Constraint-Based 3D Isothetic Object Layout for Declarative Scene Modeling

Olivier LE ROUX, 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: { leroux, gaildrat }@irit.fr

***Abstract*—The aim of declarative modeling is to help the user to build a three-dimensional scene from a high level description. As underlying problems are combinatorial, constraints programming and related solving techniques are a neat and effective way to depict and to solve such problems. In this paper we focus on object layout in a three-dimensional environment that is usually a consuming time and tedious task. This study is restricted to isothetic[1] objects only. In this way, it is related to classical bi-dimensional space planning problems.**

***Index Terms*—Declarative modeling, Constraint-based 3D-scenes generation, constraint solver, combinatorial problems, isothetic object layout.**

I. Introduction

Declarative modeling [10] is an emergent paradigm in the world of computer-aided design tools. A declarative modeler allows designers to produce 2D or 3D models from a high-level description. Unlike the imperative geometric modeling, it does not require particular skills or a long training. Moreover, consistency of the description can be automatically and continuously maintained by the system.

A declarative modeler is the combination of three sub-systems [6]: a *description unit* to describe the problem, a *generation unit* to generate solutions and an *insight unit* to visualize the results. These three main components interact in a spiral design process to progressively converge towards the solution.

The generation system can be considered as the kernel of the application. Its task is to produce one or several models that match the description given by the user. Capabilities and performance of the application depends on the *generation phase*: it is a critical stage.

In this paper we focus on this problem. This study considers constraint-based 3D generation process but is restricted to isothetic objects only. Our contribution is the description of an efficient geometric constraint solver, *Manhattan*, able to solve a wide range of problems. Originality of this work is to take into account the 24 possible isothetic orientations of each 3D-object.

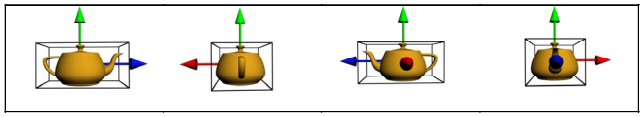

Fig. 1: four of twenty-four possible isothetic orientations of a 3D object. These orientations are the canonical ones for most of our 3D-objects.

This study is a part of our federative project DEM²ONS[2]. At this time our tool runs on a restricted platform (i.e. the multimodal layer is missing) DEMONS which allows to compare various generation approaches (see screenshots fig. 4 and 5).

This paper is organized as follows. Part II presents some related works. Part III introduces our approach and describes the inner architecture of our Manhattan constraint solver. In part IV, we give first experimental results. Part V discusses on advantages and drawbacks of our tool. We also present a qualitative comparison of our system with related ones. Finally part VI concludes this study.

II. Related works

Although procedural methods [7] [18] and shape grammars [14] [15] are used in automatic scene generation, most of the applications lead on the constraint programming paradigm.

In constraint programming there are mainly two approaches to deal with the object layout problem:

- stochastic approach: this efficient and flexible approach is based on the iterative improvement [17] or metaheuristics [16] (e.g. simulated annealing or evolutionist strategies). However, it suffers from incompleteness;
- constructive appoach (e.g. [2] [3] [9] [12]): the solution is built step by step. Used algorithms are determinist and exhaustive and based on an enumeration process. This approach has proved to be very efficient, dealing with under-constrained problems.

[1] isothetic = parallel and/or orthogonal to world coordinate system

[2] DEM²ONS = Declarative Multimodal ModeliNg System

Most of object layout and space planning applications are bi-dimensional systems (e.g. [5] [12]). But 3D systems can also be found [3] [9]. However, none of these applications handle all possible 3D isothetic orientations.

Finally, tractable problem subclasses can be obtained, admitting some restrictions on constraints [1]. In this case, related algorithms are polynomial. Unfortunately, required conditions are too restrictive for our class of problems.

III. OUR APPROACH: THE MANHATTAN CONSTAINT SOLVER

A. Overview

In this section we present our constraint solver called Manhattan. This generation tool is based on the combinatorial CSP approach (i.e. the main algorithm is a backtracking process). To take into account the geometric specificities of our problem we introduce a well-suited local-consistency inspired by [5].

B. Definitions

Our generation process is based on an extended CSP paradigm.

