Eurographics Symposium on Rendering 2008
Steve Marschner and Michael Wimmer
(Guest Editors)

Volume 27 (2008), Number 4

Ptex: Per-Face Texture Mapping for Production Rendering

Brent Burley[†1] and Dylan Lacewell[‡1,2]

[1]Walt Disney Animation Studios
[2]University of Utah

Abstract
Explicit parameterization of subdivision surfaces for texture mapping adds significant cost and complexity to film production. Most parameterization methods currently in use require setup effort, and none are completely general. We propose a new texture mapping method for Catmull-Clark subdivision surfaces that requires no explicit parameterization. Our method, Ptex, stores a separate texture per quad face of the subdivision control mesh, along with a novel per-face adjacency map, in a single texture file per surface. Ptex uses the adjacency data to perform seamless anisotropic filtering of multi-resolution textures across surfaces of arbitrary topology. Just as importantly, Ptex requires no manual setup and scales to models of arbitrary mesh complexity and texture detail.
Ptex has been successfully used to texture all of the models in an animated theatrical short and is currently being applied to an entire animated feature. Ptex has eliminated UV assignment from our studio and significantly increased the efficiency of our pipeline.

1. Introduction

Texture mapping is the cornerstone of our studio's rendering pipeline. We use Catmull-Clark subdivision surfaces [CC78] for the vast majority of our models, and we often apply dozens of texture map layers on each surface. These maps define surface color and texture, coarse and fine displacements, specular intensity, roughness, anisotropic lighting direction, masks to blend between different looks across a surface, and many additional attributes.

The painterly look that art directors on our films typically demand can only be achieved with hand-painted textures. When we do use a procedural texture, it is often just a base coat for painting, and we always precompute it and store it in a texture map – we prefer the filter quality of texture maps because procedural textures are notoriously hard to anti-alias in the shader [AG99]. We also cache rendered attributes such as ambient occlusion in texture maps for rendering efficiency. Thus we require that all models be mappable in our pipeline.

Previous texture mapping methods for subdivision surfaces were inefficient, lacked generality, and often required painstaking manual setup. We tried various atlas and volumetric methods which promised to be more general and efficient, but could not get acceptable filtering results. Before subdivision surfaces, NURBS patches were our modeling standard and texture mapping was trivial due to their intrinsic 2D parameterization; we sought to bring the same simplicity of texturing to subdivision surfaces.

We desired a new method that was:

- **Film-quality.** The method must provide anisotropic filtering, no visible spatial or temporal aliasing, smooth filtering for displacement maps, and no visible seams.
- **General.** The method must work on any geometric model in our pipeline, regardless of topology or complexity.
- **Efficient.** The method must be cost-effective in storage, I/O, memory, and CPU usage.
- **Setup-free.** The method must allow models to be texture mapped without any special preparation.

Our new method, Ptex, meets these goals by mapping textures using the intrinsic per-face parameterization of the subdivision mesh. There is an elegance and efficiency in using the intrinsic parameterization, and this by itself is not a new idea [LMH00], but film-quality filtering of per-face textures has not been previously addressed.

A key benefit of using the intrinsic parameterization is

[†] e-mail: brent.burley@disney.com
[‡] e-mail: lacewell@cs.utah.edu

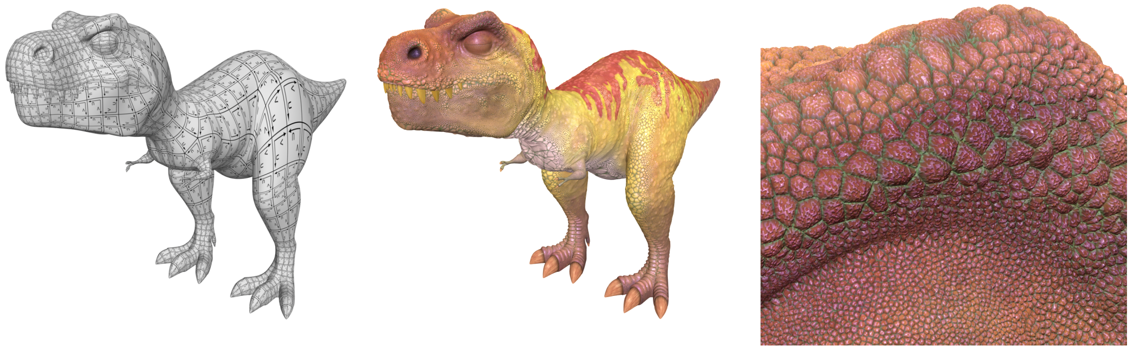

Figure 1: *T. Rex with 2694 faces rendered with Ptex. No explicit UV assignment was used. The largest texture layer, the fine-scale displacements, has 836 million texels stored in a single Ptex file with individual face resolutions ranging from* 64×64 *to* 2048×2048 *texels. No seams are visible across faces, even under close magnification. (© Walt Disney Animation Studios)*

the freedom from having to assign UVs before painting. Assigning UVs traditionally has been the responsibility of the Look Development department which creates the textures and shaders that define the look of the models. Because of tight production schedules, Look Development happens in parallel with other departments which means that many departments only have access to unparameterized models. And even if UVs were assigned when the models were created, the Look Development department may reparameterize a model at any time if the current UVs do not allow for adequate resolution, and this would invalidate all existing textures including those textures created in other departments. For these reasons, most departments have not previously had access to texture maps.

With Ptex, this is no longer the case as shown in Figure 2. Modelers can now sculpt displacement maps, layout artists can sketch in rough features, animators can create attribute maps to control deformations, etc., before a model reaches the Look Development department and without fear of the textures being later invalidated. This has also eliminated one of the more tedious tasks of texture mapping – creating an explicit parameterization – along with the overhead and hassle that comes with it such as passing the parameterization data down the pipeline, reparameterizing after model revisions, etc.

