Visual Comput (2005)
DOI 10.1007/s00371-005-0283-5

ORIGINAL ARTICLE

Textures revisited

Hitoshi Yamauchi
Hendrik P. A. Lensch
Jörg Haber
Hans-Peter Seidel

Published online: 12 May 2005

Hitoshi Yamauchi · Jörg Haber · Hans-Peter Seidel
MPI Informatik, Saarbrücken, Germany,

Hendrik P. A. Lensch
Stanford University, USA

Abstract We describe texture generation methods for complex objects. Recently developed 3D scanning devices and high-resolution cameras can capture the complex geometry of an object and yield high-resolution images. However, generating a textured model from this input data is still a difficult problem.
This task is divided into three subproblems: 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 using our new texture combination method.
We also demonstrate a texture restoration method that 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 for the registration process between a 3D model and 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 the texture of a complex object.

Keywords 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 used 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 *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 to scan 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 in order 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 into three subproblems:

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 to synthesize a texture.

The example given in this paper of an application of texture generation is the generation of 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 of sufficient quality.

The main contributions presented in this paper are as follows:

- For parameterization, we present
  1. a method to temper the discontinuity effect in geometric stretch parameterization, and
  2. a view-dependent parameterization method for the effective use of texture memory.
- In texture combination from several photographs, we present a visibility-aware multiresolution 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 Sect. 2. Next, we detail each subproblem: parameterization (3), texture combination from several photographs (Sect. 4), and texture restoration (Sect. 5). We show some results in Sect. 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 face two problems: parameterization and texture image generation. The texture image generation problem is also subdivided into two subproblems. One is texture combination. Texture combination is the process of making a single texture from several input photographs. The other problem is texture restoration, which is needed when the input photographs do not provide enough information to cover 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 early days of the technology, due to hardware limitations, texture mapping was applied to relatively simple polygonal meshes only. Manually creating a parameterization was therefore acceptable.

However, recent graphics hardware can display larger and more complex polygonal meshes with higher-resolution texture images than before. It has become essential 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 of these charts.
- Energy function: the basic idea to solving the parameterization problem is to minimize an 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 nonlinear.
- Boundaries: for effective use of hardware resources, a texture should fit into texture memory and should have a square shape. On the other hand, this restriction may be too stringent for low-distortion parameterization. Boundary conditions on the texture image establish the weight of each restriction.

2.1.1 Texture atlas

The traditional approaches of parameterization are to cut a mesh into *charts* and pack them into 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 using our preferred parameterization method.

Once a texture atlas is introduced, it becomes relatively easy to keep the distortion low inside 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 charts, and (3) it introduces artifacts while MIP-mapping.

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

Let us first focus on objects that 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 proposed a technique to map an isoparametric curve of a 3D surface onto curves of a plane with two distortion metrics [6]. Their energy function is based on 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 to satisfy geodesic curvature preservation.

Maillot et al. defined distance energy and surface energy for parameterization [47]. The distance energy tries to minimize distortion, while on the other hand, it may introduce triangle flipping. To avoid this, a nonlinear surface energy term is introduced. This surface energy term helps to reduce 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. proposed the concept of 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 nonlinear systems. However, triangle flipping might occur with this method as well.

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

The conformal surface parameterization was introduced by Haker et al. [31]. We can find interesting similarities with Duchamp et al. citebib:Duchamp1997 in a remeshing context, and with Desbrun et al. [17] in a fairing context. 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 establishes some criteria for calculating the optimal λ and to obtain the natural boundary. At almost the same time, Levy et al. citebib:Levy:2002 presented least squares conformal maps. The parameterization is the same as in 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] is the same, and the main difference is 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 nonlinear 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 the linear and nonlinear methods are similar.

Nonlinear energy: Hormann and Greiner proposed the MIPS (Most Isometric ParametricS) parameterization [37]. They attempted 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. introduced the root-mean-square of the singular values of the matrix, called the 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 introduced the idea of signal-stretch energy to improve the quality of the parameterization with respect to a given texture [63], color, or normals.

Balmelli et al. proposed space-optimized texture maps [4]. They used the frequency information of the input image as a signal. The difference from the previous

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

approach [63] is that they applied 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 of these methods [4, 37, 62, 63] generate high-quality parameterizations for texture maps; however, they are based on time-consuming nonlinear optimization.

In order to speed up the calculation, Sander et al.[63] used a multiresolution approach. First, they calculated the energy on a coarse mesh. Then, they sequentially added vertices similar to progressive meshes [36] and again minimized 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 of *flattening*. Sheffer and de Sturler [65] proposed a parameterization method that minimizes an angle distribution error around each vertex. The error criterion is linear, but they introduced several nonlinear constraints to avoid unwanted situations, e.g., boundary intersections. Later, Zayer et al. [81] improved the computational cost and introduced a boundary shape controlling factor. Sorkine et al. [70] proposed 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 methods is that there is no limitation of a predefined boundary condition, e.g., fixed boundary, convex boundary.

Table 1. Classification of parameterizations. The column labeled "Atlas" shows whether generating an atlas is necessary or not. "NB" stands for "natural boundary." If this column is labeled Y(es), it is possible to minimize energy while 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
Nonlinear 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	–

Other interesting methods: Igarashi proposed 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. The signal to choose or the importance given to each is defined by the user. When the size of the painted area is relatively small, this method works well, since only the user-specified important parts are parameterized by the projection.

