Geometry Synthesis on Surfaces Using Field-Guided Shape Grammars

Yuanyuan Li, Fan Bao, Eugene Zhang, *Member, IEEE,* Yoshihiro Kobayashi, and Peter Wonka *Member, IEEE,*

Abstract—We show how to model geometric patterns on surfaces. We build on the concept of shape grammars to allow the grammars to be guided by a vector or tensor field. Our approach affords greater artistic freedom in design and enables the use of grammars to create patterns on manifold surfaces. We show several application examples in visualization, anisotropic tiling of mosaics, and geometry synthesis on surfaces. In contrast to previous work we can create patterns that adapt to the underlying surface rather than distorting the geometry with a texture parametrization. Additionally, we are the first to model patterns with a global structure thanks to the ability to derive field-guided shape grammars on surfaces.

Index Terms—shape grammars, tensor fields, vector fields, surfaces, geometry synthesis

1 Introduction

In this paper we introduce the concept of field-guided shape grammars. In previous work, shape grammars have been used to model geometric or organic patterns in the plane and three-dimensional objects like architecture and trees. Extending shape grammars with fields will result in a modeling approach that can create three-dimensional geometry or texture on surfaces. This approach can also provide more freedom in artistic design.

One of the main results of our framework is that we can not only generate local and stationary patterns (geometric textures) on surfaces such as previous work [1] but also patterns with a global structure. We show two example patterns with a global structure. In the tree example (Fig. 2) the branching structure is global and cannot be produced by a texture synthesis or geometric tiling algorithm. In the radial pattern (Fig. 5) the structure is global because shapes are aligned according to concentric circles increasing in size. As comparison we show an example for a local pattern in Fig. 3 right.

Our approach is to use field-guided shape grammars. We consider a variety of fields, such as vector fields, second-order tensor fields, and higher-order tensor fields. A field can be used to guide a grammar in multiple ways. First, the spatial relationship of shapes is defined through curves that are streamlines [2] or hyperstreamlines in a field. Second, the grammar can use a field to influence parameters such as parameters for rotation, scaling, and color. Third, the grammar can use the field in the rule selection process itself.

The major contributions of this paper are the following:

- We introduce field-guided shape grammars. We show that field-guided shape grammars are able to create a large class of designs that are difficult to create by other methods.
- In contrast with the existing geometric texture synthesis approaches, we are the first to show the design of patterns with a global structure on surfaces.

In Section 2, we review previous work related to our approach including a discussion on strength and limitations in designing global geometry patterns on surfaces. In Section 3, shape grammars that can procedurally generate patterns in the Euclidean plane are illustrated. Section 4 describes our main framework for Field Guided Shape Grammars, including how to generate a guiding vector or tensor field as well as how the guiding field is used for pattern synthesis on surfaces. We also describe a simple pattern optimization technique in this section. In Section 5 , we demonstrate the capability of our approach with a number of applications such as image-based mosaic tiling, tensor field visualization, and modeling of geometry on surfaces. We discuss the strength and limitations of our approach in Section 6 and conclude in Section 7.

2 Related Work

In the following section we review related work in the area of procedural modeling, vector and tensor field design and processing, graph and tiling design, and texture and geometry synthesis on surfaces.

Procedural Modeling with grammars has several successful applications in computer graphics. L-systems have been very effectively applied to plant modeling [3] and street modeling [4]. Comparable to our work L-systems use transformations on a local coordinate system, similar to a LOGO-style turtle. While L-systems have been extended to interact with their environments, such as general surfaces [5], [6], height fields [4], and curves [7], they usually do not consider several challenges of the geometry itself that are important for generating patterns on surfaces. Existing methods can grow patterns in the plane or 3D space but not on surfaces. Our work is also related to shape grammars in architecture [8], [9], [10] in that we use the notation and scope concept (coordinate

- *Y. Li, F. Bao, Y. Kobayashi, and P. Wonka are with the PRISM lab, Arizona State University, Tempe.*
  E-mail: liyuan84|dr.yoshihiro.kobayashi|pwonka@gmail.com
- *E. Zhang is with the School of Electrical Engineering and Computer Science, Oregon State University, Corvaliis.*
  E-mail: zhange@eecs.oregonstate.edu

```
start ~ S(0.5, 0.5, 0.5) P;

P ~ [RZ(Rand( -5,   5)) T(Rand(0.4, 0.45), 0, 0s) M A]
    [RZ(Rand( 85,  95)) T(Rand(0.4, 0.45), 0, 0s) M A]
    [RZ(Rand(175, 185)) T(Rand(0.4, 0.45), 0, 0s) M A]
    [RZ(Rand(265, 275)) T(Rand(0.4, 0.45), 0, 0s) M A];

A ~  RZ(Rand( -5,   5)) T(Rand(0.6, 0.65), 0, 0s) M C ;

C ~ [RZ(Rand( -5,   5)) T(Rand(0.6, 0.65), 0, 0s) M A]
    [RZ(Rand(-85, -95)) T(Rand(0.4, 0.45),
        Rand( 0.3,  0.35), 0s) M A]
    [RZ(Rand( 85,  85)) T(Rand(0.4, 0.45),
        Rand(-0.3, -0.35), 0s) M A] : 0.5;

C ~ [RZ(Rand( -5,   5)) T(Rand(0.6, 0.65), 0, 0s) M A]
    [RZ(Rand(-85, -95)) T(Rand(0.6, 0.65),
        Rand( 0.3,  0.35), 0s) M A] : 0.2;

C ~ [RZ(Rand( -5,   5)) T(Rand(0.6, 0.65), 0, 0s) M A]
    [RZ(Rand( 85,  95)) T(Rand(0.6, 0.65),
        Rand(0.25,  0.35), 0s) M A] : 0.2;

C ~  RZ(Rand( -5,   5)) T(Rand(0.6, 0.65), 0, 0s) M A :
     0.1;

M ~ [ S(0.1, 0.55, 0.1) RZ(90)
      I("cylinder.obj")]
    [ S(0.1,  0.1, 0.1) T(0, Rand( 1,  1.2), 0s)
      I("ico2.obj")]
    [ S(0.1,  0.1, 0.1) T(0, Rand(-1, -1.2), 0s)
      I("ico2.obj")];
```

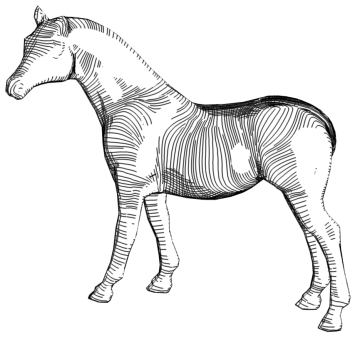


Fig. 1. In this paper we describe how to model geometry patterns on surfaces. We use a shape grammar (left) that is guided with a user-designed tensor field (middle) to produce the final result (right).

system plus scale) presented by Mueller et al. [8]. At a lower level, generative mesh modeling can produce complex surfaces from simpler ones [11] similar to subdivision surfaces.

Vector and Tensor Field Design: Vector field design refers to generating a vector field that satisfies user constraints. Vector field design has many applications such as texture synthesis [12], non-photorealistic rendering [13], [14], and hair modeling [15]. Vector field design has been recently extended to tensor field design [16] and higher-order tensor field design [17], with applications in non-photorealistic rendering [16], [17], street modeling [18], geometry remeshing [19], [20], [16], and surface parametrization [20]. There has been an abundance of recent research in developing techniques and systems for the design and processing of vector and tensor fields [21], [16], [22], [17], [23], [24].

Pattern Design: Similar to L-systems, leaf venation patterns [25] can be created by growing a graph. Tile patterns can be classified according to the regularity of the patterns and according to the complexity of the shapes. The simplest designs are circle layouts or points samples [26], [27], [28], [29]. In the planar case rectangles can be used to create mosaic patterns guided by a vector field [30]. Jigsaw image mosaics can place more general shapes, but do not offer much control over the orientation [31]. Shapes can also be placed by example [32]. In architecture, optimization can be used to place tiles on a surface [33], [34], [35]. An interesting application of tiles on surfaces are brick designs [36].

Texture Synthesis: Example-based texture synthesis [37], [38], [39], [40] can be used to improve the visual detail on meshes. Patterns of geometric elements have also been generated using a biologically-motivated cellular development simulation [41] together with a constraint to keep the cells on a surface. The patch-based approach [42] [43], [44] is a popular technique among example-based texture synthesis that minimizes the seams between patches through searching for the "min-cut". Bhat et al. proposed an algorithm [45] to extend the idea of texture synthesis to 3D volumetric textures. Mesh quilting [1] is a geometric texture synthesis algorithm to spread a 3D texture sample given in the form of a triangle mesh. Although this approach can generate nice geometry patterns, it also has several limitations. First, due to the inherent characteristics of texture synthesis, it can only generate local and stationary patterns which look similar everywhere. Patterns with global structures (see Fig. 2) cannot be generated with texture synthesis. Second, as an example-base approach, a manually designed mesh is required as input. In addition, how to spread the pattern is usually hard-coded. In contrast, in our approach the user can flexibly control this procedure using grammar rules. Finally, mesh quilting on surfaces depends on local parameterizations of surface patches, so that patterns in regions with very high curvature can be significantly distorted.