Adopting the intentionally trivial per-face parameterization was far from a trivial task. A custom file format, cache, and filtering engine needed to be built and integrated with a commercial renderer. We encountered and solved a number of research problems related to filtering across face boundaries, filtering with very large filter widths, and anisotropic filtering.

In this paper we present our original contributions: (1) data structures and methods for efficiently storing and accessing face adjacency data in the texture map; (2) a method for seamless, anisotropic filtering over the multi-resolution per-face textures using the stored adjacency information; and (3) a method for blending data across faces of an arbitrary mesh to handle filtering when the individual faces are subpixel in the final image. In addition, we offer details of our working implementation, provide an analysis of our filtering choices, and show production results.

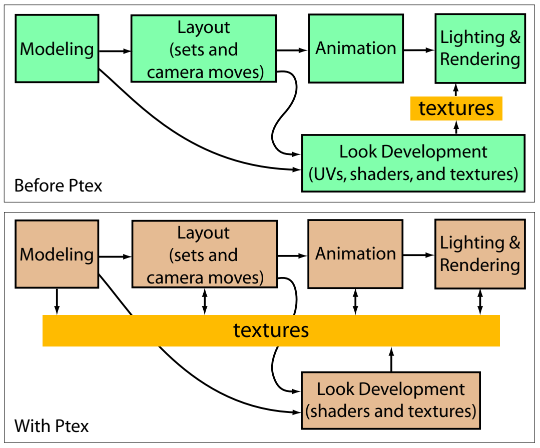
Modeling
Layout (sets and camera moves)
Animation
Lighting & Rendering
textures
Look Development (UVs, shaders, and textures)
Before Ptex
Modeling
Layout (sets and camera moves)
Animation
Lighting & Rendering
textures
Look Development (shaders and textures)
With Ptex

Figure 2: *Before Ptex, only departments downstream from Look Development could use textures. With Ptex, any department can use textures.*

2. Previous texture mapping methods

We evaluate previous methods according to our list of requirements above (Film Quality, General, Efficient, and Setup-free), with the results summarized in Table 3.

Pelting. A common method for simple objects is to cut the mesh to make it topologically disk-like and map it to a single connected region of the plane, often referred to as a

	Film-quality	General	Efficient	Setup-Free
Pelting	✓		✓	
Atlas methods	maybe	maybe	✓	
Projection	✓			✓
Volumetric textures		✓		✓
Prior per-face methods		maybe	✓	maybe
Ptex	✓	✓	✓	✓

Figure 3: *Evaluation of methods according to our requirements*

pelt. These mappings can be applied to a subdivision control mesh [DKT98], but seams and distortion are problems. Seam artifacts can be hidden by painting and blending between overlapping maps [PB00], or can be obscured by placing seams in areas of high curvature or occlusion [SH02], but neither of these generalize to arbitrarily complex meshes. Distortion, caused by flattening curved surfaces to the plane, can be reduced by making more cuts, and can be minimized using a solver. Nonlinear solvers can produce optimal results [HG00]; linear solvers are faster [DMA02], but have higher distortion because they only optimize area locally, and can still be slow when solving an $N \times N$ linear system for N vertices (we have seen meshes with 400,000 vertices in production).

Atlas methods. Some methods generalize planar mapping to models of higher complexity by splitting the mesh into multiple disjoint pieces, or *charts*, which map to separate regions of the plane [MYV93,LPRM02], and pack those pieces into a shared texture referred to as an *atlas*. The mapping step is easier for atlases because charts have relatively few faces and low Gaussian curvature. However, packing efficiently can be difficult. Also, charts that are adjacent in the atlas may not be adjacent on the model which creates filtering challenges. Purnomo et al. [PCK04] hide seams by storing a small gutter around each chart at every mipmap level, and adjust the texture coordinates so that a trilinear filter never spans a gutter, but trilinear filtering is not sufficient quality for production.

Projection. Projection mapping [AG99] does not require assigned UVs; texture values are looked up with a projection onto a plane or other parametric surface. Projection mapping is in fact the primary texture mapping method at some studios. A depth map is used to prevent texture values from projecting onto occluded geometry. Complex geometry can be problematic though, requiring multiple projections for coverage, possibly with manual blending between the projections, and there can be depth bias artifacts. Texture lookups are also expensive, requiring multiple perspective matrix transforms plus depth map and color map lookups per shading point.

Volumetric textures. Texture values can be stored in a volumetric data structure such as an octree or a brick map [BD02, DGPR02, CB04]. Volumetric textures do not require assigned texture coordinates and thus do not have any setup costs. However, storage and lookup are less efficient than for 2d textures. There are also filtering issues because filtering takes place in 3d rather than on a surface. Colors can bleed through nearby surfaces with similar normals, so such surfaces must be assigned separate maps. Only trilinear filtering has been described in the literature and adjacent voxels with differing resolutions also present filtering challenges.

Prior per-face methods. Some methods store textures per-face on a simplified mesh. Lee et al. [LMH00] simplify a Loop subdivision surface to one of equivalent topology, and use per-face displacement maps to recover the original surface. They also store arbitrary attributes such as color in the per-face textures and use the Loop subdivision rules to interpolate the values. PolyCube maps [THCM04] store textures on the faces of a set of connected cubes which roughly approximate the base mesh, but are not practical for complex meshes. TileTrees [LD07] store textures on the faces of cubes organized in an octree and support more general geometry than PolyCubes. None of these methods however provide production-quality filtering; notably, no anisotropic filtering is provided.

In summary, only two of these methods, pelting and projection, meet our filtering requirements, but neither of these is general, and projection is particularly inefficient. Prior to Ptex, most of our models were broken into small pieces that could be separately mapped using standard methods, but this came at significant cost in complexity and system resources.

