The Role of Semantics in Games and Simulations

TIM TUTENEL and RAFAEL BIDARRA
Delft University of Technology
and
RUBEN M. SMELIK and KLAAS JAN DE KRAKER
TNO Defense, Safety and Security

Powerful graphics hardware is enabling strong improvements in both the appearance and the complexity of virtual worlds for games and simulations. However, current practices in the design and development of virtual worlds mostly resemble high-tech variants of traditional handcrafts, resulting in increasingly unbearable design costs.

In this article we state that an essential key to overcoming these problems lies in the enrichment of object models with several kinds of semantic data. We discuss numerous and promising uses for semantic information in virtual worlds, and show, for many of them, how previous results of recent research can be successfully applied. We also identify the fundamental challenges in this new cross-disciplinary area, and point out a number of open issues lying ahead, including the need for (i) a suitable way of specifying semantic data, providing a powerful vocabulary that is useful and usable for all disciplines involved in game design and development; (ii) a seamless integration of semantic data integrated with procedural generation techniques, in order to provide designers with a new and powerful generation of tools; and (iii) a consistency maintenance among evolving objects in a changeable environment, for which powerful constraint-solving methods will be instrumental.

We conclude that, as the expectancy for future games and simulations steadily shifts from improved graphics and appearance towards improved character behavior, plausible realism and coherent gameplay, embedding the game world and its objects with richer semantics is going to play a crucial role. We can therefore expect that, in the near future, increasing research efforts and influential results will be emerging in this new exciting area.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Geometric algorithms, languages, and systems*; H.5.1 [**Information**

This research was supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NOW) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

Authors' address: T. Tutenel and R. Bidarra, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, The Netherlands; email: timt@graphics.tudelft.nl, r.bidarra@ewi.tudelft.nl; R. M. Smelik and K. J. de Kraker, Modeling and Simulation Department, TNO Defense, Safety and Security, The Netherlands; email: {ruben.smelik, klaas_jan.dekraker}@tno.nl.

Interfaces and Presentation]: Multimedia Information Systems—*Artificial, augmented, and virtual realities*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism; D.3.3 [**Programming Languages**]: Language Constructs and Features—*Constraints*

General Terms: Design, Experimentation, Languages

Additional Key Words and Phrases: Semantics, virtual environments, constraint solving, modeling, games and simulations

ACM Reference Format:

Tutenel, T., Bidarra, R., Smelik, R. M., and de Kraker, K. J. 2008. The role of semantics in games and simulations. ACM Comput. Entertain. 6, 4, Article 57 (December 2008), 35 pages. DOI = 10.1145/1461999.1462009 http://doi.acm.org/10.1145/1461999.1462009

1. INTRODUCTION

Rapid advancements in graphics hardware performance are making it possible to support[1] a much higher level of detail in virtual worlds for current video games and simulations, thus increasing the market pressure to boost their complexity and extension. Hence, creating such virtual environments costs an increasingly large amount of time and money. However, many methods currently used in this task are very labor-intensive and, in essence, resemble high-tech variants of handicrafts. They can become very complex when editing huge worlds, as we can observe in the screenshot of the Crytek Sandbox 2 editor in Figure 1. Even worse, the resulting models often consist of little more than annotated geometric representations of the environment and of the objects embedded in it. In particular, much of the designers' intent in guiding the whole design process is not captured anywhere in the model and so gets lost, making it very cumbersome to iteratively modify a design, or simply to reuse elements from previous designs. Finally, using rigid object representations, as for example pure geometric mesh models, further hinders current attempts to achieve object dynamic behaviors or in-game adaptivity in the virtual world.

In the last decade, more and more research has been done on modeling techniques that incorporate extra design information in the representation of objects, thus going beyond the mere geometric data. The successful application of such techniques, for example in the area of CAD/CAM [Bidarra and Bronsvoort 2000], gives us a good basis to conclude that any solution to the above problems will have to include a richer representation of the virtual worlds and of each of its objects. This extra, nongeometric information falls under what we in this article will call *world* (and *object*) *semantics*.

According to Wikipedia, "***semantics*** (Greek *sēmantikos*, giving signs, significant, symptomatic meaning, from *sēma* ($\sigma\tilde{\eta}\mu\alpha$), sign) refers to aspects of meaning, as expressed in language or other systems of signs." Thus in generic terms, *semantics* denotes the meaning of a word or sign. Applied to virtual environments, we define *semantics* as the information conveying the meaning of (an object in) a virtual world. Ultimately, by including semantic information in their representation, we aim at defining life-like behavior for virtual objects. A semantically rich object representation, therefore, goes beyond its basic geometric model by associating it with data that embeds a substantial amount of knowledge about itself and possibly its surroundings. For example, besides

Fig. 1. Editing huge game worlds in the Crytek Sandbox 2 editor[1] can become very complex.

its geometry, an object can have various parameters defining, for example, its appearance, physical properties, roles, behavior and, even more importantly, which services it can provide. Virtually every imaginable object plays some role(s) in its environment: objects offer services to other objects or characters in the world. An object can offer cover for a soldier, a lighter or a matchstick can provide fire, which offers services like heat and light, and batteries, solar cells, or a wall plug can offer power for electrical appliances under specific conditions. Accordingly, some objects will rely on the services of other objects to offer their own services. Expectedly, this association of information will typically result in highly interactive objects and, ultimately, help overcoming the drawbacks pointed out above.

Moreover, when semantic data is linked to procedural generation techniques, the power, quality, and realism of these techniques can be improved as well as extended to automatically propagate semantic information to the procedurally created worlds and objects. This combination will give designers very powerful tools with which they can create larger and more complex worlds, without having to spend a proportional amount of extra time.

In this article we first provide a detailed overview of what semantic information in virtual worlds entails, distinguishing different levels in this

[1]Crytek Sandbox 2 editor is a level editing tool for *Crysis,* from developer Crytek (http://www.crytek.com/).

semantics, and pointing out which kind of information is useful at each level (Section 2). We next discuss how object relationships and rules can be described and maintained in changeable worlds by utilizing constraint-solving (Section 3). Finally, we give an overview of the contribution of semantics to virtual worlds, both in the design phase (Section 4) and in the runtime phase (Section 5).

2. LEVELS OF SEMANTIC SPECIFICATION

Normally, we see an object represented in a virtual world "through the vertices and rendered triangles" of its geometric model. But subconsciously, we transform this "physical object" to an abstract concept, in much the same way as we do in real life: when we look at a tree, we see "beyond" the branches and the leaves and just think about the concept of tree.

However, although most of the reasoning we do focuses on this abstract view of an object, very little of this abstract layer is currently integrated in game worlds. The objects very often remain nothing more than geometric elements consisting of textured triangles in the virtual world.

A virtual world not only needs to look realistic, it also needs to *feel* realistic. A user of a virtual environment application needs to be able to interact with a virtual object just as one would do with the real-world counterpart of that object. A semantically rich world provides the information—the *vocabulary*—necessary to create realistic and compelling interaction.

We distinguish three levels in virtual world semantics. The most basic level is the object level. An object contains a number of physical or functional properties that are specific to that object. On a higher level, we have the relationships between different objects. Objects can, for example, be mutually related by class, inclusion, proximity, coordination, or many other kinds of relations. And on an even higher level, we find more global semantics, as for example time, weather, and climate properties. We see the examples of these semantic levels in the scene of the military simulation game *Armed Assault* in Figure 2. For example, we see the sandbag bunker, providing "cover functionality," the barbed wire which is related to the road to block vehicles, and the location of the checkpoint near the crossroads is one of strategic importance in the world. These three levels of semantics will now be covered in more detail.

2.1 Object Semantics

The first level where a more descriptive approach is needed is the object level. Objects need to transcend the geometry level and more information needs to be incorporated into the object's description. Many properties of real-world objects should be represented in their virtual counterparts to allow an algorithm to perform some kind of reasoning on the objects (e.g., the physical attributes define whether or not the object is too heavy to carry, or the functional information is necessary to decide if an AI character can use the object to reach a goal). Object properties might be basic attributes like physical and material characteristics. But also more high-level elements, like the functionality of the object, its influence on other objects and its interaction possibilities, should be defined.

Fig. 2. A scene from the military simulation game *Armed Assault*[2] that can potentially contain loads of semantic information: the building rooftops are ideal look-out positions from where the troops on the ground can be protected, the sandbag bunker is an excellent cover position, the barbed wire is strategically located to block the road and the checkpoint location near the crossroads is also of strategic importance in the world.

An example of such objects are Kallmann and Thalmann's [1998] "smart objects" which contain information about the possible interaction methods that can be executed on them. Peters et al. [2003] take the notion of smart objects further by creating objects with information about the functionality of the objects, how NPCs (non-player characters) can interact with them, and where important features of the object are situated. The objects include information about the location where a character needs to stand to interact with the object and where he or she should look at while performing a specific action (e.g., when interacting with a computer, the character should look at the screen while typing something on the keyboard).

As more information is represented, the design of such objects will initially take more time and effort than the usual design of plain geometric objects. To avoid having to define each property of each object in the virtual world, Ibanez-Martinez and Delgado-Mata [2006] introduce a system that defines object data more efficiently. They introduced an architecture that provides a general common level on which to build specific semantic representations. The common-level data bundles information that is useful in all virtual environment applications, such as the position and dimensions of an object. Another advantage of this approach is that it takes work out of the hands of designers

[2]*Armed Assault* (http://www.armedassault.com/), from the developer Bohemia Interactive (http://www.bistudio.com/).

by automatically calculating some of these application-independent properties (e.g., the dimensions of an object based on the object model). This decreases the time a designer needs to spend manually annotating features. To remain application-independent, the common level only contains low-level information and the designer can later add more specific data, depending on the application. Basically, the user needs to enter the object type, which is used to generate properties like width, height, location, and orientation automatically, based on the geometric model of the object.

Techniques like these, that reduce the amount of manual annotation or tweaking of object properties, are very important, since the addition of semantic data should not have a negative influence on the time and effort designers spend creating their virtual worlds. Many aspects such as object dimensions can be generated based on the geometry. But we need even more properties than dimensions and orientation only in order to create a powerful system. For example, by only defining a material for an object, a substantial amount of physical properties can be automatically derived. A more complex design system is necessary that includes methods like these in order to create semantically rich virtual worlds without significantly increasing the design time. In the next sections we will show that, in many cases, semantically rich objects actually save much time in the design process as well, so the time necessary to add this semantic information is well spent.

