Eurographics Symposium on Rendering 2010
Jason Lawrence and Marc Stamminger
(Guest Editors)

Volume 29 (2010), Number 4

Grammar-based Encoding of Facades

Simon Haegler[1], Peter Wonka[3], Stefan Müller Arisona[1,2], Luc Van Gool[1,4], Pascal Müller[2]

[1]ETH Zurich; [2]Procedural Inc.; [3]Arizona State University; [4]KU Leuven

Abstract

In this paper we propose a real-time rendering approach for procedural cities. Our first contribution is a new lightweight grammar representation that compactly encodes facade structures and allows fast per-pixel access. We call this grammar F-shade. *Our second contribution is a prototype rendering system that renders an urban model from the compact representation directly on the GPU. Our suggested approach explores an interesting connection from procedural modeling to real-time rendering. Evaluating procedural descriptions at render time uses less memory than the generation of intermediate geometry. This enables us to render large urban models directly from GPU memory.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

In this paper we introduce a new grammar called *F(acade)-shade* that is useful to encode facade structures. The research problem tackled is how to encode facade structures according to the following two goals. *Compactness:* The representation should consume as little graphics memory as possible. An additional benefit should be reduced file sizes on disk and faster network transmission times. *Fast Random Access:* The representation should allow for fast random access given (u, v) texture coordinates. This enables rendering from a compressed representation directly, e.g. using ray-tracing or rasterization.

Our representation is motivated by rendering applications of large urban models. Due to the size of urban models, memory consumption and memory transfer has become a critical bottleneck in rendering. For example, the 55M triangles *Munich* model used in our results consumes over 3.3GB of memory which still poses problems for many consumer GPUs. An application of particular interest is interactive 3D rendering of urban models on devices with little memory, like smart phones, netbooks, tablet PCs, and car navigation systems for urban navigation.

A facade structure in our system is a layout of rectangular textured regions. Each region has assigned a texture atlas id, texture coordinates to look up a texture value in the atlas, a constant displacement depth, and material parameters. We assume that within a facade and especially within a larger city several rectangular regions will share the same textures for doors, windows, walls, etc. See fig. 1 for an example.

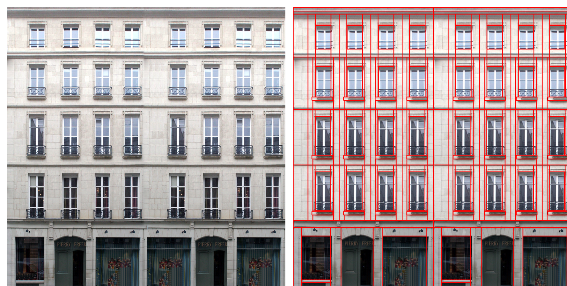

Figure 1: *On the left we show a rectified facade image and on the right the layout of rectangular textured regions that has been encoded using* F-shade. *The rules have been simplified to make all window regions within a floor share the same window texture.*

We consider two alternative approaches to encode facade structures. The first approach is to use shape grammars, e.g. [MWH*06]. Shape grammars are compact representations but in our tests they were about two orders of magnitude too slow for real-time derivation. We could only generate less than 100 buildings per second for the Munich model. The

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.
Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

second approach is to encode facade structure using geometry. Triangulating the rectangular layout makes it possible to reuse textures in the texture atlases and graphics hardware is optimized for rendering triangle meshes. In our results we will compare to the geometry representation and we will show that the memory requirement is still significant and rendering would require out-of-core methods. To represent the texture atlases existing compression algorithms (S3TC / DXTC) are very suitable and they can be used in combination with any representation for facade structures.

The main idea of *F-shade* is to use only simple rules for each facade to develop a grammar that can be derived in real-time. A more powerful rule set is more compact but the rendering speed is typically slower. The real-time grammar introduced in this paper is what we believe to be a good trade-off between rule complexity and evaluation time. Some of our main insights are that conditional rules, parametric rules, and context-sensitive rules make the derivation at least an order of magnitude too slow.

The contribution of our work are the introduction of *F-shade* and a prototype rendering system for it. We limit the scope of the rendering part to demonstrate how buildings encoded by *F-shade* can be rendered. We do not address system issues such as integration with other representations (e.g. pure geometry, impostors, and geometric LODs), and aspects like occlusion culling, memory management, and model transmission over the network.

1.1. Related Work

Our contributions impact the modeling, rendering and the representation of urban environments. We will briefly review related work in these three areas.

Modeling: Synthetic urban models can be generated using procedural methods. Previous work showed how urban layouts consisting of street networks and parcels [PM01] and individual buildings can be modeled using grammars [WWSR03, MWH*06, LWW08]. Even though our grammar shares similarity with existing shape grammars, the main difference is that existing grammars, e.g. [MWH*06] are not suitable for real-time rendering because they are too complex and cannot be evaluated per pixel. Image-based modeling is a great source to obtain *F-shade* models. One of our test scenes was generated by building on recent work in facade analysis [MZWG07, XFT*08].

Rendering: Real-time rendering systems use a combination of established methods, such as occlusion culling [GK, MBW08], image-based simplification [SLS*, GM05], level-of-detail techniques [BGB*05], and triangle data structure optimization [SNB07, Hop], and out of core texture management [BD05]. Alternatively, real-time ray-tracing made significant progress in recent years [WSBW, RSH05] and many developers are speculating that the future of real-time rendering will move towards hybrid ray-tracing and rasterization systems. An example of a hybrid system is the use of rasterization for the coarse geometry, and the rendering of details in a fragment shader ray tracer [POJ05, Don05, BD06, Tat06, CDG*07, AYRW09].

