The Visual Computer manuscript No.
(will be inserted by the editor)

Textures Revisited (REVISED paper number 357)

Hitoshi Yamauchi[†], Hendrik P. A. Lensch[††], Jörg Haber[†], Hans-Peter Seidel[†]

[†]MPI Informatik, Saarbrücken, Germany, [††]Stanford University, USA

Received: date / Revised version: date

Abstract We describe texture generation methods for complex objects. Recent 3D scanning devices and high-resolution cameras can capture complex geometry of an object and provide high-resolution images. However, generating a textured model from this input data is still a difficult problem.

This task is divided into three sub-problems: parameterization, texture combination, and texture restoration. A low distortion parameterization method is presented, which minimizes geometry stretch energy. Photographs of the object taken from multiple viewpoints under modestly uncontrolled illumination conditions are merged into a seamless texture by our new texture combination method.

We also demonstrate a texture restoration method which can fill in missing pixel information when the input photographs do not provide sufficient information to cover the entire surface due to self-occlusion or registration errors.

Our methods are fully automatic except the registration between a 3D model with input photographs. We demonstrate the application of our method to human face models for evaluation. The techniques presented in this paper make a consistent and complete pipeline to generate a texture of a complex object.

Key words multiresolution texture synthesis, mesh parameterization, image inpainting, image restoration, facial modeling, frequency decomposition

1 Introduction

Texture mapping is one of the oldest techniques in computer graphics [9] — yet it is one of the most powerful techniques today. In its original form, texture mapping is used to convey realism of objects, which are modeled in a comparatively less complex and hence less realistic way. Extensive research has led to many improvements and related techniques such as, for instance, *bump mapping* [8] or *environment mapping* [29], which are commonly used to enhance the visual quality of rendering in many real-time applications. One of the main reasons for the success of texture mapping and related techniques is probably due to the hardware support of these techniques on high-end graphics machines in the early years and on low-cost commodity graphics boards today.

One easy way to generate realistic objects is scanning real objects. For example, we can capture the 3D geometry of an object using a 3D range scanner. In order to obtain textures we can also take photographs with a digital camera. 3D geometry together with a high-quality texture may, for example, be used to represent real objects in the context of virtual museums. Even in object design where no real counterpart of the object exists, one may start with an acquired object as a template.

Several problems have to be solved to apply this approach successfully. For example, we need robust techniques for efficient 3D object acquisition, mesh denoising, and registration between 3D geometry and 2D photographs. Furthermore, in this paper, we will focus on texture generation. This problem is subdivided three sub-problems.

1. **Parameterization:** For efficient storing and processing of the texture we need to compute a 2D parameterization of a 3D mesh.
2. **Texture combination from photographs:** The layout of the 2D parameterization needs to be optimized to reduce occupied texture memory. The information of several input views has to be merged.
3. **Texture restoration:** The derived texture often does not cover the entire surface. We need some interpolation or fill-in techniques for synthesizing a texture.

The example application of this paper is to generate a textured human face from a 3D scanned head model and several photographs. Because we are very well trained

to recognize real human faces, it is very challenging to produce a head model in sufficient quality.

The main contributions presented in this paper are:

- For parameterization, we present
  1. an alleviation method for the discontinuity effect in geometric stretch parameterization, and
  2. a view dependent parameterization method for effective use of texture memory.
- In texture combination from several photographs, we present a visibility-aware multi-resolution spline technique to remove boundary effects due to uncontrolled illumination.
- For texture restoration, we propose a technique that combines image inpainting and texture synthesis with non-parametric sampling methods through frequency analysis of input images.
- If a 3D model and its corresponding 2D photographs are registered, all of the techniques proposed in this paper are fully automatic.

This paper is organized as follows. First, we survey related work in Section 2. Then, we will detail each subproblem: parameterization (Section 3), texture combination from several photographs (Section 4), texture restoration (Section 5). We show some results in Section 6. Section 7 concludes the discussion.

2 Related Work

A common approach to display a texture mapped model with current graphics hardware needs a polygonal mesh, its parameterization, and a texture image. When only a polygonal mesh and several photographs are given, we are facing two problems: parameterization and texture image generation. The texture image generation problem is also subdivided into two sub-problems. One is texture combination. Texture combination is to make a single texture from several input photographs. The other one is texture restoration which is needed when the input photographs do not provide enough information for covering the entire surface of the input model.

First we will overview related parameterization methods. Then, we will review texture combination and texture restoration methods.

2.1 Parameterization

Parameterization is often represented as the mapping from 3D vertex coordinates on a mesh to 2D uv texture coordinates in a texture image. In this paper, we focus on parameterization of triangle meshes in 3D space.

In the good and old days, due to hardware limitations, texture mapping was applied to relatively simple polygonal meshes only. In this case, manually creating a parameterization was acceptable.

However, recent graphics hardware can display larger and more complex polygonal meshes with higher resolution texture images than before. It has become an essential problem to generate a high quality parameterization automatically.

There are several aspects of parameterization.

- **Texture Atlas:** To avoid high distortion, some parameterization methods cut the input mesh into charts and parameterize each chart individually. A texture atlas combines all these charts.
- **Energy Function:** The basic idea to solve the parameterization problem is to minimize energy function based on some distortion metric between 3D space and 2D space. This means the energy function should have large energy when the mapping introduces high distortion. According to the applied energy function, methods are classified as linear or non-linear.
- **Boundaries:** For effective use of hardware resources, a texture should fit to texture memory and should have square shape. On the other hand, this restriction may be too strong for low-distortion parameterization. Boundary conditions on the texture image model the importance of each restriction.

2.1.1 Texture Atlas The traditional approaches of parameterization are to cut a mesh into *charts*, and pack them to a *texture atlas* [6, 12, 38, 42, 45, 47, 60, 62, 70, 71]. Once a mesh is cut into charts, we can parameterize each chart by our favorite parameterization method.

Introducing a texture atlas, it becomes relatively easy to keep low distortion inside of the charts. There are however three main drawbacks of the texture atlas approach: (1) it introduces discontinuities between charts. (2) it introduces a mesh cutting problem when creating the chars, (3) it introduces artifacts while MIP-mapping.

Quite some effort has been taken to solve these problems, for example, the discontinuity problem [30, 41, 42, 44], or the mesh cutting problem [42, 45, 66]. However, it seems we still do not have a comprehensive method and some of these problems remain open. The third problem is substantially hard to solve because MIP-mapping usually assumes continuity in the uv domain.

Let us first focus on objects which are topologically equivalent to a disc. Those objects can be parameterized by a single chart avoiding the disadvantages of a texture atlas.

2.1.2 Energy Function Bennis et al. pioneered the use of energy functions in parameterization. They propose a technique to map an isoparametric curve of a 3D surface onto curves of a plane considering two distortion metrics [6]. Their energy function is based on the geodesic curvature preservation and arc length preservation. During the calculation, the mesh will be cut if the intermediate distortion energy is larger than a certain threshold.

The method unfortunately requires C^2 continuity of the surface because of the geodesic curvature preservation.

Maillot et al. define distance energy and surface energy for the parameterization [47]. The distance energy tries to minimize distortion while on the other hand it may introduce triangle flipping. To avoid this, the non-linear surface energy term is introduced. This surface energy term tries to avoid triangle flipping, but cannot guarantee that no triangles are flipped. The total energy is a weighted combination of both energies. Remaining triangle flipping can be removed using an interactive tool. Compared to the previous method, C^2 continuity is no longer required.

Quadratic Energy/Solving Linear Systems: Eck et al. propose the discretized harmonic map energy [20]. The harmonic map assigns non-uniform spring constants to the mesh edges which resemble a kind of Dirichlet energy. Since this energy is quadratic the minimum energy is found at the zero crossing of the derivative. Thus we can use any linear system solver to compute a parameterization of the input mesh. Solving linear systems is usually fast and more stable compared to solving non-linear systems. However, triangle flipping might occur for this method as well.

Floater proposes a similar method [24]. The advantage of this method is it can guarantee that no triangle is flipped when the parameterized mesh has a convex boundary. The energy function tries to keep triangle shape with approximating geodesic polar angle at each one-ring neighbor. This method is linear and stable. Later, Floater improves smoothness of the function based on mean value theorem for harmonic function [25].

The conformal surface parameterization has been introduced by Haker et al. [31]. We can find interesting similarity with in remeshing context by Duchamp et al. [19], and in fairing context by Desbrun et al. [17]. Conformal mapping finds the parameterization as the solution of a second order partial differential equation (PDE) defined on the input mesh, which keeps the conformality of the one-ring around each vertex.

Desbrun et al. presented the method of intrinsic parameterization [16] where the energy $\mathbf{U}$ is a linear combination of the conformal energy $\mathbf{U}^{\text{conformal}}$ and the authalic (area preservation) energy $\mathbf{U}^{\text{authalic}}$: $\mathbf{U} = \lambda\mathbf{U}^{\text{conformal}} + (1-\lambda)\mathbf{U}^{\text{authalic}}$. This paper also shows some criteria on how to calculate the optimal λ and how to obtain the natural boundary. Almost the same time, Levý et al. [45] presented least squares conformal maps. The parameterization is the same as the discrete natural conformal parameterization in [16][1]. This paper also deals with automatic texture atlas generation.

The basic structure of the above methods [16, 20, 24, 25, 31, 45] are the same and main differences are the definition of the distortion energy. One considers conformality, the other considers shape preservation, and so on. These methods solve a sparse linear system with different coefficients of the mesh connectivity matrix. One common feature of these methods is fast computation. However, we experienced lower quality of these linear methods compared to the non-linear methods, which we will address next. In cases where the geometry of the surface is more or less similar to the parameter domain, the results of linear and non-linear methods are similar.

Non-linear Energy: Hormann and Greiner propose the MIPS parameterization [37]. They attempt to preserve the isometry of triangles over the parameterization by fixing the condition number of the transformation matrix.

Instead of using the condition number of the transformation matrix, Sander et al. introduce the root-mean-square of the singular values of the matrix called geometric-stretch metric for parameterization [62]. Intuitively, this measures how a unit circle on the domain is stretched when it is mapped onto the surface. Later they introduce signal-stretch energy to improve the quality of the parameterization with respect to given texture [63], color or normals.

Balmelli et al. propose space-optimized texture maps [4]. They use the frequency information of the input image as a signal. The difference to the previous approach [63] is that they apply a warping function on the texture image instead of using distortion energy on the mesh. To make efficient use of texture memory, the high frequency areas of the image are stretched, whereas low frequency regions are shrunk.

All these methods [4, 37, 62, 63] generate high-quality parameterizations for texture maps, however, they are based on time consuming non-linear optimization.

In order to speed up the calculation, [63] uses a multi-resolution approach. First, they calculate the energy on a coarse mesh. Then, they sequentially add vertices similar to progressive meshes [36] and again minimize the energy. This process is repeated until all vertices are added. While the computation time is drastically improved it still takes around ten times more time than the linear optimization methods.

There are also parameterization methods based on the idea *flattening*. Sheffer and de Sturler [65] propose a parameterization method that minimizes an angle distribution error around each vertex. The error criterion is linear, but they introduce several non-linear constraints to avoiding unwanted situations, e.g., boundary intersections. Later, Zayer et al. [81] improved the computational cost and introduced boundary shape controlling factor. Sorkine et al. [70] propose another flattening method based on a modified geometric-stretch energy. Their method can guarantee the maximum error, because when the error is too large, the mesh is cut to achieve low distortion. One advantage of those flattening

[1] `http://www-grail.usc.edu/pubs/CD02.pdf`

methods is there is no limitation of predefined boundary condition, e.g., fixed boundary, convex boundary.

Other Interesting Methods: Igarashi proposes an interactive user guided parameterization method [38]. The user selects a painting area and a view direction and the local parameterization is found by projecting the selected area into the view plane. In Sander's paper [63], this method is mentioned as a signal based method, which signal or importance are defined by the user. When the amount of painted area is relatively small, this method works well since only the user specified important parts are parameterized by the projection.

Piponi and Borshukov propose a cutting and blending technique for textures on subdivision surface called *pelting* [57]. Since the cutting is guided by the user it is relatively easy to generate intuitively parameterized meshes for painting. The energy function is basically the same as in [47]. In order to reduce the artifacts across the cuts, texture seams are blended using a linear function.

2.1.3 Classification We can classify the different parameterization techniques according to their definition of the energy function. For example, the system is linear or not, whether the energy is globally defined. Further aspects are the topological limitation which the method imposes, if the parameter domain consists of a single chart only, or if multiple charts are grouped into a texture atlas, and so on.

Table 1 shows one example of such kind of classification. In this table, the column Atlas is Y(es) means that this method needs to cut the input mesh during the parameterization process. Therefore when this is N(o), input mesh and output mesh always have the same topology. This difference is slightly ambiguous, because some methods make an atlas in the preprocessing stage. For example, the methods [45, 62] first decompose the input mesh to submeshes, then each submesh is parameterized. However, the topology of each submesh does not change. Although these methods generate an atlas, it is not necessary to classify them as atlas generating method since the atlas generation is at the preprocessing stage.

Most of the methods need a boundary condition, e.g., fixed boundary of the input mesh. For example, some approaches treat the parameterization problem as the Dirichlet problem. Fixing the mesh boundary may introduce high distortion area around the mesh boundary. To break this limitation, some methods alleviate this condition. Such methods are marked as NB (Natural Boundary) is Y in the table.

Surface parameterization is a fundamentally problem, and attracts a lot of research interests. If the reader is interested in more theoretical details in this area, we refer to the recent survey by Floater and Hormann [26].

Table 1 Classification of parameterizations. Column "Atlas" shows generating an atlas is necessary or not. "NB" stands for Natural Boundary. If this is Y(es), it is possible to minimize energy with optimizing the boundary shape.

