaces appear again as prominent elements. More generally, components' geometric interfaces appear as key entities to process assemblies efficiently. However, these interfaces are not readily available in DMUs and specific geometry processing is needed to obtain them [4,5,23,9].

In this paper we propose a method to rapidly detect and produce the B-Rep model of the imprint of each component onto each of its neighbouring components as illustrated in Figure 1. We do not assume any prior knowledge on the interfaces, we are given only the CAD BRep model of the assembly, and from this we detect and reconstruct the BRep

of the interfaces.

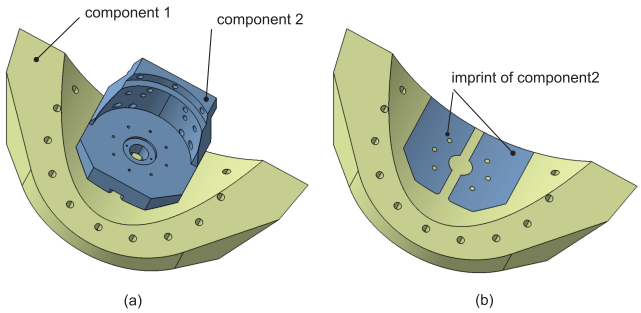


Fig. 1. Neighbouring components and their imprint (courtesy ANTECIM).(a) component1 and component2 in their assembly position, (b) The imprint of component1 onto component2 representing the precise geometric interface defined as the common cylindrical surface between these two components.

Based on our study of the current state-of-the-art regarding how interfaces are dealt with in CAD/CAM academic literature and industrial software (see section 2 for details), we can make the following observations. First, most of the previous work does not incorporate a precise description of components' geometric interfaces. Furthermore, no standard is effectively implemented in industrial CAD/CAM software that can export geometric interfaces to other ones. When it comes to large assemblies, this lack of detailed description of geometric interfaces becomes a real challenge. Low level interactive operators enabling the surface trimming of components' boundaries become too tedious for a practical computation of imprint faces [4,5,2]. To scale up this approach to highly complex assemblies, new and more automated approaches are necessary. Highly complex assemblies are subject to multiple downstream processes with their own objectives in focus and their own tools. In such cases, it is essential to have the ability to handle standard CAD/CAM models and still be able to use the precise description of geometric interfaces between assembly components. We also observe that high level geometrical operators that are used to generate geometric interfaces, such as geometric offset computation, imprint generation [5,2] and CSG-type operators [4], do not scale well to complex assemblies in terms of time and robustness.

Based on these observations, we propose an approach that takes as input any standard CAD/CAM assembly model and automates the generation of geometric interfaces between components. Our approach can detect and accurately describe the geometry of interfaces while scaling up well to highly complex assemblies. Our main contributions are:

- A novel method to compute interface imprints in B-Rep models, using ray-casting and a series of post-processing steps, to produce upgraded B-Rep models which include the imprints of the interfaces in the components.
- The efficient implementation of the most compute-intensive parts of the method on the Graphics Processing Unit (GPU), allowing the process of thousands of B-Rep faces within minutes.

We have applied our framework to complex real-world cases and the results were validated by our industrial partners. The remainder of this paper is structured as follows. Section 2 analyzes related works. Section 3 presents our method. In section 4, the implementation on GPU is described. Results on real-world industrial CAD/CAM assemblies are presented in section 5. We finally conclude and sketch future work in section 6.

2. *Background and Motivation*

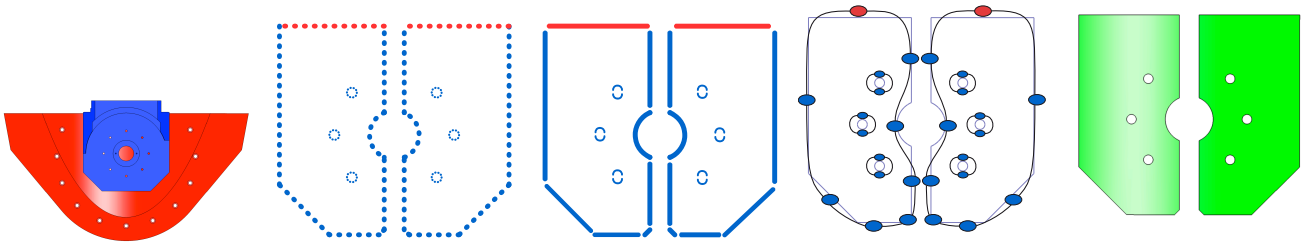
Let us first define vocabulary. The *components* of an assembly, as modeled within current CAD systems, are facetted solid models represented as B-Rep NURBS. An *interface* in CAD modeling is defined as the imprint of one component onto the other. It is a contact surface which may be composed of planar, cylindrical, conical, spherical, ruled surface or general free-form surface parts.

Approaches to functional design [16,15,10,14] contribute to the setting of relationships between functional parameters of a product and its 3D CAD model. However, generating these relationships requires designers' interactions to locate mating surfaces between components, which becomes too tedious for highly complex assemblies [12]. Additionally, even if B-Rep faces of components are identified as functional surfaces (i.e. component interfaces), the common surface, or imprint, of each component onto the other is not available.

Interfaces have typically been modeled using geometric constraints, such as mating surfaces or coaxiality, in functional design [7], virtual assembly simulations [23] or in standard CAD modules. While sufficient to enforce the relative positioning of the components, these constraints neither define the imprints between these (see Figure 1), nor do they necessarily identify all the contact surfaces. Indeed, when an engineer selects two surfaces as input of a mating constraint, e.g. S_{11} and S_{21} in Figure 3, the other contacts implementing the kinematical interface, such as S_{11} and S_{22} in the same figure, are not identified as contacts. The same issue arises in coaxiality constraints, creating ambiguous configurations that require specific treatments [23].

Although feature-based modeling can provide component boundary decompositions that can meet the requirements to correctly identify the mating surfaces [17,21], it is a prescriptive approach because it requires user input. Therefore, the user's assignment of features becomes error prone, especially when processing large assemblies. Also, it has to be pointed out that features cannot be propagated through model exchange between industrial software because most of them rely on STEP API 203 or 214 [1] that do not incorporate feature description. When processing assemblies for FE simulations in an industrial context, model exchange between CAD and CAE is often necessary, hence reducing the interest of feature-based models.

