Multiresolution Geometric Details on Subdivision Surfaces

Przemyslaw Musialski*
Bauhaus-University Weimar
VRVis Research Center

Robert F. Tobler†
Stefan Maierhofer‡
VRVis Research Center

Charles A. Wüthrich§
Bauhaus-University Weimar

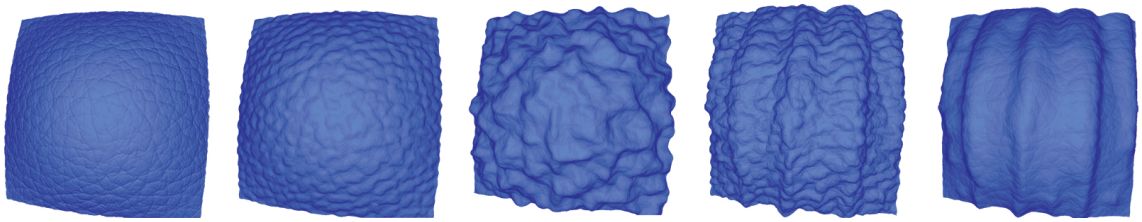


Figure 1: *A patch with calf leather pattern. From left to right different frequency bands of the underling displacement map has been enhanced on different subdivision resolutions. Last two patches on the right-hand side use low frequencies from a different sample.*

Abstract

Subdivision surfaces are methods for creating smooth surfaces out of coarse polyhedral meshes. Due to their recursive nature they are ideally suited for adding geometric detail on different resolutions. When modeling real-world surfaces it is possible to extract the fine surface details from a material and apply these on dense meshes in the form of vertex displacements. Material characteristics are a mixture of features at different scales, which can be recovered by a frequency decomposition of an input height map. Applying these sub-bands as displacement in a recursive multiresolution fashion allows the ability to influence or mix details obtained from one or more sources.

This paper presents a method for computing multiresolution displaced subdivision surfaces on the GPU that performs in real-time and provides an interactive control over the obtained results.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Boundary representations

Keywords: subdivision surfaces, displacement mapping, multiresolution, real-time rendering

1 Introduction

Real-life surfaces are very rarely completely smooth. Most natural surfaces exhibit many levels of roughness and coarseness, especially if they are inspected in detail. Therefore many computer-generated models of real-life objects which are rendered with perfectly smooth surfaces appear sterile and artificial. Over the years computer graphics has developed several approaches to avoid this problem. One of the first methods in this category was bump mapping [Blinn 1978], and its improved descendant, parallax mapping [Oliveira et al. 2000]. Both of these methods do not influence the actual geometry of the underlying model, but change its rendered appearance.

On the other hand, a whole area of computer graphics research deals with subdivision surfaces, a powerful tool for representing objects with smooth surfaces. Due to their nature, subdivision surfaces can be either rendered using successively finer approximations or evaluated exactly for selected surface parts. Since subdivision surfaces can be generated from base geometry of any topology, they are very easy to handle in the modeling process, and have therefore become the tool of choice in applications which aim at highly realistic appearance of smooth models.

In this paper we present a method to apply multiresolution geometric detail on subdivision surfaces in real-time. The additional surface detail can be generated artificially, handcrafted or based on an analysis of a real surface sample. During the rendering process, it can be applied automatically in form of multi-band displacement maps on different subdivision levels of the underlying surface.

As a basis for our work, the next section presents an overview of subdivision surfaces and the displacement approach as well as a brief explanation of sub-band decomposition. The novelty of our work lies in the combination of these techniques and its implementation on graphics hardware, which are explained in sections 3 and 4. Sections 5 and 6 present our results and conclude the work.

2 Related Work

2.1 Subdivision Surfaces

Mesh subdivision is a technique for generating smooth surfaces that has been introduced quite some time ago [Catmull and Clark 1978; Doo and Sabin 1978]. The standard subdivision process starts out with a mesh $M^{(0)}$ composed of vertices, edges, and faces that serves as the base for a sequence of refined meshes $M^{(0)}, M^{(1)}, M^{(2)}, ..., M^{(k)}$ which converges to a limit surface for $k \to \infty$. This surface, also called the subdivision surface, possesses certain continuity properties (i.e. C^2).

For a long time the theoretical foundation of the subdivision process was not as thoroughly developed as for other modeling techniques such as B-splines and the more general NURBS, and thus it

*musialski@vrvis.at
†rft@vrvis.at
‡sm@vrvis.at
§caw@medien.uni-weimar.de

took a while for subdivision methods to become widely adopted. In the last decade this has been rectified by the introduction of methods to analyze and evaluate subdivision surfaces at any point [Reif 1995; Stam 1998], a method for extending subdivision surfaces to approximate NURBS [Sederberg et al. 1998], the addition of controlled boundaries [Biermann et al. 2000], and a method to derive NURBS patches from subdivision surfaces [Peters 2000]. A number of other extensions to subdivision surfaces [DeRose et al. 1998; Lee et al. 2000; Ying and Zorin 2001; Boier-Martin and Zorin 2004] have established them as the modeling tool of choice for generating topologically complex, smooth surfaces.