Linear System		
Energy Function	Atlas	NB
harmonic map [20]	N	N
shape preserving [24]	N	N
chord length [57]	N	N
conformal [31]	N	N?
feature match, gradient constraint, regularization [44]	N[2]	Y
conformal + area [16]	N	Y
improved shape preserving [25]	N	N
Non Linear System		
Energy Function	Atlas	NB
geodesic curvature, arc length [6]	Y	N
distance, surface(flipping) [47]	Y	N
isometry (MIPS) [37]	N	Y
geometric stretch [62]	N[2]	N
geometric stretch, signal stretch [63]	N	Y
flattening, angle distortion [65, 81]	N	Y
flattening, geometric stretch[3] [70]	N	Y
Others		
Energy Function	Atlas	NB
boundary smoothness [4] [42]	Y	–
simple projection [38]	Y	–

2.2 Texture Generation

Once a 2D parameterization is constructed, we have to create the corresponding texture image.

Soucy et al. propose a texture generation method for a complex triangle mesh of arbitrary topology [71]. First, 3D triangles of the mesh are sorted according to a certain criterion like area, largest edge length, and so on. Each triangle is independently inserted onto the texture map. They are packed as half-square triangle into the 2D texture. The texture atlas generated by this method contains one separate chart for each triangle.

Rocchini et al. propose an approach for mapping and blending the texture on the 3D geometry [60]. For a 3D scanned object, they start with a set of uncalibrated photographs that are registered interactively and stitched onto the 3D mesh. The relevant image regions are integrated into a single texture atlas map. The assignment of regions from the input images to the corresponding parts of the texture depends on the visibility according to the registration. Slight mis-registration is accounted for by automatic local feature alignment.

[2] These papers generate an atlas in preprocessing stage.

[3] The geometric-stretch energy [62] is defined as root-mean-square of the singular values of mapping matrix. However, this method [70] uses the max function of them.

[4] This method is for making an atlas, it does not care the parameterization method for parameterizing each charts.

Later, they propose preserving attributes detail on a simplified meshes [12] where the attributes range from color textures, to bump, displacement, or shape maps. The heuristic texture atlas packing method called irregular triangle packing is more sophisticated than former methods [60, 71].

During the combination of several images, visual artifacts may occur at the boundaries between original image regions. These artifacts are due to several reasons including difference in lighting condition, view dependent shading, or registration errors. To delete such unintended artifacts, the images are blended or blurred, e.g., by applying a Gaussian filter or continuous blending functions [57]. Pighin et al. propose a more sophisticated blending weight function, which considers self-occlusion, smoothness, positional certainty, and view similarity [56]. This method changes the blending weight according to the view direction. While the artifacts can be eliminated by these methods they also destroy image detail in the boundary area. In order to keep more details Burt and Adelson [11] propose a multi-resolution spline technique. Multi-resolution analysis decomposes the images into high and low-frequencies and images are blended at each level separately. This effectively eliminates the artifacts and still keeps high-frequency details of the images.

2.3 Image Restoration

In this paper, the term "image restoration" means filling in damaged or missing pixels. Image restoration plays an important role for texture generation. Typically some surface regions exist for which no information can be gathered from the input images due to occlusion, registration errors, or other reasons.

There are two main image restoration methods:

1. diffusion based image restoration,
2. texture synthesis based image restoration.

2.3.1 Diffusion-based Image Restoration In the image processing area, diffusion processes are useful for a large range of applications. Perona and Malik [55] introduce the anisotropic diffusion. Diffusion is formulated as $\partial u_t = div(G(u)\nabla u)$, where $G(u) = g(|\nabla u|^2)$. The function g represents an arbitrary edge detection operator. Because an edge region usually has a high gradient, the diffusion process will change its magnitude according to the absolute value of its gradient. Later, Saint-Marc et al. present adaptive smoothing [61]. This method adaptively changes the term $G(u)$ of the anisotropic diffusion. The term $G(u)$ does not only rely on the signal itself, but also the first derivative. Anisotropic diffusion has successfully been applied to problems such as de-noising, image segmentation, image enhancement, and so forth.

Bertalmio et al. introduce anisotropic diffusion as an automatic digital inpainting method [7]. Their anisotropic diffusion process fills the interior of user-defined image regions keeping isophoto lines.

Oliveira et al. propose a simpler and faster inpainting method [53]. They assume that the usual defect of an image is rather small and anisotropic diffusion needs to be applied only in exceptional cases. The user manually indicates such exceptional cases, and the defect part is repaired by isotropic diffusion otherwise.

Ballester et al. represent image interpolation scheme by solving the variational problem [3]. The diffusion process is represented by second order partial differential equations. It consists of a gray-level intensity diffusion term and a keeping gradient orientation term.

Pérez et al. propose a Poisson image editing method [54]. Using this method, images are edited by solving Poisson equations with respect to a user-prescribed guidance vector field under given boundary conditions. This is a versatile editing method. Some of the applications of this method, in particular seamless image insertion and feature exchange, are useful for image restoration.

2.3.2 Texture Synthesis-Based Image Restoration Recently, texture synthesis has drawn a lot of research interest. Based on a given sample, a large, new texture with a similar (but not identical) pattern is created automatically. Textures synthesis can also be used to fill in holes in image restoration. In this context, the word "texture" is defined as an image which has some spatial coherence.

Procedure-based texture synthesis: Texture synthesis has started with procedural texture generation, like a checker board. Basic procedure-based texture synthesis method describe the texture by a mathematical function. The major advantage of this method is that it is resolution independent. The texture can be generated at any resolution. However, you need to design a function and parameters for each texture. Other procedural-based texture synthesis methods use, for example, fractals [27], reaction-diffusion [75], and cellular particles [23]. There are also some explicit methods for certain textures, e.g., [49].

Statistical texture analysis and synthesis: Gaber proposes many basic methods for texture synthesis methods [28]. His paper includes several models: the N-gram model, autoregressive model, autoregressive linear model, algebraic reconstruction model, field definition model (segment a source image and transfer texture information with its mask), and best-fit model. Nowadays, his best-fit model is rediscovered and is called texture synthesis method [22].

Popat et al. present a cluster-based probabilistic modeling technique for high-dimensional vector sources [58]. A vector is constructed by concatenating a pixel and its neighborhood pixels of the input texture.

These vectors are analyzed to estimate the probability density function (PDF) of the pixel occurrence in the texture. The estimated PDF is used for synthesizing a texture. They also propose hierarchical texture synthesis method using multi-resolution analysis.

Heeger and Bergen propose a pyramid-based texture analysis/synthesis method [33]. The input texture is analyzed using a multi-resolution method, and a steerable pyramid is created. The output texture is initialized with random noise, then the histogram distribution is matched to the input texture at each pyramid level. By construction, the output image has the similar histogram distribution through all the levels of the pyramid. Since this method only considers the histogram distribution of the input and the output, the input image should be a homogeneous texture. It cannot handle periodic or non-homogeneous patterns. Later, Bonet introduced joint occurrence of pixels across multiple resolutions to Heeger's method [14]. Because this method considers the hierarchical structure between pyramid levels, it can reconstruct larger structures than Heeger's method.

Simoncelli and Portilla present a texture synthesis method based on statistical measurements of the seed image and its wavelet decomposition [67]. The authors apply higher order statistics while the previous methods [14, 33] consider only first order statistics, i.e., histograms.

An approach based on statistical learning to reproduce the appearance of an input texture has been proposed by Bar-Joseph et al. [5]. Their method treats the input as a signal which accords a hierarchical statistic model. Using wavelets, they construct a tree representing a hierarchical multi-scale transform of the input signal. A new texture is synthesized by tracing this tree according to a similarity path. The approach allows for mixing different input textures by mixing their corresponding trees. Even to one-dimensional signals like sound sequences the methods have been applied successfully and video sequences are intended. The presented results are quite impressive compared to other statistical method.

Some statistical texture synthesis models also expand to the temporal domain, namely synthesizing video sequences. Szummer and Picard propose such a model [72]. They assume temporal texture can be modeled by Gaussian noise with an autoregressive signal, leading to a spatio-temporal autoregressive model (STAR). They demonstrated their method on wavy water, rising steam, and fire. They consider the consistency between frames in their model.

Texture synthesis: In 1948 Claude Shannon mentioned a method for producing English-sounding text based on N-grams [64]. The new text is synthesized by searching for similar text sequences in the seed text. Popat and Picard's approach [58] is a kind of extension of this idea to two dimensions. While Popat and Picard used a parametric sampling, Efros and Leung propose a non-parametric sampling method which creates a look up directory from the input source prior to texture synthesis [21]. The same idea is found in Garber's forgotten work [28]. It is based on Markov random fields which depend on the generated pixels and the input seed texture. Compared to other statistics-based texture synthesis models, this approach has no problems in reproducing the spatial structure of the input texture.

Wei and Levoy present an acceleration method of the non-parametric texture synthesis using tree-structured vector quantization [77]. They apply the causal kernel [58] (it is called "best-fit model kernel" in [28]) to scanline order. The output image boundaries are handled toroidally to obtain a tileable texture. In order to capture a large structure in the input texture, multi-resolution texture synthesis is considered. They also propose an extension to three dimensional textures, e.g., to the temporal domain.

A coherent match method for the similarity lookup in source texture has been presented by Ashikhmin [2]. The coherent match selects a similar subimage according to the history of the synthesis process. Once a similar subimage is found, the next similar subimage is very likely to be located adjacent to the last selected one, because texture has some local coherence. In addition, a user can guide the process to obtain a specific output texture. The user inputs rough structures of the output texture which are then together with the already synthesized pixels considered during the search for the similar subimage in the seed texture. The coherent match method is rather efficient.

Zelinka et al. propose a realtime texture synthesis version of the coherent map [82]. They first analyze the input image to make a similar subimage link map from each pixel, called Jump Map. Synthesis is performed by either copying the next pixel or jumping to a similar place according to a random number and then copying the pixel. To speed up the computation times, no similarity comparison is done during synthesis.

Hertzmann et al. present an image processing framework by example, called "Image Analogies" [34]. This filter makes a mapping from source image A to destination image A', and apply that map to source image B to output the destination image B'. That is why it is called Image Analogies. This filter synthesizes an image using both of the above mentioned non-parametric sampling texture synthesis functions, i.e. a best approximate match [21, 28, 77] and a best coherence match [2]. The distances resulting from these two matching functions are weighted by a user-specified parameter to determine which of the two matches is selected. The authors demonstrate many applications of the image analogies filter, e.g., traditional image filter, texture synthesis, super-resolution, texture transfer, artistic filter, texture by numbers, and so forth.

Most of these methods search for similar vectors in the spatial domain. Soler et al. propose to search in

the frequency domain [69]. They reinterpret similarity search as correlation, and the correlation of two functions can be computed in $O(N \log N)$ in Fourier space through the FFT algorithm instead of $O(N^2)$. When the input is static, a kD-tree makes the calculation complexity $O(N \log N)$ for the search. However, the tree must store all vector elements. This consumes a large amount of memory since each pixel typically has an associated subimage vector which size is 10 to 100 pixel. Instead of this, Soler's method only requires the size of input image (DCT) or the doubled size (FFT).

To exploit more coherence in the source texture, patch-based approaches have been proposed instead of per-pixel-based approaches.

Efros and Freeman propose a patch-based texture synthesis method called image quilting [22]. First, a seed texture is subdivided into smaller blocks. Second, these blocks are randomly replaced such that neighboring blocks overlap. Finally, the overlapping regions are blended with minimal error bounding cut.

Liang et al. present a patch-based sampling texture synthesis method [46]. To exploit the coherence in the texture image, they transfer the complete subimage instead of a single pixel at each search. They introduce a new tree structure to solve ambiguities by multiresolution analysis.

A patch-based technique for image and texture synthesis has been proposed by Cohen et al. using Wang tiles [13]. Wang tiles are squares in which each edge is assigned a color. A valid tiling requires all shared edges between tiles to have matching colors. They propose a stochastic tiling method which can generate nonperiodic pattern and a seamless Wang tile seed generation method applying the image quilting [22]. Since only a color match test is required for construction, this method is much faster than other methods that need an expensive similarity search for generating a non-periodic texture.

Patch-based texture synthesis methods are usually very fast because they generate several pixel set at once. The drawback of these methods is when the input has large structures like complex depth or low frequency shading effects, artifacts become visible in the synthesized images.

Nealen et al. propose a hybrid method combining patch-based methods with pixel-based methods [50]. The patch similarity is calculated as described in [69], then they calculate the error of the surrounding patch boundary. Each pixel with too large error is optimized using pixel-based texture synthesis. This method can handle large structures as well as patch-based methods but can produce better boundaries. This advantage comes along with a loss of speed compared to the patch-based methods.

Usual synthesis method use L_2 norm as similarity measure. Harrison uses an entropy measure with Manhattan distance (L_1) to calculate the similarity measure [32]. He states that the L_2 norm emphasizes outliers. In addition, he introduces a constraint map to handle non-homogeneous textures. The user marks an importance map containing weights for each source pixel. He demonstrates that some of the non-homogeneity in the texture can be handled with this importance map.

Another subimage similarity metric is proposed by Brooks and Dodgson [10] for editing a texture image. Similarity is used to replicate editing operations to all similar pixels in the texture. For the similarity metric all pixels in the input image are encoded by a single value computed as the sum of squared differences between each corresponding neighborhood pixel. Instead of comparing subimages the similarity is defined as the L_2 norm between the encoded values.

These texture synthesis methods assume the homogeneity in the source texture, they use similarity search with relatively small kernels. Therefore, it is hard to capture large structures which are introduced by the depth of the scene or by shading effects. Multi-resolution analysis to some extend alleviates these effects [34, 77].

