Procedural Mesh Features applied to Subdivision Surfaces using Graph Grammars

W. Thaller[1 1], U. H. Augsdörfer[2 1] and D. W. Fellner[3 1,2]
[1]Institut für ComputerGraphik & Wissensvisualisierung, TU Graz, Austria
[2]TU Darmstadt & Fraunhofer IGD, Germany

Abstract

A typical industrial design modeling scenario involves defining the overall shape of a product followed by adding detail features. Procedural features are well-established in computer aided design (CAD) involving regular forms, but are less applicable to free-form modeling involving subdivision surfaces. Current approaches do not generate sparse subdivision control meshes as output, which is why free-form features are manually modeled into subdivision control meshes by domain experts. Domain experts change the local topology of the subdivision control mesh to incorporate features into the surface, without increasing the mesh density unnecessarily and carefully avoiding the appearance of artefacts.

In this paper we show how to translate this expert knowledge to grammar rules. The rules may then be invoked in an interactive system to automatically apply features to subdivision surfaces.

Keywords: Procedural features; feature-based modeling; free-form surface features; graph grammars; Catmull-Clark subdivision surfaces; automating changes in mesh topology

1. Introduction

The design of products that have to fulfil engineering requirements as well as aesthetic criteria, typically involves free-form shapes. The styling activity of aesthetic product design comprises two steps [1]: First the product's overall shape is defined. This is followed by local refinements where features are added to the overall shape. Often, the same or similar features are repeatedly applied to one design.

A procedural feature consists of a set of parameters along with an algorithm for applying the feature to an underlying model. The designer can then manipulate the feature directly on a high semantic level of abstraction. The underlying shape can be edited while leaving the procedural features in place. Also, the designer is able to control specific aspects of a feature shape, while the overall shape remains fixed. This is referred to as *feature-based modeling*.

For feature-based modeling to work, features must be well-defined in terms of their parameters. Therefore, feature-based modeling was first introduced in the context of solid modeling, where procedural features have become firmly established. A cylindrical hole drilled into an overall shape is the classic example of a procedural feature. The use of free-form surface features, rather than regular-shaped features, is an active area of research [2].

Because feature-based modeling systems need to be compatible with a larger *modeling pipeline*, it is desirable for the system's input and output to be a standard CAD representation.

Non-uniform rational B-splines (NURBS), a patch-based surface representation, are the current standard for free-form modeling in CAD. Adding features to free-form surfaces represented by NURBS frequently involves increasing the resolution of surface patches, which leads to many redundant vertices in the representation.

Subdivision surfaces, already established in the animation industry, have recently gained popularity as an alternative to NURBS in CAD. Subdivision methods are a generalisation of traditional spline patch methods to arbitrary topology; for example, Catmull-Clark [3] generalises bi-cubic patches.

Because subdivision surfaces may also include extraordinary vertices, that is vertices with a valency either more or less than the regular valence, features may be introduced into the surface by locally changing the topology of the control mesh.

However, changes to the topology of the control mesh may give rise to artefacts in the limit surface. To apply free-form design features to a subdivision surface, a CAD expert meticulously adjusts the resolution and the mesh topology of the subdivision control mesh in order to keep changes to the overall shape small and to avoid or hide the visual appearance of artefacts [4]. Once a good topology change is identified, the expert applies the same procedure each time the same or a similar feature has to be applied.

In this paper we describe how this expert knowledge can be formulated as graph grammar rules. We propose to use a graph grammar on top of a scripting language; the basic local modifications are scripted, and the graph grammar allows us to organize the problem.

2. Related Work

Grammars have proved very useful for the procedural modeling of buildings where contex-free split grammars have been used [5, 6]. There, too, a grammar provides the organizing

paradigm for something that could theoretically also be hard-coded.

Procedural free-form features have been a lively area of research since the 1990s. For a comprehensive introduction to and overview of feature-based modeling in CAD, we refer the reader to Pernot et al. [2].

2.1. B-spline based methods

In [7], Pernot et al. introduce a method for using three-dimensional curves for defining deformations applied to B-spline surfaces. This method was later extended [8, 9, 10] as a method for defining a range of freeform surface features on freeform surfaces. Application of the features is done using a force-based deformation system. The method can operate on NURBS data, but no new control points are added. The input geometry must have sufficient resolution to represent the features.

Chen et al. [11] solve this limitation by using *hierarchical NURBS* [12] to represent the resulting surface. In this approach, higher-resolution patches are used to refine the lower-resolution input patches and add high-frequency detail. The resulting geometry is tied to the use of non-standard hierarchical NURBS as a geometry representation and can therefore not be used in existing modeling pipelines.

Another way to avoid the problem of insufficient resolution in the input control mesh is to generate the feature during rendering instead of attempting to output a control mesh. Displacement mapping [13] is a wide-spread and efficient method for applying high-frequency detail to surfaces when the output is only used for visualization.

2.2. Subdivision based methods

A popular method for adding a limited repertoire of features to subdivision surfaces has been defined by De Rose et al. [14]. In their approach, edges of a subdivision control mesh may be marked as being exempt from the subdivision process. This introduces sharp and semi-sharp creases into an otherwise smooth limit surface. However, the output surface is not a generalisation of traditional spline patch methods, which typically is a requirement further down the product design pipeline.

Khodakovsky and Schröder [15] describe an algorithm allowing the creation and manipulation of fine scale feature curves on subdivision surfaces. Creation of the features happens during the subdivision process, so there is no sparse control net for the resulting surface.

In the context of Sketch based modeling, Olsen et al. [16] have developed a method for incorporating linear features into subdivision control nets by locally increasing the mesh resolution. The transition between the higher-resolution patched faces and the lower-resolution surroundings is handled by a set of fixed patching template. However, this approach may increase control mesh density more than necessary.

In product design today, topolgical changes to a subdivision control net to incorporate high frequency features are still modeled manually by the CAD expert.

3. Adding Features to Subdivision Surfaces

The problem of applying free-form features to a subdivision surface can be seen as consisting of three subproblems, namely 1) defining the feature, 2) changing the topology of the subdivision control mesh, and 3) shifting the control points appropriately.

Of these, the second step is the most time consuming and the one where expert knowledge is required.

Typically, a CAD expert with a thorough understanding of the surface representation carefully designs the subdivision control mesh in order to locally increase the resolution around the area where the feature is to be placed. This is not a trivial task:

CAD modelers typically take great care to avoid the appearance of surface artefacts.

In regular regions surface artefacts are known to arise if features are not aligned but run skew to grid lines of the control mesh [4, 17, 18]. Because of this CAD modelers take great care to align features with the underlying grid.