Piponi and Borshukov proposed a cutting and blending technique for textures on a 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, whether the system is linear or whether the energy is globally defined. Further considerations are the topological limitations that 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 a kind of classification. In this table, when the column Atlas is labeled Y(es), it means that this method needs to cut the input mesh during the parameterization process. Therefore, when the column is labeled is N(o), the 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, some 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 methods since the atlas-generation is at the preprocessing stage.

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

[3] The geometric-stretch energy [62] is defined as the root-mean-square of the singular values of the mapping matrix. The method [70] uses a modified geometric stretch energy.

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

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

2.2 Texture generation

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

Soucy et al. proposed 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, such as area, largest edge length, and so on. Each triangle is independently inserted onto the texture map. They are packed as half-square triangles into the 2D texture. The texture atlas generated by this method contains one separate chart for each triangle.

Rocchini et al. proposed an approach for mapping and blending the texture on a 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 misregistration is accounted for by automatic local feature alignment.

Later, they proposed preserving attribute detail on 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 conditions, 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. proposed 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.

[4] This method is for making an atlas; the parameterization method used for parameterizing each chart is unimportant.

While the artifacts can be eliminated by these methods, these techniques also destroy image detail in the boundary area. In order to keep more detail, Burt and Adelson [11] proposed a multiresolution spline technique. Multiresolution 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 detail in 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] introduced 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. presented 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 denoising, image segmentation, image enhancement, and so forth.

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

Oliveira et al. proposed a simpler and faster inpainting method [53]. They assumed 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. presented an 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 gradient-orientation term.

Pérez et al. proposed a Poisson image editing method [54]. Using this method, images are edited by solving Poisson equations with respect to a user-prescribed guid-

ance vector field under given boundary conditions. This is a versatile editing method. Some of the applications of this method, in particular, in seamless image insertion and feature exchange, are useful for image editing.

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. Texture synthesis can also be used to fill in holes in image restoration. In this context, the word "texture" is defined as an image that has some spatial coherence.

Procedure-based texture synthesis: Texture synthesis started with procedural texture generation, like a checker board. The basic procedure-based texture synthesis method describes a 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., that presented by Miyata [49].

Statistical texture analysis and synthesis: Gaber proposed many basic methods for texture synthesis [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. His best-fit model has since been rediscovered and is now called the texture synthesis method [22].

Popat et al. presented a cluster-based probabilistic modeling technique for high-dimensional vector sources [58]. A vector is constructed by concatenating a pixel and the 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 to synthesize a texture. They also proposed a hierarchical texture synthesis method using multiresolution analysis.

Heeger and Bergen proposed a pyramid-based texture analysis/synthesis method [33]. The input texture is analyzed using a multiresolution 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 a similar histogram distribution through all 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 presented a texture synthesis method based on statistical measurements of the seed image and its wavelet decomposition [67]. The authors applied higher-order statistics while the previous methods [14, 33] considered 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 that satisfies a hierarchical statistic model. Using wavelets, they construct a tree representing a hierarchical multiscale transform of the input signal. A new texture is synthesized by tracing this tree according to a similarity path. The approach allows for the mixing of different input textures by mixing their corresponding trees. The method has been successfully applied to 1- and 2-D signals like sound sequences and images, and it will be extended to process video sequences. The presented results are quite impressive compared with other statistical methods.

Some statistical texture synthesis models also expand to the temporal domain; namely, synthesizing video sequences. Szummer and Picard proposed such a model [72]. They assumed that 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 considered 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 proposed a non-parametric sampling method that creates a lookup 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 presented an acceleration method of the non-parametric texture synthesis using tree-structured vector quantization [77]. They applied the causal kernel [58] (called "best-fit model kernel" in [28]) in 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, multiresolution texture synthesis is considered. They also proposed 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 considered together with the already synthesized pixels during the search for a similar subimage in the seed texture. The coherent match method is rather efficient.

Zelinka et al. proposed a real-time texture synthesis version of the coherent map [82]. They first analyzed the input image to make a similar subimage link map from each pixel, called a 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. presented an image-processing framework by example, called "image analogies" [34]. This filter makes a mapping from source image A to destination image A′, and applies 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. proposed searching in the frequency domain [69]. They reinterpreted the 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 vector typically contains 10 to 100 pixels. 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 proposed 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 a minimal error bounding cut.

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

A patch-based technique for image and texture synthesis was 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 proposed a stochastic tiling method that can generate a non-periodic pattern and a seamless Wang tile seed-generation method applying 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 to generate a non-periodic texture.

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

Nealen et al. proposed a hybrid method combining patch-based methods with pixel-based methods [50]. The patch similarity is calculated as described in Soler and Angelidis [69]. Then, they calculated the error of the surrounding patch boundary. Each pixel with too large an 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 methods use the L_2 norm as a similarity measure. Harrison used an entropy measure with Manhattan distance (L_1) to calculate the similarity measure [32]. He stated that the L_2 norm emphasizes outliers. In addition, he introduced a constraint map to handle non-homogeneous textures. The user marks an importance map containing weights for each source pixel. Harrison demonstrated that some of the non-homogeneity in the texture can be handled with this importance map.

Another subimage similarity metric was proposed by Brooks and Dodgson [10] for the editing of 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 homogeneity in the source texture, and they use a similarity search with relatively small kernels. Therefore, it is hard to capture large structures that are introduced by the depth of a scene

or by shading effects. Multiresolution analysis to some extent alleviates these effects [34, 77].

To overcome the limitation caused by large structures in the image, Oh et al. separated the image into 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 keeping sharp edges. Therefore, this filter suits texture editing with relighting.