To break the limitation caused by large structures in the image, Oh et al. separate the image to segments manually according to the depth [52]. The texture synthesis method can use this depth information. However, the segmentation needs severe manual preprocessing. It is mentioned in the paper that processing one example takes around ten hours by hand. The structure of the illumination is extracted by applying the (bilateral) SUSAN filter [68, 74]. This filter can smooth out small details while it keeps sharp edges. Therefore, this filter suits texture editing with relighting.

Frequency domain transfer: Hirani and Totsuka propose an image restoration algorithm which can handle both texture and intensity variation [35]. Their algorithm is based on projections onto convex sets and employs a Fourier transform together with a clipping procedure. Projection and Fourier transformation are carried out alternating for a user-defined number of iterations. In addition, the user must interactively select a sample region, which will be used to repair the damaged area. The sample region is restricted to be a translated version of the defective region w.r.t. its texture. Care is taken to automatically adapt intensity variation between the sample and the defected regions.

Texture synthesis on surface: Traditionally texture synthesis has been carried out on two dimensional images, however recently there are several texture synthesis methods that work directly on surfaces of 3D objects [76, 78, 80, 83]. The main difficulty is to define an appropriate regular neighborhood on arbitrary surfaces. Wei and Levoy generate this regular pattern by local parameterization of the mesh [78]. Turk's method uses a direct vector field generation technique on surfaces [76]. Zhang et al. also employ a user-guided vector field on

the surface. In addition, they use a texon mask to keep features of the seed texture and resample pixels for obtaining rotation and scale effects [83]. This approach requires the user to input the initial course direction of the vector field. According to the vector field a regular sampling can be achieved. Lexing et al. solved the problem by introducing a texture atlas containing a sufficient number of charts for a smooth parameterization [80]. There are several methods to make such charts, for instance [42].

Restoration method based on texture synthesis: Igehy and Pereira introduce an image replacement method [39] based on the histogram distribution texture synthesis method introduced by Heeger [33].

Drori et al. propose an iterative approximation restoration method [18]. The problem of restoration methods based on texture synthesis is that the result is very sensitive to each pixel selection. This means that once a wrong pixel is selected, it becomes a fatal visual effect in the restoration. To avoid this binary decision effect, Drori introduce a confidence map. During the restoration process, a missing part is selected with its surrounding subimage, and the subimage is assigned a certain confidence value. If most of the subimage belongs to the non-missing part, the confidence value is high. If, however, a subimage is only reconstructed, its confidence value is low. This method iteratively updates the missing part and the confidence map to avoid wrong selection.

3 Parameterization

Let us first consider the case of parameterization of a 3D input mesh over the 2D domain $[0,1]^2$. In order to avoid artifacts when texturing the 3D object we will first concentrate on parameterization of the entire mesh at once, i.e., without introducing cuts on the surface resulting in a *texture atlas* containing a single patch. The result introduces no problems when MIP-mapping is applied, but it is clear that this goal cannot be achieved for arbitrary meshes.

The focus application of this paper is on representing human faces. Fortunately, human faces are topologically equivalent to a disk, since one can introduce a boundary around the neck and faces typically do not contain any handles. Some other example meshes in this paper also have disk topology.

In this section, we will compare the output of various parameterization algorithms and introduce a new method which adds two new terms to the L^2 geometric stretch energy in order to obtain a less distorted and controllable parameterization.

3.1 Parameterization Techniques

Figure 1 shows comparisons of several parameterization methods of the Stanford bunny model. For texture mapping, the L^2 geometric stretch method [62] usually produces the best results because it can keep both conformality and low area distortion. However, this method needs an initial valid parameterization, and it is usually time consuming since it minimizes a non-linear energy function. In our case, an initial valid parameterization is generated using Floater's [24] method.

Other examples produced by L^2 geometric stretch parameterization are given in Figure 2. In fact, since these models are more or less geometrically similar to a disk, even some of the linear energy minimization methods (e.g., [16, 24, 25, 31]) can produce good parameterization results for these models.

Figure 3 shows the results of geometrically rather complex models. For most parts of the models, good conformality and low area distortion are achieved. However, the L^2 geometric stretch parameterization sometimes produces severe cracks on the textured model as shown in Figure 3. For example, both the Stanford bunny model and the mannequin head model have several cracks on the backside of their neck. These cracks are referred to as "parameter cracks".

Praun and Hoppe introduce L^p regularization term to L^2 geometric stretch energy to alleviate this problem [59].

$$L^p = \epsilon \left(\frac{A'(T_i)}{4\pi} \right)^{p/2+1} (\Gamma(T_i))^p \qquad (1)$$

where p and ϵ are user defined parameters, $A'(T_1)$ is the area of the ith triangle T_i in the 3D mesh, and $\Gamma(T_i)$ is the largest singular value of the transformation matrix from 3D domain to 2D domain. This regularization term tries to punish triangles which have large singular values more badly than only using L^2 geometric stretch energy. Instead of applying this L^p term to inverse stretch calculation as in [59], we applied this term to planar domain in the same way. However, the results in Figure 4 show that even this term alleviates the effect of parameter cracks a little, it also introduces other unwanted distortions. Moreover, the L^p term requires two non-intuitive parameters, ϵ and p. The effect of these parameters are hardly predictable and in our experiments the optimization gets stuck more easily in local minima compared to the pure L^2 geometric stretch energy case. In Figure 4 we test several parameters starting with the recommended values in [59] ($\epsilon = 0.001, p = 6$). In this experiment, the area normalization coefficient 4π of L^p term in Equation (1) is 1.0 since our parameter domain is a unit square instead of a unit sphere.

As several applications require preserving the mesh structure [40], we only apply this regularization term to parameterization unlike [59] which uses remeshing. It

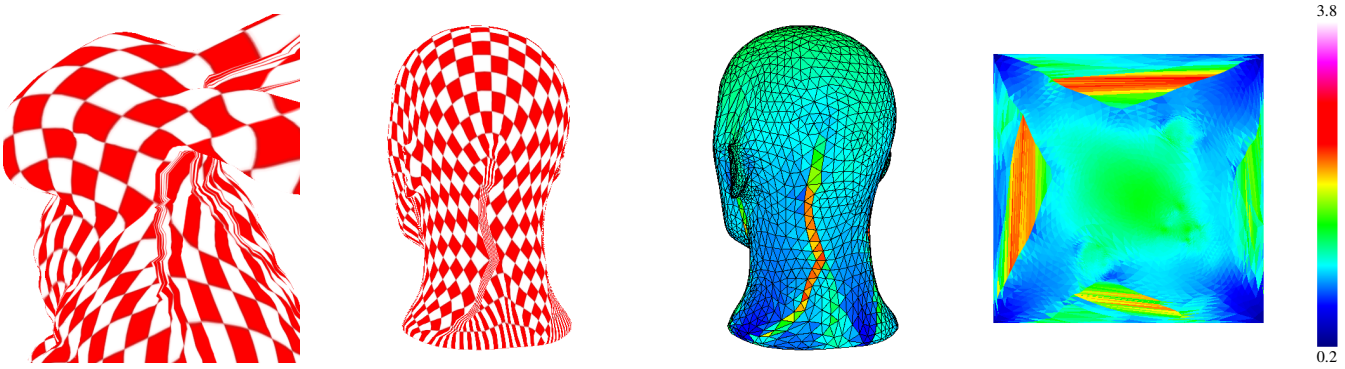


Fig. 3 The left two images show examples of parameter cracks on 3D textured model. Notice that there is a large distortion around the neck part. The right two images show the distribution of distortion for the mannequin model both on the 3D model and on its 2D parameterization. The distortion ranges from 0.2 (blue) to 3.8 (white) in terms of $L^2(T)$ of [62]. For the bunny model, the total distortion $L^2(\text{bunny}) = 2.17 \times 10^4$, and for the mannequin model $L^2(\text{mannequin}) = 7.11 \times 10^3$.

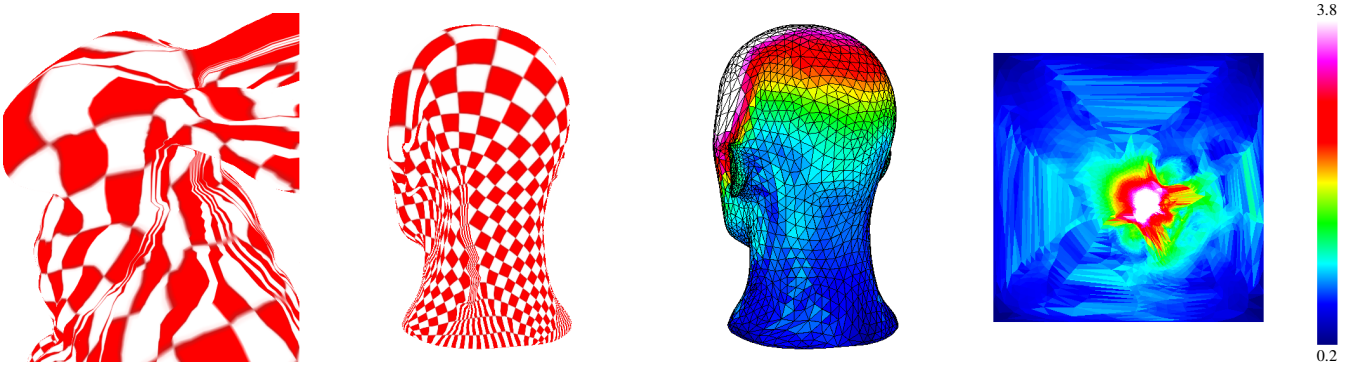


Fig. 4 Results of a $L^2 + L^6$ parameterization. For the bunny model, a choice of $\epsilon = 0.01, p = 6$ results in $L^2(\text{bunny}) = 3.01 \times 10^4$, and for the mannequin model, $\epsilon = 0.001, p = 6$ gives a total distortion of $L^2(\text{mannequin}) = 1.07 \times 10^4$.

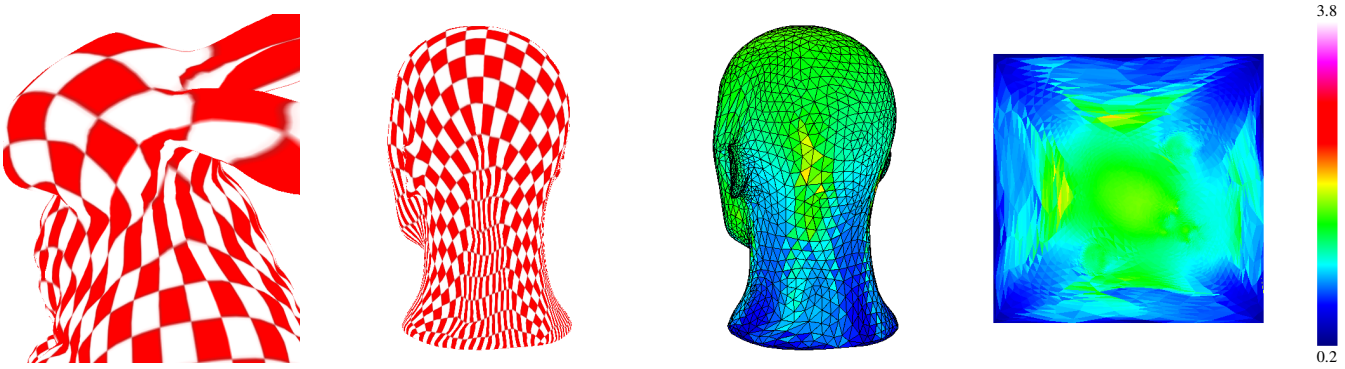


Fig. 5 Influence of the triangle shape term $s(T_i)$ on the distortion. Compared to Figures 3 and 4, parameter cracks are alleviated. Here, $L^2(\text{bunny}) = 4.40 \times 10^4$, and $L^2(\text{mannequin}) = 7.58 \times 10^3$.

might be more effective to achieve low distortion by combining a parameterization with regularization term and remeshing. The paper [59] shows a good result of this combination if a remeshing is possible. In the next section, we will also discuss avoiding the parameter cracks effect without changing input mesh structure.