Some CAD software propose operators to compute geometric interfaces. In CATIA V5, this operation is applied to



(a) Two components (b) Ray-surface intersections (c) Boundary curves (d) Curve connectivity (e) Imprint faces

Fig. 2. Overview of the algorithm.

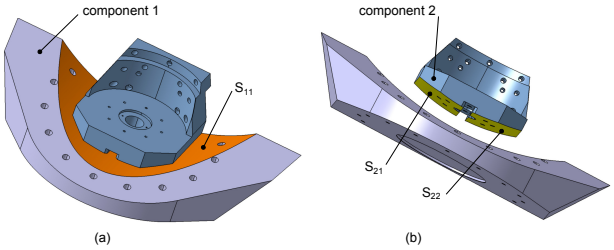


Fig. 3. Geometric constraints do not define the imprint between two components (courtesy ANTECIM).(a) component1 and component2 positioned using a mating constraint defined using S_{11} on component1 and S_{21} on component2 only, (b) exploded view of component1 and component2 for visualization purposes showing that S_{21} and S_{22} coincide with the same cylindrical surface. The imprint derived from the contact between component1 and component2 must relate to the couple of surfaces (S_{11}, S_{21}) and (S_{11}, S_{22}).

a facetted representation of the B-Rep NURBS model of the components. As a result, the approximation taking place is not defining precisely the geometric interface and this operator cannot robustly distinguish contact from clearance configurations. In SolidWorks, geometric interfaces defining contacts can be visualized but cannot be converted to imprints. To the best of our knowledge, there is no industrial CAD software that produces accurate geometric interfaces with operators that scale up robustly to process highly complex assemblies. Additionally, if these softwares can position components using geometric constraints, the consistency of the corresponding equation system is still an issue [11]. This is critical since positioning components, even in a rather small assembly like the hydraulic pump of Figure 13 with 46 components, can lead to a large number of constraints, i.e. more than two hundreds of geometric constraints are necessary for the hydraulic pump. Therefore, aircraft and automotive companies favor the storage of large assemblies with components located at their absolute positions. Consequently, there is no geometric relationship between assembly components.

Some approaches have been proposed to identify geometric interfaces between components. Chouadria [4] used a facetted representation of components. Because the facetted representation is the input model, telling contacts from clearances becomes inherently difficult since there is no reference to an underlying B-Rep NURBS geometry. In [5], the authors introduce an approach of tolerant imprinting using a tolerance greater than that of the underlying geometric modeler used to split edges and faces of each component. All the boundary transformations are performed using tolerant projection and split/cut operators of a volume modeler. This results in difficulties to scale up to highly complex assemblies. As mentioned by the authors, their approach strongly relies on the behavior of the operators of the volume modeler, which is not generic.

Contact has also been extensively studied in Computer Graphics, to detect and react to collisions [19]. It is typically modeled as distance constraints within pairs of low-level geometric primitives such as vertices, edges, polygons, spheres, and boxes. Recent methods leverage the power GPUs to discretize the boundaries of the intersection volumes between arbitrarily complex polyhedra [8,22]. None of these approches accurately compute the contact surface between tangent objects.

The above analysis shows that current approaches in functional design, feature-based modeling, geometric constraints and industrial practices are not able to produce the precise geometric interfaces between assembly components in a robust and scalable manner able to process highly complex assemblies.

3. *Imprint computation method*

Our approach is based on the following assumptions:

(A1) The imprint is a 2-manifold surface with boundaries.
(A2) Every point on a boundary of an interface belongs to the boundary curve of at least one component.

Assumption (A2) is reasonable, since interfaces that include neither of the components boundaries only occur in some degenerate cases of point contact, line-contact or free-form shapes, but are rather rare in assembly modelling. We defer the study of a completely general method to future work.

An overview of our method is shown in Figure 2. Given two components (fig.2a), we shoot short rays from the facet boundaries and detect detect nearby intersections (fig.2b). These are clustered to represent the boundary curves of the imprint (fig.2c). We compute the connectivity of the curves (fig.2d) to detect cycles, which define the imprint faces (fig.2e).

3.1. *Interface boundary points*

We shoot short rays only from the discretized boundaries of the surfaces to compute the boundaries of the imprint, consistently with assumption (A2). The rays are created on a per-face basis and casted in the direction of the normal to the surface. The *rays* are generated for all components B-Rep faces along their boundary curves by sampling the B-spline curves uniformly at the precision of the CAD model. Thus, from a boundary shared by two B-Rep faces of the same component, rays are casted in two directions.

We compute pairs of closed points (P, Q) where P as the origin of the ray belongs to a boundary curve of some component's face, and Q belongs either to the interior of another component's face, or to its boundary curve. The latter corresponds to the frequent case where two faces in different components share a section of the same boundary curve, see for example the three plates in Figure 9(a).

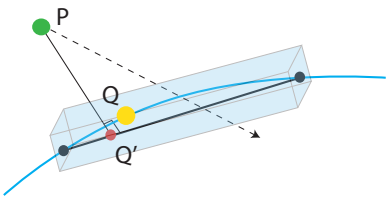


Fig. 4. Post-processing proximities with boundary curves. The ray (dashed arrow) casted from its origin P (green point) enters the box of an edge (gray line) discretizing a piece of a boundary curve (blue curve). The closest point Q' (red) to P on the segment is computed. The position of Q' in the edge is used to linearly interpolate the parameters of the two end points (black) on the boundary curve. This gives us the position and parameter of the point Q on the curve segment close to P.

For the sake of clarity, the technical details of the GPU implementation of ray casting is presented in section 4. For the time beeing it is sufficient to know that the GPU provides pairs of points (P, Q') where P is the origin of the ray and Q' is a point, close to P, lying on a discretization of the components. This discretization consists in edges for the boundary curves, and triangles for the B-Rep faces.