Frequency domain transfer: Hirani and Totsuka proposed an image restoration algorithm that can handle both texture and intensity variations [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 alternately carried out 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 a translated version of the defective region with respect to its texture. Care is taken to automatically adapt intensity variations between the sample and the defected regions.

Texture synthesis on a surface: Traditionally, texture synthesis has been carried out on two-dimensional images; however, recently, there have been several texture synthesis methods proposed 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 generated 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 employed a user-guided vector field on the surface. In addition, they used a *texton* mask to keep features of the seed texture and resample pixels to achieve rotation and scaling 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, the method proposed by Lee et al. [42].

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

Drori et al. proposed 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 a wrong pixel being selected can lead to a fatal visual effect in the restoration. To avoid this binary decision effect, Drori et al. introduced 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 an incorrect 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 more 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 keeps both conformality and low area distortion. Assume a parameterization map M1 from S to S' of triangle mesh. There exists another map M2 for area of each triangle i, $A_{T_i} \rightarrow A'_{T_i}$. The distortion of M2 called area distortion. So, it could not be changed. However, this method needs an initial valid parameterization, and it is usually time-consuming, since it minimizes a nonlinear 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 Fig. 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.

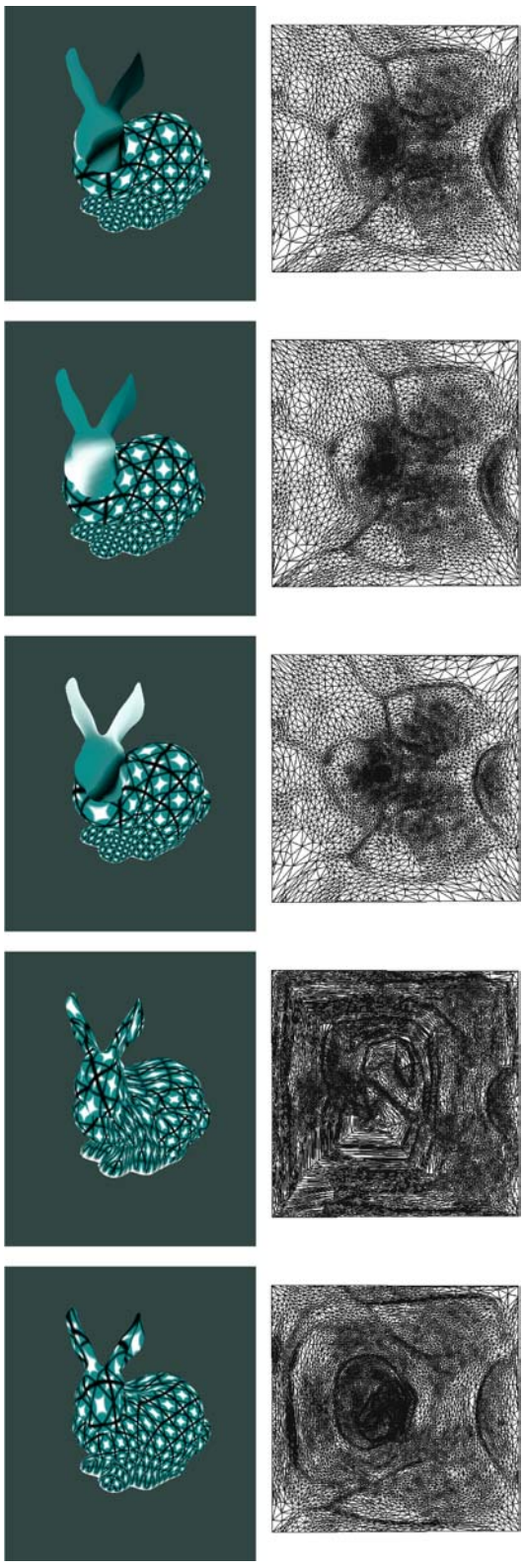


Fig. 1. Comparison of parameterization methods. From top to bottom, 1. Floater [24], 2. Intrinsic [16] ($\lambda = 0.5$), 3. MIPS [37], 4. L^∞ geometric stretch [62], 5. L^2 geometric stretch [62]

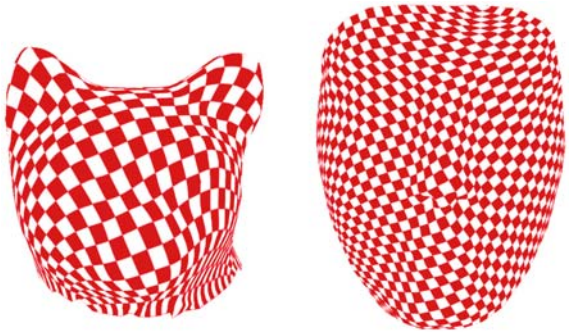


Fig. 2. Well parameterized cat head and forehead obtained by L^2 geometric stretch parameterization

Figure 3 shows the results of rather geometrically 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 Fig. 3. For example, both the Stanford bunny model and the mannequin head model have several cracks on the backside of their necks. These cracks are referred to as "parameter cracks."

