Title:
Fast Dexelization of Polyhedral Models Using Ray-Tracing Cores of GPU

Authors:
Masatomo Inui, masatomo.inui.az@vc.ibaraki.ac.jp, Ibaraki University
Kohei Kaba, 19nm425n@vc.ibaraki.ac.jp, Ibaraki University
Nobuyuki Umezu, nobuyuki.umezu.cs@vc.ibaraki.ac.jp, Ibaraki University

Keywords:
Dexel Modeling, Polyhedron, Ray-Tracing, Data Conversion, Graphics Processing Unit

DOI: 10.14733/cadconfP.2020.188-192

Introduction:
Boundary representation (B-reps) is a standard method of solid modeling for CAD systems of mechanical products. Most operations related to B-reps models, especially Boolean set operation, require techniques like intersection calculation of the surface elements, trimming operation of the elements based on calculations, and reconstruction of adjacency relationships of the trimmed elements. These operations are computationally expensive, and the associated topological reconstruction process tends to be unstable owing to the existence of unavoidable floating-point errors in the calculations. Techniques using voxels, dexels, rays, or layered depth images based on the uniform decomposition of the 3D space have been widely used as solid modeling methods that do not suffer from these problems. With the popularization of these methods, dexel-based solid modeling [3] is rapidly becoming a standard method to represent object shapes in the NC milling simulation.

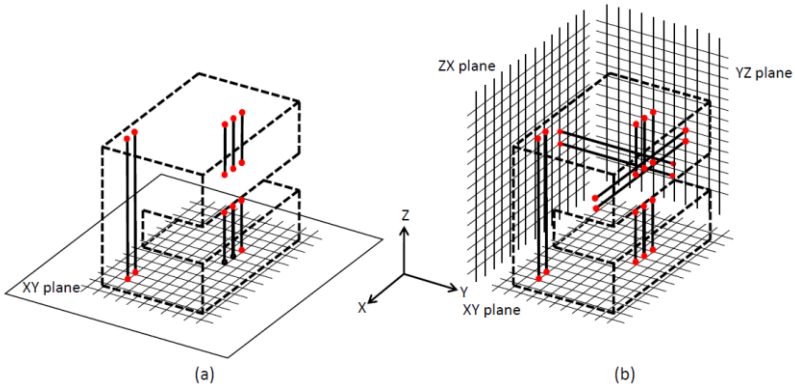


Fig. 1: (a) Dexel model and (b) Triple-dexel model with X, Y, Z axis aligned dexels.

In dexel modeling, the workpiece shape is represented by a bundle of Z-axis-aligned segments defined corresponding to each grid point of a square mesh in the XY plane (consult Fig. 1(a)). As no topological information is used in the model representation, Boolean operations in dexel modeling are much more robust than the operations used in B-reps modeling. During dexel modeling, near-vertical surfaces inevitably exhibit significant shape errors caused by finite grid resolution. The triple-dexel model was proposed to overcome this non-uniformity in shapes and to realize an accurate shape representation (consult Fig. 1(b)). In this representation, besides being defined by a Z-axis-aligned dexel model, the 3D

shape is also defined by an X-axis-aligned dexel model based on a square mesh in the YZ plane and a Y-axis-aligned dexel model based on a mesh in the ZX plane.

During dexel-based milling simulation, initial workpiece models are usually available as B-reps models. The simulation converts them to their equivalent dexel models during the initial step. The workpiece model is affixed to a table in the milling environment during the simulation. During conventional 3-axis machining, the orientation of the workpiece is changed after the machining for a certain side is finished, and it is remounted on the table to machine another of its sides. As the definition of the dexel model is based on a square mesh fixed in the coordinate plane, rigid body transformations, especially rotation of the object, are difficult. Therefore, it is necessary to initially convert the dexel model to a B-reps model, and revert it to a dexel model again following any necessary rigid body transformations.

During the conversion of a B-reps model into a dexel model, ray algorithm proposed in [1] is usually used. It defines a set of rays perpendicular to the coordinate plane corresponding to all grid points of the square mesh in the plane. The intersections between the rays and the B-reps model are computed, and subsequently converted into dexel segments. A B-reps model representing a resultant shape from a milling operation often comprises a high number of small polygons — sometimes exceeding 10,000,000 polygons. This causes the ray algorithm to take upwards of 10 minutes to convert the model into an accurate dexel model defined based on a 5,000×5,000 resolution grid. Attempting to reduce of the conversion time has become a serious problem. State-of-the-art GPUs are equipped with special hardware named RT cores dedicated to image processing called ray-tracing in 3D computer graphics. As an initial step in the application of RT cores in dexel modeling, we propose a novel method for fast dexelization of complex polyhedral models using RT cores in this paper.

