Procedural Skeletons: Kinematic Extensions to CGA-Shape Grammars

Martin Ilčík* Stefan Fiedler* Werner Purgathofer* Michael Wimmer*

Institute for Computer Graphics and Algorithms
Vienna Technical University

Figure 1: *Procedurally generated excavator models with different geometry, skeletons and poses. All have been created by a single set of production rules. Our kinematic extensions to shape grammars allow easy posing of procedurally generated models.*

Abstract

Procedural modeling for architectural scenes was as yet limited to static objects only. We introduce a novel extension layer for shape grammars which creates a skeletal system for posing and interactive manipulation of generated models. Various models can be derived with the same set of parametrized rules for geometric operations. Separation of geometry generation and pose synthesis improves design efficiency and reusability. Moreover, by formal analysis of production rules we show how to efficiently update complex kinematic hierarchies created by the skeletons, allowing state-of-the-art interactive visual rule editing.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling F.4.2 [Mathematical Logic and Formal Languages]: Grammars and Other Rewriting Systems I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: procedural modeling, architecture, skeletal animation, shape grammars

1 Introduction

The main contribution of our work is a set of kinematic rules generalizing grammar-based procedural modeling. These rules allow incorporating semantic variations of models by creating diverse poses. We focus on urban and architectural scenes enriched by deformable objects composed of rigid parts (see Figure 1).

1.1 Motivation

Efficient creation of realistic, large scale, highly detailed models is very important for development of successful computer games, movies, and digital art. Manual modeling of such content is very expensive, plus it requires cooperation of many highly skilled artists producing stylistically consistent output. Procedural generation of urban scenes and architecture is a very active research field that provides great solutions for interactive massive design of 3D content. Efficient grammar systems exploit the fractal nature of urban scenes and a high number of identical items organized in regular patterns. Only a small set of production rules is needed to represent the geometry. Shape grammars [Wonka et al. 2003; Müller et al. 2006] turned out to be a great choice for efficiently generating realistic, high resolution models of cities. A major shortcoming in becoming more general and powerful is the static nature of generated models. It is possible to change the scene setup by taking movable objects like cars as whole and incorporate variations in their placement. However, even in scenes with no living creatures or plants there are objects maintaining the same basic geometry while changing their appearance by adopting diverse poses. Moreover, styling an object by using a certain posture (e. g., clock hands on a tower clock) can significantly improve the power of its semantic expression. The semantic attributes related to poses and expressions actually make up an important part of the content in a movie or a game, even if the amount of work behind this is not directly visible.

Shape grammars were designed for static models only, but we want to go further by adding a space of valid poses that allows more variable results. Our motivation is to enable opening doors and windows, to see a rotating weathercock, moving cranes, excavators and other complex machines. Yet there was no shape grammar-based solution able to divide a procedurally generated model into movable parts, define its agility and set a specific pose. L-Systems are a great option for procedural animation, however shape grammars are better suited for architectural modeling. Possible combination of shape grammars and L-systems in one production environment would only lead to unnecessary overhead, syntax confusion and complex evaluation issues.

*e-mail: {ilcik | stf | wp | wimmer}@cg.tuwien.ac.at

1.2 Approach overview

Our main step towards general semantic modeling with grammars is a posing interface controlling the motion freedom limits and the current model pose. We propose to store the kinematic information as an additional attribute hierarchy in a separate graph in parallel to the original model derivation graph. This additional graph is a skeleton represented by a hierarchy of joints and bone relationships – a technique widely used in skeletal animation [Maestri 1999]. The nodes of this graph are the leaf nodes of the original model derivation graph, while the edges represent kinematic joints. It structures mesh parts stored in leaf nodes according to functional links between them. One significant advantage of this data structure is its modularity.

Using the original CGA shape grammar [Müller et al. 2006], a system of procedural rules for a class of objects is very complex to implement and maintain and it is even more time-consuming to create a new set of rules whenever a pose changes. Our proposed kinematic skeleton system is created automatically during the application of geometry production rules. This approach allows using the existing rule sets while adding kinematic information to any procedurally generated model without interfering with each other. The same geometry derivation might be at any time extended and transformed to various models with different semantic meaning. The posing can be adapted through kinematic rules from the grammar or afterward as a post-processing step in a 3D modeling suite.

We provide an overview of related research mainly on the field of procedural architecture modeling in Section 2. Basics of CGA shape are shortly explained in Section 3. Our main contribution – the kinematic extensions – is presented in Section 4.

2 Related Work

Procedural modeling including all grammar based approaches has been used for compression of object description and parametrization for decades. Skeletal animation has become standard computer graphics knowledge as well. We focus on recent trends in both fields, related to procedural architecture.

2.1 Procedural modeling of architecture

Many procedural techniques have been developed in context urbanism and architectural design. The usage of grammars gained popularity when Prusinkiewicz and Lindenmayer showed, that for geometric plant modeling impressive results can be achieved by using L-systems [Prusinkiewicz and Lindenmayer 1991] as modeling of biological objects is based on growth. Man-made structures are better characterized as a sequence of partitioning steps describing the spatial distributions of objects [Prusinkiewicz et al. 2001; Wonka et al. 2003]. For the analysis and construction of architectural design, shape grammars [Stiny 1975] working with geometrical shapes instead of strings were successfully used by many authors [Downing and Flemming 1981; Beirão and Duarte 2005]. Efficient procedural production of buildings uses shape grammars adapted to the needs of computer graphics. Split grammars [Wonka et al. 2003] are focused on adding geometric detail to façades, their successor CGA (*Computer Generated Architecture*) shape grammars [Müller et al. 2006] were developed to produce large scale mass models of buildings. Recently, structural feasibility for masonry CGA shape models [Whiting et al. 2009] was achieved by adding mass and stress properties to the shapes. There is no more need for manual rule editing in text form, as an interactive visual editor [Lipp et al. 2008] enables rapid content creation without bothering with the grammar syntax. CGA shape can be considered as a state-of-the-art tool for procedural modeling of architecture. We will examine this class of grammars in more detail in Section 3. More related techniques are discussed in a recent state-of-the-art report [Vanegas et al. 2010].