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

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

where p and ϵ are user-defined parameters, $A'(T_i)$ 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 the 3D domain to the 2D domain. This regularization term tries to punish triangles that have large singular values more than those only using L^2 geometric stretch energy. Instead of applying this L^p term to an inverse stretch calculation as in [59], we applied this term to a planar domain in the same way. However, the results in Fig. 4 show that although this term alleviates the effect of parameter cracks a little, it introduces other unwanted distortions. Moreover, the L^p term requires two unintuitive 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 Fig. 4, we test several parameters starting with the recommended values in Praun and Hoppe [59] ($\epsilon = 0.001$, $p = 6$). In this experiment, the area normalization coefficient 4π of L^p term in Eq. 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

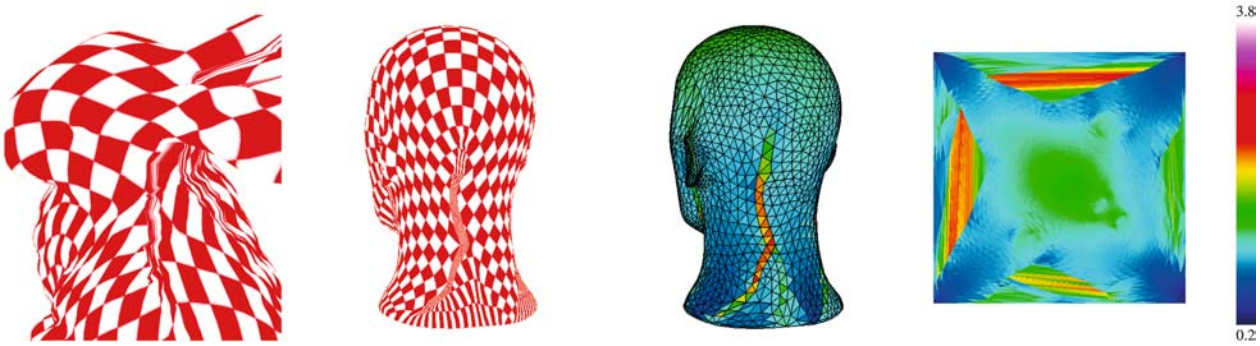


Fig. 3. The two images on the left show examples of parameter cracks on a 3D textured model. Notice that there is a large distortion around the neck part. The two images on the right show the distribution of distortion for the mannequin model both on the 3D model and on its 2D parameterization. The distortion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

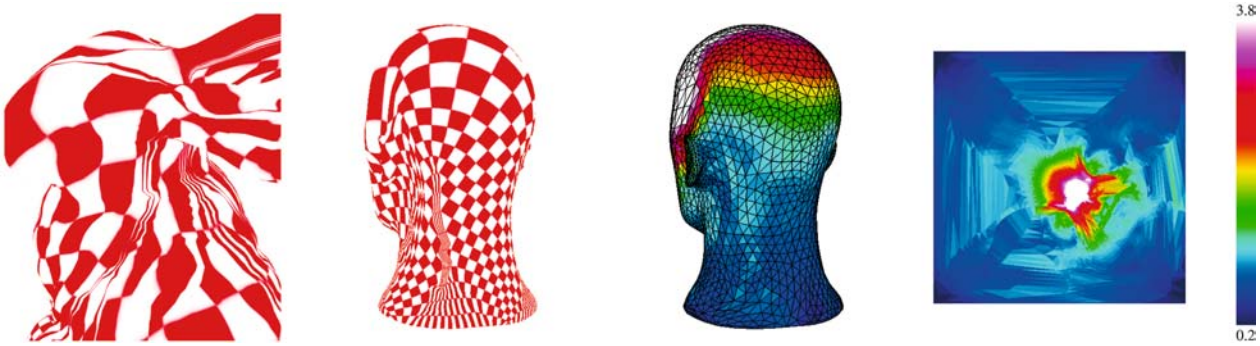


Fig. 4. Results of an $L^2 + L^6$ parameterization. For the bunny model, a choice of $\epsilon = 0.01$, $p = 6$ results in $L^2(\text{bunny}) = 3.01 \times 10^4$, and for the mannequin model, $\epsilon = 0.001$, $p = 6$ gives a total distortion of $L^2(\text{mannequin}) = 1.07 \times 10^4$

parameterization, unlike [59], which uses remeshing. It might be more effective to achieve low distortion by combining parameterization with a regularization term and remeshing. The paper by Praun and Hoppe [59] shows a good result of this combination if a remeshing is possible. In the next section, we will also discuss avoiding the parameter crack effect without changing the 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 Eq. 2. One is to increase the effective use of the texture area, and the other is to avoiding the parameter crack effect.

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

with

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

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

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 the 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

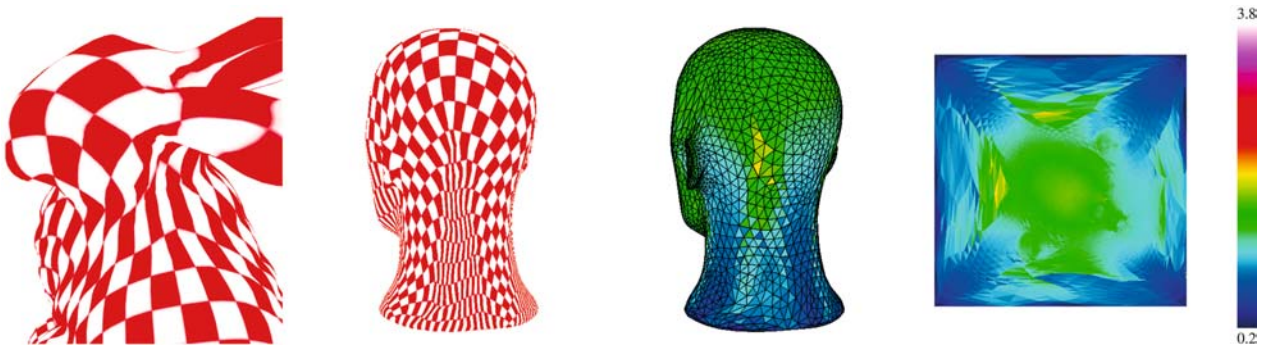


Fig. 5. Influence of the triangle shape term $s(T_i)$ on the distortion. Compared to Figs. 3 and 4, parameter cracks are alleviated. Here, $L^2(\text{bunny}) = 4.40 \times 10^4$, and $L^2(\text{mannequin}) = 7.58 \times 10^3$

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 back-facing triangles become smaller. We call this a view-dependent parameterization, since parameterization is dependent on the view deemed to be most important. The user can control the degree of ω's influence on 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 ω is considered as another signal in Sander's signal-based method [63].