Locally refining a subdivision control mesh to incorporate the feature typically gives rise to irregular regions: For a subdivision scheme based on quadrilateral meshes these occur around vertices with more or less than four edges, referred to as extraordinary vertices, and non-quad faces, which give rise to extraordinary vertices after one subdivision step.

Subdivision surfaces do not guarantee C^2 continuity at extraordinary vertices [19], and undesirable artefacts are likely to appear [4, 20, 21].

Typically, when applying features to a subdivision surface, expert mesh modelers meticulously identify changes to the topology of subdivision control meshes such that the visibility of unavoidable artefacts around irregular regions is minimized.

Furthermore, when different features meet in one place on a surface, they have to interact. The interaction of features is domain-specific and cannot be determined without access to domain-specific expert knowledge.

As is clear from the above, incorporating features to a free-form design is time consuming and often requires an expert understanding of the underlying geometric representation in order to integrate features.

Once a method for good integration of a feature to a subdivision control mesh has been identified, the expert CAD designer has to use the same methods repeatedly in order to manually incorporate free-form features of similar type into subdivision surface control meshes. There are currently insufficient intelligent tools to support or automate this laborious task.

Automating some of the repetitive tasks of the feature-based subdivision modeling will accelerate the design process and frees the designer from needing extensive knowledge of the underlying geometry representation.

4. Contribution of the Paper

We demonstrate that grammar rules are suitable to translate expert knowledge of how to change the topology of a control mesh to incorporate a feature. Compared to existing methods for automatically applying features to subdivision meshes, the

rule-based approach adds significantly fewer control points to the mesh.

Changing the topology of a control mesh unavoidably introduces changes to the limit surface. To keep the alteration of the original surface to a minimum we apply an optimization step after the changes in mesh topology.

To demonstrate our approach, we have developed a prototype system to automatically change the topology of a Catmull-Clark subdivision control mesh [3] to apply various surface features with only a minor increase in resolution to achieve good visual properties.

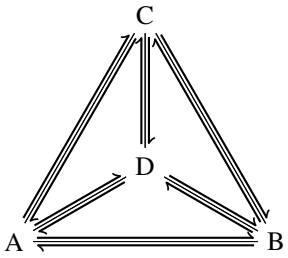

Figure 1: A halfedge mesh representing a tetrahedron. Vertices A, B, C and D are connected with pairs of halfedges in opposing directions.

5. Our Approach

A mesh data structure can be interpreted as a graph. Therefore, graph grammars can be used to define operations on meshes. We employ graph grammars to express the expert knowledge of how to integrate features into a subdivision control mesh in a way that changes its mesh topology with respect to pre-defined requirements.

The vertices and halfedges of the subdivision control mesh form the nodes of the graph. Arbitrary attributes can be added to the nodes. Relationships between these objects are represented as edges in the graph.

Features are defined in terms of rules that find the position of a given pattern in the graph and then apply some local predefined operation to the mesh at that position. These rules capture the CAD expert's knowledge of the required changes in the topology of the subdivision control mesh to integrate a feature.

The feature may then be applied using a simple point-and-click interface by non-expert users.

Our prototype system is built on top of the Generative Modeling Language (GML) [22], a scripting language with special support for the manipulation of Catmull-Clark [3] subdivision surfaces. Local modifications to the mesh are described using the scripting language; the questions of where and when the local modifications are to be applied, and of how they interact with each other, are handled by the graph grammar. While any scripting language with library support for manipulating subdivision control meshes could be used, using a GML script has two advantages: First, GML's use of Euler Operators as mesh manipulation primitives always guarantees that the invariants of the mesh data structure are preserved. Second, the description is often shortened because unchanged parts of the graph need not be repeated.

5.1. Meshes and Graphs

Using a standard halfedge datastructure, a mesh consists of vertices, halfedges and faces. Each halfedge connects exactly two vertices, its source and its target, and belongs to exactly one face. Each halfedge is related to exactly one other halfedge by a symmetric relation. These two halfedges connect the same two vertices, but in opposite directions, i.e. source and target are swapped. For each vertex, all outgoing halfedges form a cycle, and all halfedges belonging to a face form a cycle. As an example, Figure 1 shows a two-dimensional representation of the halfedge mesh of a tetrahedron. Meshes with a boundary can be handled by representing the boundary as a single, non-planar face that is marked as invisible.

Figure 2 shows a tetrahedron represented as a graph. To obtain a graph representation of a mesh, we choose to map each element of the mesh, i.e. each vertex and each halfedge, to a node in the graph, and the relationships between the elements to edges in the graph. Furthermore, grammar rules are allowed to add arbitrary additional labeled edges to the graph, and to assign arbitrary attributes to nodes (name-value pairs). Geometric information, such as the position of a vertex, is stored as an attribute for each vertex.

For the purpose of our system, we can define:

Definition 1. *A graph G is a tuple (N,E,A,L,V) where*

- *N is a set of nodes*
- *$E \subseteq N \times N \times L$ is a set of edges*
- *L is a set of labels*
- *V is a set of possible attribute values*
- *$A \in N \times L \to V$ is a function defining attributes for each node.*

The set V of attribute values corresponds to the set of all values supported by the scripting language. The set L is the set of identifiers.

Definition 1 defines a directed graph whose edges are unique for each label, i.e. given a pair of nodes $a, b \in N$, there is at most one edge from a to b for each edge label.

5.2. Graph Grammars

Graph grammars, by definition, consist of a set of rules applied to a graph [23]. Each rule matches a subgraph against the graph; this subgraph is then replaced by a different subgraph as defined in the rules. Rules are applied until no rule matches any more.

As an example let us look at the simple rule that replaces one quad with two triangles, as seen in Figure 3. In this example the rule defines what elements (nodes and edges) of the graph on the left side correspond to which elements of the graph on the right side.

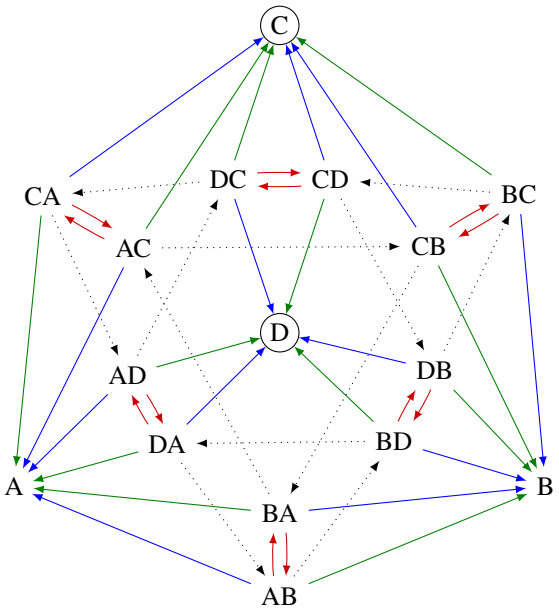

Figure 2: The graph corresponding to the mesh of a tetrahedron. The circled nodes A, B, C and D correspond to the vertices of the tetrahedron. The two-letter nodes correspond to the halfedges, i.e. node AB represents the halfedge leading from A to B, and node BA represents its "mate", going from B to A. The various graph edges represent different relationships between vertices and halfedges. Blue and green denote relations of halfedges to their source and target vertices, respectively. Red edges denote a symmetric relation between halfedges. The dotted edges represent faces for which associated halfedges form a circle. Thus, the edges $AB-BD-DA$ form a cycle corresponding to the face ABD.

