HOSTED BY

ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Journal of Computational Design and Engineering 2 (2015) 55–66

www.elsevier.com/locate/jcde

Visualizing sphere-contacting areas on automobile parts for ECE inspection

Masatomo Inui, Nobuyuki Umezu*, Yuuki Kitamura

Department of Intelligent Systems Engineering, Ibaraki University, 4-12-1 Nakanarusawa, Hitachi, Ibaraki 316-8511, Japan

Received 27 September 2014; received in revised form 21 October 2014; accepted 23 October 2014
Available online 6 December 2014

Abstract

To satisfy safety regulations of Economic Commission for Europe (ECE), the surface regions of automobile parts must have a sufficient degree of roundness if there is any chance that they could contact a sphere of 50.0 mm radius (exterior parts) or 82.5 mm radius (interior parts). In this paper, a new offset-based method is developed to automatically detect the possible sphere-contacting shape of such parts. A polyhedral model that precisely approximates the part shape is given as input, and the offset shape of the model is obtained as the Boolean union of the expanded shapes of all surface triangles. We adopt a triple-dexel representation of the 3D model to enable stable and precise Boolean union computations. To accelerate the dexel operations in these Boolean computations, a new parallel processing method with a pseudo-list structure and axis-aligned bounding box is developed. The possible sphere-contacting shape of the part surface is then extracted from the offset shape as a set of points or a set of polygons.

Keywords: ECE safety inspection; Automobile part design; Collision detection; Offsetting; GPU

1. Introduction

Safety is an important concern in the design of automobile parts. The Economic Commission for Europe (ECE) defines several safety regulations on part shapes so that they do not hurt pedestrians, drivers, or passengers in the event of a car crash [1]. For instance, ECE-17 defines the shape conditions for structural parts of automobile seats, while ECE-21 and ECE-25 specify conditions on the roundness of corners in the car's interior and headrest, respectively. ECE-26 is related to ECE-21, and controls the surface roundness of exterior parts. Similar regulations are defined in other countries such as Japan, USA, and China [2].

Fig. 1 illustrates the concept of ECE-21 and ECE-25. The corners of interior parts and headrests that can be contacted by an imaginary sphere of radius 82.5 mm must have a roundness greater than R3.2. This radius is equivalent to the average head size of an infant. ECE-17 places a similar shape constraint on seats. As a result, seat designers often put thick wires under the seat (see Fig. 2) so that the head cannot come into direct contact with sharp corners. ECE-26 states that any exterior surface parts with which a sphere of radius 50.0 mm could contact must have a roundness greater than R2.5.

As exterior and interior parts strongly affect the appearance and comfort of an automobile, they are often initially designed in terms of their function and esthetics. Currently, ECE regulations on part shapes are only inspected by specialists at the final design stage. The most important task in the inspection is to detect those surface regions that can be contacted by a sphere of 50.0 mm radius (for exterior parts) or 82.5 mm radius (for interior parts). To manually check the contact conditions of each part, many 2D section drawings are produced. This work is tedious, time consuming, and prone to human errors. Thus, a fast, automatic inspection method that allows the designers themselves to check the regulations is highly desirable.

Various commercial systems enable the visualization of a sphere-contacting part shape (e.g., CAVA [3]). However, the visualization quality of such systems is not sufficient for a precise understanding of how the sphere contacts the part. According to our research, some Japanese companies have developed in-house systems for automating the inspection task.

*Corresponding author. Tel.: +81 294 38 5262; fax: +81 294 38 5229.
E-mail address: umezu@mx.ibaraki.ac.jp (N. Umezu).
Peer review under responsibility of Society of CAD/CAM Engineers.

http://dx.doi.org/10.1016/j.jcde.2014.11.006

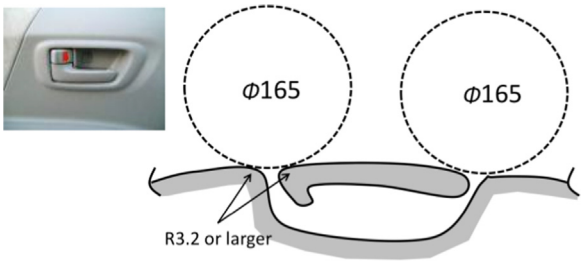

Fig. 1. Geometric description of safety regulations ECE-21 and ECE-25.

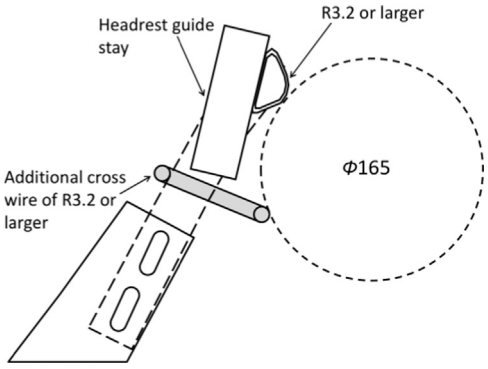

Fig. 2. Geometric description of safety regulation ECE-17.

For example, in a virtual milling method, a fixed-axis milling simulation is executed on the car part, using a ball-end cutter of specific radius to distinguish the sphere-contacting shape. The contacting sphere is assumed to approach the part along the direction of the cutter axis (the z-axis in the case of a three-axis milling). To improve the inspection quality, milling simulations must be conducted along multiple axes, which leads to considerable computation time.

To assist part designers, this paper proposes a new method for automatically detecting the sphere-contacting shape of car parts. The input data consists of a polyhedral model that precisely approximates the part shape. Consider the surface of the part offset by the radius of the sphere. The sphere can contact the part surface when its center point is located on the offset surface. In Fig. 3, $\mathbf{S}$ represents the part surface and $\mathbf{S}'$ represents its offset. The proposed method computes the offset shape of the part as the Boolean union of expanded shapes of all surface triangles of the model. A 3D model in triple-dexel representation [4–7] is adopted for stable and precise Boolean union computation, and a new parallel processing method with a pseudo-list structure and Axis-Aligned Bounding Box (AABB) is developed to accelerate the dexel operations in the Boolean computation.

Those points on the part that possibly contact the sphere are extracted and visualized using two methods. The first samples many points on the offset surface (for example, point $\mathbf{p}'$ in Fig. 3) and computes their normal vectors. Sphere-contacting points on the part are obtained by projecting the sampled points back toward the part surface along the direction of the normal to the offset surface. Point $\mathbf{p}$ in Fig. 3 corresponds to the contact point of a sphere with center point $\mathbf{p}'$. The contact region can then be visualized by coloring these contact points.

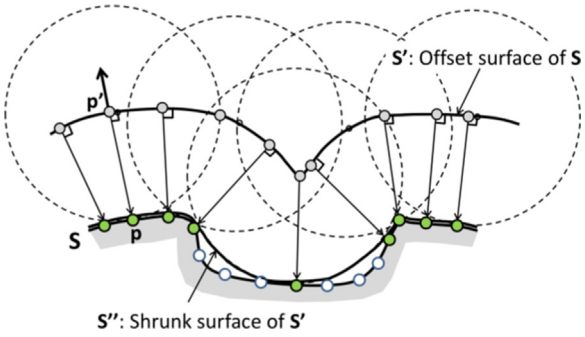

Fig. 3. Relationship between points on the offset surface and sphere-contacting points.