Figure 6 shows view-dependent parameterization results for various view directions V. In this figure, we use the energy function of Eq. 3 without the $s(T_i)$ term to clarify the effect of the visual importance function alone. 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, for example, 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 retains conformality and a low area distortion. We also assume the h/b ratio parameter to be more intuitive than the L^p energy in Eq. 1, since the h/b ratio is more directly coupled to the triangle shape. One large difference compared tothe method presented in Tarini et al. [73] is that our method can control this parameter crack effect.

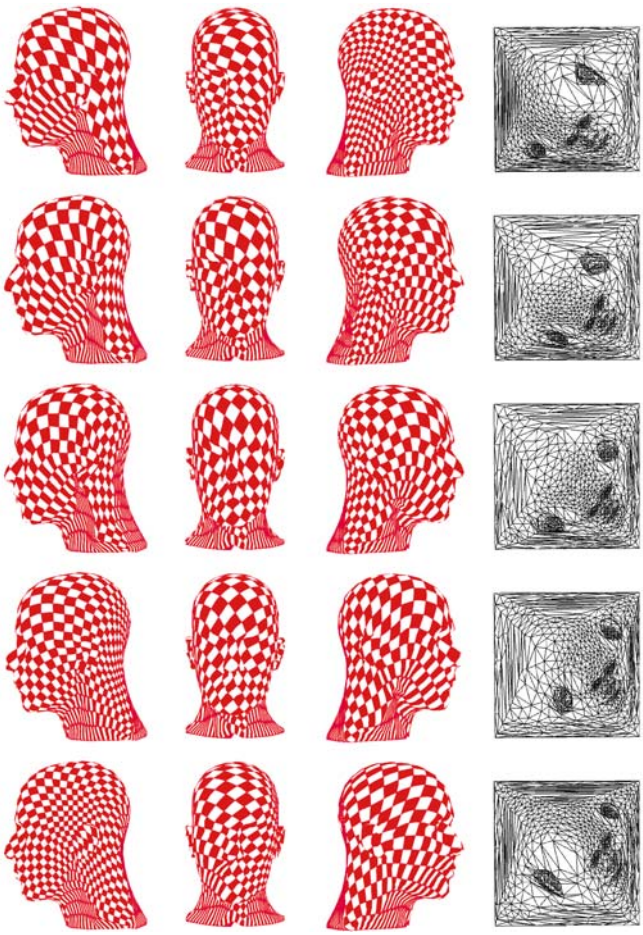


Fig. 6. The effect of the visual importance function $\omega(T_i)$. From top to bottom, the view direction V is changing from the left side of the face to the right side. In each row, from left to right, we see the left side, front, and right side of the mannequin head model, and the parameterization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

Introducing the $s(T_i)$ term makes the total distortion energy higher than the original L^2 energy (see Figs. 3 and 5); however, this term equalizes the distortion distribution and achieves the lowest number of 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 Rocchini et al. [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 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 Rocchini et al. [60]. A vertex v is visible in a photograph if the projection of v on the image plane is contained in the photograph, 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. In contrast to the approach in Rocchini et al. [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 to classify 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 the 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 the input image. First, we calculate the 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. In addition, we use Gaussian convoluted pixel samples with a 3×3 subimage mask, since we do not rely on a single pixel sample. If the distance is larger than a given threshold value, then the vertex is reclassified as an unbound vertex.

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

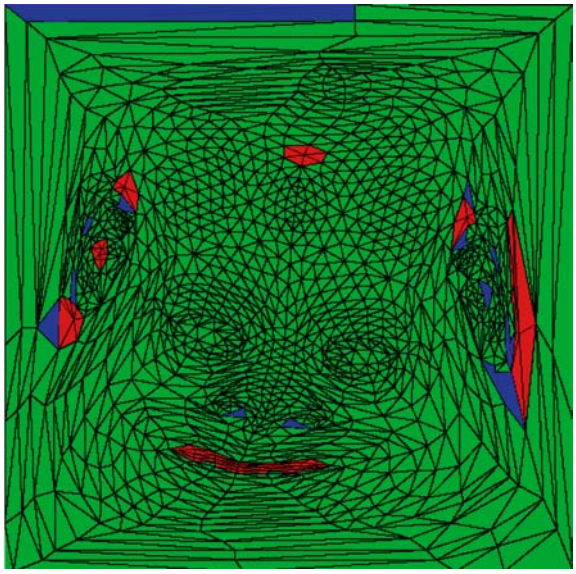


Fig. 7. Color-coded triangles of the textured mesh: each green triangle has at least one common photograph to which all of its vertices are bound; the vertices of blue triangles don't have a common photograph, but they are all bound; red triangles have at least one unbound vertex

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 Δ (green triangles).
2. All of the vertices of Δ are bound to at least one photograph, but no common photograph can be found for all three vertices (blue triangles).
3. At least one vertex of Δ is not bound to any photograph (red triangles).