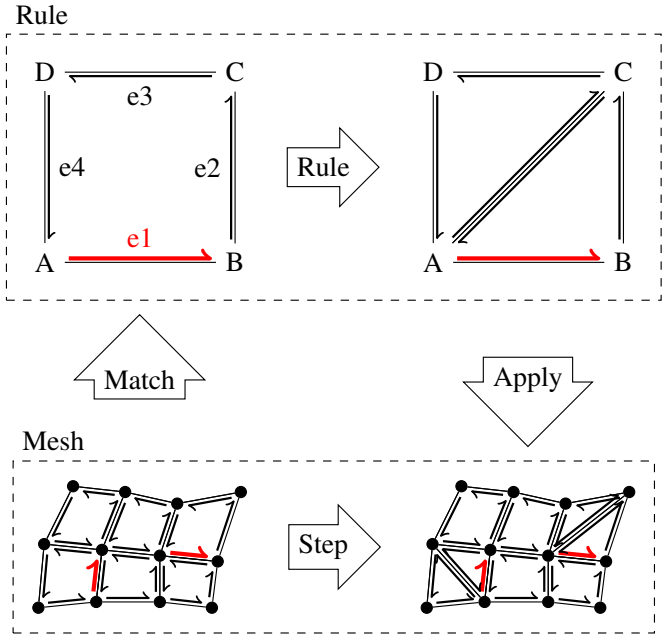

Figure 3: Example of a graph grammar rule (top) and its application to a mesh (bottom): Find a quad-face, made up of vertices A, B, C, and D connected via halfedges e1, e2, e3 and e4, where one edge (e1) is marked with the "red" attribute, as shown on the left. Split this quad-face into two triangles along the AC diagonal. The system matches the pattern to the mesh (lower left) and applies the rule everywhere it matches, yielding a new mesh (lower right).

In our system, the left sides of the rules are represented using a straightforward textual representation of the graph to be matched, and the right sides are represented as a script that describes the operation performed to change the subgraph matched by the left side. This approach is analogous to the use of operations in split grammar systems such as CGA shape [6].

Our example rule from Figure 3 can be textually represented as follows in our system:

```
rule Example {
    e1(color="red")
        faceCCW> e2 faceCCW> e3
        faceCCW> e4 faceCCW> e1
} {
    action { e1 e3 makeEF pop }
}
```

The "left side" of the rule — the code between the first pair of curly braces above — represents a graph pattern. We match against four nodes in the graph, `e1` through `e4`, which are related to each other by faceCCW edges, which correspond to the dotted edges in Figure 2. Additionally, to match our pattern, the first of these nodes must be marked by a `color="red"` attribute. The "right side" of the rule consists of GML code that defines what should happen whenever the left side of the rule is matched: `e3 e1 makeEF pop`. Because GML is a stack-based language, this means that the makeEF operation is applied to two parameters, `e1` and `e3` (which were defined while matching the left side of the rule), and its result is discarded (`pop`). The makeEF Euler operator (supplied by GML) creates a new face (F) by adding a new edge (E) between its parameters.

We can also script arbitrary additional conditions to the pattern to formulate any special cases used by the expert. In the above example, we could decide to make the rule apply to all quads, but with the orientation of the diagonal split chosen in such a way that the newly inserted diagonal is the shorter diagonal.

For more complex rules, where new vertices are inserted, the scripting language code calculates their coordinates. In practice, this is often a simple linear combination of the existing vertices.

This is not the first time grammar concepts have been applied to meshes. Spicher et al. [24] describe a system based on topological collections rather than graph grammars, intended for the declarative specification of subdivision algorithms. By contrast with our system, the right sides of the rules in [24] are defined using declarative expressions that evaluate to topological collections of cells (i.e., collections of vertices, edges and faces). While the declarative approach leads to a more mathematically beautiful formulation, the fact that cells on the right-hand side have to be explicitly generated by this expression may lead to increased verbosity.

5.3. Surface Optimization

It is important that the application of a feature keeps the alteration of the original overall shape to a minimum.

Changing the topology of a control mesh inadvertently introduces changes to the surface. After performing the topology changes, but before shifting the control points to create the feature, we therefore add an optimization step to minimize the changes to the limit surface of the overall shape. Introducing this optimization step allows many graph grammar rules to make the simplifying assumption that the surface is flat; and new control points need only be placed correctly within the flat face of the control mesh. Errors introduced by the assumption of flatness are corrected by the optimization.

We have chosen a simple optimization algorithm that yields reasonable results very quickly. Essentially, for each control point we pick a fixed direction and move the control point along that direction until its limit point falls on the desired surface.

Below, L^* is the original limit surface before any rules were applied, c_i^0 are the control points after application of the graph grammar rules, and c_i^k are the control points after iteration k of the optimization algorithm.

1. Determine the normal vectors n_i of the control mesh at each control point c_i^0.
2. Let l_i^0 be the limit points corresponding to the control points c_i^0.
3. Determine scalars s_i and points $p_i^0 = l_i^0 + s_i n_i$ such that $p_i^0 \in L^*$.
4. Let m_i be the normal vector of L^* at point p_i^0.
5. Repeat for $k = 0, 1, 2, \ldots$:
   6. Calculate the limit points l_i^k corresponding to the control points c_i^k.
   7. Determine scalars t_i^k and points $p_i^k = l_i^k + t_i^k m_i$ such that $p_i^k \in L^*$.
   8. Let $c_i^{k+1} = c_i^k + (p_i^k - l_i^k)$.
   9. Terminate when $|t_i^k| < \varepsilon$.

For any of the meshes used in this paper the optimisation required less than twenty iterations.

Minimizing the error only for the limit points of the control points, as opposed to over the entire surface between the control points, makes this optimization algorithm very cheap, but also bears the risk of causing oscillation in the optimized surface. However, the effect is strongest in the area where it is dominated by the displacement introduced by the feature itself, and thus mostly hidden from view.