- In case Q' belongs to an edge, we need to compute the corresponding point Q on the boundary curve segment. To this end we linearly interpolate the parameter values of the end-points of the edge using the position of Q' in the edge, as illustrated in Figure 4. Since the parameterization may be distant from chord-length, we check that the distance between P and Q is smaller than a threshold ε. If not we reject the pair (P, Q) and proceed to the next ray. If yes, we store the position Q, its parameter value and the index of the boundary curve it belongs to.
- In case Q' belongs to the interior of a triangle, we don't need the exact intersection point with the B-Rep face the triangle belongs to, we only need to store the index of this B-Rep face.

Figures 12(d) and 13(b) show the result of the ray-casting process on complex assemblies. Figure 5 shows a simpler example, corresponding to the assembly of Figure 9.

Output of the ray-casting part is finally a set of points lying on the boundary of the interfaces. In Sections 3.3 and 3.4 we will present our method for reconstructing the geometry of all interfaces from this set of points.

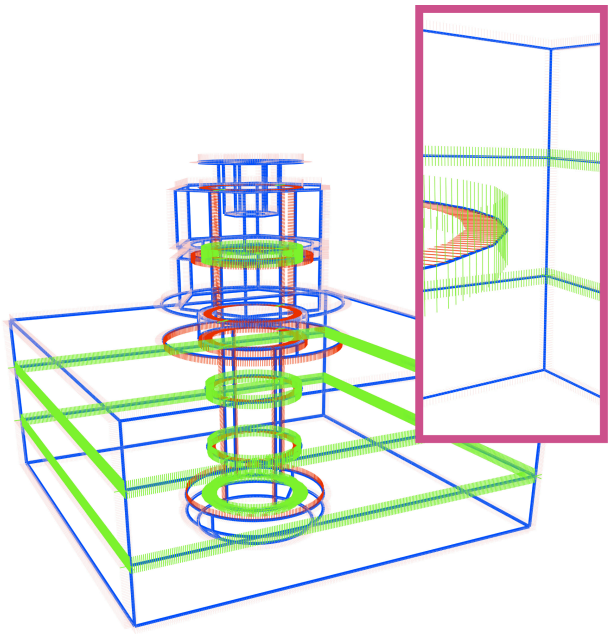
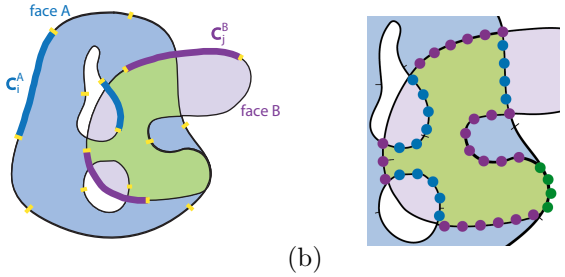


Fig. 5. Ray-casting results. All rays are casted from the boundary curves of the components. The rays are subsampled and magnified for visibility purposes. Rays appear pink if they do not correspond to component proximities, red if they correspond to proximities between a boundary curve in one component and the interior of a B-Rep face in another component, green if they correspond to proximities between two boundary curves in different components. The inset shows a zoom on the cylindrical bolt and the cylindrical hole in the plates.

3.2. *Notations*



(a) (b)

Fig. 6. (a) Interface I (in green) between B-Rep face A (in blue) and B-Rep face B (in purple). A (resp. B) is bounded by the curves C_i^A (resp. C_j^B), two of them are highlighted in dark blue (resp. dark purple). (b) Points surrounding the interface, collected by ray casting. The points P_k^A collected from face A are shown in dark blue, the points P_l^B casted from face B are shown in dark purple. The green points belong to the boundary of both faces.

In order to describe our method for reconstructing the interfaces from the points collected at their boundary, we choose the following notations.

Let I be an interface between B-Rep face A of one component and B-Rep face B of another component. Let C_i^A, C_j^B denote the boundary curves of the faces A and B as given from the input B-rep models. The B-rep model also provides for each face an ordering of these curves and a parametric representation with coherent orientation for each curve such that the curves $\{C_i^A\}_i$ form one or more closed oriented boundary curves. See for example the blue face in Fig. 6(a) which is composed of 11 curves C_i^A.

Using ray casting as explained in the previous section, we have collected points along the boundary of I. These points correspond to rays that were launched from the boundaries of face A (resp. B) and that have intersected face B (resp. A) either at its boundary or in its interior. Let us denote these points P_k^A in case the ray has been launched from A, and P_l^B otherwise. The collected points belong either to the boundary of one face (see blue and purple points in Fig. 6(b)) or to both faces (see green points in Fig. 6(b)).

For each P_k^A, the ray-casting process described in the previous section gives us the following information:
· $P_k^A \longleftarrow (i, s)$ if the ray intersects the interior of B, or
· $P_k^A \longleftarrow (i, s, j, t)$ if the ray intersects the boundary of B,
where i is the index of the boundary curve P_k^A belongs to (e.g. $P_k^A \in C_i^A$), s is the parameter of P_k^A in C_i^A, and t is the parameter of the closest point of P_k^A in the boundary curve C_j^B of face B.
Similarly, for each P_l^B we know:
• $P_l^B \longleftarrow (j, t)$ if the ray intersects the interior of A, or
• $P_l^B \longleftarrow (j, t, i, s)$ if the ray intersects the boundary of A,
where j is the index of the boundary curve P_l^B belongs to (e.g. $P_l^B \in C_j^B$), s is the parameter of P_l^B in C_j^B, and s is the parameter of the closest point of P_l^B in the boundary curve C_i^A of face A.

3.3. *Grouping interface boundary points*

The first step of our interface reconstruction method consists of grouping together and ordering the points P_k^A of face A so that each group describes a section of curve. Therefore, for each curve C_i^A with a non-empty set of points associated to it, the points P_k^A are sorted by increasing value of their parameter s. Then, whenever the distance between two successive points is larger than a threshold ε', a new group is created. This may happen when two or more seperate sections of the same curve C_i^A belong to the interface boundary, e.g. see the two left-most boundary curves of face B in Fig. 6(a).

Theoretically, a new group should be created as soon as one ray didn't intersect the other face. Nevertheless, in order to account for numerical inacurracies due to the initial sampling of rays along the boundary curves, we use a distance threshold ε' larger than the distance given by the initial sampling. In practice, we choose ε' to be 5 times larger than the distance in the initial sampling.

