Constraint-Based 3D-Object Layout using a Genetic Algorithm

Stéphane Sanchez Olivier Le Roux Hervé Luga Véronique Gaildrat

Department of Computer Science IRIT
UPS, 118 route de Narbonne, 31062 Toulouse Cedex 4, France
Phone. (33) 05 61 55 83 29 Fax. (33) 05 61 55 62 58
E-mail: { sanchez, leroux, luga, gaildrat }@irit.fr

Abstract

Object layout in a large 3D-scene or in a complex virtual environment is a time-consuming and tedious task. In order to assist the user in this task, we present a general-purposed constraint-based system. The underlying constraint solver is built on a genetic algorithm. It is able to process a complex set of constraints, including geometric and pseudo-physics ones. Moreover, universal quantifiers and Boolean combination of constraints are allowed to make the layout description easier. Finally, to get realistic scenes, objects can take any orientations, not only isothetic ones.

Keywords: non-isothetic 3D-object layout, genetic algorithm, constraint solving.

1 Introduction

Realistic object layout in a large 3D-scene is currently a time-consuming task. In this paper we study and experiment a genetic algorithm as core of a constraint-based generation system. Once validated, this tool will be included in our declarative modelling platform DEM²ONS [1] [Kwaiter et al. 97].

By now, most of the placement tools take into account only isothetic [2] objects. This simplification allows minimizing size of the search space and thus computation time. Excepting some systems (e.g. [Xu et al. 02]) that choose to drastically simplify the allowed constraints, no one is interactive. In fact, the classical problem is known to be NP-complete (i.e. there are no complete tractable algorithms to solve it). In complex cases, only stochastic heuristics give suitable results. However, in a classical 3D environment, the problem is widely under-constrained and some systems generate interesting results using standard CSP[3] algorithms [Charman 95] [Kwaiter et al. 97] [Bonnefoi et al. 99] [Ruchaud et al. 02].

Unfortunately, using such family of algorithms leads to difficulties to obtain realistic scenes because of exhaustive enumeration; hard constraints are usually added to get a satisfying solution. Another approach is to consider the problem as a numeric optimization one. The formulation can be linear [Vries et al. 00] or non-linear [Donikian et al. 93]. However, strict restrictions (especially on NLP[4]) must be set on the constraint system to ensure satisfying results.
A third approach consists in using metaheuristics such as local search (simulated annealing, tabu search) or evolutionist strategies [Masui 92] [Vassilas et al. 02].

In the following we present some reasons to choose genetic algorithm for our problem:

- huge search space (each object has seven degrees of freedom);
- some constraints, in their algebraic formulation, are not differentiable (e.g. the non-overlapping constraint);
- object layout must be as realistic as possible: a stochastic process is well suited;
- universal quantifiers and Boolean combination of constraints must be used to improve expressiveness in description of layout problem;

[1] DEM²ONS = Declarative Multimodal ModeliNg System
[2] isothetic = parallel and/or orthogonal to world coordinate system
[3] CSP = Constraint Satisfaction Problem
[4] NLP = Non Linear Programming

- some descriptions involve hard problems including cyclic constraint graph;
- finally, our object layout system has to manage optimization problems (like “add two more chairs around the table”).

This paper is organized as follows. Part 2 introduces our approach. We describe the inner architecture of the system: objects, domains and constraint modeling. The genetic algorithm is also presented. In part 3, we give some experimental results on classical problems. Part 4 discusses on advantages and drawbacks of our approach. We also compare our system with related ones. Finally part 5 concludes this study.

2 Our constraint-based system

2.1 A brief overview of Genetic Algorithm

In 1975, J. H. Holland introduced genetic algorithms as a new technique that can be applied to solve a wide variety of problems [Holland 75] and D. Goldberg presented first real applications that made them popular in 1989 [Goldberg 89].
J. R. Koza [Koza 92] has given a good definition of genetic algorithm:
"The genetic algorithm is a highly parallel technique that transforms a population of individual objects, each with an associated fitness value, into a new generation of the population using the Darwinian principle of reproduction and survival of the fittest and naturally occurring genetic operations such as crossover (recombination) and mutation. Each individual in the population represents a possible solution to a given problem."