6. Results

To demonstrate the approach presented in this paper we have built a prototype system to apply free-form surface features to a Catmull-Clark subdivision surface.

In our prototype interface, the location where the feature is to be applied is defined by the user by marking, via mouse click, edges in a subdivision control net. The system allows the user to either mark edges to automatically add the feature at the centre of the edges or the user may choose specific locations along an edge in the subdivision control mesh which the feature is meant to cross. These markings are represented as attributes according to Definition 1. The designer can pre-select different types of features, which leads to different attributes being set. The feature is then incorporated to the subdivision control mesh by applying pre-defined rules at locations in the control mesh where these attributes are found. Rules are applied until no further rules match.

In this section, we will demonstrate how several standard types of surface features can be created from very simple rules. Figure 4 shows a set of diagrams that describe a simple rule set for creating linear surface features. For brevity's sake, we only show the graphical representation of the rules, and we omit trivial variations of the depicted rules.

Figure 5 shows the application of a groove-feature on a subdivision surface. It is modeled by choosing the type of feature to be applied ("groove") and then marking edges in the control mesh via mouse click with the attribute "red". The system then automatically inserts three new vertices along each marked edge in the control mesh according to Rule #1 shown in Figure 4. The topology changes are the same, independent of the parameters of the feature, i.e. the width or depth of the groove.

When the marked edges are on opposite sides of a face in the control mesh, as in Figure 5, the feature is applied like a simple translational sweep. This involves #2 from Figure 4. Using the graph grammar approach, an arbitrary number of different scenarios of feature application may be formalised, handling various situations, such as corners (Rule #3), crossings (Rule #4) and three-way junctions (Rules #5 and #6). T-junctions in subdivision surfaces have to be handled very carefully. The inserted control mesh lines cannot simply end at a T-junction, as this would lead to irregular faces that markedly degrade surface quality due to the appearance of artefacts. The patterns seen in Rules #5 and #6 are often used to minimize these problems. Observe how the right sides of these rules contain only quadrilateral faces. Rule #6 also moves the geometric position of the vertex originally at the top right of the pattern to be in line with the feature that ends in the T-junction. Just like with the new vertices added by the rules, the scripting language code can calculate the coordinates using the simplifying assumption that the surface is flat; the placement with respect to the third dimension is determined by optimization (section 5.3).

Figures 6(b) to 6(e) show the results of applying features to the Catmull-Clark subdivision surface shown in Figure 6(a). The three columns show on the left the control mesh modified to incorporate the feature, in the centre the limit Catmull-Clark surface with the feature and on the right the limit surface colourised to highlight differences between the new and the original limit surface. By extending the rule database with more rules, different feature types can be applied, such as a groove feature (Figure 6(b)), a ridge feature (Figure 6(c)) or the two-dimensional "dent" feature shown in Figure 6(d). Surfaces 6(b) to 6(d) are applied by choosing the type of feature followed by marking the control mesh as shown in Figure 6(a). More elaborate markings may create features with intersections and T-junctions as in Figure 6(e).

Figure 7 illustrates the effect of the optimization step described in section 5.3 applied to the feature applied in Figure 6(e). Without the optimisation step (left), the error introduced

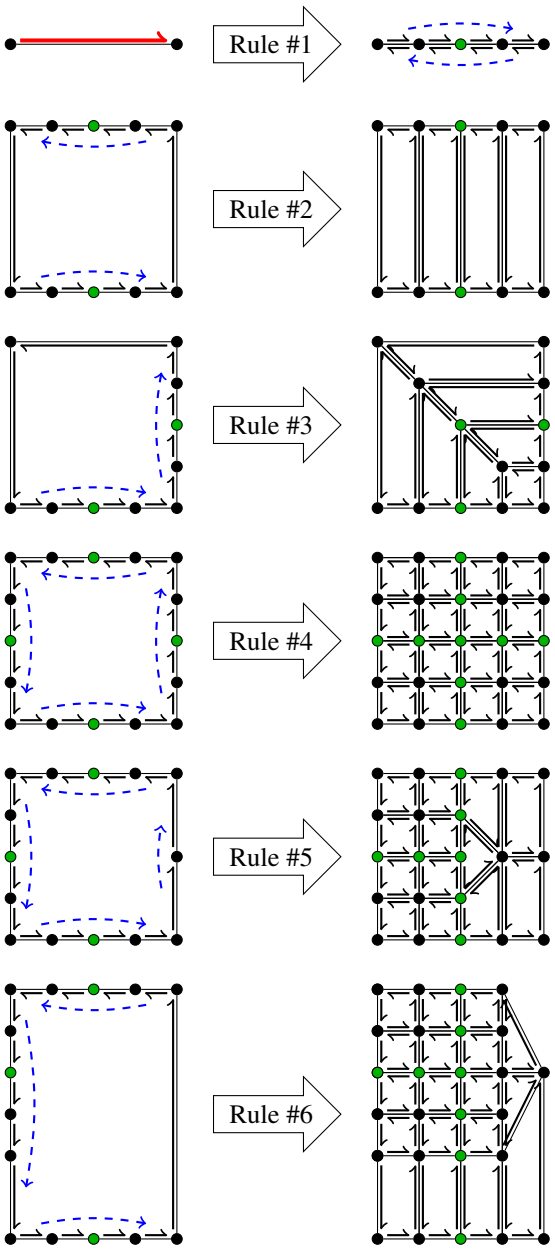

Figure 4: A rule set for generating simple seams, grooves or ridges as surface features. Initially, all mesh edges to be crossed by the seam are marked in red by the user. Rule #1 splits those edges into smaller edges by adding three vertices along each marked edge. The remaining rules connect the newly created points in different situations: straight feature across a quad (Rule #2), corners (Rule #3), crossings (Rule #4), tee-crossings (Rule #5 and #6). Two extra graph edges, shown as blue dashed arrows, are added by Rule #1 to communicate to the later rules which mesh edges belong together. When none of the rules apply anymore, the actual feature is created by shifting all vertices marked in green along the surface normal vector towards the interior of the surface for simple seams and grooves, or away from the surface for ridges.

