Vision, Modeling, and Visualization (2010)

ZIPMAPS: Zoom-Into-Parts Texture Maps

Martin Eisemann[1] and Marcus Magnor[2]

[1]Computer Graphics Lab, TU Braunschweig, Germany

Abstract

In this paper, we propose a method for rendering highly detailed close-up views of arbitrary textured surfaces. Our hierarchical texture representation can easily be rendered in real-time, enabling zooming into specific texture regions to almost arbitrary magnification. To augment the texture map locally with high-resolution information, we describe how to automatically, seamlessly merge unregistered images of different scales. Our method is useful wherever close-up renderings of specific regions shall be provided, without the need for excessively large texture maps.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.3]: Picture/Image Generation—Computer Graphics [I.3.6]: Methodology and Techniques—Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Computer Graphics [I.4.3]: Enhancement—

1. Introduction

In most interactive graphics applications, the scale at which some 3D object may be rendered during runtime is a-priorly unknown. For small-scale depictions, well-known mipmaps [Wil83] avoid aliasing artifacts caused by *texture minification*. On the other hand, if a textured 3D object ought to be displayed at a scale larger than the available texture map resolution, detail-deprived, washed-out renderings are the result due to simple interpolation techniques. We address this problem of texture magnification.

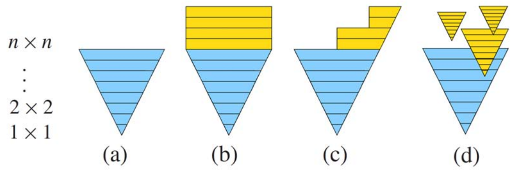

Figure 1: *Comparison between (a) standard mipmapping – specific texture information is only provided up to a specific level; (b) clipmaps – texture information is loaded on demand; (c) multiresolution textures – a quadtree structure represents texture information at different levels; (d) our zipmaps – a sparse representation to texture specific details at high resolution.*

Zoom-into-parts texture maps (zipmaps) enable rendering detailed close-up views of specific texture regions. In contrast to recent approaches like Gigapixel images [KUDC07] or clipmaps [TMJ98], we don't use a complete high-resolution texture map; instead high-resolution texture insets are seamlessly merged into low-resolution textures. We show how zipmaps are almost as simple to use and render as standard texture mapping.

As particular contributions our paper presents:

- a new hierarchical texture mapping scheme, called zipmaps, which naturally supports enhanced magnification of specific regions.
- a fast rendering algorithm for zipmaps, which enables applying zipmaps to arbitrary meshes in a single rendering pass.

The remainder of this paper is organized as follows. After reviewing relevant related work in Section 2 we introduce our new zipmap textures in Section 3 and show how they are applied and efficiently rendered. In Section 4 we exemplarily show how to create zipmap textures. Section 5 presents our results in detail before we discuss limitations and conclude in Section 6.

DOI: 10.2312/PE/VMV/VMV10/291-297

2. Related Work

Texture mapping was introduced in computer graphics as early as 1974 as a very effective way to increase visual rendering complexity without the need to increase geometric detail [Cat74]. To overcome the aliasing problems apparent when the texel-to-pixel ratio exceeds unity, also known as minification, Williams introduced the mipmap representation [Wil83], a pre-calculated image pyramid at different resolutions of the texture. Advanced variations, like ripmaps [McR98] or fipmaps [BF02], solve this problem with even higher quality, but at the cost of higher memory requirements or slower rendering. Other possibilities are summed area tables or elliptical weighted average filters. A classic survey of texture mapping can be found in [Hec86].

While the problem of texture minification is well solved, the problem of texture magnification, i.e. if the view zooms into a part of a texture, is still a very active area of research.

Interpolation: The standard and most simple approach, which is still used in most computer games due to its simplicity, is to linearily interpolate the color values between neighbouring texels. Using a nearest-neighbour approach results in blocky artefacts, while linear interpolation gives blurry results.

Super-resolution: There are probably hundreds of papers dealing with the problem of super-resolution, i.e. how to increase texture or image resolution beyond the resolution provided (in the following we will use pixels and texels interchangeably to denote single image elements). Most of these approaches are based on exemplar-images or learning-based methods which derive images statistics from either the image itself or a database of images [HJO*01, SnZTyS03, HC04, YWHM08]. Other successful approaches make use of edge and gradient information or combine these with learning-based methods [FJP02, DHX*07, Fat07, SXS08]. Despite good quality at moderate magnification of the images, super-resolution approaches are usually far from real-time capable and are not applicable to high magnification factors.