Representation: The philosophy of our work is to render urban models from a compressed representation. Two approaches exist in this context. One approach is to decompress the representation before it is transferred to GPU memory which gives great results for terrain rendering [LH04, GMC*06, DSW]. The second approach is to render the compressed representation directly, which makes fast *per pixel access* essential. In real-time rendering, a very popular method is to use matrix or tensor factorization. This has been suggested for precomputed radiance transfer [LK03], BRDF data sets [KM], and also facade textures [AYRW09]. Another approach is to use epitomes to factor repeating content in large image collections [WWOH]. While previous work is also able to encode facade textures, it is not possible to efficiently encode uv coordinates in a facade structure. Therefore, previous approaches work with procedural textures, but not with texture atlases. Another interesting aspect of our representation is that it encodes boundaries with sub-pixel precision which has also been recognized as an important feature [Sen04].

1.2. Overview

First, we explain how *F-shade* works by describing selected examples and by describing the syntax. We also explain the grammar derivation for *per pixel access* (See section 2). To demonstrate the decompression performance for per pixel access, we present a real-time rendering prototype based on GPGPU and deferred shading (See section 3). Different applications scenarios are presented, i.e. how the rules are created (See section 4). Finally, we present results (section 5) to compare *F-shade* against alternative representations and discuss advantages, limitations, and future work in section 6.

2. F-shade

F-shade is a grammar to encode facade structure over a base polygon that is parametrized by (u,v) texture coordinates. It encodes a layout of rectangular regions where each region has assigned a texture atlas id, texture coordinates (s,t) for all four corners, a constant displacement depth, a diffuse material color, and a specular material color. The most important use of *F-shade* in our real-time rendering framework is to query a facade structure with (u,v) texture coordinates to obtain the texture atlas ID, (s,t) texture coordinates, displacement depth, and the diffuse and specular material color for the corresponding (u,v) sampling point. This output of the grammar derivation can then be combined with existing shading computations, such as Phong shading, per pixel ray tracing, normal mapping, environment mapping, or BRDF evaluations.

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

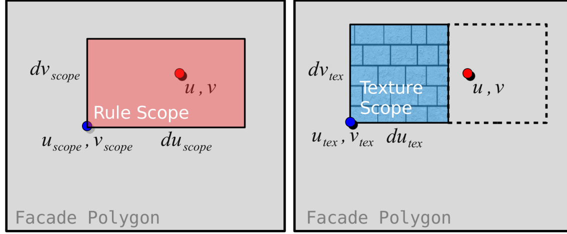

Figure 2: *The grammar manages a rule- and a texture-scope in parallel on the same facade polygon to evaluate a sample at location (u,v). Expressing the sample relative to* (u_{tex}, v_{tex}) *gives the texture coordinates* (s,t). *We mostly use tiled textures.*

The differences to existing shape grammars are the following: *F-shade* cannot access shape information. Therefore it is no longer a shape grammar and we introduce the term *shade grammar*. The grammar does not use conditional rules, stochastic rule selection, or context-sensitive rules. We found that current hardware is still slowed down considerably when the rules are allowed to contain branching instructions. Therefore, to maximize thread-parallelism on the GPU, it is important that the computation of a pixel color can be done with as little branching as possible. This design choice could be re-evaluated with future hardware.

The grammar operates on a fixed set of attributes and only uses relative size and location values. The relative attributes are an important aspect of procedural modeling and allow to generate details for facade polygons of different sizes. Grammar rules are also restricted to one operation per rule and cannot contain any nesting of rules.

These choices seem restrictive, but designing a grammar for real-time rendering is a trade off between compactness and fast per-pixel access. We found that using more complex rules, e.g. semantic rules, parametric rules, or context sensitive rules, quickly gets too expensive in terms of render time. In contrast, adding a few more simple rules or rule parameters can be done without dramatically affecting rendering performance.

Grammar Description: The grammar operates as a state machine that manipulates the following state variables (fig. 2): (1) a *scope* of the rule, given as a parametrization of the current rectangular region. The scope consists of the u_{scope} and v_{scope} position of the lower left corner of the polygon and the extent of the polygon in parameter space (du_{scope} and dv_{scope}). Typically the grammar is invoked with $u_{scope} = 0$, $v_{scope} = 0$, $du_{scope} = 1$, and $dv_{scope} = 1$. A fifth component is the scope depth. (2) A *texture-scope* that is described with four parametrization values similar to the scope. 3) Material properties (e.g. Phong parameters). The grammar is invoked with a sample location (u,v), and the initial rule ID. We use nine rule operations: `Split`, `Repeat`, `Trafo`, `TrafoTex`, `NormTex`, `LookupTex`, `Material`, `Overlay`, and `Multiply`.

`Split` splits the rule-scope along one axis into two or more successors. The first parameter of the rule is the type of axis, u or v. Then comes a list of pairs consisting of split positions and successor rules. `Repeat` splits the scope along one axis to fit as many successors of the same kind as possible. `Trafo` and `TrafoTex` transform the corresponding scope in size and position. Similarly, `NormTex` resets the texture-scope back to 0,0,1,1. `LookupTex` is a terminal and reads a region from the texture atlas specified by an ID. `Material` sets material and shading attributes. `Overlay` and `Multiply` behave similarly. In both cases the grammar generates two or more successors with scopes on top of each other. `Overlay` uses alpha testing to decide the final color and `Multiply` multiplies the colors component-wise. Please note that we only implemented `Overlay` and `Multiply` with the alternative rendering method described in the additional materials. The complete syntax of *F-shade* is described in detail in app. A.