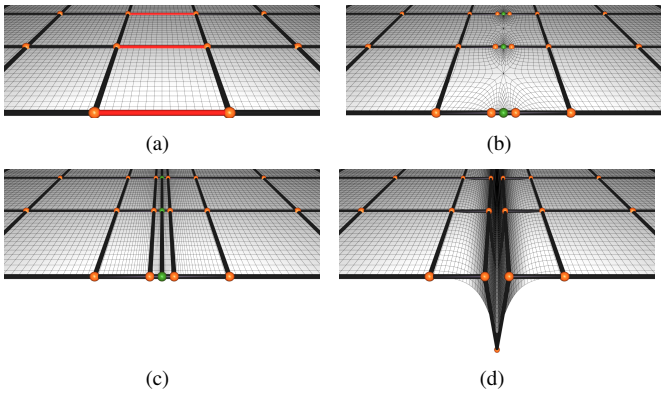

Figure 5: Topology changes to incorporate a groove-feature into a subdivision surface by applying graph grammar rules shown in Figure 4: The designer marks the affected edges in the control mesh (a). The system automatically applies Rule #1 to each marked edge: Marked edges are split by inserting additional control points (b). The resulting control mesh (b) contains several 10-gons that match the left side of Rule #2. Rule #2 is applied, that is new points are connected across the faces (c). Finally, the centre vertices of the seams are shifted along the negative direction of their normal vectors, resulting in the final control mesh (d), the limit surface of which includes the groove feature.

by applying the feature is more pronounced, especially in areas where the underlying surface has a higher curvature. With the optimisation, the error is not eliminated but significantly reduced (right).

An experienced designer will align features with grid lines. If a feature does not follow the existing control lines, artefacts will appear [4, 17, 18]. Typically, CAD experts will avoid this situation in practice by planning ahead when designing the overall shape of a model.

Our method is, however, not restricted to aligning features with grid lines. It can be used to also formulate changes to the topology to also apply features skew to the grid lines. In Figure 8 we used the system presented in this paper to apply features not aligned with the control mesh. The three columns in the figure show again the modified control mesh (left), the limit surface (centre) and the error of the new surface to the original saddle shape without the feature (right). In Figure 8(a) a free-shape feature intersects a feature aligned with the grid lines. In Figure 8(b) the boundaries of the dent feature follow an arbitrary curve. As long as the feature to be added keeps clear of existing control points, the existing rules for straight (Rule #2) and corner (Rule #3) features are sufficient. For the case where the feature comes too close to an existing vertex, we can add special-case rules, Rules #7 and #8 (Figure 9), that shift the vertex out of the way. This introduces additional error into the surface, but it allows the feature to be realised without adding many new vertices and without non-local changes to the control mesh.

It is worth noting that the graph grammar concept is capable of expressing different approaches to the problem of adding features to the mesh; for example, patch patterns for adaptive subdivision described in [16] map directly to grammar rules (Figure 10). These rules serve as a general mechanism for locally increasing the resolution of a mesh in order to add high-frequency detail to an otherwise sparse control mesh. By com-

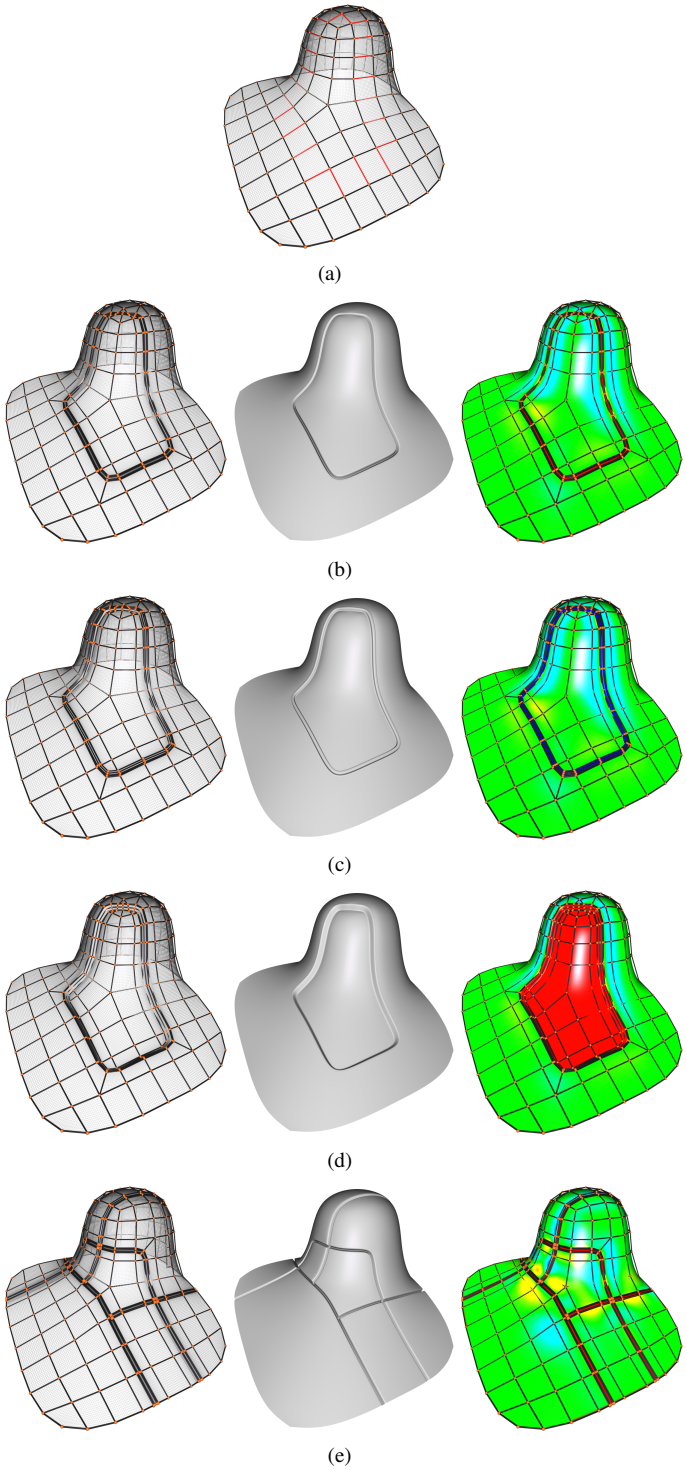

Figure 6: After marking the affected edges on the original mesh (a), various features can be automatically applied: grooves (b) and ridges (c), as well as two-dimensional features like an indentation (d). Given different markings, features may also interact with each other or form T-junctions and crossings as in (e).

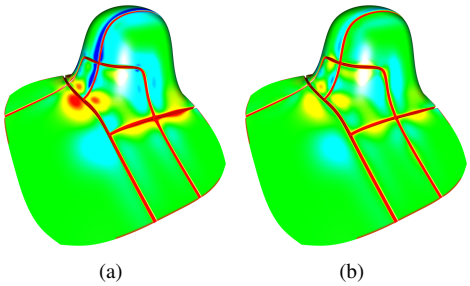

Figure 7: Effect of the optimization step described in Section 5.3. Without optimization (a), the error is much more pronounced than with optimization (b).