3. Overview

Ptex stores a separate texture per face of the subdivision control mesh, each of which can be independently sized, and any surface can be mapped with a single Ptex file containing all of the per-face textures regardless of mesh complexity. The Ptex file also contains mesh adjacency data that is used to filter across faces. Seamless filtering is particularly critical to avoid lighting artifacts when using displacement maps.

In the following sections, we describe our per-face parameterization and how to compactly store adjacency data in the texture file; we describe how to size and store per-face textures efficiently; we present our method for filtering per-face textures in a production renderer; we provide production results; and we discuss our implementation choices.

4. Per-face parameterization

The input to our algorithm is an arbitrary quad mesh which is completely and uniquely parameterized using the coordinate $\langle faceid, u, v\rangle$ where *faceid* is implied by the position of each face in the mesh description, u and v are the intrinsic subd parameters, and the UV orientation is defined by the vertex

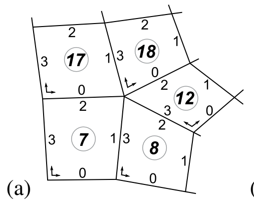

Figure 4: ***a)*** *Portion of a control mesh showing intrinsic faceids and edgeids.* ***b)*** *Corresponding limit surface showing continuous isolines across faces.*

ordering (assumed to be $\langle 0,0\rangle, \langle 1,0\rangle, \langle 1,1\rangle, \langle 0,1\rangle$). While *faceid* is obviously discontinuous, the parameterization is still C^2 continuous across faces on the Catmull-Clark limit surface once orientation has been taken into account. For example, in the mesh in Figure 4, $\partial\mathbf{P}/\partial u$ on face 12 aligns with $\partial\mathbf{P}/\partial v$ on face 8.

The four *edgeids* of each quad face are numbered 0 to 3 and follow the vertex ordering. Adjacency data needed for filtering is stored as an array of the four adjacent faceids and edgeids per face. A faceid of -1 indicates that the edge is on a mesh boundary. For instance, face 7 shown in Figure 4 would have adjacency data {adjfaces:$\langle -1,8,17,-1\rangle$; adjedges:$\langle x,3,0,x\rangle$, $x =$*don't care*}.

To account for the difference in orientation between neighboring faces, the number of 90 degree rotation steps from face i to face j can be computed from the edgeids for the shared edge (assuming consistent winding order) as $r = (e_i - e_j + 2) \bmod 4$ and the valence of a vertex can be determined by simple traversal.

With 32 bits allocated to each faceid and 2 bits allocated to each edgeid, the storage cost of the adjacency data is only 17 bytes per face. This data is further compressed in the file along with all of the other header data.

5. Texture sizing and storage

Even though no explicit parameterization step is needed, the textures still need to be sized efficiently as shown in Figure 5. We sample the surface derivatives over each face f to find the minimum texel density in u and v (which occurs at the maximal point of the surface derivative) and compute per-face scale factors $s_u(f) = 1/(\max\partial\mathbf{P}/\partial u)$ and $s_v(f) = 1/(\max\partial\mathbf{P}/\partial v)$. We are careful to avoid sampling near the singularity at extraordinary vertices. The scale factors are computed whenever a model is loaded into our 3d paint system. Users can then adjust the global texel density ρ (given in texels per object-space unit) and each face will be allocated a texture with resolution $\rho s_u(f) \times \rho s_v(f)$ increased to the nearest power of two. Additionally, users have full local control of the resolution of each face, a capability not present in most texture mapping systems. For instance, users can increase the resolution on just those faces for which the camera is expected to zoom in close.

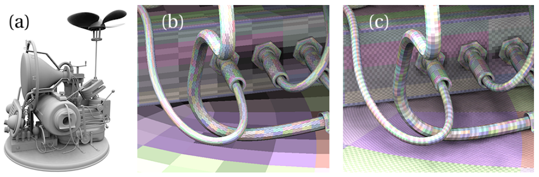

Figure 5: *Mesh with 212,356 faces mapped with a single Ptex file.* ***a)*** *Ambient occlusion map.* ***b)*** *Detail image showing individual faces color-coded and mapped with 4×4 texels per face; solid-colored regions are texels.* ***c)*** *Face resolutions allocated based on surface derivatives, showing much better coverage using the same 3.4 million texel budget.*

As in most texture systems, we store box-filtered symmetric reductions of each texture, i.e. mipmaps, in the Ptex file. However, rather than store a separate image pyramid for each texture, we store full sets of face textures in *resolution layers*. The first layer has the full-res textures, the next layer has all the once-reduced textures, and so on. Each texture within a layer is still stored in its own directly-accessible block. The resolution layers merely serve to improve locality of disk access.

We also use asymmetric reductions for anisotropic filtering; without them, the texture footprint of the filter could grow arbitrarily large in one direction in texture space and significantly degrade performance. Other systems store asymmetric reductions in the texture file [Pea90], but this is typically not used in practice due to the high storage cost; a full set of asymmetric reductions increases file size by 300% versus $33\frac{1}{3}$% for symmetric reductions. To avoid the storage and I/O cost, Ptex generates asymmetric reductions dynamically from the nearest available resolution, a feature we have not seen in other systems.

In addition to the mipmaps, we store a single data block containing the average values of the textures, one value per face. For faces which are constant, a common occurrence in many production textures, this is the only storage required (i.e. no mipmaps are needed). Also, for distant objects that project to a small region of the image plane, this is the only data block that needs to be read from the file for any face, constant or not, which provides a significant reduction in file I/O. Further, if all faces intersecting the filter kernel are constant then no filtering calculation is needed.