Texture Synthesis approaches create larger texture maps from one or more small exemplar patches. One well-known approach is the image quilting technique by Efros and Freeman [EF01], in which a new image is synthesized by stitching together small patches of existing images. Kwatra *et al.* built upon this approach by using a graph cut technique to determine the optimal patch region to be used for synthesis [KSE*03]. Constrained texture synthesis tries to guide the texture creation process. The usual approach is to take neighbour information of a pixel into account and minimize some cost function which varies from approach to approach [LH05, Ram07].

For faster generation, tile-based approaches can be used. While the creation of periodic texture tiles is relatively simple, the periodicity can be annoyingly apparent for certain textures. Wang tiling can be used to allay this effect by creating patches, called Wang Tiles, which can be arranged together to non-periodically tile the plane [CSHD03, Wei04].

All these approaches only synthesize textures at a specific scale, i.e. features are usually not enlarged or shrunk in any way. In contrast Ismert *et al.* [IBG03] add detail to undersampled regions in an image-based rendering setup if more detailed versions of the same texture are available in the image. Wang and Mueller present an approach where a low resolution image guides the texture creation process for the higher resolution details [WM04]. Only recently Han *et al.* have presented an approach that uses patches at different scales for the synthesis process [HRRG08].

The problem with any of these texure synthesis approaches is that they are only suitable for textures with relatively similar repeating structures (though non-periodically arranged). The addition of specific details at specific positions is not possible. Lefebvre *et al.* presented an interactive approach to add small texture elements, called texture sprites, onto an arbitrary surface [LHN05]. While their implementation is very memory efficient and allows for various artistic effects it is less suited for rendering realistic details into an existing texture, e.g. merging two photographs. Eisemann *et al.* [EESM10] presented an interesting approach to fill this gap. They compute a dependency graph for an unordered image collection and seamlessly merge the input images at different scales hallucinating details for regions not covered by any input image.

Vector Textures: Texture maps are usually represented as a collection of discrete image elements and are therefore always limited in representable spatial frequency. Instead of using samples taken from the underlying texture function, vector textures represent the function using resolution independent primitives. Tumblin and Choudhury save sharp boundary conditions at discrete positions for every texel to prevent some of the strong blurring apparent in usual texture magnification [TC04]. Sen uses silhouette maps to maintain sharp edges in the texture while blurring at smooth transitions [Sen04]. A complete support for all primitives of a SVG description in a vector texture was presented by Qin *et al.* [QMK08], building on their own previous work in [QMK06]. Recently Jeschke *et al.* [JCW09] showed how to render surface details using diffusion curves onto arbitrary meshes.

The drawback of vector textures is that they can only preserve the low and very high frequency components, while mid-frequencies and new details are not present in a close-up view. This can give vector textures a quite cartoony and unnatural look.

Large Textures: The most straight forward idea for providing detail in textures is to simply use large enough textures which are dynamically loaded on demand. But usually hardware as well as bandwidth is limited, restricting textures to be of a certain maximum size. Tanner *et al.* address this problem by introducing clipmaps [TMJ98]. In this approach the necessary data at the best matching resolution is loaded on demand depending on the viewers position. This approach works particularly well for mapping height fields [Hüt98, Los04], needed e.g. in geographic information systems (GIS). Another work in this direction are the Gigapixel images presented by Kopf *et al.* [KUDC07]. A separate thread fetches the texture tiles of the Gigapixel images needed for rendering.

In all these approaches only scenes are considered where the needed data is in direct relation to the current viewpoint, which makes texture prefetching possible because the needed data does not change abruptly. However, this is not always the case in general texture mapping applications.