Grammar Derivation: The grammar derivation returns texture information and has the following four steps: 1) take a rule, 2) update the state variables according to the operation (e.g. split finds out in which of the regions the sample position (u,v) lies) and invoke the successor rule. 3) If the successor is a terminal rule, return the state information.

Grammar Example: We use an example to explain the functionality of *F-shade*. The example rules are shown below and a derivation using these rules is illustrated in fig. 3. The first rule splits the complete facade into a ground floor and several other floors ($Floors$) and a cornice on top. In the figure we see how the red position that is going to be sampled falls into the scope of the successor $Floors$. Therefore, rule 2 is selected where the texture scope is modified. The change in texture scope implicitly tiles the texture 5 times in u and six times in the v direction, by scaling the extent by 0.2 and 0.15 respectively. In the example we would subsequently select rules 3, 4, 5, and finally 7. In this case, rule 6 is bypassed by the derivation in the figure. Rule 8 would set the texture-scope to be identical to the scope. This is the standard way of modeling windows and doors.

1: *Facade* $\rightsquigarrow$ `Split` v 0.15 *Groundfloor* 0.8 *Floors* 0.05 *Cornice*
2: *Floors* $\rightsquigarrow$ `TrafoTex` 0 0 0.2 0.15 *FloorsTex*
3: *FloorsTex* $\rightsquigarrow$ `Repeat` v 0.25 *Floor*
4: *Floor* $\rightsquigarrow$ `Repeat` u 0.16 *Tile*
5: *Tile* $\rightsquigarrow$ `Split` u 0.2 *Wall* 0.6 *SubTile* 0.2 *Wall*
6: *SubTile* $\rightsquigarrow$ `Split` v 0.1 *Wall* 0.7 *Window* 0.2 *Wall*
7: *Wall* $\rightsquigarrow$ `LookupTex` wallTextureID
8: *Window* $\rightsquigarrow$ `NormTex` *WindowTex*
9: *WindowTex* $\rightsquigarrow$ `LookupTex` windowTextureID
10: *Groundfloor* $\rightsquigarrow$ ...

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

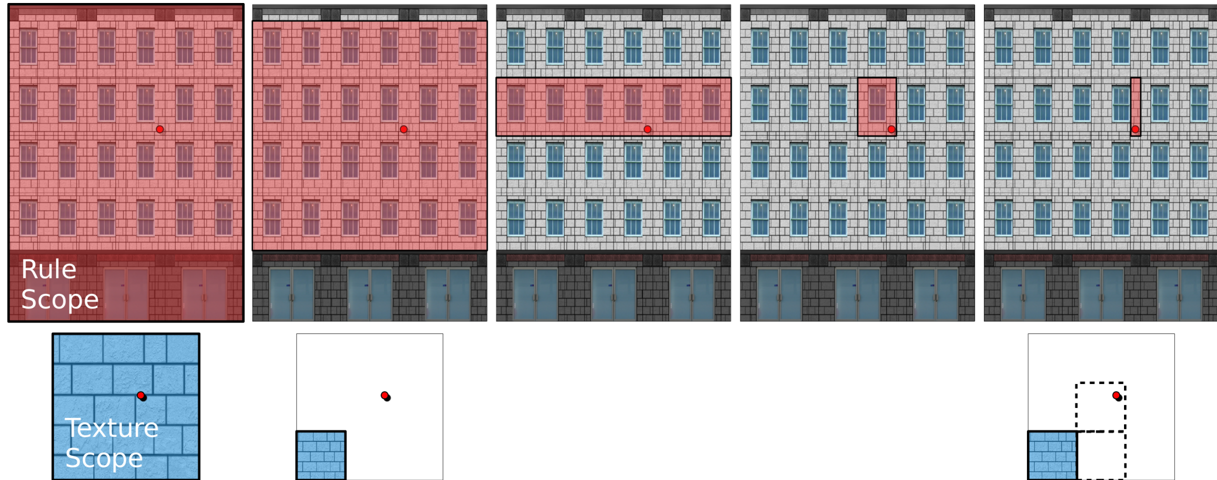

Figure 3: *This figure visualizes the rule set from the grammar example in the text. The images show from left to right the derivation sequence of the rule- and texture-scope for the sample marked with a red dot. Note how the rule scope zooms in on the pixel. As the last step - which is rule 7 for the shown sample - a tileable texture is evaluated (bottom right).*

3. GPGPU Rendering of Compressed Facades using Deferred Shading

In this section we introduce a real-time rendering system based on *deferred shading* that can render building mass models with *F-shade* facade structures. The main goal is to demonstrate the performance of per-pixel random access for a larger model. Our system directly renders from a compressed facade representation and we assume that the complete model is stored in graphics memory. We acknowledge that a competitive real-time rendering system also requires several additional components and we discuss possible extensions and alternative system designs at the end of this section and in section 6.

Deferred shading [Shi05, Koo07] is a popular alternative to computing pixel colors in the fragment shader directly. In the following we will discuss the three main steps of our solution: 1) geometry rendering to establish geometry and shading information of visible fragments into viewport-sized textures, 2) per fragment rule evaluation to decompress the facade representation, 3) final pixel shading including displacement mapping.