Objects containing interaction information, for example the smart objects by Peters et al. [2003], could aid in the creation of vast and highly interactive virtual worlds. If designers could be provided with fast and efficient tools to incorporate actions in objects, game worlds could be created that are completely interactive. For this, all objects should contain information about the ways a user can interact with them: what actions can the user perform on the object (e.g., pick up, throw, drink, read, wear) and what other objects are necessary to perform this interaction (e.g., a key to open a door or a striking surface to light a matchstick). When performing an action on an object, this object will provide services to the user or to its surroundings. Wearing a coat will provide warmth only to the one who is wearing it, but a campfire will provide warmth to everyone within a certain range. A detailed classification of these properties and services is necessary to explicitly define them. It can also be integrated in tools to quickly edit the properties of an object by allowing the designer to pick a related object class. From this related class, information can be inherited to eliminate the need to define every detail over and over again.

2.2 Object Relationships

Objects in a scene contain certain relationships between each other. These relationships can be between instances of objects, for example, geometric relationships in a scene, or between object classes. Different objects may share similar properties (e.g., have the same material, but they may also be similar on a functional level, for example a candle and a flashlight can both offer light).

A first and important relationship is that of inheritance. Two different objects that have the same ancestor will share some similar properties but will be in another child branch in a hierarchy tree. This kind of relationship can be expressed in a taxonomy, of which a very detailed example can be found in the feature classification model of Bitters [2002]. But next to a hierarchy based on properties, more detailed relationships can be defined.

In Levison's [1996] Ph.D. dissertation, we see an example of a functional hierarchy. The dissertation describes a system that enables the decomposition of a high-level task into a set of action directives. The objects have information about how they can be manipulated. They contain "sensible" knowledge, which is basic knowledge like its size, position or temperature, but also "symbolic" knowledge (e.g., about its functionality). The system uses a hierarchical class structure where every child class inherits all functionality of its parent classes. The lower a class is in the functional hierarchy, the more specialized the functionality becomes. An example of this is the class *Thing*. This class is broken up into *Object* and *Artifact*. *Artifact*, in turn, is subdivided in *Tool*, *Container*, *Cover*, and *Support*. *Artifact* is the category that defines all man-made objects. *Tool* is a tool that contains a function it can perform, for example a hammer or a screwdriver.

This kind of information about the functionality of an object is very useful in virtual environments. When creating a virtual world, the functionality of an object can help in deciding which objects should be placed where. When a level designer creates (e.g., a workplace of a furniture builder as a part of the game world), his workload could be significantly reduced if the world editor contained a method to logically place some woodworking tools in that virtual workplace. Obviously, the interaction part of the game could benefit from this functional data too, as we explained in Section 2.1.

Huhns and Singh [1997] use ontologies with different types of relationships between objects. Next to the basic data-modeling relationships, that is, inheritance, aggregation, and instantiation; they also cover relationships like *owns*, *causes,* and *contains*. The system for which they proposed this idea was intended to handle communication between software agents with different knowledge domains. This could be a customer agent communicating with a travel agent about a trip. The travel agent can give details about a certain flight with a 777 type plane that might not be present in the knowledge base of the customer agent. But because of the relationships, the travel agent can inform the customer agent that this 777 has *airplane* as a parent object, and since the customer agent will have the general term *airplane* in its knowledge base, he will understand the information of the travel agent. This kind of information can be used by the AI system of a game or simulation, but it could also be a source to provide meaningful and life-like interaction to the user. For example, when the user needs to find a power source and the virtual world contains information that a car contains an engine and that an engine is a type of a power source, this car engine could be used without the designer having to define this explicitly. In this way, virtual worlds become more interactive, and can thus lead to more emergent behavior.

2.3 World Semantics

Next to the object level we can also define semantic information on a more global level. To identify vegetation types inside a region, parameters like soil fertility, and soil nutrients play a role but also climatic circumstances like temperature and rainfall should be taken into account. These parameters are not related to specific objects but to an area in the world or perhaps the entire world.

We see this approach in the work of Deussen et al. [1998] that consists of an ecosystem simulation model to populate an area with vegetation. The input of this simulation model is terrain data, ecological properties of plants, and, optionally, an initial distribution of plants. Based on this data and taking into account rules for competition for space, sunlight, and soil resources, a distribution of plants inside an area is generated. When a designer has the ability to include these kinds of properties in the world, many physically-based models could integrate them to create a more realistic world.

Semantic world data is also very useful when designing adaptive virtual worlds. These are worlds that adapt to global parameters (e.g., weather conditions and time) or evolve based on the user interaction (e.g., clearing forests and constructing buildings) in strategy games. Global game world parameters like weather and time have a huge influence on the virtual world in its entirety as well as on the individual objects. When we look back at the vegetation example above, we see that an ecosystem model could, for example, also be applied to calculate changes in a virtual forest. Trees become older, new trees and plants appear, and others die. In a strategy or empire-building game where resources need to be gathered from forests, this could add not only realism but also strategic difficulties (cutting down trees faster than the growth of the forest will deplete wood resources). Depending on seasons, the geometric appearance of plants and trees change but also their properties, for example the resources they offer. In summer time, fruit grows on trees and can be gathered to feed the population in a strategy game. Corn fields can create an excellent hiding place from enemies when the crops are fully grown.

Weather and time also have an influence on materials, for example surfaces begin to reveal cracks in the paint, unprocessed wood becomes weaker, and metallic objects start to rust. These parameters can be taken into account just to add visual realism: for example, modeling paint cracks [Paquette et al. 2002] or aging and weathering effects on textures [Lu et al. 2007]; but at the functional level can also alter the role of an object in a game. A young, strong, wooden plank will be perfect to attack an enemy while an old, mossy branch will break more easily, and is therefore useless as a weapon.

Finally, we can also include contextual information in world semantics. Examples are parameters like the economic or living conditions or how safe inhabitants feel. This kind of information is more important in games such as city-builders or strategy games; but also in military simulation safety levels, and, in general, the state of the global economy can play a role as well.

2.4 Discussion

Careful consideration needs to be given to creating the methods for generating and editing semantic data. For this, the use of a system as that proposed by Ibanez-Martinez and Delgado-Mata [2006] can be interesting, since it tries to automatically generate numerous values of the semantic data level to reduce the design time. The types of calculated data should however be extended. If a designer did not only choose the object type but also the material of the parts, the weight and other physical properties of the object could be generated. This way, without the designer having to enter a huge amount of parameters for every single object, the game would know, for example, if an object picked up by the player's character is heavy enough to injure an opponent (and if it is not too heavy to pick up in the first place).

Algorithms simulating physical phenomena, as for example, the ecosystem simulation model of Deussen et al. [1998], can be significantly improved in semantically rich worlds. When detailed information about the soil of a terrain, the climatic conditions, the growth cycle of the plants, and information about other plants in the environment are included into these kind of algorithms, a more realistic effect can be acquired.

In this section we discuss various forms of object and world semantic data useful for inclusion in virtual worlds. Obviously, not all information is of practical use in every game genre or in every simulation, but any flexible system should at least enable the integration of a variety of this data, from physical attributes to contextual properties.

Once semantic information has been incorporated into such rich object representations, we face an even bigger challenge in the maintenance of that semantics as virtual worlds evolve. Similarly, relationships that were defined among different objects need to be maintained when changes are made to the world. These changes can take place not only during the design phase (e.g., when area parameters are changed or objects are moved), but also at runtime in adaptive worlds, for example an explosion can break the structure of an object and affect its services. Moving objects might result in conflicts, since specific physical laws are perhaps not met in the new location, for example a tree that is moved to an area that has too high a slope or has too few soil nutrients to feed such a tree. To maintain these laws, a system is required that can in real-time maintain relationships and detect and possibly solve conflicts between objects or between an object and the environment. We believe that such a solution can be found with the help of constraint-solving techniques. In the next section we will discuss the close connection between semantic data and constraints and review how constraint-solving techniques apply to virtual environments.

3. MAINTAINING SEMANTICS USING CONSTRAINTS

Since semantic relationships between objects need to be maintained while editing a scene, many objects will be influenced by other objects. For example, when a road with some lamp-posts alongside is displaced, all lamp-posts should likely be moved as well. And if the road is made longer, more lamp-posts will need to be placed alongside the road. So together with the specification of semantics in

the virtual world, we also need mechanisms that maintain this semantics in a changing environment.

The most effective and promising approach to handle this is to use constraint-solving methods. Relationships between objects, but also the behavior of an object, can be expressed in rules or constraints. However, since semantic data is a very broad subject, these constraints can significantly vary from each other. This section discusses what types of constraints are necessary to express the semantics we need in virtual environments and the techniques that currently exist to solve these types of constraints. We also review where constraints are currently being used in virtual environment applications.

3.1 Constraint Types

In the previous section we described three levels of semantics: object semantics, object relationships, and world semantics. All these aspects can be expressed with the help of constraints, which are rules on one or more constraint variables that need to be satisfied. We distinguish a number of different constraint types. First of all we have basic *numerical* constraints: algebraic equations that need to be satisfied. We also need *geometric* constraints that express geometric concepts like the distance between objects or an object's dimensions. Certain physical properties of an object can be calculated with the help of *physical laws,* and can often be expressed in an algebraic equation with extra parameters, for example gravity or temperature. Properties of objects can also change over time, and for this we need *time* constraints. In some cases these can be incorporated into the physical law constraints by making time a parameter in the physical equations; but there should also be conditional rules that express whether or not a property or a certain behavior is available on a given time or date. Finally, we have *distribution* or *variation* constraints. These express how many objects in a given area need to be present or at which ratio different objects should be distributed (e.g., a forest containing 40% pine trees and 60% birch trees). We now review the three semantic levels from the previous section and what constraint types are particularly interesting on each level.

Semantic properties of individual objects can be expressed with time and physical law constraints. If the material properties of an object are specified, physical law constraints could be deduced from them, like the degradation of material over time or the maximum load an object can bear. Time can also influence the services an object provides. In the previous section we gave the example of trees that produce fruit during summer and corn crops that offer cover when fully grown. This can be expressed in a conditional time constraint.