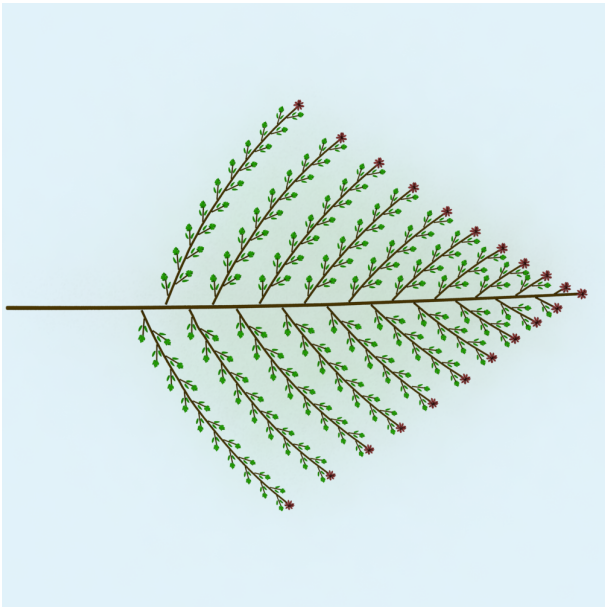


Fig. 2. A tree pattern with a global structure on a plane. This pattern cannot be generated by taking small example patches of a tree and quilting them.

3 Shape Grammars in the Euclidian Plane

In this section we explain how to use shape grammars to model patterns in the plane. This lays the foundation for designing

patterns on surfaces (see Section 4).

In this paper, we build upon existing work in grammar-based modeling, mainly *CGA Shape* as introduced by Mueller et al. [46]. For the examples illustrated in this paper we will mainly use the scope commands *add*, *scale*, *translate*, and *rotate* (originally used in L-systems [3]). Our contribution is the incorporation of vector and tensor fields into the grammars and an extension to modify the behavior of rules by local optimization using collision detection and shape merging. These parts are described in Section 4. We will refer to the grammar as *Field-Guided Shape Grammar* (FGSG). The grammar is classified as a sequential grammar of which Chomsky grammars [47] and *CGA Shape* are typical examples. In the following we briefly review the most relevant concepts of the grammar and give examples of how the grammar can model patterns in the plane. Then we will explain how the grammar can be embedded into fields and moved onto surfaces.

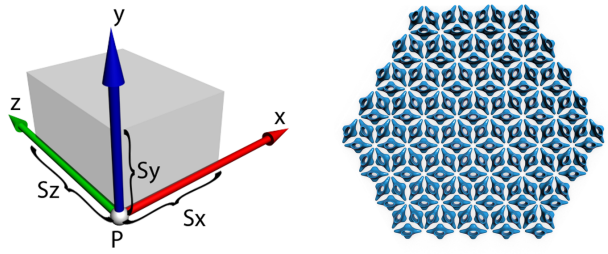


Fig. 3. Left: The scope of a shape. The point P, together with the three axes X, Y, and Z and a size vector S define a box in space that contains the shape. Right: A pattern in the plane.

Shape: The grammar works with a configuration of shapes. A shape consists of a symbol (string), geometry (geometric attributes), and numeric attributes. Shapes are identified by their symbols which is either terminal ($\in \Sigma$) or non-terminal ($\in V$). The corresponding shapes are called terminal shapes and non-terminal shapes. The most important geometric attributes are the position P, three orthogonal vectors X, Y, and Z describing a coordinate system, and a size vector S. These attributes define an oriented bounding box in space called *scope* (see Fig. 3).

Production process: A configuration in grammar is a finite set of basic shapes. The production process can start with an arbitrary configuration of shapes A, called the axiom, and proceeds as follows: (1) select an active shape with a symbol B in the set, (2) choose a production rule with B on the left hand side to compute a successor for B, a new set of shapes $BNEW$ (3) mark the shape B as inactive and add the shapes $BNEW$ to the configuration and continue with step (1). When the configuration contains no more non-terminals, the production process terminates. To control the derivation, we assign a priority to all rules to obtain a (modified) breadth-first derivation: we simply select the shape with the rule of highest priority in step (1). If priorities are not used we can select to follow a depth-first or breadth-first derivation.

Notation: Production rules are defined in the following form:

id: predecessor : cond $\rightsquigarrow$ successor : prob

where *id* is a unique identifier for the rule, *predecessor* $\in V$ is a symbol identifying a shape that is to be replaced with *successor*, and *cond* is a guard (logical expression) that has to evaluate to true in order for the rule to be applied.Later we will see that *cond* is helpful to adapt the production with respect to the properties of surface geometry and tensor field (see Section 4.3). The rule is selected based on a probability *prob*. For example, the rule

1: A $\rightsquigarrow$ M C : 0.5

replaces the shape A with shapes M and C with a probability of 0.5.*prob* is very helpful in generating stochastic patterns (See Fig. 1). To specify the successor shapes we use different forms of rules explained in the remainder of this section.

Scope-transformation rules: We use scope commands *add*, *scale*, *translate*, and *rotate* to modify shapes: $T(t_x,t_y,t_z)$ is a translation vector that is added to the scope position P, $Rx(angle)$, $Ry(angle)$, and $Rz(angle)$ rotate the respective axis of the coordinate system, and $S(s_x,s_y,s_z)$ sets the size of the scope. We use [and] to push and pop the current scope on a stack. Any non-terminal symbol $\in V$ in the rule will be created with the current scope. Similarly, the command $I(objId)$ adds an instance of a geometric primitive with identifier $objId$. Any three-dimensional model can be used.

Example: The example grammar below illustrates the design of a planar pattern by growing outwards from a seed shape. The result is depicted in Fig. 4. An example of a pattern with a global structure is shown in Fig. 5.

```
1: Axiom ⇝ P;
2: P ⇝ [ RZ(30) T(0.6, 0, 0) A ]
     [ RZ(150) T(0.6,0, 0) A]
     [ RZ(-90) T(0.6,0, 0) A];
3: A ⇝ B [ T(0.6, 0, 0) RZ(60) T(0.6, 0, 0) A ]
     [ T(0.6, 0, 0) RZ(-60) T(0.6, 0, 0) A ];
4: B ⇝ S(1.0, 1.0, 0.6) I("torus");
```

4 Field-Guided Shape Grammars

In this section we describe how to design patterns on surfaces using field-guided shape grammars. We first give a motivation for our design choices (Section 4.1). Then we describe the tools we use to design a vector or tensor field on surfaces (Section 4.2). In Section 4.3 we explain a new algorithm for deriving shape grammars on surfaces while querying the field. During the grammar derivation, the pattern is optimized locally by integrating collision detection and shape merging (see Section 4.4).

4.1 Why Field-Guided Shape Grammars?

At the core of our approach is the use of fields in various aspects of the original shape grammars. These aspects include:

1) using a field to guide the translation command *translate*.

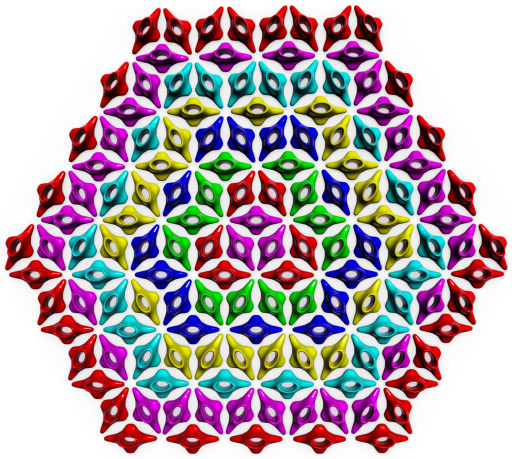


Fig. 4. This example shows a design in the plane. The colors denote the order of placement and are not part of the design. The pattern grows outward starting from a single scope (symbol) placed in the middle of the pattern.

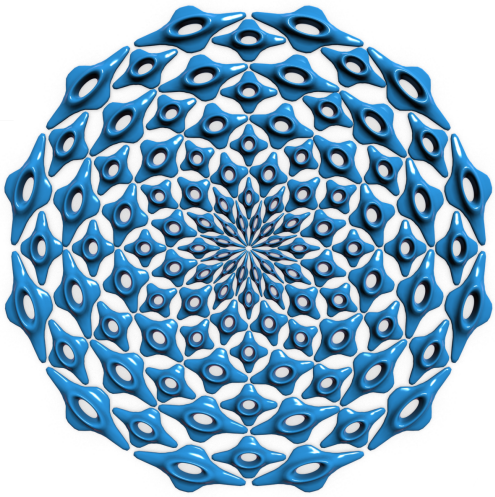


Fig. 5. A radial pattern with a global structure.