2.1.1 Evaluation Approaches of Subdivision Surfaces

Many researchers have focused on the problem of fast evaluation and real-time rendering of subdivision surfaces. In general, besides the exact evaluation [Halstead et al. 1993; Stam 1998; Zorin and Kristjansson 2002] three major strategies can be distinguished: (1) forward differencing, [Pulli and Segal 1996; Bischoff et al. 2000], (2) precomputed and tabulated basis functions [Bolz and Schröder 2002; Bolz and Schröder 2003] and the most known and widely used (3) recursive evaluation [DeRose et al. 1998; Biermann et al. 2000]. In our work we use the latter one, since it is more flexible than precomputed basis functions and more stable, than the numeric method of forward differencing. In addition to that, it is naturally suited to introduce variations at each recursive evaluation level. One disadvantage of this approach is the exponential growth of the geometric data, where usually only a small part is visible at a particular moment.

In order to reduce the enormous space complexity of the subdivision process, Pulli and Segal have introduced a strategy for evaluating and rendering which they called sliding window [Pulli and Segal 1996]. This technique uses a fixed memory portion which contains only a (small) piece of the actual surface. This piece becomes evaluated with the forward differencing method and rendered. After that the data is discarded, and a next part of the surface can be processed in the same way. This has been adapted for a modern-hardware implementation of Loop surfaces [Bischoff et al. 2000]. A similar strategy has been presented for recursive subdivision, where mesh parts become evaluated depth-first [Müller and Havemann 2000; Settgast et al. 2004]. We take advantage of this strategy by combining it with a hardware implementation approach. Shiue *et al.* have introduced a hardware kernel based on recursive evaluation [Shiue et al. 2005]. They split the mesh into fragments, where the neighborhood is spirally wrapped around each vertex. In their subdivision kernel the neighborhood can be obtained in terms of the spiral ordering. Our implementation uses quadrangular mesh patches and benefits from the capabilities of hardware which allows to reinterpret textures directly as vertex data in video memory.

2.1.2 Catmull-Clark Subdivision Scheme

Catmull-Clark subdivision is a generalization of the bivariate cubic B-spline surfaces. It accepts any polygonal input domain with convex polygons and even very complex topological shapes. There are no limitations to the valences (number of incident edges) of the vertices, but all vertices with a valence of four are called regular. Regions containing such vertices converge exactly to bicubic B-spline patches with C^2 continuity. Vertices with other valence are called *extraordinary* and the surface regions in their vicinity do not match a regular B-spline, albeit they still maintain C^1 smoothness. By the recursive evaluation, after the very first refinement step, any general mesh is split into quadrangles only. Also only in this step new extraordinary vertices are introduced at faces different to quadrilaterals. In all following recursions, the algorithm proceeds always in the same manner by splitting each quad into four new pieces and the number of extraordinaries remains constant.

For our approach we have adapted the Catmull-Clark scheme as presented by DeRose [DeRose et al. 1998] and implemented it for the GPU kernel. Section 4 presents the details of our implementation.

2.2 Displacement Mapping

Displacement mapping is a well known and rather simple method to apply geometric detail to a surface. It was introduced by Cook in order to enhance silhouettes of objects [Cook 1984]. Other applications of this technique can be found in the terrain rendering domain, where it is often referred as height field mapping.

2.2.1 General Displacement

The general approach of displacement mapping is to shift each parametrized surface point (u, v) in some direction by an offset value fetched from a map $d(u, v)$. The most obvious choice for the direction is the local surface normal vector $N(u, v)$. Let the surface S be parameterized by $X : U \subset \mathbb{R}^2 \rightarrow S \subset \mathbb{R}^3$. Following this parametric space, a displacement function is defined as

$$\hat{X}(u, v) = X(u, v) + d(u, v) \cdot N(u, v) \tag{1}$$

In the case of discrete meshes the normal $N(u, v)$ can be easily computed on each vertex $\mathbf{v}_i$ as an (weighted) average of its incident face normals. Furthermore, in case of Catmull-Clark surfaces also an exact limit surface normal can by computed [Halstead et al. 1993].

2.2.2 Displacement of Subdivision Surfaces

Displacement mapping has already been used in combination with subdivision surfaces in order to achieve mesh compression [Lee et al. 2000]. It has also been researched in order to manually model high detailed geometry. Bunnell proposed a method in hardware to model complex shapes (in [Pharr and Fernando 2005]). An other approach presented a method to define arbitrary curves along subdivision surfaces which serve as a path for displacement in order to generate sharp or semi-sharp creases [Khodakovsky and Schröder 1999]. This method has been extended to cut mesh pieces away by enabling trimming curves on the surfaces [Biermann et al. 2002].

Velho *et al.* introduced a way to synthesize shape features on subdivision surfaces using multiscale procedural displacement [Velho et al. 2001]. Similar to our approach they displace particular subdivision levels with variable scales. As a result they can synthesize several new classes of objects. A similar approach uses fractal displacement to generate naturally looking tree bark [Tobler et al. 2002]. Both apply the detail in a pre-processing step and thus do not benefit from any savings that can be achieved by procedural rendering.

2.3 Image Pyramids

In our approach we decided to focus on different frequencies of a specified displacement function. Therefore we construct the laplacian image pyramid which was introduced in the early eighties [Burt and Adelson 1983]. This algorithm decomposes a source image into individual frequency bands by sub-sampling and smoothing the image with a gaussian filter, super-sampling it into original size and differencing the result from the previous step. By repeating this operation recursively the well known laplacian pyramid is constructed with steps of approximately one octave magnitude.

