Graphics Interface Conference 2018, 8–11 May, Toronto, Ontario, Canada

gMotion: A Spatio-Temporal Grammar for the Procedural Generation of Motion Graphics

Edoardo Carra*   Christian Santoni†   Fabio Pellacini‡

Sapienza University of Rome

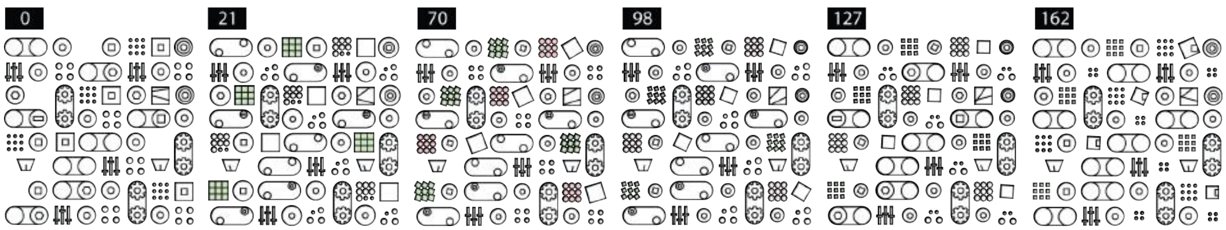

Figure 1: An example animation created by our *timeslice grammars*. Here we show the first and five intermediate frames, with the labels indicating the timestamp of each frame. For the full animation please refer to the supplemental material. This animation is composed by more than 200 shapes and was completely generated in less than a second.

Abstract

Creating by hand compelling 2D animations that choreograph several groups of shapes requires a large number of manual edits. We present a method to procedurally generate motion graphics with timeslice grammars. Timeslice grammars are to time what split grammars are to space. We use this grammar to formally model motion graphics, manipulating them in both temporal and spatial components. We are able to combine both these aspects by representing animations as sets of affine transformations sampled uniformly in both space and time. Rules and operators in the grammar manipulate all spatio-temporal matrices as a whole, allowing us to expressively construct animation with few rules. The grammar animates shapes, which are represented as highly tessellated polygons, by applying the affine transforms to each shape vertex given the vertex position and the animation time. We introduce a small set of operators showing how we can produce 2D animations of geometric objects, by combining the expressive power of the grammar model, the composability of the operators with themselves, and the capabilities that derive from using a unified spatio-temporal representation for animation data. Throughout the paper, we show how timeslice grammars can produce a wide variety of animations that would take artists hours of tedious and time-consuming work. In particular, in cases where change of shapes is very common, our grammar can add motion detail to large collections of shapes with greater control over per-shape animations along with a compact rules structure.

Index Terms: Computing methodologies—Procedural animation; Theory of computation—Grammars and context-free languages

1 Introduction

In this paper, we consider the problem of procedurally generating two-dimensional motion graphics where many, often abstract, shapes move in unison to form compelling spatio-temporal patterns. Those kind of animations are often used for advertisement, web sites, news and generally videos that focus on an abstract and minimalistic style, or to procedurally add more motion detail to an animated scene.

Today, the creation of motion graphics follows mainly three different approaches. The most popular form is keyframing, in which the values of object properties are specified for keyframes and interpolated throughout the rest of the animation. Despite the constant improvements to animation softwares, animating shapes in this manner requires significant manual work, often hours for hundreds of shapes, that cannot be usually re-used to create new animations. A second popular approach is the use of simulation engines, which control these animations by simulating either natural forces applied to the objects, e.g. particle systems [15], or actors in groups, e.g. boids [16]. One drawback of these approaches is that they can only express very specific types of motion, either physics-driven or actor-based, so they are not suitable for many motion graphics used for example in motion graphics. Finally, one could write procedural scripts to control objects properties over time. While this would support all types of motion, choreographing hundreds of shapes with individual scripts remains cumbersome, especially considering that such scripts are not re-usable.

In this paper, we propose a formal model for the procedural generation of motion graphics with enough expressive power to encode the fundamental characteristics needed to create a large variety of complex animations. We formally model motion graphics with *timeslice grammars*, a grammar system that handles both the generation of the temporal subdivisions of the objects timelines and spatial assignment of specific animation effects to groups of objects. Timeslice grammars can be considered as an extension of *split grammars* [23] and *group grammars* [17]. We extend split grammars by considering time as an additional dimension and using splitting operations to guide the rhythm of the animation. We extend group grammars by using multiple group tags to choreograph multiple objects. In our model, shape animations are specified as sets of affine matrices sampled in space on regular grids and in time by keyframing. We define a small set of operators for both the spatial and temporal part, which combined with our grammar can generate complex animations, including looping animations. Compared to other procedural animation methods, the main advantage of timeslice grammars, just like other grammar-based methods, is that they can provide a wide amount of variation in the produced results with a very compact description, that is formally well-defined.

We implemented our grammar in a prototype system that can automatically generate a large variety of motion graphics, shown throughout the paper and in the supplemental material. Fig. 1 shows

*e-mail: carra@di.uniroma1.it
†e-mail: santoni@di.uniroma1.it
‡e-mail: pellacini@di.uniroma1.it

an example animation generated by our system. We believe that the main contributions of our work are: (1) the formalization of motion graphics with timeslice grammars, (2) a small set of operators that are efficient and easily composable with all the other operators, allowing the creation of complex animations together with a wide amount of variations in the generated results.

2 State of the Art

Grammar Systems. Formal grammar systems have been shown to be an expressive method for modeling tasks in computer graphics. The first example is *L-Systems* [6], a grammar-based modelling method for simulating the growth of simple multi-cellular organisms. Since its first definition, L-Systems have being extended to modeling plant ecosystems, e.g. [9, 14], cities and road networks [10]. A comprehensive description of these models can be found in [12].