2.2 Skeletal animation

In the field of robotics the problem of forward and inverse kinematics has been extensively studied [Denavit and Hartenberg 1955] and numerous methods for calculating solutions have been developed [Wang and Chen 1991; Zhao and Badler 1994], which are also used for posing and animating humanoid models [Smidt 1998]. A common method for animation of complex models uses a skeletal system to describe poses and movements [Maestri 1999]. Generally speaking, a kinematic skeleton consists of rigid sections connected by rotational joints. Each section has a joint that connects it to a parent section, and the joints describe the rotation and translation of a section relative to its parent. Together they form a hierarchy where the position of each section depends on the pose of all those which precede it in the skeleton hierarchy. Different poses can be applied to a model by rotating the joints, and it is possible to limit the rotations of joints to an arbitrary range relative to a rest position. Kinematic skeletons were introduced to computer animation as a part of layered system for mesh deformations [Chadwick et al. 1989], they are often derived from the model geometry [Bloomenthal 1999]. Deformations of shapes are out the scope of this paper, as are many other related animation techniques focused mainly on human body animation [Collins and Hilton 2001]. Procedurally generated plants also use hierarchies of joints for animation of motion in wind [Sakaguchi and Ohya 1999].

2.3 Grammar-based animation

L-Systems have been used not only for modeling growing objects, but for animation of the growth process [Prusinkiewicz et al. 1993] and environment interactions [Noser et al. 1992; Měch and Prusinkiewicz 1996] as well. Most of these approaches use rule selection driven by a time parameter with no explicit skeleton structure. Extracted skeletons have been used for animation of human organ growth [Durikovic et al. 1998]. On a higher level, behavioral animation of virtual creatures including synthetic sensors for sensing the virtual environment [Noser and Thalmann 1999] can be achieved by using L-Systems.

3 Grammar-based mass modeling

CGA shape grammar is a well established approach for procedural modeling of architecture [Müller et al. 2006]. Our kinematic extensions build upon the original grammar rules, thus we would like to introduce the basic concepts in this section.

3.1 CGA shape grammar overview

This language employs conditional, context-sensitive, stochastic evaluation of rules to retain realistic layouts of architectural elements while allowing for a wide variety of buildings generated from a set of rules. The basic rules are general enough to support the development of rule sets for different architectural styles, and the model generation does not require user input to select rules. Key elements of CGA shape are the notion of shape, the definition and geometric interpretation of basic rules, and the control of their evaluation.

Each *shape* consists of a string symbol as well as geometric and numeric attributes. *Symbols* relate to the semantics of shapes and can identify them, especially for the purpose of selecting applicable rules. Geometric attributes define the visible form of a shape, and

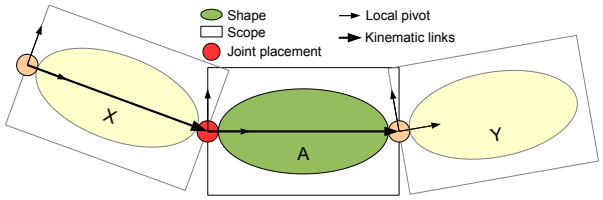

Figure 2: *Shape, scope and skeleton. An elliptic shape is located inside a bounding box representing its scope. The scope is bound to a kinematic section with a joint (red circle), a parent on the left and one child section on the right. Joints located at the origin of each local coordinate system (thin arrows) determine the kinematic transformation. Thick arrows represent the bones of the skeleton – rigid connections between adjacent sections.*

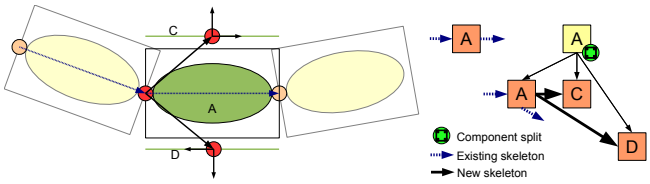

Figure 4: *A component split rule creates new shapes for lower dimensional components of a shape. Applied to the shape* A *in Figure 2 it can e. g. create new shapes for the top and bottom sides. The kinematic section of each component is connected to the original shape's section. New joints are placed at the centers of sides, with z-axis pointing along the normal vector of side faces. This results in a rotated coordinate system for the bottom component.*

most importantly include an oriented bounding box called *scope*. Shapes can be three- or lesser-dimensional. Numeric attributes allow to parametrize rules and to further control the derivation process.