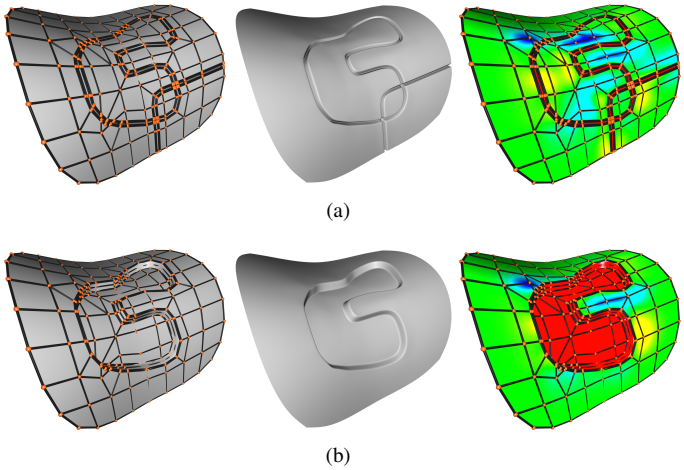

Figure 8: Non-grid-aligned features, created using two additional rules (Figure 9) that move aside the control points of the underlying surface where they would otherwise interfere with the features.

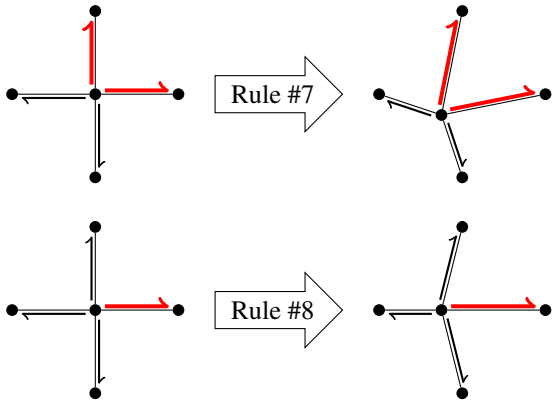

Figure 9: Two rules for enabling non-grid-aligned features by moving aside control points that are in the way. As before, a half-edge marked red indicates that a feature will cross this edge; the exact place where the feature should cross is stored as a further attribute. These rules are conditional — they are only applied if the feature would otherwise be too close to the vertex in the center of the pattern. The amount by which the central vertex is shifted is likewise determined by script code.

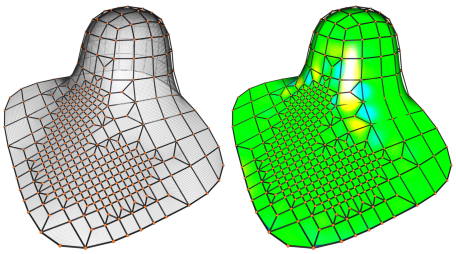

Figure 10: Adaptive control mesh refinement using the patching patterns from [16] can be directly implemented using graph grammar rules to serve as a general mechanism for adding detail. Compared to specialized rules, this usually leads to a greater increase in resolution.

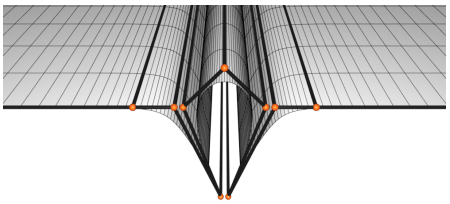

Figure 11: Complex features may be defined using graph grammars: A piped seam is a seam where an extra piece of material, the *piping*, is sewn into the seam. For our purposes, we can model this by using a variant of Rule #1 that applies a different profile to the edges.

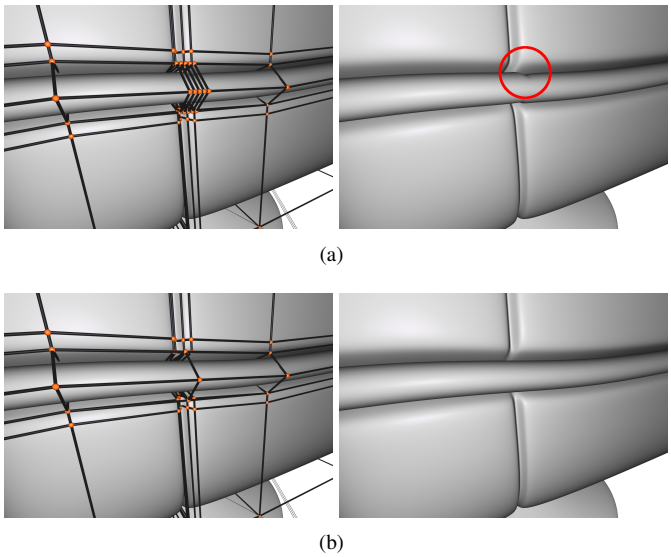

Figure 12: The crossing between two features (a regular seam and a piped seam) may lead to artefacts if standard crossing rules (Rule #4, Figure 4) are applied (a). To avoid artefacts special rule may be employed to handle these situations. In this example, visually more pleasing results are achieved if the regular seam ends where it meets the piped seam and starts again on the other side (b).

parison, a special-purpose ruleset as described in this paper uses far fewer control points to model a feature.

7. Example Application

Let us now consider the real-world application of a system which supports a non-expert designer to incorporate different types of seam features to the upholstery of chairs modeled using Catmull-Clark subdivision surfaces (Figure 13).

The most important attribute for defining a seam feature is its profile, where the user may adjust parameters such as width and depth of the seam.

We demonstrate the ability of our system to deal with interactions between different features by using different profiles to create different kinds of seams, such as piped seams, where an extra piece of material, the so-called piping, is sewn into the seam. This gives the seam a characteristic appearance with a protruding center. We can simulate this using a more complex profile for the seam, as shown in Figure 11. Figures 13(d) and (e) show a surface feature pattern involving piped seams applied to our example model.

When two features cross it is sometimes sufficient to apply existing rules to the mesh. In other situations new mesh patterns are required to avoid artefacts caused by the interaction of both features. The ability to formulate ad-hoc special cases used by experts as rules that can be applied automatically is one of the strengths of the approach presented in this paper.

Interactions between piped seams and regular seams are such special cases; a three-way junction where a regular seam ends in a piped seam (Figure 13(d)) is very similar to the a regular three-way junction, but a crossing between a piped seam and a regular seam feature, as shown in Figures 13(e), leads to artefacts in the surface if handled by a variant of Rule #4 . Figure 12(a) shows a piped seam crossing a regular seam, constructed using the regular crossing rule, Rule #4. Artefacts occur in the limit surface where the many parallel edges running cause a "kink" in the piping, see Figure 12(a) on the right. To improve the appearance of the subdivision surface we add another rule to the graph grammar to handle this special case in a more appropriate manner, such as to use the strategy from Rule #5 to eliminate the extra control points. The result is shown in Figure 12(b), where no artefacts are visible in the limit surface.