The past decade has seen the extensive use of formal grammars for modeling architecture, starting from the original formulation of *shape grammars* [21] that operated on labeled arrangements of geometric shapes, representing the symbols of the grammar. Later, *set and split grammars* were presented [22, 23], as a simplification of shape grammars. In these grammars, rather than having an initial string of symbols, the system has an initial shape, and each application of a production rule affected directly the geometry of the initial shape. There are several important works in this context. For the sake of brevity we refer the reader to [18]. These grammars have also been extended to generate patterns on surfaces [5] or tangle art with *group grammars* [17].

All the grammar approaches described so far produce static objects (e.g.: plants, buildings, road networks) without providing any control over their animation. Some works have proposed methods to add animation aids to the static models, such as [2] that augments L-Systems with additional information for plants and buildings to make them ready for simulation. These approaches though only generate animation aids and not the motion itself.

[4] propose a motion grammar to generate animated scenes using a data driven approach for the animations. This approach is context specific to the animations used, and it does not allow to take control of the spatial and temporal components of the animation itself.

Time slices have yet been used by [13], which propose a model for simulating the plant development, but in this case the context is very specific too, and it does not allow to take absolute control of animation in both spatial and temporal components.

To the best of our knowledge, our *timeslice grammars* are the first grammar-based approach that considers both spatial and temporal features as first class entities, in all aspects of the model, i.e. representation, grammar symbols and production rules.

Simulation. Simulation systems are one of the most popular approaches that have been followed to solve the problem of animating a large set of objects. These approaches usually divide themselves into two main categories: physics simulations and agent-based simulations.

A vast literature can be found in the field of physically based animation of groups of objects, starting with the seminal work of [15]. These methods are extensively used both in academia and commercial softwares and encompass particle systems, rigid body solvers (e.g. [3]) and generic simulation engines (e.g. [7, 20]). In these systems, animations cannot be controlled directly by a designer, but are instead produced by the simulation of a physics system. In this sense, they solve a different problem than the one we want to address with our work.

Agent-based simulations are based on simple concept: each agent perceives its own state and decides its following actions based on a set of rules. This approach has been explored in many works, addressing from the simulation of flocks and herds [16], to crowds of people [8, 11, 19]. These methods suffer from being too application-specific, and considering the heterogeneity of the effects applied creating motion graphics, they would require the definition of a specific set of rules for each type of animated object.

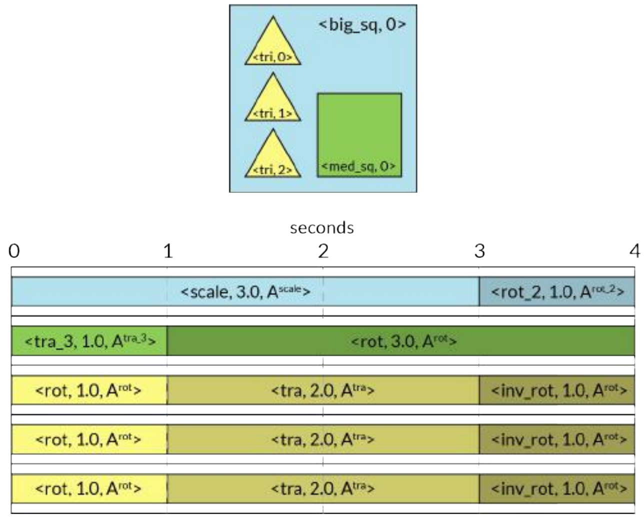

Figure 2: Representation of the final configuration for an animation created with a toy grammar. *Top:* The shapes that compose the animation, identified by a tag, matched by the grammar's rules, and a shape indentifier unique within the same tag. Shapes are color-coded depending on the value of their tag I_S. *Bottom:* The timelines associated to the shapes of the animation. Each timeline is composed by a list of timeslices, that are the symbols of our grammar, and are identified by their tag, their duration and animation data, here indicated as A^x. Timelines are color-coded depending on the shape they refer to and changes in color luminosity are used to highlight different values for the tags of the timeslices.

3 Timeslice grammar

3.1 Motion Graphics Model

We model motion graphics with a timeline of animated properties for each shape, as shown in Fig. 2.

The timeline is represented as a sequence of timeslices, each of which holds the information necessary to animate a particular object in a specific slot of time. In our implementation each shape has its own timeline so that we can have maximum flexibility when applying animation effects. One key difference of our work compared to other procedural animation systems is that we represent shape animations as uniformly sampled spatial grids of affine transformations that are keyframed in time, as shown in Fig. 3. Each matrix defines the transformation to be applied to the shape. Transformations are selected based on the shape centroid and either applied with respect to it or to the global reference frame. We augmented this representation with animations for the colors of each shape. This representation is simple enough to be manipulate directly by animation editing operations while at the same time allows us to support single and groups animations with rigid and non-rigid transformations.

To output the final animation at a fixed framerate, we uniformly sample the timeline of each shape using standard keyframe animation methods. Colors are interpolated trilinearly between the two keyframes and the values in the spatial grid. Affine transforms can be either interpolated trilinearly, using the method introduced by [1], or using nearest neightbor lookups. In the example of this paper, we use linear interpolation over time and nearest lookup in the spatial grid since we found this to be a good compromise between speed and quality.

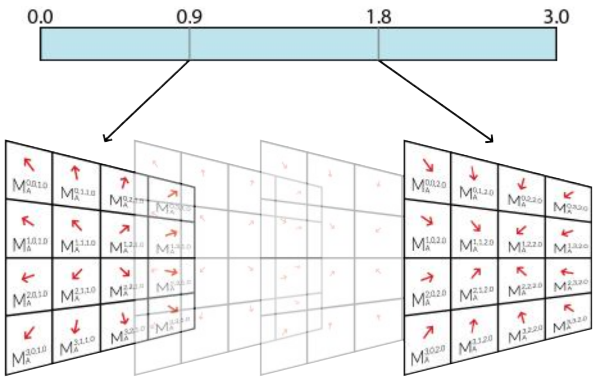

Figure 3: Visual representation of a timeslice (*top*) with its associated animation (*bottom*), represented in this figure as a 4x4x2 array of affine translations. The red arrows represent the direction of each translation. The middle semi-transparent grids represent the interpolations computed during the animation rendering process.