Rasterization In the first step the geometry is rendered using vertex buffer objects (VBOs). The VBOs store vertex locations, normals, (u,v) texture coordinates, and the facade ids (i.e. the *F-shade* grammar start symbols). We use the standard OpenGL methods to a) transform the geometry, b) interpolate vertex locations, normals, and (u,v)s using perspective correct interpolation, and c) write all values to an off-screen buffer (FBO). Z-buffering is used to ensure the correct visibility. If displacement mapping is enabled we also compute and store the light direction and the view direction in image space.

`Split`	`axis`	`n`	`args(n)`
`Repeat`	`axis`	`dim`	`succ`
`Trafo`	`args(5)`	`succ`	`0`
`Trafotex`	`args(4)`	`succ`	`0`
`Material`	`args(8)`	`succ`	`0`
`LookupTex`	`tid`	`0`	`0`
`NormTex`	`succ`	`0`	`0`
`Overlay`	`n`	`args(n)`	`0`
`Multiply`	`n`	`args(n)`	`0`

Table 1: F-shade *rule layout on the GPU. The rules can appear in arbitrary order in the main* `int4` *rule array. Each* `succ` *element stores the array index of the successor rule.*

Per-pixel rule evaluation In the second step we decode the facade description. Our implementation uses CUDA, but alternatives such as OpenCL and DirectCompute could be used equivalently. *F-shade* rules are stored as arrays of integer and floating point numbers. We use a main integer array with an `int4` data type for efficient look-up. Each `int4` vector represents one rule and optionally references a float array for arguments. As an exception, the arguments of the split operation have variable length and are put into separate int/float arrays. Each row in table 1 shows a possible element in the main `int4` array.

Based on the syntax introduced in section 2 we encode the rule arguments as follows. The first column identifies the rule (integer constant). `axis` uses 0,1 for the u,v-directions. `succ` is the start address of the successor rule or - in case of the `Split` rule - the start address of `n` integer/float ar-

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

```
foreach pixel {
  shaderparams.reset();
  state.init(FBO.lookup());
  while(state.rule)
    evalRule(state, &shaderparams);
  TBO.write(shaderparams);
}
```

Listing 1: *Pseudo-code of the derivation loop in the main CUDA kernel. First, the FBO values generated in the rasterization step for the current pixel are fetched to setup the initial state. The rule evaluation is repeated as long as there is a valid successor rule. Finally, the resulting shading parameters are written to the texture buffer (TBO).*

guments in the split arrays. `args(n)` is the start address for `n` successive arguments (the type depends on the current rule) and `tid` indicates the index of the texture. Our material setup uses an RGB diffuse and specular component including a weight (8 values in total).

The rule arrays are stored in CUDA memory (located in the GPU VRAM) and accessed as read-only and hardware-cached textures. The output of the rule evaluation is stored in texture buffer objects (TBO) mapped by CUDA. The most important outputs are the (s,t) coordinates of the diffuse facade texture. Additionally, we store the corresponding texture atlas id (i.e. the index into the texture array), depth, and material properties.

The rule derivation is implemented as a loop in listing 1. For each pixel, the function *lookup* reads the FBO values generated in the rasterization step at its corresponding position. *init* sets the initial state for the grammar derivation based on the FBO values. *evalRule* is a sub-function for rule evaluation. This sub-function uses a flow control statement to jump into a code block based on the rule type, reads the data of the corresponding rule, and updates the current state. Thereby, rule operations such as `LookupTex` or `Material` also update the shader parameters. For the execution of the overlay and multiply commands, we extend the scope with a stack data structure to keep track of the parallel branches. After the derivation, *write* writes the final shading parameters into the texture buffer.

This CUDA-based method is the result of a number of evolutions. An earlier approach to evaluate *F-shade* directly in GLSL is described in the additional material to this paper. That method was abandoned due to the limited number of rules it could evaluate with present hardware.

Pixel shading In the third step we apply pixel shading. Based on the pixel position, the fragment shader first fetches the texture ID and the position (s,t) of the pixel in facade coordinates. This information is then used to look-up the color components (diffuse, specular, dirt, environment) in the texture atlas. If enabled, a displacement mapping in image space is inserted before the texture look-ups. This displacement step traces a ray in screen space until it intersects with the facade surface. The ray tracer is an adaptation of the method by Policarpo et al. [POJ05]. The main difference is that we trace a ray in the intermediate buffer rather than in facade tangent space.

4. Application Scenarios

In this section we will sketch several applications of *F-shade*. The detailed descriptions of these applications is beyond the scope of this paper. We do not claim a contribution to modeling, but we believe a better understanding of the data is helpful to interpret the results.

Modeling *F-shade* manually: Rule sets for facade designs can be created using a text editor. The appropriate use of the repeat rule will make the designs size independent. This means that a facade can be retargeted to differently sized rectangular starting shapes. However, complex variations such as the selection of a variety of textures for windows and doors will not be possible. Therefore many designs have to be created to fill a complete city.

Converting orthographic facade textures: Orthographic facade images can be segmented and classified by a combination of interactive editing and automatic analysis [XFT*08], and *F-shade* rules can be extracted [MZWG07]. In order to save texture memory, textures can be processed to select one representative for repeating facade elements. The Paris test scene in the video was modeled using this approach.