The same procedure is applied to group adjacent points P_l^B in face B. Figures 2(c), 7(c) and 8(c) show the group of points on three examples.

The min and max parameter values of points in each group determine a section of a face boundary curve, which also bounds the interface. Thus each group of adjacent points may also be considered as a section of the interface boundary curve. But vice-versa, the interface boundary is composed only of a subset of the previously computed groups of points. There may be redundancy due to the fact that groups of points are partly overlapping when the points belong to the boundary of both faces. See green points in Fig. 6(b) or the circular hole in Fig. 8(c). These points generate in fact two groups, i.e. two curve sections for the same piece of boundary, one for face A and another for face B.

3.4. *Connecting interface boundaries*

First, redundancy is reduced by making the following observation. When both faces share a common section of boundary curve, it may be the case that one group of points in face A is completely included in another group of points in B. In that case we remove it, since the other group of points already includes this piece of interface boundary. In order to select among all groups of points complete subset of groups of points representing all boundaries of the same interface, we introduce a graph-based approach.

A directed graph is defined as follows. All nodes in this graph are the groups of adjacent points. Two types of arcs are established in this graph:
- Arcs between nodes in the same face. When two nodes correspond to adjacent sections of boundary curves of the same face, then an arc is established between the nodes, in the direction corresponding to the orientation of the curves. This information can be recovered from the B-Rep model of the face.
- Arcs between nodes in different faces. Let P_k^A be the point with max parameter in a node N in face A. If P_k^A corresponds to a ray intersecting the boundary of B, then we know (i, s, j, t) where t is the parameter of the closest point of P_k^A in C_j^B. We then search for the unique node $\tilde{N}$ in face B, for which t lies between the min and the max parameter of this node. An arc is established between N and $\tilde{N}$.

An example of such a graph is shown in Figure 7(d). It corresponds to the two components shown in Figure 7(a,b). There are 12 nodes in this example. The nodes are colored according to the component from which the rays were casted to detect the intersection points. 4 nodes belong to the lower component in Figure 7, and 8 nodes to

the upper component. 8 arcs join nodes in different faces, and 4 arcs join nodes in the same face. Note that the two red nodes at the bottom correspond to different sections of the same boundary curve of the red component. In this example there is no redundancy, two nodes corresponds to sections of boundary curves that intersect in a single point. As a result the graph is a cycle.

In general, the graph has at least one strongly connected component [1] corresponding to the outer boundary of the interface. Furthermore, additional strongly connected components appear for each hole (interior boundary) in the interface. Since a section of the interface's boundary curve may belong to both faces, the connected components are not always cycles, as shown in Figure 8(d). In all cases, a cycle can be extracted from any strongly connected component, since strongly connected directed graphs are cyclic. Once the graph is built, each interface boundary can be extracted as a cycle. We extract a cycle from each strongly connected component using the algorithm 1. In this algorithm, priority is given to a successor node belonging to the same face as the first node. As a result, for the circular hole in the example of Figure 8 , this algorithm extracts cycles with all nodes belonging to the same face, e.g. two red of two yellow nodes.

Algorithm 1 Extract Cycle from face A (resp. B)

Choose any node $StartingNode$ in face A (resp. B)
$CurrentNode \leftarrow StartingNode$
repeat
  if there exists a direct successor N of $CurrentNode$ in face A (resp. B) **then**
    $CurrentNode \leftarrow N$
  else
    Choose any direct successor $\tilde{N}$ of $CurrentNode$
    $CurrentNode \leftarrow \tilde{N}$
  end if
until $CurrentNode = StartingNode$

3.5. *B-rep model of interface*

The interface boundaries are now extracted from the graph as cycles. Each node represents a curve section parameterized either in the B-Rep face A or the B-Rep face B. The last step of our reconstruction method consists in computing a B-rep model of the interface surface. It is in fact a trimmed surface of both A and B. The interface can thus be parameterized either in the parameter domain of face A or B. If we choose the parameter domain of A, then each section of the boundary interface in B must be reparameterized in the parameter domain of A. This reparameterization is implemented using the OpenCascade library.

[1] In graph theory, a directed graph is said to be strongly connected, iff for every pair of nodes u, v, a directed path exists from u to v and from v to u [6].

In our implementation we choose to parameterize the interface using the parameter domain of the face which supports the largest number of nodes in the graph, in order to reduce the number of reparameterizations. For each node in the cycle, we store the parameterization directly in the B-Rep model if the corresponding curve section is defined in the chosen parameter domain, otherwise we reparameterize if it is defined in the parameter domain of the other face. Using this procedure, the outer boundary of the interface as well as all inner boundaries are entirely encoded in the B-Rep model, with the required orientation. If necessary, this model can be stored in a plain STEP file for downstream treatment in other systems.

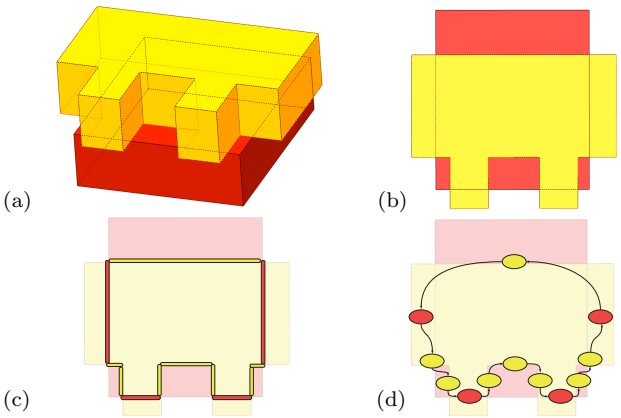


Fig. 7. (a) and (b): two components. (c): group of adjacent points (see section 3.3). (d): corresponding graph (see section efsubsec:graph).