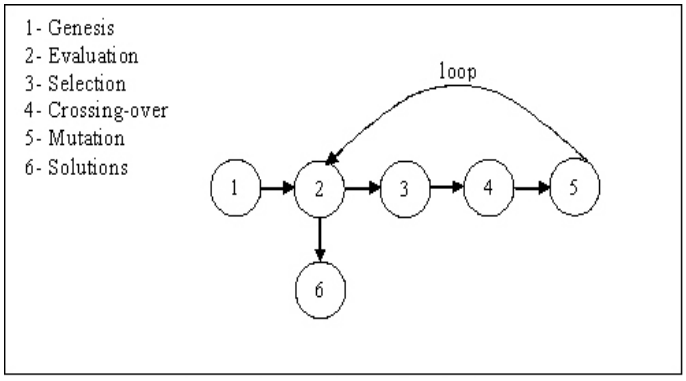

Figure 1: Main loop of a genetic algorithm

2.2 Solver requirements

To be used as a valuable generation tool the solver must have some features.
It must solve topological constraints (“the plate is on the table”), metrics constraints (“the chair is at one meter of the desk”), angular constraints (“dispatch uniformly three chairs around the table”), orientation constraints (“the sofa is in front of the TV”), and size constraints (“the table is 1.80 meter long”).
The resulting scene must be realistic: 3D-objects with any orientations, no overlapping and respect of gravity constraints.
Constraint's hierarchy is used to express user’s preferences and to solve over-constrained problems.
The solver must be able to treat cyclic constraint graph.
Disjunctive constraints (“set the chair against *A* wall”, “set the waste basket to the right *OR* the left of the desk”) and negative ones (“the wardrobe is *NOT* in front of the window”) must be managed.

2.3 Problem modeling

In this section, we present our problem modeling. This stage is decisive and determines the performance of the system.

In our evolutionist approach:
- placement space (i.e. domains) and objects already in the scene represent the environment of the population of genetic algorithm;
- objects to set represent genetic components of individuals;
- combination of constraints represents the pressure to apply on individuals in order to make the genetic algorithm converge toward a suitable solution.

2.3.1 Domains and objects

The placement space, in which new objects are added, is a limited three-dimensional space, oriented around the height axis.

To be realistic, objects in the scene are identified with a hierarchical set of bounding boxes. Usually, when an object is put on shelves, it must be on an inner shelf and not on top of them.

Each bounding box is defined using three kinds of parameters:

- *position*: location of the box in the world coordinate space. A variable of position is made up of three one-dimensional values.
- *orientation*: an angle in radians. Usually the three Euler angles (roll, pitch, head or yaw) are used. To minimize computation time, we choose to maintain only head (rotation around the height axis). This choice is justified by usual object orientation in real world.
- *size*: a one-dimensional parameter along an axis. Three size parameters are used to define the dimensions of a box.

Remark: to lower processing time and to achieve correct convergence of genetic algorithm, the objects to set in the scene are defined only by their global bounding box. Consequently, if the user wants to set a table in the scene and a stool under it, he must first set the table in place and then set the stool relatively to hierarchical structure of the table. Nevertheless, ten chairs and associated plates and forks can be set around the table at one time.

2.3.2 The constraints

The constraints used, mainly inspired of the ones proposed by P. Charman [Charman 95], are low-level ones: they each describe a local layout problem that the solver can easily process in order to generate a suitable solution. However, these constraints can be combined to create high-level properties.

We consider two kinds of constraints: the first ones, we name *physical constraints*, ensure coherence and realism of the generated scene. The others, called *geometric constraints*, will be combined to express the problem.

The *physical constraints* consist in two main constraints:

- *non-overlapping* avoids collisions between objects ;
- *gravity* invalidates every object that is not on the ground or balanced on another valid object.

The *geometric constraints* aim to describe each a simple and elementary spatial problem. They usually use the Vandeloise's terminology: (with exception of *size* constraint as seen below):

object $_{target}$ **S_P (*parameters*) object** $_{landmark}$

Where **object** $_{target}$ is the object the user intends to put in the scene relatively to **object** $_{landmark}$ in relation with spatial preposition **S_P**.

The user can give any geometric constraint either in a positive way or in a negative one (operator NOT).

Four kinds of *geometric* constraints are considered:

- *zone*: “the chair is on the *LEFT* of the table”, “put the vase *ON* the shelf”;
- *orientation*: “the sofa is *FACING* the TV”.
- *distance*: “the monitor is *AT* 50 CM of the desk chair”;
- *size*: “the table is 2 METERS *LONG*”.

And each one is defined by a set of basic mathematic and/or geometric properties.