Converting procedural urban models: Procedural models can be created using the commercial software *CityEngine* and *CGA-shape*. The conversion of the procedural model requires several steps. We first extract facade polygons and then compute the layout of rectangular textured regions for each of the facades. Using an approach similar to [MZWG07], we then extract *F-shade* rules. The Munich test scene was created in this manner.

A common challenge to all these approaches is that many facades will initially have their own rule set. In order to exploit coherence between facade designs, we implemented an additional optimization algorithm to cluster identical rules. It iteratively traverses the rule tree of the complete model in a bottom-up manner and merges identical rules. In this way, the rule tree is converted into a directed acyclic graph.

5. Results

We evaluate the two most important aspects of *F-shade*: compactness and decoding time during rendering (i.e. time per frame). Our test platform is a Dell Precision T7500 workstation with a Nvidia Quadro 4800 graphics card. Fig 4 summarizes our test scenes: (1) three selected facade designs

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

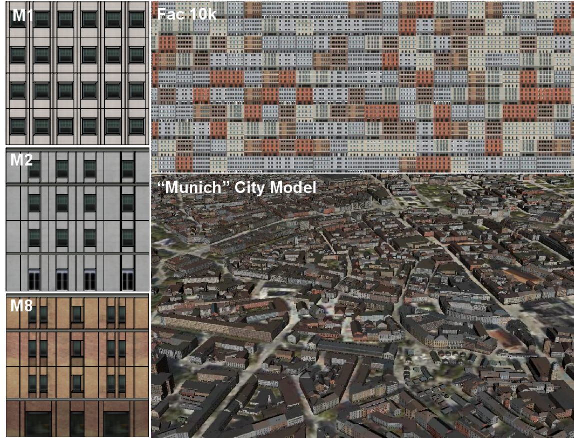

Figure 4: *The three scenes used in the measurements in table 2 and fig 7: (1) 3 selected facade designs (left), (2) 8 designs randomly distributed on a 100x100 facade grid (top right), (3) Munich city model with 42'000 uniquely generated buildings (bottom right).*

Scene	M1	M2	M8	Fac 10k	Munich
Fac Tris	2	2	2	20k	688k
Fac Geo	336b	336b	336b	3.2M	110M
Rules	0.7k	1.4k	1.0k	15.5k	170M
F-shade	1.0k	1.7k	1.3k	3.2M	280M
Full Tris	520	182	388	4.7mio	55mio
Full Geo	35.0k	7.69k	14.0k	244M	3.3G
Compr.	**3%**	**22%**	**9%**	**1%**	**12%**

Table 2: *To evaluate the compactness we measure the memory consumption for the scenarios shown in fig 4. We use the following values:* Fac Tris/Geo*: Number of triangles and size of the facade polygons if stored in a non-indexed VBO layout on the GPU (*F-shade *needs 56bytes per vertex).* F-shade*: The size of our* F-shade *representation for the facade details.* Full Tris/Geo*: Number of triangles and size if the facade model is fully represented in geometry (20bytes per vertex).*

with varying amount of repetition, (2) a random distribution of eight designs on a 100x100 facade grid and (3) the Munich city model consisting of 42'000 uniquely generated buildings.

Compactness: We evaluate the compactness of the *F-shade* representation by measuring the memory consumption of the shade rules and the facade polygons in GPU memory. Table 2 compares our results (*F-shade*) against storing the facade structures using polygons (*Full Geo*). The total model size for *F-shade* is the sum of the facade polygons and the *F-shade* representation. Compared to the full facade geometry, the *F-shade* representation is about 5-30 times smaller for single facades, about 100 times smaller for the facade grid and 8 times smaller for the Munich model. These numbers show that the size of *F-shade* is not directly dependent on the number of facades in a model but has instead a strong dependency on the number of *different* facade designs and the amount of *repeating patterns* in the model (e.g. when comparing facade M1 to M2 and M8, M1 has the smallest rules because it exposes a higher number of repetitions).

Scene	Single Fac	Fac 10k	Munich
Cam 1: Pass 1	1/1/2	1/1/2	8/8/9
Cam 1: Rules	1/1/3	1/2/7	1/5/10
Cam 1: Total	1/2/5	2/4/9	9/14/19
Cam 2: Pass 1	1/1/2	1/1/2	8/8/9
Cam 2: Rules	2/2/3	3/4/5	1/3/5
Cam 2: Total	3/4/5	5/6/7	10/12/14
"Unity" (Avg)	<0.5	3	160 (Ext)

Table 3: *The min/mean/max frame-times in ms of the deferred shading passes. We compare our results with the average render speed of the commercial render engine "Unity". Please note that the frame time for the Munich model in Unity has been extrapolated. The second render pass has been omitted as it is constant and negligible.*

We omitted the size of the texture atlas in tab. 2 as it is used in the *F-shade* and the full geometry representation in the same way. The size of the uncompressed atlas for the Munich model is 12MB. If we represented our facade designs M1 to M8 as single textures, they would have an average size of 20MB. Extrapolating this to the 300k facade designs in the Munich model, a model representation which uses one single texture per facade is clearly not practical.