On the level of object relationships, constraints can really prove their worth. At the beginning of this section we discussed the hierarchical structure of a road concept. When describing a road, many constraints can be applied. We want to express, for example, the width of the road and the pavement or the distance between the lamp-posts to the side of the road. These are geometrical constraints, which actually form the basis for constraint solvers in virtual environments because finally we need to define positions and orientations for objects in the world; but also numerical constraints like how many lamp-posts per kilometer

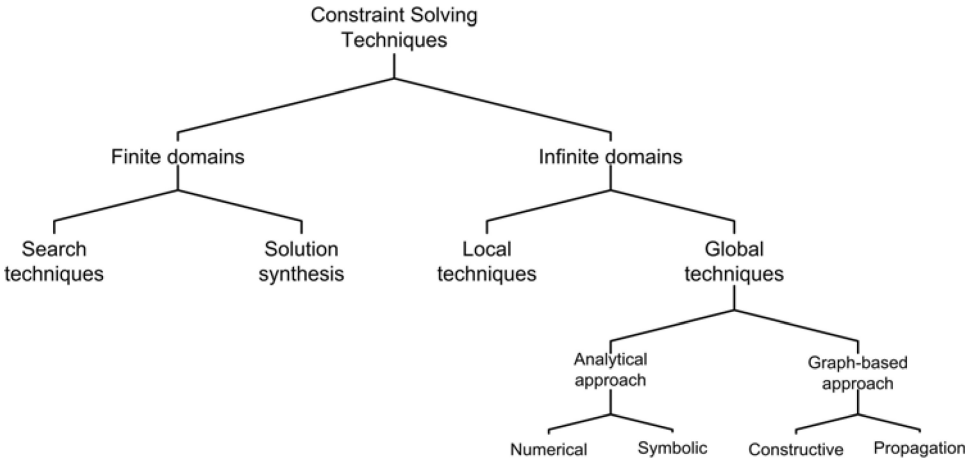

Fig. 3. A schematic overview of the constraint-solving techniques.

of road are necessary. We could also transform this to a physical law by expressing how much lumen is required for the road. This is an easy way to express that we want more lamp-posts or more powerful lamps on larger roads or roads with multiple tracks. Other examples of numerical constraints are the number of rooms a building should contain or how many windows per room.

To express the world semantics level, different types of constraints can be applied as well. The vegetation of an area can be expressed with variation and distribution constraints. However, it is also possible to use physical laws to create an ecosystem simulation, as described in the previous section. Another example of how physical laws can be helpful at the world level is adaptation to weather conditions. A region with an earth surface will become mud when rain starts to pour. The properties of this surface will therefore change due to the transition from earth to mud. When soldiers run through the mud, it will be harder and slower for them to do so, and they will be more prone to falling.

We discussed the different types of constraints necessary to express semantic information in virtual worlds. In the following section, an overview is presented of what techniques can solve these constraints.

3.2 General Constraint-Solving

A constraint-solving technique tries to find some or all solutions for a set of constraints. Many of such techniques have been developed in the past; we will cover the main categories [Dohmen 1995; Hoffmann and Joan-Arinyo 2005]. A schematic overview of the different approaches is found in Figure 3.

The first major division can be made between constraint variables with finite domains and those with infinite domains. Finite domains have the advantage in that, in the worst case, every possible solution can be evaluated. The finite domain constraint problems are mostly solved in two ways: via search techniques and solution synthesis [Tsang 1993]. The search techniques are based on backtracking algorithms in which a possible value is chosen for a variable, and

recursively every variable is filled until some constraint no longer holds. Then the algorithm steps back and chooses another value for the variable. Especially when just one solution needs to be found in a loosely constrained problem, this technique is adequate, since few backtracks will be necessary to find a solution.

In solution synthesis, a solution for all constraints on a subset of variables is generated. With this solution, a new solution is determined for the constraints of all variables in the first subset plus one new variable. By employing this method, all possible solutions are found.

Infinite domain problems require a different approach. Since variables can have infinite values, a problem can potentially have infinite solutions. When the constraints consist of algebraic constraints, a solution can be found by looking at the degrees of freedom of each variable [Leler 1988]. When a variable is restricted by many independent constraints, it is said to have few degrees of freedom. The variables with few degrees of freedom are solved based on already known variables and connected constraints. After this, the results are propagated to the variables with more degrees of freedom. This is a local technique, since it tries to solve a problem by solving local sub-problems first.

In problems that contain loops, these techniques do not suffice. A constraint problem can be expressed in a graph where the variables are nodes and the constraints are the edges between the nodes. If a graph that represents a problem in this way contains a loop, the local propagation does not work anymore because some variables can be determined by multiple paths in the graph. To handle these problems, global techniques are necessary.

The global techniques can be analytical or graph-based. The analytical solvers can be further divided into numerical and symbolical solvers. An example of a numerical technique is relaxation [Hillyard and Braid 1978] that starts with initial guesses for variables. These guesses are assessed on their errors with each constraint and new guesses are made that eventually converge to a solution for which the errors are within an acceptable range. Other numerical techniques include Newton-Raphson and homotopy continuation. Both methods can find the roots of a nonlinear function. The Newton-Raphson method starts from an initial guess and then iteratively calculates closer approximations. These approximations do not necessarily converge to the solution however, so the method does not always find a suitable result. Homotopy continuation techniques [Choi et al. 1996; Lamure and Michelucci 1996] try to find the roots of a set of nonlinear functions by creating an interpolation function between this set and another set of functions close to the original set for which the solution is known.

Symbolic solvers use algebraic elimination rules to reduce a system of polynomial equations to an easier one. An often-used technique is transforming the system into an equivalent triangular system, called a Gröbner basis, with the same solution as the original system, but which is easier to solve with back-substitution and univariate root-solving [Buchberger 1985; Kondo 1992].

Many different graph-based-solving techniques have been researched. The constructive methods (e.g., Bouma et al. [1995] and Hoffmann and Joan-Arinyo [1997]) are often used to solve constraints in computer-aided design, since CAD drawings can usually be expressed by a sequence of basic construction steps.

The constraints are satisfied by incrementally constructing the drawing. The top-down techniques recursively split a construction problem into smaller problems until a problem is found with a known solution. The bottom-up techniques start by solving subproblems and then combine these to solve the main problem.

Other graph-based techniques use a propagation approach. In degrees of freedom propagation [Glenn 1991], the degrees of freedom limited for each constraint are fixed and the remaining degrees of freedom are propagated to the other constraints.

Geometric constraints can also be defined as specifications of geometric operations. These geometric operations are executed incrementally and independently of other constraints. They only need the results of previous operations to execute the next operation on.

When we take a look at the previous section, we recollect that many different constraint types are necessary in systems that maintain semantics. We not only need geometric constraints but also numeric constraints. Some of the variables will have finite domains, yet others infinite domains. Physical law constraints especially could contain constraints that influence many variables at once. But the end result will be a modeled scene, hence the solver will probably be geometrically based, that is, all objects need to get a fixed position in the resulting scene.

The combination of different constraint types has been achieved in some generic constraint solvers. In his PhD dissertation, Fernández [2004] (see also Fernández and Hill 1998]) explains the necessity for a generic constraint solver for logic programming. The main disadvantage he indicates for specialized solvers in constraint logic programming (CLP) applies to constraint-solving in virtual environments as well. When a constraint type is not supported by a solver, it needs to be transformed into one that is supported. Perhaps the supported domains can only handle finite sets or perhaps the possible constraint types are too limited. A generic system that allows the addition of new constraint types (as we discuss next) is therefore more declarative, since domains or types that are not yet supported can be added to the solver.

Fernández proposes a unified framework that includes all the domains for which specific CLP systems are available (integer values, Boolean values, real numbers, and sets of values). But the system also provides for a glass-box constraint system in which it is easy to add new customized constraints by the user. Moreover, for these custom constraints, the user is not limited to the domains that were integrated in the system, but can also define new domains and, for each new constraint on these new domains, specific propagation behavior. In Figure 4 we can see how this double glass-box approach is structured. This approach allows for much more flexibility, as opposed to previous black box approaches. The implemented system does have an important drawback, however: using logic programming to solve geometric constraints is significantly less efficient than other methods specifically targeted to the geometric domain, and, according to the author, the architecture used for the implementation is not the most efficient one either. The idea of a glass-box system is interesting, but this logic-based implementation is inefficient in the context of semantic virtual world constraints.

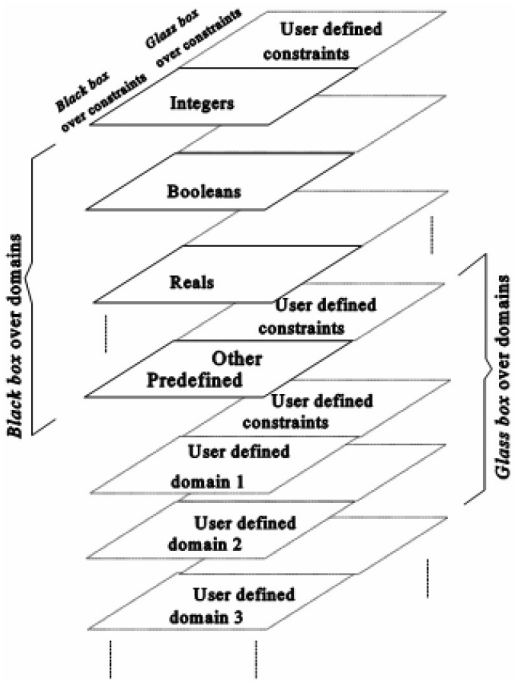

Fig. 4. The glass box approach proposed by Fernández over both constraints and domains. (Figure 1.3 from Fernández [2004]).

In geometric modeling applications the generic constraint-solving idea is useful as well, as shown in the work of Le Roux et al. [2001]. In their solver for a declarative modeling application, both constraints and domains are generic. The search algorithm uses a nonrecursive backtracking system and generic dynamic heuristics. To test the system, they created a scene with 30 objects (a TV, walls, tables, etc.) with description constraints such as "the screen is on the table, table1 is to the right of table2" etc. The description was internally translated into 210 variables and 245 basic constraints. Their paper mentions that to allow inequality constraints, they chose a voxel array, which obviously does not scale well, and hence is not immediately usable.

These and other systems show that a generic solution is flexible and offers a significant advantage in ease-of-use over specific solvers; but the main drawback of this approach is the loss in efficiency. A logic-based system is quite slow in solving geometric problems. Specialized algorithms can obviously find solutions faster since they can apply heuristics or algorithms only usable on a specific domain for example. The general idea behind these systems is useful to keep in mind, but the loss of efficiency needs to be overcome, since the amount of constraints in a virtual world can become very large.

3.3 Constraint-Solving in Virtual Environments

In the scope of this article, the field of declarative modeling is interesting because it combines constraint-solving and modeling in virtual environments.