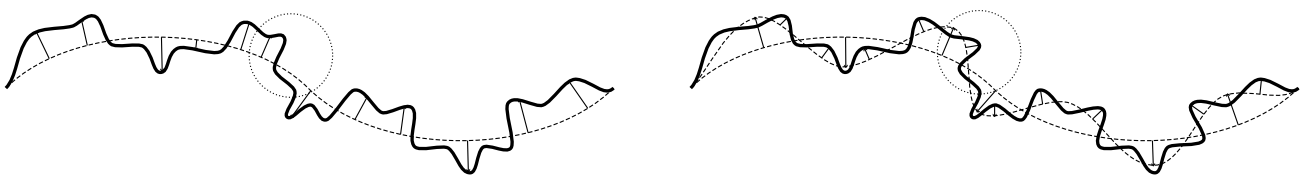


Figure 2: *Displacement of a surface along normals. Left: one-time displacement. Right: Multiresolution displacement. Note that in this case concave folds are possible (dotted circle).*

3 Multiresolution Displacement

The main goal of displacement methods is the generation of more complex shapes by introducing geometric variation across the surface. Our approach uses the same basic idea whereby we consecutively enhance local detail geometry by features obtained from spectral sub-bands of one or more sample patterns. In this section we will describe the details of our method.

3.1 Idea Outline

Recalling the recursive subdivision surfaces method, it appears quite naturally to research the behavior of a surface while some additional geometric information might be added into this sequential process. Since at each step a parametrized and explicit representation of the mesh is automatically generated, it is an easy task to perform displacement on its vertices. Doing it recursively induces a – perhaps uncontrolled – successive surface distortion.

The interesting property of this combination is the fact that the resolution of the mesh becomes twice as dense with each step. For this reason consecutively finer details can be modeled on the net – or lets say modulated on the net – with growing density.

To achieve multiresolution details it is necessary to examine a given displacement texture for its coarse and fine characteristics. As it is known from signals in the frequency domain, their globally prominent characteristics are carried by the low and the fine traits by the high frequencies. These particular bands often play a key role for the appearance of an image sample. A repeating pattern of a tileable texture depends mainly on the high frequencies, while the low bands might be stretched over many small tiles or might be replaced. As a result, especially for human perception, an uniform but non-repeating field with the same pattern can be generated. Hence it is interesting to take this fact into account in order to create differently sized features on a surface.

Finally also the size of the details play a role during the rendering. While viewed from the far, only very coarse features are of interest, in the closeup fine ripples enhance the authenticity of the models' surface. If an object is inspected in a closeup view, only a limited portion of its entire shape is currently visible in the view port. Our approach provides the ability to balance the actual amount of data provided to the display on-the-fly by an adaptive tessellation. It can be used to achieve an optimal visual appearance while rendering in real-time.

3.2 Sub-Band Displacement Maps

Proper displacement maps are usually created by examining an existing pattern and encoding its fine surface properties into a height field. Adding these heights to a smooth surface by a one-time shift results in a proper reconstruction of these details (see Figure 2, left-hand side). In contrast, our approach is to separate these characteristics and to apply them consecutively to the surface on different subdivision levels (Figure 2, right-hand side).

We isolate the sub-bands by creating the laplacian pyramid of an input image (see section 2.3). The result of such an analysis is a set of displacement maps with a scope of one octave each. In the spatial domain each band represents features of similar spatial sizes relative to the entire displacement range. Their sum results in the original signal back again. Applying the low-frequency bands on a rather coarse subdivision surface resolution deforms the shape of the given object more significantly and modulates the prominent features. A sequential adding of more higher frequency bands on the increasingly dense net adds the desired detail onto it.

3.2.1 Resolution Matching

An interesting analogy to the subdivision procedure can be established in this context, since each sub-band differs from its predecessor by exactly one power of two. This means that an image of the size $(2^n) \times (2^n)$ can be decomposed in n different bands with growing density of contained features. Also the subdivision procedure refines a mesh in a power of two manner, thus the density of the vertices of one quad increases by $(2^k + 1) \times (2^k + 1)$, where k is the subdivision level.

This analogy can be used to map particular sub-bands to corresponding sub-levels. Of course the minimal reasonable resolution is that of the mesh, since values for some vertices would have to be interpolated otherwise. In our implementation the maximal subdivision level in hardware has been set to 7 due to memory constraints. Including two previous software subdivision steps, an initial mesh might be subdivided up to ninth level. This would result in $(2^9+1)^2$ vertices for a single input quad, which has to be converted by a sufficient large displacement map. Because objects are often covered entirely by one texture coordinate layer in range $[0..1]$ the size of the displacement map has to be adapted accordingly.

Note that in order to model realistically looking surfaces, the actual size of the features in the height field in relation to the objects' size in the embedding space requires additional semantic information. Since this issue cannot be determined algorithmically it is left to the responsibility of the user to supply input patterns and initial meshes with properly balanced sizes.

3.2.2 Combining Sub-Bands

Applying the displacement consecutively on each subdivision level enriches the resulting surface by normal perturbations which are a consequence of the previous steps. Figure 2, left-hand side, depicts this issue. Note that even concave folds can be created by this sequential approach. While this effect might be desired in some cases, it can also cause undesired artifacts (i.e. self-intersection) on the surface.

In order to influence this behavior, certain subsequent sub-bands can be combined to one. Subdivision steps corresponding to these frequencies then can be performed without displacement up to the

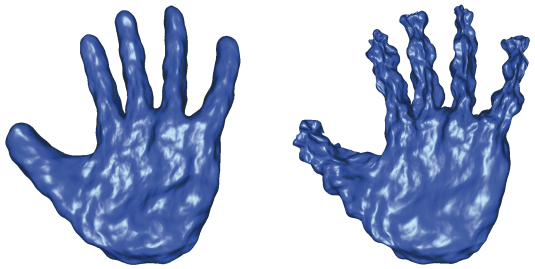


Figure 3: *Left: local scale factor – it nestles to the curvature. Right: static value – the offset on the fingers is the same as on the palm.*

particular mesh resolution and after that shifts to the vertices can be applied accordingly to the combined displacement map along the same normals. Since the addition of the sub-bands does not change any properties of the pattern — in fact it delivers just a sum of few particular frequencies — it can be easily done without any impact on the correctness of the displacement. This extension is especially useful if two different input patterns should be combined. The main shape of the surface is modeled by the characteristics of one source and the finer details are added from a second source along another set of normals (see Figure 12).