6. Filtering

In this section we present our method for anisotropic multiresolution filtering of per-face textures. We support both magnification and minification in a unified framework. Our method is designed to work well with Pixar's Photorealis-

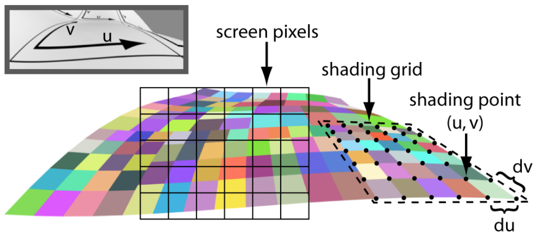

Figure 6: *Quad subdivision face diced along u and v isoparametric lines into grids with roughly pixel-sized micropolygons. The micropolygon size, du × dv, is constant over each grid.*

tic Renderman (PRMan), but should be adaptable to any production-quality renderer.

6.1. General case

In a REYES renderer [CCC87] such as PRMan, texture maps are sampled at *shading points* as shown in Figure 6. The renderer *dices* geometric primitives into rectangular *shading grids*, and communicates the grid spacing to the shader via the built-in variables du and dv. Because Catmull-Clark surfaces are diced face-by-face along parametric lines, per-face textures are aligned with the shading grids and the filter region is simply a rectangle of size $du \times dv$. This allows fully anisotropic filtering to be achieved with a separable filter. Any filter kernel could be used with our method, but separable filters are more computationally efficient than radial or elliptical filters [Hec89].

To achieve our filter quality requirements, we use a bicubic filter kernel:

$$k(x) = \frac{1}{6}\begin{cases} (3+6S)|x|^3 + & \text{if } |x| < 1 \\ (-6-9S)|x|^2 + (4+2S), & \\ (-1-2S)|x|^3 + (6+9S)|x|^2 + & \text{if } 1 \le |x| < 2 \\ (-12-12S)|x| + (8+4S) & \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The parameter S is exposed to the user as a sharpness control. For displacement maps, we use $S = 0$ which produces the C^2 cubic B-spline. For color maps, the B-spline appears blurry; a sharper value is needed. The value $S = 1$ (our production default for color maps) produces the much sharper Catmull-Rom interpolating spline, but a continuum of values from 0 to 1 may be used; the value of $S = 2/3$ for instance produces the popular Mitchell filter [MN88]. Kernels with $S \neq 0$ are C^1 which is sufficient for most maps.

For a given filter region of size $du \times dv$, we choose the

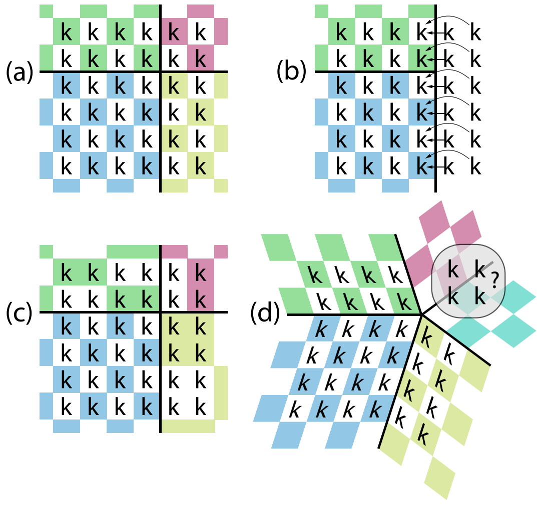

Figure 7: *A 6 × 6 kernel overlapping a corner of the face.* ***a)*** *In the regular case, the kernel is applied piecewise to each overlapped face for a seamless result.* ***b)*** *Beyond the mesh boundary, kernel weights are applied to the adjacent edge texels.* ***c)*** *When an adjacent face has insufficient resolution, each kernel weight is applied to the nearest available texel.* ***d)*** *Near an extraordinary vertex, it is less clear how to apply the rectangular kernel.*

texture resolution, $R_u \times R_v$, that has texels just smaller than the filter size:

$$R_u = 2^{\lceil \log_2 \frac{1}{du} \rceil}; \quad R_v = 2^{\lceil \log_2 \frac{1}{dv} \rceil} \quad (2)$$

The kernel weights are:

$$k_{ij}(u,v) = k\left(\frac{u-(i-\frac{1}{2})/R_u}{du}\right) k\left(\frac{v-(j-\frac{1}{2})/R_v}{dv}\right) \quad (3)$$

and the filter convolution with texel data values d_{ij} is:

$$f(u,v) = \frac{\sum_i \sum_j k_{ij}(u,v) d_{ij}}{\sum_i \sum_j k_{ij}(u,v)} \quad (4)$$

We clamp the resolution against the highest available resolution for the current face's texture. The filter width thus varies between 1.0 and 2.0 texels and the bicubic kernel (which has a support width of 4) will require between 4 and 8 samples in each direction. When the filter kernel is clamped to the highest resolution, it acts as a magnifying filter and is equivalent to cubic spline interpolation.

When the kernel extends beyond an edge of the face, we split the kernel and convolve it piecewise with each overlapped face as shown in Figure 7a. If the kernel overlaps

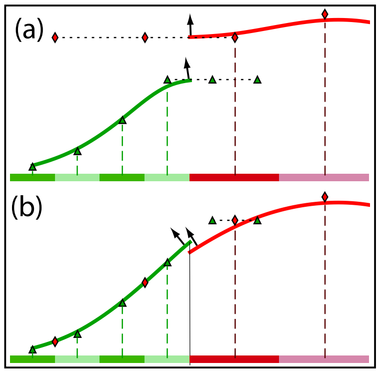

Figure 8: *Two adjacent faces displaced with textures of different resolution, shown in 1d cross-section.* ***a)*** *Border clamping produces a discontinuity, and also bends the reconstructed normals toward the undisplaced surface.* ***b)*** *Filtering across edges reduces the discontinuity, but more importantly, unbends the normals.*