One part of the basic rules modifies the scope by translation, rotation or scaling, respectively, which also affects the geometry contained within the scope. The essential rule, a *split* rule, creates two or more shapes of the same dimensionality by splitting the scope along one or more of its axes. The *repeat* split rule works similarly but creates as many shapes of the same kind as will fit into the original shape. Since both split rules should work well on a range of differently sized scopes, some of the split sizes have to be scaled, but some elements are more suitable to scaling than others. To accommodate for this fact, split sizes can be absolute or relative to the size of the original scope. Finally, the *component* split rule decomposes a shape into its lower-dimensional features, for example to create a shape for each face of a three-dimensional shape. To go back to higher dimensions, shapes can be *extruded*, expressed in the grammar as scaling along a scope axis.

CGA shape is a sequential grammar, which means that one rule is applied at a time. The order of application is determined by the priority of rules, so that rules applied earlier coarsely structure a model, and later rules gradually add more details. Each rule can also have preconditions based on numeric attributes of shapes to decide on its suitability. If several rules are applicable, one is selected based on a probability value for each rule.

4 Kinematic extensions

Starting from a set of original rules describing a static model, we add kinematic rules to define movement limitations and the current pose. Adding these extension rules requires defining how to deal with kinematic structures when applying original rules as well. We now provide an overview of the kinematic rules and examine the interpretation updates to original rules.

To simplify the kinematics integration to the existing rules system, each scope is now embedded in a kinematic section. It stores the joint limitations, the current kinematic transformation and linking to the parent section and children. We only allow creation of rigid bodies, thus scaling can be omitted. Joint positions and orientations are relative to their parent's coordinate system, not to the world coordinate system.

4.1 Skeleton grammar

Our main goal is to extend CGA grammars with new rules and concepts to allow easy control of modeled shapes by a skeleton system. During a model derivation, the skeleton always links leaf nodes of the current parse tree (Figure 3) and describes their possible movements and actual positions of mesh parts, i.e., terminal shapes. The skeleton is a directed tree created automatically along with the shapes of the model, as the production rules for shapes also change the structure of the skeleton. For this, rules create new joints and kinematic sections (see 2.2) if necessary and connect them to existing sections. An additional set of rules allows to modify each joint and its current pose. The shapes of a model are themselves attached to the sections and move with the skeleton.

Whenever a production rule is applied to a shape, the skeleton structure of the resulting shapes must be defined. Most importantly, the rules define for each resulting shape the section to which it is attached, as well as its parent, rotation and translation. For rules that produce only a transformed shape copy, the result replaces the original shape in the skeleton and therefore its section is the same as the original. We define the evaluation of kinematic sections for rules that produce more than one shape as follows:

Kinematic rules

The main purpose of introducing procedural skeletons is to separate the model generation and pose synthesis. There is a set of kinematic rules in our grammar responsible only for model posing. Since the default placement of joints may not be the best for all cases, a *joint placement rule* exists which places a joint at an arbitrary position relative to the connected shape geometry. *Limitation rules* set transformational limits for joints movement. All joints are by default blocked to be fixed – both translation and rotation are limited to zero. Their rest rotations are set by the scope transformation. Finally, the current transformation of a joint is changed by *current translation* and *current rotation* rules, which are used to apply actual poses to the model while staying within allowed movement limits.

The interpretation of model positioning needs to be reconsidered as well. The model pivot is placed at the root joint of the skeleton. All children shapes are placed by accumulating the kinematic transformations. The original CGA scopes are applied to shapes as the last transformation, without being accumulated for descendants. In the following we describe how the original CGA shape rules are modified so that they correctly update the skeleton.

Component split

In a component split, the original shape is copied to the result and it becomes the parent of all lower dimensional components. The joint of each component is placed at its center, and its section is rotated according to the orientation of the component.

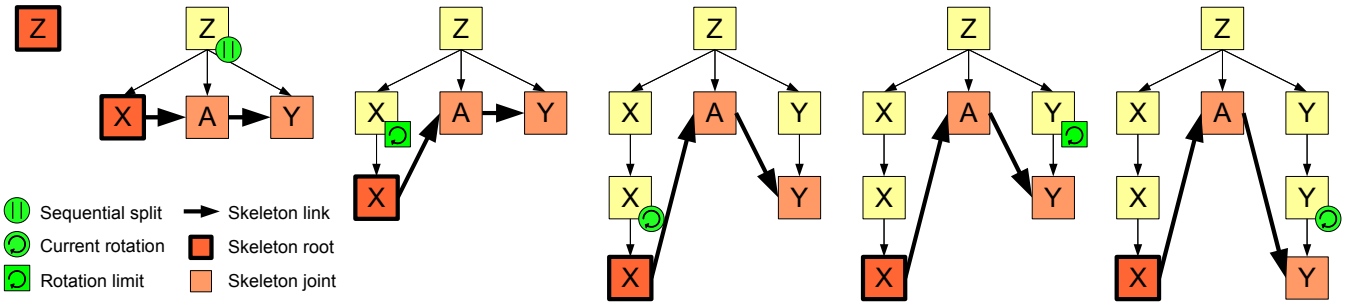

Figure 3: *Parse tree evolution for the shapes on Figure 2 including the skeleton hierarchy. First a split rule is applied, then two rotations. Strong arrows link skeletal structures, soft arrows shapes in parse tree. The skeleton root is emphasized with thick border.*

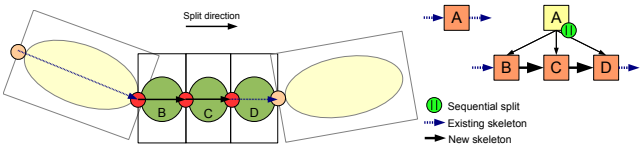

Figure 5: *An example sequential split rule applied to Figure 2 creates three new shapes and connects them in an ordered sequence. A direction vector defines the split planes, the order of resulting shapes and the placement of new joints. In this split the first shape (on the left) has the same parent as the original, and the child gets connected to the last shape.*