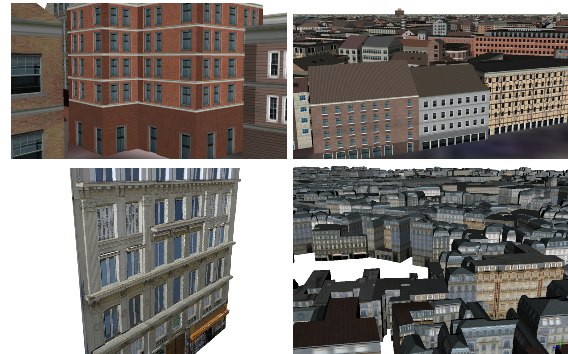

Figure 5: *The top row shows the Munich model, the bottom row shows the Paris model. The frames on the left show detailed views with screen-space displacement enabled.*

Decoding Speed: We evaluate our shading algorithm for the three test scenes. We use two different camera animations: *Cam1* zooms to the model from far away and *Cam2* pans over the model. Fig. 5 contains selected frames from

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

the accompanying video in the supplemental material, fig. 6 exemplifies five frames from the *Cam1* sequence, and fig. 7 shows the resulting timings for the three passes of our deferred shading implementation. Table 3 summarizes and compares the rendering times with the state-of-the-art rendering engine "Unity". Because the model consumes over 4GB of memory, we only managed to load a part of the Munich model into Unity, which renders at 40ms per frame with 10mio triangles. As expected, rendering using *F-shade* is slower as geometry based rendering for smaller models, but becomes more efficient for larger models (about 10 times faster).

6. Discussion

Our results show that for large city models with many textures our *F-shade* representation is smaller and renders faster in-core than a model which contains all the facade details as geometry. In the following, we examine a number of aspects in more detail.

Limitations of the Representation: In our current implementation we are limited to one constant displacement value per rectangular region. However, a straightforward extension could include additional displacement maps for finer structures, such as brick and mortar or facade ornaments. Our representation is therefore also suitable for complex details that can be stored in displacement maps. Another simple extension would be to use classification maps to label facade regions thereby encoding higher level semantics. In contrast, *F-shade* will not work well if each rectangular region has a unique texture or if the arrangement of elements is not aligned properly. Therefore, *F-shade* is not a solution to encode general textures and structures.

Resolution Independence: *F-shade* encodes rectangular regions on a facade polygon where the boundaries can have arbitrary floating point locations. If we resample a facade structure into a single image, the boundary locations have to be rounded to the closest pixel locations. These errors will be most visible for facade elements that are thin compared to the size of a facade, such as window frames.

Rule Complexity: One important design choice of the grammar is to decide on how many types of rules to use. An alternative approach would be to simplify the rule set even further. For example, we also experimented with grammars that only have simple split rules. These grammars more closely resemble axis aligned BSP-trees or kD-trees. However, we manipulate several variables ((s,t), depth, material parameters, texture atlas id) and our experiments indicate that we need to store changes to the variables at interior nodes to ensure compactness. Consequently, the derivation of the grammar cannot be greatly simplified as most of the operations manipulate different state variables. We also tested the combination of relative and absolute split dimensions which allows for better size-independent operations. Unfortunately, we found the additional branching necessary to evaluate these mixed arguments to be too expensive.

Render Performance: The render times in fig. 7 show three major dependencies: (1) a major dependency on the number of *active* pixels (i.e. pixels which are inside a facade), (2) a strong dependency on the number of active pixels belonging to *different* facade designs, (3) a minor dependency on the size of the *F-shade* rules. The first dependency is explained by the rule evaluation time being approximately proportional to the number of evaluated pixels. The second observation is explained by the thread divergence of CUDA if a lot of different facades are rendered. The third dependency is explained by the impaired look-up performance when the *F-shade* rule arrays get larger (the arrays are implemented as cached CUDA textures). Therefore, the slowest frame-times correspond to situations when a large number of pixels are covered by the model, but there is little coherence between neighboring pixels.

We do not yet use any visibility-based acceleration method to render the facade polygons, therefore the first render pass is more or less constant and defines the lower bound of the frame time (about 1ms for the small models and 8ms for the Munich model). We confirmed the performance of the first pass with other VBO based renderers which take 5ms to render the Munich model without facade details. The difference of 3ms is due to the multiple render targets in our implementation, which make the first render pass also fragment-limited. Visibility computation would make a big difference on the number of rendered polygons, especially at street level.

Render Quality: Anti-aliasing is not yet included in our rendering method. An extension that works well with deferred shading is a version of multi-sampling that evaluates more samples close to internal edges of a facade and uses alpha-blending to smoothen the transitions between facade regions. A related idea is to stop the evaluation of scopes that have a similar size as a pixel and return a reference to a (precomputed) average color instead. As a positive side-effect, the latter method will also serve as a level-of-detail constraint and reduce evaluation time. Our displacement mapper is designed for the typical rectangular regions inside a facade and cannot deliver the same visual quality as full geometry rendering.

Future Work: Our current displacement mapping implementation is basically a simple ray caster. In a future implementation, it would be worthwhile to explore the use of *F-shade* in a pure ray tracing architecture. Another interesting extension for devices with low graphics memory would be to develop a rule optimization scheme which trades visual quality for smaller rule size (i.e. a "lossy" compression). In a similar way, lossy compression could also be applied to the facade regions itself by merging together *similar* texture regions, e.g. to reduce the number of different window types.

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

Figure 6: *The figure shows five frames of the "Cam 1: City Model" sequence as measured in fig. 7. The Munich city model is the biggest test set we evaluated. Such a large model can be easily rendered in the core independently from the point of view. On the left, the whole city model is visible and on the right, facades details are recognizable - and both viewpoints are rendered at almost similar frame time using* F-shade.