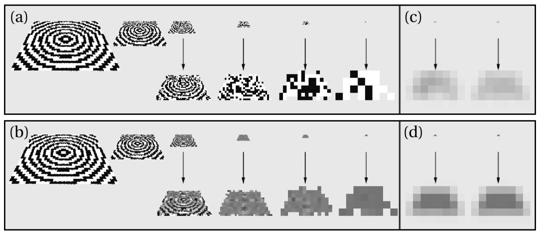

Figure 9: *Mesh with* 32×32 *constant colored faces rendered at resolutions ranging from* 256×256 *to* 8×8 *pixels without face blending (a) and with face blending (b). To reveal the aliasing, the pixel filter was turned off by using a box filter with 1 pixel sample. For comparison, the lowest-res images from (a) and (b) were re-rendered in (c) and (d) using a Gaussian pixel filter with* 5×5 *and* 9×9 *samples. At* 5×5 *samples, a common production setting, aliasing is reduced but still visible (c, left). At* 9×9*, the spatial aliasing is gone, but a bias is still visible that would likely result in flickering during animation (c, right).*

a boundary, we apply the kernel weights from beyond the boundary to the adjacent edge texels, shown in 7b. This is commonly referred to as *border clamping*.

If a neighboring face has insufficient resolution, or if the kernel overlaps an extraordinary vertex, i.e. of valence $N \neq 4$, then we use special methods described in the following subsections.

6.2. Resolution mismatch

When an adjacent face has insufficient texture resolution for the current filter size, the texels do not line up with the kernel weights. In this case we apply each kernel weight to the nearest available texel as shown in Figure 7c. Border clamping could be used (as is typically done with NURBS patches), but when used with displacement mapping, this causes the displaced normals to bend towards the undisplaced surface, as illustrated in Figure 8a, and can introduce significant lighting artifacts. Filtering across the edge produces more consistent normals as shown in Figure 8b, and typically produces a smaller discontinuity. The discontinuity itself, though, is not a problem, because PRMan will automatically stitch the grids back together by adding micropolygons to fill the gap.

It is worth noting that the resolution mismatch only occurs under magnification – a rare occurrence in production because artists always try to create textures with more than enough resolution – and only causes visible artifacts with large, smooth displacements. For a modest amount of magnification, the grid spacing will be close to the texel spacing and the method described here is sufficient to hide any possible seam artifacts; the discontinuity will in fact be no larger than that caused by transitions between prefilter resolutions, an issue further discussed in Section 8. If high magnification is important, artifacts can be completely avoided simply by creating all of the per-face textures with the same resolution.

6.3. Extraordinary vertices

The method of splitting the kernel into as many as four pieces, as described above, works well near regular vertices. It is less clear how to extend the rectangular kernel across an extraordinary vertex (EV) such as the one in Figure 7d.

To form a smooth reconstruction near EVs, a Catmull-Clark surface could be formed from texels in a manner similar to [LMH00]. In addition to the computational cost and complexity, the biggest problem with this approach is that it does not support anisotropic minification. Fortunately, in almost all practical cases, the Catmull-Clark reconstruction is unnecessary.

Under minification or slight magnification, the area around the EV is subpixel and smooth reconstruction is not important. We simply ignore the corner faces and renormalize the kernel. This creates discontinuities along edges near the EV but produces negligible artifacts. Under significant magnification of large, smooth displacements, seam artifacts may be visible. In this case the Catmull-Clark reconstruction can still be used, but this has not yet been necessary in practice.

6.4. Large filter widths

When a face is rendered smaller than a pixel, PRMan will only sample the corners of each face and the filter widths, *du* and *dv*, will increase beyond 1.0 (i.e. larger than the face).

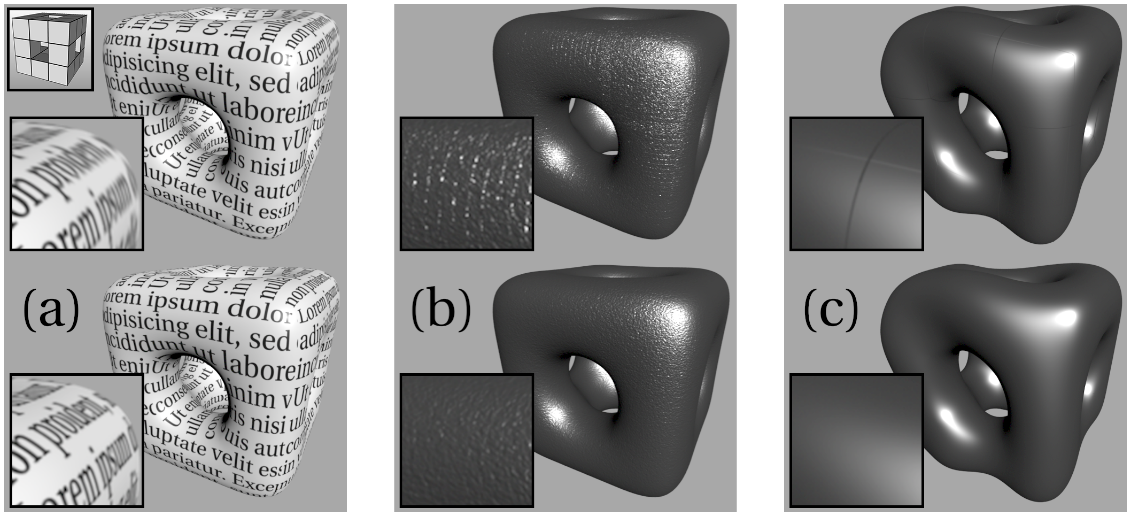

Figure 10: *Surface rendered with color and displacement maps under minification using control mesh shown in upper left. The top images show artifacts that can occur with other methods: (a) isotropic filtering results in undesirable blur; (b) bicubic filtering using $S = 1$ kernel shows aliasing with fine displacements (bilinear is much worse); (c) clamping kernel at face boundaries reveals seams with large, smooth displacements. The bottom images show (a) our anisotropic filtering, (b) $S = 0$ kernel, and (c) seamless filtering across edges.*

Further, when many faces are subpixel, the renderer may undersample the surface (because only 1 micropolygon per pixel sample is used) and aliasing can occur.