2) using another field or its properties as parameters in the shape grammar commands. We give two common examples: (1) controlling the rotation of the scope relative to a vector or tensor field and (2) using the length of vectors in a vector field or eigenvectors in a tensor field to control the scaling or amount of translation of a shape.
3) allowing attributes of the field to select production rules.

Here a tensor refers to a matrix, i.e, the tensor is of *second-order*. However, our system can also make use of a scalar field, a vector field or a high-order tensor field, in which the order is higher than two.

There are a number of justifications for a field-guided shape grammar. First, it can be directly applied to surfaces. In contrast, the original shape grammar would need a (global) parametrization. We believe that it is significantly simpler to design a field then to design a parametrization. The field as well as the surface normal provide a good local frame for propagation of shapes on a surface. Second, by mapping properties of a field to the shapes generated by the grammar, we provide control for local pattern properties, such as the size and the orientation of shapes. Third, by incorporating tensor fields into the grammar, we can achieve different layouts by providing different input tensor fields which are used to guide the growth of the shape tree (Fig. 12 shows different fields and their corresponding artistic design or image mosaicing).

In the remainder of this section, we will describe what modifications are needed for field-guided shape grammars. We start out by describing the field design process (Section 4.2). Then we describe field-guided shape grammars (Section 4.3) and two strategies for optimizing shape placement (Section 4.4).

4.2 How to Design a Field?

While FGSG works with a variety of fields, such as vector fields, second-order tensor fields, and higher-order tensor fields, we limit our description to the second-order tensor field case. A tensor field T for a manifold surface M is a smooth tensor-valued function:

$$T(p) = \begin{pmatrix} T_{11}(p) & T_{12}(p) \\ T_{21}(p) & T_{22}(p) \end{pmatrix}, p \in M \tag{1}$$

A point p is degenerate (singular) when $T_{11}(p) = T_{22}(p)$ and $T_{12}(p) = T_{21}(p) = 0$. At a degenerate point, the tensor is isotropic and the eigenvectors are undefined. There are a variety of restrictions that can be placed on the field. For example, eigenvectors of non-singular tensors can be limited to vectors of unit length, or the angle between both unit eigenvectors can be limited to $\frac{\pi}{2}$. Using both of these restrictions yields tensors of the form:

$$T(p) = \begin{pmatrix} cos(2\theta) & sin(2\theta) \\ sin(2\theta) & -cos(2\theta) \end{pmatrix} \tag{2}$$

with major and minor eigenvectors directions θ and $\theta + \pi/2$, respectively. To design a tensor field on surfaces, we adapt the interactive tensor field design algorithms of [16], which has two stages, initialization and editing. We describe our implementation next.

During the initialization stage, the tensor field is designed with user-specified design elements placed on surface. The user can choose either a regular element, which can guide the nearby tensor's direction, or a singular element, which can be a wedge, a trisector, a node, a center or a saddle (see Fig. 6 [16]). The tensor elements are propagated over the surface usingGaussian *radial basis functions* (RBF) to define a basis field. This propagation requires the computation of a *geodesic polar map* [48]. For each vertex p on the surface, we assign polar coordinates (ρ, θ) with respect to the center c of a design element d. The shortest path between p and c on the

surface is called a *geodesic* [48]. The magnitude component ρ is the geodesic distance, i.e. the length of the geodesic. The angular component θ is the angle of the geodesic with respect to some local frame at c. Wecalculate the geodesic distance using fast marching with a spherical wavefront propagation algorithm [49] to get the distance component ρ. Based on geodesic distances, the angular component can be computed by accelerated particle tracing. The user can place multiple elements on the surface and the final result is obtained by adding upthe tensor components contributed by each individual basisfield.

$$T(p_i) = \sum_j e^{-d_j \rho_{ij}} T_j(\rho_{ij}, \theta_{ij}) \quad (3)$$

The variable p_i denotes a vertex on the surface, $T_j(\rho,\theta)$ denotes the j-th basis function (using polar coordinates), and d_j denotes the decay constant for $T_j(\rho,\theta)$. ρ_{ij} and θ_{ij} are the polar coordinates of vertex i with respect to basis element j. The generated field is a continuous tensor field on the surface, but there are usually some unnecessary singular points.

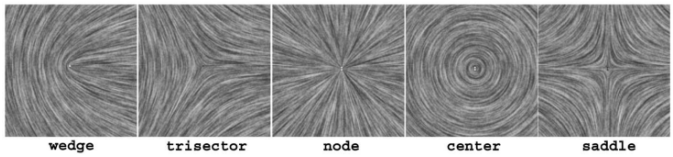


Fig. 6. Five types of singular elements used for tensor field design.

In the editing stage, the user can smooth regions by componentwise discrete Laplacian local smoothing [16] and modifying tensors at vertices directly. The main purpose of this stage is to reduce the number of unwanted singularity pointsFig. 7 provides an example that shows how tensor field smoothing can reduce the number of singularities in the field and increase the smoothness of the field.

Field-Guided Shape Grammars are independent of the field design. While we reimplemented many components of a field design system, we additionally allow the loading of an arbitrary field from a file. The fields that we consider store vectors or tensors at vertices. Note that the field elements are defined in the tangent plane of the vertex. The tangent plane of a vertex is typically not coplanar with any of its adjacent faces. Therefore, the interpolation of fields within polygons (triangles) on the surface is a non-trivial problem. We use a piecewise interpolation scheme [16] to obtain a continuous tensor field on the mesh surface based on the values at vertices. This scheme was adapted from vector fields on surfaces [21].

4.3 Field-Guided Shape Grammars

As with the original shape grammar we model most patterns by growing them from one (or multiple) seed shape outwards. With our interactive user interface it is possible to model the axiom (Section 3) of the grammar by placing one or multiple scopes in the field and to assign a symbol to each of

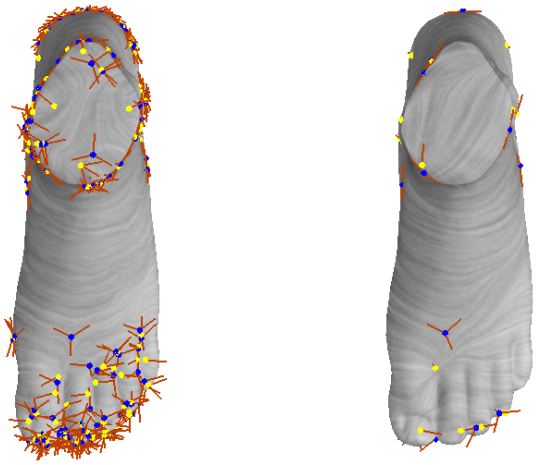


Fig. 7. Tensor fields before and after smoothing on a foot model. The colored dots are singularities (yellow=wedge, blue=trisector). The left one is the field before smoothing, while the right one is the field after 300 iteration of local Laplacian smoothing. Notice smoothing reduces both the geometric complexity as well as the topological complexity (number of degenerate points) in the field.

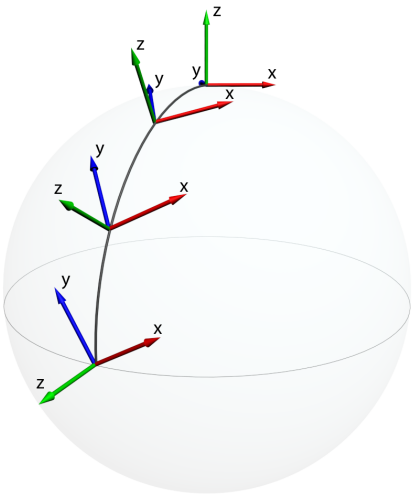


Fig. 8. Translating local frames along a curve which may be a streamline in a vector field or a hyperstreamline in a tensor field.

these scopes. We consider three ways how a tensor field can guide a shape grammar during the derivation: 1) The field can change the translation command, 2) The field can be queried to determine parameters, and 3) The field can be queried to select rules. We describe the three concepts in the following.

The grammar can use an arbitrary number of input fields. A typical setup is to use one tensor field to guide the translation command F_1 and a second tensor field to guide the rotation of shapes on a surface F_2. This is important for pattern design, because the direction of growth of a pattern is not necessarily identical with the direction of shape alignment (even though these fields are also often the same).