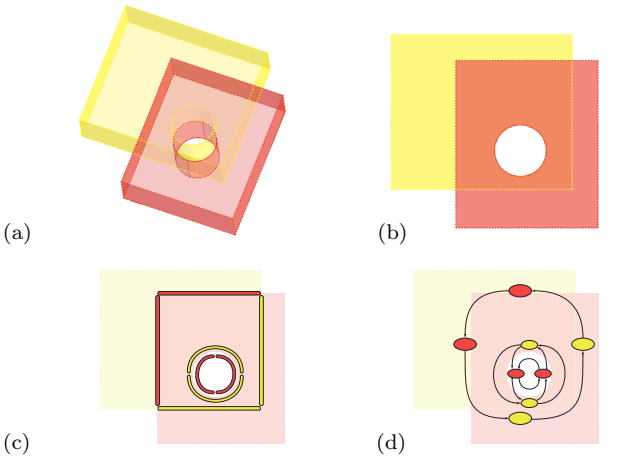


Fig. 8. (a) and (b): two components sharing the same circular hole at their interface, but with a different parameterization. (c): group of adjacent points (see section 3.3). (d): corresponding graph (see section 3.4). Due to redundancy, the strongly connected component corresponding to the circular hole is not a cycle.

4. *Implementation*

GPU Ray casting

The core of our system consists in launching rays through the object and looking for pairs of neighbouring intersec-

tions between one ray and two components. Such a pair indicates the existence of an interface between the two components. We use OptiX, a highly efficient, freely available ray-tracing engine implemented on the GPU [13]. It is a programmable system for efficiently implement ray-casting and ray-tracing applications on GPU. The user sends a set of geometric primitives to the GPU. The system initializes an acceleration structure stored in GPU, based on bounding-volume hierarchies. This structure enables to quickly find potential intersections between rays and geometric primitives. The user provides a set of programs, for computing the bounding-boxes of the geometric primitives, actually testing and computing intersections with the geometric primitives, and for handling intersections, all these on a per-ray or per-primitive basis. All the parallel aspects of the implementation are addressed by the library, transparently to the user.

We discretize the input model into edges for the boundary curves and triangles for the B-Rep faces, and we generate axis-aligned bounding boxes for these geometric primitives. A ray passing nearby a component's boundary curve or B-Rep face will always intersect one of the boxes. Note that the discretization is only used for accelerating the casting of rays by OptiX. However, for all other computations (interface boundary computations and interface surface generation) the exact Nurbs representation is used.

We proceed in two steps: first, OptiX finds all potential proximities between rays and components based on bounding-boxes of the geometric primitives. Then, the potential proximities are post-processed, still in the GPU, in order to remove invalid proximities, and to identify the parameters of the pair of closed points in the B-Rep model, as explained in section 3.1.

Model preparation

The input of our algorithm are one or several STEP files containing the components of the assembly as a B-Rep Nurbs. In a STEP file, each surface forming the boundary faces of the components is represented as a (possibly trimmed) parametric B-spline surface defined on a given parameter domain, and each boundary curve has its individual parameterization. The OpenCasCade (OCC) library [3] is used to read the STEP files and to compute a mesh of each component in the assembly. The edges and triangles in these meshes are used by the OptiX library, as explained above. The triangles do not need to be regular, the only requirement is that the maximum distance between the triangulation and the face should be less than a given error tolerance ε, so that an intersection between the ray and a triangle indicates a nearby intersection between the ray and a B-Rep face.

5. *Results*

We have implemented our interface computation algorithm on a linux workstation with 4 Intel Xeon 2.67 Ghz CPUs, 8 GB RAM and a NVIDIA Quadro 4000 graphics card with 2 GB memory.

Model description

Various models are presented, two of them are handmade academic models, the other four are real-world industrial assemblies. The models exhibit planar and cylindrical interfaces (Figures 1, 7, 12 and 13(a,b,c)). as well as freeform interfaces (Figure 10 and 13(d,e,f)). The first industrial example is an aircraft assembly, part of a root joint, for joining the root of the wing to the fuselage, see Figure 12. Additionally to the simple academic models, where the interfaces are known a-priori, this aircraft assembly served us to validate our method on a real industrial part with feedback from an industrial partner. The hydraulic pump assembly, Figure 13(a), has 46 components composed of 2156 B-Rep faces and 5389 NURBS curves. 565 interfaces have been computed. They are shown in Figure Figure 13(c). In the Imprint example (Figs. 1, 3), the holes are modeled by substraction of cylinders from objects with non-planar faces. As a result, the cylindric holes are represented in the input B-Rep model, not as usual with only two half cylinders, but with as many as 50 cylinder arcs connected together, which explains the large number of B-Rep faces and curves in this model. Nevertheless, our method is able to handle the assembly and produces topologically valid B-Rep STEP files for the interfaces.

Timings

For the aircraft part (Fig. 12), the total processing time is 12s. According to our industrial partners, building an FEM model for such an assembly is a question of days, of which several hours of manual work are required to extract the interfaces between components. The total processing time for computing the interfaces is at most 106s for the hydraulic pump (Fig. 13). Thanks to the parallel GPU implementation (see section 4), a huge number of rays can be casted very efficiently. More precisely the time to cast rays is logarithmic in the number of rays. For the Aircraft root joint example, 10 million rays are casted in just 2,3 seconds. Setting up the graph (see section 3.4) and building the step model of the interfaces (see section 3.5) takes at most 1s for the Aircraft example. Most of the time is spent by the OptiX library for initializing the acceleration structure before casting the rays (see section 4). This initialization time of OptiX is proportional to the number of triangles used to discretize the model, therefore it is directly related to the complexity of the geometry of the components. For the aircraft root joint and bolted junction assemblies which have a lot of planar faces and some cylindrical faces, this initialization time takes 60% to 70% of the total time. For the Imprint, Freeform, Car door and Pump examples, many triangles are required for the discretization, and as a consequence the time to initialize OptiX account for about 95% to 99% of the total time.

Numerical issues

- We choose the length of rays coherently with the CAD

model tolerances. If the rays are too short, then some interfaces may not be detected by the ray-casting. In Figure 11(b) we show the result of ray casting on the Imprint model with rays that are too short. Interface boundary points are not detected when the distance between the components is larger than the length of rays. This behavior may also be used to detect defaults in assemblies, as illustrated in Figure 13(e). Two parts seem to be missing in the interface between the car door outer surface and the inner frame. In fact, at these two places the components have a distance larger than the length of rays. It is up to the user to decide whether or not this corresponds to missplaced components in the assembly. Thanks to the GPU implementation, the user can quickly experiment with different length of rays if the default length is not adapted to the model. In the examples of the paper, we only had to increase slightly the length of rays for the Imprint example.