For example, a *zone* constraint is described as a volume made of a control polygon as base and a set of allowed heights:

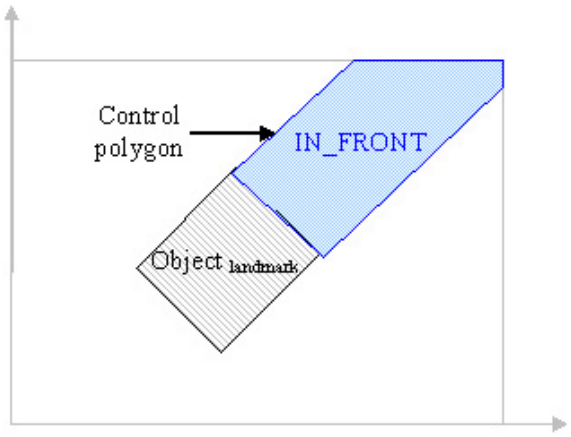

Figure 2: Control polygon of *zone* constraint

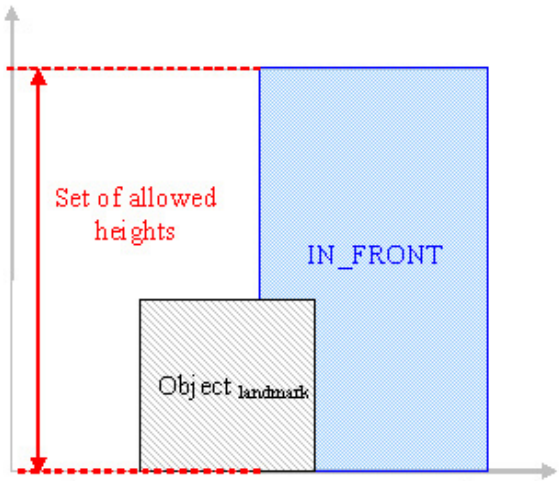

Figure 3: Set of allowed heights of *zone* constraint

2.4 Solving the problem

The layout object problem to solve consists of two associated parts:
- a genome that represents the objects to set in the scene,
- associated with a combination of constraints.

2.4.1 The genome

An individual of the genetic population represents a potential solution to the problem. The evaluation of its genome (using a fitness function) allows estimating the quality of the adaptation of the individual to the current problem.
The genome consists of at least one chromosome. Each chromosome corresponds to a set of equivalent objects (i.e. objects related to the same *size* constraint) and it is associated to one combination of constraints. Each gene of the chromosome represents a unique object to set in the scene.

2.4.2 Setting the object layout problem

As said, each problem can be described with a combination of *geometric constraints*. The combination is made up of a Boolean tree with constraints as leaves and using the following node operators: AND, OR, XOR and NOT.
Using a Boolean tree with these four operators allows many combinations of heterogeneous constraints and a really natural way of expressing problems. We could note that a tree structure facilitates introduction of a priority factor, at leaf level, in order to create a constraints hierarchy.

- "sofa facing TV at 2 meters of the TV" (cf. Figure 4):
  C1 = Sofa ZONE(*in_front*) TV
  C2 = Sofa ORIENTATION(*pi*) TV
  C3 = Sofa DISTANCE (2.0)TV

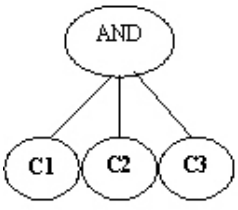

Figure 4: Constraint tree #1

- "the plate on the table facing one of the two chairs but not on the vase" (cf. Figure 5):
  C1 = Plate ZONE(*on*) table
  C2 = Plate ORIENTATION(*pi*) chair#1
  C3 = Plate ZONE(*in_front*) chair#1
  C4 = Plate ORIENTATION(*pi*) chair#2
  C5 = Plate ZONE(*in_front*) chair#2
  C6 = Plate NOT ZONE(*ON*) vase

2.4.3 Evaluation of individuals

This step is critical to the solving process. Indeed, convergence of the genetic algorithm towards a valuable solution depends only on the quality of the evaluation process: the fitness function applied to individuals must express, the most exactly, how they satisfy the set of constraints they are associated with. Besides, results of fitness functions associated with the different *low-level* geometric constraints must be homogeneous in order to combine them (using Boolean tree) without introducing an unwanted hierarchy during the evaluation process.

