Constraint-based Automatic Placement for Scene Composition

Ken Xu
University of Toronto

James Stewart
Queens University

Eugene Fiume
University of Toronto

Abstract

The layout of large scenes can be a time–consuming and tedious task. In most current systems, the user must position each of the objects by hand, one at a time. This paper presents a constraint–based automatic placement system, which allows the user to quickly and easily lay out complex scenes.

The system uses a combination of automatically–generated placement constraints, pseudo–physics, and a semantic database to guide the automatic placement of objects. Existing scenes can quickly be rearranged simply by reweighting the placement preferences. We show that the system enables a user to lay out a complex scene of 300 objects in less than 10 minutes.

1 Introduction

Object layout is an important and often time consuming part of modeling. It takes six degrees of freedom to fully lay out an object, while standard input devices have only two degrees of freedom (DOF). In addition, the layout must satisfy physical constraints such as non-interpenetration and physical stability. As a result of the high cost of object layout, computer graphics scenes are often unrealistically simple or overly tidy.

Previous efforts to facilitate object placment have resulted in better input devices and more efficient object manipulation techniques, which allow individual objects to be placed more quickly and accurately. However, the layout of large, complex scenes remains a difficult problem because users must still manually place objects into the scene one at a time.

In this paper we present CAPS, a Constraint-based Automatic Placement System. CAPS makes feasible the modeling of large, complex scenes: A scene consisting of more than 300 objects can be laid out in less than 10 minutes, as shown in Figure 1. The system uses a set of intuitive placement constraints to allow the manipulation of large numbers of objects simultaneously. Through the use of pseudo–physics, objects can automatically be placed in physically stable configurations, once their placement constraints have been set. A user need not be concerned with the details of placement, unless he or she wishes to. In addition, CAPS attaches semantic information to objects, which allows placement constraints to be generated automatically. As a result, CAPS can place hundreds of objects into a scene and can quickly redistribute them into semantically meaningful locations with absolutely no user intervention.

2 Related Work

A number of researchers have addressed the issue of object placement in 3D environments. These techniques can be broadly categorized into four groups:

1. techniques which attempt to reduce the number of DOF to which users are exposed;
2. techniques which abandon low DOF input devices altogether in favor of more complicated devices — some with as many as six DOF — in order to facilitate direct object manipulation;
3. techniques which employ pseudo–physics to automatically compute a physically stable position and orientation for an object, after having dropped it from some spot in the scene; and
4. techniques which use semantic information to guide object placement.

The vast majority of existing approaches have treated object placement as a purely geometric problem. However, more recent approaches have begun to utilize semantic information in the layout process.

2.1 DOF Reduction Techniques

Some DOF–reduction methods attempt to compensate for the two DOF of common input devices by mapping 2D input vectors to higher dimensional vectors [20, 32]. Others ease object placement with “snap to” constraints, such as snapping to grids, object faces, or auxiliary helper geometry [1]. A final group of methods makes the assumption that surfaces are planar, and tends to restrict object motion along those surfaces, thereby reducing the DOF to which users are exposed [11, 6, 27].

2.2 Input Devices and Manipulation Techniques

The limited DOF available on standard input devices has motivated much research on higher DOF input devices. Such devices include the Space Ball (a six DOF joystick), the bat (a six DOF mouse) [34], the Roller Mouse

(a three DOF mouse) [32], and the Data Glove (a six DOF device which can encode the position of each of the user's fingers) [36]. These devices are often used in conjunction with other specialized hardware, such as head mounted displays, to completely immerse the user in a virtual environment. A number of existing VR systems [15, 14, 33, 7, 18, 22] make use of these specialized input devices.

The new six DOF devices allow users to reach out a hand, grab an object, and manipulate it as one would in the real world [25]. This direct mapping metaphor has some problems. First, the physical arm is confined to a small space around the user, so many objects cannot be directly reached and the user must travel to the location of the object before being able to handle it. Second, manipulation of large objects is difficult, not because they are heavy, but because they obscure the user's view during the placement task.

Techniques which overcome these problems include World In Miniature [29], Automatic Scaling [19], Go-Go [24], and ray casting techniques [18, 9, 22]. Bowman [4] and Poupyrev [23] provide very nice categorizations of existing techniques.

Other research using specialized hardware includes object manipulation techniques in Augmented Reality, where virtual and real objects appear together in a scene. As an example, Kitamura [12] discussed ways of using haptic feedback to make object manipulation feel the same as in the real world.