Gaildrat [2007] provides an extensive overview of issues in declarative modeling Declarative modeling of virtual environments tries to offer a simple design process for nonprofessional users that can help them create complex scenes based on specific rules (e.g., create scenes in a specific architectural style). To achieve this, it uses both constraints and semantic knowledge, that is, implicit constraints, to guide the user. This semantic knowledge is represented by pragmatic and functional properties of the objects.

Gaildrat considers three phases in these modeling systems; but the description and generation phases are the most interesting in the context of this article. In the description phase, the designer uses different types of properties to express his mental image of the scene. Semantic knowledge can solve the ambiguity and imprecision of the description, for example, object positions can vary according to context. Plates on a kitchen counter will be stacked waiting for the washing up, but on a dining table they will be spread out in front of each chair. To achieve this, practical and functional knowledge is used in the generation process. The descriptions need to be translated into a set of constraints in order to find a solution. Le Roux et al. [2004] reviewed a number of possible constraint-solving methods, and concluded that the search tree approach is well-suited for problems in the generation phase. The reasons given are the flexibility of the approach, the fact that it can generate all solutions, and the possibility of user intervention in branch pruning.

In scene composition and manipulation, constraints are applied as well [Goesele and Stuerzlinger 1999; Xu et al. 2002]. Xu et al. present a system that helps a designer with the placement of objects in a scene by using pseudophysics and placement constraints based on a hierarchical database in which objects have an orientation constraint with respect to their parent object and have properties indicating whether or not these objects can physically support other objects or be supported by other objects. Goesele and Stuerzlinger [1999] use a similar system, in which hierarchical groups of objects are created that can maintain their local constraints when the group is manipulated. These scene-composition techniques are discussed in more detail in Section 4.1.

3.4 Discussion

Considering the different domains and constraint types required for virtual world modeling, it is obvious that a powerful, integrated constraint-solving system that can cope with different kinds of constraints is necessary.

As discussed above, a generic constraint solver could be the solution, as it would not only allow different types of domains (finite-infinite domains, 2D- or 3D-domains, etc.), but also different types of constraints. Together, these could provide a robust and flexible vocabulary, since implicit constraints and custom domains can then be described more easily. The great disadvantage of this generic approach is execution speed. A specialized solver can utilize certain characteristics of a specific domain or of a specific constraint type to find solutions faster than its generic counterparts.

Another possible approach might be a high-level tool that provides a designer with different types of constraints to express the rules and behaviors of

the virtual world and its objects, while internally transforming all constraints to a specific type. In virtual environment applications, the most likely candidate would be the geometric constraint type. This would mean that existing geometric-solving methods can be employed [Bouma et al. 1995; Glenn 1991; Hoffmann and Joan-Arinyo 1997] and that more implicit constraints should be mapped to constraints allowed by these techniques. This could keep the advantage of flexibility for the user, but also have the speed advantage of specific solvers. However, it is unlikely that the same level of flexibility can be provided as with a generic constraint solver. In fact, even in a generic approach, the implicit to explicit conversion is still not completely solved [Gaildrat 2007]; hence it is clear that it would be even more difficult when a more limited range of explicit constraint types is available.

There are several characteristics of modeling virtual worlds that alleviate part of the problem of the large amount of constraints and their complexity. For one, the geometric hierarchy between object instances makes it possible to find the solutions hierarchically. For example, there will be some constraints between terrain elevation and soil properties and the buildings on the terrain. Inside the building, there will also be a number of constraints among the rooms, walls, and other objects. But since the interior constraints are not directly influenced by the exterior constraints between terrain and buildings, it is possible to first decide on the location of the buildings by using the exterior constraints and then move on with solving the insides of the buildings based on interior constraints.

Overall, a good solution for the constraint-solving integration problem, providing all functionality described in this section, still has to be found. Techniques used in declarative modeling approaches do provide a good basis, but the issues on transforming implicit to explicit constraints are far from completely solved. Moreover, the efficiency of generation methods still needs significant improvement.

4. APPLYING SEMANTICS IN THE DESIGN PHASE

When designing a virtual world, for example in a level editor for games, semantic data can alleviate the process by automating tasks of the modeling process. The constraint-solving techniques, discussed in the previous section, can maintain the relationships in the design phase and thereby support the designer so he or she can do the work better and more quickly.

But semantic information can also be applied when generating a scene procedurally. More realistic scenes could be created if the generation algorithm took the properties and constraints of the objects that need to be placed into account.

4.1 Manual Scene-Editing

Geometric constraints have been regularly used in applications where objects need to be placed or moved to speed up the design process. Besides the typical physical constraints, Xu et al. [2002] use semantic-based constraints as well. The system has a semantic database. Each object in the scene belongs to a

Fig. 5. A scene of 300 objects laid out in less than 10 minutes with the help of the Constraint-Based Automatic Placement Systems or CAPS. (Figure 1 from Xu et al. [2002]).

specific database class that can contain parent classes and child classes. The objects of the parent classes are those that can physically support the class, and the objects of the child classes can be supported by the class. A plate is an example: its parents are tables and counters and its children are food, utensils, and plates. Two flags define whether or not the class can support objects not in the child classes or can be supported by objects not in the parent classes. The example scene in Figure 5 contains 300 objects, but was laid out in less than 10 minutes with the help of this system.

Smith et al. [2001] use *offer* and *binding* areas on objects. The binding areas can be connected to offer areas of other objects. These areas do not need to be part of the object geometry. By using what they call "virtual constraints" an empty area of an object might be a binding or offer area. An example is the space underneath a table between two legs, where chairs can be put. This space can thus be an offer area where the binding area of the chairs can be constrained. These constraint types lead to a hierarchical structure, that is, constraints between parents and children, but some properties of objects need constraints between siblings. A couple of chairs in a waiting room (which would be all children on the floor of the waiting room) typically stand next to each other in a line. When one of the chairs is moved, the others need to follow. For this behavior, dual constraints are offered that can express constraints between siblings.

The WordsEye system [Coyne and Sproat 2001] is a text-to-scene conversion system that allows the user to describe a scene using natural language. The descriptive sentences are converted into constraints between the properties of the objects in the sentence. The system contains an extendable database of objects with a geometry file and other properties. These properties might

Fig. 6. Both the plants and trees in a terrain and the roads and buildings in a city can be realistically modeled with procedural generation techniques: (left: Figure 16 from Deussen et al. [1998]; right: Figure 18 from Müller et al. [2006b]).

be a skeleton-structure for humans and animals, shape displacements (e.g., emotions on a human face) or object parts that might be used in a sentence (e.g., a car contains headlights, doors, etc.). Another property is the color part. This is the part of the object that should be colored when the user defines a color (e.g., a blue flower should have blue petals, but the stem should remain green). Every object also has a default size, opacity parts (the glass of a window) and functional properties (a car or a bike are road vehicles, so when a user says "John rides to town," one of the road vehicles should be chosen). Finally, there are the spatial tags, which are 3D shapes that define areas of an object like "under," "on top of" or "in". When spatial relations are applied using these spatial tags, ambiguous results may be achieved. When someone says "the elephant is under the chair," he or she may mean that a small elephant is somehow underneath the chair, but may also mean that the chair is on the back of the elephant. This kind of ambiguity is solved in a way that needs the least resizing.

We see that many of these techniques already use semantic information about objects to aid the designer in placing objects in a 3D scene. The objects that are placed are instantiations of classes, and the properties of these classes can be constrained. Spatial relationships between objects are frequently used, too, since they avoid having to move every single object when they are somehow linked together. And as we saw in the WordsEye system [Coyne and Sproat 2001], these semantic properties can also enable the user to design the virtual world in a descriptive manner.

4.2 Procedural Scene Generation

Numerous techniques exist to algorithmically generate parts of a virtual environment. The encompassing term for these techniques is procedural content generation. There are algorithms that can generate terrains and add plants, trees, and rivers to it as well. Streets, buildings, and interiors can be created automatically as well. In Figure 6, we see some convincing results of techniques to procedurally model vegetation in terrain and roads and buildings in a city.

We will review where semantic data assists these techniques and where it could improve them.

The basis for a terrain is often an elevation or height map. Numerous fractal techniques (e.g., Perlin noise, midpoint displacement) have been used to generate a random but realistic looking height map. Other techniques try to mimic real physical phenomena. Some take a fractal height map as a basis and transform it with erosion algorithms based on wind or water streams. Some techniques also work the other way around: they create a river network and create a height map based on that network.

Erosion techniques can be implemented with the help of cellular automata [D'Ambrosio et al. 2001; Musgrave et al. 1989]. Musgrave et al. [1989] use this to simulate different erosion types. For instance, hydraulic erosion is caused by rain falling on the terrain, flowing to lower areas, and causing erosion. Soil information is used to create more realistic results: water flow will influence areas with a soft soil the most. Another type of erosion is thermal weathering, which is a process where surface material from higher areas falls down on lower areas, thus creating taluses. Semantic terrain data on the hardness of the soil and the depth and hardness of the bedrock could improve this technique further. D'Ambrosio et al. [2001] also use many different types of soil data in their cellular automata model, including parameters like vegetation density and water depth. It is obvious that to procedurally create a realistic landscape, this semantic information is vital.

To utilize these erosion techniques, the terrain generation algorithms should generate more than just elevation data and terrain types; detailed soil information including the parameters of the soil material and the bedrock should be generated as well. The same semantic information could also be applied to the distribution of vegetation, as we already explained in Section 2.3. In general, we can state that most procedural generation techniques that mimic physical behavior behave optimally in semantically rich virtual environments. Moreover, this data will also be useful in automated techniques for human-made structures like roads or buildings.

To add roads to a terrain, numerous and various techniques have been researched, of which we discuss four. The simplest technique is the squared grid [Greuter et al. 2003], which basically generates streets in a grid that result in square blocks, which are subdivided into building lots on which buildings can be placed. Displacement noise can be added to the grid points to create a less repetitive network.

Another approach is template-based. Sun et al. [2002] generate a road map based on different road patterns, and for each pattern a corresponding template: a population-based template, a raster and radial template, or a mixed template. The main arteries of the road map are the highways. They cover the whole city and bear the main transportation flow, and always take on certain patterns. They are then checked for validity (tracks over oceans are discarded, diverted, or kept and transformed into a bridge). Roads are often curved naturally to avoid large gradients or obstructions due to different altitudes.

Parish and Müller [2001] use a technique based on L-systems to create their road network. The L-system is goal-driven and the goals are the population

density (i.e., the roads try to connect places with large populations) and specific road patterns. Examples of such patterns are the raster or checkerboard pattern or the radial pattern.