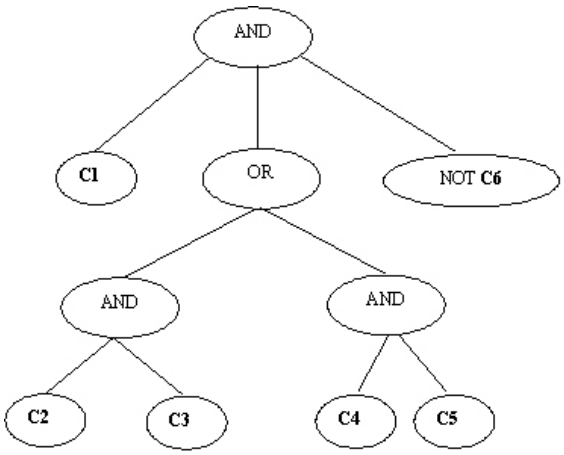

Figure 5: Constraint tree #2

The evaluation of both *physical* constraints (*non-overlapping* and *gravity*) validates, or invalidates, a gene of the individual (i.e. an object to set in the scene). Each fitness function, matching with one geometric constraint, is applied to each valid gene and uses a method that returns a percentage of satisfaction (in order to maintain homogeneity). For example, in case of positive *zone* constraint the evaluating process will be:

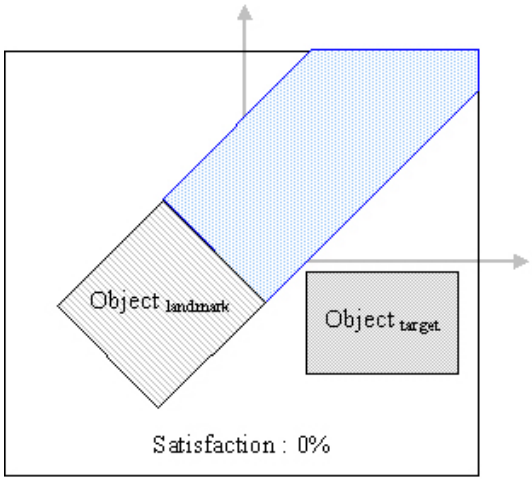

Figure 6: Fitness of *zone* constraint

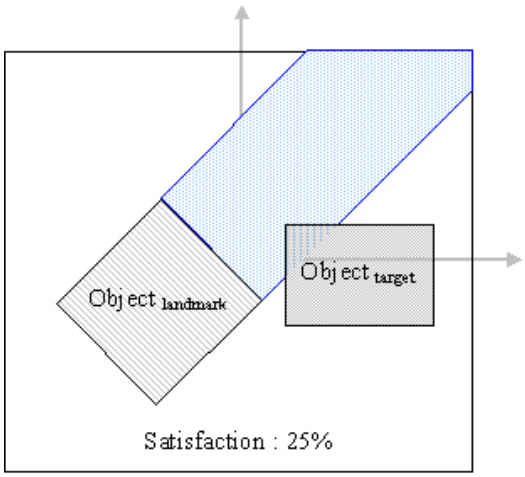

Figure 7: Fitness of *zone* constraint

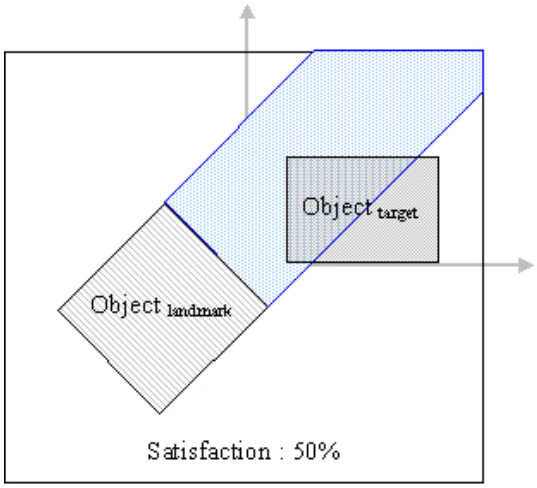

Figure 8: Fitness of *zone* constraint

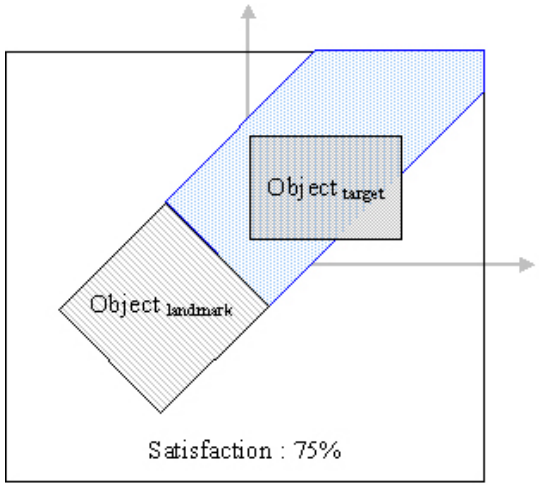

Figure 9: Fitness of *zone* constraint

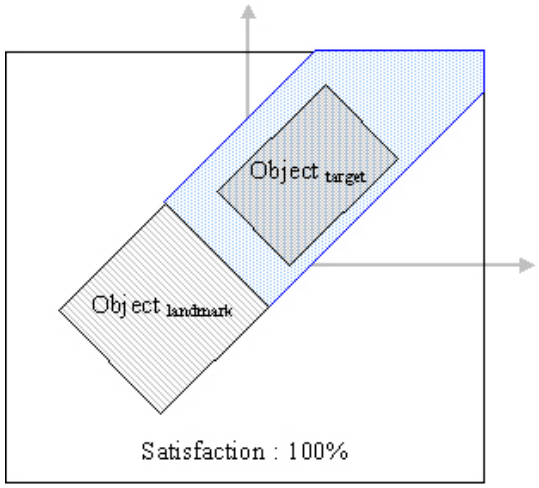

Figure 10: Fitness of *zone* constraint

The results of fitness functions are processed according to the node operators and to the priority factors so as to obtain the satisfaction value of each gene. For example, the fitness value f_g of the gene corresponding to the plate in the following problem, "plate on the table facing one of the two chairs but not on the vase", will be:

$$
\begin{aligned}
f_g = AND(\quad & prio_{C1} \times C1, \\
& OR(\quad AND(\quad prio_{C2} \times C2, \\
& \qquad\qquad prio_{C3} \times C3 \quad), \\
& \qquad AND(\quad prio_{C4} \times C4, \\
& \qquad\qquad prio_{C5} \times C5 \quad) \quad), \\
& prio_{C6} \times NOT(C6) \quad)
\end{aligned}
$$

With Ci the fitness value of gene g for constraint Ci,
$prio_{Ci}$ the priority factor of constraint Ci

Then, the fitness value f_{ch} of each chromosome of the individual will be a simple average value:

$$f_{ch} = \frac{\sum f_g}{nb_{Genes}}$$

At last, the final fitness value F is obtained with an average of fitness values of chromosome or, if we need to add a hierarchy between simultaneous problems, with a weighted sum:

$$F = \frac{\sum prio_{ch} \times f_{ch}}{\sum_{nb_{chromosomes}} prio_{ch}}$$

3 Experimental results

In order to evaluate how the solver computes different kinds of problem, we use the following 3D-scene as reference:

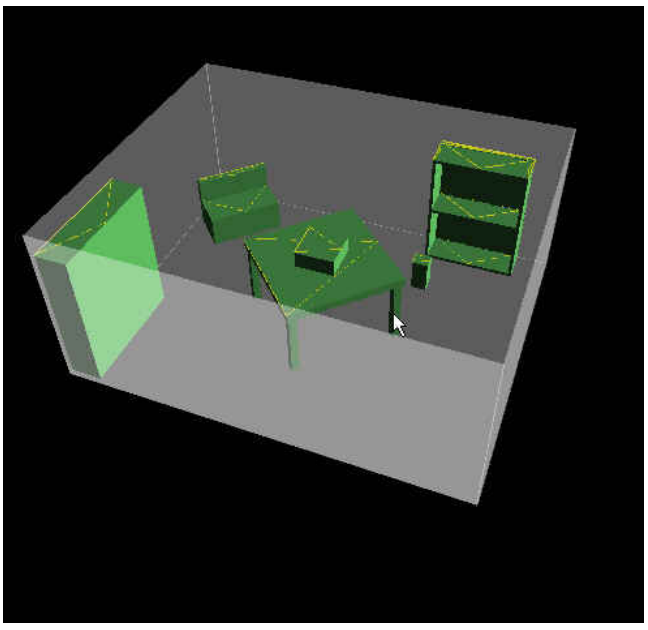

Figure 11: The reference scene

For information, the solver has built this scene, composed of a wardrobe, a table with a box on it, a stool, a sofa and shelves, in about 18 seconds.

The reference computer is a **Pentium III 1.2 GHz**. The solver is compiled with **Java SDK 1.4**. All the computation times given for the following problems are average values of 10 consecutive runs.