Multiresolution and Compressed Textures: Multiresolution and multiscale textures represent textures by using a hierarchichal representation. They most resemble our work presented in this paper. In the early days hierarchical texture representations were mostly used for multiresolution paint programs [BBS94, PV95] where wavelet or bandpass representations are used in a quadtree representation created on demand. Finkelstein *et al.* use binary trees of quadtrees to encode multiresolution video [FJS96]. Related to our work is the approach by Ofek *et al.* [OSW97, OSRW97] and Mayer *et al.* [MBB*01], who create a quadtree texture from a series of photographs. However quadtree structures might not be the best representation for texture maps, as, depending on the implementation, it may take up to $\log(n)$ texture look ups, plus filtering might become more difficult, as neighbouring texels might not be available. In contrast our approach can make use of the inbuilt hardware texture filtering of the GPU. Kraus and Ertl divide an already given high-resolution image (or 3D or even 4D volume) into a regular grid of fixed sized blocks [KE02]. The information residing in these blocks is resampled into a common texture map, reducing the size of blocks with only little information. The grid then serves as an indirection table into the actual data during rendering. Using the same texture for all patches may however result in problems when applying mipmapping to the texture. Lefebvre and Hoppe use a compressed adaptive tree structure which allows for fast random access on current graphics hardware while achieving large reduction in memory requirements [LH07]. The input however, is again a given high-resolution image.

To overcome the need of explicit parameterization Benson and Davis introduce octree textures [BD02]. Using an octree instead of a quadtree allows for encoding the spatial relationship directly in the position in the octree. It also overcomes the problem of wasted texture space usually encountered in classic 2D texture atlases [gDGPR02, LBJS07].

3. Zipmap Rendering

Zipmap textures can be thought of as a sparse sample representation of a large mipmap with almost arbitrary resolution, where only higher details for interesting parts of the texture are saved in separate texture patches and are drawn on top of each other during rendering (see Figure 1). Up to a specific level n the whole texture pyramid is saved in a base level mipmap texture, called the *root*. This way standard minification methods can be used to prevent aliasing in cases where the texels projected into image space are smaller than a single pixel. To incorporate details for specific regions during magnification, additional texture pyramids, called children, are added at specific positions, if needed in a recursive manner. Note that the base levels of these additional texture pyramids do not necessarily need to be at the highest level of the lower resolution image pyramid. This is beneficial for more efficient rendering or if the detail samples have been acquired at different time steps or from different viewpoints, as the affected portions of the parent patch are hidden behind the detail patches, as we will see later in Section 5.

The following is a description of the complete algorithm for rendering zipmaps onto arbitrary meshes. An overview of the complete process is also given in Figure 2. For rendering, the root and children are reassembled into a collection of ordered texture patches. Each one is associated with a specific texture matrix M_i which transforms texture coordinates from the root to the i-th child patch for lookup. Essentially, a zipmap texture is a simple collection of texture patches which are rendered in a specific order to texture an arbitrary surface. Patches containing the coarse overall information are rendered first, while child patches containing details are drawn later, on top of their parents.

Rendering: During rendering the color values C_i from all patches are acquired by multiplying the current texture coordinate provided by the application with the texture matrices of every patch separately. This transforms the texture coordinate from the root patchs coordinate system into the child coordinate system. A simple texture lookup fetches the corresponding color for the needed output pixel. In order to prevent drawing child patches if the calculated texture coordinates are outside the $[0 \dots 1]$ range we can make use of hardware texture clamping. The most efficient way to do this, is to do the multiplication in the vertex shader and pass the interpolated texture coordinates to the fragment shader. We then compute the final color value of the rgba-quadruple C by combining all texel rgba-values using the following simple formula:

$$C = \sum_i w_i C_i \quad , where \qquad (1)$$
$$w_i = \alpha_i \prod_{j>i} (1 - \alpha_j),$$

i.e. we simply mix the color value C_i of a patch with the already computed color according to the alpha channel of the patch. So in most cases a new patch is simply drawn over the old one, as most parts of the texture patches are opaque.

We can render up to 30 patches on a NVidia GeForce 8800 GTX, using GLSL, in a single render pass with this tech-