3.3 Applying Displacement

During the recursive subdivision we shift the vertices accordingly to the offset contained in a particular sub-band. Since the maps represent planar functions but the surface is usually curved, constant displacement leads often to undesired effects (Figure 3). Therefore, we normalize each of the bands to the range of $[0..1]$ and rescale it back to an appropriate size in consideration of local surface properties.

3.3.1 Local Scaling

We define these as a *local scale factor* for each vertex on each subdivision level. In order to express the local properties, we derive it from the average edge length $L(\mathbf{v})$ of a vertex. Recall that in terms of the subdivision the edges are nearly bisected at each refinement step, such that it can be stated: $L(\mathbf{v}^k) \approx \frac{1}{2} L(\mathbf{v}^{k-1}) \approx \frac{1}{2^k} L(\mathbf{v}^0)$. This is due to the fact that the density of the mesh changes at each step by $f(k) = 2^k$. It can be seen as the frequency f of the subdivision as well as of the image pyramid. Its inverse proportional delivers fractal-similar scaling of the form $\frac{1}{f}$. Adjusting the offset by this quantity recovers the features of the original signal, the scale however reflects the local properties of the surface.

Note that L can be measured in both parametric as well as euclidean coordinates. While in the first case the resulting measure is global space dependent, in the second one it depends on the supplied texture coordinates (refer to [Velho et al. 2001]). Both measures have their advantages and problems. The global one performs in any case but it is sensitive to the tessellation and the objects' dimension in the embedding space. The parametric one is independent of the latter and represents an intrinsic objects' measure, but it can lead to cracks if texture coordinates are not provided continuously.

Additionally to the local scale factor, in order to allow the user more control over the resulting surface, we supply an equalizer-like tool which provides the ability to influence each sub-band individually by hand. Using this tool more customized surface distortion can be achieved with a rather convenient method of control.

3.3.2 Preserving Volume

In order to preserve the objects' volume, the offset $d \in [0..1]$ can be linearly translated by $-\frac{1}{2}$ such that it affects the surface in both directions with respect to its orientation ($d \in \left[-\frac{1}{2}, \frac{1}{2}\right]$). Otherwise, the consecutively added shifts would lead to an expansion of the entire shape in the positive direction of the normals. Instead of just taking $-\frac{1}{2}$ it can be done more accurately by computing the mean value of the histograms for each input map separately and the shifts can be performed with respect to these. After that the zero level of a displacement map lies obviously in its gray values between $[0..1]$. Due to the linearity it can be rescaled as mentioned in the previous section.

3.3.3 Displacement Function

By taking the derived instruments into account the final offset D which is applied along the unit normal $\mathbf{n}$ of a vertex $\mathbf{v}$ can be defined as

$$D(\mathbf{v}^k) = C(k)\ L(\mathbf{v}^k)\ d(k, u, v)\ , \tag{2}$$

where k is the subdivision and an appropriate sub-band level and C is an user defined scaling constant for each level. Usually it is just set to $C = 1$. If particular sub-bands have been combined, the scale factor L has to be accumulated over these steps in order to recover it properly. Finally, the displacement is performed by

$$\hat{\mathbf{v}}^k = \mathbf{v}^k + D(\mathbf{v}^k)\ \mathbf{n}^k \tag{3}$$

on each vertex according to equation 1.

4 Implementation on a GPU

In todays computer graphics it has become a challenge to implement powerful algorithms for the rather limited programmable graphics hardware. For geometry processing it also means that essential properties like mesh connectivity have to be stored or calculated in different manners. This section describes the implementation of our hardware-subdivision and -displacement method.

4.1 Shader Architecture

In section 2.1.1 we mentioned the depth-first rendering approach (sliding window) which reduces the memory consumption of highly tessellated meshes. While our implementation differs in several ways from the mentioned one, in an essential issue it is based on the same idea.

4.1.1 Mesh Patches

In our work we have implemented a Catmull-Clark subdivision kernel. Thus, after the first subdivision step in software the resulting mesh can be decomposed in a series of equally structured patches defined by the quadrangles (In fact we apply two initial software steps in order to isolate extraordinary vertices. This is discussed in the next section). We take advantage of this constellation and encode each quadrangle into a small texture, where the vertex positions are stored in the RGB channels. A second texture holds vertex normals and the (u, v) coordinates are kept each in the alpha channels of both textures. Since there is no possibility to establish any connection between particular pieces in video memory, an overlapping neighborhood is included in order to maintain continuity

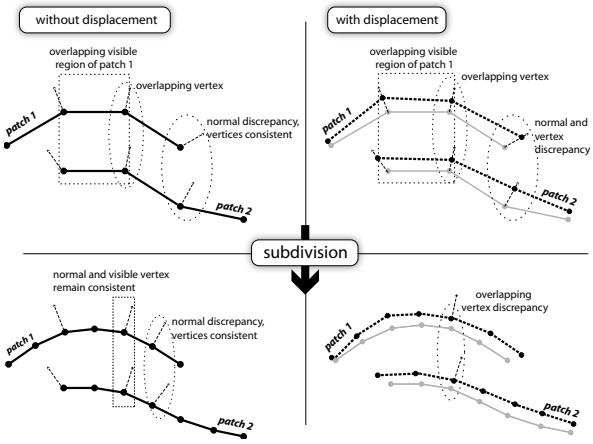


Figure 4: *Left: pure subdivision can be kept continuous with one vertex overlap. Right: if overlap-normals of the patches are not consistent, the displaced vertices would not be either.*

across patches. For subdivision only one ring of overlapping vertices is sufficient to ensure coherence, such that each quad is stored in a 4 by 4 texture.