2.3 Pseudo–Physical Techniques

The use of pseudo–physics can help to automatically place objects in physically stable positions, without incurring the computational cost of a full physical simulation. In systems which use pseudo–physics, users need not be concerned about the details of placement; they need only to drag the object into an approximate location, drop it, and let the pseudo–physics do the rest. There are many implementations of pseudo–physics, including those of Shinya and Forgue [26], Snyder [28], Breen [5], and Milenkovic [16, 17].

2.4 Semantic Techniques

The vast majority of layout systems consider only geometry. A few layout systems also exploit semantic information about the objects that they manipulate, including Houde's system [11] and the MIVE system [27]. Houde attaches to objects "narrative handles," which are positioned and shaped to indicate their manipulation capability. MIVE attaches semantic information to objects in the form of labels, "binding areas," and "offer areas." If the labels of two objects are compatible, objects are placed together by connecting binding areas to offer areas. The binding and offer areas are specified manually by the user. To assist in scene manipulation, grouping of objects is automatically performed [30].

Semantic information in the form of *constraints* has long been used in editing complex objects. Examples include Sutherland's original Sketchpad system [31], Borning's ThingLab system [2], and the Cassowary constraint solver [3].

Recently, Coyne and Sproat [8] demonstrated the power of semantic information in assisting scene composition. Their WordsEye system uses a text description to gather semantic information about the scene. From a pre–existing database, three dimensional (3D) objects are matched to the objects described in the text, and are placed in locations consistent with the text description. WordsEye allows prototype scenes to be quickly generated, based on a few lines of text.

2.5 Issues to Address

The various techniques described above have significantly improved user manipulation of individual objects. Nearly all of these techniques eliminate the need for multiple projected views. Certain techniques, such as that of Bier [1] and Smith *et al.* [27], can make very accurate object placements. Six DOF input devices have the advantage that manipulation feels more natural, and pseudo–physical techniques eliminate the need for users to be concerned about the details of placement. Finally, the Wordseye system demonstrates that layout can be made trivial if the system in question considers the *semantic information* associated with the objects that it manipulates.

Despite these advances, however, weaknesses still exist. Techniques such as those by Nielson [20] and Venolia [32] suffer from lack of control, and it may be quite troublesome to define the alignment manifolds as suggested by Bier [1]. Six DOF manipulation techniques have the disadvantages that the specialized hardware is expensive, is not universally accessible, and can cause noticeable physical fatigue.

While the Wordeye system demonstrated the power of semantic information for object layout, it may not be appropriate as a general layout tool because of the inherent ambiguity of natural language: The system could easily misinterpret the intent of the user. Many systems (with the exception of pseudo–physical techniques, of course) have no sense of the physical constraints that govern object placement, while others have only limited pseudo–physical support, and assume that objects are to be placed in an upright position only.

Most importantly, nearly all of the techniques examined here manipulate *only one object at a time*. A realistic scene having hundreds or thousands of objects cannot be efficiently laid out one object at a time.

Figure 1: A scene of 300 objects which was laid out in less than 10 minutes with CAPS.

3 Constraint–based Automatic Placement

CAPS, which is the subject of this paper, allows users to create and manipulate large scenes quickly and easily. CAPS can lay out large numbers of objects simultaneously. It has a rich pseudo–physics which allows objects to be placed in arbitrary, stable configurations. It exploits semantic information to aid in the placement. It permits objects to be placed randomly (within the limits of their placement constraints and pseudo–physical constraints), which results in scenes that exhibit a high degree of visual richness and realism.

In the sections that follow, each of pseudo–physics, placement constraints, and semantics will be discussed in turn.

3.1 Pseudo–physics

CAPS uses the pseudo–physics engine of Shinya and Forgue [26], which provides several features: non–interpenetration of objects, object stability (using a support polygon), and a limited form of friction. Using this model, objects that are dropped from above a surface will come to rest in a physically realistic position on the surface. Figure 2 shows a scene created using the pseudo–physics engine in CAPS.

3.2 Placement Constraints

CAPS uses constraints to facilitate the placement of objects. A set of constraints is associated with each object to define where the object may or may not be placed. The constraints can be as precise or as vague as the user requires.