3.2 Extensions to the L^2 Geometric Stretch Energy

We introduce two new terms to the L^2 geometric stretch energy as shown in Equation (2). One is for increasing the effective use of texture area and the other is for avoiding the parameter cracks effect.

$$L^2(M) := \sqrt{\frac{\sum_{T_i \in M} \left\{ \left(L^2(T_i)\right)^2 \omega(T_i) A'(T_i) + s(T_i) \right\}}{\sum_{T_i \in M} \omega(T_i) A'(T_i)}} \tag{2}$$

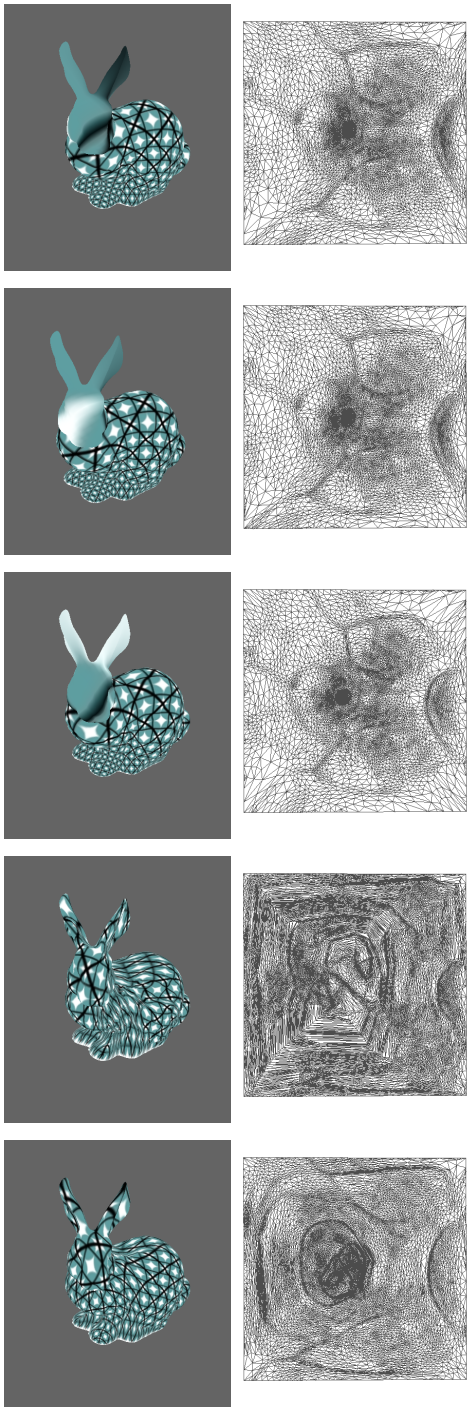


Fig. 1 Comparison of parameterization methods. From top to bottom, 1. Floater [24], 2. Intrinsic [16] ($\lambda = 0.5$), 3. MIPS [37], 4. L^∞ geometric stretch [62], 5. L^2 geometric stretch [62].

with

$$\omega(T_i) = \frac{1}{\langle N(T_i), V\rangle + k}$$

$$s(T_i) = \begin{cases} 0 & \min \frac{h}{b} >= \alpha \\ \text{infinite energy} & \min \frac{h}{b} < \alpha \end{cases}$$

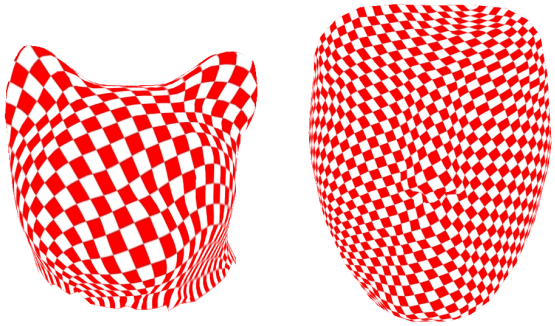


Fig. 2 Well parameterized cat head and forehead obtained by L^2 geometric stretch parameterization.

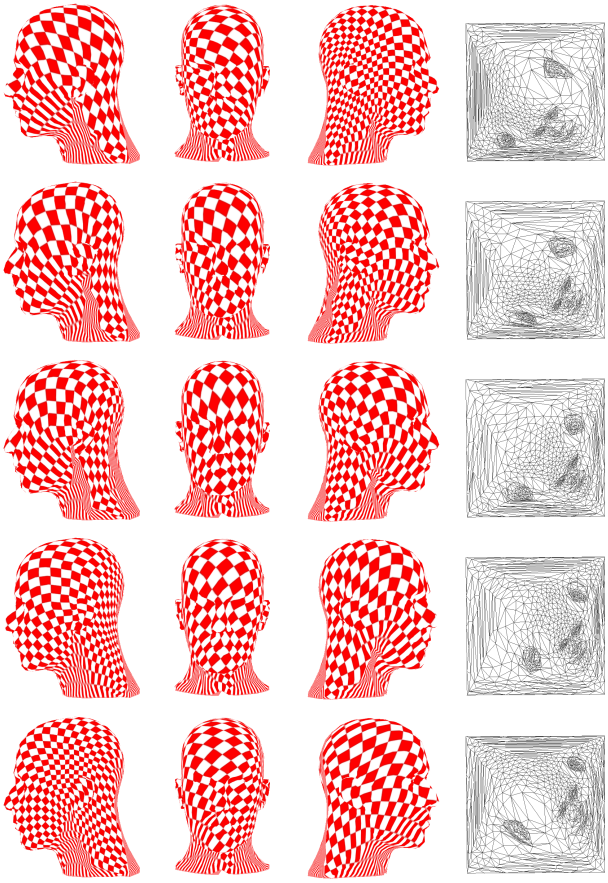


Fig. 6 The effect of the visual importance function $\omega(T_i)$. From top to bottom, the view direction V is changing from the left side of the face to the right side. Each row left to right: left side, front, and right side of the mannequin head model, and the parameterization result. For all images, $k = 1.2$ has been chosen.

where $M = \{T_i\}$ denotes the triangle mesh, $A'(T_i)$ is the surface area of triangle T_i in the 3D mesh, $N(T_i)$ is the triangle's normal, V is a direction vector, and $k(> 1)$ and α are user prescribed parameters (see below).

The first term $\omega(T_i)$ models "visual importance." Triangle geometric stretch energy is minimized over the whole mesh equally. However, we would like to use as

much texture space as possible for the "important" regions of a model while minimizing the texture space allocated to "unimportant" regions since the size of textures that can be handled by graphics hardware is typically limited. For instance, the front face is more important for the viewer than the ears or even the back of the head in human head model. Once an important view is defined by a user through the direction vector V, the visual importance function ω thus favors the triangles on the face by diminishing their error while penalizing the triangles on the back of the head by amplifying their error. As a consequence, triangles on the face become larger in the texture mesh while backfacing triangles become smaller. This we call a view-dependent parameterization since parameterization is depend on the important view. The user can control the degree of ω's influence to the parameterization result through the parameter k. To enlarge the influence of visual importance, k should be close to 1.0. From our experience, useful values for k are within $[1.01, 2]$, which results in $\omega(T_i) \in [1/3, 100]$. The role of this ω is considered as another signal of [63].

Figure 6 shows view-dependent parameterization results for various view directions V. In this figure, we use the energy function of Equation (3) without the $s(T_i)$ term for clarifying the effect of visual importance function only. The user-defined parameter k is set to $k = 1.2$.

The second term $s(T_i)$ controls each triangle's shape. The regularity of a triangle is represented by the ratio h/b of the height h and the baseline length b. There are three ratios in a triangle, and we define h/b to be the smallest ratio of the triangle. When this ratio is too small, the energy becomes infinite, which punishes this triangle. The triangle shape threshold α is given by the user. α depends on the input mesh, the resolution of input texture image, and an additional criterion. One plausible criterion for texture mapping is, e.g., that all triangles should cover at least one complete pixel. From our experiences, we recommend $\alpha \in [0.05 \ldots 0.15]$ for $[512 \times 512 \ldots 4096 \times 4096]$ texture images with up to 10^4 triangles. We also assume the input mesh triangle more or less satisfies this h/b ratio.

Figure 5 shows the effect of this triangle shape term on reducing cracks. The term introduces less distortion than in $L^2 + L^p$ geometric stretch energy results, and it still keeps conformality and low area distortion. We also assume the h/b ratio parameter to be more intuitive than the L^p energy in Equation (1), since the h/b ratio is more directly coupled to the triangle shape. One large difference to [73] is that our method can control this parameter cracks effect. Introducing the $s(T_i)$ term makes the total distortion energy higher than the original L^2 energy (see Figures 3 and 5), however, this term equalizes the distortion distribution and achieves the lowest visual artifacts.

4 Texture combination

After having created the 2D parameterized mesh from the 3D input mesh, we resample the texture mesh from the input photographs that have been registered with the mesh. We will show how to combine a single texture from several input photographs using a face model and photographs as an example.

4.1 Resampling input images

First, we perform a vertex-to-image binding for all vertices of the 3D face mesh. This step is carried out as suggested in [60]: Each mesh vertex v is assigned a set of *valid photographs*, which is defined as the subset of the input photographs such that v is visible in each photograph and v is a non-silhouette vertex. A vertex v is visible in a photograph, if the projection of v on the image plane is contained in the photograph **and** the normal vector of v is directed towards the viewpoint **and** there are no other intersections of the face mesh with the line that connects v and the viewpoint. A vertex v is called a silhouette vertex, if at least one of the triangles in the triangle fan around v are oriented opposite to the viewpoint. For further details see [60]. In contrast to the approach in [60], we do not require that all vertices of the face mesh are actually bound to at least one photograph, i.e. the set of valid photographs for a vertex may be empty.

Theoretically, this is enough for classifying vertices. However, there might be some error because of registration or numerical errors especially in the neighborhood of silhouettes. Some of the vertices can be bound to background pixels of the input photographs. Such a vertex should be classified as an unbound vertex. We detect this registration error by comparing the color value with the background color of input image. First we calculate projection coordinate of a vertex to its bound photograph. Then, we sample the pixel value and calculate the distance between this pixel value and the background color. To avoid misdetection, we reduce the noise in the input photographs by applying a usual median filter and not just compare with a single sample pixel but perform the Gaussian convolution to with a 3x3 subimage mask. If the distance is larger than a given threshold value, then the vertex is re-classified as an unbound vertex.

Let $\triangle = \{v_1, v_2, v_3\}$ denote a triangle of the face mesh and $\widetilde{\triangle} = \{\tilde{v}_1, \tilde{v}_2, \tilde{v}_3\}$ be the corresponding triangle in the texture mesh. For each triangle $\triangle$, exactly one of the following situations might occur (see also Figure 7):

1. There exists at least one common photograph in the sets of valid photographs of the three vertices v_1, v_2, v_3 of $\triangle$ (green triangles).
2. All of the vertices of $\triangle$ are bound to at least one photograph, but no common photograph can be found for all three vertices (blue triangles).

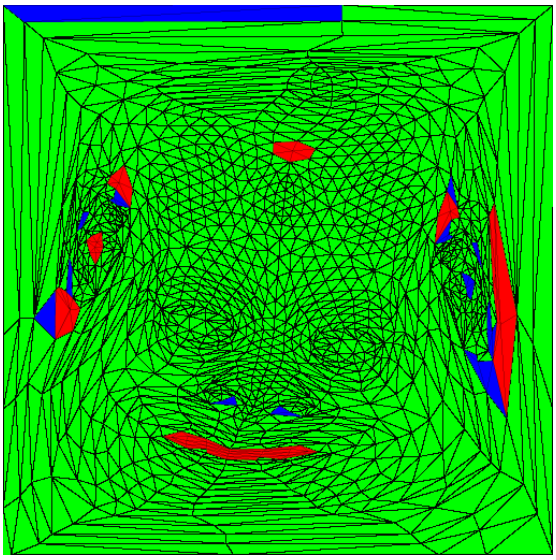


Fig. 7 Color-coded triangles of the textured mesh: each green triangle has at least one common photograph to which all of its vertices are bound; the vertices of blue triangles don't have a common photograph, but they are all bound; red triangles have at least one unbound vertex.

3. At least one vertex of $\triangle$ is not bound to any photograph (red triangles).

In the first case, we rasterize $\widetilde{\triangle}$ in texture space. For each texel $T_{\tilde{}}$ we determine its barycentric coordinates ρ, σ, τ w.r.t. $\triangle$ and compute the corresponding normal N by interpolating the vertex normals of $\triangle$: $N = \rho N(v_1) + \sigma N(v_2) + \tau N(v_3)$. For each common photograph i in the sets of valid photographs of all vertices of $\triangle$, we compute the dot product between N and the viewing direction V_i for the pixel P_i that corresponds to T. Finally, we color T with the color obtained by the weighted sum of pixel colors $\sum_i \langle N, V_i \rangle \cdot \text{Color}(P_i) \,/\, \sum_i \langle N, V_i \rangle$.

In the second case, we color each vertex $\tilde{v}_j$ of $\widetilde{\triangle}$ individually by summing up the weighted pixel colors of the corresponding pixels in all valid photographs i of $\tilde{v}_j$ similarly as in the first case: $\text{Color}(\tilde{v}_j) := \sum_i \langle N(v_j), V_i \rangle \cdot \text{Color}(P_i) \,/\, \sum_i \langle N(v_j), V_i \rangle$. The texels of the rasterization of $\widetilde{\triangle}$ are then colored by barycentric interpolation of the colors of the vertices $\tilde{v}_1, \tilde{v}_2, \tilde{v}_3$. Alternatively, we tried to use as much information as possible from the input photographs if, for instance, the vertices v_1, v_2 of $\triangle$ share a photograph and the vertices v_2, v_3 share another photograph. However, this situation always happens near the silhouette of object and the extrapolation of a missing vertex on the photograph will be unstable. [51] mentions about similar situation. They recommend to use this kind of uv coordinate extrapolation since at least the boundary has plausible color and this is better than just a hole. We try this extrapolation and then perform our registration error detection scheme, we found that it fails in most of the case. Therefore, the plain color interpolation from reliable vertices usually produces much better results in our case.

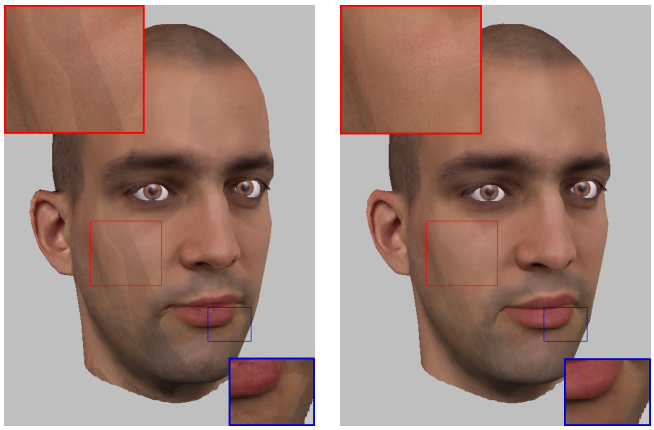


Fig. 8 Boundaries in the skin texture (left) are removed using multi-resolution spline techniques (right).