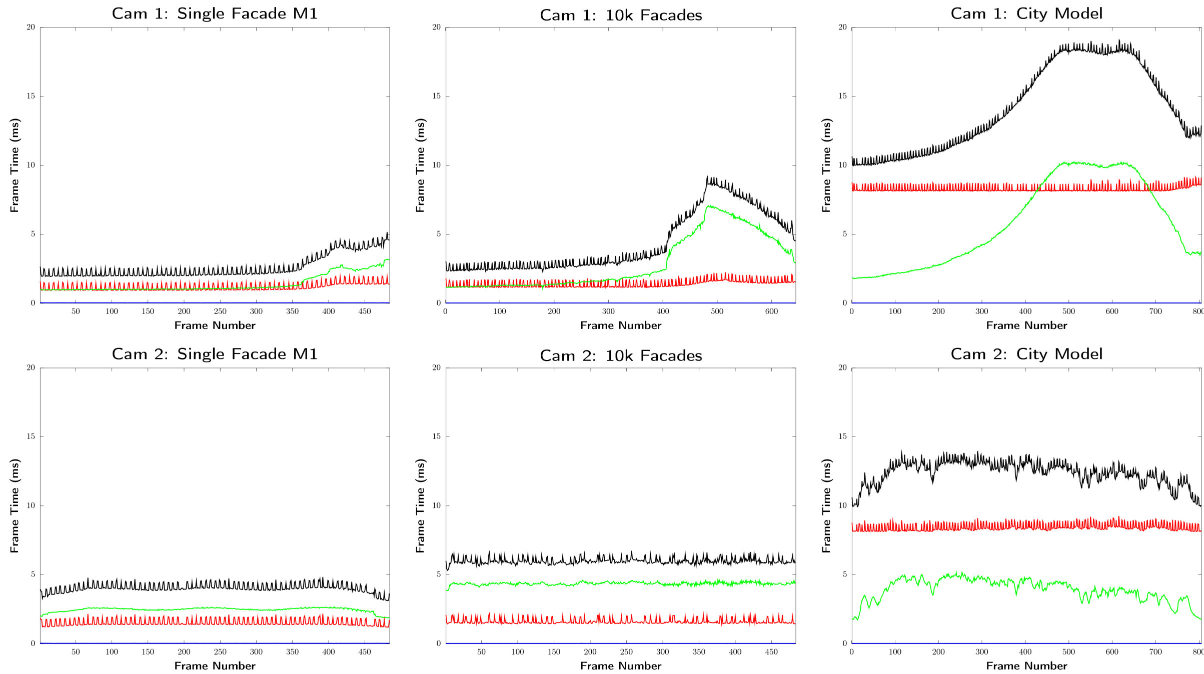

Figure 7: *We measured the total frame-times (in black) for the three test scenes (single facade M1; facade grid; Munich model) and two camera animations (Cam 1: zoom, Cam 2: pan) at 1024x768 pixels resolution. We measure the timings of the three main components of our deferred shading algorithm (section 3): (1) The first render pass is the rasterization step (in red). (2) The second pass executes the* F-shade *rules using CUDA (in green). (3) Finally, a GLSL shader reads the evaluated* F-shade *rules and performs texture look-up and Phong shading (in blue, almost zero). The screen-space displacement was disabled for the measurements. See the accompanying video and table 3 for a visual correlation of frame-time and camera position. We suppose that the oscillation between the two visible values in the first render pass (red) is an effect of the VBO handling of the GPU driver.*

7. Conclusion

In this paper we described *F-shade*, a grammar-based representation for facade textures. We argued how the demands of rendering applications require a facade representation to be *compact* and to provide fast *per-pixel* access. Our results show that our representation is able to fulfill these two goals simultaneously in contrast to existing alternatives.

Acknowledgments

The authors would like to thank Andreas Ulmer, Basil Weber, Matthias Specht and Matthias Buehler for their contribution to the city models and Michael Wimmer and Oliver Mattausch for reference rendering tests. This work has received funding from the European Community's Seventh Framework Programme (projects 3D-COFORM and V-CITY), NVIDIA, EURO #33234234, and the National Science Foundation.

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.

Appendix A: *F-shade* Syntax

```
Rule         =   Predecessor ⤳ ShadingOp
ShadingOp    =   Split | Repeat | Trafo | TrafoTex | LookupTex
                 | Material | Overlay | Multiply
Split        =   "Split" Axis Float Successor {Float Succes-
                 sor}
Repeat       =   "Repeat" Axis Float Successor
Trafo        =   "Trafo" Float Float Float Float Float Suc-
                 cessor
TrafoTex     =   "TrafoTex" Float Float Float Float Succes-
                 sor
NormTex      =   "NormTex" Successor
LookupTex    =   "LookupTex" TextureID
Material     =   "Material" Float Float Float Float Float
                 Float Float Float Successor
Overlay      =   "Overlay" Successor Successor {Successor}
Multiply     =   "Multiply" Successor Successor {Successor}

Predecessor =    String | Integer
Successor    =   String | Integer
TextureID    =   Integer
Axis         =   "u" | "v"
```

References

[AYRW09] ALI S., YE J., RAZDAN A., WONKA P.: Compressed facade displacement maps. *IEEE TVCG 15*, 2 (2009). 2

[BD05] BUCHHOLZ H., DÖLLNER J.: View-dependent rendering of multiresolution texture-atlases. In *IEEE Visualization* (2005), p. 28. 2

[BD06] BABOUD L., DÉCORET X.: Rendering geometry with relief textures. In *GI '06: Proceedings of the 2006 conference on Graphics interface* (Toronto, Ont., Canada, Canada, 2006), Canadian Information Processing Society, pp. 195–201. 2