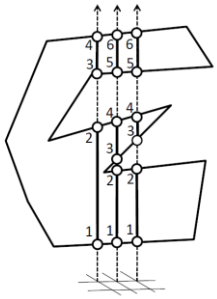


Fig. 2: Polyhedron-to-dexel conversion.

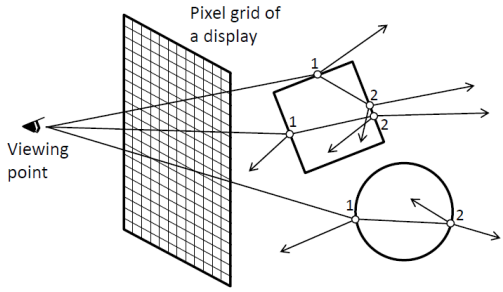


Fig. 3: Computation process in the ray-tracing.

Dexelization of Polyhedral Solid Model:

Fig. 2 illustrates the fundamental processing steps involved in the ray algorithm to convert a polyhedral solid model into its equivalent dexel model. We assume that no gaps or overlaps exist between the surface polygons in the polyhedral model. Several polyhedral models — STL ones in particular — do not satisfy this condition. However, polyhedral models in boundary representations always satisfy this condition.

Consider an axis-aligned regular square mesh on the XY-plane that contains the projection of the object onto the XY-plane. The appropriate resolution of the mesh is determined by considering the representation accuracy of the dexel model and the memory capacity of the computer. Corresponding to each grid point, a ray perpendicular to the coordinate plane (XY plane) is extended along the Z-axis, and its points of intersection with the surface polygons of the object are computed. These intersection points are then sorted along the direction of the ray. By connecting odd-numbered intersection points with even-numbered intersection points using lines, a dexel model equivalent to the polyhedral model is obtained. If the dexel model is required along the X-axis or the Y-axis, the same computation is repeated for equivalent grids on the YZ-plane or the ZX-plane.

In order to realize the fast conversion, the acceleration of the following two processes is necessary.

1. Computation of the points of intersection between surface polygons of the polyhedron and a ray.
2. Sorting of intersection points along the ray.

Real-Time Ray-Tracing:

Ray-tracing is a computer graphics technique used to generate photo-realistic images. As depicted in Fig. 3, this method transmits a ray of light passing through each pixel of the display from the viewing point. When the ray collides with a solid shape placed in a virtual space, it is divided into two rays, one of which is reflected from the solid surface, and the other, which is refracted and advances into the solid's interior. This process is recursively repeated, and when any light source is reached, the corresponding color component is transmitted back to the viewing point to generate a three-dimensional computer image. Although this technique is known to produce high quality images, its computational cost is very high and its use in applications requiring real-time performance is difficult.

Most currently available GPUs adopt many core architectures and comprise several computational units, e.g., CUDA cores in NVIDIA's GPUs. Among the GPUs supplied by NVIDIA, products whose names include the prefix, RTX-, comprise a set of processing cores specialized for ray-tracing computations. They are called RT (ray-tracing) cores. For example, RTX-2080 GPU comprises 46 RT cores in addition to 2944 CUDA cores. By using RT cores, ray-tracing images of complex 3D virtual environments including multiple polyhedral objects can be generated in real-time for a full-HD display.

During ray-tracing, the following two operations are repeatedly executed.

- A straight line (ray) is extended through each grid point of the pixel-grid of a display. The ray can be set to be perpendicular to the display in the orthogonal projection.
- Corresponding to each ray, its point of intersection with the solid surface is computed. A part of the ray subsequently proceeds into the solid's interior, and the intersection point computation with respect to the solid's surface is recursively executed. As a result, a series of ordered intersection points are obtained along the ray.

These operations are almost identical to the ones necessary during the conversion of polyhedral models into dexel models using the ray algorithm. Therefore, fast dexelization of the polyhedral models is enabled by using RT cores.

Optix, API for Ray-Tracing Computation:

NVIDIA Corporation provides an API library named Optix for ray-tracing computations [2]. The function of an RT core is automatically available through the API function of Optix. In order to evaluate the effectiveness of the RT core in dexel processing, a dexelization software of the polyhedral model was implemented using the API of Optix, and computational experiments were performed.