To handle filter widths larger than 1.0, a special prefiltering method is used where the 1×1 texel per-face textures are repeatedly blended by box-filtering with their immediate neighbors. Convolving a box filter n times generates a B-spline of order n and has an effective filter width proportional to $sqrt(n)$ [Hec89]. A 3 unit box filter convolved n times has an effective filter width of approximately $\frac{3}{2}sqrt(n)$. The processing cost is modest as only the 1×1 per-face textures are used, and the filtered values can be computed once and cached. Results are shown in Figure 9.

7. Results

Figure 10 demonstrates that our method meets our stated filter quality requirements. Our anisotropic filtering produces high-quality minification of color and displacement maps and has good preservation of detail. Our $S = 0$ kernel filters displacement maps without visible aliasing. And filtering across face boundaries produces seamless results even for large, smooth displacements, and even in the presence of several EVs of valence 3 and 5.

Our latest feature film, "Bolt," is about 60% complete through Look Development and has thus far painted more than $80,000$ subdivision surfaces with Ptex, using an average of 7 texture layers per surface, and 2.3 mega-texels per Ptex file. The largest Ptex file has more than 3.3 giga-texels, a resolution unachievable with traditional texture files. A final production frame is shown in Figure 12. A theatrical short, "Glago's Guest," has also been completed using Ptex to texture all of the models, several of which are shown in Figure 13.

Production efficiency gains have been significant. The removal of UV processing has streamlined our pipeline and improved artist efficiency, and there have been I/O performance gains as well. Our current production's upgraded render farm has 4 times the rendering capacity of the previous production, and texture usage per render has increased substantially. Despite the increased demand, Ptex has actually decreased the load on the texture I/O servers, which were running near capacity on the previous production, and allowed us to avoid an expensive upgrade.

We attribute the reduction in I/O demand largely to the fact that Ptex only requires one file per surface, versus hundreds with previous methods, but the other optimizations we have described likely contribute as well. CPU times are also competitive with and often faster than our previous texture methods.

Feedback from artists has been extremely positive. Artists report that models that would have been very difficult to paint with previous methods are easy with Ptex, and levels of visual complexity are possible that were previously unattainable.

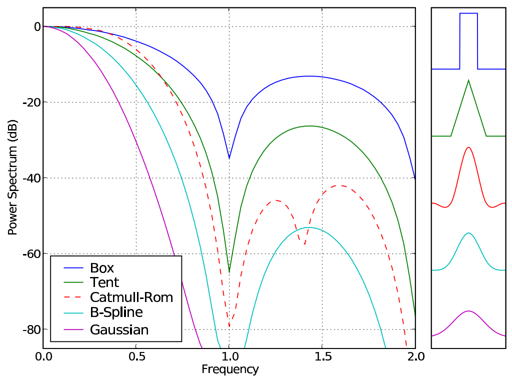

Figure 11: *Frequency response of various filters.*

8. Discussion

What about non-quad meshes? We already had a quad-only restriction in our pipeline and do not consider it a significant modeling burden. That said, we believe extending Ptex to support embedded non-quads should be straightforward; PRMan already subdivides non-quads into quads before dicing into grids, so for non-quad faces we would just assign one texture per sub-face. We would need to extend our adjacency structure and filtering algorithms to handle the introduced “T” intersections between the subfaces of the non-quads and adjacent quads, but the basic ideas should still apply. In the future we would also like to extend Ptex to triangle meshes for Loop subdivision.

Should the filter area be based on the screen pixels rather than the shading grid? A circular image of a pixel has an elliptical *pre-image* in texture space and thus the ideal filter region is generally considered to be an arbitrary ellipse [Hec89], but this cannot be determined from the undisplaced surface and thus is not applicable to displacement mapping. In PRMan, it is common practice to use the pre-image of the micropolygon (rather than the final pixel) as the filter region [AG99]. If a subpixel shading rate is used and undersampling by the pixel filter is a concern, then aliasing can still be reduced by simply scaling the filter size by $(1/\sqrt{shadingRate})$ to keep the filter area approximately pixel-sized.

Why not use a higher-order filter for prefiltering and simple bilinear interpolation for reconstruction (as in [PCK04])? The box filter has the least attenuation of the non-sharpening filters typically used in practice as shown in Figure 11. Gaussian, by comparison, attenuates significantly more detail near the filter frequency and is undesirably soft. The Gaussian filter must also be properly windowed and given adequate support or it will have significant aliasing in addition to appearing blurry. A sharpening filter such as Catmull-Rom may have a better low-pass response than box, but could create ringing artifacts that get magnified during reconstruction. Spatial aliasing can occur with a box filter, but is minimized because the filter is always texel-aligned as a $4:1$ or $2:1$ reduction. Note that temporal aliasing, which is far more noticeable than spatial aliasing, is not an issue because the prefilter is temporally invariant.

For reconstruction, we have shown in Figure 10 that even using a bicubic filter can produce visible aliasing for displacement maps. A more primitive filter like bilinear performs significantly worse, and simply coupling it with a higher-order prefilter would not eliminate the aliasing.

What are the implications of allowing the filter width to vary continuously vs. using a fixed 4×4 kernel as is normally used with bicubic splines? A fixed 4×4 kernel provides a continuous reconstruction within a mipmap level, but when the filter switches resolutions there can be a visible discontinuity. Linear interpolation, or "lerping", between the two nearest mipmap levels is commonly used to eliminate the discontinuity, but the price paid is two-fold. First, detail is lost because the lower-res mipmap level is blended into the result. Second, lerping increases the filter cost, especially when used with anisotropic prefiltering where lerping must be done in both the u and v directions – 64 kernel samples are required, 16 each from 4 separate texture resolutions.