3.2 Timeslice Grammars

The core idea of our work is to use formal grammars to control both the temporal and spatial components of an animation. Fig. 4 shows an example grammar together with the final generated animation. In this chapter, we first give the intuition behind timeslice grammars and then provide a succint description of their formal definition.

Timeslices are the terminal and non-terminal symbols in our grammar, and are defined by a grammar tag and animation data, hence the name *timeslice grammars*. Timeslices are directly manipulated by a small set of operators to obtain the animation of all shapes. The operators are composable with one another, allowing us to achieve different procedural effects by composing a small set of simple operations with the grammar.

To associate animation operations to shapes, we identify shapes with a tag and a unique id. The tag is used during grammar expansion to select which rules to apply to each shapes. The id uniquely identifies a shape and can be used to change the rule behaviour, e.g. apply different translations to different shapes.

Each shape animation starts with a single timeslice that covers the whole timeline. Timeslices are then recursively split by the grammar to apply different animation effects at different times. The final timeline is a partition of the initial timeslice obtained by subsequent slots. This observation motivated our choice of using grammars derived from split grammars to manipulate time as well. In a way, *timeslice grammars are to time what split grammars are to space*.

To support looping animation, we found it helpful to tag a timeslice as either playing forward or backward time. The latter are indicated within the grammar with inverted tags. For inverted time slices, the animation transformation and colors are interpolated by swapping the start and end values within the timeslice.

3.3 Grammar Operators

We introduce two types of operators to manipulate the animation: the *time split operator*, that modifies the temporal structure of the animation, splitting timeslices into new arrangements, and the *animate operator* that assigns changes to both the animation transforms and decorative attributes in each timeslice.

The time splitting operator, shown in Fig. 5, splits an input timeslice into multiple output timeslices at specified relative times. For inverted timeslices, the time splits are applied in backward, thus keeping the same semantic in the looping animation without needing duplicated rules.

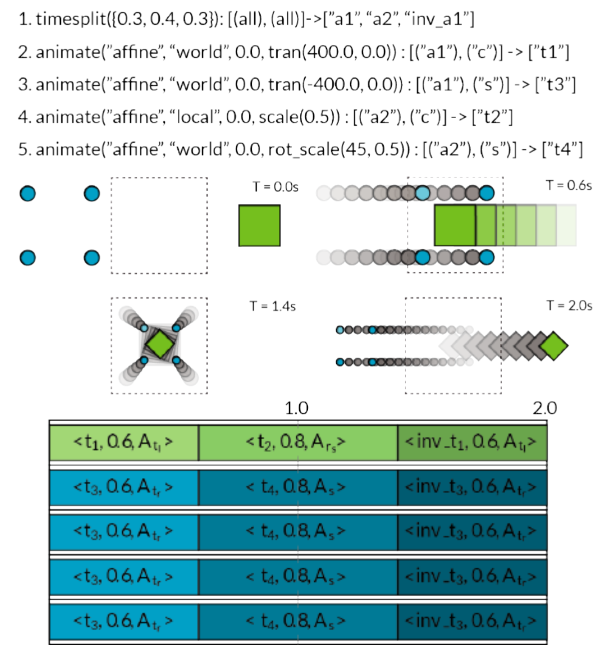

Figure 4: A toy grammar (*top*), together with frames picked from the generated animation (*middle*), and the final timeline configuration (*bottom*). The square and circles are tagged respectively with the tags "s" and "c". *tran*, *scale* and *rot_scale* represents affine transformations that we do not report on full for readability.

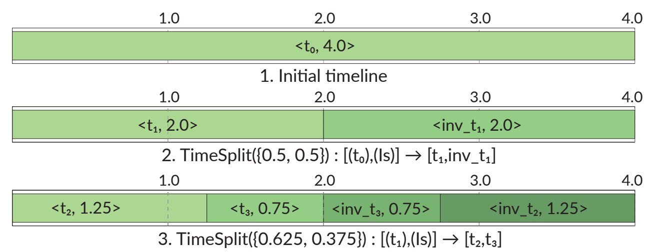

Figure 5: Recursive application of the time split operator. (1) Initial timeline configuration. (2) Each application of the operator subdivides the input timeslice into a set on new timeslices, labeled accordingly to the operator parameters. (3) Example of inverted tags. Note how both the tag assignment and the split points are inverted.

Animation effects are applied with a single grammar operator that sets the spatial grid of affine transformations at the start and end of the selected timeslice. We support different animation effects by defining different spatial grids of affine matrices. For the results of this paper we used the following effects: (1) *affine transform* that applies a constant transformation to all transforms on the 2D grid; (2) *move towards* that sets all affine transforms on the 2D grid as translations from their center toward a point; (3) *follow path* that sets all affine transforms on the 2D grid as translation from their center toward the closest point on a path; (4) *fill/border color* that sets the colors for the selected shapes.

Note how we always sample operators on the spatial 2D grid, for example for *move towards* and *follow path*. The main reason for this is to ensure that grammar operators are composable by using the same input and output representation for all. In this manner, very complex effects can be controlled completely by the grammar formulation. This is similar to [17]. Most procedural systems instead rely on some form of programmable "plugin" which is typically not composable, so complex effects need to be replicated for each plugin.

3.4 Grammar Expansion

The configuration of an animation at a particular step of the expansion process is represented by all timelines, one for each shape, each composed of a list of individual timeslices. Starting from an initial configuration, productions are applied sequentially to the current state of the animation to produce the next step in the expansion process. For each production, matched timeslices are modified either by splitting them or by assigning animation effects. This process continues until all timeslices are terminal symbols in the grammar.