Optix is implemented using C++ and it defines following three basic classes:

- **Context class:** A fundamental class for defining ray-tracing functions. Almost all events are defined as instances of this class.
- **Geometry class:** A class for defining geometric properties and operations, e.g., coordinates of polygons, a bounding box, and a function for intersection computations. We use geometry triangles class which is specialized for dealing with triangular polygons.
- **Material class:** A class that defines the reflectivity, refractive index and other material properties of a solid object.

Though various functions can be implemented via Optix, the following three kinds of functions are used in our polyhedron-to-dexel conversion software.

- **Ray-generation program:** This is a function called by the launch function that starts Optix processing. Ray-tracing processing is initiated here. Functions in this program are defined in the context class.
- **Closest hit program:** This function is called when a ray collides with the nearest polygon during the ray-tracing process. Information about intersection points that is necessary in the polyhedron-to-dexel conversion is recorded by using this function. Functions in this program are defined in the material class.
- **Miss program:** This function is called when a ray does not intersect any solids during the ray-tracing process. This function is used to terminate the ray-tracing operation. Functions in this program are defined in the context class.

To initiate Optix processing, the launch function of the context class is invoked based on the number of rays. Functions of Optix can be used in the threads invoked in the CUDA environment. In this case,

a function named *rtTrace* is called in each thread, together with the information of the polyhedral model and the corresponding ray.

Dexelization Algorithm Using Optix:

The processing flow of our dexelization algorithm using Optix is as follows (consult Fig. 4). In our implementation, the processing steps between Step 5 and Step 7 are executed by threads in a CUDA program, enabling the parallel computation of dexels for multiple rays.

Step 1: A context is generated. Optix constants and buffer for I/O are defined, followed by the definition of the ray-generation program and the miss program.

Step 2: Geometry triangles are generated. Following that, vertex data array of the triangles of the input model are transferred to the Optix buffer.

Step 3: A material class is created to define a closest hit program.

Step 4: The number of rays are specified to the launch function to initiate Optix processing.

Step 5: A set of rays is defined to be used during the Optix processing. Grid points on the XY-plane are indicated as starting points for the rays, and a direction perpendicular to the XY-plane is indicated as direction of the rays. These operations are simultaneously executed by multiple threads.

Step 6: Ray-tracing computations are performed using the *rtTrace* function for each thread.

Step 7: Information regarding the intersection points is stored using the closest hit program which is invoked whenever the intersection points are calculated (Fig. 4(a)). The starting point of the ray is then updated to be the intersection point, and Step 6 is executed again (Fig. 4(b)). If no intersection point is detected for a ray, the miss program is called and the exit flag is raised (consult the two lines at the left end in Fig. 4(c)). In this case, the Optix process is terminated for the ray.

Step 8: The obtained intersection points for the ray have already been sorted along the ray. For each ray, pairs of odd-numbered intersection point and even-numbered intersection point are checked, and then connected to obtain the dexels corresponding to the ray.

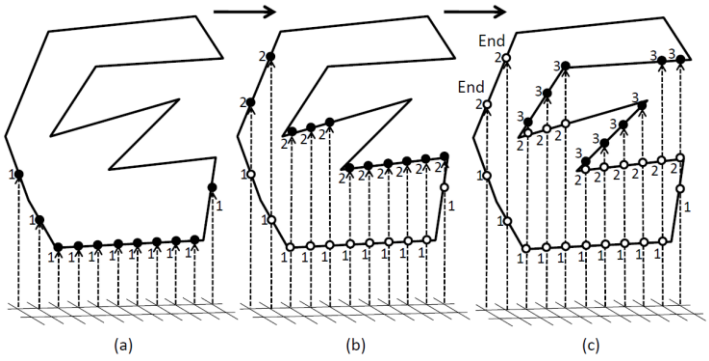


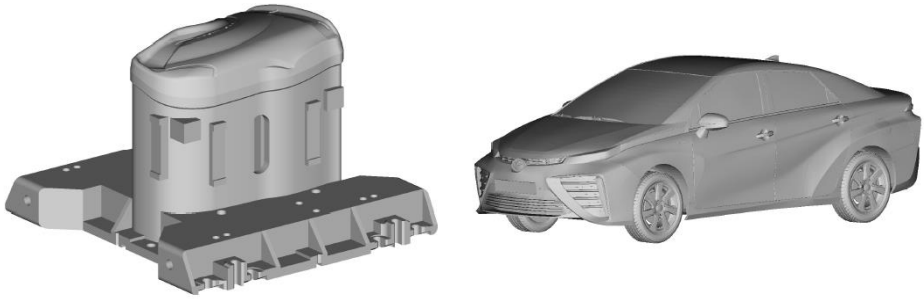
Fig. 4: Polyhedron-to-dexel conversion process using Optix.