Since we do not require that each vertex of the face mesh is bound to at least one photograph, there might exist some vertices that cannot be colored by any of the previously described schemes. We address this problem in a two-stage process: First, we iteratively assign an interpolated color to each unbound vertex. Next, we perform the color interpolation scheme from the second case for the remaining triangles of $\widetilde{\triangle}$ that have not yet been colored. The first step iteratively loops over all unbound and uncolored vertices of the face mesh. For each unbound vertex v, we check if at least $p = 80\,\%$ of the vertices in the one-ring around v are colored (either by being bound to a photograph or by having an interpolated color). If this is true, we assign to v the average color of all the colored vertices around v, otherwise we continue with the next unbound vertex. We repeat this procedure until there are no further vertex updates. Next, we start the same procedure again, but this time we only require $p = 60\,\%$ of the vertices in the one-ring around v to be colored. As soon as there are no more updates, we repeat this step twice again with $p = 40\,\%$ and $p = 20\,\%$. Finally, we update each unbound vertex that has at least one colored neighbor. Upon termination of this last step, all vertices of the face mesh are either bound or colored and the remaining triangles of $\widetilde{\triangle}$ can be colored.

This color interpolation method is fast and easy to implement, and it can fill all missing pixels. However, texture detail can not be reconstructed by this scheme. More sophisticated pixel filling methods, for example, texture reconstruction, will be discussed in Section 5.

4.2 Combining images with resampling

If the input photographs have been taken under uncontrolled illumination, the color might differ noticeably between the images. In this case, boundaries might appear in the resampled texture. We then apply a multi-

resolution spline technique as proposed in [11, 43] to remove visual boundaries. Figure 8 shows a comparison between a textured head model with and without the multi-resolution spline technique applied. The multi-resolution spline technique needs a mask to determine the overlapping region where is resampled from different input photographs. We propose an automatic computation method of this mask for each region. Because we have the 3D model and its registration information, we can test the visibility of each triangle on the input photographs to make a mask for combination. Then we remove the outmost ring of triangles around the region, see Figure 9. Such a shrinking is necessary to ensure that there is still some valid color information on the outside of the mask boundary, because these adjacent pixels might contribute to the color of the boundary pixels during the construction of Gaussian and Laplacian pyramids.

We choose polygon resolution rather than pixel resolution for this shrinking because of its simplicity and robustness against noise in the input photograph and inaccuracies during projection along the silhouette.

We generate a texture triangle by triangle. Since texture coordinates are assigned to each vertex through OpenGL functions, triangle-based resampling is more simple and straight forward than pixel-based resampling.

For our validity test, we need to separate background and foreground of the input photographs. We apply a median filter and Gaussian denoise filter for the separation. However, from our experience pixel-based validity tests are rather unreliable. Moreover, the calculation of 3D to 2D back projection near the object silhouette becomes inaccurate since the dot product of the normal of near-silhouette triangles and the view direction is close to zero.

Therefore, we use a conservative method. If all three backprojected vertices of a visible triangle in 2D are valid, we set all pixels inside the triangle to be valid. Some of the adjacent pixels of such a triangle might also be valid, but since this is difficult to determine due to input noise and backprojection inaccuracy, we simply discard them.

Here we assume that each backprojected triangle in 2D covers several pixels. However, when the triangle mesh is very dense and the resolution of the input image is low, it might happen that a backprojected triangle covers less than one pixel, in which case this method is problematic. However, this situation is unusual for texture mapping. In the opposite extreme case, the mesh resolution is too low and each triangle occupies a large area in the 2D image. In this case, our method may discard many potentially valid pixels. However, this means that the model to be textured is geometrically rather simple and the problem is trivial.

In addition to the masks for each input photograph, we create one more mask that is defined as the complement of the sum of all the other masks. This mask is used together with the resampled texture to provide some color information in those regions that are not covered by any input photograph (e.g. the inner part of the lips). As described above, these regions have been filled by color interpolation in the resampled texture. By blending all of the masked input photographs and the masked resampled texture with the multi-resolution spline technique, we obtain a final texture with no visual boundaries and with crisp detail. We use this additional mask for texture restoration to distinguish missing regions in Section 5 since this mask indicates if each pixel is resampled or not.

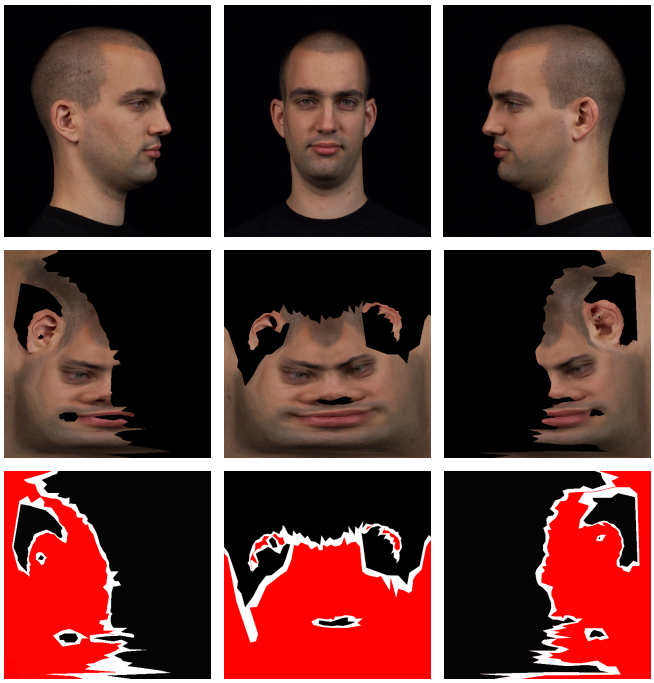


Fig. 9 Multi-resolution spline masks: input photographs from three different view points (top), texture meshes resampled from their corresponding photographs (middle), and their automatically generated masks shown in red (bottom).

5 Texture restoration

In Section 4, we introduce the color interpolation method to fill in the missing area. The method can produce smooth color interpolation, but it is hard to reconstruct details like texture.

If we can assume that our reconstructing object is easy to access and it is static, we can take other photographs and register them until all pixel information is given by photographs. However, some objects like a human face are not static, and it is difficult to take photographs under the same condition, e.g., the same lighting condition. Moreover, some objects are difficult to access, for instance, historically valuable objects or already lost objects, but several photographs remain. In

these cases, if the total number of missing pixels is relatively small, we should consider about reconstructing these missing pixels from given incomplete information.

So far, there exist two different main approaches to fill in missing pixels: image inpainting and texture synthesis. After giving an overview of these methods we also propose a new approach which exploits the advantages of both existing methods.

Image inpainting: Image inpainting methods [7, 53] are based on a diffusion process to fill missing regions. Each pixel value in the missing region is calculated from surrounding pixels according to a partial differential equation. In one of the simplest case, this partial differential equation takes a Laplacian operator to "diffuse" surrounding pixels to the missing pixels.

An anisotropic diffusion method is proposed [7] by Bertalmio et al. to deal with isophotolines. This method can keep some kind of edges in the input image. On the other hand, Oliveira et al. [53] state that there is no need to keep isophotolines in almost any case, however, when they should be kept, their method adds the diffusion barrier manually.

Both methods work well in missing areas which are relatively small or thin, like scratches, blotches, and texts. However, the problem arise when the missing area will be large. Because these methods fill missing area with a continuous function by solving partial differential equation, the reconstructed area will become smooth. Therefore, if the considered image contains small details, these details will not be reconstructed by the diffusion process. Figure 15 includes a result of this method and shows that the inpainting method deals well with text and scratches, but fails at the large area.

Texture synthesis: Three main techniques are proposed to synthesize texture:

1. procedural methods (e.g. fractal images),
2. stochastic methods (e.g. histogram equation, N-gram equalization),
3. non-parametric sampling methods.

If an image generation function is available or can be assembled, procedural methods typically are the best choice, since they usually results in resolution free images. But it is usually difficult to find such a function.

Dealing with arbitrary input at first, stochastic texture synthesis methods are proposed for synthesizing a texture. Later, these methods are applied to image restoration or image replacement [39]. These methods capture the features of a texture through some certain stochastic measurement (e.g., mean intensity value, or the histogram distribution of image). Then all we need is to just an example texture, not an image generation function. Even these methods can transfer some stochastic parameter from source image to target image, the image usually needs some homogeneous structure which can be handled by this stochastic parameters [14, 33].

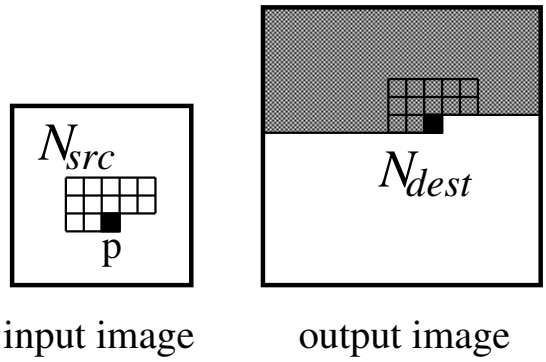


Fig. 10 Non-parametric sampling texture synthesis. The black pixel 'p' is the reference pixel of kernel N. The gray region of the output image is completed region.

A non-parametric sampling texture synthesis method is demonstrated in Figure 10. This method get an input source image and generate a similar-looking destination image as an output. In the texture synthesis process, a subimage N_{dest} called "neighborhood kernel" is extracted from the destination image, and a similar kernel N_{src} is looked up in the source image which then determines the destination pixel color. The kernel shape in Figure 10 is called best-fit [28] or causal [76] kernel and is frequently used for texture synthesis. The kernel has a reference point which is marked as "p" in Figure 10. Several criterions have been proposed for measuring similarity in order to find the matching source kernel, for instance, Euclidean distance (L_2 norm), Manhattan distance (L_1 norm), entropy, and so forth. These distances are also dependent on the applied color space model. When a kernel with the smallest distance has been detected, the pixel value of reference point is transfered from the source image to the destination image. This is repeated until all destination pixels are copied.

This non-parametric sampling method also needs some homogeneity in the source image. This method can capture the local detail information of image by using the kernel. The problem arise when the algorithms tries to capture global structures of the source image like a large shadow region because the algorithm uses a kernel of certain size to capture the features of image. When a large size kernel is used to capture global features, the search space becomes too small and loses the freedom to capture the feature since the source image size is fixed. This is a typical "curse of dimensionality" problem. Namely, we can not extract enough sampling information in the fixed size data source, when the dimension of the sampling kernel becomes larger. To break this limitation, multi-resolution search [77], coherent match [2], or combinations of both [34] have been proposed. Figure 15 includes a result of this method and shows that the texture synthesis method can reconstruct texture but fails to capture global structures.

Image restoration: Image inpainting and texture synthesis methods assume an image as a height field and try to fill the missing part of the field. We can summarize the both methods in this point of view:

- **Image inpainting method:** The height field is reconstructed by a certain continuous function. Global structures are captured, but local detail information is usually lost.
- **Texture synthesis method:** Similar height field patterns are looked up using a small kernel and transferred during reconstruction. Some local details like a texel are captured, but global structures can usually not be reproduced.

Image inpainting methods are good at reconstructing global structure in an image and texture synthesis methods are good at reconstructing local details of an image. These methods looks complementary, however, these methods are not based on the same mathematical theory: one is based on partial differential equation and the other is based on non-parametric search. In order to combine them we need further study.

Let us consider these properties from the signal processing manner:

- **Image inpainting method:** The lower frequency part of an image is reconstructed.
- **Texture synthesis method:** The higher frequency part of an image is reconstructed.

Since the lower frequency part of an image represents the global structure of the image and the higher frequency part represents the local structure of it, we propose a method that combines the advantages of both image inpainting and texture synthesis without inheriting their disadvantages. The main idea is that an input image is first decomposed into a lower frequency part and a higher frequency part by a frequency decomposition technique, e.g., Fourier transform. Then, the lower frequency part is reconstructed by image inpainting method, and the higher frequency part is reconstructed by non-parametric sampling texture synthesis method. Finally, both reconstructed images are combined to obtain the overall result. Here we will explain an enhanced method of [79], newly included a rotation invariant search method and a shrink order reconstruction method. In [52], the authors use a bilateral filter to extract the illumination structure. The filter smoothes out small details while it keeps sharp shadow edges. This method works well for image editing, for instance relighting images. However, for our purpose, keeping sharp shadow edges is not preferable since sharp edges include a high-frequency component. Because the texture synthesis method has a much higher potential to reconstruct the high-frequency component than a diffusion based process, we apply a frequency decomposition method and combine the two techniques described above.

