Computer-Aided Design 49 (2014) 1–7

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

A multi-threaded algorithm for computing the largest non-colliding moving geometry

Evan Shellshear*, Sebastian Tafuri, Johan Carlson

Geometry and Motion Planning Fraunhofer-Chalmers Research Centre, Gothenburg, Västra Götaland, 41288, Sweden

HIGHLIGHTS

- Introduce a novel algorithm to compute largest volume to pass along a path.
- Introduce also a multi-threaded version of the algorithm.
- Novel use of the existing data structures that scale linearly with the problem size.
- Abstracts away the geometry so it can be used with any combination of geometries.

ARTICLE INFO

Article history:
Received 2 September 2013
Accepted 5 December 2013

Keywords:
Collision detection
Multi-threaded
Largest non-colliding volume

ABSTRACT

In this article we present an algorithm to compute the maximum size of an object, in three dimensions, that can move collision-free along a fixed trajectory through a virtual environment. This can be seen as a restricted version of the general problem of computing the maximum size of an object to move collision-free from a start position to a goal position. We compute the maximum size by dividing the object into numerous small boxes and computing which ones collide with the virtual environment during the movement along the given trajectory. The algorithm presented is optimized for multi-threaded computer architectures and also uses data structures that leave a small memory footprint making it suitable for use with large virtual environments (defined by, e.g., millions or billions of points or triangles).

© 2013 Elsevier Ltd. All rights reserved.

1. Introduction

Being able to determine whether a virtual object can pass through a virtual environment without collision is a fundamental problem in virtual design. Depending on the exact problem at hand, there exist hundreds of papers addressing this problem from all sorts of aspects. In this paper we focus on a lesser studied problem of utmost importance for industrial designers. We are interested in being able to compute the largest object that can travel collision-free from a start configuration to a goal configuration. By the largest object we mean the object that has the largest volume according to some user defined way of measuring volume (as explained below). In this paper, we confine the object to travel along a fixed trajectory through a fixed virtual environment.

This academic investigation is motivated by numerous real life problems (see below) and the basic motivation for this article is the virtual verification of car designs. In particular, one wants to know if a new car design can pass through an assembly line without colliding with other objects and if it does collide, what are the minimal design changes that need to be made to avoid the collisions. This information can also be used for future design problems if the environment and trajectory remain the same (as is typical for factory installations).

Another motivation for this study was to improve the sustainability of current technologies. In particular, the results of this paper are intended to allow companies to reuse the existing facilities by giving designers a guarantee that their new designs fit in the current industrial installation and also letting them know the minimum changes to both the object they are designing and/or the industrial environment if necessary. This virtual verification saves the designer significant time much earlier in the design phase. Currently to determine whether a new model design fits along an assembly line or not, one is required to build a physical mock-up of a car (or whatever is being produced) and run it through an assembly line to check for collisions. Such a process is obviously much slower and costlier than having a simple virtual tool to directly test a new design and suggest which minimal changes need to be made.

Significant design changes are often very costly, hence the goal of many designers is to know what the minimal structural design changes are to make sure objects do not collide with each other. Typical applications where computing the largest non-colliding designs plays an important role include, *inter alia*, the ability to route and path plan objects through tight spaces such as engines [1] and other assembly components, designing robots for tight spaces

* Corresponding author.
E-mail address: evan.shellshear@fcc.chalmers.se (E. Shellshear).

0010-4485/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cad.2013.12.001

in path planning applications [2,3], etc. Once the set of colliding parts has been discovered one can then apply any of the numerous algorithms to effect the necessary design changes to avoid collisions (e.g. [4,5]).

The problem we are interested in can be defined more exactly as follows. Let A be some geometrical object (set of NURBs, polygon soup, points, etc.) that bounds some volume of space and B another geometrical object (representing the surrounding environment). We wish to compute the remaining non-colliding space bounded by A after A has traveled along a given trajectory (motion, path, etc.) defined by a set of rigid transformations. We would like to remove all parts of the space bound by A that collide with any part of B during the set of rigid transformations. From the design perspective, the remaining non-colliding volume can be used to alter the original geometry so that it can pass along the original path collision-free. In addition, we are also interested in determining how much larger A can be and still have parts of the enlarged volume not colliding with any part of B.

To compute the largest remaining non-colliding volume to travel along a given path, we developed an algorithm as follows. Our algorithm first allows the user to specify the largest volume object they are interested in, say A. We then compute a bounding box around A and divide the box into smaller subboxes with fixed side lengths defined by the user (so-called tolerance). This size represents the level of error acceptable to the user. An hash-based octree containing the surrounding environment is then created and used to compute collisions between the environment and the object during its motion. As collisions are detected between the subboxes and the octree, subboxes are removed. The final remaining set of subboxes left at the end of the motion represents the largest shape that can pass through the environment along the path with the given user tolerance. The algorithms used here are located at the confluence of algorithms for finding the maximum empty subset of a given domain [6], collision detection [7] and being able to compute boolean operations for arbitrary geometries [8].