Definition 1: our geometric CSP P = <V, D, C> is defined as follows:

- a finite set of 3D-boxes V = { b_1,…b_n } ; each box b_i is defined by 5 sub-variables:
  - xyz: position of the reference point;
  - o: orientation of the 3D-box (see section II.C);
  - dx, dy, dz: dimension of the box along the three axis.
- a set of domains D = { D_1,…D_n }, where D_i is the domain associated with b_i (i.e. D_i defines the only authorized instanciation of b_i).
- a set of binary constraints C = { C_1,… C_n }; each c∈C is a subset of the Cartesian product $\prod_{i \in V_c} D_i$ where $V_c \subset V$ is the set of variables of c. Each constraint is defined *intentionally* from a procedural method.

Definition 2: solution of the CSP

A solution is an assignment of a value to each variable from its domain so as to satisfy all the constraints.

Definition 3: partial (or local) consistencies (PC)

A partial consistency is a consistent relaxation of a CSP that can be computed in a polynomial time.

Instead of testing individually each possible configuration, we employ a suitable partial consistency. This consistency is based on a spatial reasoning that use the Minkowski sums and differences over 3D-boxes. The resulting filtering algorithm is much more efficient than classical arc-consistency [11].

Definition 3: Minkowski sum

Let A and B be two 3D isothetic boxes. The *sum* of A and B is defined by:

$$A \oplus B = \{ x + y \mid x \in A,\ y \in B \}$$ [13]

Note:

- the set A ⊕ B and A ⊕ -B are also 3D isothetic boxes.
- unlike general algorithms on complex polygon, computing Minkowski sums on 3D-boxes is a basic and very efficient operation (time cost is O(1)).

Definition 4: semi-geometric arc-consistency (SG-AC)

Let C be a constraint on two boxes b_1 and b_2 and *ext*(C) be the set of authorized tuple for

$$\{v_{xyz1}, v_{o1}, v_{dx1}, v_{dy1}, v_{dz1}, v_{xyz2}, v_{o2}, v_{dx2}, v_{dy2}, v_{dz2}\}.$$

By definition, the domains D_{xyz1}, D_{o1}, D_{dx1}, D_{dy1}, D_{dz1}, D_{xyz2}, D_{o2}, D_{dx2}, D_{dy2}, D_{dz2} are SG-AC with the constraint C if and only if:

$\forall v_{o1} \in Do_1, \forall v_{xyz1} \in D_{xyz1}, \exists v_{dx1} \in D_{dx1}, \exists v_{dy1} \in D_{dy1}, \exists v_{dz1} \in D_{dz1}, \exists v_{o2} \in Do_2, \exists v_{xyz2} \in D_{xyz2}, \exists v_{dx2} \in D_{dx2}, \exists v_{dy2} \in D_{dy2}, \exists v_{dz2} \in D_{dz2}$ such that

$(v_{xyz1}, v_{dx1}, v_{dy1}, v_{dz1}, v_{o1}, v_{xyz2}, v_{dx2}, v_{dy2}, v_{dz2}, v_{o2}) \in ext(C)$

and

$\{\forall v_{o1} \in Do_1, \forall \mathbf{v_{dx1}} \in D_{dx1}, \exists v_{xyz1} \in D_{xyz1}, \exists v_{dy1} \in D_{dy1}, \exists v_{dz1} \in D_{dz1}, \exists v_{o2} \in Do_2, \exists v_{xyz2} \in D_{xyz2}, \exists v_{dx2} \in D_{dx2}, \exists v_{dy2} \in D_{dy2}, \exists v_{dz2} \in D_{dz2}$ such that

$(v_{xyz1}, v_{dx1}, v_{dy1}, v_{dz1}, v_{o1}, v_{xyz2}, v_{dx2}, v_{dy2}, v_{dz2}, v_{o2}) \in ext(C)$

(resp. $\mathbf{v_{dy1}}$ and $\mathbf{v_{dz1}}$)$\}$

Informally speaking, this definition indicates that a compatible configuration for the two boxes can be found according to C.

Note: the original definition given in [5] has been extended to 3D but restricted to binary constraints. It can easily be extended to any n-ary constraints.