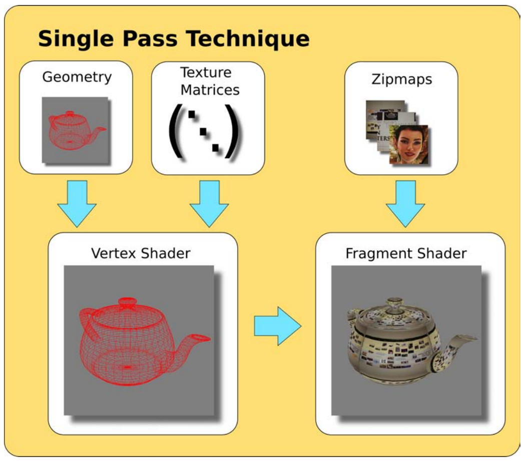

Figure 2: *Complete overview of the rendering technique using zipmaps. Applying zipmaps is almost as simple as plain texture mapping. The incoming texture coordinates are multiplied with the zipmap texture matrices and can then be used in the fragment shader for texturing.*

nique, because 60 floats assigned to varying variables is the limit. If a zipmap consists of more patches, we use a slight variant of this strategy. In a first pass, the first 30 patches are drawn and written to the framebuffer as described before. Using multiple render targets, we also render the current texture coordinates of the root patch into the red and green channel of another buffer B_{tc} which is initialized to zero beforehand, and set the alpha value to one, to mark affected fragments. In the next pass, we bind the next texture patches to the texture units plus the buffer containing the texture coordinates. Now instead of rendering the whole textured mesh again, we simply draw a screen filling quad and calculate the texture coordinates of the children in the fragmentshader by making use of B_{tc}. If its alpha value is zero, we discard the fragment, keeping the old color value. Otherwise we multiply every M_i with the queried texture coordinate from B_{tc} to calculate the correct texture coordinate for the i-th patch and color the output fragment as usual. We can repeat this process until every texture patch has been processed. If a detail is repeatedly used at different positions, we simply use different texture matrices M_i for this texture. This method is especially efficient for complex textured geometry or in scenes with much occlusion.

Blending Patches: Current graphics hardware poses another problem whenever texture patches are drawn on top of each other. If texture values close to a patch boundary are queried, hardware interpolation will not always be able to query the correct texture value, which will create a seamless blending with the background, even if exactly the same colors are used. This is due to the employed hardware interpolation methods for border conditions which causes visible seams (Figure 3 left). We solve this problem by setting the alpha-channel at the border of zipmap patches to zero (Figure 3 right). We do this for every level of the mipmap pyramids during the zipmap generation process, Section 4. Another advantage of this approach is that patches becoming smaller than one pixel in the output image simply disappear and do not produce small pixel artefacts that would otherwise be visible.

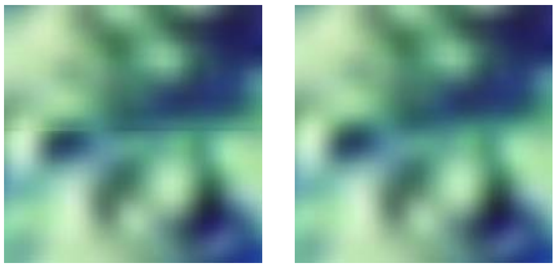

Figure 3: *Left: Close-up view with artefacts at patch borders (horizontal line in image middle). These appear even if the actual texel values are the same for the patch and the background. Right: Setting the alpha value to zero at patch boundaries removes seams.*

4. Zipmap Generation

One way to create zipmaps is of course by hand by an artist, who arranges the input patches to his convenience. The necessary transformation matrices are then computed and we adopt the gradient-based blending technique of Eisemann *et al.* [EESM10] to merge the images seamlessly. This step is especially necessary if there is a large scale difference between the overlapping patches. We therefore establish a gradient map I_i^g for each color channel of the patch that is to be drawn on top of another patch:

$$I_i^g = ||\nabla I_i||_1 = |I_{i_x}| + |I_{i_y}|, \tag{2}$$

where I_x and I_y are the gradients in x and y direction respectively. The gradient-density map I_i^{gdm} is then created from I_i^g by searching for each pixel the path with smallest cost derived from the sum of the according pixels in I_i^g to one of the border pixels. The final blending mask is then computed using a combined thresholding and scaling:

$$\alpha = \min(1.0, \frac{I_{max}^{gdm}}{\tau}), \tag{3}$$

where I_{max}^{gdm} is the maximum of the three color channels for which the gradient-density map was computed. This blends the patch nicely with most backgrounds. Of course, if results would be unsatisfying an artist could simply change the blending mask by hand if needed.