The second method internally offsets (shrinks) $\mathbf{S}'$ back toward the initial surface to derive a surface $\mathbf{S}''$. This is the surface formed by sliding a sphere over the part surface. The intersection between the part surface $\mathbf{S}$ and the shrunk surface $\mathbf{S}''$ corresponds to the sphere-contacting shape of the part. Our method derives the intersection surface as a set of small polygons.

This paper is organized as follows. In the next section, some related studies on the ECE regulations and offset computations are briefly reviewed. The novel contributions of our sphere-contacting shape detection algorithm are explained in Section 3. Sections 4 and 5 contain details of the dexel-based parallel offsetting algorithm, including the pseudo-list structure of the dexel data updating mechanism and the AABB thread management method. In Section 6, we describe visualization methods for the sphere-contacting shape based on the offset surface. Experimental extractions and visualization results with some complex CAD models of automobile parts are discussed in Section 7, and we summarize our conclusions in Section 8.

2. Related studies

2.1. ECE inspection

Although ECE inspections are an important topic for automobile manufacturers, technologies for automating the inspection task have rarely been studied. Many manufacturers still use manual inspection methods. The Toyota Motor Corporation submitted some patents concerning the automatic inspection of ECE-17, -21, and -25 in 2005 [8]. Toyota's method uses the offset surface of a CAD model to extract the possible sphere-contacting surface. The offset computations for complex 3D shapes are generally difficult and unstable, and Toyota's patent did not describe the technical details of their computation method. Some automobile companies use the CAVA software [3], the technical details of which are unpublished.

Yamazaki et al. proposed an ECE regulation inspection system based on detecting the intersection between a CAD model and spheres of 82.5 mm radius placed on the model surface [9]. A parallel intersection detection algorithm combined with hierarchical geometric data management was introduced to accelerate the computation. However, this method cannot be applied to parts with sharp edges and

corners, because no appropriate method for placing spheres on such surfaces was given. The current authors proposed an improved virtual milling method for the rapid detection of the sphere-contacting shape [10], with an inverted offsetting method accelerated with the depth buffer mechanism of a Graphics Processing Unit (GPU) used for the milling simulation [11]. Because this system is based on three-axis milling simulation technology, it does not overcome virtual milling's key limitation of dependence on the direction of approach of the sphere.

2.2. Offset computation

Our ECE inspection method uses the offset surface of the part model. Offsetting is one of the most fundamental operations in geometric modeling. The mathematical issues of offsetting were comprehensively studied in earlier work by Rossignac and Requicha [12]. The offsetting operation for curves and surfaces is well known [13–15]. However, the complexity of offsetting increases significantly when we consider a 3D model, because the offsetting must handle both the individual surfaces in the model as well as topological reconstruction by trimming and reconnecting the offset surfaces into a closed model. Earlier techniques for offsetting 3D models [12,16,17] are often computationally expensive, and model reconstruction can be unstable.

To overcome these difficulties, new offset computation methods based on the discrete representation of the 3D model have become popular. Known representation schemes utilize points, voxels, dexels [18], rays [19], and Layered Depth Images (LDI) [20], and various improvements, e.g., triple-dexels [4–7] have been reported. As discrete 3D models do not have surface elements, the topological reconstruction step, which is the most critical process in conventional offsetting, is not necessary. After offsetting, a polyhedral model of the offset shape is derived by applying some surface extraction technology, such as marching cubes [21] or dual contouring [22], to the discrete model.

Chen et al. proposed a point-based offsetting method [23,24] in which points are densely sampled on the surface of the input polyhedral model. Candidate points on the offset surface are generated by simply shifting the points on the polygon or replacing points on the vertices and edges with points on spheres and cylinders. After removing points located inside the offset model, a polygonal offset mesh is generated by connecting the remaining points. Liu and Wang proposed another point-set-based offsetting method [25]. Their method assumes that each sample point on the object surface has a unique normal vector. It is difficult to satisfy this condition in automobile parts with sharp edges and vertices, because points on such elements have multiple normal vectors.

Offsetting a 3D object can be recognized as a Minkowski sum between the object and a sphere of the offset radius. Lien proposed a point-based Minkowski sum operation [26] whereby the surfaces of two objects are converted to two groups of points and then summed. Points located inside the Minkowski sum object are discarded by applying a series of "filters" to determine the offset surface. The most expensive process in point-based offsetting is this filtering step, and the cost can become huge for cases, such as in a typical ECE inspection, with a large offsetting radius.

Consider a 3D object in a box-like space. The distance field is the spatial grid structure in which the distance from the point to the closest surface of the object is recorded at each grid point. Many researchers have proposed distance field-based offsetting methods [27–30]. For some offset radius r, the offset surface of the model goes across an edge connecting a grid point with a distance greater than r with another point whose distance is less than r. After detecting such edges, a marching cubes method (or similar) can determine the polygonal offset surface. A similar idea was used in [31], where several filtering methods were developed to reduce the computation cost of the distance field.

Li and McMains discussed a voxelized Minkowski sum computation with culling techniques [32,33]. Their method first generates possible surface elements of the Minkowski sum shape of two objects. Voxels corresponding to the Minkowski sum shape are then selected according to the surface elements. Unfortunately, the use of such spatial grid-based offsetting methods for ECE inspections would consume large amounts of memory to record the distance field or voxel model.

The offsetting method proposed by Wang and Manocha uses LDI to record the object shape and temporal result of the offset computation [34,35]. Zhao et al. developed a Compact LDI (CLDI) approach, which offers improved data storage technology to reduce the amount of memory required [36]. These works also use the parallel processing capability of GPUs to accelerate the computation. Because the triple-dexel representation is geometrically equivalent to LDI and CLDI, the offsetting methods proposed in [34–36] have similarities to our work. The main differences are explained in the next section.

3. Contributions of the proposed method

Our collision detection method requires the input of a precisely tessellated CAD model of automobile parts. Most commercial CAD systems provide a function to output the model data as a group of triangular polygons, for example in the STL format. The offset shape of the model is obtained as the Boolean union of expanded shapes of all surface triangles of the model. In contrast to point-based offsetting, this technique does not require any filtering operations. The Boolean operation uses the triple-dexel representation of the 3D model. This representation requires less memory than spatial grid-based methods to produce an offset shape of the same accuracy.

Our offsetting method has similarities to the shape representation techniques proposed by Wang and Manocha [34,35] and by Zhao [36], and also uses the parallel processing capability of a GPU to accelerate the computation. In contrast to these prior studies, the method proposed in this paper has the following novel features:

- Our method computes the offset shape as a union of the expanded shape of all triangles of the part. Because the

edges and vertices of the triangles contribute the resultant offset surface, this is a robust method of detecting collisions. The algorithms given in [34,35] compute the offset shape using spheres placed at points distributed on the part surface. This approach is liable to overlook collisions around sharp edges and vertices, the most critical condition in the ECE inspection.

- We introduce a new pseudo-list method to support the parallel construction of the dexel model. This enables more efficient processing than in previous approaches [34,36], which used a data storage mechanism with several redundant operations. Our pseudo-list technique also enables the dexel model to be partially updated in the model construction process, a problematic operation in the previous approaches.
- In addition to conventional point-based visualization techniques [9,10], the proposed method computes the sphere-contacting shape as a set of small polygons. This is not only desirable for visualization purposes, but also for transferring the collision detection result to other CAD systems in the standard polygon data format.