- "20 boxes on the table but not on a box already on the table":

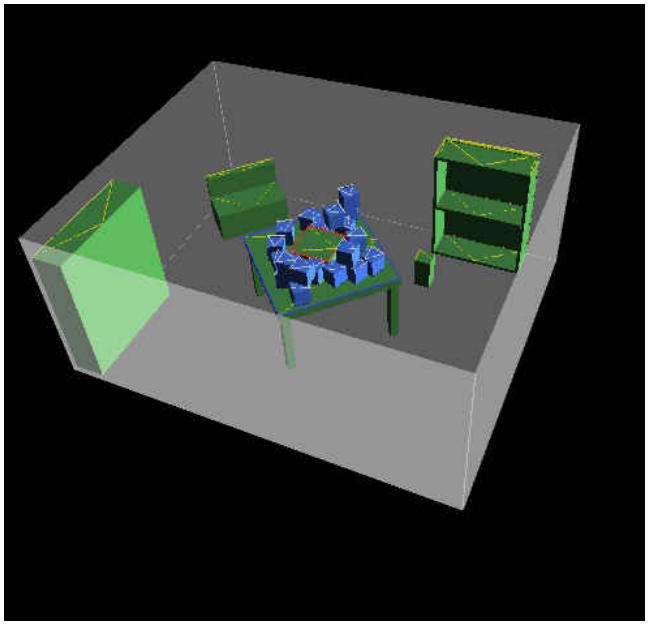

Figure 12: Computation time = 52.1 s

- "20 books correctly arranged on first shelf of shelves ":

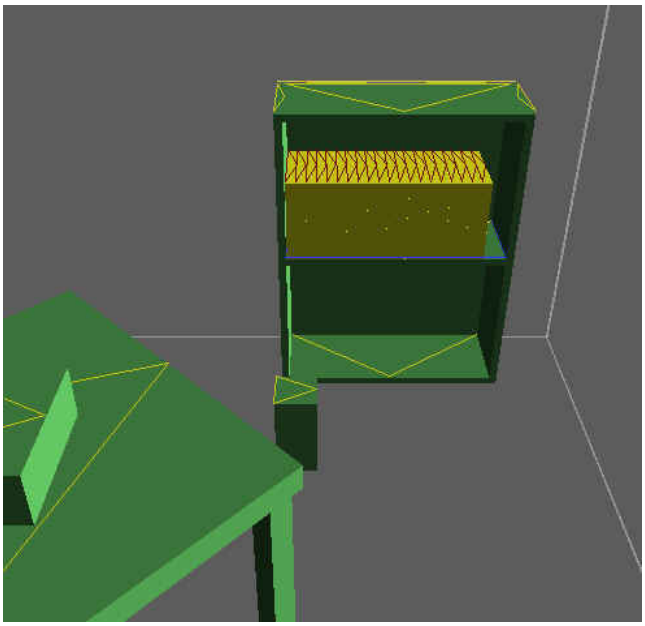

Figure 13: Computation time = 58.4 s

- "8 chairs around the table, more or less facing it":

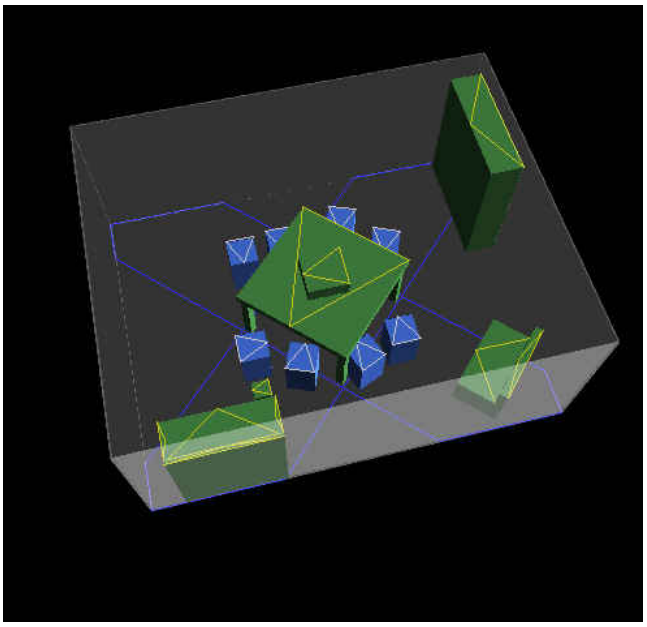

Figure 14: Computation time = 12.7 s

- "4 objects around the box on the table, cyclically oriented at 90° from the previous one":