A single expansion step, shown in Fig. 6, is performed in the following manner. (1) We first select a set of non-terminal timeslices with the lowest tags, or their inverted versions, referring to the lowest tagged shapes; tags are sorted by their creation time so we effectively process timeslices in a breadth-first manner. (2) We then find the set of grammar rules that match both the timeslices tag as well as the shapes tag. (3) From these, we choose randomly a rule. (4) We then generate a new set of timeslices by applying the operator referred by rule to each matched timeslice. (5) We finally update the configuration by substituting the matched timeslices with the newly created ones.

3.5 Formal Grammar Definition

Let us now define the grammar more formally. Timeslice grammars act on shapes S and timeslices T that are formally defined as $S = \langle I_S, s_S, \Theta_S \rangle$ and $T = \langle I_T, d_T, A_T \rangle$ where I_S, I_T are the shapes and timeslices tags, s_S the shape's identifier, Θ_S the shapes vertices and colors, d_T the timeslice duration and A_T the timeslice animation. Animation data A_T is stored as a $n \times m \times p$ array of affine matrices, defined by a $n \times m$ uniform spatial grid, keyframed at p times. Productions R in timeslice grammars are written as $R = O(\{p\}) : [\{I_t\}, \{I_s\}] \rightarrow \{I'_t\}$, where O is the operator that will be used when the rule is applied, $\{p\}$ its parameters, $\{I_t\}$ is the set of timeslice tags matched by the rule, $\{I_s\}$ the set of matched shape tags, and $\{I'_t\}$ are the tags assigned to the newly created timeslices.

Timeslice grammars defined two operators. The time splitting operator is defined as $timesplit(\{s_i\}) : \mathscr{T}^R \rightarrow \{I'_{t_i}\}$, where $\{s_i\}$ are the split times, $\{I'_{t_i}\}$ the output tags, and $\mathscr{T}^R$ represents the timeslice and shape matching tags $[\{I_t\}, \{I_s\}]$ of the rule the operator refers to. The animation operator is written as $animate(type, ref, off, \phi) : \mathscr{T}^R \rightarrow \{I'_{t_i}\}$, where $type$ is one of the animation types defined above ("affine transform", "move towards", "follow path", and "fill" or "border" color), ref indicates whether the animation is specified in shape or world coordinates, off the time offset to apply the transform at, and ϕ are the parameters of the transformation (a matrix for affine transforms, the point for move towards, the path for follow path, and the colors for fill and border colors).

Starting from an initial configuration $\mathscr{C}_0$, productions are applied sequentially to the current state $\mathscr{C}_e$ of the animation, until all timeslices are terminals. In timeslice grammars the configuration at a particular step e in the expansion process is described as $\mathscr{C}_e = \langle \mathscr{S}, \mathscr{T}_e \rangle$ where $\mathscr{S}$ are all animated shapes and $\mathscr{T}_e$ the set of all timeslices at that expansion step. At each expansion, the configuration is modified with the procedure described previously, which amounts to removing all timeslices $\mathscr{T}_e^R$ matched by the randomly selected rule R, and adding all timeslices $\mathscr{T}_e^O$ created the the rule operator O. This can be written as $\mathscr{C}_{e+1} = \mathscr{C}_e \cup \mathscr{T}_e^O \setminus \mathscr{T}_e^R$.

3.6 Discussion

The main advantage of timeslice grammars, in fact most grammars for that matter, is that the number of rules remains very compact even for complex animation effects. This comes mainly from four reasons. First, only one time splitting operator is sufficient to specify the temporal structure of complex motion, since the operator is recursively and selectively applied by the grammar to different shape and timeslice groups. Second, the animation operations are closed with respect to composition, since all animations have a common, sampled, representation. This in turn means that complex motion can be choreographed by combining simpler ones with the grammar, without requiring special operations. Third, complex grouping behaviour can be expressed by matching rules to shapes with similar timeslice and shape tags. This in turn lets us easily express complex choreographies using the rule matching mechanism. Finally, the introduction of inverted tags allows us to produce looping animations without duplicating rules.

Table 1: Summary information for all generated animation

Name	Rules	Shapes	Duration	Expansion	Timing
teaser	44	208	6s	60	0.04s
cube	13	54	6s	16	0.04s
abstract	19	68	5s	22	0.03s
panel	19	106	8s	40	0.02s
crack	53	358	4s	55	0.46s
intersections mini	105	44	4s	161	0.07s
intersections	105	8214	8s	161	22s

From left to right: the number of rules and shapes, animation duration, number of expansion steps, and execution times (taking into account both the expansion process and the animation rendering phase).

4 Results and Discussion

4.1 Animations

Grammars. Throughout the paper we show a variety of animations created with our system. Each one was created by hand designing grammars given an input set of shapes. Table 1 summarizes the statistics regarding each grammar for all figures in this paper. In the supplemental material we show the grammar videos and all their rules.

Animation Types. Each animation presented in Fig. 7 tries to focus on some of the advantages that derive from using our grammars. The *cube* animation shows how the keyframed spatial sampling of affine transformations makes this model more compact. To animate the circles in most commercial softwares would mean either applying an individual animation to each of the circles, or the production of a specific script. The *abstract* animation shows one of the applications for the attribute changing effects and how, even in presence of complex asymmetric time subdivisions, this system can still seamlessly produce a looping animation. This is achieved with the use of inverted tags. The *panel* animation displays an example of the variety of effects that can be achieved by creating animations that combine different reference space, either local or global, with and without specifying an offset. The *crack* animation shows, with small individual units connected spatially and temporally, that our model is expressive enough to capture animations with cause-effect relations and that imitate physics-based models (although in this case, we do not think grammars are optimal). One advantage of grammars is that we can prototype animations on simple examples and then scale them to large and intricate sets of shapes. This is shown in the *intersections* example of Fig. 9, where a complex animation is composed by small patches of episodic animations. Finally, the *teaser* animation shown in Fig. 1, demonstrate all the effects applied together.