Guiding the translation command: Unlike the original shape grammar, the translation command $T(t_x,t_y,t_z)$ cannot be simply computed by vector addition in the Euclidean space.Instead we want to transport the scope on the surface while maintaining the angle between the orientation of the scope and the eigenvectors of the tensor field. Each scope defines a local coordinate system with axis X_{scope}, Y_{scope}, and Z_{scope} and has an origin P_{scope}. The origin is associated with a point on a surface $P_{surface}$. The point $P_{surface}$ is itself associated with a vector or tensor in the field on the surface. A translation along (t_x,t_y) is treated differently than a translation along t_z. A translation in the t_z direction simply changes the offset of the position P from the surface, i.e. a translation along t_z moves the scope along the surface normal at $P_{surface}$. A translation in the plane spanned by (t_x,t_y) is handled by mapping the translation direction into the tangent plane. This mapping defines a (hyper)streamline l in the field that goes out in that direction. We then canmove the scope in the field such that the angles between the scope and the (hyper)streamline as well as the offset to the surface remain constant. See Fig. 8 for an illustration of a scope being transported along a streamline (a vector field) or a hyperstreamline (a tensor field).A (hyper)streamline is a curve that is tangent to an (eigen)vector field everywhere along its path. Please note that in general we trace at an angle within the field,i.e. along a rotated version of the original vector field or eigenvector field. Everywhere along the streamline l the streamline has the same angle to the vectors in the vector field or the major eigenvectors in the tensor field. This angle is $\theta = atan2(t_x,t_y)$, and the length of streamline is $\sqrt{t_x^2+t_y^2}$. This is slightly more complicated for tensor fields that have arbitrary angles between eigenvectors, i.e., asymmetric tensor fields. When using the translation command we implicitly assume that the translation uses the first specified field. We adopt the approach in [13] to trace the hyperstreamline. Tracing a hyperstreamline near a degenerate point, such as a wedge or trisector, can lead to undesirable artifacts in geometry placement. For example, if an integration point is exactly on the degenerate point, tracing will stop as no outgoing direction is available. While stopping tracing is one option, it is not optimal as the growth of patterns might be accidently terminated in certain directions. To overcome this problem, we propose the following heuristic.

Suppose that a triangle T contains a degenerate point d in its interior. Consider a hyperstreamline l that has just entered T from one of the edges e. The intersection point between l and e is p_{in}. Our heuristic involves a three step process: 1) We continue regular tracing starting at p_{in} until a jump condition is met at a point p_n. A robust jump condition for a point p_i on the hyperstreamline is to test if p_i is close to the singularity d, i.e. if $||p_i - d|| < \varepsilon$. 2) We reflect p_n with respect to the singularity d to obtain the jump destination p', $p' = 2d - p_n$. If p' is outside of T, we clamp it to the edge of T. 3) We compute a new tracing direction at p' and continue regular tracing until the hyperstreamline leaves the triangle at point p_{out}. For the trisector case, we require that the outgoing direction at p' has the same oriented angle with respect to p_np' as the incoming direction at p_n. For the wedge case we use the negative oriented angle. As a result the angle of the tracing direction at p' with respect to the tensor field is usually different from the angle at p_n with respect to the field. While the heuristic is relatively straightforward, it works reasonably well in practice, i.e., for the test models we have employed for this paper. See Fig. 9 for an illustration.

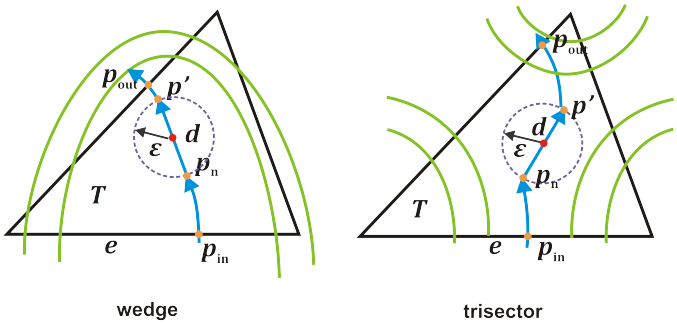


Fig. 9. Illustration of tracing through a singular point. In the wedge case (Left), if the streamline is too close to the singular point, it would turn back to the incoming direction without special treatment. See the text in the paper for an explanation of the tracing algorithm in this special instance.

Querying parameters of the field: We can access field values associated with the current surface point $P_{surface}$ in the grammar. Similar to previous shape grammars we use the notation of the form $Fi.xxx$, e.g. $F1.vector1$ to denote the major eigenvector of the first (tensor) field. In the following we describe three different categories of values that can be queried:

Given a vector field V and a point $P_{surface}$, we can query the following information: (a) the field value $V(P_{surface})$, (b) the Jacobian $\nabla V(P_{surface})$, and (c) the divergence, curl, andcurvature ([50], page 6).

Given a second-order tensor field T, we can query these parameters: (a) the field value, (b) the tensor gradient $\nabla T(P_{surface})$, (c) the major and minor eigenvalues and eigenvectors, (d) the mean, principal, and Gaussian curvatures when the tensor field is a curvature tensor field, and (e) the divergence and curl of each eigenvector field.

In addition, we can query the relationship between multiple co-existing fields at point $P_{surface}$ and relationships between the scope and the field. For example, we found it often helpful to query the angle between an axis of the scope, e.g. the X axis, and a direction vector of the field, e.g. the first major eigenvector of a tensor field. In this example we would use the notation $F1.Xangle1$. This is useful to align the scope with a field just before using the instancing command to place the shape. We also have commands to query derived information, such as the length of vectors (e.g. $F1.length1$).

Selecting rules based on the field: The rules in a shape grammar can have a condition block that can also make use of field values. We simply allow the condition block of a rule to access the same values as the command parameters. The

example below shows how glyph shapes can be selected based on curvature values for the visualization in Fig. 11.

```
1: S : (F1.mincurv>=0) && (F1.maxcurv>=0) ⇝ I("Ellipse1.obj")
2: S : (F1.mincurv>=0) && (F1.maxcurv<0) ⇝ I("Ellipse2.obj")
3: ...
```

4.4 Optimized Shape Placement

During the process of shape distribution, a shape may be assigned to a position that partially overlaps with other shapes. In order to test and remove unwanted spatial overlap, optional collision detection and shape merging are introduced in our solution. The mechanisms are explained as follows.

Collision: A collision detection mechanism tests for intersections between shapes and between patterns. The low level collision detection tests whether a newly generated shape overlaps with any other previously-generated shape. Several different mechanisms can be use to check whether two shapes intersect each other: (1) calculating the Euclidean distance of representative points of two shapes (e.g. the shape centers, or points on the boundary), (2) calculating the geodesic distance of representative points of two shapes, (3) calculating the minimal distance between two shapes using a distance field, (4) detecting a collision based on the precise mesh representation. The first two approaches are less accurate, but much faster. The latter two are more precise, but need more time to compute. After experimenting with all four methods, we have finally decided that the collision detection algorithm method (4) is fast enough and has the least drawbacks for our application. We use the Bullet Physics Library [51] to detect the collision between shapes. We use a specific instancing command *IC* to insert a shape with collision testing enabled or alternatively turn on collision testing for all shapes via a global flag. We also allow the user to use collision detection on a set of shapes. The goal is to consider a collision of one shape in the set as collision of the whole set. A new Command *Lock* is introduced to generate a set of shapes as a whole and test collision on the complete set. An example is given below:

```
1: Pattern ⇝ Lock{ Component_1 | Component_2 | ... | Component_k }
2: Component_1 ⇝ ...
3: ...
```

Another issue is how to stop the derivation when a collision is detected. One strategy is to omit the shape in collision and to continue to derive new production rules. This can cause problems in some situations, because the placement of new shapes only makes sense if the previous shapes have been successfully placed. Therefore, we also use a second option, where we cull the rest of a command string in a production if a collision is detected. In the example below, we assume that $I(\text{"}X.obj\text{"})$ results in a collision. We would further derive shape B, but cull shapes D and E.

```
1: A ⇝ B I("X.obj") D E
```

Unfortunately, the instancing command often only appears after other intermediate rules. In the example below instancing only happens in rule 3. In our experience, it is important to detect this case and consider the shape C to be in collision and still cull shapes D and E. We therefore derive simple rules that only place one shape in a depth-first manner to detect collisions.

```
1: A ⇝ B C D E
2: C ⇝ F
3: F ⇝ I("X.obj")
```

Shape Merging: Sometimes a shape of the same kind overlaps with another shape. In this case we might not want to remove the second shape, but merge the position and orientation of both shapes. To do this we introduce a new instancing command *IM* that can give a threshold for the maximally allowed distance and maximally allowed rotation for two identical shapes to be merged.

Shape merging can be divided into two categories, one is merging of points (i.e. origin of two coordinate frames), the other is merging of line segments. Merging of point shapes is simple, when the distance (either Euclidean or geodesic) between two shapes is smaller than a certain threshold, we only keep one of the shapes which we move towards the other shape as if it was attracted by some force.

Shape merging of line segments is particularly important because it can remove the discontinuity at the intersection of different lines. A linear structure is composed of a series of short segments, each one of which is connected with its direct predecessor and successor. These short segments are used to regenerate a smooth mesh of a linear shape using special shape replacement commands. A linear structure is constructed by *Line* and *LineSeg* commands in the grammar. In a shape grammar, two lines have a high probability to intersect when they are growing towards each other. Since a linear structure consists of a series of short line segments (usually a cylinder or cuboid shape) in our approach, simply removing segments according to collision detection would leave a noticeable gap between two lines. We connect one line to the other at their intersection point to remove this discontinuity. Every time when we generate a new line segment a, we first find Set S of potentially intersecting line segments. S is constructed so that for every line segment b in it, the distance of central points of a and b must be less than a threshold r. Next, for each such a and b, we test whether they intersect each other. In case of one or multiple intersections we clip the new line to end at the first intersection. Whether a line is merged or not can be specified as a parameter in the *Line* command. Once a line is merged, it will not be deleted as in collision detection, but the derivation of this line command will be stopped here. An example of shape merging is illustrated in Fig. 10.