4. Preprocessing operations

To prepare data that is suitable for the proposed method, the surface polygons of the input model are classified into small groups according to their proximity. This classification proceeds according to the hierarchical structure of the AABB [37]. Consider n triangular polygons forming the model surface. An AABB that tightly contains the polygons within is defined by measuring the coordinate ranges of the polygons in the x-, y-, and z-directions. Define one root AABB holding all triangular polygons of the given model. We project the center points of all polygons in the AABB to a line parallel to its longest axis, and sort the polygons according to the order of the projected points on the line. Two AABBs are then formed by the first $n/2$ sorted polygons and the remaining $n/2$ polygons, respectively. These are considered to be two children of the original AABB. The sorting and grouping operations of polygons and the process of defining child AABBs are iterated, and a binary AABB tree is obtained. The tree construction process is terminated when all leaf AABBs of the tree contain only *nmax* or fewer polygons, where *nmax* is the maximum number of polygons allowed for each leaf AABB. In our implementation, we set $nmax=8$ based on numerical experiments. Each leaf AABB retains the number of polygons within and the geometric data (coordinates of vertices) of the polygons.

The triple-dexel representation of the 3D model is used in the offset computation and to represent the offset shape. In the original dexel modeling [18], object shapes were represented by a series of z-axis-aligned vertical segments defined for each grid point of a square mesh in the x–y plane (see Fig. 4(a)). In the dexel model, near-horizontal surfaces are precisely represented by the end points of dexel segments, whereas vertical or near-vertical surfaces have inevitable shape errors caused by the finite grid resolution in the x–y plane. The triple-dexel model was proposed to overcome this non-uniformity of representation accuracy. In the triple-dexel representation, the object shape is not only defined by a z-axis-aligned dexel model, but also an x-axis-aligned dexel model based on a grid in the y–z plane and a y-axis-aligned dexel model based on a grid in the z–x plane (see Fig. 4(b)). In this representation, the geometric information of vertical or near-vertical surfaces is accurately represented by the end points of horizontal x- or y-axis-aligned dexels [4–7].

5. Parallel offset computation

To implement parallel offsetting software, we use the Compute Unified Device Architecture (CUDA) [38]. Current GPUs are designed to have thousands of small streaming processors (SP) on a chip. By using CUDA, programmers can utilize a GPU as a general purpose parallel processor in which each SP executes a computation unit (or thread). The acceleration gained by using a GPU is mainly due to the replacement of iterative executions of a function by the parallel execution of its equivalent threads. Under CUDA, programmers can specify the execution of up to 65535 × 65535 × 512 simultaneous threads. To properly manage such a tremendous number of threads, CUDA provides grid and block

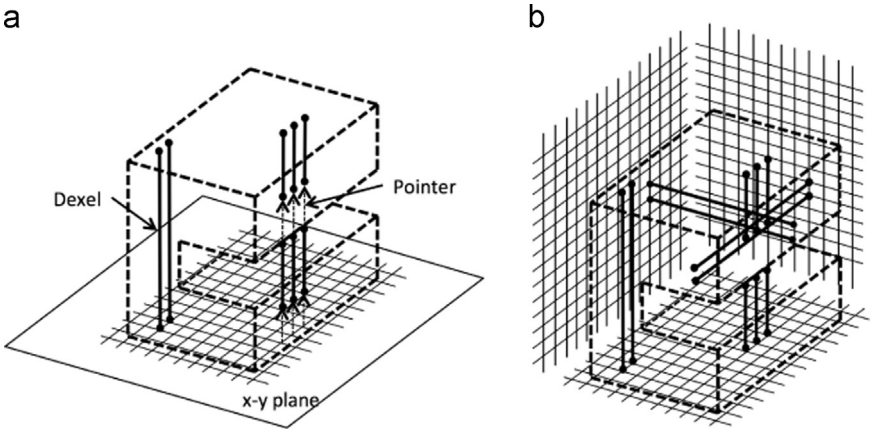

Fig. 4. Definition of dexel model and triple-dexel model.

structures. Each block is a 1D, 2D, or 3D array of threads, and each grid is a 1D or 2D array of blocks.

Within a PC, the CPU and GPU are independent units with their own memory systems. Data for GPU processing are usually prepared in the main CPU memory before being transferred to the graphics memory. In our parallel offset computation, the basic data are the vertex coordinates of the triangular polygons within each leaf AABB. Before executing the threads for the offset computation, these are transferred to the GPU. After the threads have completed their execution, the triple-dexel model representing the offset shape of the part is recorded in the graphics memory. The model data is then transferred from the graphics memory to the main CPU memory and used in the visualization process.

5.1. Parallel offsetting of polygons within single AABB

In the following sections, we explain the offset computation with z-axis-aligned dexels. In the actual processing, the same operations are repeated for the x- and y-axis-aligned dexels. Before the processing commences, a null z-axis-aligned dexel model (the Offset Dexel Model, *ODM*) is prepared on a regular square grid in the x–y plane. The grid is defined so that it holds the projection of the offset model to the x–y plane within. The resolution of the grid is determined according to the accuracy requirement of the offset computation and the available memory size. The *ODM* is updated in the following computation, and finally forms the offset shape model of the part in the dexel representation.

Each leaf AABB of the AABB tree is the basic processing unit of the offset computation. A single leaf AABB holds at most *nmax* polygons in its neighborhood. These are expanded by radius r (where r is 50.0 mm for exterior parts and 82.5 mm for interior parts). The expanded shape of the polygons is equivalent to a Boolean union shape of spheres, cylinders, and thick plates (slabs) placed on the polygons as follows:

- Spheres of radius r are placed on all vertices of the polygons.
- Cylinders of radius r are placed along each edge **e** of the polygons, with the center axis of the cylinders coinciding with **e**.
- Slabs with the area of each face and thickness $2r$ are placed on each polygonal face **f**, with the center plane of the slabs coinciding with **f**.

According to the expansion of the polygons, the AABB holding the polygons is also enlarged by r to properly enclose the expanded polygons within. This operation is achieved by simply shifting the six rectangles of the AABB in their outward directions, as shown in Fig. 5.

The projection of the enlarged AABB to the x–y plane limits the range of vertical dexels that can possibly intersect the expanded triangles in the AABB (see Fig. 6). For each grid point within the projection, we compute the intersection of a vertical line starting from that point with the expanded

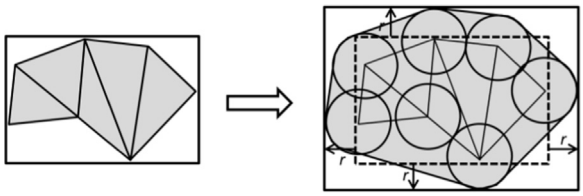

Fig. 5. Enlargement of an AABB for holding expanded polygons.

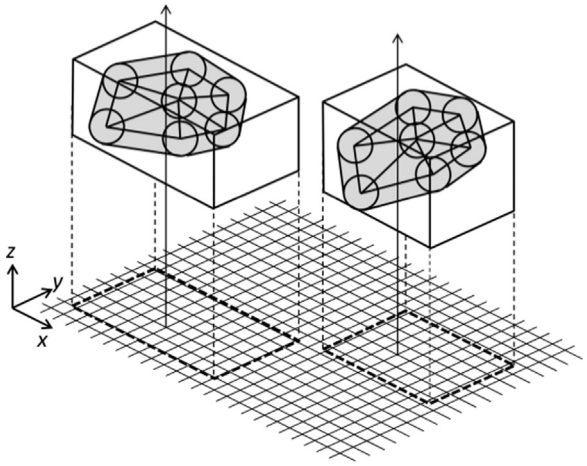

Fig. 6. Projection of two enlarged AABBs with expanded polygons.

polygons, and generate a temporal dexel model of the expanded polygons (see Fig. 7(a)). The dexel-wise Boolean union of the temporal dexel model and the *ODM* is then computed, and the result gives the new *ODM* (Fig. 7(b)). In this process, the Boolean union of a series of dexels in the expanded triangles and another series of dexels in the *ODM* on the same grid point is calculated for all grid points within the projection.