Expressiveness. The examples shown so far cover a large variety of animation effects, which were created using only 2 operators. This in turn shows that, as a formal model, timeslice grammars are an expressive model for motion graphics. This fact is reinfored by

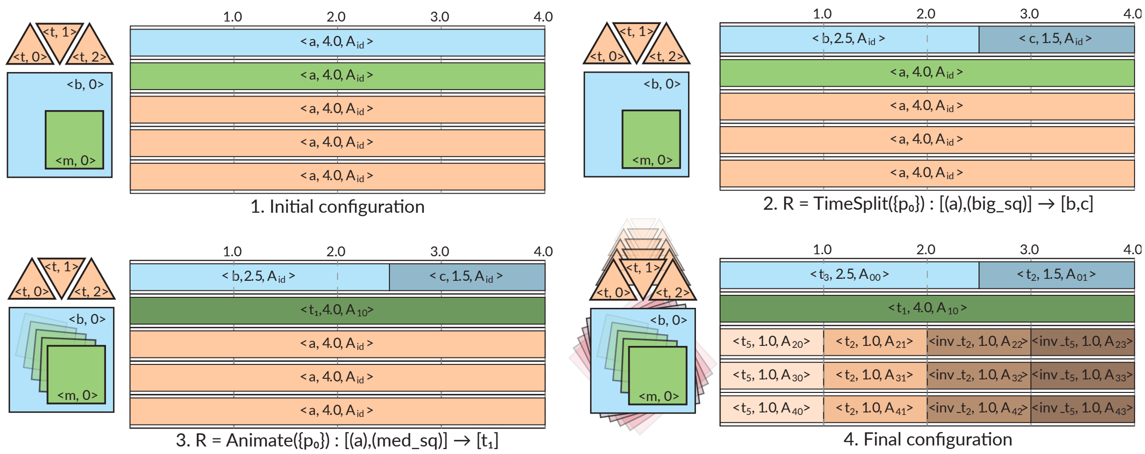

Figure 6: First steps in the expansion process of a simple grammar. (1) The expansion starts from an initial configuration composed by a set of shapes and a set of timelines (one for each shape), in turn initially composed by a single timeslice. (2) The application of the time split operator subdivides the input timeslices into a new set of timeslices, that will be constructed and labeled accordingly to the rule semantic. (3) The application of the animate operator doesn't further subdivide the timelines, but assigns changes to both the animation transforms and decorative attributes in each input timeslice. (4) The final configuration, after all rules are applied. Shapes trails represent the initial frames of their animation, once applied.

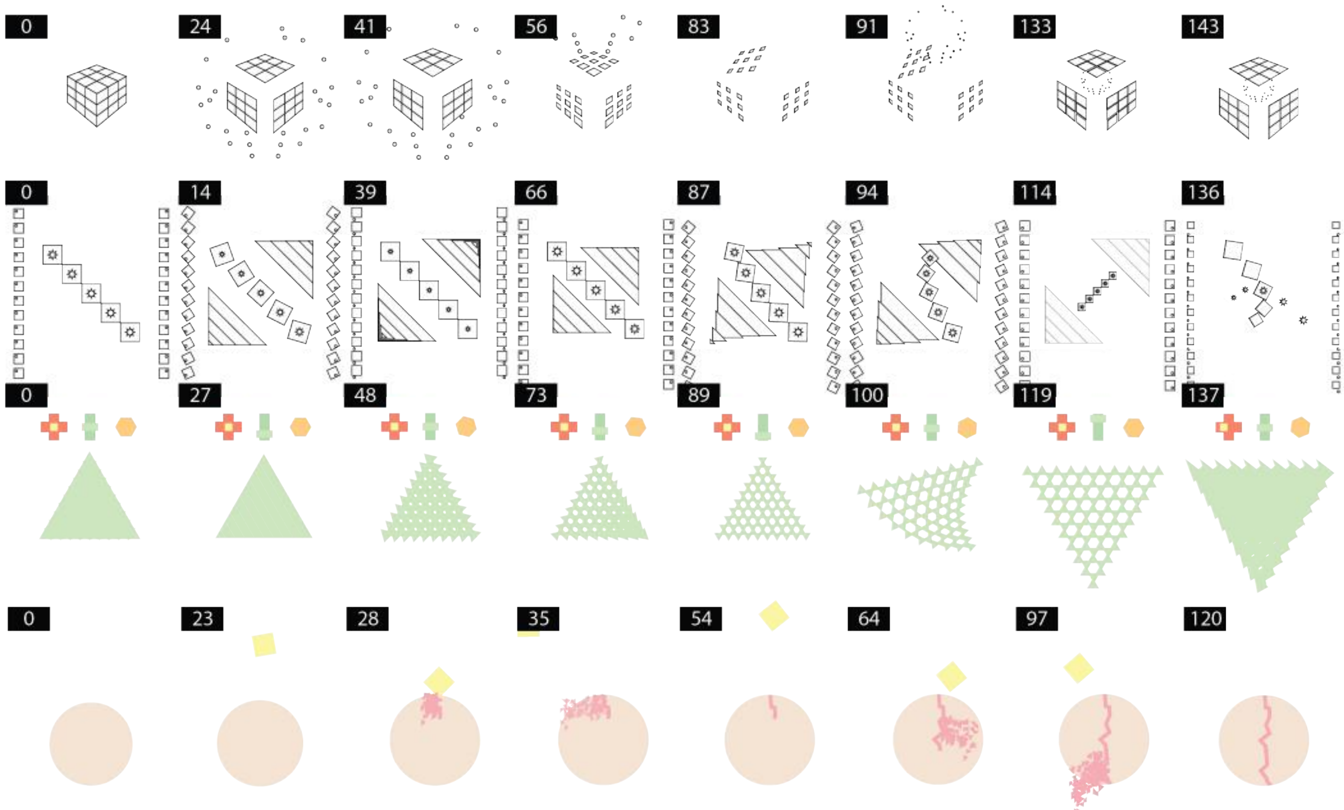

Figure 7: Eight frames from example animations generated with timeslice grammars, composed up to several hundreds of different shapes. The label at the top left of each image indicates the frame number. For the full animations please refer to the supplemental material.

the fact that timeslice grammars are stochastic in nature and can generate motion variations, as shown in Fig. 8 by a series of animations produced with the same grammar.