Computational Experiments:

To evaluate the performance of polyhedron-to-dexel conversion using RT cores, Step 1 to Step 7 of the aforementioned algorithm are implemented using Optix. A polyhedral model in the STL format is taken to be the input. Our software generates a set of vertical rays starting from grid points of the square mesh placed on the XY-plane, and then computes all intersection points between the rays and the polyhedral model. The obtained points are already order along the rays during the ray-tracing process. The software was implemented using VisualStudio 2017, CUDA 10.1, Optix 6.0, and Cmake 3.14. The specifications of the PC used in the experiment are CPU: Intel Core i9-9900 (3.60 GHz), RAM: 32GB, GPU: GeForce RTX-2080.

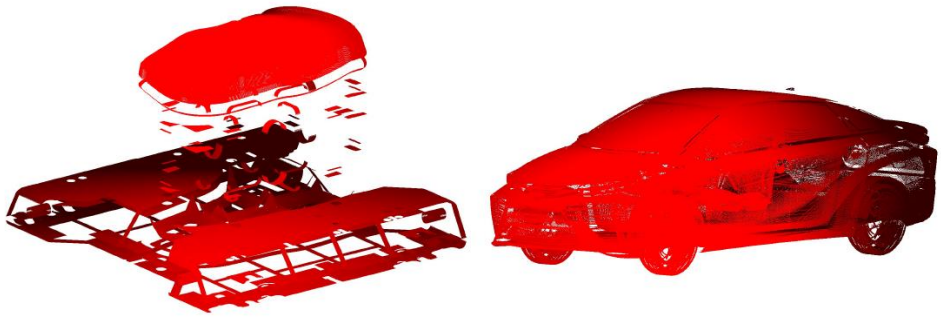
Instead of the workpiece models representing the milling result, we use two polyhedral models, model *A* and model *B* depicted in Fig. 5 in our conversion experiments for the purpose of maintaining confidentiality. Number of polygons of the models and resolutions of the square mesh for the dexel representation are illustrated in the figures. Fig. 6 shows the conversion results for the models given in Fig. 5. In this figure, ordered intersection points between the vertical rays and the models are only

illustrated. By connecting odd-numbered intersection point and even-numbered intersection point with a line segment, dexel models equivalent to the models given in Fig. 5 are obtained. The time required for the processing was 0.524 s for model *A* and 0.369 s for model *B*. We have implemented another ray-algorithm-based conversion software. This software realizes the parallel conversion using CUDA cores of GPU. In the conversion, this software needs 7.26 s for model *A* and 1.95 s for model *B*. The conversion software using RT cores is 13.9 times faster for model *A* and 5.3 times faster for model *B*.



(a) Model *A* 1,972,196 polygons (b) Model *B* 160,918 polygons

Fig. 5: Sample models.



(a) 4,000 x 4,190, 0.524 s (b) 4,000 x 1,726, 0.369 s

Fig. 6: Sorted intersection points for model *A* (a) and for model *B* (b).

Conclusions:

State-of-the-art GPUs are equipped with special hardware called RT cores that are dedicated to ray-tracing. In this paper, we proposed a novel method for fast dexelization of a complex polyhedral model using RT cores. NVIDIA Corporation provides an API library named Optix for ray-tracing computations. The function of the RT core is automatically available through the API function of Optix. In order to evaluate the effectiveness of the RT core in dexel processing, a dexelization software of polyhedral models was implemented using the API of Optix, and computational experiments were performed. At present, it is necessary to mediate the API for computer graphics using systems such as Optix to use RT cores, and several restrictions remain on programming. For instance, the number of polygons that can be simultaneously handled in Optix is limited, and it is difficult to realize a computation using objects with more than hundred million polygons. We are currently investigating the problems and limitations of using RT cores and Optix, and examining their solutions.

References:

[1] Menon, J.; Marisa, R.J.; Zagajac, J: More powerful solid modeling through ray representations, IEEE Computer Graphics and Applications, 14(3), May 1994, 22-35. https://doi.org/10.1109/38.279039

[2] NVIDIA OptiX™ Ray Tracing Engine, https://developer.nvidia.com/optix

[3] VanHook, T.: Real-time shaded milling display, Computer Graphics (Proceedings of ACM SIGGRAPH), 20(4), 1986, 15-20. https://doi.org/10.1145/15886.15887