This Boolean union computation of dexels on a grid point is independent of those on other grid points. Thus, the dexel-wise Boolean union computation can be parallelized. A single CUDA thread is assigned to each grid point. This thread computes a series of dexels for the temporal dexel model of the expanded triangles on the grid point. It then executes the above dexel-wise Boolean union computation. By invoking threads for all grid points within the projection of the expanded AABB, we obtain part of the offset dexel model for the triangles in the leaf AABB. This operation is executed for all leaf AABBs, and the offset computation of the part model is complete.

In the current implementation, the following values are stored in each dexel of the *ODM*:

top_v*, *bottom_v: z-coordinate values of the top and bottom end points of a dexel. For the x- or y-axis-aligned dexels, the x- or y-values of the end points are stored instead.

top_ID*, *bottom_ID: Identification numbers of component elements (spheres, cylinders, or slabs) of the expanded triangles forming the top and bottom end points of the dexel. This information is obtained during the conversion process from expanded triangles to dexel segments. Identification numbers are then transferred to the dexels of the *ODM* in the following dexel-wise Boolean union operation.

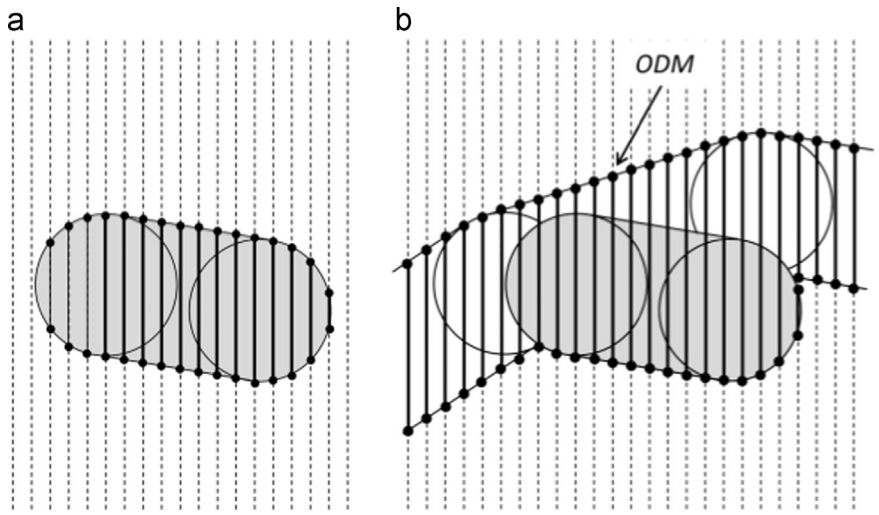

Fig. 7. Computation process of a temporal dexel model (a) and Boolean union computation of the temporal dexel model and *ODM* (b).

5.2. Parallel dexel storage mechanism

As CUDA threads can simultaneously execute the dexel-wise Boolean union operations for multiple grid points, we require a data storage mechanism that is suitable for the parallel updating of the dexel model. Prior work [34,36] has implemented such mechanisms using the counting function and the exclusive scan function [38].

5.2.1. Counting and scan-based method

Consider the resultant dexel model to be stored in a global array *dexel*[ ]. The counting function and scan-based method store the data in a two-pass manner. On the first pass, the number of dexels in the model resulting from the Boolean union operation is counted for each grid point. The counting results are stored in a global array *count*[ ], where *count*[*i*] records the number of dexels for a grid point **gp***i*. A new array *start*[ ] is prepared and filled by applying the exclusive scan operation to *count*[ ]. In this operation, the sum from *count*[0] to *count*[*i*–1] is set to *start*[*i*]. The Boolean union computation of the two dexel models is then repeated to store the resultant dexels in *dexel*[ ], where resultant dexels for **gp***i* are stored from *dexel*[*start*[*i*]] to *dexel*[*start*[*i*]+*count*[*i*]−1].

The mechanism described above has two drawbacks. First, there is a degree of redundancy in executing the same dexel-wise Boolean operation twice. Second, it is difficult to deal with the partial modification of the dexel model. Because the dexel data of the *ODM* are packed in a 1D array *dexel*[ ], the addition of new dexels to modify part of the dexel model requires all of the data stored in *dexel*[ ] to be updated.

5.2.2. Pseudo-list method

To overcome these problems, we have developed a new pseudo-list data storage mechanism for the parallel updating of the dexel model. A linked list structure is suitable for recording a series of dexels for each grid point. As shown in Fig. 4(a), a pointer to the first dexel element in the series is assigned to each grid point. Each dexel element has a pointer to the next element in the series; the last dexel has a null pointer. Although there are several methods for implementing a list structure in CUDA, none have been published or sufficiently investigated. As CUDA does not allow the dynamic allocation of device memory during processing, most of the known methods allocate a sufficient amount of memory in advance as an array. Instead of a pointer, they use the index number of each array element to reference a specific element in a list. The pseudo-list structure proposed in this paper follows this strategy to realize list-like processing of dexels in the parallel dexel model construction.

A sufficient number of dexel elements with empty data are generated and prepared in the global array *dexel*[ ]. A global index counter *idx* pointing to the first available element in *dexel*[ ] is also prepared and initialized to 0. The pseudo-list structure uses the following 1D integer arrays:

start*[ ]**: *start*[*i*] corresponds to a pointer from a grid point **gpi* to the first element in the associated dexel list. The size of *start*[ ] is the same as the total number of grid points in the *x*–*y* plane.

end*[ ]**: *end*[*i*] represents another pointer from **gpi* to the last element in the dexel list. This array holds the same number of elements as *start*[ ].

***next*[ ]**: Each element of *next*[ ] represents a pointer to the next dexel in the same list. The memory allocated to *next*[ ] must be sufficient to allow all dexels generated in the model construction process to record their next elements.

Before starting the model construction, all elements in *start*[ ], *end*[ ], and *next*[ ] are initialized to –1.

Consider the addition of a new dexel element to the end of a dexel list starting from grid point **gp***i*. A dexel with null data is obtained from the array *dexel*[*idx*]. The index number *idx* is then updated via the atomic increment function [39]. (In a parallel processing environment, “atomic” functions are guaranteed to execute sequentially. The increment operation of a global value *a* is not guaranteed to change its value in order,

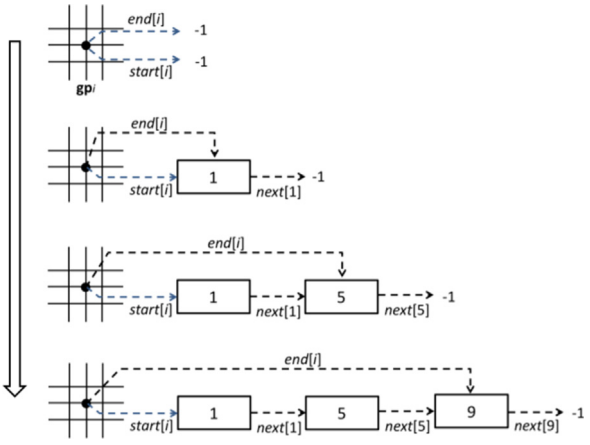

Fig. 8. Updating a dexel list to insert three new dexels.

because multiple threads can update the value of *a* simultaneously. By using the atomic incremental function, such conflicts can be avoided and *idx* can be used as an incremental counter.)