In the first case, we rasterize $\tilde{\Delta}$ in texture space. For each texture element (texel) T, we determine its barycentric coordinates ρ, σ, τ with respect to $\tilde{\Delta}$ and compute the corresponding normal N by interpolating the vertex normals of Δ: $N = \rho N(v_1) + \sigma N(v_2) + \tau N(v_3)$. For each common photograph i in the sets of valid photographs of all vertices of Δ, 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 $\tilde{\Delta}$ individually by summing up the weighted pixel colors of the corresponding pixels in all valid photographs i of $\tilde{v}_j$ 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 $\tilde{\Delta}$ 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 Δ share a photograph and the vertices v_2, v_3 share another photograph. However, this situation always happens near the

silhouette of an object, and the extrapolation of a missing vertex on the photograph will be unstable. Neugebauer and Klein [51] mention a similar situation. They recommend the use of this kind of uv coordinate extrapolation, since at least the boundary has plausible color, and this is better than just a hole. We tried this extrapolation and then performed our registration error detection scheme, and we found that it fails in most cases. Therefore, the plain color interpolation from reliable vertices usually produces much better results in our case.

Since we do not require that each vertex of the face mesh be 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 $\tilde{\Delta}$ 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 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 $\tilde{\Delta}$ can be colored.

This color interpolation method is fast and easy to implement, and it can fill in 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 Sect. 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 multiresolution spline technique as proposed in Burt and Adelson [11] and Lee and Magnenat-Thalmann [43] to remove visual boundaries. Figure 8 shows a comparison between a textured head model with and without the multiresolution spline technique applied. The multiresolution spline technique needs a mask to determine the overlapping region that 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 the combination. Then, we remove the outmost ring of triangles around the region (see Fig. 9). Such 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.

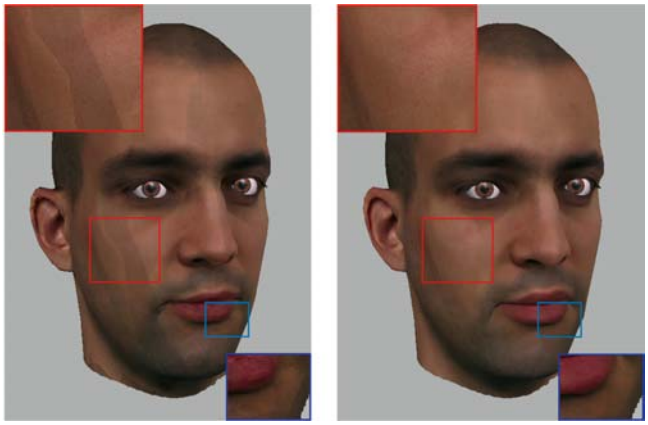


Fig. 8. Boundaries in the skin texture (left) are removed using multi-resolution spline techniques (right)

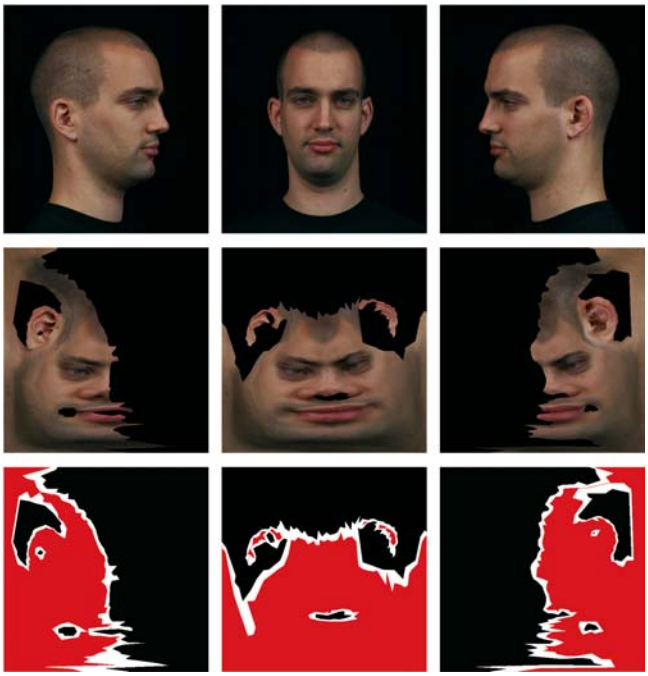


Fig. 9. Multi-resolution spline masks: input photographs from three different view points (top), texture meshes resampled from their corresponding photographs (middle), and their automatically generated masks shown in red (bottom)

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 simpler and more straight forward than pixel-based resampling.

For our validity test, we need to separate the background and foreground of the input photographs. We apply a median filter and Gaussian denoising filter for the separation. However, from our experience, pixel-based validity tests are rather unreliable. Moreover, the calculation of the 3D-to-2D backprojection near the object silhouette becomes inaccurate since the dot product of the normal of the 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 rather simple geometrically, 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 of 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 photographs (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 multiresolution 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 Sect. 5 since this mask indicates if each pixel is resampled or not.

5 Texture restoration

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

If we can assume that our object being reconstructed is easy to access and that it is static, we can acquire additional photographs and register them until all pixel information is given by the photographs. However, some objects like a human face are not static, and it is difficult to take photographs under the same conditions, e.g., the same lighting conditions. Moreover, some objects are difficult to access – for instance, historically valuable objects or already lost objects – although several photographs of these objects may remain. In these cases, if the total number of missing pixels is relatively small, we should consider reconstructing these missing pixels from given incomplete information.