The remainder of this paper is organized as follows. In the next section, we present our algorithm and its analysis as well as the existing literature relevant to each part of the algorithm. We then present our experimental results, which are followed by a discussion of the results from the previous section. In the final section we conclude.

2. Methods

In this section, we describe an algorithm that solves the problem stated in the introduction. When designing the algorithm we wish to bear a few goals in mind. First of all, if the algorithm is to be run in the main memory, then it needs to have minimal space requirements because it will be used for point clouds with millions (or billions) of points. In spite of this, we do not want an unnecessarily slow algorithm for computing the final non-colliding geometry because the algorithm will be used for design purposes. When designing either the trajectory or the geometry one often requires the ability to make small changes and then rerun the simulation to see the effect of the changes. Hence, the given algorithm should also be fast.

To achieve these goals we chose to solve the problem in the following way. To represent the moving object we computed its bounding box and subdivided it into a number of smaller subboxes. The size of the subboxes were defined by a tolerance chosen by the user. Simultaneously we chose to represent the cloud via a hash-based octree, [9], whereby the user also specifies a minimum tolerance level which defines the octree's leaf cells' side lengths. Allowing the user such a choice is natural because for scanned point clouds one has a level of error in the scan results as well as a certain minimum distance between each scanned point. However, the biggest advantage of representing the geometries as such, is that it could be used to represent other types of geometries than merely points. One could also build a similar octree based on triangles or other geometric primitives and simply mark cells as occupied if they contain a point or part of a triangle. We describe now these choices in more detail as well as a justification for the choices.

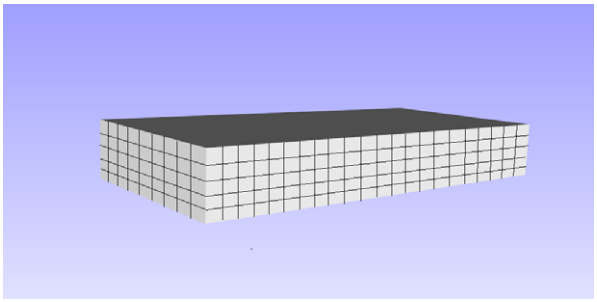

Fig. 1. The subdivision of the bounding box into subboxes.

As stated in the introduction, it is useful for the designer to be able to not just know if the original object can pass along the trajectory collision free but also the extra space around it. This extra volume can be used to compute the maximum amount of clearance a certain part of the object has with its surroundings when traveling along the trajectory. This can be easily facilitated by our choice of using the bounding box of the object to represent the object. To find the maximum clearance around the object, one can increase the size of the bounding box in each dimension which can then be used to find the largest set of boxes that can move along the trajectory collision free.

So, we first find the bounding box of the object that is to pass along the trajectory. We then divide the box up into smaller subboxes with equal side lengths, equal to a user-defined tolerance, see Fig. 1. We then move the collection of boxes along the trajectory and check whether the boxes collide with the surrounding environment or not with help of the environment's octree representation. The interpretation of this representation is that the point cloud represents a solid object and if a point (represented by an octree leaf cell) intersects one of the subboxes then some solid part of the environment would intersect the subbox too. This interpretation also puts a minimum bound on the octree leaf cell size, which should not be smaller than the average spacing between points so that we do not miss intersections caused by solid parts of the environment.

Apart from using an octree to facilitate collision detection, one could carry out the collision detection using other methods. One possibility is to compute a swept volume of the object for the whole path and then check for collision, [10]. However, such methods have been shown to be very time consuming, often taking minutes for short paths and small meshes [11]. We intend to use our algorithm to compute the motion of an arbitrarily complex object over hundreds of meters, which would mean any swept volume computation would be completely impractical. Hence, the use of swept volumes in our case is inappropriate. Other possibilities are to calculate the minimum distance of each subbox to all points along the entire path. This can be efficiently done by conservative advancement [12,13] and is similar to the way we have chosen to solve the problem, however, for non-convex geometries the algorithms we use are much simpler and also avoid the extra memory overhead that [13] imposes (which is too large for point clouds with millions or billions of points). In addition, to use the conservative advancement for non-convex geometries, as developed in [13], requires one to build a bounding volume hierarchy around the point cloud (as done in, e.g., [14]) which can be impractical for an algorithm to be run in the main memory for point clouds with billions of points.

In addition to providing fast collision queries via our octree, we also naturally wish to provide a guarantee that all possible collisions will be found. To do so, we use results from [15] to repeatedly move each subbox as far as possible along the trajectory in a

Download English Version:

https://daneshyari.com/en/article/439516

Download Persian Version:

https://daneshyari.com/article/439516

Daneshyari.com