3.2.1 Constraints

A **surface constraint** indicates how the object is to be placed on the surface of another. The constraint is specfied by

- The supporting surface.
- A boolean flag indicating whether the placement is to be exact.
- If placement is exact, the exact placement location on the surface; otherwise, one or more *container polygons* and zero or more *forbidden polygons* on the surface.

If placement is exact, the object must be placed at the exact location specified. Otherwise, the object must be placed inside one of the containing polygons and outside all of the forbidden polygons.

A **proximity constraint** indicates how close the object should be placed to relative to another object, and is specified by a *proximity polygon*. The NEAR constraint causes placement within the polygon; the AWAY constraint causes placement outside the polygon.

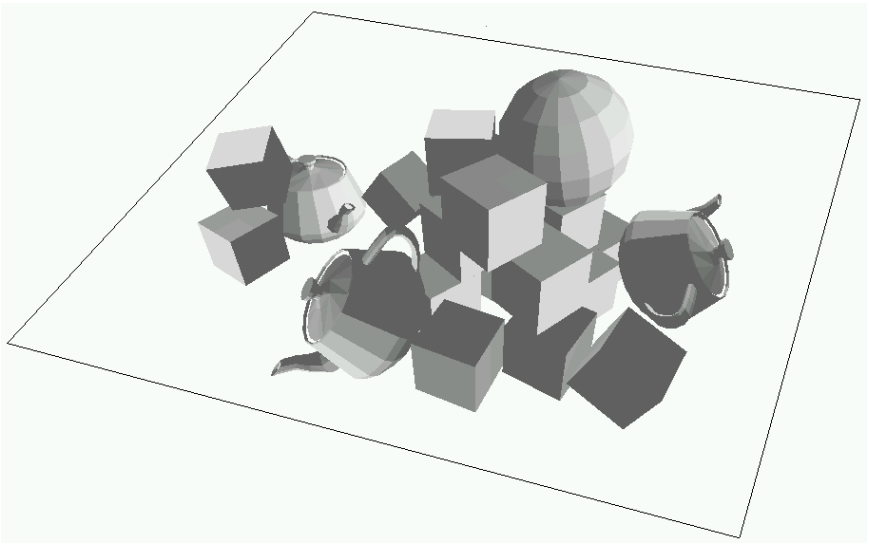

Figure 2: Pseudo–physics in CAPS automatically ensures the stability and non–interpenetration of objects.

A **support constraint** indicates whether the object can support others and whether it can be supported by others. The CANSUPPORT boolean flag is true if and only if the object can have others on top of it. The CANBESUPPORTED boolean flag is true if and only if the object can be on top of others.

3.2.2 Spatial Planning with Constraints

Given the constraints on an object's placement, two dimensional spatial planning [10] is used to find the set of allowable placements: Let S, F, and O be point sets representing, respectively, a surface, forbidden areas on that suface, and an object. Let O^t be the translation of O from its default position by the vector t. Then the problem is to find a t such that O^t is entirely inside of S (i.e. $O^t \subseteq S$) and entirely outside of F (i.e. $O^t \cap F = \emptyset$).

To use 2D spatial planning for the 3D objects in a scene, we use the *footprint* of each object: The footprint is the convex hull of the projection of the object directly downward onto the ground plane. Only the footprints are used to plan where an object is to be placed (an extension to 3D spatial planning is discussed in section 5). The constraints are implemented as follows:

- **Surface constraints:** Let S be the union of the container polygons, F be the union of the forbidden polygons, and O be the object.
- **Proximity constraints:** For a proximity polygon P, the NEAR constraint causes S to be restricted to P (i.e. $S \leftarrow S \cap P$) and the AWAY constraint causes P to be used as a forbidden polygon (i.e. $F \leftarrow F \cup P$).
- **Support constraints:** If an object's CANSUPPORT flag is false, that object's footprint is viewed as a forbidden polygon when placing any other object. If an object's CANBESUPPORTED flag is false, the footprints of all other objects are treated as forbidden polygons when placing that object.

All constraints are thus reduced to an instance of the spatial planning problem.

3.2.3 Solving the Spatial Planning Problem

The spatial planning problem has classically been solved using the theory of Minkowski sums and differences. The Minkowski sum of two point sets A and B is defined as

$$A \oplus B = \bigcup_{b \in B} A^b,$$

where A^b is A translated by b. The Minkowski difference of two point sets A and B is defined as

$$A \ominus B = \bigcap_{b \in B} A^{-b}.$$

Given two point sets, A and B, it has been shown [13] that Minkowski sums can be used to compute the set of *forbidden translations* T_f such that $B^t \bigcap A \neq \emptyset$ for $t \in T_f$. It has also been shown [10] that Minkowski differences can be used to compute set of *permitted translations* T_p such that $B^t \subseteq A$ for $t \in T_p$.