Proposition 1: let P be a geometric CSP; by definition of SG-AC, the following proposition holds:

$$\Phi_{SG\text{-}AC}(P) \leq \Phi_{AC}(P)$$

where $\Phi_{PC}(P)$ is the closure by partial consistency of P.

C. Data structures

A specific data structure is associated to each kind of variable:

- dimension parameter: a disjunctive union (sorted in an ordered linked-list) of 1D-intervals;
- position parameter: a disjunctive union (a linked-list) of isothetic 3D-boxes;
- orientation parameter: a 24 values bit-field; each bit shows if the associated orientation is allowed. Most of the time, only the 4 orientations around the Z-axis are considered.

See appendix for algorithms and illustrations of these structures.

D. Kernel of the solver

The search algorithm is based on a classical backtracking algorithm completed with SG-AC filtering and dynamic heuristics. Both forward-checking (FC) and real-full lookahead (RFL) have been implemented.

E. *The constraints*

The system is able to handle several kinds of constraints (see example 2 in section IV) including:

- dimensional constraints: size of an object along one axis, relative sizes of two objects, etc;
- topological constraints: adjacency, non-overlapping, etc;
- constraints over orientation: the 24 initial possible orientations can be reduced using either unary constraints or binary constraints, linking the orientation of two objects.

A specific filtering method is defined for each constraint (see example on fig. 2 and 3).

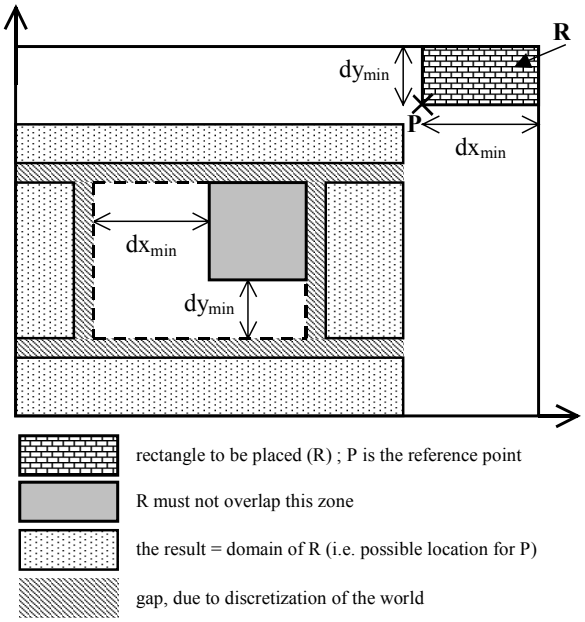

Fig. 2: Authorized positions for P (i.e. the pair (R.*x*, R.*y*)) after application of the non-overlapping constraint between R and the gray zone. In 2D, the resulting domain generates 0 to 4 zones. This illustration is inspired by [5].

F. *Improvements*

Symmetries: most of objects own one or several planar symmetries. Taking into account such symmetries allows pruning of similar branches into search trees to avoid redundant computation.

Scene realism: to avoid compacted scenes generation, we replace the classical ordered enumeration technique by a stochastic enumeration. Nevertheless, the used stochastic technique preserves exhaustivity.

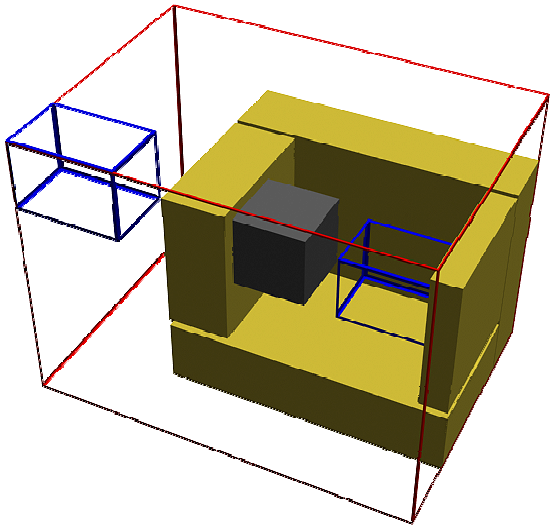

Fig. 3: Extended to 3D, our reduction algorithm for the non-overlapping constraint generates 0 (in case of inconsistency) to 6 boxes. A *position domain* is represented using a disjunctive union of boxes. On the above picture, the blue box must not overlap the gray one: the resulting domain is composed of 4 boxes.