Instead of using L-systems to create a road network, Lechner et al. [2003] take an agent-based approach, which also divides the city into components. Besides the common components like residential, commercial and industrial areas, it also has special components like government buildings, squares, and institutions. It takes a terrain as input and starts off at a seed position. Then the two agents, the extender and the connector, start their work. The extender tries to find unconnected areas in the city. When it finds an area that is not yet connected to the existing road system and is not too far from the existing road network (to avoid placing roads where no buildings will be placed), it will find the best path to connect the area with the existing network. The connector agent will start from a point on the existing network and will randomly search for another spot on the network within a certain radius. It then checks how far one needs to travel on the existing network to reach that randomly chosen spot. If the travel time is too long, a direct road segment is added to that spot.

We noticed above that many different approaches are used to procedurally create a roadmap. Almost all of them need data about the terrain in order to avoid, for example, very steep roads or roads on rivers and lakes. Sun et al. [2002] and Parish and Müller [2001] also use information such as population density to generate roads between highly populated areas. But the information on population density could also help decide the number of lanes or the automatic placement of speed bumps and other speed-limiting structures in residential areas. In the previous section, the idea of using geometric object relationships to place roadside objects like lamp-posts was discussed. In combination with procedural generation of the roadmaps, this yields powerful tools with which terrains can be automatically fitted with a road network complete with numerous roadside objects. The number of lanes can influence the types and number of lamp-posts; the layout of the roads can be used to place traffic lights and road signs automatically; and the placement of bus stops, mailboxes, and parking spots could be decided based on information like the population density and the type of district.

The procedural road and city techniques define the areas where buildings need to be placed; the buildings can be generated procedurally as well. Most of these techniques use basic building blocks to generate the buildings. The method described by Yong et al. [2004] uses building components to create vernacular–style southeast Chinese houses. They use a grammar consisting of semantic components. The system divides the urban area into hierarchical components such as streets, gates, windows, walls, and roofs. Through a number of control rules valid buildings can be created in a realistic building style. The rules define, for example, how a component like a wall can be split up into smaller components with window or door components in between. Another rule will determine how a street is divided into a road with buildings on both sides of the road. By applying these grammar rules, a typical ancient southeast Chinese town can be created rather realistically, since this style of town is very rigidly structured.

Müller et al. [2006a, 2006b] start with a union of several volumetric shapes which define the basic outline and the outer walls of the building. The walls are then divided into floors and the facades of these floors are subdivided into walls, windows, and doors by means of a grammar system. In a final step, the roof is constructed on top of the building.

Another example of a shape grammar approach is Wonka et al.'s [2003] Instant Architecture system. This technique uses a 3D design grammar that allows for a uniform framework to design the entire structure. A specific building style can be acquired by setting this as an attribute of the start symbol for the building. The attributes are propagated to the subparts of the building. Sometimes a style needs to be different within one building (e.g., an apartment building with shops on the ground floor), but usually the same style is propagated to the entire building. Although small differences between floors are necessary, the subparts of one floor need to be coherent. On the end symbols, these attributes will be transformed from a high-level building style to very specific material information. Hence this technique is very useful when one wants to create semantically rich models.

The shape and look of a building cannot be chosen at random, however. There are physical properties that need to be considered (e.g., is a building of a certain height suited for placement on a specific soil type); but attributes like the purpose of the building can also influence its appearance, and some techniques already take this into account. However, there are other parameters that are (currently) missing. As an example, to decide what type of buildings should be placed and how they should look, we could take the wealth of the residents of a district into account (e.g., impoverished neighborhoods could have old or damaged row houses while wealthy neighborhoods could have smart-looking villas). As we mentioned, the purpose of a building influences the outside appearance and structure, e.g. more windows in an office or balconies on apartment buildings, but it has even more influence on the interior layout of the building.

To generate the interiors of buildings, some techniques that work gradually have been developed. They start off with a rough shape and step-by-step fill it with smaller parts [Hahn et al. 2006], or they start from small basic units and combine them to generate the entire building [Martin 2006; Rau-Chaplin et al. 1996; Rau-Chaplin and Smedley 1997].

The LaHave House Project [Rau-Chaplin et al. 1996; Rau-Chaplin and Smedley 1997] uses a shape grammar and a tile library to create a design engine that can generate a library of many different base house designs. The tiles in the tile library are individual rooms and walls. The system uses a hierarchy of shape grammars for each stage of the building. The base element is a tray which is a combination of atomic elements such as machines, rooms, bays, and end-bumps. A combination of different trays forms a floor-plan, which is transformed to a zoned plan by the Functional Zoning system that, for example, identifies the public and the private parts of a house. The public part is situated near the front entrance and includes rooms like the hallway, the kitchen, or the living room, while the private part of a house contains the bathrooms and bedrooms. Next, the number of floors, the width of the zones, and the roof shapes

are added to the plan. Finally, a specific function is assigned to each area (e.g., kitchen, study).

Hahn et al. [2006] describe a lazy generation scheme that allows a seamless walkthrough of a building interior and the use of only a fraction of the memory that a model of the entire interior would require. The technique generates only those rooms of a building that are visible in the current camera view. There are a number of generation rules that are used by the system. Examples of such rules are that every region should be accessible because dead space will only confuse users trying to explore the entire building. In most cases, private rooms should be accessible from a public room or hallway.

The generation process takes place in multiple stages. First the building setup is generated, including everything that affects multiple floors, like elevators or staircases. Then this setup is divided into uniformly spaced floors, and each one gets hallways assigned. These can be straight segments or rectangular loops when a floor is wide enough to allow it. The remaining space forms room clusters. Hallways are added until the room clusters are small enough. The room clusters are then divided into rooms and objects are placed inside them.

The work of Martin [2006] on building procedural interior generation is based on graphs. Each node corresponds to a room, and each edge corresponds to a connection between rooms (e.g., a door). To generate the graph, a grammar is used with a user-defined rule set to fit the local and global semantic room information. First, undefined public rooms (rooms that do not yet have semantic data about their function) are added, starting from the front door. Then, these public rooms get a purpose (a dining room, living room, etc.). Subsequently, private rooms are added to the public ones and, finally, stick-on rooms like closets or pantries are added to the graph. Then the graph is transformed to a 2D graph in which every node gets a 2D position. Based on the desired size of the room, "pressure" is attached to each room node to make them expand. The more pressure is assigned to a room, the more space it will take up in the final building plan.

Many of the rules that are used in the grammar systems for these interior-generation systems are in fact semantic constraints. When, for example, the privacy level of a specific room type is known, it could be used more efficiently to position rooms from public to private, instead of using rules to first place all public rooms and then all private rooms as in Martin [2006]. The purpose of a building plays an important role in the interior as well, for example the size of an office space will depend on what job needs to be performed in that office; the number of rooms in a house will depend on the family situation of the residents; and the layout of a shop and the type of racks will depend on the goods that are being sold. It is important to note that many of these techniques focus on a few specific building types like office spaces or residential homes, but with the help of semantic constraints, an all-purpose algorithm should be developed that handles multiple building types, which will allow for combining them (e.g., shops on the first floor with a living area on the second floor).

4.3 Discussion

Both manual and procedural world-modeling techniques can be improved by introducing semantic information. Objects in a level editor have relationships with each other, which are significant and need to be maintained while editing the world. In the procedural generation field, we see that besides the help semantics offer to the individual techniques, the data generated and used on one level can also be used when other levels of the virtual world are procedurally generated. Terrain data is important when placing vegetation, roads, and buildings; and the high-level city parameters are useful when designing its buildings. Moreover, it became clear that when a world is procedurally generated, a lot of semantic data can be generated automatically as well. We identified several techniques that could derive material information for buildings, soil information for terrains, and functional data for buildings while generating the models automatically. In the next section we also note the importance of this kind of data at runtime. So it would be very interesting to capture this information, which is readily available in these generation techniques, to use at runtime.

Whenever the properties cannot be generated automatically, the user should be able to enter them manually in an efficient way. Such manual edits can be done more easily by using object templates and the inheritance relationship that is applied to the object classes. Object templates define abstract classes with general attributes, and can be applied to newly created object classes. This way the user of the editor only needs to enter some parameters in the object template when creating a new class. By declaring an inheritance relationship between an existing class and a new class, the new class can also inherit the attributes and parameters of the parent class. To minimize editing time, only data that is unique for the new class needs to be entered.

When we look at game worlds we see more possibilities for semantic data in level editors. For example, in many editors it is still necessary to manually define all areas that are off limits for NPCs. It would be much easier to employ semantic data to define basic rules where an NPC can or cannot walk (e.g., areas with a slope above a certain percentage are off limits).

We can also think of using semantic data and constraints to specialize the procedural generation techniques for games, for example to generate specific constraints based on the limitations in the movements of the player's character. We can define how far and how high the character can jump and how quickly he can move from one area to the next. Coupled with path-finding algorithms, this data can alter the virtual world to make sure the character is able to reach all important locations without requiring testers to try this out for every location. If a building is too high to jump on, an object that fits the environment, that is, not just a random object but one that is logical in the given location, could be placed somewhere along the wall, or a shed might be added to the building in order to let the character climb the shed and subsequently climb the building.

Before semantic information can be effectively applied in procedurally generated worlds, another important issue needs to be handled first. Many

stand-alone techniques exist for terrains, for roads, for vegetation or for buildings, but a unifying approach is lacking that seamlessly combines and accommodates for most of these techniques. This will give designers the opportunity to quickly create (parts of) virtual worlds, which they can further develop into full-fledged worlds for games or simulations or, for example, as quick prototypes to test new gameplay elements. A declarative framework is proposed by Smelik et al. [2008] that can combine these techniques in a layered-based approach.

As mentioned above, the semantic data can also play an important role while playing the game, which is discussed in the next section.

5. APPLYING SEMANTICS AT RUNTIME

When a game world is built up and is being used, the semantic data can improve many aspects of the game. Much more immersive results can be accomplished if abstract knowledge of the game state is known; for example, which area or what building plays a crucial role in a military strategy. There can also be a benefit from the semantic data to improve existing techniques in the field of AI and interaction with the user.

Current virtual worlds are mainly static during the game. They do not evolve over time and they do not adjust to variable conditions. Semantics can bring these worlds to life by adding rules about how certain objects react to changes in the world. Rules can be added on how objects change depending on the time of day (e.g., store opening hours, or time of the year, e.g., trees will loose their leaves in autumn). With the relationship constraints discussed earlier, objects could dynamically adjust to the changes of related objects. In this section we show, with the help of existing work, where semantics can improve virtual worlds at runtime.