[BGB*05] BORGEAT L., GODIN G., BLAIS F., MASSICOTTE P., LAHANIER C.: GoLD: interactive display of huge colored and textured models. *ACM Transactions on Graphics 26*, 3 (July 2005), 869–877. 2

[CDG*07] CIGNONI P., DI BENEDETTO M., GANOVELLI F., GOBBETTI E., MARTON F., SCOPIGNO R.: Ray-casted blockmaps for large urban models visualization. *Computer Graphics Forum 26*, 3 (2007), 405–413. 2

[Don05] DONNELLY W.: Per-pixel displacement mapping with distance functions. *GPU Gems 2* (2005). 2

[DSW] DICK C., SCHNEIDER J., WESTERMANN R.: Efficient geometry compression for GPU-based decoding in realtime terrain rendering. *Computer Graphics Forum 2009*. 2

[GK] GREENE N., KASS M.: Hierarchical Z-buffer visibility. In *SIGGRAPH 93 Conference Proceedings*. 2

[GM05] GOBBETTI E., MARTON F.: Far voxels: a multiresolution framework for interactive rendering of huge complex 3D models on commodity graphics platforms. *ACM Transactions on Graphics 24*, 3 (July 2005), 878–885. 2

[GMC*06] GOBBETTI E., MARTON F., CIGNONI P., BENEDETTO M. D., GANOVELLI F.: C-BDAM – compressed batched dynamic adaptive meshes for terrain rendering. *Computer Graphics Forum 25*, 3 (Sept. 2006). 2

[Hop] HOPPE H.: Optimization of mesh locality for transparent vertex caching. In *SIGGRAPH 99 Conference Proceedings*. 2

[KM] KAUTZ J., MCCOOL M. D.: Interactive rendering with arbitrary BRDFs using separable approximations. In *Proceedings of the Eurographics Workshop on Rendering 99*. 2

[Koo07] KOONCE R.: Deferred shading in Tabula Rasa. *GPU GEMS 3* (2007). 4

[LH04] LOSASSO F., HOPPE H.: Geometry clipmaps: terrain rendering using nested regular grids. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 769–776. 2

[LK03] LEHTINEN J., KAUTZ J.: Matrix radiance transfer. In *Proceedings of the ACM SIGGRAPH 2003 Symposium on Interactive 3D Graphics* (Apr. 2003), pp. 59–64. 2

[LWW08] LIPP M., WONKA P., WIMMER M.: Interactive visual editing of grammars for procedural architecture. *ACM Transactions on Graphics 27*, 3 (Aug. 2008), 102:1–10. Article No. 102. 2

[MBW08] MATTAUSCH O., BITTNER J., WIMMER M.: Chc++: Coherent hierarchical culling revisited. *Computer Graphics Forum (Proceedings Eurographics 2008) 27*, 2 (Apr. 2008), 221–230. 2

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural Modeling of Buildings. In *Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics* (2006). 1, 2

[MZWG07] MÜLLER P., ZENG G., WONKA P., GOOL L. V.: Image-based procedural modeling of facades. *ACM Transactions on Graphics 24*, 3 (2007), 85. 2, 5

[PM01] PARISH Y. I. H., MÜLLER P.: Procedural modeling of cities. In *Proceedings of ACM SIGGRAPH 2001* (2001), Fiume E., (Ed.), ACM Press, pp. 301–308. 2

[POJ05] POLICARPO F., OLIVEIRA M. M., JO A. L. D. C.: Real-time relief mapping on arbitrary polygonal surfaces. In *Symposium on Interactive 3D graphics and games* (2005). 2, 5

[RSH05] RESHETOV A., SOUPIKOV A., HURLEY J.: Multi-level ray tracing algorithm. In *ACM SIGGRAPH* (2005). 2

[Sen04] SEN P.: Silhouette maps for improved texture magnification. In *Graphics Hardware* (2004), pp. 65–74. 2

[Shi05] SHISHKOVTSOV O.: Deferred shading in S.T.A.L.K.E.R. *GPU GEMS 2* (2005). 4

[SLS*] SHADE J., LISCHINSKI D., SALESIN D., DEROSE T., SNYDER J.: Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH 96 Conference Proceedings*. 2

[SNB07] SANDER P. V., NEHAB D., BARCZAK J.: Fast triangle reordering for vertex locality and reduced overdraw. *ACM Trans. Graph. 26*, 3 (2007), 89. 2

[Tat06] TATARCHUK N.: Dynamic parallax occlusion mapping with approximate soft shadows. In *Symposium on Interactive 3D graphics and games* (2006). 2

[WSBW] WALD I., SLUSALLEK P., BENTHIN C., WAGNER M.: Interactive rendering with coherent ray tracing. In *EG 2001 Proceedings*. 2

[WWOH] WANG H., WEXLER Y., OFEK E., HOPPE H.: Factoring repeated content within and among images. In *ACM SIGGRAPH 2008 papers*. 2

[WWSR03] WONKA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *ACM Transactions on Graphics 22*, 3 (2003), 669–677. 2

[XFT*08] XIAO J., FANG T., TAN P., ZHAO P., OFEK E., QUAN L.: Image-based façade modeling. *ACM Transactions on Graphics 27*, 5 (2008), 1–10. 2, 5

© 2010 The Author(s)
Journal compilation © 2010 The Eurographics Association and Blackwell Publishing Ltd.