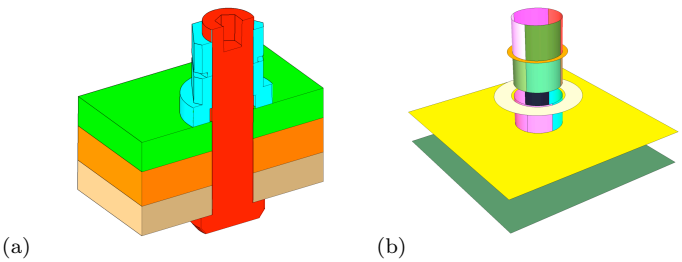
- We also did initial experiments to adapt our method for computing interferences between components. In order to do this we translated the origin of the rays. Instead of choosing the origin of rays exactly on the boundary curves of each face of the components, we translated this origin in the direction opposite to the normal of the face, thus slightly inside the component. Therefore the rays also intersect other components with which there is an interference. On the other hand, by increasing the length of rays, we are able to detect nearby components even if they are not in contact, thereby paving the way for clearance fit detection and computation. We reserve a full study of clearance and interference computation in assemblies for future works.
- Other parameters of the algorithm include the chordal tesselation error that is used by OCC to triangulate the components for accelerating OptiX (see section 4), the threshold ε used to assess boundary/boundary proximities (see section 3.1) and the threshold ε' used to build the group of interface boundary points (see section 3.3). We used the same value of these parameters in all our results. For the chordal tesselation error, we used `deflection`= 0.1 in the OCC function `BRepMesh::Mesh`. This is a relative measure taking into account the absolute maximum distance between the object and its tesselation and the lengths of the edge/triangle used to tesselate it. Thus for the same `deflection` parameter, applying the function `BRepMesh::Mesh` on two homothetic surfaces will result in the same number of triangles and edges. For the threshold ε we used the length of the ray. The value of ε' has already been given in section 3.3. This value assumes that an edge of the interface cannot be shorter than 5 times the distance between the origin of two consecutive rays.

6. *Conclusion*

Interfaces detection and computation between components in B-Rep CAD models is a key step for the efficient analysis of assemblies. Interfaces are essential for component idealizations, FE simulations or Virtual reality simulation of assemblies. In this paper we have presented the first ray-casting based interface computation method that scales up robustly to process highly complex assemblies. Our method combines GPU accelerated adaptive ray-casting for proximity computations with a graph-based interface geometry reconstruction. The algorithm is extremely fast without requiring any user-interaction and is thus able to treat complex assemblies in minutes only. The resulting interfaces are standard high-level B-Rep Nurbs faces computed at an accurate precision, and can be stored in plain STEP files for further processing.

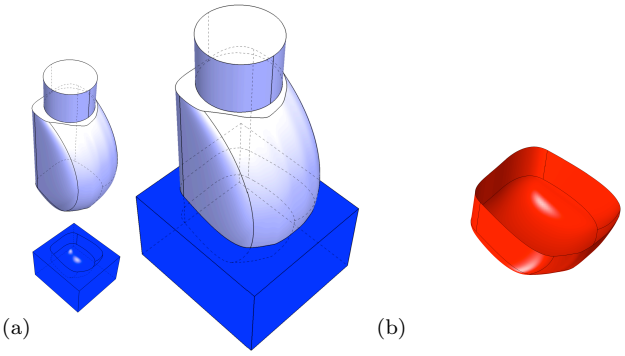
Currently, we have made assumptions concerning the boundaries and the manifoldness of the interfaces. These assumptions exclude particular cases such as line contact and point contact between components. Note however that the ray-casting part of the algorithm will still be able to detect the interfaces for these particular degenerate cases, but the graph based approach is not adapted to these cases. This limitation will be addressed in future work.

Our implementation uses a discretization of the B-Rep



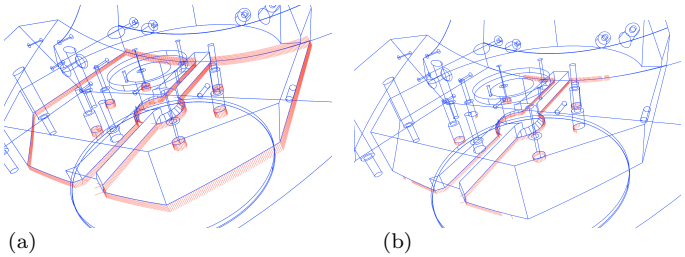
(a) (b)

Fig. 9. Bolted junction with three plates (a) Cut view (b) Interfaces.



(a) (b)

Fig. 10. Freeform surface example. (a) two components (b) their interface.



(a) (b)

Fig. 11. Influence of the length of rays. The actual length is 10 times smaller. (a) Correct length (b) Rays too short.

Table 1

model	#comp.	#B-Rep faces	#curves	#interf.	#curves in interf.	#triangles	#rays	#Interf. boundary points	Init Optix (ms)	Launch Rays (ms)	Build graph (ms)	Build step model (ms)	Total (ms)
Academic Fig. 7	2	20	48	1	12	56	118,496	10,022	16	118	7	2	143
Imprint Figs. 1, 3, 2	2	314	887	2	75	19,241	1,743,644	88,254	3758	328	31	9	4126
Freeform Fig. 10	2	20	42	5	20	27,620	189,452	52,679	5516	150	20	25	5711
Bolted Junction Figs. 9, 5	6	69	154	25	106	1,438	176,330	64,318	264	154	25	25	468
Aircraft part Fig. 12	12	498	1780	20	324	50,880	10,275,797	1,650,456	8189	2386	1083	30	11688
Car door Fig. 13	2	537	1728	122	501	139,863	5,435,386	168,978	32486	804	76	11	33377
Hydraulic pump Fig. 13	46	2158	5389	119	565	447,181	8,465,558	1,095,595	103472	1683	402	126	105683