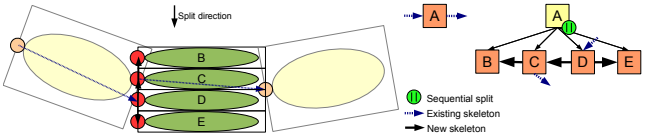

Figure 6: *Here the shape of Figure 2 is sequentially split by three horizontal planes. The split direction is always independent of bones direction. As shown here, any of the new sections can replace the original section in the skeleton hierarchy. All other shapes resulting from the split become its direct or indirect children.*

Basic split rule and Repeat rule

Since the repeat rule can be interpreted as a special case of the split rule, we focus only on the latter one. The split rule can create two different skeletal structures. The first type of split is *sequential*. It connects pairs of adjacent shapes with joints and thus creates a chain of shapes which replaces the original shape in the skeleton. The first shape in the result is placed in the same section as the original, and the last shape becomes the new parent of the original children in the skeleton graph (Figure 5). This can be seen as splitting a section into several ones, with joints between them. The sequential split rule places new joints on the split planes and on a line that is parallel to the split direction and lies on the joint position of the original shape. The split direction influences the order in which the split shapes are connected and should usually point away from the parent of the original shape to its children. Connecting parent and children to other than the first and last section is also possible (Figure 6).

The second type of split represents a fork in the skeleton and can be used to model legs or fingers (Figure 7). In this *parallel* split each new shape is connected to the parent of the original shape. The joints are placed at the midpoint between split planes, on a line that is parallel to the split direction and goes through the original

Figure 7: *The parallel split connects each new shape to the original parent. This allows to create an arbitrary number of children at each level of the skeleton hierarchy. There must be no children before the application of a parallel split.*

joint position. Before applying a parallel split, there must not be any kinematic children of the shape and it must have a kinematic parent. These restrictions serve to avoid confusion of having more parents or choosing a single one for each section.

If we want to reuse models created with the original CGA shape rules set, we need to make a decision at each split rule application whether to take the sequential or the parallel version. This can be done locally by taking a sequential split if the split direction and the shape bone are nearly parallel and a parallel split otherwise, if there are no kinematic children.

Occlusion and snap rule

The *occlusion rule* interpretation needs to be adapted to the model transformations introduced by kinematic sections. While testing for intersections of shapes, these have to be correctly placed and rotated according to the skeleton configuration. A very similar problem occurs by application of a *snap rule*. It is closely related problem of visual debugging handled in section 4.3, which shows how to obtain the final kinematic transformation for an arbitrary shape.

Epsilon symbol

The ε symbol in shape grammars represents empty geometry. When a shape with no kinematic children is substituted by ε, it gets destroyed. Otherwise it is dealt with as a terminal shape and stays stored in the configuration string to keep the skeleton consistent.

4.2 Interoperation properties

We have combined the original CGA shape rules with our kinematics rules to encode a secondary hierarchy of semantic information into the shape configuration. For later considerations, we now provide some details about special interoperation properties of the two rule sets. We mainly examine the behavior of production rules that change the skeleton structure: *basic splits* and *component split*. All other rules only apply transformations to valid section dupli-

cates, thus we may implicitly exclude them from most of the proofs: *scope rules*, *occlusion rule, snap rule, limitation rules* and *current pose rules*.

Kinematic skeleton basic properties

Definition A kinematic section is called *valid* only when it belongs to a leaf node of the parse tree.

Lemma 4.1 *For each non-empty model there is always at least one valid kinematic section.*

Proof An axiom is always the first skeleton root. Each production rule applied to a shape creates at least one new shape added to the skeleton instead of the original. The only way to destroy a shape is to replace its symbol by ε. In case the skeleton consists of a single shape, substitution by ε leads to a contradiction by an empty result model. If there are more shapes in skeleton it means the root has at least one child. Because ε-shapes with kinematic children are preserved, the root is never destroyed. □

Lemma 4.2 *All valid kinematic sections of the parse graph are organized in a single, rooted tree.*

Proof An axiom is a single rooted tree. First we show by structural induction over parse trees, that the graph of all valid kinematic sections is continuous. A *Component split* copies the previous valid section and adds new branches, a *sequential split* adds a sequence of valid sections. Only a *parallel split* would cause problems by dividing the tree into several components when applied to a section with no parent. Since we have defined the parallel split to be applicable only to shapes with a kinematic parent and no children, that case can not occur.

Second we show that there is only one root for each skeleton. If a section is the skeleton root i. e. it has no kinematic parent, each rule (except parallel split – not applicable to the skeleton root) produces exactly one child with no parent – becoming the new root. If a section has a kinematic parent, then all derived shapes are connected to a parent as well. Similar to the proof of Lemma 4.1, if the root is replaced by ε the shape is considered to be a terminal with empty geometry.

Third we show that the graph is a tree i. e. it contains no cycles. For a rooted tree it means that each node has exactly one kinematic parent. Kinematic links between sibling shapes created by an arbitrary rule satisfy this condition. Similar to the first proof part, inherited kinematic links cause problems only by a *parallel split* applied to a shape with kinematic children. From the child's point of view it would split its single parent into several more. This is one of the reasons why we have decided to forbid application of parallel splits to shapes with kinematic children. □

Corollary 4.3 *No kinematic section has more than one parent.*

Previous statements were related to the restrictions of parallel splits. An example of its wrong usage is depicted on Figure 8.