5.1 Object and Scene Interaction

Although many games already have immense virtual worlds for the player to explore, the player can only interact with a small number of the objects in it. If interaction can take advantage of semantic data embedded into the world objects, game developers could add much more interaction into their games without much extra work.

In the field of object interaction, semantics has already been used by Kallmann and Thalmann [1998] to define which interaction techniques can be used on the objects in a virtual world. Information is added to objects to define how a user can interact with that specific object. Each movable part of an object has its own geometry, and these parts are stored in a hierarchy. A description for the possible movement of each part is described as well. A hand posture file defines which gestures are necessary to interact with the object. When the user wants to interact with a drawer, for example, he or she can perform a pulling gesture and the drawer opens based upon the end position of the gesture and on the physical constraints on the drawer (a drawer can typically be moved along just one axis); in this article such objects are called “smart objects”.

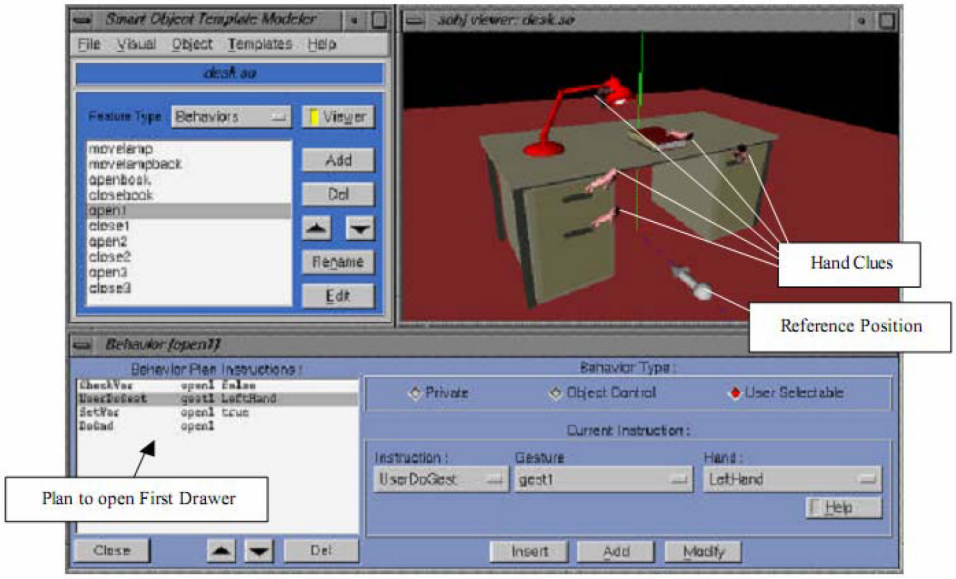

Fig. 7. The interface in which the smart objects, in this case a smart desk, are modeled. (Figure 1 from [Kallmann and Thalmann 1999]).

In later work, Kallmann and Thalmann [1999] extend the phase where the smart objects are modeled. They define four different interaction-features: First, the intrinsic object properties are stored, like the specific movements of the object parts. The interaction information is the location of interactive object parts and the hand shapes associated with those interactions (e.g., a grasping shape on the handle of a drawer). In Figure 7 we see how this interaction information is modeled in this system. The possible behaviors of objects are available as well; they depend on the object state: a door can only be closed if it is open. Finally, the expected user behavior is modeled, too. This is associated with the object behaviors, and used to show the user where he or she should put his/her hand.

The application presented by Kallmann and Thalmann uses gestures as input; but we can see the importance of this interaction information in games and simulations played with mouse and keyboard, as well game controllers. It is still interesting to know what kind of actions can be executed on an object and what the result of these actions will be.

5.2 Adaptive Environments

The types of adaptive environments we are going to discuss in this section are environments adapting to global parameters (e.g., weather conditions), and to the user's performance. We also take a look at techniques that can extend the game world procedurally to create almost infinite worlds or to create worlds based on the player's style.

Lanctot and Verbrugge [2004] present a generic model for efficiently changing a virtual environment automatically. The examples to which the technique

was applied are a weather system, in which a terrain changes according to wind and rain, and a reputation system in which agents respond to actual player behavior. The model starts with a terrain that consists of a map of grid tiles. Every grid has a number of parameters, depending on the type of implementation. These parameters are altered based on events in a grid tile or in the neighborhood of the grid tile. This technique is comparable to cellular automata where the cells are changed iteratively based on the neighboring cells. The first example where the weather conditions generate the adaptation can obviously benefit from semantic data on the climate, the soil, and the vegetation (discussed elaborately in other sections).

Adaptive environments can also achieve player-centered gameplay. Charles et al. [Charles and Black 2004; Charles et al. 2005] have researched the notion of games adapting to the level and playing style of the individual player. Instead of working with user studies before and during the development of the game, they propose in-game adaptation to a player's profile and to the evolving skill level of the player. This can help create continuously challenging situations, but also avoid the player getting stuck at a certain level. The adaptation can go two ways: the first is to create several predetermined states that can be chosen on the basis of the player's performance; the second way is really learning from the player's behavior and transforming the game based on that. The second way is obviously more powerful, but does have a disadvantage. The game cannot be tested as exhaustively as other games, since unexpected results could emerge from the transformation. The idea of this approach is to, for example, change the number of opponents or the difficulty of the opponents based on the player's skill level. But this skill level might depend on many factors, and here semantic data might contribute to these systems. Perhaps a player of a strategy war game plays on an average level in most cases, but might perform rather badly in specific circumstances (e.g., in missions in urban settings or when the player needs to attack an area on a hill). This means that properties of the environment need to be taken into consideration when adapting to the player level; but without extensive knowledge about the environment this is not possible.

The *Charbitat* system [Nitsche et al. 2006] generates game spaces procedurally. It can generate game worlds on-the-fly, based on the style of the players. The game itself represents a kind of dream world that is based around the five elements of Taoism. *Charbitat* uses a game world divided into tiles of 500×500 meters with their own seed value. A height map is generated with this seed value and is combined with some filters to generate rivers, lakes, cliffs, and coastlines. Based on the main element of a tile, objects related to that element are placed in the tile (e.g., a fire-themed tile will contain mostly fire-themed objects and creatures). Based on the actions the player performs in the game, weights are defined for each element and used to generate the next tile. An example of a *Charbitat* world and the Java back-end that shows its structure can be seen in Figure 8.

Another example of a technique that can extend a virtual world on-the-fly is given by Greuter et al. [2003]. The buildings of the "pseudo-infinite" city are generated based on a random number generator seeded by the position

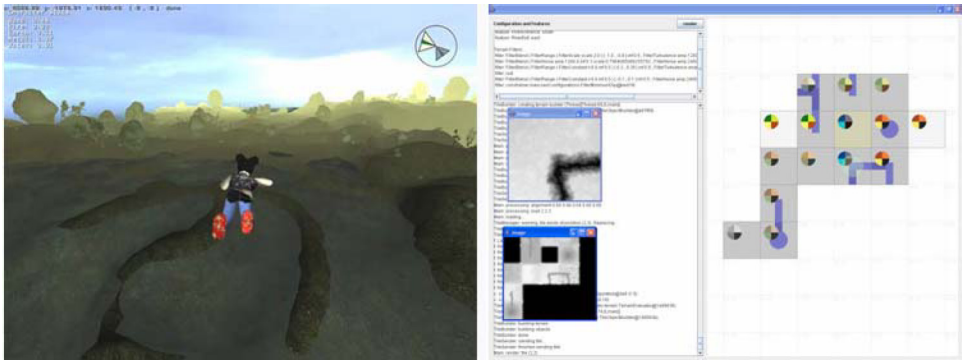

Fig. 8. In the *Charbitat* game, the game world grows adaptively to the user's behavior in the game. The Java backend shows the growing world in a tiled height map. (Figures 1 and 3 from Nitsche et al. [2006])

of the building; the technique to generate the buildings was discussed in Section 4.2. Only those buildings are generated that are absolutely necessary, meaning the buildings that are located within the viewing frustum. The city model is a simple 2D grid layout, which means that no special techniques need to be used to generate the streets. This, however, does decrease the level of realism, since only grid-shaped cities can be modeled.

We could also apply these runtime procedural generation techniques to improve the previously mentioned player-centered games. Instead of merely adjusting the number of enemies, the world itself could adjust to the player's skill levels (e.g., more objects providing cover could be placed). In simulations, virtual worlds can be generated with a specific problem that needs training: a driving simulator can generate many crossroads where the right priority applies if the trainee has problems with those.

5.3 Improving Immersive Elements

The graphical details of the models and environments continues to improve, adding to the realism of the game; but to truly immerse the player in a virtual world, other elements need to be perfected as well. We now give a couple of examples where such immersive elements can be improved by means of semantic information.

Unrealistic and unexpected behavior of NPCs can really spoil the immersion level of a virtual environment. Peters et al. [2003] apply the concept of "smart objects" on the automatic generation of animations for AI agents. The objects have user slots that characters need to go to in order to interact with the object. They need to run through a procedure of ordered usage steps (e.g., a bartender needs to wait for the customer to finish an order before beginning to pour), and each of these steps has information about whether someone can chat while performing an action or where the character needs to look during a usage step.