The subdivision kernel is implemented in the pixel shader stage of the GPU and proceeds on a fixed set of textures located in the hardware memory. Each one is capable of storing the vertices of a particular subdivision level. This fix-sized buffer (the sliding window) is used to subdivide, displace and render basically unbounded large meshes, since mesh patches can be supplied sequentially. It is equivalent to successively moving the window along the initial mesh. It is filled with each patch and all desired operations are performed (including output to the frame buffer), before moving to the next one. This sequence has to be performed for all – in best case only for all visible – mesh patches in one frame.

In order to distinguish particular vertex types of the scheme (refer to [DeRose et al. 1998]) we supply a reusable lookup table with a chessboard-like pattern for each level. It allows to identify each vertex type in the regular patch such that the neighborhood can be determined and an appropriate subdivision rule can be chosen in a dynamic branch. The calculated results are rendered into the following texture in the sliding window which can be then either provided to the final rendering pipeline, or again reused as an input for the next subdivision step.

Since we use SM 3.0 extensions, the entire data flow can be closed in video memory without the usage of the CPU. This is possible because the entire texture can be reinterpreted directly as vertex data and rendered. In order to establish proper connectivity of the final triangles, we provide an index buffer which is precomputed for each level and always reused, since vertices in each patch are arranged in the same scheme. Moreover, overlapping regions can be excluded from the rendering in this way.

4.1.2 Extraordinary Vertices

Unfortunately there is no possibility to fit extraordinary vertices (refer to section 2.1.2) into the regular pattern. To overcome this we allow each patch to contain one of these at most. It is stored together with its 1-ring neighborhood in an additional 1D texture, where the adjacent vertices are arranged circularly around the vertex (Figure 5). Because the number of extraordinary vertices is constant after the first subdivision step and they can be isolated by one more step in software, the algorithm can be prepared to deal with this case.

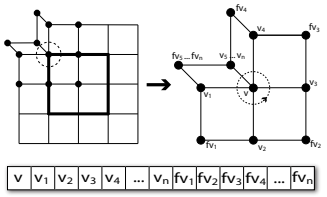


Figure 5: *Extraordinary vertex ordering scheme in a 1D texture.*

Thus each patch supplied to the hardware contains at most one irregular vertex and furthermore – by properly choosing the orientation of the patch – it is always located at exactly the same position. This fact can be used to encode that position into the lookup texture, such that during the processing of the fragments a hit on this position can be determined. Here the shader program jumps into another branch and recalls the proper position from the 1D texture. To compute this position an additional render pass has to be performed prior to the regular subdivision, such that the irregular vertex and its neighbors can be obtained from the 1D texture. Then they become overwritten on the regular patch.

Note that due to hardware resource constraints the extraordinary algorithm cannot be implemented in a generic program. This has been resolved by providing a software routine which generates specific shader programs for each particular valence. This routine is called once in the preprocessing stage. During the runtime these shader-programs are executed as additional passes and a context switch does not have to be performed.

4.1.3 Normals Estimation

The consistency between two overlapping patches is easily ensured for the Catmull-Clark subdivision by maintaining the 1-ring-overlap (Figure 4, left column). However, the introduction of displacements at each subdivision level leads to an additional problem on the visible boundary of a patch.

If every normal is computed in its local tangent frame the positions of all 1-ring neighboring vertices have to be maintained identically. However, the normal of the first invisible row of vertices of a patch differs from its counterpart in the neighbor-patch due to the lack of further information behind the first-ring. If displacement is performed, this discrepancy is also propagated to the first-ring vertices behind the visible patch border and thus it produces a gap between adjacent patches in the next subdivision step. The root of the problem is shown in Figure 4, right-hand side.

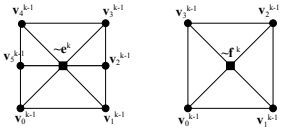


Figure 6: *Interpolation scheme for an edge-point $\tilde{\boldsymbol{e}}^k$ (left) and a face-point $\tilde{\boldsymbol{f}}^k$ (right) by points of the previous level $\boldsymbol{v}^{k-1}$.*

There are two possible remedies: it is possible to ensure consistency if the 2-ring neighborhood around each patch is considered. Since this represents a significant computational and above all a huge storage overhead, we resort to the second possible solution: to use estimated normals for displacement before the subdivision, such that they remain consistent across overlapping patches.

Of course, a rather good estimation of normals can be done only on

vertex-points of the current subdivision level because of the presence of the vertex and its 1-ring. If subdivision has not been performed, new face- and edge-vertices are not given yet. Our solution is to temporarily estimate the missing edge- and face-points in a rather rough way. Each particular point $\tilde{\mathbf{v}}^k$ (which will be introduced in the just after following subdivision step as $\mathbf{v}^k$) is bilinearly interpolated by its surrounding points at level $k-1$. Then its normal is computed as a normalized sum of the cross product normals of the faces given by the interpolated point $\tilde{\mathbf{v}}^k$ and its neighbors $\mathbf{v}_i^{k-1}$ (see Figure 6). In fact this approximates the normal on the previous level.