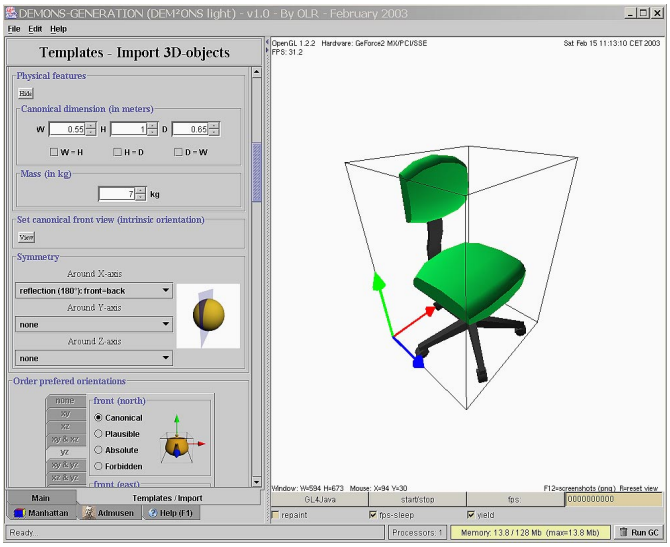

Fig. 4: screenshot #1 of our experimentation platform DEMONS - 3D-objects manager: it allows the import of 3D objects from others applications and the definition of canonical parameters for each object (including dimension, orientation, planes and axis of symmetry, etc.)

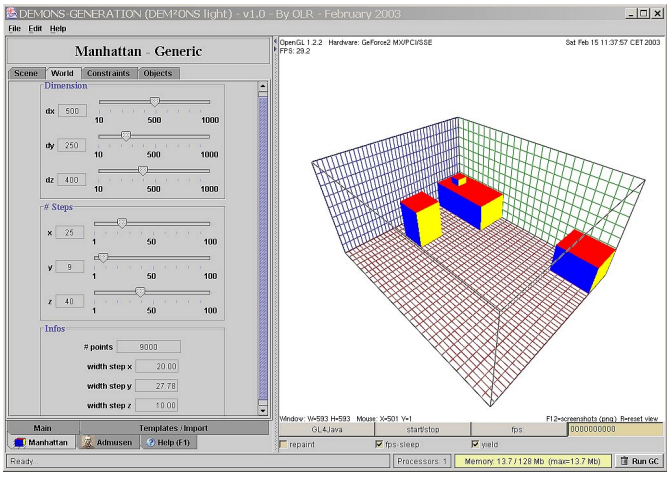

Fig. 5: screenshot #2 of our experimentation platform DEMONS - a Manhattan view (4 objects in a discretized world). Final scenes can be exported into the VRML format.

IV. First Experimental Results

Our experimentation platform, DEMONS, is developed in Java. All tests are performed on a PC Athlon 2Ghz / 256 Mb using Java JDK 1.4.1 (option–server).

Example1: the nine perfect squares problem.