In contrast, with our variable-width kernel and anisotropic prefiltering, the mipmap discontinuity is reduced to the point of being unnoticeable, and lerping is not needed. The cost is also reasonable: on average, 30.25 kernel samples are needed, and only a single texture resolution is required for any given shading point. Like the fixed kernel, our variable-width kernel provides a continuous reconstruction within a mipmap level, given that $k(x)$ is continuous and the filter width varies continuously. The fact that the grid spacing changes abruptly from grid to grid is not a problem thanks to PRMan’s *smooth derivatives* feature; PRMan filters the du and dv values to make them continuous across the surface before passing them to the shader.

There is one theoretical quality advantage to the fixed-width kernel; Mitchell [MN88] recognized that for separable cubic filters, *sample-frequency ripple* (noise with a period equal to the sample spacing) can be reduced to zero by using a kernel with a fixed width of four. When the kernel width is allowed to vary, the kernel must be normalized to keep the volume equal to 1.0 and achieve a good *flat-field response* (the ability to reconstruct a constant function), and normalization introduces ripple.

For our variable-width kernel, we tried normalizing our bicubic filter kernel using the analytic volume integral, but this resulted in significant sample-frequency ripple and poor flat-field response for non-integral filter widths. Instead, we found that normalizing the kernel weights discretely as shown in Equation 4 drastically reduced the sample-frequency ripple and achieved perfect flat-field response.

The ripple that remains is proportional to the data slope. We calculated a peak error of 0.43% of the data slope at a filter width of 1.61 for our $S = 0$ kernel (the ripple is somewhat larger for other values of S). This is visibly insignificant and smaller than the ripple in PRMan's smoothest native filter, "radial-bspline".

9. Conclusion

We have presented a novel texture mapping method for Catmull-Clark subdivision surfaces using the intrinsic per-face parameterization of the control mesh. We have shown how to efficiently store and access face adjacency data in the texture map and use that data to seamlessly filter across faces.

We have demonstrated that our method, Ptex, meets our stated requirements:

- Ptex is *film-quality*, providing seamless, anisotropic filtering of texture maps including smooth displacement maps.
- Ptex is *general*, scaling to models of arbitrary complexity and textures of arbitrary resolution, allowing levels of detail that were previously unattainable. By keeping the texture grid aligned with the shading grid, Ptex preserves detail better than previous methods.
- Ptex is *efficient*, providing independent per-face control over texture resolution, and requiring only one file per surface, reducing memory and I/O requirements significantly.
- Ptex is *setup-free*, requiring no UV assignment or other pre-processing, and thus allowing access to texture maps in every department of our studio.

We analyzed our filtering choices, gave details of our working implementation, and showed that our approach results in significant efficiency gains for production. Ptex has been fully and successfully deployed across our entire feature animation studio.

10. Acknowledgments

At Disney we would like to thank Chuck Tappan, for motivating and championing this work; Dan Teece, for adding Ptex support to our 3d paint system; and Joe Marks, Andy Hendrickson, and the Bolt and Glago crews for their enthusiastic support. Additionally, we would like to thank Solomon Boulos, Dave Edwards, and Pete Shirley for their thoughtful criticism of earlier versions of this paper.

References

[AG99] Apodaca A. A., Gritz L.: *Advanced RenderMan: Creating CGI for Motion Pictures*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. 1, 3, 8

[BD02] Benson D., Davis J.: Octree textures. In *ACM SIGGRAPH 2002* (2002), pp. 785–790. 3

[CB04] Christensen P. H., Batali D.: An irradiance atlas for global illumination in complex production scenes. In *Eurographics Symposium on Rendering 2004* (2004), pp. 133–142. 3

[CC78] Catmull E., Clark J.: Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design 10*, 6 (1978), 350–355. 1

[CCC87] Cook R. L., Carpenter L., Catmull E.: The REYES image rendering architecture. In *ACM SIGGRAPH 87* (1987), pp. 95–102. 5

[DGPR02] DeBry D., Gibbs J., Petty D. D., Robins N.: Painting and rendering textures on unparameterized models. In *ACM SIGGRAPH 2002* (2002), pp. 763–768. 3

[DKT98] DeRose T., Kass M., Truong T.: Subdivision surfaces in character animation. In *ACM SIGGRAPH 1998* (1998), pp. 85–94. 3

[DMA02] Desbrun M., Meyer M., Alliez P.: Intrinsic parameterizations of surface meshes. In *Eurographics 2002* (2002), pp. 209–218. 3

[Hec89] Heckbert P. S.: *Fundamentals of Texture Mapping and Image Warping*. Master's thesis, U.C. Berkeley, 1989. 5, 7, 8

[HG00] Hormann K., Greiner G.: MIPS: An efficient global parametrization method. In *Curve and Surface Design: Saint-Malo 1999*, Laurent P.-J., Sablonnière P., Schumaker L. L., (Eds.). Vanderbilt University Press, Nashville, TN, 2000, pp. 153–162. 3

[LD07] Lefebvre S., Dachsbacher C.: Tiletrees. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2007), ACM, pp. 25–31. 3

[LMH00] Lee A., Moreton H., Hoppe H.: Displaced subdivision surfaces. In *ACM SIGGRAPH 2000* (2000), pp. 85–94. 1, 3, 6

[LPRM02] Lévy B., Petitjean S., Ray N., Maillot J.: Least squares conformal maps for automatic texture atlas generation. In *ACM SIGGRAPH 2002* (2002), pp. 362–371. 3

[MN88] Mitchell D. P., Netravali A. N.: Reconstruction filters in computer-graphics. In *ACM SIGGRAPH 1988* (1988), pp. 221–228. 5, 8

[MYV93] Maillot J., Yahia H., Verroust A.: Interactive texture mapping. In *ACM SIGGRAPH 1993* (1993), pp. 27–34. 3

[PB00] Piponi D., Borshukov G.: Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *ACM SIGGRAPH 2000* (2000), pp. 471–478. 3

[PCK04] Purnomo B., Cohen J. D., Kumar S.: Seamless texture atlases. In *Symposium on Geometry processing 2004* (2004), pp. 65–74. 3, 8

[Pea90] Peachey D.: *Texture on demand*. Tech. Rep. 217, Pixar, San Rafael, CA, USA, 1990. 4

[SH02] Sheffer A., Hart J. C.: Seamster: inconspicuous low-distortion texture seam layout. In *IEEE Visualization 2002* (2002), pp. 291–298. 3

[THCM04] Tarini M., Hormann K., Cignoni P., Montani C.: Polycube-maps. In *ACM SIGGRAPH 2004* (2004), pp. 853–860. 3

[Wil83] Williams L.: Pyramidal parametrics. In *ACM SIGGRAPH 1983* (1983), pp. 1–11.