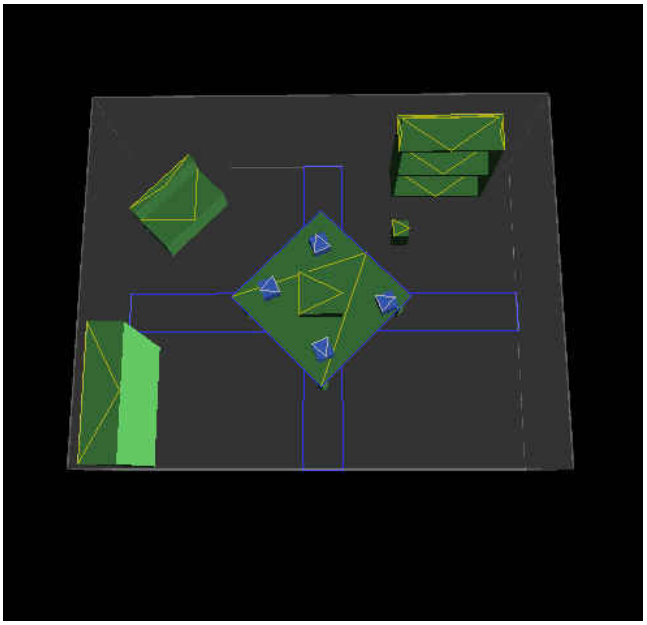

Figure 15: Computation time = 2.1 s

4 Discussion

Our set of constraints allows describing many basic problems {on, against, near, facing, above, in, parallel to, etc.}) but also more complex ones using Boolean trees of constraints.

Furthermore, in our tests:
- All the under-constrained problems have been solved.
- For each of the over-constrained problems, the solver has generated a suitable solution.
- The only problems that are not solved (no convergence of the genetic algorithm) are the ones with no possible solution (for example, "the chair in front of *and* behind the table").
- All the cyclic problems have been successfully processed.
- Negative problems are solved just as positive ones: the solver makes no difference.
- Constraints hierarchies have been correctly solved.

Nevertheless, many improvements are still possible:
- The computation times are erratic from one problem to another. They do not seem to be directly correlated with number of objects and/or number of constraints. A study on convergence of the algorithm seems necessary.
- Evaluation of individuals using a weighted sum does not allow the solver to generate a diversified set of solutions. The use of multi-criteria methods, as MOGA [Fonseca et al. 95], might correct, at least partially, this drawback.

Currently, there is no standard benchmark on non-isothetic 3D-object layout problems. Consequently, results produced by our system cannot be easily compared with related works. To give some useful informations, the next table provides a qualitative comparison of some similar projects.

Project	EAAS	DEM²ONS ORANOS	MultiFormes 4	MultiFormes 5	MultiCAD GA	DEM²ONS GA
Year	1993-1995	1997-1998	1999	2002	2002	2000-2003
Main reference	[Charman 95]	[Kwaiter 97]	[Bonnefoi 99]	[Ruchaud 02]	[Vassilas et al. 02]	This paper
Approach	constructive	constructive	constructive	constructive	iterative	iterative
Is result guaranteed?	YES	NO (discretization)	YES	NO (discretization)	NO (stochastic)	NO (stochastic)
Is algorithm sound?	YES	YES	YES	YES	NO	NO
Objects / degrees of freedom	rectangles 4	3D bounding boxes 9	3D bounding boxes 6	3D vertices 3	3D bounding boxes 6	3D bounding boxes 7
Orientation	0° or 90°	isothetic only	isothetic only	any	isothetic only	any around the height axis
Boolean combination of constraints	AND	AND	AND	AND	AND	A tree using AND, OR, XOR, and NOT
Hierarchical constraints	NO	YES	NO	NO	NO	YES
Relaxation if over-constrained?	NO	Partial	NO	NO	YES, implicit	YES, implicit
Algorithm	**S-CSP** FC or RFL + CBJ (optional) + filtering AC-SG (derived from AC3) + dynamic heuristic (FFP based)	**DHN-CSP** dynamic and hierarchical numerical pre-filtering + incremental FC or RFL + dynamic heuristic (FFP based)	**CSP** incremental FC or PL (multi-process) + BJ + AC5-filtering + dynamic heuristic (FFP based)	**CSP** BT or FC + dynamic heuristic (FFP based)	**GA** About 20 individuals constraint programming (to generate the initial population) + classical genetic algorithm	**GA** About 300 individuals domain pre-filtering + classical genetic algorithm (see section 2)

CSP = Constraint Satisfaction Problem; S = Spatial; DHN=Dynamic Hierarchical Numerical; BT = Backtracking; FC = Forward Checking; RFL = Real-Full-Lookahead; PL = Partial Lookahead; BJ = Backjumping; CBJ = Conflict directed Backjumping; FFP = Fail First Principle; AC = Arc Consistency.