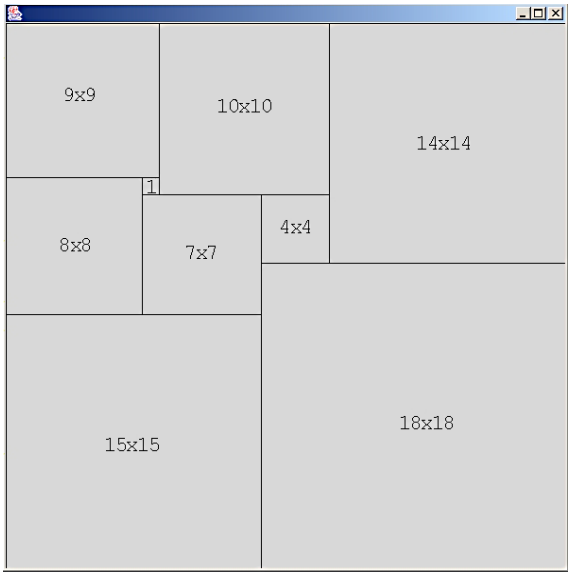

Fig. 6: the first solution found to the 9 perfect squares problem (in ~0.45 seconds). The goal is to place 9 squares (1x1, 4x4, 7x7, 8x8, 9x9, 10x10, 14x14, 15x15, 18x18) in a 33x32 world. As this problem is well-constrained (there are only 4 solutions), it is hard to solve: it corresponds to a peak of complexity also called the transition phase. To solve this 2D problem with our tool, we fix the dimension along z-axis and we restrict possible orientations to only one value. The classical *bigger first* heuristic has been used.

Example2: a simple 3D scene example.

```
/* Script Manhattan 1.0 – by OLR, 01/2003 */
/* Scene: outside */

createWorld "theWorld" 180 180 100 5 5 5

/* Add objects to the scene */
/* Templates are extracted from our object database */
/* Syntax: add [#instance] "name" instanceOf "template" */

/* Generates tree0, tree1, ... tree9 */
add 5 "tl1" instanceOf "templateTree1"
add 5 "tl2" instanceOf "templateTree2"
add 10 "tl3" instanceOf "templateTree3"
add 3 "tr1" instanceOf "templateTree1"
add 2 "tr2" instanceOf "templateTree3"
add "theRoad" instanceOf "templateRoad"
add "theCar" instanceOf "templateCar"
add "theEagle" instanceOf "templateEagle"

/* Apply constraints */
/* Note: dimensions are extracted from the database */
/* Syntax: apply constraint arg1...argn to obj1... objn */
apply in "theWorld" to all
apply orientationAroundZOnly to all
apply orientationAroundZEqual 0 to "theCar" "theEagle"
apply setXYZ 0 50 0 to "theRoad"
apply setZ 0 to "tl1" "tl2" "tl3" "tr1" "tr2"
apply on "theRoad" to "theCar"
apply above "theRoad" to "theEagle"
apply nonOverlapping to "theCar" "theEagle"
apply onTheLeft "theRoad' to "tl1" "tl2" "tl3"
apply onTheRight "theRoad' to "tr1" "tr2"
apply nonOverlapping to "tl1" "tl2" "tl3"
apply nonOverlapping to "tr1" "tr2"
```

The scripted description corresponding to the following scene.

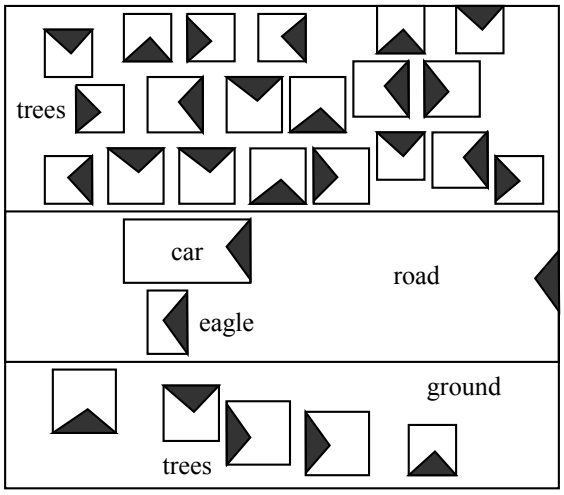

Fig. 7: orthographic projection of a resulting scene. This scene contains 28 objects. Each object can take one of the four possible isothetic orientations around the Z-axis; in the above scheme the dark gray arrows symbolize orientations. To improve the realism of the result, our stochastic enumeration heuristic is used. Computation time: ~10 seconds. Note that the problem is widely under-constrained.

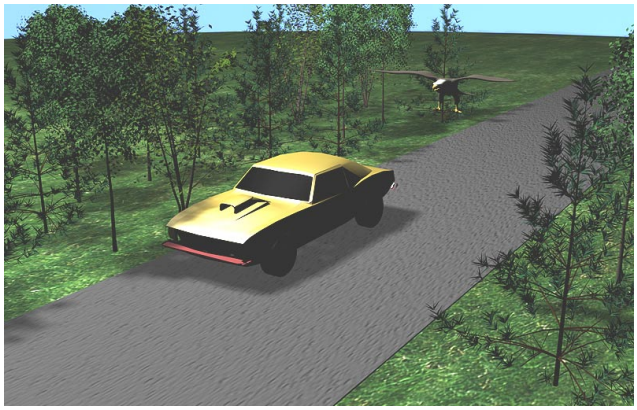

Fig. 8: The same scene after rendering (exported to VRML then ray-traced after some minor improvements including texturing and skydome).