The atomic exchange function is then used to replace the value of *end*[*i*] with *idx*. This function returns the old value of *end*[*i*]. If the returned value *old_idx* is –1, *dexel*[*idx*] is the first dexel element of a dexel list for **gp***i*. In this case, the value of *start*[*i*] is also updated to *idx*. Otherwise, *old_idx* represents the index number of the last dexel element in the dexel list for **gp***i*. *idx* is thus set to *next*[*old_idx*] to record that the next dexel element of *dexel*[*old_idx*] is *dexel*[*idx*]. Fig. 8 illustrates the update process for *start*[*i*], *end*[*i*], and *next*[ ] to insert *dexel*[1], *dexel*[5], and *dexel*[9] to an empty dexel list for grid point **gp***i*.

Employing the pseudo-list allows the simple insertion of a new dexel element to a dexel list. In contrast to the counting and scan-based method, our technique realizes the parallel data storage of the dexel model in a single-pass operation. Because the modification of a dexel list of a grid point does not affect the dexel lists of other grid points, we can partially update the dexel model. That is, the operation of inserting dexels to a pseudo-list does not have to be executed sequentially, and multiple threads can execute insertion operations to the same pseudo-list in parallel. As the atomic incremental function and atomic exchange function automatically control the execution sequence of the actual insertions, we can maintain a pseudo-list with consistent references to dexel elements during the construction process.

However, the pseudo-list method does not easily allow dexel elements to be eliminated from a list. In our current implementation, dexel elimination is realized by simply marking dexels as invalid. As the dexel operations continue, more invalid dexels are generated, and these may eventually constitute all predefined dexel elements in *dexel*[ ]. To prevent this problem, our offsetting program prepares a kind of garbage collection mechanism. At certain processing intervals, the dexel model data in the graphics memory are transferred to the main CPU memory. Pointer references to the invalid dexels are properly eliminated by the host program in the CPU, and a new dexel model with only valid dexels is written back to the graphics memory for continued processing on the GPU.

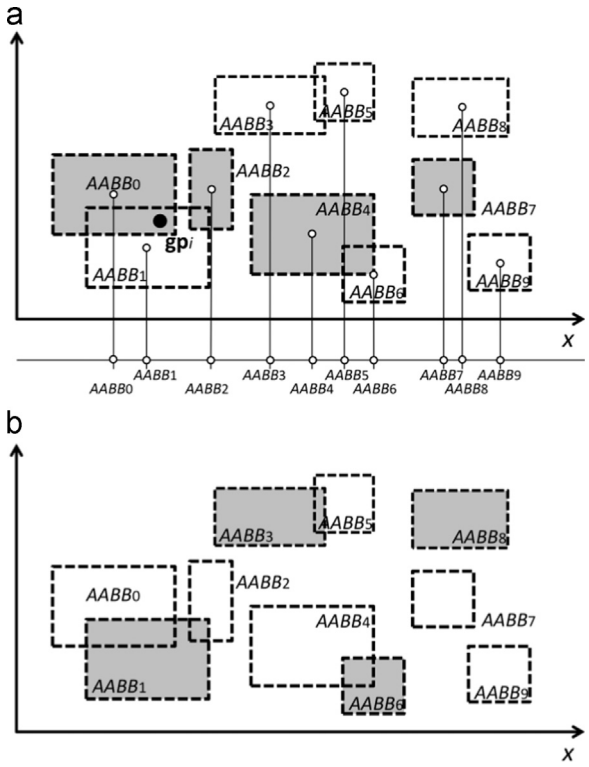

Fig. 9. Selection process of AABBs without mutual intersections of their projections to the *x*–*y* plane.

5.3. Parallel offsetting of polygons within multiple AABBs

To further utilize the parallel processing capability of GPUs, the offset computation for a single leaf AABB given in Section 5.1 is extended to parallel offset computations with multiple AABBs. In this case, we need a mechanism to avoid conflicts in the dexel-wise Boolean operation. Fig. 9(a) illustrates such a case. Consider the offset computations for polygons in *AABB*0 and *AABB*1 being simultaneously invoked. Because the expanded boxes of *AABB*0 and *AABB*1 have intersections in their projection to the *x*–*y* plane, a CUDA thread for *AABB*0 and another thread for *AABB*1 may simultaneously try to update dexels on the same grid point **gp***i* in the intersection region. To avoid such conflicts in our parallel offsetting framework, multiple AABBs must be selected so that the projections of their expanded shapes do not intersect.

This selection of expanded AABBs is realized as follows. We project the part model to the *x*–*y* plane to check the coordinate ranges of the projection in the *x*- and *y*-directions. If the *x*-coordinate range is larger than the *y*-coordinate range, then all leaf AABBs are sorted according to the *x*-coordinates of their center points. Otherwise, the AABBs are sorted by *y*-coordinates. In Fig. 9(a), small white circles represent the center points of AABBs. In the following explanation, we assume the sorting result proceeds according to the *x*-coordinates. A global variable *Rmax* is initialized to a small value. The sorted leaf AABBs are then visited in ascending order. For each visited AABB, the *x*-coordinate range of the expanded box [*xmin*, *xmax*] is checked. If *xmin* is larger than *Rmax*, then

this expanded leaf AABB is selected and the value of *Rmax* is updated to *xmax*. This selection process is repeated until all sorted leaf AABBs have been visited and a set of expanded AABBs without mutual intersections in their projection has been obtained. The selection of AABBs according to *y*-coordinates is realized in a similar manner.

The parallel offsetting operations are executed with the selected AABBs. The same AABB selection and offsetting operations are iterated until all AABBs are used in the offset computation. Consider the leaf AABB placement shown in Fig. 9. AABBs are sorted and indexed according to the *x*-coordinate of their center points. In the first selection process, four AABBs are obtained: *AABB*0, *AABB*2, *AABB*4, and *AABB*7 (see Fig. 9(a)). The same selection process is then repeated, giving *AABB*1, *AABB*3, *AABB*6, and *AABB*8 (see Fig. 9(b)). In our implementation, the offsetting operation of selecting multiple AABBs with the above algorithm generally reduces the computation time by 10–15% over offsetting with a single AABB.

The set of AABBs selected by the proposed method is not optimal. In the first selection process, *AABB*3 (or *AABB*5), *AABB*8, and *AABB*9 could also be chosen, because they do not intersect with each other or with *AABB*0, *AABB*2, *AABB*4, and *AABB*7 (see Fig. 9(a)). Although we tested several sophisticated methods for selecting more AABBs without mutual intersections, they generally increase the computation time of selection and decrease the overall performance of the offsetting operation.

6. Visualization of sphere-contacting shape

The sphere-contacting shape is visualized using the triple-dexel representation of the offset part model. Two visualizations are developed, one that displays the sphere-contacting shape with points on the part surface and another that displays the shape as a group of small polygons.