So far, there exist two main approaches to fill in missing pixels: image inpainting and texture synthesis. After giving an overview of these methods, we 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 cases, this partial differential equation takes a Laplacian operator to "diffuse" surrounding pixels into the missing pixels.

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

Both methods work well in missing areas that are relatively small or thin, like scratches, blotches, and areas of text. However, a problem arises when the missing area is large. Because these methods fill missing area with a continuous function by solving a 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 in a 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 equalization, 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 result in resolution-free images. But it is usually difficult to find such a function.

We first deal with an arbitrary input; stochastic texture synthesis methods are proposed to synthesize 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 measurements (e.g., mean intensity value, or the histogram distribution of the image). Then, all we need is simply a sample 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 parameter [14, 33].

A non-parametric sampling texture synthesis method is demonstrated in Fig. 10. This method gets an input source image and generates a similar-looking destination image as an output. In the texture synthesis process, a subimage N_{dest}, called the "neighborhood kernel," is extracted from the destination image, and a similar kernel N_{src} is located in the source image, which then determines the destination pixel color. The kernel shape in Fig. 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 Fig. 10. Several criteria 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 the reference point is transferred 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 the image by using the kernel. The problem arises when the algorithm 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 the image. When a large 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 cannot extract enough sampling information in the fixed-size data source when the dimension of the sampling kernel becomes larger. To overcome this limitation, multiresolution searching [77], coherent matching [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.

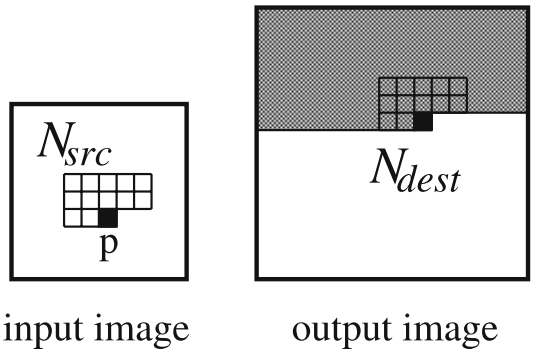


Fig. 10. Non-parametric sampling texture synthesis. The black pixel "p" is the reference pixel of kernel N. The gray region of the output image is the completed region

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

- **Image inpainting method:** the height field is reconstructed with 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 cannot usually 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 appear to be complementary; however, they are not based on the same mathematical theory: one is based on a partial differential equation, and the other is based on a non-parametric search. In order to combine them, we need further study.

Let us consider these properties in a signal-processing context:

- **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 the image, 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., a Fourier transform. Then, the

lower-frequency part is reconstructed by the image inpainting method, and the higher-frequency part is reconstructed by the non-parametric sampling texture synthesis method. Finally, both reconstructed images are combined to obtain the overall result. Here, we will explain an enhanced method based on [79], which includes a rotation-invariant search method and a shrink order reconstruction method. In Oh et al. [52], the authors used 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, for relighting images. However, for our purposes, 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.

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 at the end of Sect. 4. However, the proposed image restoration algorithm is independent of how the image is created and how 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 (see Fig. 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) (see Sect. 5.2).
2. The fast image inpainting algorithm proposed in Oliveira et al. [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 lookup is built [1]. However, the search space for texture synthesisin level H_i not only contains the non-masked areas of H_i, but also 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* ([77]),

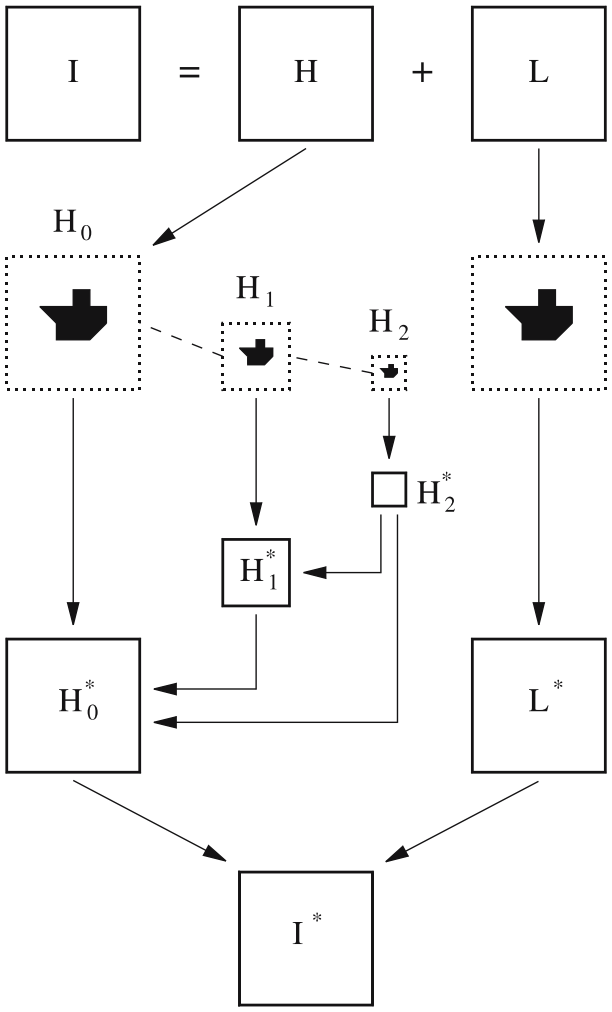


Fig. 11. Overview of our method. Top to bottom: the input image I is decomposed into a high-frequency image H and a low-frequency image L using a DCT. Image inpainting is applied to the low-frequency part L to obtain the inpainted image L^*. The high-frequency part H is decomposed into a Gaussian pyramid (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 (see [77]) is composed of the kernels of that level and of all higher levels. In this way, coherence 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^*