Table 1: Comparative synthesis of our study with related works

Here are summarized advantages and drawbacks of our system.

Advantages:

- the system is general-purpose, easy to understand and easy to extend;
- maximum computation time allowed can be defined by the user;
- unlike traditional constructive approaches (e.g. CSP), a partial solution can be visualized and selected at any moment;
- as it is a stochastic process, final scene is very realistic;
- very complex problems can be formulated: in our system, universal quantifier and Boolean combination of constraints are allowed;
- fuzzy description are naturally taken into account;
- when the system encounters an over-constrained problem, a suitable solution is proposed according to the weight of constraints;
- the system can both deal with satisfaction and optimization problems.

Drawbacks:

- as all metaheuristics techniques, simulation parameters (population size, mutation rate, etc.) are problem-dependant; several tries can be necessary to get good results;
- global convergence is not guaranteed;
- inconsistency of a description cannot be proved;
- the algorithm is not sound: scenes which are not solution can sometimes be generated.

5 Conclusion

In this paper we have presented a constraint-based system for non-isothetic 3D-object layout built on a genetic algorithm. The first results (see section 3) are promising. However, particular benchmarks are needed to allow quantitative comparisons between different approaches.

In the future, we plan to include this tool in our DEM²ONS [Kwaiter et al. 97] declarative platform, to compare it more precisely with others existing constraint solvers and others metaheuristics (e.g. local search).

However, to get an easy-to-use object-layout software, the description of the scene can be made easier by the use of semantic and functional features associated with the objects. So, the user can obtain a complex and realistic scene without giving all details.

Figure 16: Example of a realistic 3D-scene (88 objects) generated with our tool (ray tracing rendering)

References

[Bonnefoi et al. 99] P-F Bonnefoi, D. Plemenos. Object Oriented Constraint Satisfaction for Hierarchical Declarative Scene Modeling. WSCG'99, Plzen, Czech Republic, February 2-12, 1999.

[Charman 93] P. Charman. Solving Space Planning Problems using Constraint Technology. Technical report CS 57/93, Institute of Cybernetics - Estonian Academy of Sciences, 1993.

[Charman 95] P. Charman. Gestion de contraintes géométriques pour l'aide à l'aménagement spatial. Thèse de doctorat. Ecole Nationale des Ponts et Chaussées. Novembre 1995.

[Donikian et al. 93] S. Donikian, G. Hégron. A Declarative Design Method for 3D Scene Sketch Modeling. In Eurographics'93, Vol.12 (33), pp 223-236, 1993.

[Fonseca et al. 95] C.M. Fonseca, P.J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. Evolutionary Computation 3, 1 (Spring), 1-16. 1995.

[Goldberg 89] D.E. Goldberg. Genetic algorithms for search, optimization, and machine learning. Addison-Wesley, Reading, Massachusetts, 1989.

[Holland 92] J.H. Holland. Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence (Second ed.). MIT Press, Cambridge, Massachusetts, 1992.

[Koza 92] J.R. Koza. Genetic Programming. On the Programming of Computers by Means of Natural Selection. The Mit Press, 1992.

[Kwaiter et al. 97] G. Kwaiter, V. Gaildrat, R. Caubet. DEM²ONS: A High Level Declarative Modeller for 3D Graphics Applications. In Proceedings of the International Conference on Imaging Science Systems and Technology, CISST'97, pp 149-154, Las Vegas, June 30-July 3, 1997.

[Masui 92] T. Masui. Graphic Object Layout with Interactive Genetic Algorithms. Proceedings of the 1992 IEEE Workshop on Visual Languages, pp 74-80, September 1992.

[Ruchaud et al. 02] W. Ruchaud, D. Plemenos. MultiFormes: a declarative modeller as a 3D scene sketching tool. International Conference ICCVG'2002, Zakopane (Poland), September 25-29, 2002.

[Vries et al. 00] B. de Vries, A.J. Jessurun, R.H.M.C Kelleners. Using 3D geometric constraints in architectural design support systems. WSCG'00, 2000.

[Vassilas et al. 02] N. Vassilas, G. Miaoulis, D. Chronopoulos, E. Konstantinidis, I. Ravani, D. Makris, D. Plemenos. MultiCAD-GA: A System for the Design of 3D Forms Based on Genetic Algorithms and Human Evaluation. SETN 2002, pp 203-214, 2002.

[Xu et al. 02] K. Xu, J. Stewart, E. Fiume. Constraint-based Automatic Placement for Scene Composition, Graphics Interface '02, May 2002.