The displacement can be applied just after the subdivision, particularly after the new vertex position has been computed. Although this solution has an impact on the correctness of the displacement, in consideration of the density of the mesh it is negligibly small and no noticeable drawback can be observed on the visual results. Finally, this solution provides a further benefit: because the estimation can be done for each vertex without the knowledge of the properly placed neighbor points (since information from previous level is used), it can be performed just-in-time directly before the subdivision in the same shader program. This saves an expensive switch of render targets in order to compute proper normals, while exact limit normals can be computed after the last subdivision step and can be used for the illumination of the final scene.

4.1.4 Displacement Map Continuity

In order to perform displacement on overlapping patches, one final issue should be taken into account. While a continuous texture coordinates layer can be kept over adjacent pieces, texture tiles can cause holes between patches if the offset values d do not match each other at the borders. This problem can be resolved by providing exactly tileable textures as displacement maps only.

4.2 Adaptive Subdivision and Displacement

In the decomposed mesh each patch is independent, thus each can be subdivided and displaced to a certain level. This constellation implies the possibility of a step-wise adaptive subdivision of the whole model, such that adjacent patches can be rendered at different resolutions depending on the current camera position. One can profit from this issue and allow a one-step difference between neighboring patches without loss of the geometric coherence of the mesh. Vertices of mesh patches at different levels do not lie on the same positions, however the last positions at the higher sampled mesh are known exactly.

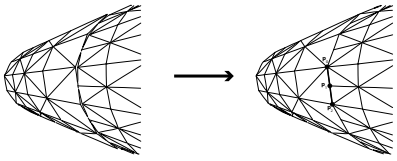


Figure 7: *Closing a T-Joint between mesh patches. Left: two patches of different resolution. Right: the vertices of the higher sampled mesh are forced to the lower level. Points $\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2$ form a zero area triangle.*

Thus, as a straight-forward solution, the border vertices between two differently sampled meshes can be forced to their positions at the lower subdivision level. This works quite satisfying for smooth subdivision, although small micro-holes of pixel-size can still appear during the rendering. The remedy therefore are zero area triangles [Losasso and Hoppe 2004]. The triangulation of the final

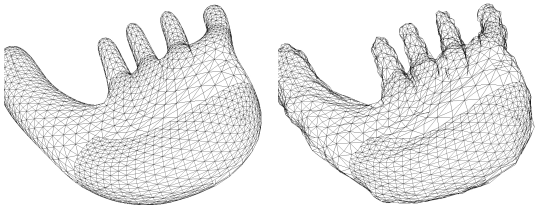


Figure 8: *Adaptive subdivision (left) with displacement (right).*

rendering does not depend on the actual connectivity of the mesh but rather is determined by a precomputed index buffer. Just for this reason, the T-faces can be introduced very easily by including them into the index-buffer computation routine. A further advantage of the zero-area-faces is the fact that indeed during displacement it is hardly possible to keep the borders of differently sampled patches consistent. Here the T-triangles do a great job by totally closing any arising discontinuity (see Figures 7 and 8).

The described procedure behaves very well on regular patch junctions, while it is difficult to implement in hardware for extraordinary vertices. Fortunately, the number of these is usually very limited in a mesh, thus our method does not allow different resolutions on adjacent patches sharing one extraordinary point in order to avoid those hard cases. In fact - this solution is numerically safe and provides a completely 'watertight' mesh.

4.3 Rendering

The final rendering routine is a cycling function containing two nested loops (Figure 9). In each frame the outer loop cycles over all patches where in an inner loop each one is subdivided (and displaced) depth-first to the desired resolution. Once a patch is fully computed, the resulting geometry-texture is reinterpreted as a vertex-buffer directly in the hardware and an one-for-all, off-line precomputed index-buffer is supplied. Here the rendering pipeline must perform a context switch to change the render target to the frame buffer and load another shader program for visual output. This is the actual bottleneck of the system. The frequent context switches of the shader cause the longest delays in the GPU pipeline, reducing the overall utilization of the graphics hardware. While not implemented in our solution yet, this drawback can be reduced by maintaining multiple subdivision buffers in video memory in order to cache and reuse results computed in the rendering stage.

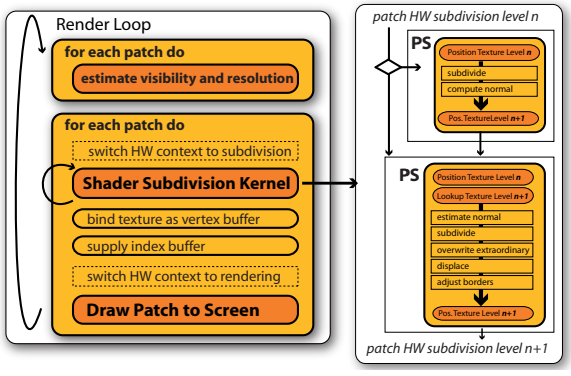
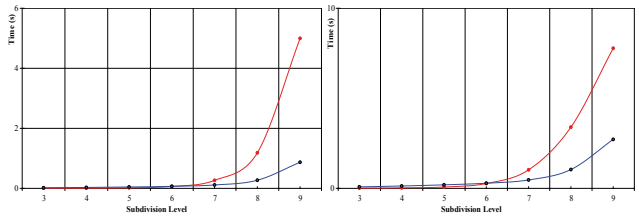