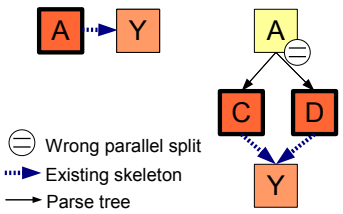

Figure 8: *Example of a parallel split applied wrong in two ways: First, it is applied to a section with no parent, thus the skeleton gets two roots. Second, the Y node gets two parents.*

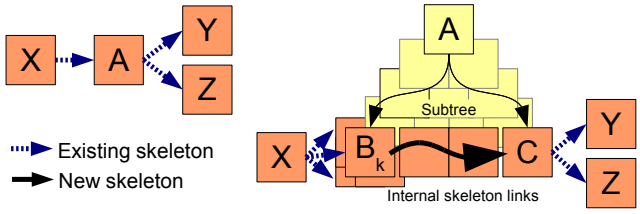

Figure 9: *Left: the shape A before evaluation with a parent and two children. Right: the subtree of A with internal kinematic links and nodes B_k and C connected outside of the subtree to kinematic parent X and children Y and Z. In case of a parallel split there might be more children of X in $s(A)$ denoted by B_k, otherwise there is only one node with kinematic parent X.*

Kinematic independence of parse subtrees

After full derivation of a parse node the outgoing kinematic connections from its parse subtree are exactly those of its root. We assume here that no other nodes except those of the subtree are evaluated. You can see an illustration of this kinematic independence on Figure 9. More formally:

Theorem 4.4 *Let the shape A be a parse tree node possibly with at most one kinematic parent $p(A)$ (see Corollary 4.3) and a finite set of kinematic children $c(A)$. The set of all parse children of A is denoted by $s(A)$. Then $\forall N \in s(A) : (p(N) = p(A) \oplus p(N) \in s(A) \setminus \{N\}) \wedge (c(N) = c(A) \oplus c(N) \subseteq s(A) \setminus \{N\})$ holds when production rules are applied only to shapes from $\{A\} \cup s(A)$.*

Proof We again show that starting with a valid configuration (see Lemma 4.1), after any rule application it stays valid. It can be proven in a very similar way to Lemma 4.2. Possible side branches as shown in Figure 6 have no effect on the linking to original parent or children. □

This theorem has several straight forward consequences:

Corollary 4.5 *If there is no kinematic parent resp. child in the subtree root, there is also no parent resp. child link leading outside from any subtree node.*

Proof Directly from Theorem 4.4 by setting $p(A) = \{\}$ and $c(A) = \{\}$ respectively. □

Corollary 4.6 *There is always at most one leaf node of the subtree with outgoing child links.*

Proof Similar to the proof of Theorem 4.4 and by Lemma 4.2, only an incorrectly applied *parallel split* to a shape with kinematic children would cause $c(A)$ to be linked to several nodes from $s(A)$ (see Figure 8). □

General kinematic independence of parse subtrees

If we want to generalize Theorem 4.4 to stay valid after evaluation of arbitrary parse nodes, we must take into account that $p(A)$ and $c(A)$ might have been influenced by further evaluation of the parse tree. Therefore, we apply the previous theorem to both of them as well.

Theorem 4.7 *Using the notation from Theorem 4.4, we supplement that $s(\{\}) = \{\}$ and $s(\{a_1 \ldots a_n\}) = \bigcup_{k=1}^{n} s(a_k)$. Then $\forall N \in s(A) : (p(N) \in s(p(A)) \oplus p(N) \in s(A) \setminus \{N\}) \wedge (c(N) \subseteq s(c(A)) \oplus c(N) \subseteq s(A) \setminus \{N\})$ holds.*

Proof We suppose that an isolated subtree evaluation of the node A followed the Theorem 4.4. Due to Corollary 4.3 $p(A)$ always

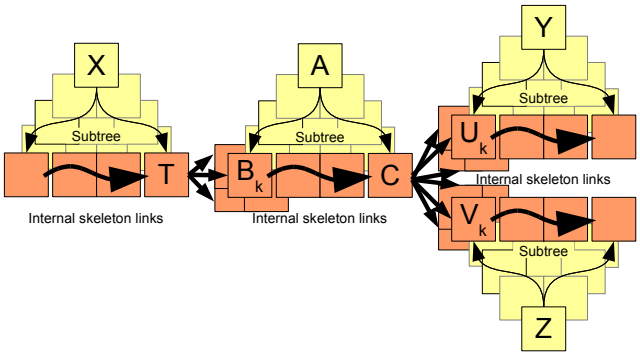

Figure 10: *The full evaluated subtree of A with internal kinematic links as in Figure9. The derivation is resumed by evaluating the parent X and both children shapes Y and Z. The kinematic parent of B is now T – a descendant of X. The number of kinematic children of C might have increased in case of parallel splitting – depicted as U_k and V_k*

has at most one element. If we apply Corollary 4.6 to the node X, there will be only one leaf node with an outgoing kinematic link – namely the node T (see Figure 10) with one or more children B_k. By Corollary 4.5 even if $|B_k| > 1$, still only one node is linked to kinematic children outside of $s(A)$. Thus, the first part of the Theorem holds. In a similar way we apply Theorem 4.4 to $c(A)$ together with Corollary 4.6 to prove the second part. □