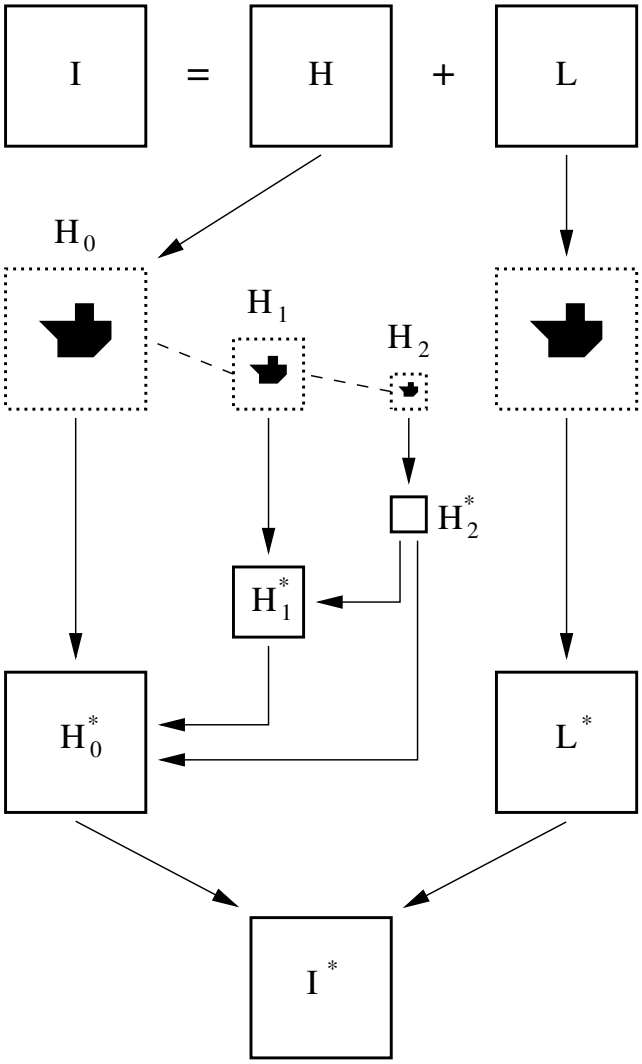


Fig. 11 Overview of our method. top to bottom: the input image I is decomposed into a high-frequency image H and a low-frequency image L using a DCT. Image inpainting is applied to the low-frequency part L to obtain the inpainted image L^*. The high-frequency part H is decomposed into a Gaussian pyramid (shown up to level 2 in this example). Starting from the highest level (H_2), multi-resolution texture synthesis is applied to the masked areas of the levels H_i. For each level, the neighborhood vector for the texture synthesis (cf. [77]) is composed of the kernels of that level and of all higher levels. In this way, coherency is maintained throughout all texture synthesis levels. Finally, the resulting high-frequency image H_0^* and the low-frequency image L^* are summed up to yield the restored image I^*.

5.1 Image restoration algorithm overview

First we need to determine the missing region in a texture image. In the usual case, classification of a missing or defect region of an input image is subjective. Therefore, this information should be given by a user. On the other hand, in our face model example, the missing parts are previously known as we described in the end of Section 4. However, the proposed image restoration algo-

I L^* H_0, H_1 H_0^*, H_1^*

Fig. 12 Multi-resolution texture synthesis. Left to right: input image I; Inpainted low-frequency image L^*; Two levels (H_0, H_1) of the Gaussian decomposition of the high-frequency image H; The same two levels after texture synthesis (H_0^*, H_1^*). The detail images H_i and H_i^* $(i = 0, 1)$ are shown with gamma correction to emphasize the high-frequency detail. The restored image I^* is shown in Figure 15 bottom right. (The original texture was obtained from the VisTex web Page, Copyright ©1995 MIT.)

rithm is independent of how the image is created and regions to be filled in are detected.

We need two inputs for this algorithm. One is the input image I and the other is a binary mask M which stores the identifying information of the region to be reconstructed, i.e. for each pixel in I we have a corresponding binary value in M.

Our algorithm proceeds as follows (cf. also Figure 11):

1. The input image I is decomposed into a high-frequency part H and a low-frequency part L using a discrete cosine transformation (DCT) (cf. Section 5.2).
2. The fast image inpainting algorithm proposed in [53] is applied to the interior of the masked areas of the low-frequency image L to obtain the inpainted image L^*. During this step, information from the entire input image may be used by the image inpainting algorithm. Here, only the pixels inside the masked areas will be modified.
3. The high-frequency image H is decomposed into a Gaussian pyramid with $n+1$ levels H_i $(i = 0, \ldots, n)$. Section 5.3 provides some more details about this step.
4. Starting from the highest level H_n, we apply multi-resolution texture synthesis [77] inside the masked areas in H_i $(i = n, \ldots, 0)$:
   4.1. First, a kD-tree for fast nearest neighbor look-up is built [1]. However, the search space for texture synthesis in level H_i does not only contain the non-masked areas of H_i, but additionally includes the corresponding areas of the already synthesized higher levels H_k^* $(k = i+1, \ldots, n)$. To obtain the source image for the highest level H_n, we simply apply the complementary mask $\overline{M_n}$ to H_n.
   4.2. Texture synthesis is applied inside the masked area of H_i. The *neighborhood vector* (cf. [77]), which is used to perform a look-up in the kD-tree, is composed of the pixel information from the texture synthesis kernel in level H_i **and** of all corresponding kernels from the higher levels H_k^* $(k = i+1, \ldots, n)$. This ensures high coherency among all texture synthesis levels.

   More details about this texture synthesis step are given in Section 5.4.
5. The synthesized high-frequency image H_0^* and the inpainted low-frequency image L^* are summed up to yield I^*, which represents the restored version of the input image I.

Details of our implementation are given in the following sections. Figure 12 shows some of the intermediate levels of our algorithm for a sample input image.

5.2 Frequency decomposition

In the first step of our algorithm, the input image I is decomposed into a set of spectral sub-bands using a discrete cosine transform (DCT).

We select the first κ sub-bands and compute the inverse DCT of this subset. The resulting image is used as the low-frequency image L_κ. The corresponding high-frequency image H_κ is obtained by subtraction: $H_\kappa := I - L_\kappa$. Obviously, the parameter κ determines an upper bound for the (low) frequencies that are contained in L_κ. Our goal is, to have as much detail (= high frequencies) as possible in H_κ, while making sure that low-frequency gradients are completely contained in L_κ.

We assume two hypotheses to find a suitable frequency parameter κ. These hypotheses are:

1. If the lower frequency part of an image is adequately eliminated, the rest part of the image will be more homogeneous,
2. Homogeneity of an image can be measured by calculating the autocorrelation matrix of an image.

To this end, we compute the autocorrelation matrix[5] A_κ of H_κ: $A_\kappa := \mathrm{DCT}^{-1}(\mathrm{DCT}(H_\kappa) \cdot \mathrm{DCT}(H_\kappa))$.

[5] The autocorrelation matrix A of a matrix H is defined as $A := \mathrm{DCT}^{-1}(\mathrm{DCT}(H) \cdot \overline{\mathrm{DCT}(H)})$, where $\overline{H}$ denotes the conjugate of H. In our case, the matrices H_κ (and thus also $\mathrm{DCT}(H_\kappa)$) contain real numbers only because we use DCT to decompose input image.

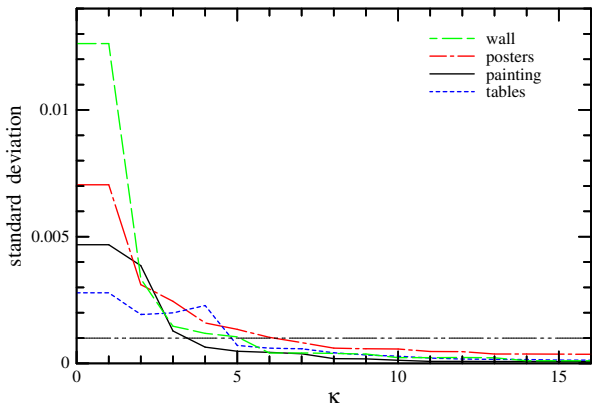


Fig. 13 Standard deviations of the autocorrelation matrices A_κ of the input images shown in Figure 17 plotted over the range of $\kappa = 1, \ldots, 16$.

For a non-square input image I, we pad H_κ with zeros to obtain a square matrix H'_κ and clip the zeroed border of the resulting $A'_\kappa := H'_\kappa \cdot H'_\kappa$. Next, we compute the standard deviation of the elements of the autocorrelation matrix A_κ. Figure 13 shows the standard deviations of the elements of autocorrelation matrices of the input images shown in Figure 17. They are plotted over the range of $\kappa = 1, \ldots, 16$. We found that choosing the lowest κ value that yields a standard deviation of less than 0.001 gives good results in general.

The effect of κ is shown in Figure 16. When the κ goes to zero, our method is identical to the texture synthesis method which is shown in Figure 15. When the κ becomes larger, our method becomes to the image inpainting method, since extracted higher frequency part is fewer. Therefore, there might be an optimal κ value which can extract enough large structure as a low frequency part, but still keep the small details.

We use the Y channel of XYZ CIE color model for this analysis. Since the large structures of an image such as shadows usually are most prominent in the Y channel [27].

5.3 Gaussian decomposition

The decomposition of the high-frequency image H into a Gaussian pyramid is based on the approach proposed by Burt and Adelson [11]. In particular, we employ the 5x5 Gaussian kernel ω proposed in [11] with the recommended parameter value $a = 0.4$:

$$\omega(u, v) = \widehat{\omega}(u)\,\widehat{\omega}(v)$$
$$\widehat{\omega}(0) = 0.4,\ \widehat{\omega}(\pm 1) = 0.25,\ \widehat{\omega}(\pm 2) = 0.05$$

The pyramid decomposition proposed in [11] requires that the input image has a resolution of $(p\,2^N + 1) \times (q\,2^N + 1)$ pixels $(p, q, N \in \mathbb{N})$ to ensure that a Gaussian pyramid of $N{+}1$ levels may be constructed. In our case, we may safely terminate the pyramid decomposition at a level $n \ll N{+}1$. This can be explained as follows: during the texture synthesis in level i, we include the pixel information from the kernels of all higher levels $i{+}1, \ldots, n$ into the nearest neighbor search. To be successful, however, the search needs to have enough candidates (contiguous groups of pixels). Thus, the size of the smallest H_n must not be too small. In practice, we obtained good results for pyramid decompositions up to level three. As a consequence, the resolution of the input image I is practically unrestricted for our method.

5.4 Texture synthesis

For the texture synthesis (step 4 in Section 5.1), we implement and test the approaches presented in [21], [77], and [2]. We finally implement the approach proposed in [34], which basically switches between the texture synthesis algorithms from [77] and [2] from level to level, depending on a local distance criterion. during the texture synthesis in level H_i, we typically use the 5x5 bestfit/causal kernel from [28, 77] within H_i, a standard 3x3 kernel for level H_{i+1}, and a 1x1 kernel for the higher levels H_k $(k = i{+}2, \ldots, n)$.

Multi-resolution texture synthesis is applied inside the masked areas of each level H_i $(i = n, \ldots, 0)$. The complementary part of H_i (i.e. the part of H_i which is outside the masked areas) is used as the source image. Since the H_i differs in size from level to level, the mask has to be adapted. Let $M_0 := M$ denote the user-defined binary mask in the size of the input image. We decompose this mask into a Gaussian pyramid up to level n using the same approach and kernel as for the image data (see Section 5.3). This operation is carried out in floating point arithmetic with 1.0 and 0.0 representing true and false for the initial level M_0, respectively. Thus, the higher levels M_i $(i = 1, \ldots, n)$ contain blurry images of the initial mask M. Next, we quantize every M_i back into a binary mask such that 0.0 maps to false and any other value in $]0, 1]$ is mapped to true. Given that the number of levels of the pyramid is typically three or four in our application, the clear distinction between 0.0 and any value larger than zero is not an issue in single precision floating point.

Image reconstruction order: The reconstruction pixel order usually has an influence to the result [18, 32], except for highly homogeneous texture [78]. Also, when the kernel shape is not symmetric (ex. bestfit/causal kernel), direction of kernel may change the result.

This is why we implemented multiple reconstruction orders: scanline order type 1 (one way, $+x$ direction), scanline order type 2 (alternative way, $+x$ and $-x$ direction), and shrinking order. The "shrinking" order fills a hole in a manner that the hole is shrinking. First, we calculate a distance map inside of the hole. Each pixel of this map has a Manhattan distance to the boundary of

the hole. Then the reconstruction order follows the distance. In our experience, we found the shrinking order usually produces the best results.

There is one issue in this hole filling process. If the missing hole has a concave shape and the kernel shape does not fit the boundary, we cannot not extract the neighborhood kernel. In this case, this missing pixel will remain and its distance will be increased by one, to be filled in the next iteration.

However, when the bestfit/causal kernel is used to find the similar kernel, this kernel sometimes does not fit the hole boundary since it is designed for the scanline order reconstruction. Therefore, we use eight different directions of bestfit/causal kernel (cf. Figure 14). During the image analysis phase (cf. Section 5.1, algorithm 4.1), all these eight kernels are used to analyze the source image and eight kD-trees are generated. In reconstruction phase (cf. Section 5.1, algorithm 4.2), a possible kernel which is fit to the hole boundary is extracted from the destination image. Then, we look up the corresponding kD-tree of the extracted kernel, and find the closest kernel.

This algorithm can shrink any holes, even if the boundary is concave. Consider the texture is synthesized in a scanline order fashion, then all possible holes would be filled if the first left top most pixel can be filled. The scanline order is a special case of shrinking order. Therefore, this hole filling is always possible. Here, we assume that it is possible to fill the first missing pixel. This means at least we have a non-missing subimage which fits the neighborhood kernel. We think this assumption is reasonable, since the neighborhood kernel is usually 5x5 or similar size. However, we usually need more non-missing pixels to get a better result in practice.

Another advantage of shrinking order to fill holes is that we can use fixed-shape-fixed-size kernels for any arbitrary boundary shapes. This means we can still use kD-trees for searching. An arbitrary shaped kernel is more powerful, and can exploit the all non-missing pixel information. However, this can not fit the kD-tree search algorithm since you can not query a vector with changing its size and structure. Furthermore, it is not practical to generate all kD-trees corresponding all patterns and to store them to the memory, because the total number of arbitrary shaped kernel is the sum of binomial function. Some methods [18, 50] use arbitrary shaped kernel for their search which yields a linear search with computational complexity of $O(n^2)$ which is larger than the $O(n \log n)$ complexity of the kD-tree search. Although the fixed shape kernel approach might miss some information during the search, we think this is a good compromising point considering the rotation invariant search and the computation speed.