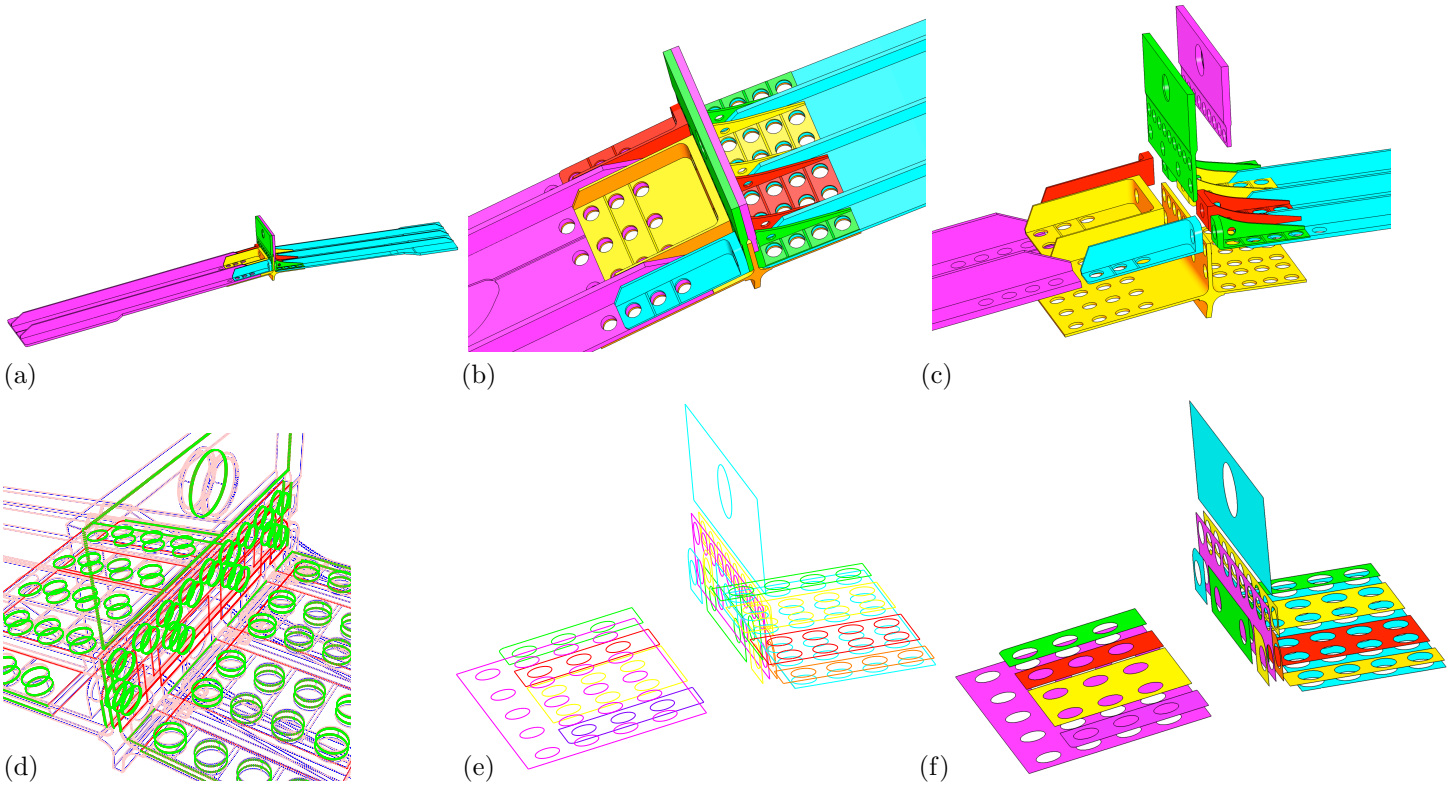


Fig. 12. Aircraft part for assembling the wings with the body of an aircraft (model courtesy of EADS). (a,b) two views of the components, (c) exploded view, (d) ray casting results, refer to caption of Fig. 5 for the color coding, (e) boundary reconstruction, (f) final interfaces

model that is stored in the GPU in order to compute the intersections with the rays. In future works, we want to store the B-Rep model on the GPU and perform the exact intersection computation without using a discretization. This should reduce memory consumption.

Also, as already mentioned in section 5, the extension to include the computation of interferences and clearance contacts is possible by adapting the ray-casting parameters, but it needs a more careful investigation of the interplay between the ray and the tolerance parameters. This will be investigated as well in the future.

ACKNOWLEDGMENTS

This work has been partly funded by the ANR project ROMMA [2]. Models are partly provided by EADS [3] and ANTECIM [4]. NVIDIA has provided professional graphics card to our research team.

[2] http://romma.lmt.ens-cachan.fr/
[3] http://www.eads.com/
[4] http://www.antecim.eu/