Design Times. The authors of this paper wrote each of the grammars from scratch and without any user interface. Design times go between half hour to a few hours. The most complex animation to create has been the teaser, since it includes several effects within the same animation, and the crack animation that require precise

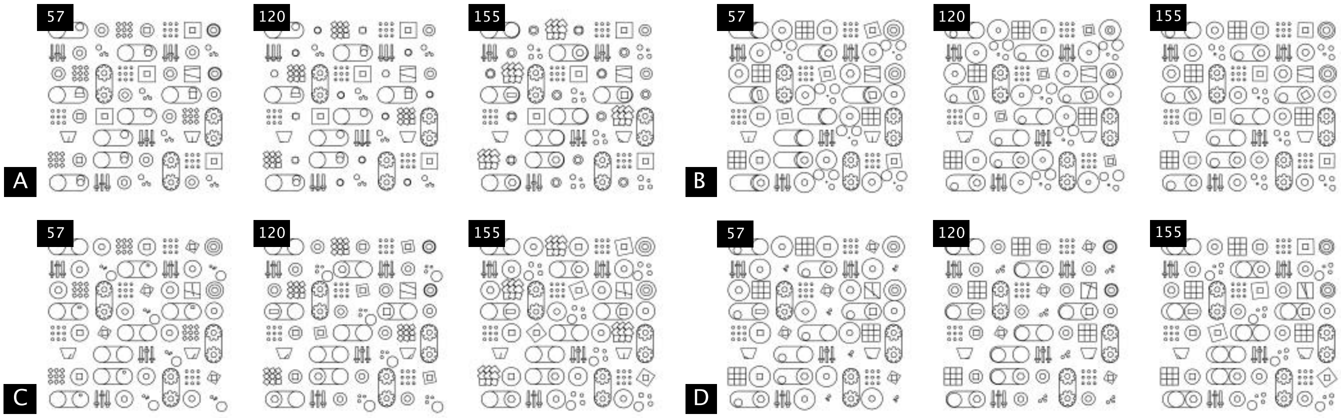

Figure 8: Four animations generated from the same grammar due to the randomized matching process. The starting point is the same for every animation. Please refer to the supplemental material for the full animation.

alignment of the transformations.

We found that overall timeslice grammars are relatively easy to use and that the design time is amortized well over the number of shapes. We found that most of the time was spent manually setting the values of the operators, rather than determining the structure of the grammar. Furthermore, just like any procedural language, we found that using the grammars well required significant training, making it not suitable for first-time or non-technical users.

One benfit of using grammars is that the number of shapes in the animations is significantly less than the number of rules applied to them. In a traditional animation editor, the number of operations would need to be roughly proportional to the number of shapes at least for selection operations.

Performance. From the point of view of the generated animation, the expansion process took less than a second, on a standard desktop, for all animations, but the largest one. This means that an interactive user interface can be built to support editing. The *intersections* animations tests the execution speed of our implementation, taking 22 seconds for 8 thousand shapes with 160 expansions. In this case, most of the time is taken by the interpolation of affine matrices. Note that we can design most of the grammar with a simpler set of shapes and then scale to effect later.

4.2 Limitations

Grammar learning. All the grammars in the paper have been manually written. A common goal in grammar-based systems is to learn the grammar from input data, a topic we have not investigated yet. We believe this might be achieved by analyzing the spatial relations that exists between the input set of shapes and inferring the best animation effects that could be applied.

New operators. Actually it is not possible to apply affine transform to individual vertices to add shape changes, or support true shape morphing. This can be a limitation in types of motion graphics, and might be a possible improvement. By the way our grammar model allows to add a new operator *shapemorph*, which explicitly applies the 2D grid of affine transformations to the single shape vertexes. Nontheless we focused mainly proposing an easy way to write a grammar both compact and enough expressive to create motion graphics.

Beneficial Usage. One question that arises when defining procedural systems is whether the approach is better than hand editing. In our opinion, the benefit of grammars is when designing animations that work in a non-trivial manner on groups of many shapes. Working on single shapes is still possible, but grammars are likely more cumbersome than a direct editing UI. An example of such animation is the crack animation, that took lots of time to create for a small number of shapes. This is a limitation of our approach not in expressiveness, but in convenience. This problem is typical of all procedural systems. For example, writing a split grammar is helpful in architectural modeling when designing cities, rather than a single building.

4.3 Extension: User Interface

User Interface. As an extension to the work presented so far, we implemented a graphics user interface to improve upon the usability of our grammar. Fig. 10 shows a screenshot of the interface and the supplemental video a full editing sequence. Starting with the input shapes and a possibly empty grammar, the interface lets user create grammar rules and edit operator parameters, visually select shape and timeslices, while visualizing the timeline and the final animation in real-time. In the supplemental video we use the interface to recreate the *cube* animation. Compared to hand-editing, the user interface speeds up grammar editing significantly, since (1) the user has real-time feedback on the operations, (2) visual shape and timeslice selection makes rule naming a lot easier, (3) the edited grammar is by construction syntactically and semantically correct, avoiding any error during grammar compilation.

One benefit of using grammars rather than arbitrary scripts is that adding a user interface is straightforward since grammars have a simpler formal model than any commonly-used scripting language, and that model is amenable to direct interfaces, like ours, our node based interface. In our case, we implemented in the interface as a web application wrapping the procedural system, an architecture that allows more technically inclined user to hand-edit the grammar directly if so desired.