6.1. Visualization with points

In the *z*-axis-aligned dexel model, a dense set of points covering the offset surface can be obtained using the *z*-values of the end points of a dexel and the (*x*, *y*) coordinates of its corresponding grid point in the *x*–*y* plane. Points covering the offset surface in the *x*- and *y*-axis-aligned dexel model can be obtained in a similar manner. The component spheres, cylinders, or slabs contributing to the offset shape at these points can be distinguished by the ID numbers recorded at the end points of the dexels. These IDs allow a precise calculation of the normal vector at the end points of the offset surface. If a dexel end point **p** on the offset is part of a sphere whose center is at **c**, then the vector from **c** to **p** corresponds to the normal vector **n** at **p** (see Fig. 10(a)). The normal vectors for points on the cylinders and slabs can be computed in a similar manner (see Fig. 10(b) and (c)).

The negative normal vector at **p** projects the point back to the part surface as $\mathbf{p} - r\,\mathbf{n}$, where **n** is the unit normal vector at **p** and *r* is the offset radius (50.0 mm for exterior parts and 82.5 mm for interior parts). This projection is repeated for all dexel end points in the triple-dexel model representing the offset shape, and the sphere-contacting shape is finally obtained as a set of points covering the part surface.

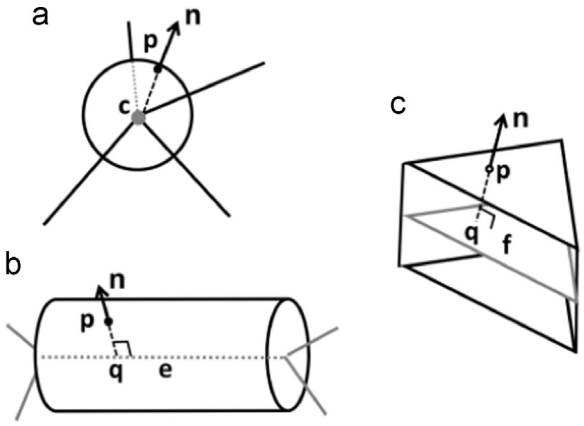

Fig. 10. Normal vector computation methods.

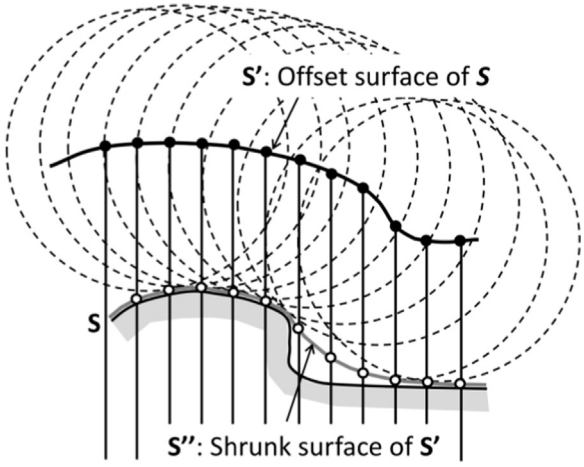

Fig. 11. Dexel of the shrunken shape.

6.2. Visualization with polygons

We have developed a second technique for visualizing the sphere-contacting shape. In this method, the offset shape is again offset in its negative direction, or "shrunk," by the same offset radius. This shrunken surface represents the surface given by a sphere sliding across the part surface. Thus, the intersection of the shrunken surface and the original surface of the given part model corresponds to the sphere-contacting shape on the part.

The shrinking operation is realized using the method proposed by Wang and Manocha [34]. We place spheres of the shrinking radius at all dexel end points in the offset model. The dexel-wise subtraction of these spheres from the given offset model gives a new dexel model of the shrunken shape (see **S″** in Fig. 11). The pseudo-list mechanism described in the previous section is used to record the subtracted dexels for

each grid point. In our current implementation, the shrinking operation uses spheres whose radius is $d=0.1$ mm smaller than the original offset. This is necessary to properly handle a model with an open surface. As such a model has no thickness, the offsetting and shrinking operations would produce null dexels, and the extraction of the sphere-contacting shape would be difficult.

After the shrinking operation, the remaining triple-dexel model is converted to an equivalent polyhedral model. The marching cubes or dual contouring algorithms could be used in this conversion (see [36] for details). In the current implementation, small polygons are placed on the top and bottom sides of the shrunken dexel model to give a set of small polygons wrapping the surface of the model. The intersection of the original surface of the part model and the surface of the shrunken model is determined by selecting the surface polygons of the shrunk model whose distance to the original surface is sufficiently small (smaller than $\varepsilon=0.15$ mm in the current implementation; this value must be greater than d). As a result, the sphere-contacting shape is obtained as a set of small polygons.

7. Experimental results

The sphere-contacting shape visualization system was implemented using Visual C++, CUDA 6.0, and OpenGL, and a series of computational experiments were performed using an Intel Core i7 Processor (3.4 GHz) with 12 GB memory and an nVIDIA GeForce GTX-650 Ti GPU. Table 1 lists the time required to compute offset models in the triple-dexel representation. We applied the system to six polyhedral models of automobile parts. One model was provided by an automobile parts manufacturer, and the other five models were selected from CAD demonstration models and an automobile company archive [40]. To evaluate the performance of the system under equal conditions, the offset radius was set to 82.5 mm for all parts. Table 1 also lists the number of polygons representing the part shape, and the grid resolutions for the triple-dexel model in the x–y, z–x, and y–z planes. The computation times for the offsetting operation with multiple AABBs indicate that the offset shapes of the complex part models were computed in 10–30 min.

Fig. 12 illustrates six sample parts. Their offset shapes in the triple-dexel representation are given in Fig. 13. In this figure, the shaded image of the offset shape was generated by painting each end point of the dexels with normal vector information at the point on the offset surface. These normal vectors were calculated using the method explained in Section 6.1. Figs. 14 and 15 show the detection results of the possible sphere-contacting shapes of these parts. Fig. 14 illustrates the resultant shapes with colored points on the part surface, and Fig. 15 visualizes the result using small polygons. These were extracted by applying the shrinking operation to the offset surface. Fig. 16 illustrates the difference between these two

Table 1
Computation times required for offsetting.

Sample	No. of polygons	Grid Res. in x-y plane	Grid Res. in z-x plane	Grid Res. in y-z plane	Required time (s)
A	597,455	895 × 1827	1827 × 709	895 × 709	774.53
B	369,408	1794 × 774	774 × 913	1794 × 913	594.58
C	569,352	1168 × 1400	1400 × 658	1168 × 658	305.21
D	1,488,731	672 × 1559	1559 × 642	672 × 642	1865.50
E	2,097,122	1432 × 736	736 × 431	1423 × 431	573.82
F	2,097,122	1473 × 711	711 × 512	1473 × 512	566.48

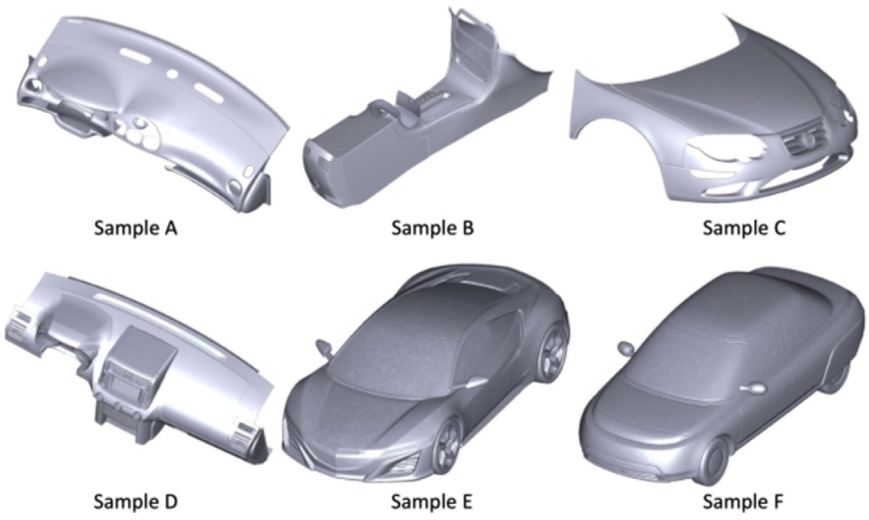

Fig. 12. Sample parts.