Note that the case of two piped seams crossing each other is yet another special case. It is not a trivial crossing, since the pipings for the two seams cannot occupy the same space. One of the pipings needs to end where it meets the other seam, and resume on the other side. This special case can also be automated by our system; it cannot be dealt with by a general algorithm, as it is a special case that arises purely from the application domain.

8. Discussion

The methods that experts use to manually add a feature to a mesh depend on the existing structure of the mesh, and so does the graph-grammar based approach.

Some of the other methods described in Section 2 apply brute force to this problem: While the grid lines of the control mesh do not follow the generated feature, thus decreasing the quality of the surface due to artefacts, the resulting errors can be made arbitrarily small by increasing the resolution [17].

This is a general trade-off. Generating a higher-resolution output reduces the amount of artefacts, while at the same time makes the output less useful further down the product design

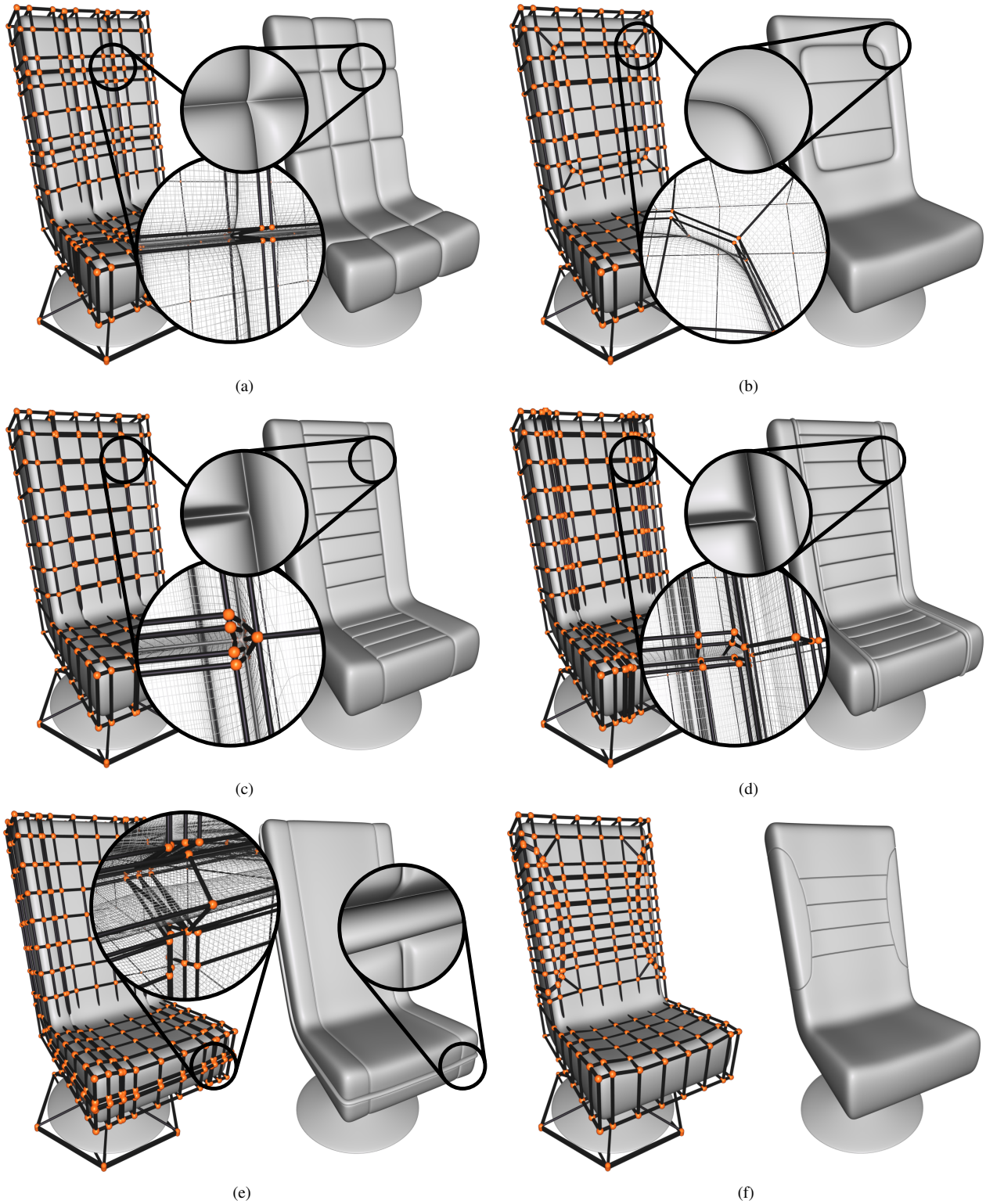

Figure 13: Various different patterns of seam features applied to the same model of a chair. Figures (a), (b) and (c) showcase the crossing rule (Rule #4), the corner rule (Rule #3) and a T-junction rule (Rule #6) when applying regular seams. Figures (d) and (e) demonstrate the application of different feature types, namely regular and piped seams. While the T-junction in (d) follows the same rule as involved in (c), the crossing in Figure (e) requires a special case rule to improve the appearance of the limit surface. Finally, (f) shows that features need not be aligned to grid lines.

pipelines, since alterations to the dense control mesh are more difficult to realise.

In our approach the output is a modified subdivision control mesh which incorporates a chosen surface feature at a small increase in control mesh density which is only locally applied. Therefore, the resulting control mesh is ideally suited for further manual modification.

At this point, writing or extending a graph grammar ruleset is not a trivial task and cannot be left to the designer; while the graph grammar abstraction markedly reduces the effort required compared to hard-coding the mesh manipulations as a monolithic script or program, an expert is still required. Applying surface features by invoking the rules, by contrast, is now open to non-experts. Once a rule has been formulated it may be applied countless times via a mouse click.

Special-purpose rules are not to be understood as rules that are used only once, but rather as a way of creating domain-specific modeling tools.

If no rule is available for a specific situation, a modeling system can signal the failure to the user by marking that specific part of the model. The designer can then deal with this particular special case by hand while still profiting from the automated feature application everywhere else in the model. Manual post-modification is made possible by the fact that the output is a subdivision control mesh that is still as sparse as possible.

Graph rewriting is Turing complete, so concerns about the undecidability of the halting problem are reasonable at a first glance. We have, however, found that this is no more a concern than the Turing completeness of the underlying scripting language used for defining the local mesh modifications. Almost all rules that were used for the figures in this paper are non-recursive, with the exception of Figure 10, which shows a recursive refinement process that can easily be controlled by including the desired refinement level as a condition.