Direct scope and kinematics transformations

Lemma 4.8 *No rule applied to a shape performs geometric or kinematic transformations on any other shapes.*

Proof Detailed examination of all rules finds two exceptions: After the *joint placement rule* moves its own section it needs to move all direct children sections in opposite directions to maintain relative positions in the skeleton. The *sequential basic split* rule places pivots of created shapes at the splitting hyperplanes, requiring to compensate by an inverse movement of the last shape's children. Storing the children translation in the parent section and querying when necessary resolves this problem for both production rules. □

Transformations lock after birth

Definition We define the *birth state* as the derivation configuration after a production rule was applied to a shape and all its direct children have been created – but no rule has been yet applied to any of them.

Theorem 4.9 *Only skeletal links of a shape change after its birth.*

Proof A kinematic section consists of a joint and skeletal links. According to Lemma 4.8, only direct scope and kinematics transformations are applied. For all rules it happens only at birth time, thus there will be no change to the joint afterward. It follows that only skeletal links change after birth. □

4.3 Visual debugging

Visual debugging of models is a very important tool for understanding the whole derivation step-by-step. It is carried out by highlighting non-terminal shapes including their kinematic transformations, so that the user gains insight into any intermediate configuration of the model. To render a shape in the correct pose it is necessary to accumulate all kinematic transformations back to the skeleton root. The kinematic links are always valid only in leaf nodes of the parse tree. After each further production rule application, the kinematic

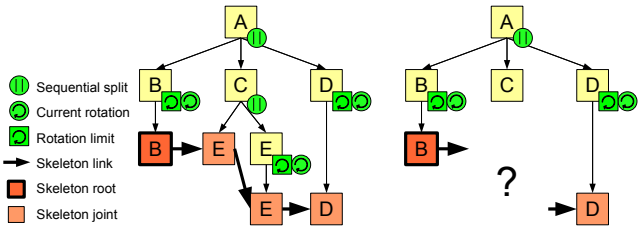

Figure 11: *A broken skeleton after removal of a subtree rooted in C. The algorithm needs to fix the skeleton by finding the broken links and connecting them to C. We use a shortened notation of the parse tree accumulating transformational rules in this figure.*

links are transferred to the new leaf shapes, destroying the links in previous shapes. For visual debugging of arbitrary parse tree nodes, it must be possible to reconstruct the broken links.

Our solution requires all shapes to store kinematic information valid for the skeleton birth state configuration. The kinematic links of birth snapshots are always valid, thus the whole skeletal chain up to the root can be traversed using the birth snapshots. By Theorem 4.9, the joint state in the birth snapshot is always identical to the final one. Using the birth snapshots we can show any shape in the parse tree correctly placed and oriented according to the skeleton configuration during its creation.

4.4 Interactive editing

Recent trends prefer visual editing of grammar systems [Lipp et al. 2008] instead of manual text writing. Interactivity and fast evaluation of models play a very important role during the visual design process. To reduce the number of shape computations after a user action, it is sufficient to select and regenerate only those parts of a model that could possibly be affected by one of the updated rules.

Looking at the model parse tree, all subtrees rooted in nodes with symbols of relevant rules need to be reevaluated. This means that leaf nodes containing final skeleton parts get destroyed. The concept of procedural skeletons as presented in previous sections has no possibility to recover in such a situation (see Figure 11).

In section 4.3 we have explained how each shape stores birth time kinematic information for the retrieval of the skeleton configuration. The problem of subtree regeneration for a non-terminal parse node is different, since we need to divert the skeleton in its final configuration from the subtree leaf nodes to its root, instead of only traversing earlier configurations. By Theorem 4.9, only skeletal links change after shape birth, so there is no need to care about restoring kinematic transformations. Again, kinematic links of the subtree root are valid only at their birth, thus they must be reconstructed from the subtree.

We work with the final parse tree with terminal shapes in leafs. The selected subtree collapses to a single leaf node, making its inner kinematic links obsolete. Only the links leading out of the subtree are interesting for the update. Kinematic independence of parse subtrees described in theorems 4.4 and 4.7 assures that the skeleton stays consistent during the evaluation to the leaf nodes. According to Corollary 4.6 there is only one leaf node with child links pointing outside of the subtree and only one external leaf node with child links pointing inside. We connect these to the subtree node to maintain the skeleton consistency. To improve the search traversal for these two nodes, we mark the first and last kinematic section in the chain created by application of each rule. Leaf nodes marked as the first resp. last are the wanted ones.

Algorithm 1 Example part of a grammar for buildings with rotating floors. *RotLimits* and *CurrerntRot* are variables for specifying the rotation of each floor. The skeleton is rooted at the basement, thus the rotations will be accumulated up to the top. The building variations are shown in Figure 12.

$$
\begin{aligned}
BuildingRoot &\rightsquigarrow Subdiv(\text{"}Y\text{"}, 3.5)\{Basement|Floor\} \\
Floors &\rightsquigarrow Repeat(\text{"}Y\text{"}, 3a)\{Floor\} \\
Floor &\rightsquigarrow KinematicRotation(RotLimits, CurrentRot) \\
&\quad Comp(\text{"}sidefaces\text{"})\{FloorFacade\} \\
FloorFacade &\rightsquigarrow Repeat(\text{"}Y\text{"}, 1.5a)\{WindowPane\} \\
&\quad \ldots
\end{aligned}
$$

The *occlusion rule* and *snap rule* are the only ones with global scope. Their results are dependent from a number of collision tests with shapes resp. snap lines spread over the whole parse tree. Thus, our subtree evaluation method is inefficient here. We see space partitioning as a possible option.