V. DISCUSSION

By now, there are no standardized benchmarks for comparing 3D isothetic layout problems. Although our solver can deal with 2D space planning problems (see fig. 6), it has not been specially designed for this task. For these reasons Table1 (see below) proposes a qualitative comparison with others related works.

In the following of this section we give a summary of the main advantages and drawbacks of our tool.

Advantages:

- most of the descriptions entail under-constrained CSP which are easy to solve (i.e. a first solution can be produced in a short time);
- global convergence is guaranteed (i.e. the algorithm is exhaustive and sound);
- all the solutions can be generated;
- if there is no solution, inconsistency of the description can be proved;
- scenes that look realistic can be produced using a stochastic enumeration technique instead of classical sorted enumeration.

Drawbacks:

- computation time cannot be foreseen. Some problems can be very time consuming;
- in the worst case, the time complexity is exponential;
- at this time, the system cannot deals with over-constrained problems;
- Manhattan is not designed for pure space planning problems. Many applications are more suited for this kind of problems (e.g. ARCHiPLAN [12]).

Project	EAAS	DEM²ONS ORANOS	MultiFormes 4	ARCHiPLAN	DEMONS GA	DEMONS Manhattan
Year	1993-1995	1997-1998	1999	2001	2003	2003
Main reference	[5]	[9]	[3]	[12]	[16]	This paper
Approach	constructive	constructive	constructive	constructive	iterative	constructive
Is result guaranteed?	YES	NO (discretization)	YES	YES	NO (stochastic)	YES
Objects / degrees of freedom	2D rectangles 4	3D boxes 9	3D boxes 6	2D rectangles 4	3D boxes 7	3D boxes 5
Orientation	0° or 90°	the 24 isothetic orientations	1	0° or 90°	any around the Z-axis	the 24 isothetic orientations
Hierarchical constraints	NO	YES	NO	NO	YES	NO
Algorithm	**S-CSP** FC or RFL + CBJ (optional) + filtering SG-AC (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)	**Branch and bound (optimization)** + dynamic space ordering heuristic (dso)	**GA** About 300 individuals + domain pre-filtering + classical genetic algorithm	**S-CSP** FC or RFL + filtering SG-AC 3D (derived from AC3) + dynamic heuristic (FFP based) + stochastic enumeration

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: qualitative comparison of our solver with related systems

VI. Conclusion and Future Works

In this paper we have presented a combinatorial approach so as to solve the constraint-based 3D isothetic object layout problem. The resulting solver, Manhattan, can be used as an efficient generation tool for declarative modeling.

First results are promising, nevertheless many improvements are necessary to get a more convenient system. For example:

- add new geometric constraints;
- make the algorithm incremental in order to improve performances;
- add functional and semantic aspects;
- handle sub-parts of objects to take into account their functionalities;
- etc.

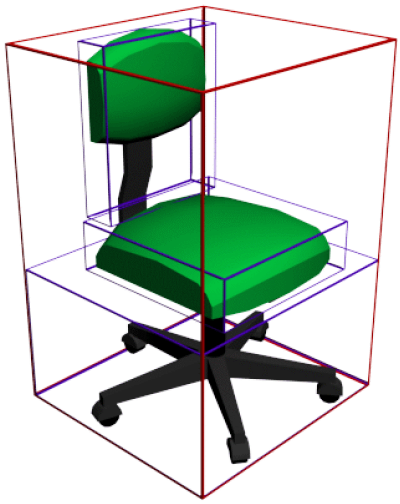

Fig. 9: three subparts - still isothetic boxes - are defined on this wheelchair: the sitting, the back and legs. Using a hierarchy of sub-objects will allow a better representation and a more accurate description.