The spatial planning problem can thus be solved using Minkowski sums and differences. Given point sets S, F, and O (as defined in the preceding section), we can find

the set of safe translations of O as follows: First, compute the set of permitted translations T_p such that $O^t \subseteq S$ for $t \in T_p$. Second, compute the set of forbidden translations T_f such that $O^t \bigcap F \neq \emptyset$ for $t \in T_f$. Then the set of safe translations is $T_p - T_f$.

It is not feasible to compute Minkowski sums and differences for arbitrary point sets. However, algorithms do exist to compute the Minkowski sum and difference for simple polygons. Even so, the general algorithm for simple polygons is slow and complicated to implement. Fortunately, algorithms for Minkowski sums and differences are simpler and faster for a number of special cases. In CAPS, we take advantage of these special cases, and restrict all forbidden polygons to be convex, and all container polygons to be simple. Thus, we need to compute the Minkowski sum for only the convex-convex case, and the Minkowski difference for the simple-convex case, using techniques by O'Rourke [21] and Ghosh [10] respectively.

3.3 Semantics–based Constraints

In a realistic scene, object layout is governed not only by physical constraints, but also by semantic ones including function, fragility, and interactions with other objects. Semantic properties of objects are often independent of the geometry, and are constant over the vast majority of scenes. CAPS therefore maintains a *semantic database* of information about objects, and uses it to generate *default placement constraints* for objects.

3.3.1 Semantic Database

With CAPS, the user assigns each object to a *class*, which represents the real–world classification of the object. For example, there might be a class for all tables, and another for all chairs. For the purpose of layout, it is useful to know the set of plausible placements that objects of one class can take relative to objects of each other class. To this end, each class C stores the following information:

- A list of *parent classes*: Objects of the parent class typically appear *under* objects of C. For example, if the current object class is a plate, parent classes might be tables and counters. These are denoted "parent classes" because the supporting object is the parent of the supported object in the CAPS scene graph.

- A list of *child classes*: Objects of the child class typically appear *on top of* objects of C. For example, if the current object class is a bookshelf, child classes might be books and plants. (The child and parent classes are symmetric; we use the two, instead of just one, to make the placement algorithm more efficient.)

- A CANSUPPORT flag, which is true if and only if objects *other than* those in C's child classes may appear on top of objects of C.

- A CANBESUPPORTED flag, which is true if and only if objects *other than* those in C's parent classes may appear under objects of C.

- For each parent class, an *orientation constraint* to control the orientation of objects of C with respect to objects of the parent class.

The child and parent classes determine the surfaces on which an object O can be placed. If such a surface already supports other objects, the CANSUPPORT and CANBESUPPORTED flags determine whether O can be "piled" on top of those objects.

For example, a book is typically placed upon a table, but if a lamp appears on that table already the book should not be placed upon the lamp. But if other books appear on that table, the new book could reasonably be placed upon the others. In terms of the semantic database: tables are in the parent class of a book; lamps have a false CANSUPPORT flag; and books have a true CANSUPPORT flag (used to allow the books on the table to support other objects), and a true CANBESUPPORTED flag (used to allow the book currently being placed to be piled upon other objects).

3.3.2 Default Placement Constraints

For a particular object O, placement constraints are automatically generated by CAPS from the semantic database, as follows:

- A default surface constraint is defined for each supporting object in O's parent classes. The default constraint has a single container polygon which is the boundary of the object's upward–pointing surface, and has no forbidden polygons.

- For each supporting object, the CANSUPPORT and CANBESUPPORTED flags, as well as any orientation constraints, are taken from the semantic database.

Automatic generation of default placement constraints greatly simplifies the scene layout task: The user can populate a scene with hundreds of objects in a matter of minutes, and can then refine the placements by modifying the constraints or by repositioning objects manually.

3.3.3 Classes vs. Instances

Surface constraints, proximity constraints, support constraints, and orientation constraints apply to object instances, while semantic constraints such as parent class and child class apply to object classes. The values

for CANSUPPORT, CANBESUPPORTED, and orientation constraints, as stored in the semantic database, may be thought of as default values that are given to new object instances of a particular class. These default values are assigned to the initial instance, and may later be changed by the user.