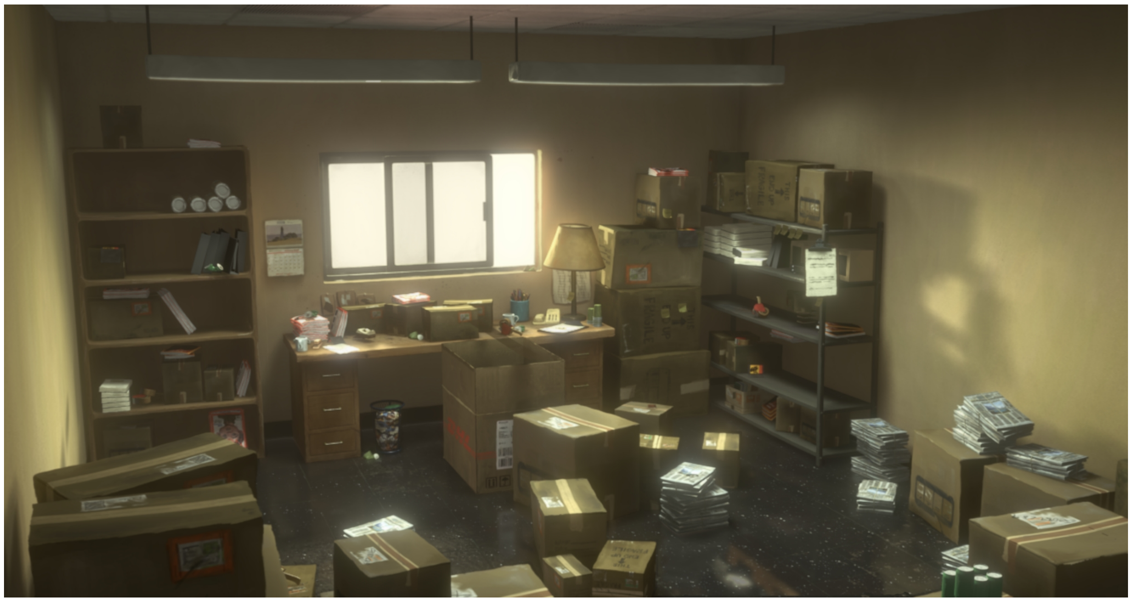

Figure 12: *A final production still from "Bolt" using Ptex for all models. (© Walt Disney Animation Studios)*

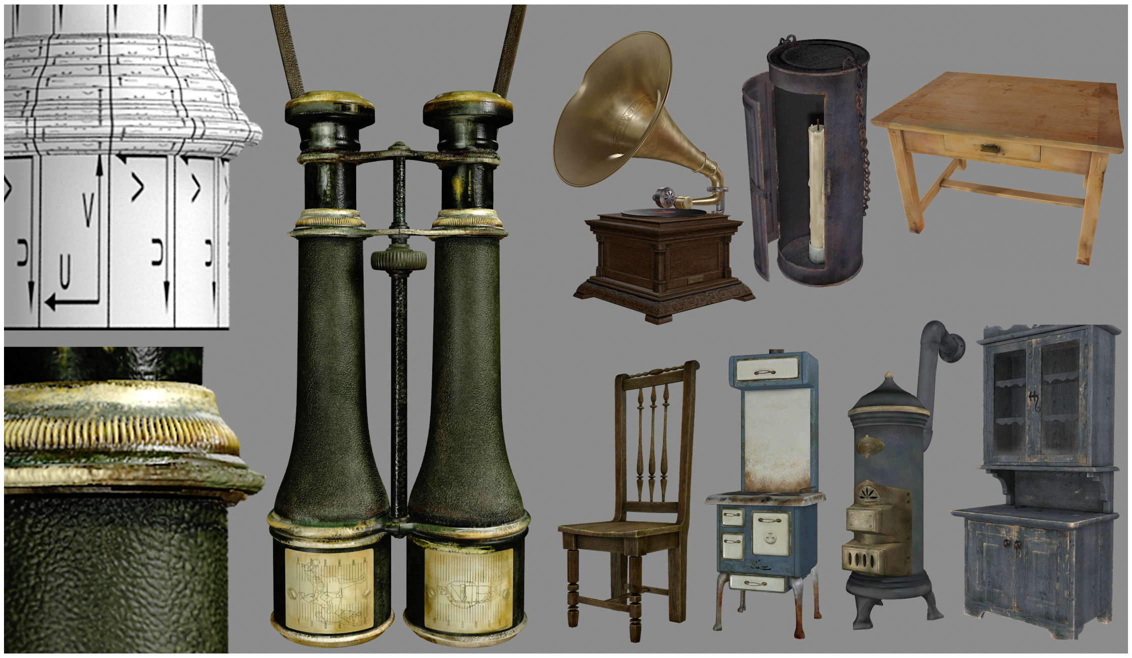

Figure 13: *Production models from "Glago's Guest," painted and rendered with Ptex. Full-res turntables included in accompanying video. (© Walt Disney Animation Studios)*