5 Results and performance

Our approach adds several kinematic rules for transformations of model parts. We explore the opportunities of this grammar extensions and demonstrate the flexibility by several examples with various grammar systems. A nice architectural use case is the concept of rotating towers [Fisher 2008]. We anchor the skeleton in the basement and accumulate rotations on each floor. The grammar extension allowing rotations is minimal (see Algorithm 1). Creation and testing of the basic model with kinematics takes only a few minutes. The poses are applicable to other skyscraper models as well.

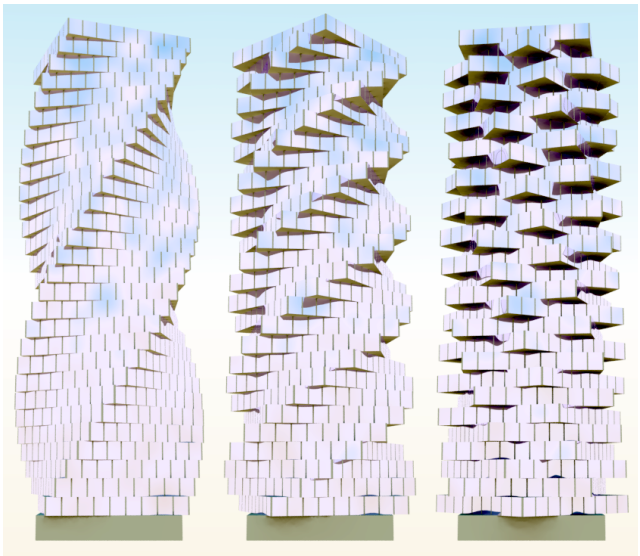

Figure 12: *Example of a building with rotating floors constructed with a simple grammar (see Algorithm 1). Since the kinematic rules are incorporated in the modeling grammar, it is possible to apply them to other models as well and they can be used later for changing the pose or animating the model.*

For a more complex example, we have chosen to design a grammar for production of various heavy construction machines. Finding the correct grammar desing using CGA rules and setting up the kinematics took us about an hour. In comparison to the towers example, the skeleton plays a much more important role here. The pose of a machine can clearly express its working status. On Figure 13 you can see two excavators. Thanks to our kinematic extensions and automatic skeleton creation it is possible to easily produce and control both machines by a single grammar. The excavators have diverse geometry and skeletons, but there are also a lot similarities due to common rule basis.

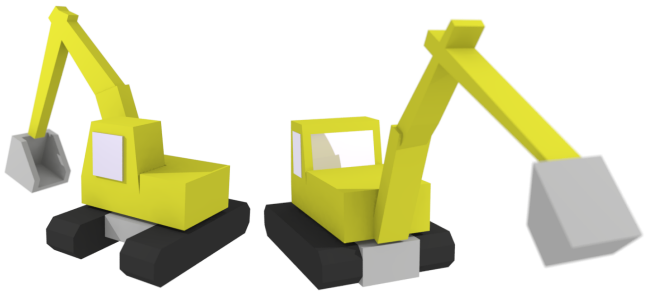

Figure 13: *Two excavators with similar skeletons. Created by one grammar, they share some geometric and kinematic properties.*

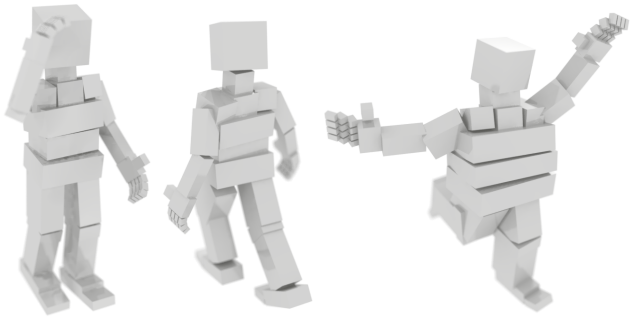

Figure 14: *A parametrized robot model with different types of physique and a complex kinematics system.*

The most complex model from our experiments can be seen on Figure 14. It is a procedurally created humanoid robot with variable body shape and a large number of joints. It is able to take a wide range of various poses and exploits nearly all CGA capabilities. Therefore, it became our favorite testing object.

Grammar interpretation including the kinematic extensions is comparable in the performance to the original system. For our implementation we have defined the production rules and shapes as program classes, so the language is not interpreted from a text file. We can manipulate the rules with a simple interface without the need of direct text editing. For efficient design we use a "normal form" for rules to improve understanding and editing the rule structure: first geometry operations are applied (geometry refinement/splits), then kinematics. The visual debugging introduced in Section 4.3 is a great help together with tree-views of the grammar structure, skeleton and parse tree. The user also has any time the possibility to switch off all kinematics for better examination of the basic model setup.

6 Conclusions and future work

We have demonstrated how CGA shape grammars can be generalized by adding a set of kinematic rules and adjusting the interpretation of existing rules. Our approach combines procedural 3D

modeling with rigging, at a minimal cost. Creation of kinematic relations is part of the modeling process. Therefore, the objects become easier to animate and kinematic characteristics can be easily applied to different models created by the same rule set.