Eisemann *et al.* [EESM10] also present a nice way to establish the relationships between images in an unordered photo collection, which we adopt to our needs, to create zipmaps from real-world footage. We first compute the hierarchical relationship between images taken at different zoom levels of the same object from which we want to extract the texture. We do this by matching pairwise SIFT features and estimating a homography to warp the images towards each other. From these pairwise matchings, we can derive a dependency graph depicting the hierarchical relation between the images. From the dependency graph we can extract our needed transformation matrix for each patch. The colors are adjusted by solving a poisson equation with Dirichlet boundary conditions following [PGB03]. The boundary conditions are given by a one-pixel border derived from the parent image warped into the child image domain for each patch, while the guidance field v for the poisson equation is given by the gradient of the child image. The final blending is performed in the same way as described before. For more details we refer the interested reader to [EESM10].

5. Results and Discussion

Zipmaps can be easily rendered in real-time, since only a single matrix multiplication and one texture lookup per patch are required. The memory requirements are in direct accordance to the number and size of the input images used. No additional information than the patches and their texture matrices (offset and scaling) need to be saved. Since the child patches are saved in relation to the root patch, the application programmer only has to define texture coordinates for the root patch, just as he would do with a conventional 2D texture, making the zipmaps very easy to use in practice.

As test data, we have taken input images with a hand-held camera. We cannot point out exact scaling differences between the input images. However, we could robustly estimate the homographies for an approximate scaling factor of up to 12 (e.g. in the poster scene in Figure 4). Figures 4 to 6 show results of zipmap rendering.

On the top left, the input patches are shown. On the right the zipmap texture is applied to different geometries, and some close-up views from different viewpoints and different distances are shown. The output screen resolution was always set to 1024×1024 pixels, so magnification is present in most views. Our zipmap textures can be easily applied to any kind of geometry. In Figure 4 we use a four patch zipmap to texture a teapot. In Figure 5 and 6 we apply a zipmap consisting of four patches, six patches respectively, to a simple quad for illustration purposes. On the right, some close-up views are shown. Zooming onto single droplets or the knot-hole is now possible.For more examples see the accompanying video.

A typical approach in the games industry is to render detail textures as textured detail geometry. While performing an optimal amount of per-pixel work this approach has the drawback of z-fighting, if the detailed geometry is too close to the original or visible seams if the border handling is not done correctly or the viewpoint gets too close. To prevent these effects the geometry is usually cut into several non-overlapping pieces, which is time-consuming and requires a lot of manual work.

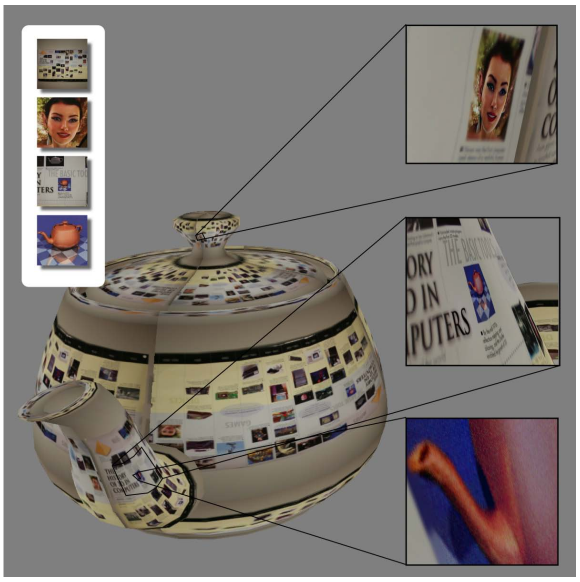

Figure 4: *Zipmap textures can be easily applied to any geometry just like conventional textures.*

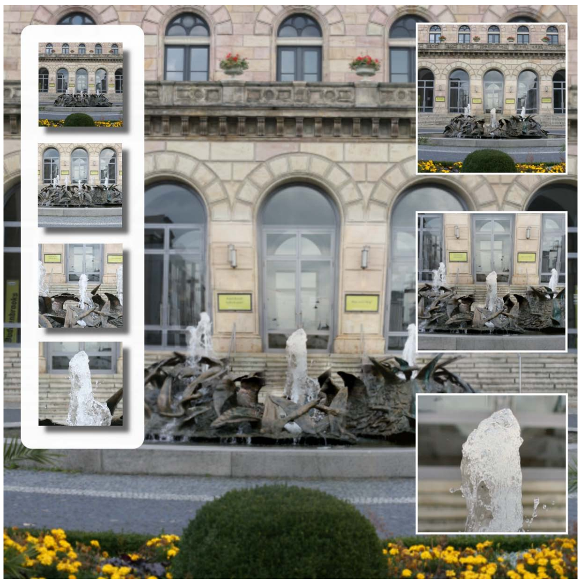

Figure 5: *Zipmap of a facade with fountain. Time-varying parts of the scene are merged into a common representation.*