A world with semantically rich objects can provide AI systems with meaningful input. We already gave the example of Huhns and Singh [1997] who

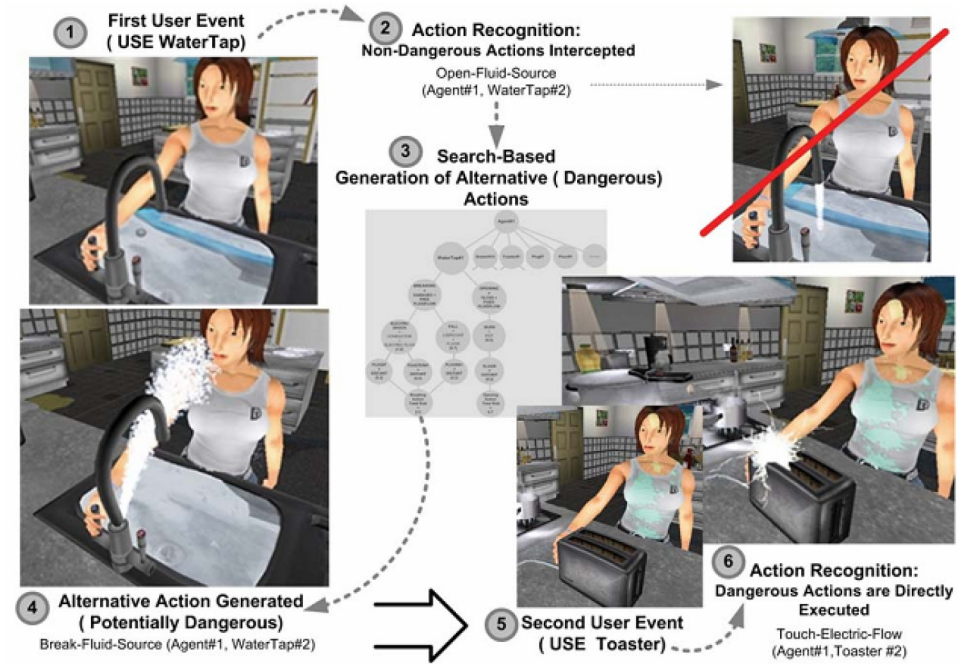

Fig. 9. The *Death Kitchen* tries to look ahead and searches for the most dangerous response to a user's action. (Figure 2 from Lugrin and Cavazza [2006]).

utilized ontologies to facilitate the communication between agents with different knowledge domains. Lugrin and Cavazza [2006] present a method supporting the AI-based simulation of object behavior. A prototype of the system was created called the *Death Kitchen*, in which a kitchen is trying to inflict as much damage on the player as possible by making the environment react in the most dangerous ways. The objects in the kitchen and the available actions are classified in an ontology. Functional reasoning is used to generate the possibly dangerous situations for the player, as shown Figure 9. The objects and sub- objects are structured hierarchically, and all have information about their functionality. This delivers a powerful intelligent environment that can be useful in various AI methods, but also in handling user interactions with the objects, and hence is a good example of how semantic data can improve gameplay.

Abaci et al. [2005] describe a system in which "smart objects" are applied to planning in virtual environments. As an example Abaci et al. describe the problem of an agent moving a box to another room. If the agent needs to open a door while moving the box, it can be planned automatically by this system via the use of these "smart objects". The basic idea is that the virtual character does not need to know how he or she can interact with every possible object, since he/she can "ask" the object which capabilities it has got. The planning system uses this information to map high-level plans to a sequence of low-level

tasks. All these techniques show that a rich semantic world can lead to much more realistic behavior for AI characters, which in turn significantly increases the believability of a virtual environment. Moreover, the characters would look more intelligent if they could navigate in a logical way; for example, not just take the shortest path towards an enemy, but search for slower and less dangerous routes. This means that, to find more realistic and plausible paths, existing path-finding techniques should integrate semantic data about the environment, as recently proposed by van Driel [2008].

When we focus on entertainment games, a big influence on the immersive experience is how a scene or an event is shown to the user. Game designers try to create stories as compelling as in movies, and to achieve this, cinematography based on movies plays an important role. In dynamic environments such as games this is not as straightforward. Several such virtual cinematography techniques have been researched [Amerson and Kime 2000; Charles et al. 2002; Tomlinson et al. 2000]. Many of these virtual cinematography approaches incorporate the film idioms used in movies. Scenes expressing specific situations have a particular structural style. Since people are familiar with scene structures from movies, game stories can be told more clearly when these structures are used in cinematography systems. It is impossible to decide which objects and events are important to focus on based solely on information alone; the system needs to know which objects could be useful to the player in upcoming events, or which locations pose a threat.

Many games already include some form of adaptive background music, but this is usually done by playing a specific song for a specific action. Users grow accustomed to these songs, and can therefore predict what is about to happen based on the song being played. A much more detailed and flexible system could improve the user experience significantly. For example, *Magenta* [Casella and Paiva 2001] is an architecture that automatically composes background music based on the emotional state of the environment. When semantic data is available in a scene, this data could help determine the emotional state of a situation or a character. Such techniques could then aid in the generation of accompanying background music.

5.4 Discussion

Many of the currently developed techniques work satisfactorily, but each one uses its own ad hoc data. AI programmers include waypoints or other tags into the world and gameplay programmers add interaction data, but it would be much more efficient to use common knowledge and share it among the various modules of the game engine. Since in the previous section we discussed that this knowledge was already useful during the design phase, the necessity for such a common knowledge level becomes even clearer. The semantic data captured in the objects and in the virtual environment during the design step could, if properly designed, be integrated in many runtime modules of the game as well. The functionality of an object is an example of this. This can be used in intelligent object placement during the design phase, but also in planning algorithms and in interactions with the user at runtime. The numerous examples discussed

in this section show that there is definitely a necessity for more "intelligent" virtual environments which can give proper information to the user (i.e., via the interaction module), to the AI system, and to other modules such as rendering and sound to create more immersive games. For the data to be usable in all these modules, we do need some form of generic specification that defines the semantic information in an explicit way. This is an important issue that needs to be resolved, because availability of semantics in multiple fields will bring about an enormous advantage, but creating such an explicit and generic specification is far from straightforward.

6. CONCLUSIONS

One of the most noticeable differences between former games and current games, besides greatly improved graphics, is the exponential growth of game worlds. Creation of larger virtual worlds obviously requires much more time, effort, and, therefore, money. We have discussed the possibilities of more extensively deploying semantic information in that process, significantly reducing costs, and creating new opportunities for designers of such large worlds. We also discussed how procedural generation techniques can be used to create more realistic and satisfactory results, and how the use of semantic constraints can help designers in their irreplaceable task of manually editing game worlds. In particular, the manipulation of huge worlds with recurring patterns (e.g., open terrains or cities) and with many interrelated objects can strongly benefit from the combination of procedural generation techniques with semantically rich objects.

The advantages of deploying semantically richer worlds in games and simulations are manifold. More lifelike character behavior increases the realism of virtual worlds, leading to a more immersive and compelling gameplay. Also, as huge worlds can be generated in which one can interact with almost every imaginable object, interaction within these worlds will become more elaborate and engaging. When objects embed knowledge about the services they can provide, the player can be justly provided with many more plausible ways of interacting with them (e.g., drink the water, soak your cap, use it to extinguish the fire, clean up the goggles, wash smudged tissue, or dissolve chemicals). In current games, we already notice a growing trend towards more elaborate interaction. For example, many action games try to enable the player to use elements of the environment as a weapon. Yet these interactions are still mainly hard-coded, so if the designer did not think of it, the player cannot do it. Embedding functional knowledge in objects can be achieved in the near future, and will enable more emergent gameplay and interaction, thus not only increasing the immersion level but the re-playability also. Later on, this kind of service-based interaction can lead to authentic sandbox-style games. Nowadays, sandbox games already allow a certain degree of freedom, but this is still limited by the designer's imagination. Interaction between two or more objects is still seldom used, except perhaps in really ad hoc ways in certain point-and-click adventures. This will, however, require some sort of robust service system that supports the description of services and can handle their conditions of use and effects, as we

discussed in Section 2.1. In any case, there remains the problem of stability and robustness, since emergent gameplay will, by definition, bring forth unexpected behavior and perhaps unforeseen issues.

We discussed some techniques that are currently used to automatically generate virtual worlds. Many of them could benefit significantly from semantic information in object classes that are to be placed in the procedural environments. At the moment, procedural techniques are already being used in game-level editors on a limited scale. They are used in specific tools, for example, level designers can use brushes that distribute some trees within the brush area instead of having to place them one by one. On a short-term basis, this could be further extended by creating more such tools that take over small repetitious chores. When we look further ahead, a combination of these tools and other procedural generation algorithms will evolve into tools that produce complete levels on the basis of designer descriptions, which can be used as prototypes or as the starting point for the designer to produce a finished level instead of having to start from scratch. The existing procedural techniques seem suitable to accomplish this, but a combining framework is still missing. On a longer term, this could grow into random map generators which could be used as an extra playmode. This has been done in the past for real-time strategy games, but with new and improved procedural generation techniques will likely become available for many more game genres. Another important milestone for the long term is the step towards fully adaptive and growing worlds. To maintain consistency in the relationships between objects in the game world, and between old sectors and new sectors in growing worlds, will require specialized constraint-solving techniques, as we discussed in Section 3.4.

An important advantage of automatically generating (segments of) a virtual world is the fact that much of the respective semantic data can be automatically generated as well. Based on the properties, rules, and relations defined in the classes of a library, procedural techniques can automatically instantiate values for the attributes of objects of these classes, possibly as a function of some instance properties, as for example its size and weight. Furthermore, when manually modifying any of these objects in a virtual world, all relations and properties previously specified are to be maintained and, if necessary, propagated according to the semantics of each editing operation. We discussed many constraint systems that could aid in this process, but a fitting generic solution is not readily available yet, and it will likely take more research and time. Altogether, these facilities will effectively help taking tedious work out of the designers' hands, while allowing for very detailed and highly customizable virtual worlds.

Moreover, a virtual world impregnated with knowledge is also an excellent way to further enrich AI and planning algorithms. AI subsystems can use semantic information to better plan actions of the NPCs. In this way, for example, instead of scripting that an AI character should walk to a given soda-vending machine once in a while, a behavior can be created where the character checks the closest object that can quench thirst, whether it is a drinking fountain, a vending machine, or a water tap. An important challenge here, however, is

the development of a generic specification layer for sharing the appropriate semantic data with all its potential client modules (e.g., AI, animation, rendering, physics, etc.). Among other problems, this will likely collide with many current practices, including the use of own ad hoc data and hard-coded scripts for expressing object behavior, features, and so on; however, as we noticed in Section 5, several research examples have proven that the use of semantic information can definitely bring about a considerable enrichment in many of these fields. A semantics-based AI system would be most useful if developed closely together with a service-based interaction system. If objects contain semantic information about how a player can interact with them, the AI system can use similar data in, for example, navigation and planning algorithms. However, the problems with the emergent behavior discussed above will be harder to handle in AI systems, since they generally require more extensive testing. Therefore, achieving proper use of semantics in AI algorithms will likely take longer than in user interaction.

In conclusion, there are numerous and rather promising uses for semantic information in virtual worlds. Many issues are still to be resolved, of which the first is the need for a generic and explicit way of defining and specifying semantic data, providing a powerful vocabulary that is useful and usable for all disciplines involved. Subsequently, the consistency of evolving objects in a changeable environment needs to be maintained, in which powerful constraint-solving methods are required. Finally, semantic data will need to be seamlessly integrated and deployed with procedural generation techniques, in order to provide designers with a new and powerful generation of tools.