3.4 Scene Layout with CAPS

CAPS automatically lays out a scene by placing objects into the scene, one object at a time. As each new object is placed, the object finds a feasible position which satisfies its placement constraints. The placement algorithm for a single object O is as follows (see Figure 3 for an example):

1. Choose a surface S from amongst those objects in O's parent classes. This choice is made randomly according to user–defined probability distribution over the available surfaces (or according to a uniform distribution if the user has defined none).

2. Identify all the forbidden regions on the surface S. The set of forbidden regions includes any forbidden polygons specified with the surface constraint of S, as well as the footprints of all objects that have already been placed on S and that have a false CANSUPPORT flag. If O has a false CANBESUPPORTED flag, the footprints of *all* objects currently on S will be considered forbidden regions.

3. Perform spatial planning as described in Section 3.2.3, using the forbidden polygons calculated above, and using a container polygon which defaults to the boundary of S. (The user may have explicitly defined the container polygon to be smaller by editing the automatically–generated surface constraint.)

4. If no safe positions exist for O, go back to Step 1 and attempt to place O on one of the other available surfaces. Note that CAPS does not attempt to undo previous placements of other objects in order to place O. If there is no surface on which to safely place O, nothing is done.

5. Of the safe positions calculated for O in Step 3, choose one at random.

6. Since there may be other objects (which have a true CANSUPPORT flag) at the chosen position, move O *above* the chosen position and drop it, using the pseudo–physics engine to compute a physically stable configuration.

3.4.1 Object Placement by the User

The user may place objects into the scene by selecting a surface and selecting multiple objects to be placed on that surface. This process is considerably simplified by the semantic database: Once the user has selected a surface, a list is presented of objects that can appear on that surface. The user may choose any number of any type of those objects, which CAPS will then place on the surface in positions that satisfy all placement constraints (using Steps 2 through 6 above). For example, each bookshelf in Figures 1 and 4 was populated simply by selecting it and instructing CAPS to add a certain number of books to it.

CAPS thus provides the user with a means of very quickly increasing the visual richness of the scene.

3.4.2 Adjustable Level of Control

CAPS provides users with the precise level of control that they require. At the highest level, users may rely on the automatically generated placement constraints. Should those prove unsatisfactory, the user may replace any automatically generated constraint with user defined ones. For example, the user can replace a default surface constraint by sketching the container and forbidden polygons on the surface. If precision is required, the user can restrict object placement to a single point, thus providing as much control as any previous method. CAPS never forces a user to make placements that are more precise than is required, which saves time in the layout process and produces a more realistic and visually rich scene.

3.4.3 Fast Object Redistribution using Scenarios

An object of class C may be placed on objects of its parent classes. A weight is assigned to each parent class of C, and the objects of C are placed on the objects of the parent classes in proportion to these weights. By default, these weights are all equal, yielding a uniform distribution.

These weights are called *scenarios* because they vary with the situation that is being modelled. Before supper, for example, plates are likely to be on the table; after supper, they are more likely to appear beside the sink. The weights of the plate's parent classes (the table and sink classes) may be modified to reflect these two situations.

CAPS permits objects to be *redistributed* in the scene simply by changing the scenarios. Upon such a change, the objects are removed from their current positions — any supported objects being settled with pseudo–physics — and are redistributed according to the new scenarios.

Objects of class C are distributed amongst the parent classes in proportion to the scenario weights. But amongst the *objects* of each parent class, a *secondary distribution* may be used (if desired) to favour placement on