Figure 9: *Rendering. Left: outer loop. Right: inner loop. Note that the additional pixel shader pass is performed on demand if an extraordinary point has to be computed.*



level	3	4	5	6	7	8	9
SW (red)	0,003	0,006	0,017	0,064	0,267	1,187	4,999
HW (blue)	0,019	0,031	0,046	0,068	0,114	0,273	0,871
FPS (HW)	100	33	25	15	10	4	1,5
outp. vertices	386	1538	6146	24578	98306	393218	1572866

Figure 10: *Evaluation time of traditional software subdivision (red) and our method in hardware (blue). Note the logarithmic scale in the right-hand side chart. Time is given in seconds.*

Nonetheless, GPU subdivision results in an significant speedup if compared to traditional software implementation based on half-edge data structures, especially at higher steps (Figure 10). Depending on the hardware, multiple vertices are processed simultaneously by the GPU. Also, the fact that the computed data stored in the texture memory can be directly reinterpreted as geometry without passing it back to system memory allows to process all this in real-time.

5 Results

Figure 10 shows the comparison of the evaluation time of recursive subdivision performed on half-edge data structures in software and our hardware accelerated solution up to level 9. Note that the first two steps before hardware subdivision have been computed in software to isolate the extraordinary points and to achieve an appropriate size of mesh patches.

The test has been done on a Intel Core Duo 2.4 Ghz machine with 2GB main memory. The graphics was supported by an ATI Radeon X1900 GT GPU with 36 pixel pipelines, 513 Mhz core clock and 256 MB installed video memory, 768 MB texture memory and 660 Mhz memory clock. The model used for the test is a simple unit cube at original zero level. It is build out of 8 vertices and 6 faces. After two steps the mesh supplied to hardware is already composed of 96 faces. The presented timings are given in seconds and show an average over 80 measured frames.

In terms of the ratio of the compared runtimes can be seen that the complexity of the hardware does not follow the same increase as of the software. This appears obvious due to the parallelization of the GPU. On the third step (which is the first one in hardware) the GPU is in fact slower than software, while on the ninth step it is almost 6 times faster. The low subdivision steps are slower than the higher due to the overhead of the initialization and bad utilization of the pipeline. It is not designed to process very small textures, because the data and instructions transfer cause a relatively large overhead if compared to the contained information. Processing of larger textures would reduce this overhead. Currently we are working on an improved implementation which resolves this problem and also takes advantage of recently introduced hardware extensions.

6 Conclusions

We have presented a method for displacing subdivision surfaces on several resolutions by the means of spectrally decomposed patterns. In fact, to the authors knowledge, such an approach has not been attempted yet.

The multiscale displacement approach provides the ability to fine-tune the visual output on-the-fly in many variations by rather intuitive control. To compare this method to classical displacement, one issue should be considered as significant extension: Usually displacement is applied only once on the final tessellation, which limits this technique to a one-time shift of the surfaces' vertices. This does not allow the ability to achieve coarse deformations of the entire object with additional small features along the deformed surface. This issue has been extended by the adaption of multiresolution displacement such that a vertex from the initial mesh can be moved several times in different directions, each affected by the previous subdivision steps.

As an additional goal, the entire subdivision and displacement procedure has been shown to be able to perform in real-time on a programmable graphics accelerator with very high mesh tessellations. This work was very much concerned on the issue of real-time performance. More precisely, many limitations had to be taken into account in order to achieve such an implementation, especially the issue of keeping the borders of adjacent patches continuous during several displacement steps. Generally, our prototype achieves interactive frame rates with approx. 100.000 vertices (10 fps), which all become re-computed each frame from approx. 30 initial points on a moderate hardware. These rates might be significantly increased by the introduction of caching strategies.

In order to improve the visual quality additional extensions are conceivable:

- In the adaptive approach, the additional detail of finer subdivision levels could be introduced continuously, by scaling the displacements based on the viewer distance during the change from one subdivision level to the next.
- The borders that are forced to the coarser subdivision level in adaptive subdivision could be modified to a linear transition zone between differing subdivision levels.

The implementation can be seen as a starting point for further optimizations in order to ensure optimal performance on different hardware. Considering the current development of the graphics accelerators, we are working on an implementation which uses new extensions (geometry shader) in order improve our method. We believe that this technique is very well suited to be used for modeling of multiscale details even on large meshes.

Figure 11: *Tree bark rendered with texture applied.*

Acknowledgments

The authors would like to thank Anton Fuhrmann and Chrystoph Toll for their supporting hints and discussions. Most of this work has been done at the VRVis Research Center in Vienna, Austria, which is partially founded by the Austrian government research program Kplus. Parts of this work have also been funded by the Austrian Science Fund (FWF) under project P17260-N04.

References

BIERMANN, H., LEVIN, A., AND ZORIN, D. 2000. Piecewise smooth subdivision surfaces with normal control. In *ACM SIGGRAPH 2000: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 113–120.

BIERMANN, H., MARTIN, I. M., ZORIN, D., AND BERNARDINI, F. 2002. Sharp features on multiresolution subdivision surfaces. *Graph. Models 64*, 2, 61–77.

BISCHOFF, S., KOBBELT, L. P., AND SEIDEL, H.-P. 2000. Towards hardware implementation of loop subdivision. In *Proceedings of the 2000 SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware (EGGH-00)*, ACM Press, N. Y., S. N. Spencer, Ed., 41–50.

BLINN, J. F. 1978. Simulation of wrinkled surfaces. In *ACM SIGGRAPH 78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 286–292.