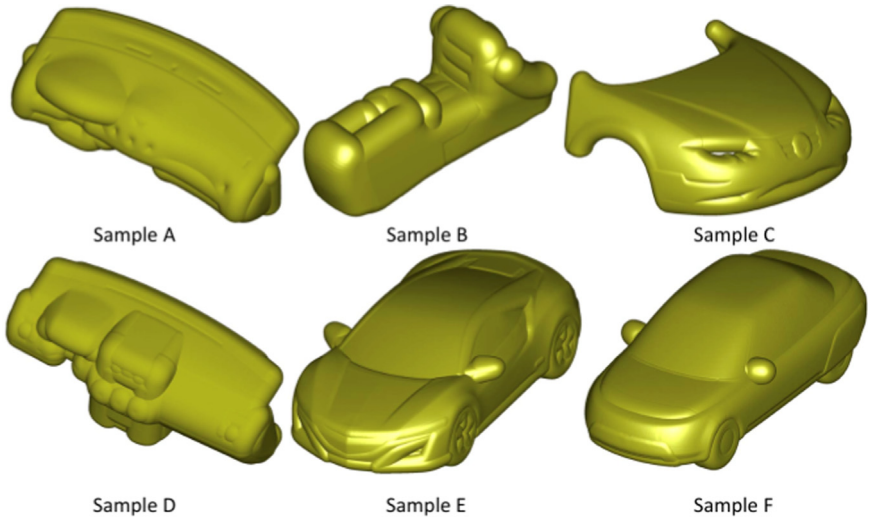

Fig. 13. Offset shapes of the sample parts in the triple-dexel representation.

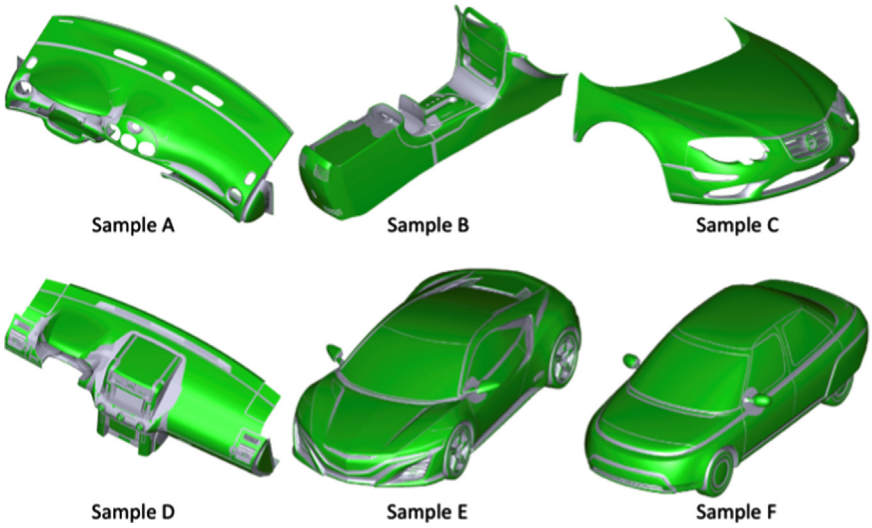

Fig. 14. Detection results of the sphere-contacting shape for the sample parts. Resulting shapes are visualized with colored points on the part surface.

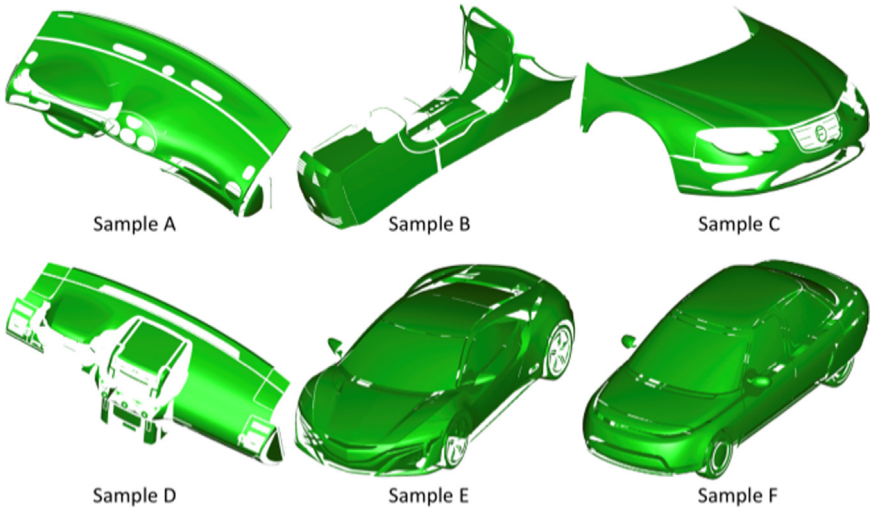

Fig. 15. Detection results of the sphere-contacting shape for the sample parts. Resulting shapes are visualized with small polygons.

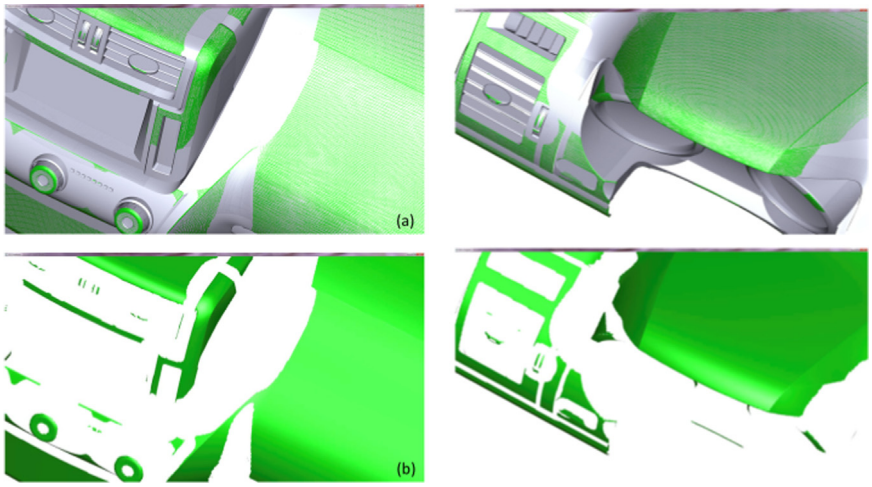

Fig. 16. Difference between (a) point-based visualization and (b) polygon-based visualization.

methods. The results are similar, but the visualization with polygons (Fig. 16(b)) produces finer details on the sphere-contacting shape.

8. Conclusions

In this paper, we have proposed a new method for assisting ECE safety inspections of interior and exterior automobile parts. The proposed method extracts the possible sphere-contacting shape from the offset surface of the part. The offset shape of the polyhedral part model is computed as the Boolean union of expanded shapes of all surface triangles of the model. The triple-dexel representation of a 3D model is adopted for stable and precise Boolean computations. A new parallel offsetting algorithm with a pseudo-list structure and AABB was developed to accelerate the dexel operations using parallel computations on a GPU.