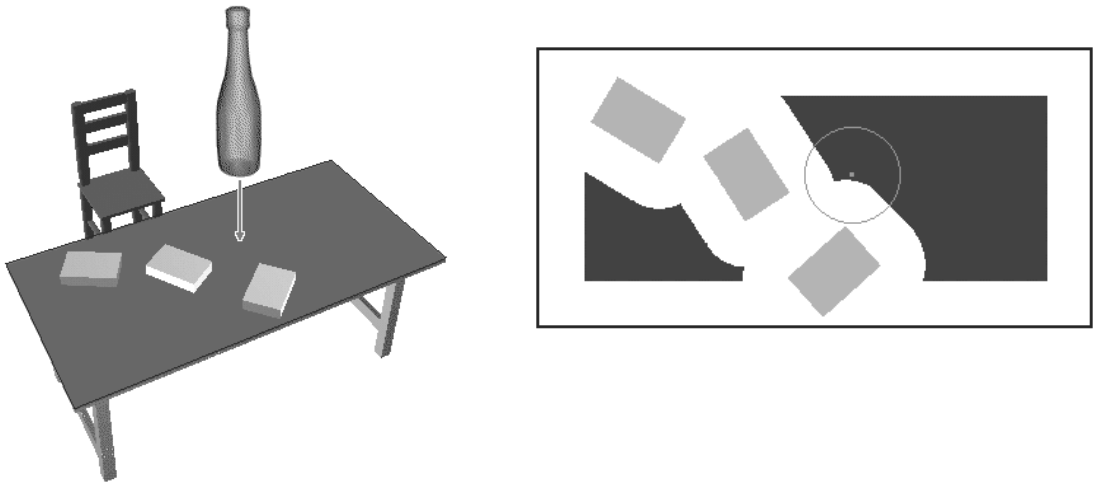

Figure 3: Left: *Positioning a carafe on a tabletop which already supports three books. The carafe has a false* CANBE-SUPPORTED *flag, so it cannot be placed on any of the books.* Right: *The two dark areas, computed with Minkowski sums and differences, represent the set of safe positions for the carafe, whose circular footprint is shown.*

certain objects of the same parent class. As with the primary distribution (i.e. the scenarios), the secondary distribution is, by default, uniform.

Objects may thus be redistributed quickly and easily by simply adjusting the scenario weights and, if more control is desired, by adjusting the weights of the secondary distributions.

3.4.4 Direct Object Manipulation by the User

In addition to its automatic layout capabilities, CAPS permits the direct user manipulation of already–placed objects. Direct manipulation could be a tedious task due to the need to explicitly group objects and to restore physical stability once changes have been made. For example, to move a table that supports other objects — without explicitly grouping the table and objects — could result in the objects floating in space. Also, moving the bottom book from a pile of books requires adjustments to all the books above it. CAPS provides dependency tracking and implicit grouping to assist direct user manipulation.

Dependency tracking maintains the physical dependences between objects: For each object, CAPS keeps a list of all the other objects that might affect the stability of that object if they are moved. When a physical instability arises, CAPS can quickly determine which objects need to be re–stabilized, and can use the pseudo–physics engine to calculate new, stable positions for them.

Implicit grouping is achieved by moving as a group all dependency–connected objects: Such a group is called a **pile**. For example, if the user moves a bookshelf that supports several books, the books will automatically move with the bookshelf.

To move objects from one surface to another, the automatic placement mechanism is used to initially place the object on the new surface, after which direct user manipulation may be done. During direct manipulation, semantic constraints are purposefully disabled to allow users full freedom of manipulation. Visual feedback from the manipulation is instantaneous. Collision detection is enabled to prevent piles from interpenetrating. Dependency tracking and implicit grouping are both implemented using a novel data structure called the "Footprint Based Dependency Graph" [35], which is updated during the automatic placement process.

4 Results

The combination of pseudo–physics and the semantic database proves to be quite powerful for purposes of scene layout. Pseudo–physics ensures that physical constraints are satisfied, while the semantic database provides plausible layouts which may easily be modified.

CAPS permits the user to act as a director, since the details of object placement are handled by the computer. For example, the user can select a bookshelf, ask that 100 books be placed on the shelf, and CAPS will do the rest. A user can thus very quickly populate a scene with hundreds of objects, which brings a visual richness to the scene not otherwise achievable in a limited amount of time. Manual refinement (with the mouse or a six DOF input device) can then be used to fine–tune the scene to its exact desired form.

Figure 4: A scene of 500 objects which was laid out in 25 minutes with CAPS. Most of the time was spent by the user in exactly positioning monitors and chairs. (Consider that a traditional modeling system would require the user to exactly position the 479 other objects.)

Once the scene has been populated, the objects can be quickly redistributed to semantically acceptable locations using the various scenarios. This is especially useful where many variants of a certain scene are required. Where the automatically generated placement constraints prove unsatisfactory, placement constraints can be quickly refined by attaching a new, user–specified placement constraint to the affected objects. Since placement constraints can be attached to multiple objects simultaneously, adjusting initial object placements is quite efficient.