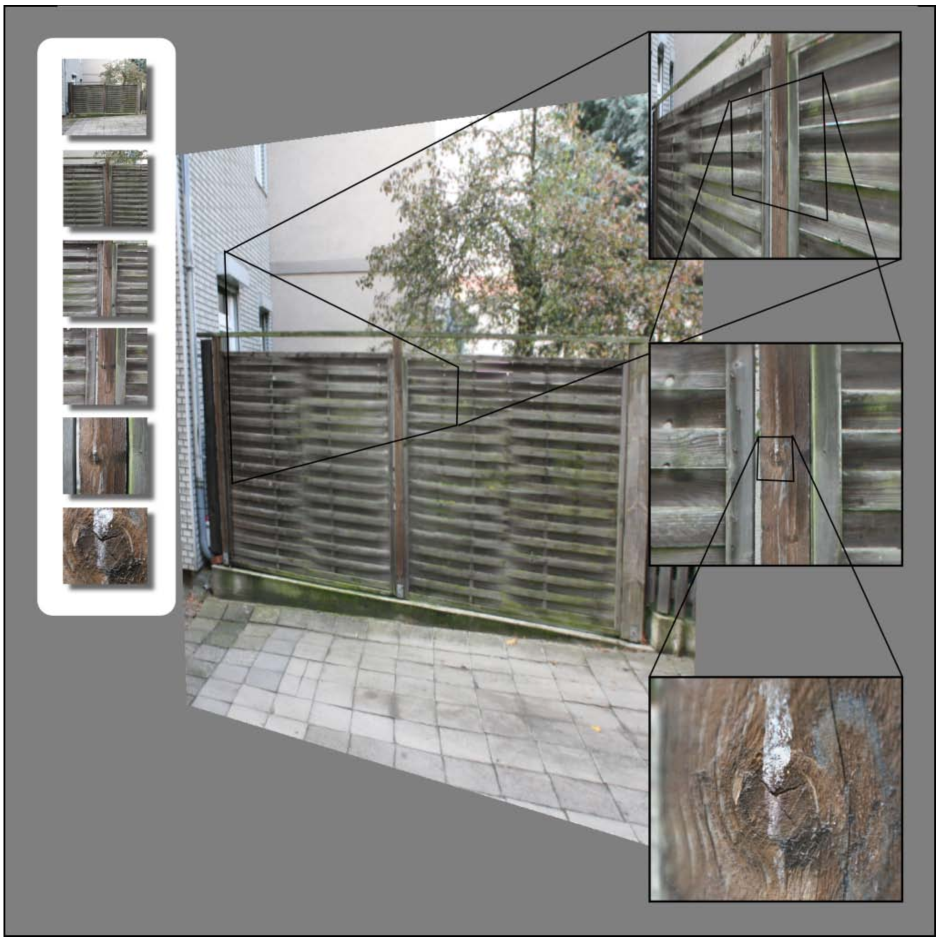

Figure 6: *A zipmap texture acquired from six photographs and applied to a simple quad.*

6. Conclusions and Future Work

We have introduced the new concept of zipmaps, a method for rendering detailed close-up views of textured surfaces. Zipmaps are easy to use and efficient to render and can be used with arbitrary images and kinds of textures, also normal or displacement maps would be possible.

For future work we are investigating animated zipmaps for video applications. Finally applying zipmaps to image-based rendering techniques like the Unwrap Mosaics [RAKRF08] will open up other new intriguing possibilities.

References

[BBS94] BERMAN D. F., BARTELL J. T., SALESIN D. H.: Multiresolution painting and compositing. *Proc. SIGGRAPH '94 13*, 3 (1994), 85–90.

[BD02] BENSON D., DAVIS J.: Octree textures. *Proc. SIGGRAPH '02 21*, 3 (2002), 785–790.