5 Applications

We demonstrate our Field-Guided Shape Grammars with three applications: tensor field visualization, curvilinear mosaic tiling, and geometry synthesis on surfaces. The fields used in these examples were created with three to ten tensor design elements and several tensor-valued smoothing operations [52].

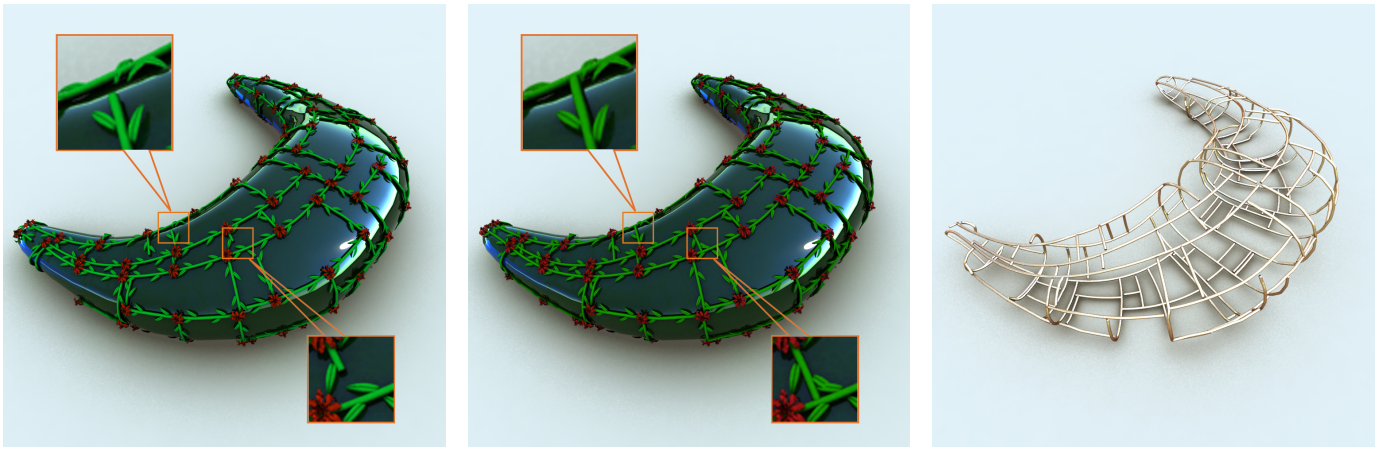


Fig. 10. This figure shows shape merging of line shapes. The left figure shows the distribution without merging of lines. The middle figure shows how the gaps are filled after lines are merged at intersections. The right images shows the graph structure of the pattern.

5.1 Tensor Field Visualization

Visualizing second-order tensor fields remains a major challenge for the visualization community. One of the main reasons has been the multivariate nature of second-order tensor fields. Even in the symmetric case, a tensor contains two scalar values (major and minor eigenvalues) and two directions (major and minor eigenvectors). A typical approach is to use hyperstreamlines following the major and minor eigenvector fields to illustrate them. Then color is used to show one of two eigenvalues or the total tensor magnitude. Glyphs such as ellipsoids have been used as an alternative which can show both the eigenvalues and eigenvectors. However, such glyphs cannot be used to show tensors with negative eigenvalues such as the curvature tensor.

We address this by using different types of shapes to represent different scenarios. When both eigenvalues are non-negative, we use ellipses as in existing methods. If one of the eigenvalues is negative, we will add a separating line segment that passes through the center of the ellipse and is parallel to the corresponding eigenvector. If both eigenvalues are negative, then two separating line segments are added which form a cross that divide the ellipse evenly along both eigenvector directions. See Fig. 11 for an example (top: ellipses only; bottom: ellipses or divided ellipses). Notice that our system can easily place different types of shapes using the same grammar.

5.2 Curvilinear Mosaic Tiling

Using mosaics tiles as an image representation is a fascinating form of art that has received attentions from the graphics community. Various techniques exist that can generate the appearance of decorative tile mosaics [53], [30]. Most of these methods are based on centriodal Voronoi diagrams when placing the tiles which are squares or rectangles whose aspect ratio is close to that of a square. In this work, we use our Field-Guided Shape Grammar to generate the curvilinear mosaics with greater aspect ratios. Such curvilinear mosaics can allow stronger emphasis on the anisotropy in the underlying image as well as allows more artistic freedom. See Fig. 12 for two examples: a cat (left) and the Mona Lisa (right). In order to generate such mosaics, a base field whose streamlines have the desired shape is designed on a plane, then mosaic tiles are distributed on this plane by the field and grammar commands.

5.3 Geometry Synthesis on Surfaces

We show several examples of patterns generated on surfaces. Instead of designing local geometric patterns by example we can design local and global patterns by specifying rules of the shape grammar. Our system is implemented using C++, and we use a computer with an Intel 2.66 Ghz CPU to produce the results. We include our examples as additional material with this submission. Table 1 shows the time and mesh size of each example result. We can see that in most cases our algorithm is able to generate complex pattern on a surface in a few seconds. In the following we give a short description of the different examples.

1) Fig. 13 shows a global pattern of branches and leaves on a surface. We show this pattern on a torus, a bunny, and an organic model.
2) Fig. 14 shows a pattern that consists of a set of small star shapes organized in two concentric circular patterns.
3) Fig. 1 and Fig. 15 left show a crack pattern that consists of linear tubes with gaps between them and smallar circular shapes next to the linear structure.
4) Fig. 15 middle shows the direct application of the planar pattern from Fig. 5 to a horse. The grammar was not modified to apply the pattern on the surface.
5) Fig. 15 right is a global pattern of a tree on a surface.
6) To study the application of a local pattern we used the example from Fig. 4 and applied it to a dragon model (Fig. 16) as well as a union of three rotated tori (Fig. 17), a genus 7 surface.

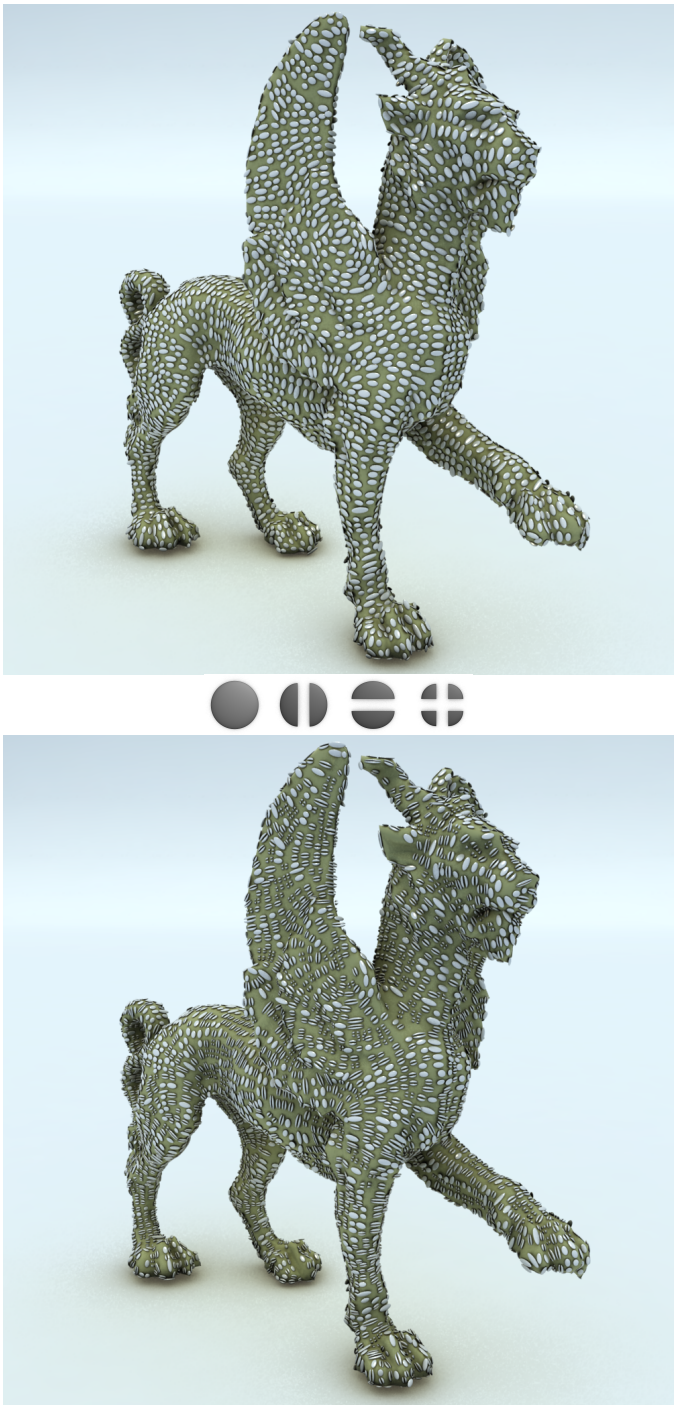


Fig. 11. Two glyph-based tensor field visualization methods applied to the curvature tensor of the feline model: (top) using ellipses only to represent the absolute values of the principle curvatures, and (bottom) negative curvatures are represented by additional shapes shown in the middle.

