ELSEVIER

Computer-Aided Design 38 (2006) 1–13

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

# A C-tree decomposition algorithm for 2D and 3D geometric constraint solving[☆]

Xiao-Shan Gao[a,*], Qiang Lin, Gui-Fang Zhang[b]

[a]*Key Laboratory of Mathematics Mechanization, Institute of Systems Science, AMSS, Academia Sinica, Beijing 100080, China*
[b]*Department of Computer Science and Technology, Tsinghua University, Beijing, China*

Received 14 June 2004; received in revised form 4 March 2005; accepted 10 March 2005

**Abstract**

In this paper, we propose a method which can be used to decompose a 2D or 3D constraint problem into a C-tree. With this decomposition, a geometric constraint problem can be reduced into basic merge patterns, which are the smallest problems we need to solve in order to solve the original problem in certain sense. Based on the C-tree decomposition algorithm, we implemented a software package MMP/Geometer. Experimental results show that MMP/Geometer finds the smallest decomposition for all the testing examples efficiently.
© 2005 Elsevier Ltd. All rights reserved.

*Keywords:* Geometric constraint solving; Parametric CAD; General construction sequence; Basic merge pattern; Decomposition tree; Graph algorithm

## 1. Introduction

Geometric constraint solving (GCS) is one of the key techniques in parametric CAD, which allows the user to make modifications to existing designs by changing parametric values. GCS methods may also be used in other fields like molecular modelling, robotics and computer vision. There are four major approaches to GCS: the numerical approach [14,26,31], the symbolic approach [15,23,25,33], the rule-based approach [2,24,34,35] and the graph-based approach [6,7,12,17,20,28,30]. This paper will focus on using graph algorithms to decompose large constraint problems into smaller ones.

In [32], Owen proposed a GCS method based on the tri-connected decomposition of graphs, which may be used to reduce a class of constraint problems into constraint problems consisting of three primitives. In [7,16], Hoffmann et al. proposed a method based on cluster formation to solve 2D and 3D constraint problems. An algorithm was introduced by Joan-Arinyo et al. in [21] to decompose a 2D constraint problem into an s-tree. This method is equivalent to the methods of Owen and Hoffmann, but is conceptually simpler.

The above approaches use special constraint problems, i.e. triangles, as basic patterns to solve geometric constraint problems. In [28], Latham and Middleditch proposed a connectivity analysis algorithm which could be used to decompose a constraint problem into what we called the general construction sequence (defined in Section 2). A similar method based on maximal matching of bipartite graphs was proposed by Lamure and Michelucci [27]. In [17], Hoffmann et al. gave an algorithm to find rigid bodies in a constraint problem. Based on this, a general approach to GCS was proposed [18]. In [19], Jermann et al. also gave a general approach to GCS based on the idea in [17].

In this paper, we propose a method which can be used to decompose a general 2D or 3D constraint problem into a C-tree (connectivity tree). The algorithm is inspired by two facts. First, the general construction sequence obtained with Latham–Middleditch's algorithm reduces the original constraint problem into smaller ones. But, in many cases these smaller problems could be further simplified. Second, we observed that not all rigid bodies in a constraint problem can be used to split the original problem. We introduced the key concept of *faithful subgraph*, which may lead to a split of the constraint problem. The C-tree decomposition algorithm

[☆] Partially supported by a National Key Basic Research Project of China (2004CB318000) and by a USA NSF grant CCR-0201253.

[*] Corresponding author. Tel.: +81 10 6254 1831; fax: +81 10 6263 0706.

*E-mail address:* xgao@mmrc.iss.ac.cn (X.-S. Gao).

0010-4485//$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2005.03.002

consists of two main steps: using the general construction sequence to find faithful subgraphs and using the faithful subgraph to split the constraint problem into two subproblems. The complexity of the algorithm is $O(n^2(n+e)e)$, where $n$ and $e$ are the numbers of geometric objects and constraints, respectively. Major advantage of the algorithm is that it can be used to decompose a general constraint problem into certain kind of smallest problems and it leads to a simple and efficient implementation.

A C-tree is a binary tree. For each node in the tree, its left child is a rigid body which will be solved first. After the left child is solved, we may use the information from the left child to solve the right child and to merge the left and right children to solve the constraint problem represented by the node. All leaves of the C-tree are general construction sequences. Therefore, solution of a constraint problem is reduced to the solution of general construction sequences with a C-tree decomposition.