Figure 1 showed a scene crafted using CAPS. That particular scene has 300 objects and was laid out in less than 10 minutes: 5 minutes to get a skeletal layout, and 5 minutes to populate and redistribute the rest of the objects. Figure 4 shows a scene with 500 objects, which required 25 minutes to lay out due to the many monitors and chairs which required exact placements. Although more extensive user tests are required to validate these productivity gains, these initial results are very encouraging.

5 Future Work

5.1 Three Dimensional Planning

Because we use footprints to plan object placements, certain effects, such as placing chairs under tables, are currently not achievable (since, in this example, the footprints would overlap). However, this *can* be achieved by performing the spatial planning simultaneously on several horizontal slices of the space. For example, we could take a horizontal slice at the foot of the chair, another in the midsection of both chair and table, and a final slice at the top of the table. CAPS would enforce the 2D constraints within each slice in the usual manner. Placement of an object would be considered safe if and only if placement is safe in each of the slices.

Although it seems natural that an extension to using 3D Minkowski sums and differences could be made, this doesn't seem to be worthwhile. Because objects tend to rest on surfaces, the layout problem is more 2D than 3D in nature. For this reason — and due to their efficiency — we favor 2D techniques over 3D ones. (However, 3D

Minkowski sums and differences could be useful in certain situations, such as detecting the free space under the table where chairs may be placed.)

5.2 Generalized Distributions

Currently, objects are distributed uniformly randomly on a surface once placement constraints are satisfied. Under this scheme, we can view forbidden regions as having zero placement probability density, while safe regions have uniform probability density. A more general probability distribution would provide more control. For example, we could use a Gaussian distribution for the NEAR constraint in order to more heavily weight closer placement. As another example, we could model the placement of forks on one side of a plate by making the probability density of the NEAR constraint very high on that side of the plate. This would permit a fork to be placed in approximately the right position, with a bit of leeway (depending upon the distribution) to add some real–world sloppiness in the fork's placement.

5.3 Semantic Database

When a new class of objects is defined in the semantic database, its CANSUPPORT relations with each already–existing class must be checked. A more powerful formal representation — using abstract classes and inheritance, for example — would be appropriate in a future extension to CAPS. We also need to extend the use of semantic information to deal with functional and non–local dependencies that are currently not considered. A simple example of such a dependency is the strong relationship that exists between the position and the direction of the monitors and the position of the chair in Figure 4.

6 Conclusion

The combination of physics, semantics, and placement constraints permits us to quickly and easily lay out a scene. We have shown that the layout task can be substantially accelerated with a simple pseudo–physics engine and a small amount of semantic information. Future work into generalized distributions and a richer set of semantic information might lead to a new modeling technique, where users can create scenes by specifying the number and distribution of each class of object to be included in the scene.

References

[1] E.A. Bier. Snap-dragging in three dimensions. *ACM Symposium on Interactive 3D Graphics*, pages 193–204, 1990.

[2] A. Borning. The programming language aspects of thinglab, a constraint oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems*, (3):353–387, 1981.

[3] A. Borning, K. Marriott, P. Stuckey, and Y. Xiao. Solving linear arithmetic constraints for user interface applications. In *ACM Symposium on User Interface Software and Technology (UIST)*, pages 87–96, 1997.

[4] D. Bowman and L. Hodges. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. *ACM Symposium on Interactive 3D Graphics*, pages 35–38, 1997.

[5] D.E. Breen, R.T. Whitaker, E. Rose, and M. Tuceryan. Interactive occlusion and automatic object placement for augmented reality. *Eurographics*, 15(3):11–22, 1996.

[6] R. Bukowski and C. Sequin. Object associations. *ACM Symposium on Interactive 3D Graphics*, pages 131–138, 1995.

[7] J. Butterworth, A. Davidson, S. Hench, and T. Olano. 3DM: a three-dimensional modeler using a head-mounted display. *ACM Symposium on Interactive 3D Graphics*, 25(2):135–138, 1992.

[8] B. Coyne and R. Sproat. Wordseye: An automatic text-to-scene conversion system. *ACM SIGGRAPH*, pages 487–496, 2001.

[9] A. Forsberg, K. Herndon, and R. Zeleznik. Aperture based selection for immersive virtual environment. *ACM Symposium on User Interface Software and Technology (UIST)*, pages 95–96, 1996.