Using the offset shape, we extracted the possible sphere-contacting shape on the part surface. We examined a novel method of visualization in which the offset surface is itself shrunk to derive the surface generated by the sphere being slid across the part surface. The intersection between the part surface and shrunken surface corresponds to the sphere-contacting shape. Our system is still in the experimental stage. We are preparing a field test of the system in the actual automobile design process, and further improvements based on the comments and requests from designers will be reflected in our future work.

Conflict of interest

The authors have no conflict of interest directly relevant to the content of this article.

References

[1] The Web Portal for The Crash Test Engineer [Internet]. European Regulations related to Crash Testing [cited 2014 Sep 27]. Available from: ⟨http://www.crash-network.com/Regulations/ECE_Regulations/ece_regulations.html⟩.

[2] Japan Automobile Standards Internationalization Center (JASIC) [Internet]. [cited 2014 Sep 27]. Available from: ⟨http://www.jasic.org/e/index_e.htm⟩.

[3] EDS Technologies [Internet]. [cited 2014 Sep 27]. Available from: ⟨http://www.edstechnologies.com/⟩.

[4] Benouamer MO, Michelucci D. Bridging the gap between CSG and BREP via a triple ray representation. In: Proceedings of ACM Symposium on Solid Modeling and Applications; 1997; p. 68–79.

[5] Muller H, Surmann T, Stautner M, Albersmann F, Weinert K. Online sculpting and visualization of multi-dexel volumes. In: Proceedings of the 8th ACM Symposium on Solid Modeling and Applications; 2003; p. 258–261.

[6] Ren Y, Zhu W, Lee Y-S. Feature conservation and conversion of tri-dexel volumetric models to polyhedral surface models for product prototyping. *Computer-Aided Design and Applications* 2008;**5**(6)932–41.

[7] Zhang W, Leu MC. NC machining simulation based on triple-dexel representation. In: Proceedings of International Symposium on Flexible Automation; 2008; ISFA2008U_100.

[8] Toyota Motor Corporation [Internet]. [cited 2014 Sep 27]. Available from: ⟨http://www.j-tokkyo.com/2006/G06F/JP2006-277304.shtml⟩ [in Japanese].

[9] Yamazaki S, Baba T, Umezu N., Inui M. Fast safety verification of interior parts of automobiles. In: Proceedings of IEEE International Conference on Mechatronics and Automation (ICMA); 2011; p. 1957–1962.

[10] Inui M, Umezu N. Fast detection of head colliding shapes on automobile parts. *Journal of Advanced Mechanical Design, Systems, and Manufacturing* 2013;**7**(5)818–26.

[11] Inui M, Ohta A. Using GPU to accelerate die and mold fabrication. *IEEE Computer Graphics and Applications.* 2007;**27**(1)82–8.

[12] Rossignac JR, Requicha AAG. Offsetting operations in solid modelling. *Computer Aided Geometric Design* 1986;**3**:129–48.

[13] Maekawa T. An overview of offset curves and surfaces. *Computer-Aided Design* 1999;**31**:165–73.

[14] Pham B. Offset curves and surfaces: a brief survey. *Computer-Aided Design* 1992;**24**(4)223–9.

[15] Hoffman CM. *Geometric and Solid Modeling: An Introduction*. Burlington (MA): Morgan Kaufmann; 1989.

[16] Satoh T, Chiyokura H. Boolean operations on sets using surface data. In: Proceedings of ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications; 1991; Austin, Texas; p. 119–127.

[17] Forsyth M. Shelling and offsetting bodies. In: Proceedings of the Third Symposium on Solid Modeling and Applications; 1995; Salt Lake City, Utah; p. 373–381.

[18] VanHook T. Real-time shaded milling display. *Computer Graphics (Proceedings of ACM SIGGRAPH)* 1986;**20**(4)15–20.
[19] Menon JP, Marisa RJ, Zagajac J. More powerful solid modeling through ray representations. *IEEE Computer Graphics and Applications* 1994;**14**(3)22–35.
[20] Shade J, Gortler S, He LW, Szeliski R. Layered depth image. Computer Graphics (Proceedings of ACM SIGGRAPH); 1998; p. 231–242.
[21] Lorensen WE, Cline HE. Marching cubes: a high resolution 3D surface construction algorithm. *Computer Graphics (Proceedings of ACM SIGGRAPH)* 1987;**21**(4)163–9.
[22] Losasso TJF, Schaefer S, Warren J. Dual contouring of hermite data. Computer Graphics (Proceedings of ACM SIGGRAPH); 2002; p. 339–346.
[23] Chen Y, Wang H, Rosen DW, Rossignac J. Filleting and rounding using a point-based method. In: Proceedings of ASME 2005 International Design Engineering Technical Conference; 2005; p. 533–542.
[24] Chen Y, Wang H, Rosen DW, Rossignac J. A Point-Based Offsetting Method of Polygonal Meshes. Technical Report. Georgia Institute of Technology; 2005.
[25] Liu S, Wang CCL. Duplex fitting of zero-level and offset surfaces. *Computer-Aided Design.* 2009;**41**(4)268–81.
[26] Lien JM. Covering Minkowski sum boundary using points with applications. *Computer Aided Geometric Design* 2008;**25**:652–66.
[27] Breen DE, Mauch S. Generating shaded offset surfaces with distance, closest point and color volumes. In: Proceedings of International Workshop on Volume Graphics; 1999; p. 307–320.
[28] Breen DE, Mauch S, Whitaker RT. 3D scan conversion of CSG models into distance volumes. In: Proceedings of IEEE Symposium on Volume Visualization; 1998; p. 7–14.
[29] Huang J, Li Y, Crawfis R, Lu SC, Liou SY. A complete distance field representation. In: Proceedings of the Conference on Visualization; 2001; p. 247–254.
[30] Liu S, Wang CCL. Fast intersection-free offset surface generation from freeform models with triangular meshes. *IEEE Transaction on Automation Science and Engineering* 2011;**8**(2)347–60.
[31] Wang CCL, Chen Y. Thickening freeform surfaces for solid fabrication. *Rapid Prototyping Journal* 2013;**19**(6)395–406.
[32] Li W, McMains S. A GPU-based voxelization approach to 3D Minkowski sum computation. In: Proceedings of the 14th ACM Symposium on Solid and Physical Modeling; 2010; p. 31–40.
[33] Li W, McMains S. Voxelized Minkowski sum computation on the GPU with robust culling. *Computer-Aided Design* 2011;**43**(10)1270–83.
[34] Wang CCL, Manocha D. GPU-based offset surface computation using point samples. *Computer-Aided Design* 2013;**45**(2)321–30.
[35] Wang CCL. Computing on rays: a parallel approach for surface mesh modeling from multi-material volumetric data. *Computers in Industry* 2011;**62**(7)660–71.
[36] Zhao H, Wang CCL. Parallel and efficient boolean on polygonal solids. *The Visual Computer* 2011;**27**(6–8)507–17.
[37] Moller T, Haines E. *Real-time Rendering*. Natick (MA): A K Peters; 1999.
[38] Thrust – Parallel Algorithms Library [Internet]. [cited 2014 Sep 27]. Available from: ⟨http://thrust.github.io/⟩.
[39] nVIDIA. CUDA Compute unified device architecture programming guide; 2007.
[40] Honda Motor Company [Internet]. Honda 3D Design Archives [cited 2014 Sep 27]. Available from: ⟨http://www.honda-3d.com/⟩.