Nowadays, the strongest expectancy for future games and simulations is steadily shifting from improved graphics and appearance towards convincing and believable gameplay. Embedding the game world and its objects with richer semantics will undoubtedly play a crucial role in this process, and we can therefore expect that in the near future increasing research effort and influential results will emerge from this new and exciting area.

REFERENCES

Abaci, T., Cíger, J., and Thalmann, D. 2005. Planning with smart objects. In *International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision* (WSCG), 25–28.

Amerson, D. and Kime, S. 2000. Real-time cinematic camera control for interactive narratives. In *Proceedings of the AAAI Conference*.

Bidarra, R. and Bronsvoort, W. F. 2000. Semantic feature modelling. *Computer Aided Design 32*, 3, 201–225.

Bitters, B. 2002. Feature classification system and associated 3-dimensional model libraries for synthetic environments of the future. In *Proceedings of I/ITSEC Conference* (NTSA).

Bouma, W. et al. 1995. Geometric constraint solver. *Computer Aided Design 27*, 6, 487–501.

Buchberger, B. 1985. Grobner bases: An algorithmic method in polynomial ideal theory.

Casella, P. and Paiva, A. 2001. MAgentA: An architecture for real time automatic composition of background music. In *Proceedings of Intelligent Virtual Agents: Third International Workshop* (IVA), Springer, Berlin.

Charles, D. and Black, M. 2004. Dynamic player modelling: A framework for player-centered digital games. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, 8–10.

CHARLES, D. ET AL. 2005. Player-centered game design: Player modelling and adaptive digital games. In *Proceedings of DiGRA 2005 Conference: Changing Views – Worlds in Play*.

CHARLES, F., LUGRIN, J. L., CAVAZZA, M., AND MEAD, S. J. 2002. Real-time camera control for interactive storytelling. In *Proceedings of the 3rd International Conference on Intelligent Games and Simulation* (GAMEON).

CHOI, S. H., HARNEY, D. A., AND BOOK, N. L. 1996. A robust path tracking algorithm for homotopy continuation. *Computers & Chemical Engineering 20*, 6–7, 647–655.

COYNE, B. AND SPROAT, R. 2001. WordsEye: An automatic text-to-scene conversion system. In *Proceedings of the International Conference on Computer Graphics and Interactive Technologies*, ACM, New York, 487–496.

D'AMBROSIO, D., DI GREGORIO, S., GABRIELE, S., AND GAUDIO, R. 2001. A cellular automata model for soil erosion by water. *Physics and Chemistry of the Earth, Part B: Hydrology, Oceans and Atmosphere 26*, 1, 33–39.

DEUSSEN, O. ET AL. 1998. Realistic modeling and rendering of plant ecosystems. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, 275–286.

DOHMEN, M. 1995. A survey of constraint satisfaction techniques for geometric modeling. *Computers and Graphics 19*, 6, 831–845.

FERNÁNDEZ, A. J. 2004. A generic, collaborative framework for interval constraint solving. In *Departamento de Lenguajes y Ciencias de la Computacion.*, Univ. of Malaga, Malaga, Spain, p. 269.

FERNÁNDEZ, A. J. AND HILL, P. M. 1998. A design for a generic constraint solver for ordered domains. In *Proceedings of the Types for Constraint Logic Programming Post Conference Workshop*.

GAILDRAT, V. 2007. Declarative modelling of virtual environments: Overview of issues and applications. In *Proceedings of the International Conference on Computer Graphics and Artificial Intelligence*, 5–15.

GLENN, A. K. 1991. Using degrees of freedom analysis to solve geometric constraint systems. In *Proceedings of the First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications*, ACM, New York.

GOESELE, M. AND STUERZLINGER, W. 1999. Semantic constraints for scene manipulation. In *Proceedings of the Spring Conference on Computer Graphics*.

GREUTER, S., PARKER, J., STEWART, N., AND LEACH, G. 2003. Real-time procedural generation of "pseudo infinite" cities. In *Proceedings of the First International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, p. 87.

HAHN, E., BOSE, P., AND WHITEHEAD, A. 2006. Persistent realtime building interior generation. In *Proceedings of the ACM SIGGRAPH Symposium on Videogames*, ACM, New York, 179–186.

HILLYARD, R. C. AND BRAID, I. C. 1978. Characterizing non-ideal shapes in terms of dimensions and tolerances. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, 234–238.

HOFFMANN, C. M. AND JOAN-ARINYO, R. 1997. Symbolic constraints in constructive geometric constraint solving. *J. Symbolic Computation 23*,2–3, 287–299.

HOFFMANN, C. M. AND JOAN-ARINYO, R. 2005. A brief on constraint solving. *Computer-Aided Design and Applications 2*, 5, 655–663.

HUHNS, M. N. AND SINGH, M. P. 1997. Ontologies for agents. *IEEE Internet Computing 1*, 6, 81–83.

IBANEZ-MARTINEZ, J. AND DELGADO-MATA, C. 2006. A basic semantic common level for virtual environments. *Int. J. Virtual Reality 5*, 3, 25–32.

KALLMANN, M. AND THALMANN, D. 1998. Modeling objects for interaction tasks. In *Proceedings of the 9th Eurographics Workshop on Animation and Simulation* (EGCAS), 73–86.

KALLMANN, M. AND THALMANN, D. 1999. Direct 3D interaction with smart objects. In *Proceedings of the ACM Symposium pn Virtual Reality Software and Technology*, ACM, New York.

KONDO, K. 1992. Algebraic method for manipulation of dimensional relationships in geometric models. *Computer Aided Design 24*, 3, 141–147.

LAMURE, H. AND MICHELUCCI, D. 1996. Solving geometric constraints by homotopy. *IEEE Trans. Visualization and Computer Graphics 2*, 1, 28–34.

LANCTOT, M. AND VERBRUGGE, C. 2004. Locally-adaptive virtual environments in persistent-state multi-player games. In *Proceedings of the Game On 5th International Conference on Intelligent Games and Simulations*, 89–96.

LE ROUX, O., GAILDRAT, V., AND CAUBET, R. 2001. Using constraint propagation and domain reduction for the generation phase in declarative modeling. In *Proceedings of the Conference on Information Visualisation*.

LE ROUX, O., GAILDRAT, V., AND CAUBET, R. 2004. Constraint satisfaction techniques for the generation phase in declarative modeling. In *Geometric Modeling: Techniques, Applications, Systems and Tools*, Kluwer, Amsterdam, 194–215.

LECHNER, T., WATSON, B. A., WILENSKY, U., AND FELSEN, M. 2003. Procedural city modeling. In *Proceedings of the First Midwestern Graphics Conference*.

LELER, W. 1988. *Constraint Programming Languages*: *Their Specification and Generation.* Addison-Wesley Longman, Boston, MA.

LEVISON, L. 1996. Connecting planning and acting via object-specific reasoning. Univ. of Pennsylvania.

LU, J. ET AL. 2007. Context-aware textures. *ACM Trans. Graphics 26,* 1.

LUGRIN, J. L. AND CAVAZZA, M. 2006. AI-based world behaviour for emergent narratives. In *Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ACM, New York.

MARTIN, J. 2006. Procedural house generation: A method for dynamically generating floor plans. In *Proceedings of the Symposium on Interactive 3D Graphics and Games.*

MÜLLER, P. ET AL. 2006a. *Procedural 3D Reconstruction of Puuc Buildings in Xkipché*.

MÜLLER, P. ET AL. 2006b. Procedural modeling of buildings. *ACM Trans. Graphics 25*, 3, 614–623.

MUSGRAVE, F. K., KOLB, C. E., AND MACE, R. S. 1989. The synthesis and rendering of eroded fractal terrains. In *Proceedings of the 16th Annual Conference On Computer Graphics And Interactive Techniques*, ACM, New York, 41–50.

NITSCHE, M. ET AL. 2006. Designing procedural game spaces: A case study. In *FuturePlay 2006*.

PAQUETTE, E., POULIN, P., AND DRETTAKIS, G. 2002. The simulation of paint cracking and peeling. In *Proceedings of the Graphics Interface*, 59–68.

PARISH, Y. I. H. AND MÜLLER, P. 2001. Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, 301–308.

PETERS, C., DOBBYN, S., AND MAC NAMEE, B. 2003. Smart objects for attentive agents. In *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*.

RAU-CHAPLIN, A., MACKAY-LYONS, B., AND SPIERENBURG, P. 1996. The LaHave House Project: Towards an automated architectural design service. In *Proceedings of the International Conference on Computer-Aided Design* (CADEX), 24–31.

RAU-CHAPLIN, A. AND SMEDLEY, T. J. 1997. A graphical language for generating architectural forms. In *Proceedings of the 1997 IEEE Symposium on Visual Languages*, 260–267.

SMELIK, R. M., TUTENEL, T., DE KRAKER, K. J., AND BIDARRA, R. 2008. A proposal for a procedural terrain modelling framework. In *Proceedings of the 14th Eurographics Symposium on Virtual Environments* (EGVE).

SMITH, G., SALZMAN, T., AND STUERZLINGER, W. 2001. 3D scene manipulation with 2d devices and constraints. In *Proceedings of the Graphics Interface 2001*.

SUN, J., YU, X., BACIU, G., AND GREEN, M. 2002. Template-based generation of road networks for virtual city modeling. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ACM, New York, 33–40.

TOMLINSON, B., BLUMBERG, B., AND NAIN, D. 2000. Expressive autonomous cinematography for interactive virtual environments. In *Proceedings of the Fourth International Conference on Autonomous Agents*, ACM, New York, 317–324.

TSANG, E. 1993. *Foundations Of Constraint Satisfaction.* Academic Press, San Diego, CA.

VAN DRIEL, L. 2008. Semantic navigation in video games. *M.Sc. thesis,* Delft University of Technology.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *ACM Trans. Graphics 22*, 3, 669–677.

XU, K., STEWART, J., AND FIUME, E. 2002. Constraint-based automatic placement for scene composition. In *Proceedings of the Graphics Interface Conference.*

YONG, L., CONGFU, X. U., ZHIGENG, P., AND YUNHE, P. 2004. Semantic modeling project: Building vernacular house of southeast China. In *Proceedings of the ACM SIGGRAPH International Conference on Virtual Reality Continuum and its Applications in Industry*, ACM, New York, 412–418.

Received March 2008; revised July 2008; accepted July 2008