Appendix

In this section we give the basic geometric algorithms required to implement the solver:

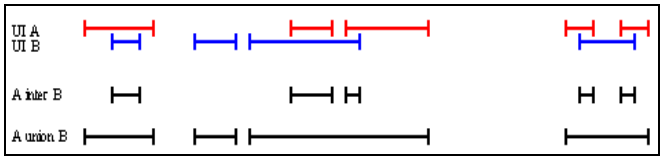

Fig. 10: geometric union and intersection of UI objects (disjunctive Union of Intervals). Possible dimensions of 3D-objects are represented using an UI along each axis.

```
// UI = Union of Intervals = an ordered list of intervals
// Computes and returns intersection between two union of intervals, O(n).
// Returns Ddx ∩ Ddy or EMPTY if the two UI are separated.
intersection ( in UI Ddx, in UI Ddy ) return UI {
    if (empty(Ddx) || empty(Ddy)) return EMPTY;
    Iterator itx = iterator( Ddx );
    Iterator ity = iterator( Ddy );
    Interval a = next( itx );
    Interval b = next( ity );
    UI result;    // create a new instance of UI
    while (true) {
        if (lo(b) < lo(a))    // compare low bound
            exchange a↔b and itx↔ity
        if (hi( a ) < lo( b )) {
            if (!hasNext( itx )) break;
            a = next( itx );
            continue;
        }
        if (hi( b ) < hi( a )) {
            addLast( res, b );
            if (!hasNext( ity )) break;
            b = next( ity );
            continue;
        }
        addLast(res, makeInterval(lo(b), hi(a));
        if (!hasNext( itx )) break;
        a = next( itx );
    }
    return result;
}
```

Algorithm. 1: the algorithm used to intersect two dimension domains (represented by a disjunctive UI = union of intervals). The *union* and *difference* operators on UI are similar functions (see fig. 10).

```
// UB = disjunctive Union of 3D-Boxes = a list of boxes
// A box is represented by a reference point (x, y, z) and its
// dimension (dx, dy, dz)
// Computes and returns intersection between two disjunctive
// unions of boxes ; O(n²).
// Returns Dxyz1 ∩ Dxyz2 or EMPTY if the two UB are separated.
intersection ( in UB Dxyz1, in UB Dxyz2 ) return UB {
    if (empty(Dxyz1) || empty(Dxyz2)) return EMPTY;
    Iterator it1 = iterator( Dxyz1 );
    UB result;    // create a new instance of UB
    while (hasNext( it1 )) {
        Box a = next( it1 );
        Iterator it2 = iterator( Dxyz2 );
        while (hasNext( it2 )) {
            Box b = next( it2 );
            Box r = intersection( a, b );
            if (!empty( r ))
                result.addLast( r );
        }
    }
    return result;
}
```

```
// Computes and returns difference between two disjunctive
// unions of boxes ; O(n²).
// W is the bounding box of the world.
// Returns Dxyz1 - Dxyz2 or EMPTY if the two UB are separated.

difference ( in UB Dxyz1, in UB Dxyz2,
             in Box w, in float gap ) return UB {
    if (empty(Dxyz1) || empty(Dxyz2 )) return EMPTY;
    Iterator it1 = iterator( Dxyz1 );
    UB result;    // create a new instance of UB
    while (hasNext( it1 )) {
        Box a = next( it1 );
        List tmpList;
        Iterator it2 = iterator( Dxyz2 );
        while (hasNext( it2 )) {
            Box b = next( it2 );
            List r = difference( a, b, w, gap );
            if (!empty( r ))
                list.add ( r );
        }
        result.addAll( tmpList );
    }
    return result;
}
```

Algorithm. 2, 3: the algorithms used to compute intersection and difference between two position domains (represented by a separated UB = union of 3D-boxes) (see fig. 11).

Note: the *difference* operator between two 3D-boxes is implemented as follows:

$$\mathbf{B_1 - B_2} = B_1 \cap \text{Complement}(B_2, \text{world})$$

To preserve separation of every box, the *union* operator is defined from the *difference* by:

$$\mathbf{A \cup B} = A \cup (A - B)$$

where $\cup$ consists in merging the two lists of boxes.