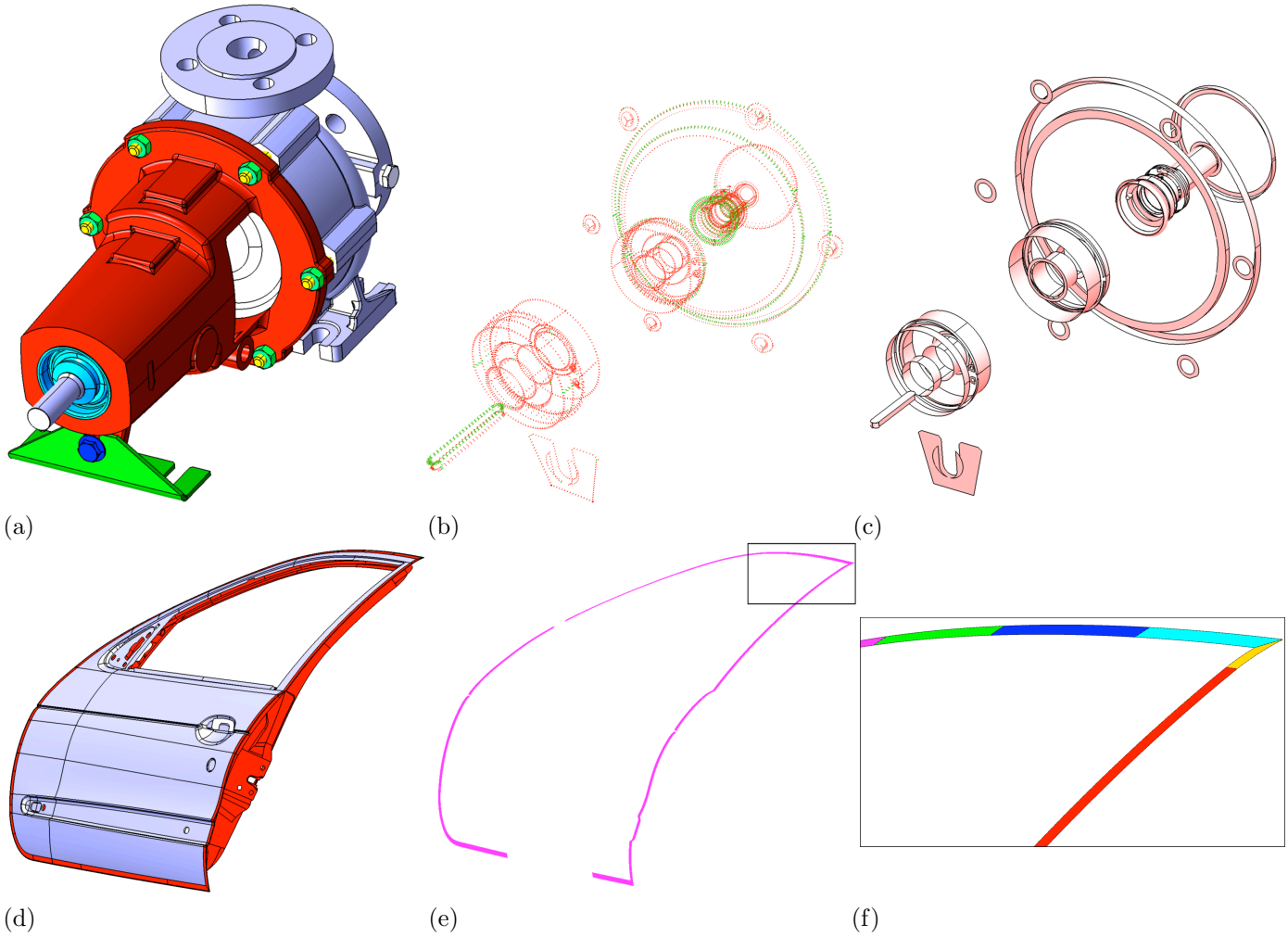


Fig. 13. Pump and car door assemblies.(a,d) Input model, (b) Results of ray-casting, refer to caption of Fig. 5 for the color coding, (c,e) Interfaces, (f) Zoom from (e) with different colors to highlight the NURBS faces of the interface.

References

[1] Iso 10303-1:1994 industrial automation systems and integration product data representation and exchange - overview and fundamental principles, international standard. *ISO TC184/SC4*, 1994.

[2] *ANSYS Workbench v.14*, 2012.

[3] *OpenCascade CAD software library*, 2012.

[4] R. Chouadria and P. Véron. Identifying and re-meshing contact interfaces in a polyhedral assembly for digital mock-up. *Eng. with Computers*, 22(1):47–58, 2006.

[5] B. Clark, B. Hanks, and C. Ernst. Conformal assembly meshing with tolerant imprinting. In *Proc. 17th Meshing Roundtable, Pittsburg, USA, October 12–15*, pages 267–280, 2008.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[7] A. Dixon and J. J. Shah. Assembly feature tutor and recognition algorithms based on mating face pairs. *CAD and Applications*, 7(3):319–333, 2010.

[8] F. Faure, J. Allard, F. Falipou, and S. Barbier. Image-based Collision Detection and Response between Arbitrary Volumetric Objects. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 155–162, Dublin, Ireland, July 2008.

[9] R. Iacob, P. Mitrouchev, and J.-C. Léon. Contact identification for assembly/disassembly simulation with a haptic device. *Visual Computer*, 24(11):973–979, 2008.

[10] K.-Y. Kim, Y. Wang, O. S. Muogboh, and B. O. Nnaji. Design formalism for collaborative assembly design. *CAD*, 36(9):849–871, 2004.

[11] A. Lee-St.John and J. Sidman. Combinatorics and the rigidity of cad systems. *CAD*, 45:473–482, 2013.

[12] C. Mascle. Feature-based assembly model for integration in computer-aided assembly. *RCIM*, 18:373–378, 2002.

[13] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich. Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics*, August 2010.

[14] K. Rahmani and V. Thomson. Ontology based interface design and control methodology for collaborative product development. *CAD*, 44(5):432–444, 2012.

[15] U. Roy and B. Bharadwaj. Design with part behaviors: behavior model, representation and applications. *CAD*, 34(9):613–636, 2002.

[16] U. Roy, N. Pramanik, R. Sudarsan, R. D. Sriram, and K. W. Lyons. Function–to–form mapping: model, representation and applications in design synthesis. *CAD*, 33(10):699–719, 2001.

[17] J. Shah and M. Mäntylä. *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*. John

Wiley & Sons, 1995.

[18] A. Shahwan, J.-C. Léon, G. Foucault, M. Trlin, and O. Palombi. Qualitative behavioral reasoning from components' interfaces to components' functions for DMU adaption to FE analyses. *CAD*, 45:383–394, 2013.

[19] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrman, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision Detection for Deformable Objects. *Computer Graphics Forum*, 2005.

[20] A. Thakur, A. Banerjee, and S. K. Gupta. A survey of cad model simplification techniques for physics-based simulation applications. *CAD*, 41(2):65–80, 2009.

[21] W. van Holland and W. F. Bronsvoort. Assembly features in modeling and planning. *RCIM*, 16:277–294, 2000.

[22] B. Wang, F. Faure, and D. K. Pai. Adaptive Image-based Intersection Volume. *ACM Transactions on Graphics*, Aug. 2012.

[23] R. Yang, X. Fan, D. Wu, and J. Yan. Virtual assembly technologies based on constraint and dof analysis. *RCIM*, 23:447–456, 2007.