9. Conclusion

We have shown that graph grammars can be employed to capture expert knowledge about what mesh operations are necessary to incorporate surface features into subdivision control meshes. Compared to other methods to automatically apply surface features, this method produces an output mesh that has significantly fewer redundant control points and thus better quality control meshes.

Using this method to automate the application of surface features into existing subdivision surface based modeling workflows will significantly speed up the modeling process. Adoption in practice is likely to happen more quickly than for other automated approaches as the output mesh is not only more efficient in terms of the number of control points used, but also very close to the result of manual modeling, allowing for a mix-and-match approach during the adoption phase.

References

[1] Fontana M, Giannini F, Meirana M. A free form feature taxonomy. In: Computer Graphics Forum; vol. 18. Wiley Online Library; 1999, p. 107–18.

[2] Pernot JP, Falcidieno B, Giannini F, Léon JC. Incorporating free-form features in aesthetic and engineering product design: State-of-the-art report. Computers in Industry 2008;59(6):626–37. doi:10.1016/j.compind.2008.03.004.

[3] Catmull E, Clark J. Recursively generated b-spline surfaces on arbitrary topological meshes. Computer-Aided Design 1978;10(6):350–5. doi:10.1016/0010-4485(78)90110-0.

[4] Sabin M, Barthe L. Artifacts in recursive subdivision surfaces. In: Cohen A, Merrien JL, Schumaker LL, editors. Curve and Surface Fitting: St Malo 2002. Nashboro Press, Brentwood, TN; 2003, p. 353–62.

[5] Wonka P, Wimmer M, Sillion F, Ribarsky W. Instant architecture. ACM Transactions on Graphics 2003;22(3):669–77. doi:10.1145/882262.882324.

[6] Müller P, Wonka P, Haegler S, Ulmer A, Van Gool L. Procedural modeling of buildings. ACM Trans Graph 2006;25(3):614–23. doi:10.1145/1141911.1141931.

[7] Pernot JP, Guillet S, Leon JC. A shape deformation tool to model character lines in the early design phases. In: Shape Modeling International, 2002. Proceedings. 2002, p. 165–72. doi:10.1109/SMI.2002.1003542.

[8] Pernot JP, Falcidieno B, Giannini F, Guillet S, Léon JC. Modelling free-form surfaces using a feature-based approach. In: Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications. SM '03; New York, NY, USA: ACM. ISBN 1-58113-706-0; 2003, p. 270–3. doi:10.1145/781606.781649.

[9] Pernot JP, Falcidieno B, Guillet S, Leon JC. Aesthetic design of shapes using fully free form deformation features. In: Tools and Methods for Competitive Engineering (TMCE); vol. 1. 2004, p. 143–54. ISBN 90–5966-023–4.

[10] Cheutet V, Catalano C, Pernot J, Falcidieno B, Giannini F, Leon J. 3d sketching for aesthetic design using fully free-form deformation features. Computers & Graphics 2005;29(6):916 –30. doi:http://dx.doi.org/10.1016/j.cag.2005.09.009.

[11] Chen W, Cai Y, Zheng J. Freeform-based form feature modeling using a hierarchical & multi-resolution nurbs method. In: Proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry. VRCAI '10; New York, NY, USA: ACM. ISBN 978-1-4503-0459-7; 2010, p. 179–84. doi:10.1145/1900179.1900218.

[12] Chen G, Esch G, Wonka P, Müller P, Zhang E. Interactive procedural street modeling. In: ACM SIGGRAPH 2008 papers. SIGGRAPH '08; New York, NY, USA: ACM. ISBN 978-1-4503-0112-1; 2008, p. 103:1–103:10. doi:10.1145/1399504.1360702.

[13] Cook LR. Shade trees. In: SIGGRAPH Proceedings. ACM Press; 1984, p. 223–231.

[14] DeRose T, Kass M, Truong T. Subdivision surfaces in character animation. In: Proceedings of the 25th annual conference on Computer graphics and interactive techniques. SIGGRAPH '98; New York, NY, USA: ACM. ISBN 0-89791-999-8; 1998, p. 85–94. doi:10.1145/280814.280826.

[15] Khodakovsky A, Schröder P. Fine level feature editing for subdivision surfaces. In: Proceedings of the Fifth ACM Symposium on Solid Modeling and Applications. SMA '99; New York, NY, USA: ACM. ISBN 1-58113-080-5; 1999, p. 203–11. doi:10.1145/304012.304033.

[16] Olsen L, Samavati FF, Sousa MC, Jorge JA. Sketch-Based Mesh Augmentation. In: Jorge JAP, Igarashi T, editors. Eurographics Workshop on Sketch-Based Interfaces and Modeling. The Eurographics Association. ISBN 3-905673-30-4; 2005,doi:10.2312/SBM/SBM05/043-052.

[17] Augsdörfer U, Dodgson N, Sabin M. Artifact analysis on b-splines, box-splines and other surfaces defined by quadrilateral polyhedra. Computer Aided Geometric Design 2011;28:177–197.

[18] Augsdörfer U, Dodgson N, Sabin M. Artifact analysis on triangular box-splines and subdivision surfaces defined by triangular polyhedra. Computer Aided Geometric Design 2011;28:198–211.

[19] Peters J, Reif U. Shape characterization of subdivision surfaces — basic principles. Computer Aided Geometric Design 2004;21(6):585–99.

[20] Karciauskas K, Peters J, Reif U. Shape characterization of subdivision surfaces – case studies. Computer Aided Geometric Design 2004;21(6):601–14.

[21] Augsdörfer U, Dodgson N, Sabin M. Tuning subdivision by minimising Gaussian curvature variation near extraordinary vertices. Computer Graphics Forum 2006;25(3):263–72. doi:10.1111/j.1467-8659.2006.00945.x.

[22] Havemann S. Generative Mesh Modeling. Ph.D. thesis; Institute of Com-

puter Graphics, Faculty of Computer Science, Braunschweig Technical University, Germany; 2005. doi:www.digibib.tu-bs.de/?docid=00000008; dOI: http://www.digibib.tu-bs.de/?docid=00000008.

[23] Rozenberg G, editor. Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations. World Scientific; 1997. ISBN 9810228848.

[24] Spicher A, Michel O, Giavitto JL. Declarative mesh subdivision using topological rewriting in mgs. In: Ehrig H, Rensink A, Rozenberg G, Schürr A, editors. Graph Transformations: 5th International Conference, ICGT 2010, Enschede, The Netherlands, September 27–October 2, 2010. Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-15928-2; 2010, p. 298–313. doi:10.1007/978-3-642-15928-2_20.