BOIER-MARTIN, I., AND ZORIN, D. 2004. Differentiable parameterization of catmull-clark subdivision surfaces. In *Eurographics Symposium on Geometry Processing*, Eurographics Association, Nice, France, R. Scopigno and D. Zorin, Eds., 159–168.

BOLZ, J., AND SCHRÖDER, P. 2002. Rapid evaluation of catmull-clark subdivision surfaces. In *Proceedings of the Web3D 2002 Symposium (WEB3D-02)*, ACM Press, New York, 11–18.

BOLZ, J., AND SCHRÖDER, P. 2003. Evaluation of subdivision surfaces on programmable graphics hardware. In *http://www.multires.caltech.edu/pubs/GPUSubD.pdf*.

BURT, P. J., AND ADELSON, E. H. 1983. The laplacian pyramid as a compact image code. *IEEE Trans. on Communications*, 4 (Apr.).

CATMULL, E., AND CLARK, J. 1978. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10* (Sept.), 350–355.

COOK, R. L. 1984. Shade trees. In *ACM SIGGRAPH 84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 223–231.

DEROSE, T., KASS, M., AND TRUONG, T. 1998. Subdivision surfaces in character animation. In *ACM SIGGRAPH 98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 85–94.

DOO, D., AND SABIN, M. 1978. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design 10* (Sept.), 356–360.

HALSTEAD, M., KASS, M., AND DEROSE, T. 1993. Efficient, fair interpolation using catmull-clark surfaces. In *ACM SIGGRAPH 93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 35–44.

KHODAKOVSKY, A., AND SCHRÖDER, P. 1999. Fine level feature editing for subdivision surfaces. In *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*, ACM Press, New York, NY, USA, 203–211.

LEE, A., MORETON, H., AND HOPPE, H. 2000. Displaced subdivision surfaces. In *ACM SIGGRAPH 2000: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 85–94.

LOSASSO, F., AND HOPPE, H. 2004. Geometry clipmaps: terrain rendering using nested regular grids. *ACM Trans. Graph. 23*, 3, 769–776.

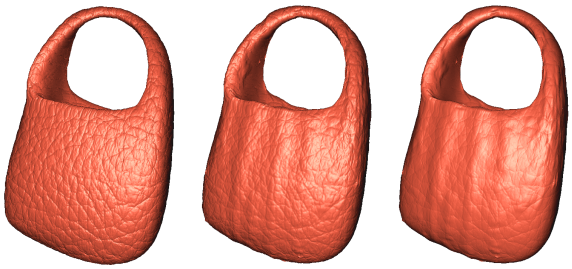


Figure 12: *A bag with calf leather and folds patterns.*

MÜLLER, K., AND HAVEMANN, S. 2000. Subdivision surface tesselation on the fly using a versatile mesh data structure. In *Proceedings of the 21th European Conference on Computer Graphics (EG-00)*, Blackwell Publishers, Cambridge, S. Coquillart and J. Duke, David, Eds., vol. 19, 3 of *Computer Graphics Forum*, 151–160.

OLIVEIRA, M. M., BISHOP, G., AND MCALLISTER, D. 2000. Relief texture mapping. In *ACM SIGGRAPH 2000: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 359–368.

PETERS, J. 2000. Patching catmull-clark meshes. In *ACM SIGGRAPH 2000: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 255–258.

PHARR, M., AND FERNANDO, R. 2005. *GPU Gems 2 : Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, March.

PULLI, K., AND SEGAL, M. 1996. Fast rendering of subdivision surfaces. In *ACM SIGGRAPH 96 Visual Proceedings.*, ACM Press, New York, NY, USA, 144.

REIF, U. 1995. A unified approach to subdivision algorithms near extraordinary vertices. *Computer Aided Geometric Design 12*, 2, 153–174.

SEDERBERG, T. W., ZHENG, J., SEWELL, D., AND SABIN, M. 1998. Non-uniform recursive subdivision surfaces. In *ACM SIGGRAPH 98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 387–394.

SETTGAST, V., MÜLLER, K., FÜNFZIG, C., AND FELLNER, D. W. 2004. Adaptive tesselation of subdivision surfaces. *Computers & Graphics 28*, 1, 73–78.

SHIUE, L.-J., JONES, I., AND PETERS, J. 2005. A realtime gpu subdivision kernel. In *ACM SIGGRAPH 2005 Proceedings*, ACM Press, New York, NY, USA, 1010–1015.

STAM, J. 1998. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *ACM SIGGRAPH 98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 395–404.

TOBLER, R. F., MAIERHOFER, S., AND WILKIE, A. 2002. A multiresolution mesh generation approach for procedural definition of complex geometry. In *SMI '02: Proceedings of the Shape Modeling International 2002 (SMI'02)*, IEEE Computer Society, Washington, DC, USA, 35.

VELHO, L., PERLIN, K., YING, L., AND BIERMANN, H. 2001. Procedural shape synthesis on subdivision surfaces. In *SIBGRAPI '01: Proceedings of the 14th Brazilian Symposium on Computer Graphics and Image Processing*, IEEE Computer Society, Washington, DC, USA, 146–153.

YING, L., AND ZORIN, D. 2001. Nonmanifold subdivision. In *IEEE Visualization 2001, October 24-26, 2001, San Diego, CA, USA, Proceedings*, IEEE Computer Society, T. Ertl, K. I. Joy, and A. Varshney, Eds.

ZORIN, D., AND KRISTJANSSON, D. 2002. Evaluation of piecewise smooth subdivision surfaces. *The Visual Computer 18*, 5-6, 299–315.