Figure 15 shows a comparison of texture synthesis [77], image inpainting [53], and our approach. The reconstruction order in texture synthesis is scanline order as proposed in [77]. Both texture synthesis and image inpainting treat small scratches and thin area well, i.e., the text. Limitations of texture synthesis and image inpainting arise when large areas are missing. This point is also stressed in [18] where the texture reconstruction from real objects is investigated. In this problem, usually some continuous missing area rather than small scratch or blotches occur. Because of this, we need a method to reconstruct large structures and small details.

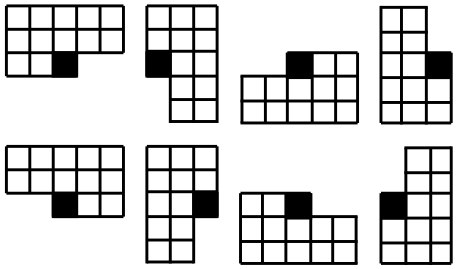


Fig. 14 Eight different orientations of 5x5 bestfit/causal kernel. The reference point is filled with black.

Our algorithm is controlled by two different parameters: the number of DCT sub-bands (κ), from which the low-frequency image L_κ is computed (cf. Section 5.2), and the number of levels $(n+1)$ in the Gaussian decomposition of the high-frequency image (cf. Section 5.1). In practice, we obtained very good results when choosing the lowest κ value that yields a standard deviation of less than 0.001 as described in Section 5.2. Thus, the choice of κ is fully automated in our approach. The optimal number of levels in the Gaussian decomposition is somewhat hard to predict, though. In general, we obtained good results when using three or four levels, i.e. setting $n = 2$ or $n = 3$.

Figure 17 shows some results obtained with our method. Each input image is shown with its mask applied. For the purpose of illustration, the color of each mask has been chosen to differ significantly from the content of the input image. We found that the restored images look plausible in general. In some cases we obtained results that looked surprisingly good. One example is the *table* image (Figure 17, right column), where the flare of the highlight that is reflected from the marble floor is restored very well after the masked tables have been removed. We have not performed numerical comparisons of the results of different image restoration techniques, though. We believe that a simple RMS comparison is useless in the context of image restoration, since it does not take into account relevant perception issues.

In our approach, we applied image inpainting to handle intensity gradients in the input images. During our simulations, we found that multi-resolution texture synthesis alone can solve the intensity variation problem to a certain extent. However, the cases of the missing areas are relatively larger and more irregularly shape, using image inpainting in additional to multi-resolution texture synthesis is more favorably.

input image texture synthesis [77] image inpainting [53] our method

Fig. 15 Comparison between texture synthesis, image inpainting, and our method for input images with texture (top row) and with texture and additional intensity gradient (bottom row). Each row left to right: input image (damaged areas are masked out); Resulting images from texture synthesis [77], from image inpainting [53], and from our new method.

$\kappa = 2$ $\kappa = 4$ $\kappa = 8$ $\kappa = 16$

Fig. 16 Effect of κ value. When κ is equal to zero, this method is identical to the texture synthesis method since no low frequency part is extracted. When κ is maximum, this method is identical to the image inpainting method, since in this case all frequency components are treated as low frequency part. An optimal κ is in between them. We use the coherence parameter [34] = 10.

Currently, our implementation is rather experimental: no optimizations have been performed, and the timings include gathering of quite a lot of statistical data. All timings were collected on a 1.7 GHz Pentium4 PC and are given for an input image size of 600×450 pixels. The time to restore an image depends heavily on the percentage of the masked pixels. In our simulations, we typically used masks that covered 4–6 % of the input image. For these masks, our algorithm took about 5–10 minutes to complete (including I/O). The initial fast image inpainting took 4–20 seconds, depending on the convergence of the (iterative) inpainting algorithm.

6 Results

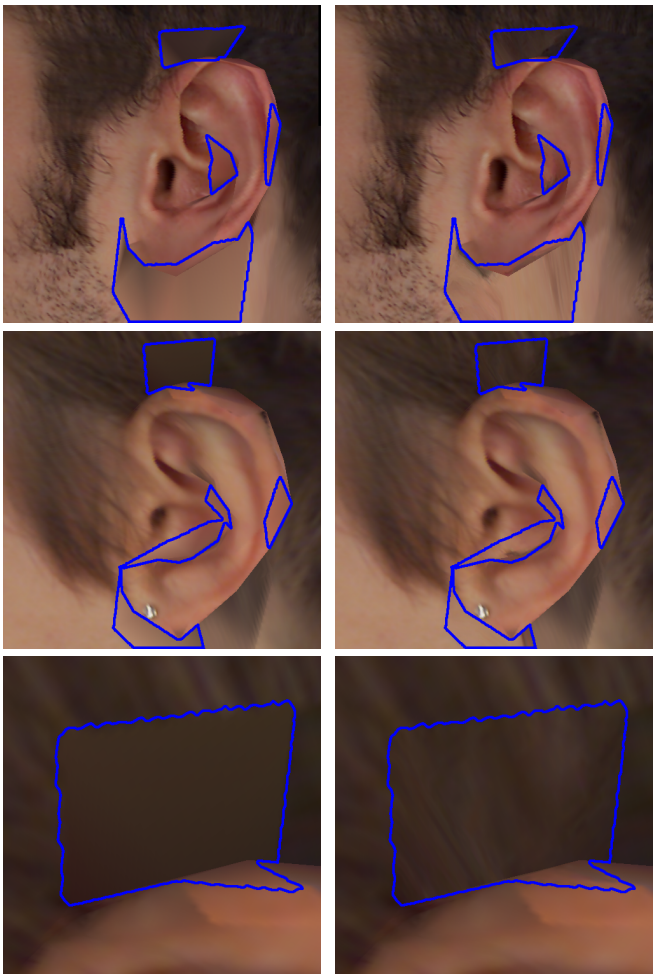
Figure 18 shows the results of our face reconstruction applied to two face models. The presented method is fully automatic except for registering a 3D scanned model with photographs. In this example, we process 3D range scan data according to [40] in order to obtain the input mesh, and register photographs to the mesh by hand. Figures 18 (a.2) and (b.2) are the parameterization results using geometric stretch energy with triangle shape (height-baseline) ratio $\alpha = 0.05$ and visual importance factor $k = 1.5$ in Equation (2) where the view vector is directed to see the front face. Resampling results are found in the same Figure (a.3) and (b.3). Blue pixels indicate that the input photographs do not provide any information for this surface region, i.e., there is no photograph where the surface points are visible. Figures 18 (a.4) and (b.4) show the color interpolation results after filling in the hole as described in Section 4, while in Figure (a.5) and (b.5) the missing part are reconstructed using the proposed image restoration tech-

painting ($n=1, \kappa=4$) posters ($n=1, \kappa=6$) wall ($n=1, \kappa=5$) tables ($n=2, \kappa=5$)

Fig. 17 Top row: input images with masked areas. Bottom row: restored images (see also Section 6). The parameter κ (= number of DCT sub-bands used to compute the low-frequency image L_κ) has been chosen automatically according to our autocorrelation metric (cf. Section 5.2). n is the highest multi-resolution level. The level is starting from 0.

niques as described in Section 5. The frequency decomposition parameters are $\kappa = 4$ in (a.5) and $\kappa = 5$ in (b.5). These κs are calculated with the criterion as described in Section 5.2. In Figures 18 (a.6) and (b.6) the texture mapped 3D model is presented. We use five photographs as the input, namely front, left, right, back, and the upper front view. An example of front, left, and right inputs is given in Figure 9. If we render the reconstructed model from a view which is found in the input photographs, it is clear that this is a simple problem as all information can be found in one of the input photographs. The problem becomes more interesting when we render novel views such as Figures (a.6.1), (a.6.3), (b.6.1), and (b.6.3). On these pictures, we can see the combined images from several (three or four) different input photographs. The images demonstrate that the reconstruction quality compares very well to the input images.

Figure 19 shows the comparison of the color interpolation method and the image restoration method. You can find the areas enclosed by blue lines on Figures 18 (a.3),(b.3). Although these regions have no input information, both methods can reproduce colors. Besides the image restoration method reconstructs some texture.



color interpolation image restoration

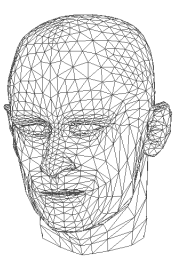
Fig. 19 Comparison of color interpolation method and image restoration method. Inside area of the blue lines has no texture information. The bottom row is zoom in of the upper part of ear at middle row. (cf. (a.3) and (b.3) in Figure 18)

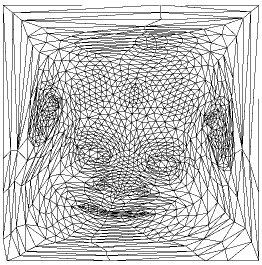
7 Conclusions and Future Work

We have proposed a method to generate textures from 3D geometry models and individual, uncalibrated photographs. Our method requires no user interaction for most processing steps. Only the feature registration step requires interactive specification of a few feature points.

Our approach consists of three sub-tasks: parameterization, texture combining, texture restoration.

For the parameterization, we introduced two signal terms for the geometry stretch energy method: the *visual*


(a.1) input model


(a.2) parameterization


(a.3) resampling


(a.4) color interpolation


(a.5) image restoration

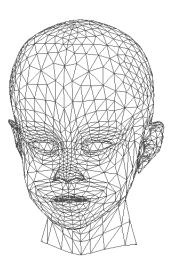

(a.6.1) left front view

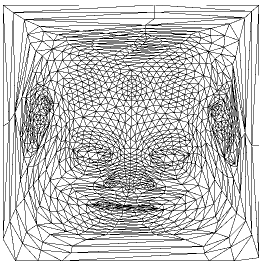

(a.6.2) front view


(a.6.3) right front view


(a.6.4) back view

(a.6) model + texture mapping result (the texture image is (a.5))


(b.1) input model


(b.2) parameterization


(b.3) resampling


(b.4) color interpolation


(b.5) image restoration


(b.6.1) left front view


(b.6.2) front view


(b.6.3) right front view


(b.6.4) back view

(b.6) model + texture mapping result (the texture image is (b.5))

Fig. 18 Face reconstruction results. In parameterization ((a.2), (b.2)), visual importance parameter $k = 1.5$, triangle shape ratio $\alpha = 0.05$, In image restoration, frequency parameter $\kappa = 4$ (a.5) and $\kappa = 5$ (b.5). These κs are automatically calculated by the criterion of Section 5.

importance term and the *triangle shape* term. The visual importance term defines the importance on the texture for efficient use of texture area. The triangle stretch term is for alleviating the parameter cracks problem of L^2 geometric stretch parameterization.

For the texture resampling problem, we apply the multi-resolution spline technique to delete the boundary artifacts which come from different illumination conditions of input photographs. We need masks for applying the multi-resolution spline technique. In our case, these masks have complex shape. We propose the automatic mask generation method using registered 3D model information.

Texture restoration is needed because sometimes we can not find pixel information on input photographs by registration error or occlusion. In our case, taking some more pictures to fill such missing pixel information might not be a good solution. The human face is a dynamic object and it is hard to reproduce exactly the same face. Therefore, introducing more pictures may add up to the error. Moreover, a face has view dependent reflection component (e.g., specular reflection component), this makes difficult to control the illumination condition.

To solve this problem, we introduced color interpolation method which exploit the 3D mesh topology to fill in the missing pixels, and image restoration method which combines image inpainting method with texture synthesis method by using the frequency analysis. We also demonstrated that this image resolution method is useful for a variety of defective images.

A remaining problem is the automatic generation of a robust registration method of a 3D model with corresponding input photographs.

Other future work for each subproblem is as follows:

- In parameterization, we gave two kinds of signals. Each signal parameter is decided experimentally. A promising research direction would be to find optimal parameter values for fine tuning the signals.
- In texture resampling, we solved the boundary problem by the multi-resolution spline technique. This method connects discontinuity regions with certain continuous function. However, the discontinuity comes from the difference in illumination conditions. Some techniques have been proposed to eliminate the view dependent element of illumination [15, 48]. We believe that these methods could improve our results further.
  We used a triangle-based mask shrinking method, which is simple to implement and robust to noise. However, image resolution and projected triangle area might not match. We are currently investigating shrinking methods taking into account image resolution.
- In image restoration, we reconstructed the missing parts of the input image using its boundary information. Therefore, the reconstruction may not be stable for certain complex boundary conditions. We are planning to investigate a robust method which can exploit the global structure in order to reliably reconstruct images.
  In this paper, we focused on a fully automatic method. However, if we can utilize some user guided information for reconstruction, a higher quality may be obtained. For example, in [83], a user can prescribe feature information (texton mask), rotation vector, and transition function. Such user-defined guidance may improve the reconstruction quality and also give some freedom to the user to control the results.

Acknowledgements This work has been partially funded by the Max Planck Center for visual computing and communication.