7) Fig. 18 shows the planar pattern from Fig. 5 applied to the happy buddha model.
8) Fig. 19 shows a global pattern that grows different layouts depending on the location on the surface. The blue tiles, red ellipsoids, and green tiles are placed with different rules.

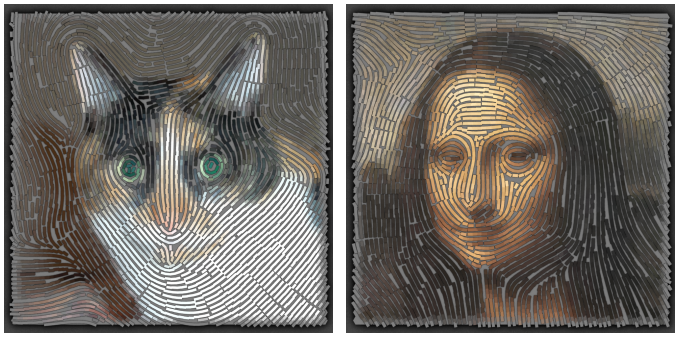


Fig. 12. A tile mosaic of a cat (left) and the Mona Lisa (right).

TABLE 1
Statistics of geometry synthesis examples

Example	# rules	input size (# of faces)	output size (# of faces)	time (in secs)
Fig. 13 Left	10	9,600	568K	0.5
Fig. 13 Middle	10	12,000	352K	1
Fig. 13 Right	10	1,280	750K	3
Fig. 14 Left	9	1,280	739K	9
Fig. 14 Right	9	21,504	723K	17
Fig. 1	7	12,000	521K	182
Fig. 15 Left	7	1,280	265K	15
Fig. 15 Middle	14	12,000	867K	1
Fig. 15 Right	13	2,748	483K	13
Fig. 16 Left	12	23,242	1.2M	10
Fig. 16 Middle	12	40,000	965K	42
Fig. 17	12	40,000	4M	12
Fig. 18	30	20,000	662K	1
Fig. 19	13	8,704	134K	3

The images in the paper are rendered with *V-Ray*.

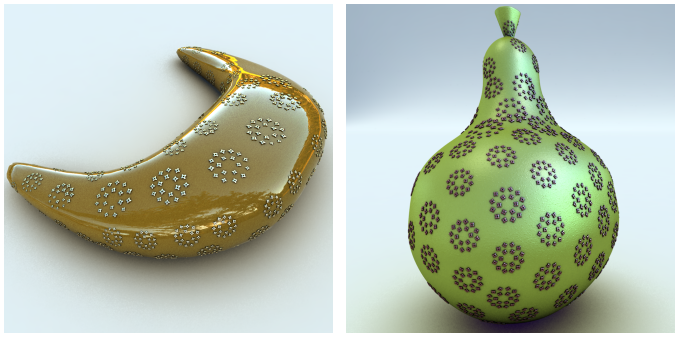


Fig. 14. A grammar applied to an organic architectural model (left) and a pear (right).

6 Discussion

Comparison to previous work: Through the applications above we can see the difference between FGSG and previous parameterization-based methods [1], [54]. Parameterization is well suited for cases where the distribution of shapes in the pattern is simple repetition, and the distortion of individual shapes is not very important. Our approach has advantages where the distribution has a more complex, i.e. global, structure, and does not allow distortion of individual shapes, which is difficult to handle with parameterization. Also, our approach

Fig. 13. A stem and leaf pattern growing on three different surfaces.

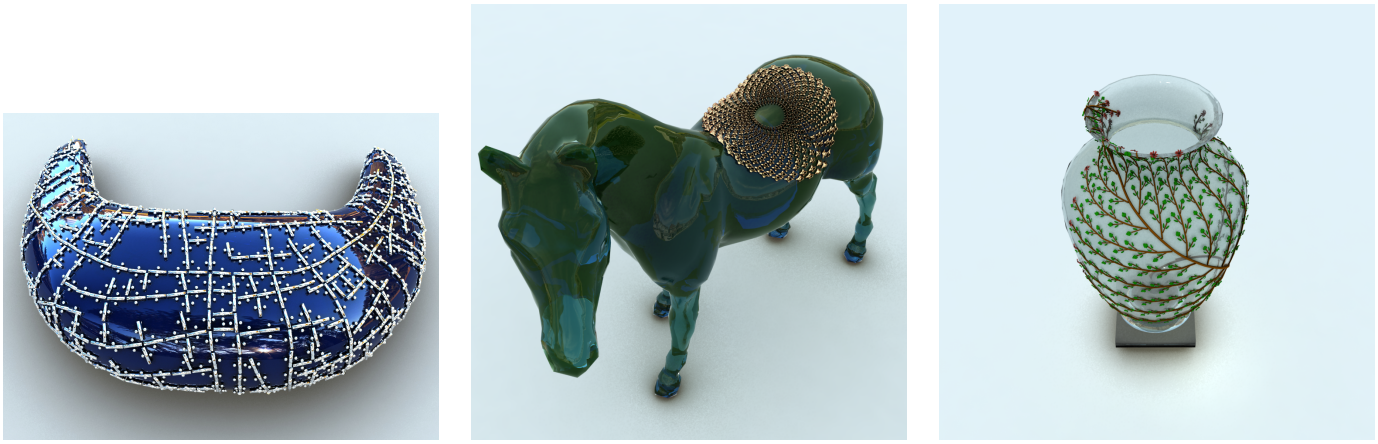


Fig. 15. Left: the pattern shown in Fig. 1 applied to an organic shape. Middle: This pattern is the direct application from the grammar that generated Fig. 5. Right: The pattern from Fig. 2 is applied to the surface of a vase.

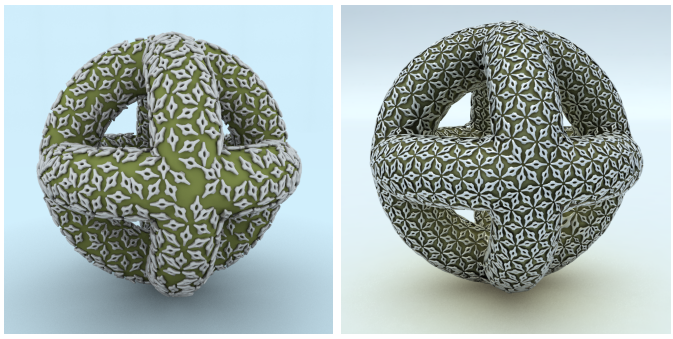


Fig. 17. This figure illustrates that porting a pattern to a surface requires careful rewriting of design rules. The left image is a simple application of the pattern in Fig. 4 to a surface using only collision detection. The right image uses a better grammar that grows linear shapes using snapping of the end points and subsequently replaces the linear shapes with more complex shapes.

can generate complex patterns from very simple shapes hierarchically while parameterization needs an example input pattern to repeat. In addition, using parameterization in shape synthesis is much slower due to the computational cost of the involved optimization algorithms. While it is also possible to use grammars in the parameter domain of a parametrized surface, distortions in the parameterization and discontinuities introduced by the seams make the grammar generation process a more challenging endeavor. In texture synthesis on surfaces, direct synthesis [39], [55] is also preferable to texture synthesis using a parameterization [54].