Geometry and skeleton are treated as a two-layered data structure. We have shown an efficient update strategy for the coupled graphs, avoiding re-evaluation of the whole derivation tree. Efficient updates of rules with global scope are one of the next research topics.

There are still many problems and challenges, possible ways of improving the grammar systems and breaking away from architecture to general objects modeling. The most important topic for future work is another improvement of rules interpretation. The shapes are not connected when kinematic transformations are applied. Leaks and overlaps emerge as you can see on most of the pictures. We would like to treat shapes directly connected by kinematic sections not only as independent rigid bodies, but also as one deformable mesh with geometric connectivity preserved depending on given attributes. There are several problems to consider: preservation of mass, extremal positions and collisions resolution. We also plan to pursue our research goal of animated scenes where not only the poses but the whole movement would be generated by grammar based controllers. The approach used for feasible masonry models [Whiting et al. 2009] shows how important it is to address statics issues – in future we could deal with pose stability in a similar way.

Acknowledgments

This research was supported by the Austrian FIT-IT Visual Computing initiative, project GAMEWORLD (no. 813387). We would like to thank Markus Lipp, Johannes Scharl and Daniel Scherzer for their supporting ideas.

References

BEIRÃO, J., AND DUARTE, J. 2005. Urban grammars: towards flexible urban design. *Proc. 23rd Int. eCAADe Conf*, 491–500.

BLOOMENTHAL, J. 1999. Skeletal methods of shape manipulation. *Proceedings Shape Modeling International '99. International Conference on Shape Modeling and Applications*, 44–47.

CHADWICK, J. E., HAUMANN, D. R., AND PARENT, R. E. 1989. Layered construction for deformable animated characters. *ACM SIGGRAPH Computer Graphics 23*, 3 (Juli), 243–252.

COLLINS, G., AND HILTON, A. 2001. Models for character animation. *Software Focus 2*, 2, 44–51.

DENAVIT, J., AND HARTENBERG, R. 1955. A kinematic notation for lower-pair mechanisms based on matrices. *J Appl Mech 23*, 215–221.

DOWNING, F., AND FLEMMING, U. 1981. The bungalows of buffalo. *Environment and Planning B 8*, 3, 269–293.

DURIKOVIC, R., KANEDA, K., AND YAMASHITA, H. 1998. Animation of Biological Organ Growth Based on L-systems. *Computer Graphics Forum 17*, 3 (August), 1–13.

FISHER, D., 2008. Dynamic architecture. `http://www.dynamicarchitecture.net/` accessed in March 2010.

LIPP, M., WONKA, P., AND WIMMER, M. 2008. Interactive visual editing of grammars for procedural architecture. *ACM Transactions on Graphics (TOG) 27*, 3, 102.

MAESTRI, G. 1999. *Digital Character Animation 2: Essential Techniques*. New Riders, Indianapolis.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND VAN GOOL, L. 2006. Procedural modeling of buildings. *ACM Transactions on Graphics (TOG) 25*, 3, 614–623.

MĚCH, R., AND PRUSINKIEWICZ, P. 1996. Visual models of plants interacting with their environment. In *Proceedings of SIGGRAPH'96*, ACM, 397–410.

NOSER, H., AND THALMANN, D. 1999. A rule-based interactive behavioral animation system for humanoids. *IEEE Transactions on Visualization and Computer Graphics 5*, 4, 281–307.

NOSER, H., THALMANN, D., AND TURNER, R. 1992. Animation based on the Interaction of L-systems with Vector Force Fields. In *Proc. Computer Graphics International*, Springer-Verlag New York, Inc., vol. 92, 747–761.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1991. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, NY, USA.

PRUSINKIEWICZ, P., HAMMEL, M., AND MJOLSNESS, E. 1993. Animation of plant development. In *Proceedings of SIGGRAPH'93*, ACM, vol. 93, 351–360.

PRUSINKIEWICZ, P., MÜNDERMANN, L., KARWOWSKI, R., AND LANE, B. 2001. The use of positional information in the modeling of plants. In *Proceedings of SIGGRAPH'01*, ACM, vol. 12, 289–300.

SAKAGUCHI, T., AND OHYA, J. 1999. Modeling and animation of botanical trees for interactive virtual environments. *Proceedings of the ACM symposium on Virtual reality software and technology - VRST '99*, 139–146.

SMIDT, W. 1998. *Verallgemeinerte inverse Kinematik für Anwendungen in der Robotersimulation und der virtuellen Realität*. Master thesis, Universität Dortmund.

STINY, G. 1975. *Pictorial and formal aspects of shape and shape grammars and aesthetic systems*. PhD thesis, University of California, Los Angeles.

VANEGAS, C. A., ALIAGA, D. G., WONKA, P., WADDELL, P., AND WATSON, B. 2010. Modeling the Appearance and Behavior of Urban Spaces. *Computer Graphics Forum 29*, 1, 25–42.

WANG, L.-C. T., AND CHEN, C. C. 1991. A combined Optimization Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators. *IEEE Transactions on Robotics and Automation 7*, 4, 489–499.

WHITING, E., OCHSENDORF, J., AND DURAND, F. 2009. Procedural modeling of structurally-sound masonry buildings. *ACM Transactions on Graphics 28*, 5, 1.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. *ACM Transactions on Graphics 22*, 3 (Juli), 669–677.

ZHAO, J., AND BADLER, N. I. 1994. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics 13*, 4 (Oktober), 313–336.