References

1. Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. *Journal of the ACM*, 46(6):891–923, 1998. `http://www.cs.umd.edu/~mount/ANN/`.
2. Michael Ashikhmin. Synthesizing natural textures. In *Proc. of 2001 ACM Symposium on Interactive 3D Graphics*, pages 217–226, March 19-21 2001.
3. C. Ballester, M. Bertalmio, V. Caselles, G. Sapiro, and J.Verdera. Filling-in by joint interpolation of vector fields and gray levels. *IEEE Trans. Image Processing*, 10(8):1200–1211, August 2001.
4. Laurent Balmelli, Gabriel Taubin, and Fausto Bernardini. Space-optimized texture maps. *Eurographics*, 21(3):411–420, September 2002.
5. Ziv Bar-Joseph, Ran El-Yaniv, Dani Lischinski, and Michael Werman. Texture mixing and texture movie synthesis using statistical learning. *IEEE Trans. on Visualization and Computer Graphics*, 7(2):120–135, 2001.
6. Chakib Bennis, Jean-Marc Vézien, and Gérard Iglésias. Piecewise surface flattening for non-distorted texture mapping. *Computer Graphics (SIGGRAPH '91 Conf. Proc.)*, pages 237–246, 1991.
7. Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Computer Graphics (SIGGRAPH '00 Conf. Proc.)*, pages 417–424, 2000.
8. James F. Blinn. Simulation of Wrinkled Surfaces. In *Computer Graphics (SIGGRAPH '78 Conf. Proc.)*, volume 12, pages 286–292. ACM Press, 1978.
9. James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547, Oct. 1976.
10. Stephen Brooks and Neil Dodgson. Self-similarity based texture editing. In *ACM Trans. Graph.*, pages 653–656. ACM Press, 2002.
11. Peter J. Burt and Edward H. Adelson. A multiresolution spline with application to image mosaics. *ACM Trans. Graph.*, 2(4):217–236, October 1983.
12. P. Cignoni, C. Montani, C. Rocchini, R. Scopigno, and M. Tarini. Preserving attribute values on simplified meshes by resampling detail textures. *The Visual Computer*, 15(10):519–539, 1999.

13. Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Trans. Graph.*, 22(3):287–294, 2003.
14. Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Computer Graphics (SIGGRAPH '97 Conf. Proc.)*, pages 361–368. ACM SIGGRAPH, Addison Wesley, 1997.
15. Paul E. Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin, and Mark Sagar. Acquiring the Reflectance Field of a Human Face. In *Computer Graphics (SIGGRAPH '00 Conf. Proc.)*, pages 145–156. ACM SIGGRAPH, 2000.
16. Mathieu Desbrun, Mark Meyer, and Pierre Alliez. Intrinsic parameterizations of surface meshes. *In Eurographics 2002 Conference Proceedings*, 21(3):209–218, 2002.
17. Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. *Computer Graphics (SIGGRAPH '99 Conf. Proc.)*, 33:317–324, 1999.
18. Iddo Drori, Daniel Cohen-Or, and Hezy Yeshurun. Fragment-based image completion. *ACM Trans. Graph.*, 22(3):303–312, 2003.
19. T. Duchamp, A. Certain, A. DeRose, and W. Stuetzle. Hierarchical Computation of PL harmonic Embeddings. Technical report, University of Washington, July 1997.
20. Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *Computer Graphics (SIGGRAPH '95 Conf. Proc.)*, pages 173–182, 1995.
21. A. Efros and T. Leung. Texture synthesis by non-parametric sampling. In *ICCV2-99*, pages 1033–1038, 1999.
22. Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. *Computer Graphics (SIGGRAPH '01 Conf. Proc.)*, pages 341–346, 2001.
23. Kurt Fleischer, David Laidlaw, Bena Currin, and Alan Barr. Cellular texture generation. In *Computer Graphics (SIGGRAPH '95 Conf. Proc.)*, pages 239–248, 1995.
24. Michael S. Floater. Parametrization and smooth approximation of surface triangulations. *Comp. Aided Geom. Design*, 14:231–250, 1997.
25. Michael S. Floater. Mean value coordinates. *Comp. Aided Geom. Design*, 20(1):19–27, 2003.
26. Michael S. Floater and Kai Hormann. Surface parameterization: a tutorial and survey. In *Advances on Multiresolution in Geometric Modelling*, pages 259–284. Springer-Verlag, 2004.
27. James Foley, Andries van Dam, Steven Feiner, and John Hughes. *Computer Graphics Principles and Practice, Second Edition*, chapter 20, pages 1020–1031. Addison Wesley, 1992.
28. David Donovan Garber. *Computational Models for Texture Analysis and Texture Synthesis*. PhD thesis, University of Southern California, 1981.
29. Ned Greene. Environment mapping and other applications of world projection. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.
30. Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder. Normal meshes. In *Computer Graphics (SIGGRAPH '00 Conf. Proc.)*, pages 95–102. ACM Press, 2000.
31. Steven Haker, Sigurd Angenent, Allen Tannenbaum, Ron Kikinis, Guillermo Sapiro, and Michael Halle. Conformal surface parameterization for texture mapping. *IEEE Trans. on Visualization and Computer Graphics*, 6(2):181–189, April/June 2000.
32. Paul Harrison. A non-hierarchical procedure for re-synthesis of complex textures. In *WSCG 2001 Conference Proceedings*, 2001.
33. David J. Heeger and James R. Bergen. Pyramid-Based texture analysis/synthesis. In *Computer Graphics (SIGGRAPH '95 Conf. Proc.)*, pages 229–238, 1995.
34. Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Computer Graphics (SIGGRAPH '01 Conf. Proc.)*, pages 327–340, 2001.
35. Anil N. Hirani and Takashi Totsuka. Combining frequency and spatial domain information for fast interactive image noise removal. *Computer Graphics (SIGGRAPH '96 Conf. Proc.)*, 30(Annual Conference Series):269–276, 1996.
36. Hugues Hoppe. Progressive meshes. In *Computer Graphics (SIGGRAPH '96 Conf. Proc.)*, pages 99–108. ACM SIGGRAPH, Addison Wesley, 1996.
37. K. Hormann and G. Greiner. Mips: an efficient global parametrization method. In P.-J. Laurent, P. Sablonnière, and L. L. Schumaker, editors, *Curve and Surface Design: Saint-Malo 1999*, pages 153–162. Vanderbilt University Press, 2000.
38. Takeo Igarashi and Dennis Cosgrove. Adaptive unwrapping for interactive texture painting. In *Proc. of the 2001 symposium on Interactive 3D graphics*, pages 209–216. ACM Press, 2001.
39. Homan Igehy and Lucas Pereira. Image replacement through texture synthesis. In *Proc. of IEEE International Conference on Image Processing*, volume 3, pages 186–189, 1997.
40. Kolja Kähler, Jörg Haber, Hitoshi Yamauchi, and Hans-Peter Seidel. Head shop: Generating animated head models with anatomical structure. In *Proc. of the 2002 ACM SIGGRAPH Symposium on Computer Animation*, pages 55–64, San Antonio, USA, 2002.
41. Andrei Khodakovsky, Nathan Litke, and Peter Schröder. Globally smooth parameterizations with low distortion. *ACM Trans. Graph.*, 22(3):350–357, 2003.
42. Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. In *Computer Graphics (SIGGRAPH '98 Conf. Proc.)*, pages 95–104, 1998.
43. W.-S. Lee and N. Magnenat-Thalmann. Fast Head Modeling for Animation. *Image and Vision Computing*, 18(4):355–364, March 2000.
44. Bruno Lévy. Constrained texture mapping for polygonal meshes. In *Computer Graphics (SIGGRAPH '01 Conf. Proc.)*, pages 417–424. ACM Press, 2001.
45. Bruno Lévy, Sylvain Petitjean, Nicolas Ray, and Jérome Maillot. Least squares conformal maps for automatic texture atlas generation. In *ACM Trans. Graph.*, volume 21, 3, pages 362–371, New York, 2002. ACM Press.
46. Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *Computer Graphics (SIGGRAPH '01 Conf. Proc.)*, 20(3):127–150, 2001.

47. Jérôme Maillot, Hussein Yahia, and Anne Verroust. Interactive texture mapping. *Computer Graphics (SIGGRAPH '93 Conf. Proc.)*, pages 27–34, 1993.
48. Stephen R. Marschner, Stephen H. Westin, Eric P. F. Lafortune, Kenneth E. Torrance, and Donald P. Greenberg. Image-Based BRDF Measurement Including Human Skin. In *Proc. of 10th Eurographics Workshop on Rendering*, pages 131–144, 1999.
49. Kazunori Miyata. A method of generating stone wall patterns. In *Computer Graphics (SIGGRAPH '90 Conf. Proc.)*, pages 387–394. ACM Press, 1990.
50. Andrew Nealen and Marc Alexa. Hybrid texture synthesis. In *Proc. of the 13th Eurographics workshop on Rendering*, pages 97–105. Eurographics Association, 2003.
51. Peter J. Neugebauer and Konrad Klein. Texturing 3d models of real world objects from multiple unregistered photographic views. *Eurographics*, 18(3):C245–C256, 1999.
52. Byong Mok Oh, Max Chen, Julie Dorsey, and Frédo Durand. Image-based modeling and photo editing. In *Computer Graphics (SIGGRAPH '01 Conf. Proc.)*, pages 433–442. ACM Press, 2001.
53. Manuel M. Oliveira, Brian Bowen, Richard McKenna, and Yu-Sung Chang. Fast digital image inpainting. In *Proc. of the International Conference on Visualization, Imaging and Image Processing (VIIP 2001)*, pages 261–266, September 3-5 2001.
54. Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, 2003.
55. Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-12(7):629–639, July 1990.
56. Frédéric Pighin, Jamie Hecker, Dani Lischinski, Richard Szeliski, and David H. Salesin. Synthesizing realistic facial expressions from photographs. In *Computer Graphics (SIGGRAPH '98 Conf. Proc.)*, pages 75–84, 1998.
57. Dan Piponi and George Borshukov. Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In *Computer Graphics (SIGGRAPH '00 Conf. Proc.)*, pages 471–478, 2000.
58. Kris Popat and Rosalind W. Picard. Novel cluster-based probability model for texture synthesis, classification, and compression. In *Proc. SPIE Visual Communications and Image Processing*, volume 2094, pages 756–768, 1993.
59. Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. *ACM Trans. Graph.*, 2003.
60. C. Rocchini, P. Cignoni, C. Montani, and R. Scopigno. Multiple textures stitching and blending on 3d objects. In *10th Eurographics Workshop on Rendering*, pages 127–138. Granada, June 1999.
61. Philippe Saint-Marc, Jer-Sen Chen, and Gérard Medioni. Adaptive smoothing: a general tool for early vision. *IEEE Trans. on Pattern analysis and machine intelligence*, 13(6):514–529, June 1991.
62. Pedro V. Sander, Steven J. Gortler, John Snyder, and Hugues Hoppe. Texture mapping progressive meshes. In *Computer Graphics (SIGGRAPH '01 Conf. Proc.)*, 2001.
63. Pedro V. Sander, Steven J. Gortler, John Snyder, and Hugues Hoppe. Signal-specialized parametrization. In *Proc. of the 13th Eurographics Workshop on Rendering*, pages 87–98, June 26–28 2002.
64. Claude Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, July 1948.
65. Alla Sheffer and Eric de Sturler. Parameterization of faceted surfaces for meshing using angle based flattening. *Engineering with Computers*, 17(3):326–337, 2000.
66. Alla Sheffer and John C. Hart. Seamster: inconspicuous low-distortion texture seam layout. In *Proc. of IEEE Visualization '02*, pages 291–298, 2002.
67. E Simoncelli and J Portilla. Texture characterization via joint statistics of wavelet coefficient magnitudes. In *Fifth IEEE Int'l Conf on Image Proc*, volume I, Chicago, October 4-7 1998. IEEE Computer Society.
68. S. M. Smith and J. M. Brady. SUSAN – A new approach to low level image processing. Technical Report TR95SMS1c, Defence Research Agency, Chertsey, Surrey, UK, 1995.
69. Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical pattern mapping. In *ACM Trans. Graph.*, pages 673–680. ACM Press, 2002.
70. Olga Sorkine, Daniel Cohen-Or, Rony Goldenthal, and Dani Lischinski. Bounded-distortion piecewise mesh parameterization. In *Proc. of IEEE Visualization '02*, pages 355–362, 2002.
71. Marc Soucy, Guy Godin, and Marc Rioux. A texture-mapping approach for the compression of colored 3d triangulations. *The Visual Computer*, 12(10):503–514, October 1996.
72. Martin Szummer and Rosalind W. Picard. Temporal texture modeling. In *ICIP*, volume 3, pages 823–826, Lausanne, Switzerland, 1996.
73. Marco Tarini, Hitoshi Yamauchi, Jörg Haber, and Hans-Peter Seidel. Texturing faces. In *Graphics Interface 2002*, pages 89–98, Calgary, Canada, May 2002. Canadian Human-Computer Communications Society, A K Peters.
74. Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *ICCV-98*, pages 839–846, January 4–7 1998.
75. Greg Turk. Generating textures on arbitrary surfaces using reaction-diffusion. In *Computer Graphics (SIGGRAPH '91 Conf. Proc.)*, pages 289–298, 1991.
76. Greg Turk. Texture synthesis on surfaces. In *Computer Graphics (SIGGRAPH '01 Conf. Proc.)*, pages 347–354, 2001.
77. Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Computer Graphics (SIGGRAPH '00 Conf. Proc.)*, pages 479–488, 2000.
78. Li-Yi Wei and Marc Levoy. Texture synthesis over arbitrary manifold surfaces. In *Computer Graphics (SIGGRAPH '01 Conf. Proc.)*, pages 355–360, 2001.
79. Hitoshi Yamauchi, Jörg Haber, and Hans-Peter Seidel. Image restoration using multiresolution texture synthesis and image inpainting. In *Proc. of Computer Graphics International*, pages 120–125, Tokyo, Japan, 2003.
80. Lexing Ying, Aaron Hertzmann, Henning Biermann, and Denis Zorin. Texture and shape synthesis on surfaces. In *Eurographics rendering workshop*, pages 301–312, 2001.
81. Rhaleb Zayer, Christian Rössl, and Hans-Peter Seidel. Variations of angle based flattening. In *Advances in Multiresolution for Geometric Modeling*. Spinger-Verlag, 2004.

82. Steve Zelinka and Michael Garland. Towards real-time texture synthesis with the jump map. In *Proc. of the 13th Eurographics workshop on Rendering*, pages 101–107, 2002.
83. Jingdan Zhang, Kun Zhou, Luiz Velho, Baining Guo, and Heung-Yeung Shum. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Trans. Graph.*, 22(3):295–302, 2003.