Limitations: A limitation of our method is that we do not currently use global optimization as a post-process. Therefore, our approach is not competitive with highly specialized pattern generation algorithms, such as surface remeshing [56]. We believe that it should be possible to integrate low level continuous optimization with a grammar driven design process. In future work we wish to develop a mixed discrete and continuous optimization where the grammar suggests discrete shape graphs that are further optimized by a global optimization algorithm, such as sequential quadratic programming (SQP) or Quasi-Newton. A second limitation is that several grammars produce artifacts when the placement of shapes is too dense, such as the Dragon example in Fig. 16. To make this example work we had to implement two additional rules that allow the grammar querying the angles to nearby shapes. It still took up to ten tries to get reasonable results. We therefore believe that our algorithm is mainly suitable for patterns that are not too dense, such as plants growing on surfaces. A third limitation is that

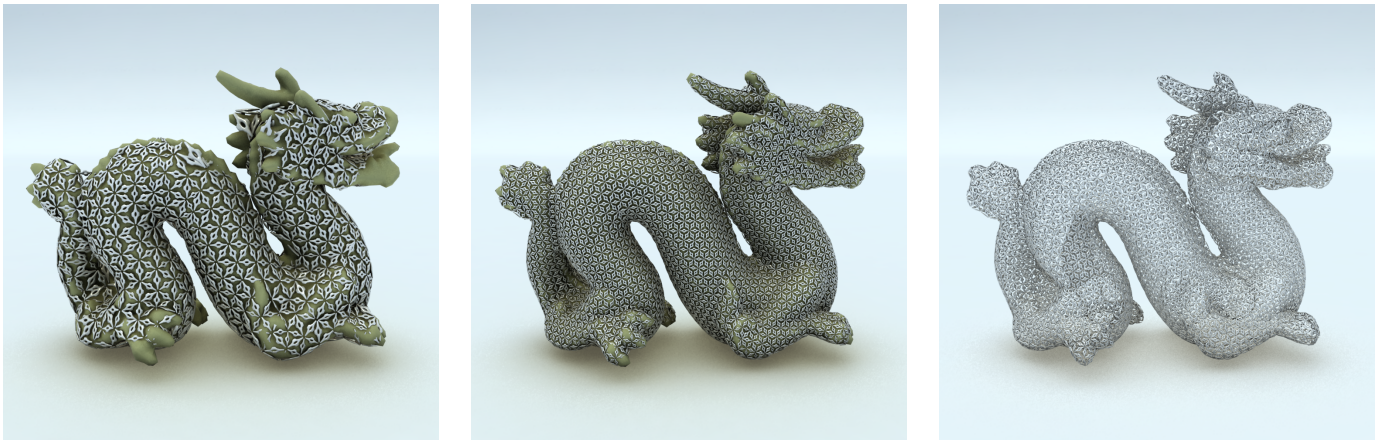


Fig. 16. The pattern from Fig. 4 adapted to grow on surfaces. Left: a pattern with big shapes. Middle: a pattern with smaller shapes. Right: A rendering of the shapes without the surface. While the shape placement algorithm can place shapes under the surface resulting in occluded shapes, this rendering shows that the surface is still fully covered.

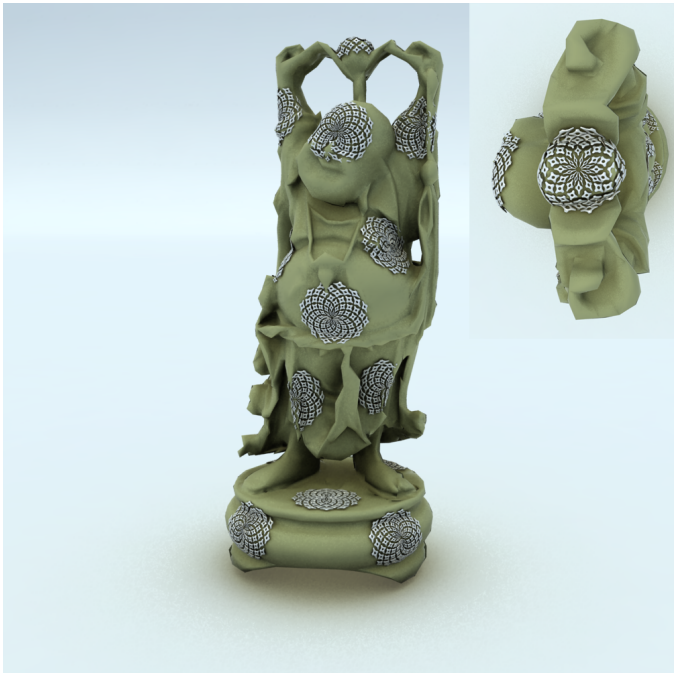


Fig. 18. The application of the pattern from Fig. 5 to the happy buddha model. The inset shows a view from the top down.

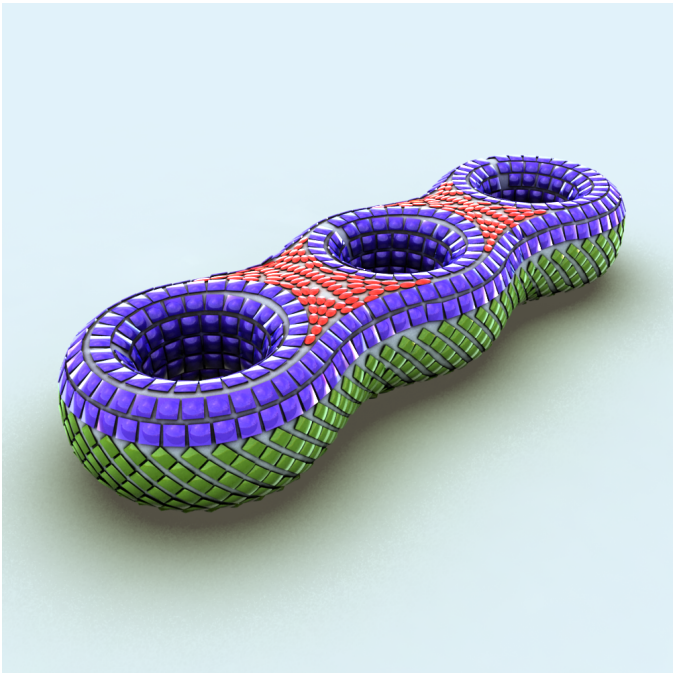


Fig. 19. A pattern that uses three different growing strategies based on the location on the surface. Blue tiles and red ellipses on top and green tiles on the bottom.

a grammar has to be designed manually using a specification in a text file. As shown in figure 17 not all strategies are successful and several design iterations are required to achieve good results. This is similar to modeling using scripting in major modeling programs like Maya and requires a skill set that not all designers or modelers possess. Therefore, we also want to explore interactive interfaces to model global patterns on surfaces as part of our future work.

7 Conclusion

In this paper we introduced Field-Guided Shape Grammars. These grammars allow us to encode a large class of pattern designs and apply them to fields in the plane and fields on surfaces. Instead of distorting a pattern design by surface parametrization, we place designs directly on the surface and adapt the pattern to the surface directly through the rules of the grammar.

As future work we plan to explore more applications such as non-photorealistic rendering and vector and tensor field visualization. In addition, subtracting geometry from a 3D surface based on a field is an interesting problem that we wish to explore.

ACKNOWLEDGEMENTS

The authors would like to thank Christopher Grasso for rendering images in this paper and thank the reviewers for their help and suggestions. We thank Marc Levoy and the Stanford's graphics Lab, Bruce Teeter, and the AIM@SHAPE Shape Repository for the models used in this work. The cat image is a courtesy of Greg Turk.This work was supported by the US National Science Foundation (NSF), contracts IIS 0757623, IIS 0915990, IIS 0917308, and CCF 0546881.

REFERENCES

[1] K. Zhou, X. Huang, X. Wang, Y. Tong, M. Desbrun, B. Guo, and H. Shum, "Mesh quilting for geometric texture synthesis," *Proceedings of ACM SIGGRAPH 2006*, vol. 25, no. 3, pp. 690–697, 2006.

[2] G. Turk and D. Banks, "Image-guided streamline placement," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM New York, NY, USA, 1996, pp. 453–460.

[3] P. Prusinkiewicz and A. Lindenmayer, *The Algorithmic Beauty of Plants*. Springer Verlag, 1991.

[4] Y. I. H. Parish and P. Müller, "Procedural modeling of cities," in *Proceedings of ACM SIGGRAPH 2001*, E. Fiume, Ed. ACM Press, 2001, pp. 301–308.

[5] P. Prusinkiewicz, M. James, and R. Měch, "Synthetic topiary," in *Proceedings of ACM SIGGRAPH 94*, A. Glassner, Ed. ACM Press, Jul. 1994, pp. 351–358.

[6] R. Měch and P. Prusinkiewicz, "Visual models of plants interacting with their environment," in *Proceedings of ACM SIGGRAPH 96*, H. Rushmeier, Ed. ACM Press, Aug. 1996, pp. 397–410.

[7] P. Prusinkiewicz, P. Mündermann, R. Karwowski, and B. Lane, "The use of positional information in the modeling of plants," in *Proceedings of ACM SIGGRAPH 2001*, E. Fiume, Ed. ACM Press, 2001, pp. 289–300.

[8] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural Modeling of Buildings," in *Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics*, 2006.

[9] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky, "Instant architecture," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 669–677, 2003.

[10] M. Lipp, P. Wonka, and M. Wimmer, "Interactive visual editing of grammars for procedural architecture," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 102:1–10, Aug. 2008, article No. 102. [Online]. Available: http://www.cg.tuwien.ac.at/research/publications/2008/LIPP-2008-IEV/

[11] S. Havemann, "Generative mesh modeling," PhD Thesis, TU Braunschweig, 2005.

[12] G. Turk, "Texture synthesis on surfaces," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 2001, pp. 347–354.

[13] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," in *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1998, pp. 453–460.

[14] A. Hertzmann and D. Zorin, "Illustrating smooth surfaces," in *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 517–526.

[15] S. Paris, W. Chang, O. I. Kozhushnyan, W. Jarosz, W. Matusik, M. Zwicker, and F. Durand, "Hair photobooth: geometric and photometric acquisition of real hairstyles," in *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*. New York, NY, USA: ACM, 2008, pp. 1–9.