[BF02] BORNIK A., FERKO A.: Texture minification using quad-trees and fipmaps. In *Eurographics 2002 Short Presentations* (2002), pp. 263–272.

[Cat74] CATMULL E.: *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, Departement of Computer Sciences, University of Utah, 1974.

[CSHD03] COHEN M. F., SHADE J., HILLER S., DEUSSEN O.: Wang tiles for image and texture generation. *Proc. SIGGRAPH '03 22*, 3 (2003), 287–294.

[DHX*07] DAI S., HAN M., XU W., WU Y., GONG Y.: Soft edge smoothness prior for alpha channel super resolution. In *CVPR '07: Proc. of the 2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2007), pp. 1–8.

[EESM10] EISEMANN M., EISEMANN E., SEIDEL H.-P., MAGNOR M.: Photo zoom: High resolution from unordered image collections. In *GI '10: Proceedings of Graphics Interface 2010* (2010), Canadian Information Processing Society, pp. 71–78.

[EF01] EFROS A. A., FREEMAN W. T.: Image quilting for texture synthesis and transfer. *Proc. SIGGRAPH '01 20*, 3 (2001), 341–346.

[Fat07] FATTAL R.: Image upsampling via imposed edge statistics. *Proc. SIGGRAPH '07 26*, 3 (2007), 95.

[FJP02] FREEMAN W. T., JONES T. R., PASZTOR E. C.: Example-based super-resolution. *IEEE Comput. Graph. Appl. 22*, 2 (2002), 56–65.

[FJS96] FINKELSTEIN A., JACOBS C. E., SALESIN D. H.: Multiresolution video. *Proc. SIGGRAPH '96 15*, 3 (1996), 281–290.

[gDGPR02] (GRUE) DEBRY D., GIBBS J., PETTY D. D., ROBINS N.: Painting and rendering textures on unparameterized models. *Proc. SIGGRAPH 21*, 3 (2002), 763–768.

[HC04] H. CHANG D.-Y. YEUNG Y. X.: Super-resolution through neighbor embedding. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'04)* (2004), vol. 1, pp. 275–282.

[Hec86] HECKBERT P. S.: Survey of Texture Mapping. *IEEE Comput. Graph. Appl. 6*, 11 (1986), 56–67.

[HJO*01] HERTZMANN A., JACOBS C. E., OLIVER N., CURLESS B., SALESIN D. H.: Image analogies. *Proc. SIGGRAPH '01 20*, 3 (2001), 327–340.

[HRRG08] HAN C., RISSER E., RAMAMOORTHI R., GRINSPUN E.: Multiscale texture synthesis. *Proc. SIGGRAPH '08 27*, 3 (2008), 1–8.

[Hüt98] HÜTTNER T.: High resolution textures. In *Visualization '98* (1998), pp. 13–17.

[IBG03] ISMERT R. M., BALA K., GREENBERG D. P.: Detail synthesis for image-based texturing. In *I3D '03: Proc. of the 2003 symposium on Interactive 3D graphics* (2003), ACM, pp. 171–175.

[JCW09] JESCHKE S., CLINE D., WONKA P.: Rendering surface details with diffusion curves. *Transaction on Graphics (Siggraph Asia 2009) 28*, 5 (12 2009), 1–8.

[KE02] KRAUS M., ERTL T.: Adaptive texture maps. In *HWWS '02: Proc. of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2002), Eurographics Association, pp. 7–15.

[KSE*03] KWATRA V., SCHÖDL A., ESSA I., TURK G., BOBICK A.: Graphcut textures: image and video synthesis using graph cuts. *Proc. SIGGRPAH '03 22*, 3 (2003), 277–286.

[KUDC07] KOPF J., UYTTENDAELE M., DEUSSEN O., COHEN M. F.: Capturing and viewing gigapixel images. *Proc. SIGGRPAH '07 26*, 3 (2007), 93.

[LBJS07] LACOSTE J., BOUBEKEUR T., JOBARD B., SCHLICK C.: Appearance preserving octree-textures. In *GRAPHITE '07: Proc. of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia* (2007), ACM, pp. 87–93.

[LH05] LEFEBVRE S., HOPPE H.: Parallel controllable texture synthesis. *Proc. SIGGRAPH '05 24*, 3 (2005), 777–786.