Informal User Study. We ran an informal user study to validate whether users can control the spatial and temporal aspect of an animation using our grammar supported by the user interface. In a manner similar to [17], we ask four subjects to match a target animation starting from given grammars, by modifying or creating grammar rules and editing operator parameters. This single task requires various spatial and temporal adjustments and the creation of new rules. Fig. 11 shows the starting and goal animation. All subjects had some knowledge of computer graphics, but were novice to both our grammar as well as animation editing. We did not approach actual designers with the system. We collected users feedback with exit interviews to gain insight into the grammar and interface use. We choose to run an informal study rather than measuring subjects performance to statistical significance, since the latter would be outside the scope of our work and since exit interviews provide more

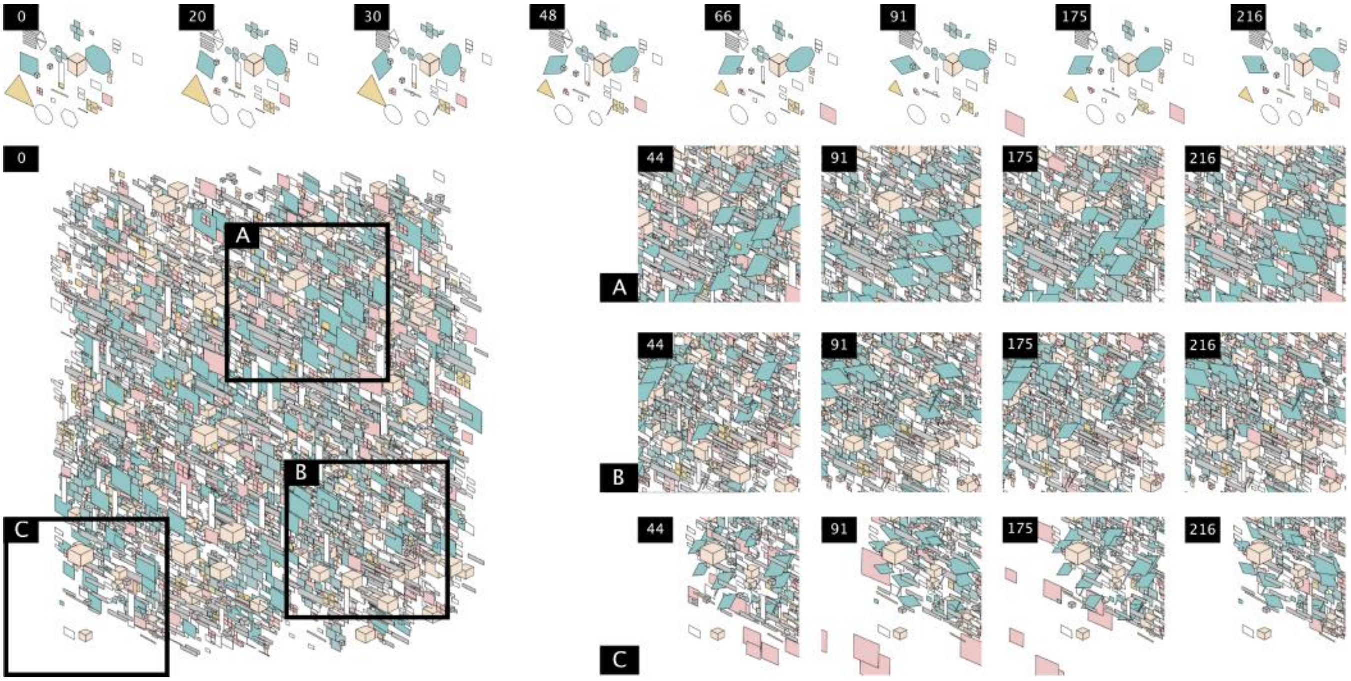

Figure 9: A grammar designed with a small set of shapes (*top*) can be used to create a much larger animation (*bottom*). For the larger case we show insets of four frames each.

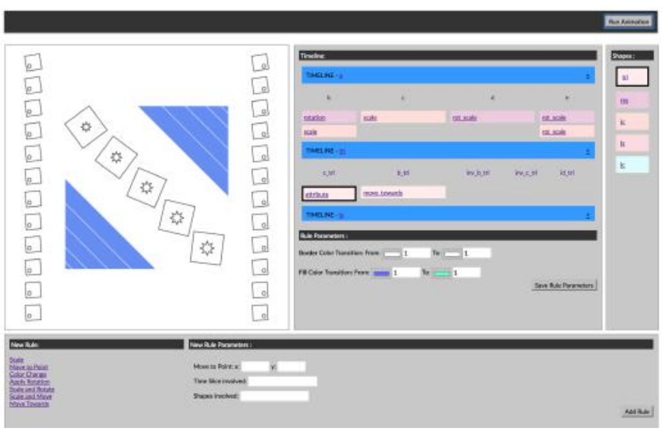

Figure 10: Screenshot of the prototype user interface for editing grammars. *Left:* Aniamtion preview. *Center:* Timeline visualization and operator adjustment. *Right:* Shape list. *Bottom:* Rule creation.

direct feedback into the grammar and interface features.

Overall all subjects were able to complete the given task in between 20 and 30 minutes. Considering that only a very short training phase was performed before the experiment (8 minutes long), this shows that timeslice grammars can effectively be used to control animations. In exit interviews, users had four main observations. First, the immediate feedback gained with the realtime preview allowed them to perform fast parameters tuning by trial and error, leading them to match the target animation with high confidence. Second, the selection preview enabled them to immediately understand which shapes' animations they were editing, without having to keep track on that during grammar writing. Third, subjects mentioned that without the user interface they would have been lost writing the grammar manually. Said another way, the interface allowed them to take advantage of the formalism without requiring lots of training. Finally, our users explicitly mentioned that, to their knowledge, no other tool would allow them to obtain similar animations in such a simple and direct manner.