[16] E. Zhang, J. Hays, and G. Turk, "Interactive tensor field design and visualization on surfaces," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 1, pp. 94–107, 2007.

[17] J. Palacios and E. Zhang, "Rotational symmetry field design on surfaces," *ACM Trans. Graph. (SIGGRAPH 2007)*, vol. 26, no. 3, p. 55, 2007.

[18] G. Chen, G. Esch, P. Wonka, P. Müller, and E. Zhang, "Interactive procedural street modeling," *ACM Trans. Graph.*, vol. 27, no. 3, p. 103, 2008.

[19] M. Marinov and L. Kobbelt, "Direct anisotropic quad-dominant remeshing," in *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 207–216.

[20] N. Ray, W. C. Li, B. Lévy, A. Sheffer, and P. Alliez, "Periodic global parameterization," *ACM Trans. Graph.*, vol. 25, no. 4, pp. 1460–1485, 2006.

[21] E. Zhang, K. Mischaikow, and G. Turk, "Vector field design on surfaces," *ACM Trans. Graph.*, vol. 25, no. 4, pp. 1294–1326, 2006.

[22] G. Chen, K. Mischaikow, R. S. Laramee, P. Pilarczyk, and E. Zhang, "Vector field editing and periodic orbit extraction using morse decomposition," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 4, pp. 769–785, 2007.

[23] M. Fisher, P. Schröder, M. Desbrun, and H. Hoppe, "Design of tangent vector fields," in *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*. New York, NY, USA: ACM, 2007, p. 56.

[24] N. Ray, B. Vallet, W. C. Li, and B. Lévy, "N-symmetry direction field design," *ACM Trans. Graph.*, vol. 27, no. 2, p. 10, 2008.

[25] A. Runions, M. Fuhrer, B. Lane, P. Federl, A.-G. Rolland-Lagan, and P. Prusinkiewicz, "Modeling and visualization of leaf venation patterns," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 702–711, 2005.

[26] D. Dunbar and G. Humphreys, "A spatial data structure for fast poisson-disk sample generation," *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, vol. 25, no. 3, pp. 503–508, 2006.

[27] G. Turk, "Generating textures on arbitrary surfaces using reaction-diffusion," in *Proceedings of ACM SIGGRAPH 91*. ACM Press, 1991, pp. 289–298.

[28] J. Kopf, D. Cohen-Or, O. Deussen, and D. Lischinski, "Recursive wang tiles for real-time blue noise," *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, vol. 25, no. 3, pp. 509–518, 2006.

[29] V. Ostromoukhov, "Sampling with polyominoes," *ACM Trans. Graph*, vol. 26, no. 3, p. 78, 2007.

[30] A. Hausner, "Simulating decorative mosaics," in *SIGGRAPH Proceedings*, 2001, pp. 573–580.

[31] J. Kim and F. Pellacini, "Jigsaw image mosaics," in *SIGGRAPH 2002 Conference Proceedings*, ser. Annual Conference Series, J. Hughes, Ed. ACM Press/ACM SIGGRAPH, 2002, pp. 657–664.

[32] T. Ijiri, R. Mech, T. Igarashi, and G. Miller, "An example-based procedural system for element arrangement," *Comput. Graph. Forum*, vol. 27, no. 2, pp. 429–436, 2008.

[33] H. Pottmann, A. Schiftner, P. Bo, H. Schmiedhofer, W. Wang, N. Baldassini, and J. Wallner, "Freeform surfaces from single curved panels," *ACM Transactions on Graphics*, vol. 27, no. 3, pp. 76:1–76:10, Aug. 2008.

[34] H. Pottmann, Y. Liu, J. Wallner, A. Bobenko, and W. Wang, "Geometry of multi-layer freeform structures for architecture," *ACM Transactions on Graphics*, vol. 26, no. 3, pp. 65:1–65:11, Jul. 2007.

[35] Y. Liu, H. Pottmann, J. Wallner, Y.-L. Yang, and W. Wang, "Geometric modeling with conical meshes and developable surfaces," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 681–689, Jul. 2006.

[36] J. Legakis, J. Dorsey, and S. J. Gortler, "Feature-based cellular texturing for architectural models," in *Proceedings of ACM SIGGRAPH 2001*, E. Fiume, Ed. ACM Press, 2001, pp. 309–316.

[37] A. A. Efros and T. K. Leung, "Texture synthesis by non-parametric sampling," in *IEEE International Conference on Computer Vision*, Corfu, Greece, September 1999, pp. 1033–1038.

[38] L. Wei and M. Levoy, "Fast texture synthesis using tree-structured vector quantization," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 2000, pp. 479–488.

[39] G. Turk, "Texture synthesis on surfaces," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, pp. 347–354, 2001.

[40] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, and H. Shum, "Synthesis of bidirectional texture functions on arbitrary surfaces," *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 665–672, 2002.

[41] K. Fleischer, D. Laidlaw, B. Currin, and A. Barr, "Cellular texture generation," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM New York, NY, USA, 1995, pp. 239–248.

[42] A. Efros and W. Freeman, "Image quilting for texture synthesis and transfer," in *Proceedings of SIGGRAPH 2001*. Los Angeles, CA, 2001, pp. 341–346.

[43] L. Liang, C. Liu, Y. Xu, B. Guo, and H. Shum, "Real-time texture synthesis by patch-based sampling," *ACM Transactions on Graphics*, vol. 20, no. 3, pp. 127–150, 2001.

[44] V. Kwatra, A. Schodl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: Image and video synthesis using graph cuts," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 277–286, 2003.

[45] P. Bhat, S. Ingram, and G. Turk, "Geometric texture synthesis by example," in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM New York, NY, USA, 2004, pp. 41–44.

[46] P. Müller, "Procedural modeling of cities," in *SIGGRAPH '06: ACM SIGGRAPH 2006 Courses*. NY, USA: ACM Press, 2006, pp. 139–184.

[47] N. Chomsky, "Three models for the description of language," *Information Theory, IRE Transactions on*, vol. 2, no. 3, pp. 113–124, 1956.

[48] B. O'Neill, *Elementary differential geometry*. Academic Pr, 1997.

[49] R. Kimmel and J. Sethian, "Computing geodesic paths on manifolds," *Proceedings of National Academy of Sciences, USA, 95(15): 8431-8435.*, 1998. [Online]. Available: citeseer.ist.psu.edu/article/kimmel98computing.html

[50] H. Theisel, "Vector field curvature and applications," *Doktorarbeit, FB Informatik, Universit at Rostock*, 1995.

[51] "Bullet physics library," http://bullet.sourceforge.net, accessed May 11, 2009.

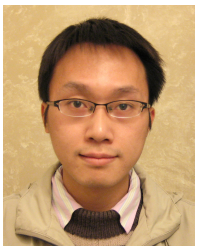
[52] P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun, "Anisotropic polygonal remeshing," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 485–493, 2003.

[53] P. Haeberli, "Paint by numbers: abstract image representations," in *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1990, pp. 207–214.

[54] E. Praun, A. Finkelstein, and H. Hoppe, "Lapped textures," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2000)*, pp. 465–470, Aug. 2000.

[55] L. Y. Wei and M. Levoy, "Texture synthesis over arbitrary manifold surfaces," *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, pp. 355–360, 2001.

[56] L. Wang, S. You, and U. Neumann, "Large-scale urban modeling by combining ground level panoramic and aerial imagery," in *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, 2006, pp. 806–813.



Yuanyuan Li received his BS in Computer Science from Zhejiang University, China in 2002 and is currently an MS Candidate at Arizona State University, Tempe. He currently works at the Partnership for Research in Spatial Modeling (PRISM) lab. His research interests include procedural modeling, computational geometry and image-based real-time rendering.

Fan Bao received his BS and MS degree in Computer Science from Tsinghua University, Beijing, China in 2005 and 2008, respectively. He is currently working toward a Ph.D. degree at Arizona State University, Tempe. His research interests include computer graphics, procedural modeling, and visualization.

Eugene Zhang received the PhD degree in computer science from Georgia Institute of Technology in 2004. He is currently an assistant professor at Oregon State University, where he is a member of the School of Electrical Engineering and Computer Science. He received an NSF CAREER Award in 2006. His research interests include computer graphics, scientific visualization, and geometric modeling. He is a member of the IEEE Computer Society and the ACM.

Yoshihiro Kobayashi received the PhD degree from University of California, Los Angeles in 2001. He is a faulty research associate at the Partnership for Research in Spatial Modeling (Prism) lab, at Arizona State University. His research interests include design computation in architecture, procedural architectural/urban modeling, visualization and simulation in virtual reality urban environments, and design information management.

Peter Wonka received the MS degree in urban planning and the doctorate in computer science from the Technical University of Vienna. He is currently with Arizona State University (ASU). Prior to coming to ASU, he was a postdoctorate researcher at the Georgia Institute of Technology for two years. His research interests include various topics in computer graphics, visualization, and image processing. He is a member of the IEEE.