[LH07] LEFEBVRE S., HOPPE H.: Compressed random-access trees for spatially coherent data. In *Rendering Techniques* (2007), pp. 339–349.

[LHN05] LEFEBVRE S., HORNUS S., NEYRET F.: Texture sprites: Texture elements splatted on surfaces. In *ACM-SIGGRAPH Symposium on Interactive 3D Graphics (I3D)* (2005), ACM Press, pp. 163–170.

[Los04] LOSASSO F.: Geometry clipmaps: terrain rendering using nested regular grids. *Proc. SIGGRAPH '04 23*, 3 (2004), 769–776.

[MBB*01] MAYER H., BORNIK A., BAUER J., KARNER K., LEBERL F.: Multiresolution texture for photorealistic rendering. In *In Proc. 17 th Spring Conference on Computer Graphics* (2001), IEEE Computer Society, pp. 174–183.

[McR98] MCREYNOLDS T.: *Programming with Opengl: Advanced Rendering*. ACM, 1998.

[OSRW97] OFEK E., SHILAT E., RAPPOPORT A., WERMAN M.: Multiresolution textures from image sequences. *IEEE Comput. Graph. Appl. 17*, 2 (1997), 18–29.

[OSW97] OFEK E., SHILAT E., WERMAN M.: Highlight and reflection-independent multiresolution textures from image sequences. *IEEE Computer Graphics and Applications 17* (1997), 18–29.

[PGB03] PÉREZ P., GANGNET M., BLAKE A.: Poisson image editing. *Proc. SIGGRPAH '03 22*, 3 (2003), 313–318.

[PV95] PERLIN K., VELHO L.: Live paint: painting with procedural multiscale textures. *Proc. SIGGRAPH '95 14*, 3 (1995), 153–160.

[QMK06] QIN Z., MCCOOL M. D., KAPLAN C. S.: Real-time texture-mapped vector glyphs. In *I3D '06: Proc. of the 2006 symposium on Interactive 3D graphics and games* (2006), ACM, pp. 125–132.

[QMK08] QIN Z., MCCOOL M. D., KAPLAN C.: Precise vector textures for real-time 3d rendering. In *SI3D '08: Proc. of the 2008 symposium on Interactive 3D graphics and games* (2008), ACM, pp. 199–206.

[RAKRF08] RAV-ACHA A., KOHLI P., ROTHER C., FITZGIBBON A.: Unwrap mosaics: A new representation for video editing. *Proc. SIGGRAPH '08 27*, 3 (2008), 17.

[Ram07] RAMANARAYANAN G.: Constrained texture synthesis via energy minimization. *IEEE Transactions on Visualization and Computer Graphics 13*, 1 (2007), 167–178.

[Sen04] SEN P.: Silhouette maps for improved texture magnification. In *HWWS '04: Proc. of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2004), ACM, pp. 65–73.

[SnZTyS03] SUN J., NING ZHENG N., TAO H., YEUNG SHUM H.: Image hallucination with primal sketch priors. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR* (2003), pp. 729–736.

[SXS08] SUN J., XU Z., SHUM H.: Image super-resolution using gradient profile prior. In *CVPR08* (2008), pp. 1–8.

[TC04] TUMBLIN J., CHOUDHURY P.: Bixels: Picture samples with sharp embedded boundaries. In *Rendering Techniques* (2004), Keller A., Jensen H. W., (Eds.), Eurographics Association, pp. 255–264.

[TMJ98] TANNER C. C., MIGDAL C. J., JONES M. T.: The clipmap: a virtual mipmap. *Proc. SIGGRAPH '98 17*, 3 (1998), 151–158.

[Wei04] WEI L.-Y.: Tile-based texture mapping on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches* (2004), ACM, p. 67.

[Wil83] WILLIAMS L.: Pyramidal parametrics. *SIGGRAPH Comput. Graph. 17*, 3 (1983), 1–11.

[WM04] WANG L., MUELLER K.: Generating sub-resolution detail in images and volumes using constrained texture synthesis. In *VIS '04: Proc. of the conference on Visualization '04* (2004), IEEE Computer Society, pp. 75–82.

[YWHM08] YANG J., WRIGHT J., HUANG T., MA Y.: Image super-resolution as sparse representation of raw image patches. In *CVPR08* (2008), pp. 1–8.