5 Conclusions

In this paper, we presented *timeslice grammars*, an extension of split and group grammars, for the procedural generation of 2D motion graphics. We showed how our grammar model, together with the described operators, can successfully create complex and compelling animations. As future work, we plan to extend our approach to learn grammars from examples, allow users to have more control over the final generated animations, and consider new operators that could improve the expressivity and compactness of our grammar model.

Acknowledgments

This work has been partially funded by MIUR (project DSurf), Sapienza University of Rome and Intel Corporation.

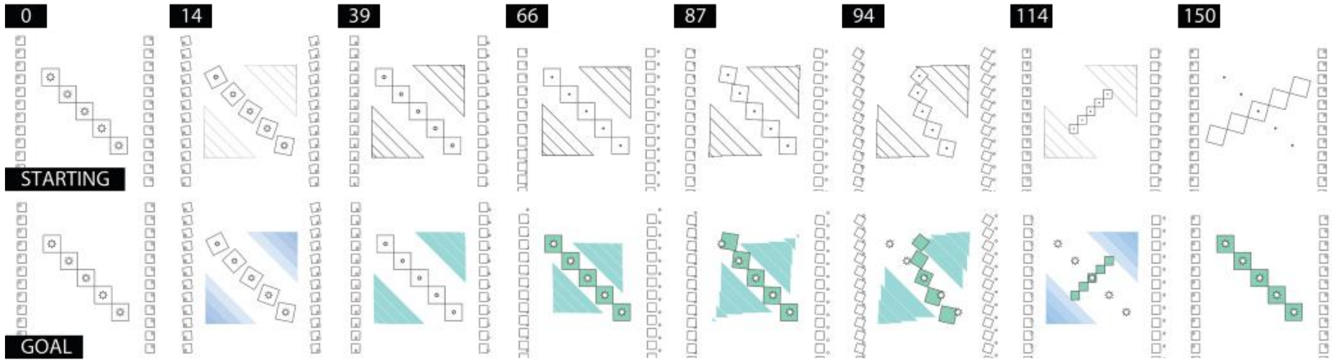

Figure 11: Starting and final animation for the user study task.

References

[1] M. Alexa. Linear combination of transformations. In *ACM Transactions on Graphics*, vol. 21, pp. 380–387, 2002.

[2] R. Baxter, Z. Crumley, R. Neeser, and J. Gain. Automatic addition of physics components to procedural content. In *Proc. of AFRIGRAPH '10*, pp. 101–110, 2010.

[3] J. Bender, K. Erleben, J. Trinkle, and E. Coumans. Interactive simulation of rigid body dynamics in computer graphics. In *EUROGRAPHICS 2012 State of the Art Reports*, 2012.

[4] K. Hyun, K. Lee, and J. Lee. Motion grammars for character animation. *Computer Graphics Forum*, 35(2), 2016.

[5] Y. Li, F. Bao, E. Zhang, Y. Kobayashi, and P. Wonka. Geometry synthesis on surfaces using field-guided shape grammars. *IEEE T. Vis. Comput. Gr.*, 17(2):231–243, 2011.

[6] A. Lindenmayer. Mathematical models for cellular interaction in development: Parts i and ii. *Journal of Theoretical Biology*, 18, 1968.

[7] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim. Unified particle physics for real-time applications. *ACM Trans. Graph.*, 33(4):153:1–153:12, 2014.

[8] S. R. Musse and D. Thalmann. *A Model of Human Crowd Behavior : Group Inter-Relationship and Collision Detection Analysis*, pp. 39–51. 1997.

[9] R. Měch and P. Prusinkiewicz. Visual models of plants interacting with their environment. In *Proc. of SIGGRAPH '96*, pp. 397–410, 1996.

[10] Y. I. H. Parish and P. Müller. Procedural modeling of cities. In *Proc. of SIGGRAPH '01*, pp. 301–308, 2001.

[11] N. Pelechano, J. Allbeck, and N. Badler. Controlling individual agents in high-density crowd simulation. In *Proc. of 2007 ACM Symposium on Computer animation*, pp. 99–108, 2007.

[12] S. Pirk, B. Benes, T. Ijiri, Y. Li, O. Deussen, B. Chen, and R. Měch. Modeling plant life in computer graphics. In *ACM SIGGRAPH 2016 Courses*, pp. 18:1–18:180, 2016.

[13] P. Prusinkiewicz, M. S. Hammel, and E. Mjolsness. Animation of plant development. In *Proc. of SIGGRAPH '93*, pp. 351–360, 1993.

[14] P. Prusinkiewicz and A. Lindenmayer. *The Algorithm Beauty of Plants*. Springer, 1990.

[15] W. T. Reeves. Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, 2(2):91–108, 1983.

[16] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proc. of SIGGRAPH '07*, SIGGRAPH '87, pp. 25–34, 1987.

[17] C. Santoni and F. Pellacini. gtangle: A grammar for the procedural generation of tangle patterns. *ACM Trans. Graph.*, 35(6):182:1–182:11, 2016.

[18] M. Schwarz and P. Wonka. Practical grammar-based procedural modeling of architecture: Siggraph asia 2015 course notes. In *SIGGRAPH Asia 2015 Courses*, pp. 13:1–13:12, 2015.

[19] W. Shao and D. Terzopoulos. Autonomous pedestrians. In *Proc. of 2005 ACM Symposium on Computer Animation*, pp. 19–28, 2005.

[20] J. Stam. Nucleus: Towards a unified dynamics solver for computer graphics. In *IEEE International Conference on Computer-Aided Design and Computer Graphics*, pp. 1–11, 2009.

[21] G. Stiny. Introduction to shape and shape grammars. *Environment and planning B*, 7(3):343–351, 1980.

[22] G. Stiny. Spatial relations and grammars. *Environ. Plan. B - Plan. Des.*, 9(1):113–114, 1982.

[23] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. *ACM Trans. Graph.*, 22(3):669–677, 2003.