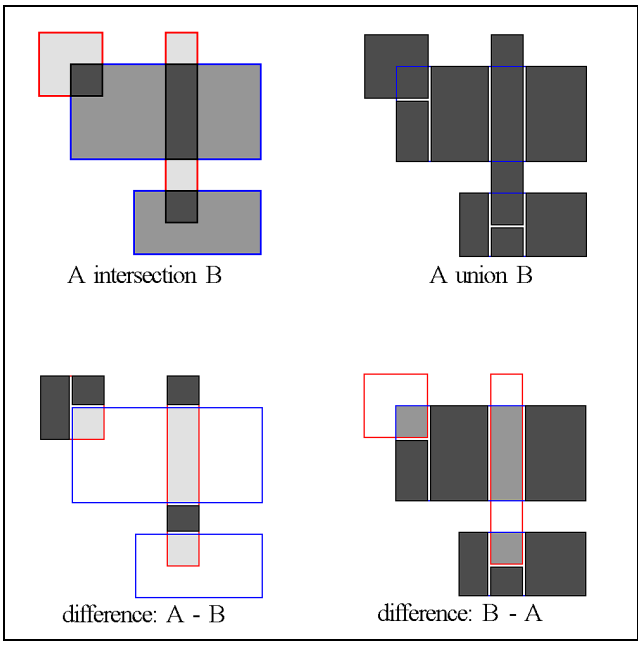

Fig. 11: orthographic projections (i.e. 2D representation) of geometric union, intersection and difference of UB objects (disjunctive Union of 3D-Boxes).

REFERENCES

[1] P. Balbiani, J-F. Condotta, L. Farinas del Cerro, "*A new tractable subclass of the rectangle algebra*", IVCAI'99, 1999.

[2] C. Baykan, M. Fox, "*Constraint Satisfaction Techniques for Spatial Planning*", in Intelligent CAD Systems III, Practical Experience and Evaluation, 1991.

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

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

[5] P. Charman, "*Gestion de contraintes géométriques pour l'aide à l'aménagement spatial*", PhD, Ecole Nationale des Ponts et Chaussées. November 1995.

[6] C. Colin, E. Desmontils, J.Y. Martin, J.P. Mounier, "*Working Modes with a Declarative Modeler*", Compugraphics'97, pp. 117-126, 1997.

[7] B. Coyne, R. Sproat, *"WordsEye: an automatic text-to-scene conversion system"*, SIGGRAPH 2001.

[8] C. Eastman, "*Automated space planning*", Artificial Intelligence 4, pp. 41-64, 1973.

[9] G. Kwaiter, V. Gaildrat, R. Caubet, "*DEM²ONS: A High Level Declarative Modeler 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.

[10] M. Lucas, D. Martin, P. Martin, D. Plemenos, *"Le projet ExploFormes, quelques pas vers la modélisation déclarative de formes"*, Journées Groplan, 1989.

[11] A. K. Mackworth, "*Consistency in Networks of Relations*", Artificial intelligence, Vol. 8 (1), pp. 99-118, 1977.

[12] B. Medjdoub, B. Yannou, "*Dynamic space ordering at a topological level in space planning*", Artificial intelligence in engineering, Vol 15, pp. 47-60, 2001.

[13] J. O'Rourke, "*Computational Geometry in C*", Cambridge University Press, 2nd edition, 1998.

[14] Y. Parish, P. Müller, "*Procedural modeling of cities*", SIGGRAPH'01, Los Angeles, pp. 301-308, August 2001.

[15] A. Rau-Chaplin, B. MacKay-Lyons, P. Spierenburg, "*The LaHave House Project: Towards an automated architectural design service*", Cadex'96, pp. 24-31, September 1996.

[16] S. Sanchez, O. Le Roux, H. Luga, V. Gaildrat, "*Constraint-Based 3D-Object Layout using a Genetic Algorithm*", in 3IA'03, Limoges, France, 2003. Submitted.

[17] K. Shahookar, P. Mazumder, "*VLSI Cell Placement Techniques*", ACM Computing Surveys, Vol. 23, No. 2, June 1991.

[18] K. Xu, J. Stewart, E. Fiume, "*Constraint-based Automatic Placement for Scene Composition*", Graphics Interface 2002.