We show that the solution of a general construction sequence can be reduced to the solution of *basic merge patterns*, which are the smallest problems we need to solve in order to solve the original problem in certain sense. We give a classification of the basic merge patterns both in 2D and 3D cases and show that some of the basic merge patterns have closed-form solutions.

We say that a graph decomposition method for GCS is a general method if it can be used to handle all constraint problems. Among the general GCS methods [18,19,24,27,28], the method MFA proposed in [18] and the C-tree method can be used to find a smallest decomposition in certain sense. The MFA and C-tree methods have the same complexity. Both can be used to solve 2D and 3D problems, although paper [18] focuses on the 2D case and this paper focuses on both 2D and 3D cases. Comparing to the algorithm in [18], our algorithm is simpler and easy to implement.

Based on the C-tree decomposition algorithm and several other algorithms proposed by us, we implemented a GCS software package MMP/Geometer in the Windows environment with VC++. Experimental results show that the software package finds the smallest decomposition for all the testing examples efficiently.

The rest of the paper is organized as follows. In Section 2, we introduce the methods to generate general construction sequences. In Section 3, we give the algorithm to generate the C-tree. In Section 4, we give a classification of the basic merge patterns. In Section 5, we report the experimental results for our implementation of the C-tree decomposition algorithm. In Section 6, we present the conclusion.

## 2. General construction sequence

### 2.1. Basic concepts

In the two-dimensional Euclidean plane, we consider two types of *geometric primitives*: points and lines and two types

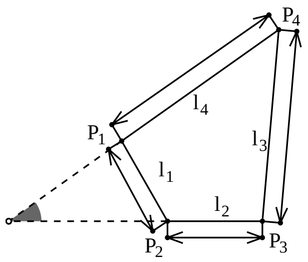

Fig. 1. A 2D problem: lengths of four edges and ANG ($l_2$,$l_4$) are given.

of *geometric constraints*: the distance constraint between point/point, point/line and the angular constraint between line/line. In the three-dimensional Euclidean space, we consider three types of geometric primitives: points, planes and lines and two types of geometric constraints: the distance constraints between point/point, point/line, point/plane, line/line and the angular constraints between line/line, line/plane, plane/plane. A *geometric constraint problem* consists of a set of geometric primitives and a set of geometric constraints among these primitives. Angular and distance constraints between two primitives $o_1$ and $o_2$ are denoted by ANG($o_1$, $o_2$) $=\alpha$ and DIS($o_1$,$o_2$) $=\delta$, respectively. We will use $p_i$, $h_i$ and $l_i$ to represent points, planes and lines, respectively.

We use a *constraint graph* to represent a constraint problem. The vertices of the graph represent the geometric primitives and the edges represent the constraints. For a constraint graph $G$, we use $\mathbf{V}(G)$ and $\mathbf{E}(G)$ to denote its sets of vertices and edges, respectively. Fig. 2 is the graph representation for the constraint problem in Fig. 1.

For an edge $e$ in a constraint graph, let DOC($e$) be the valence of $e$, which is the number of scalar equations required to define the constraint represented by $e$. Most constraints considered by us have valence 1. There are several exceptions: (1) Constraint DIS($p_1$,$p_2$) $=0$. In this case, $p_1=p_2$. In 2D case, the constraint has valence 2; in 3D, the constraint has valence 3. We assume that this case does no occur. (2) Constraint DIS($p_1$,$l_1$) $=0$ has valence 2 in 3D. (3) Constraint ANG ($h_1$,$h_2$) $=0$ has valence 2 in 3D. (4) Constraint ANG ($l_1$,$l_2$) $=0$ has valence 2 in 3D. These constraints are *degenerate cases*.

For a geometric primitive $o$, let DOF($o$) be the degrees of freedom for $o$, which is the number of independent parameters required to determine the geometric

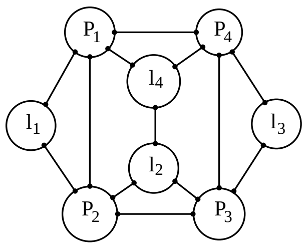

Fig. 2. Graph representation for the problem in Fig. 1.