[10] P.K. Ghosh. A solution of polygon containment, spatial planning, and other related problems using minkowski operations. *Computer Vision, Graphics, and Image Processing*, (49):1–35, 1990.

[11] S. Houde. Iterative design of an interface for easy 3-d direct manipulation. *ACM SIGCHI*, pages 135–142, 1992.

[12] Y. Kitamura and F. Kishino. Consolidated manipulation of virtual and real objects. *Proceedings of virtual reality software and technology*, pages 133–138, 1997.

[13] Z. Li. *Compaction algorithms for non-convex polygons and their applications*. PhD thesis, Harvard University, Cambridge, Massacusettes, 1994.

[14] J. Liang. Jdcad: A highly interactive 3D modeling system. *Computers and Graphics*, 18(4):499–506, 1994.

[15] D.P. Mapes and J.M. Moshell. A two handed interface for object manipulation in virtual environments. *Presence*, 4(4):403–416, 1995.

[16] V.J. Milenkovic. Position-based physics: simulating the motion of many highly interacting spheres and polyhedra. *ACM SIGGRAPH*, pages 129–136, 1996.

[17] V.J. Milenkovic. Optimization-based animation. *ACM SIGGRAPH*, pages 37–46, 2001.

[18] M.R. Mine. Isaac: a meta-cad system for virutal environments. *Computer Aided Design*, 29(8):547–553, 1997.

[19] M.R. Mine, F.P. Brooks, and C.H. Sequin. Moving objects in space: Exploiting proprioception in virtual-environment interaction. *ACM SIGGRAPH*, pages 19–26, 1997.

[20] G.M. Nielson and D.R. Olson. Direct manipulation techniques for 3D objects using 2D location devices. In *WorkShop on Interactive 3D Graphics*, pages 175–182, 1986.

[21] J.O. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1998.

[22] J. Pierce, A. Forsberg, M. Conway, S. Hong, R. Zeleznik, and M. Mine. Image plane interaction techniques in 3D immersive environments. *ACM Symposium on Interactive 3D Graphics*, pages 39–43, 1997.

[23] Poupyrev, S. Weghorst, M. Billinghurst, and T. Ichikawa. Egocentric object manipulation in virtual environments: Emperical evaluation of interaction techniques. *Computer Graphics Forum*, 17(3):41–52, 1998.

[24] I. Poupyrev, M. Billinghurst, S. Weghorst, and T.Ichikawa. Go-go interaction technique: Nonlinear mapping for direct manipulation in VR. *ACM Symposium on User Interface Software and Technology (UIST)*, pages 79–80, 1996.

[25] W. Robinett and R. Holloway. Implementation of flying, scaling and grabbing in virtual worlds. *ACM Symposium on Interactive 3D Graphics*, pages 197–208, 1992.

[26] M. Shinya and M.C. Forgue. Laying out objects with geometric and physical constraints. *The Visual Computer*, (11):188–201, 1995.

[27] G. Smith, T. Salzman, and W. Stuerzlinger. 3D scene manipulation with 2D devices and constraints. *Graphics Interface*, pages 135–142, 2000.

[28] J.M. Snyder. An interactive tool for placing curved surfaces without interpenetration. *ACM SIGGRAPH*, pages 209–217, 1995.

[29] R. Stoakley, M.J. Conway, and R. Pausch. Virtual reality on a wim: interactive worlds in miniature. *ACM SIGCHI*, pages 265–272, 1995.

[30] W. Stuerzlinger and G. Smith. Efficient manipulation of object groups in virtual environments. In *IEEE Virtual Reality*, 2002. to appear.

[31] I. Sutherland. Sketchpad: a man-machine graphical communication system. In *Proceedings of the IFIP Spring Joint Conference*, 1963.

[32] D. Venolia. Facile 3D direct manipulation. *ACM SIGCHI*, pages 31–36, 1993.

[33] C. Ware. Using hand position for virtual object placement. *Visual Computer*, 5(6):245–253, 1990.

[34] C. Ware and D.R. Jessome. Using the bat: a six-dimensional mouse for object placement. *IEEE Computer Graphics & Applications*, 8(6):65–70, 1988.

[35] K. Xu. Automatic object layout using 2D constraints and semantics. Master's thesis, University of Toronto, 2001.

[36] T.G. Zimmerman, J. Lanier, C. Blanchard, S. Bryson, and Y. Harvill. A hand gesture interface device. *Proceedings of CHI and GI*, pages 189–192, 1987.