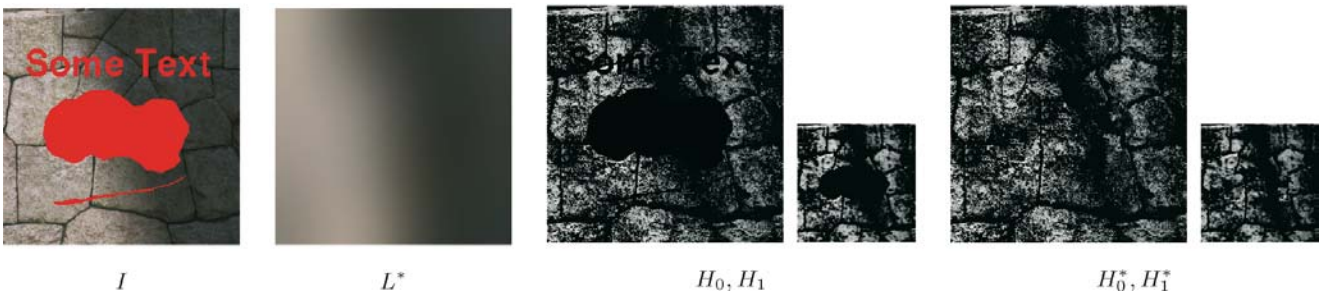


Fig. 12. Multi-resolution texture synthesis. Left to right: input image I; inpainted low-frequency image L^*; two levels (H_0, H_1) of the Gaussian decomposition of the high-frequency image H; the same two levels after texture synthesis (H_0^*, H_1^*). The detailed images H_i and H_i^* ($i = 0, 1$) are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

which is used to perform a lookup 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 coherence among all texture synthesis levels.

More details about this texture synthesis step are given in Sect. 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 next 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 subbands using a discrete cosine transform (DCT).

We select the first κ subbands 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 (i.e., as much high-frequency information) 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 as follows:

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

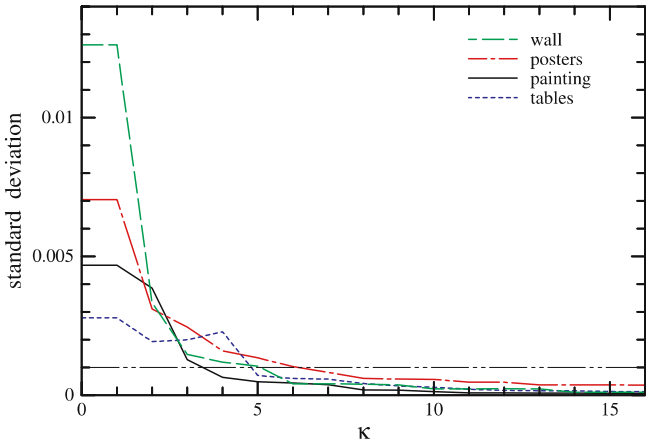


Fig. 13. Standard deviations of the autocorrelation matrices A_κ of the input images shown in Fig. 17 plotted over the range of $\kappa = 1, \ldots, 16$

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

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 the autocorrelation matrices of the input images shown in Fig. 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 Fig. 16. When κ goes to zero, our method is identical to the texture synthesis method, which is shown in Fig. 15. For large values of κ, our method is similar to the inpainting method, since the extracted higher frequency part becomes negligible.

[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 the input image.

Therefore, there might be an optimal κ value that can extract enough large structure as a low-frequency part, but can 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, are usually 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 an approach proposed by Burt and Adelson [11]. In particular, we employ the proposed 5×5 Gaussian kernel ω 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 Burt and Adelson [11] requires that the input image have 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 Sect. 5.1), we implement and test the approaches presented by Efros and Leung [21], Wei and Levoy [77], and Ashikhmin [2]. Finally, we implement the approach proposed in Hertzmann et al. [34], which basically switches between the texture synthesis algorithms from Wei and Levoy [77] and Ashikhmin [2] from level to level, depending on a local distance criterion. During the texture synthesis in level H_i, we typically use the 5×5 best-fit/causal kernel from Garber [28] and Wei and Levoy [77] within H_i, a standard 3×3 kernel for level H_{i+1}, and a 1×1 kernel for the higher levels H_k $(k = i + 2, \ldots, n)$.

Multiresolution 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 that 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 Sect. 5.3). This operation is carried out using 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 with a single-precision floating point.

Image reconstruction order: The reconstruction pixel order usually has an influence on the result [18, 32], except for highly homogeneous texture [78]. Also, when the kernel shape is not symmetric (e.g., for a bestfit/causal kernel), the orientation of the kernel affect the result.

This is why we implemented multiple reconstruction orders: scanline order type 1 (one way; $+x$ direction), scanline order type 2 (alternating ways; $+x$ and $-x$ directions), and shrinking order. The "shrinking" order fills a hole in a manner such 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 that 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 within the boundary, we cannot 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 causal/causal kernel is used to find a similar kernel, this kernel sometimes does not fit the hole boundary, since it is designed for scanline order reconstruction. Therefore, we use eight different orientations of the causal/causal kernel (see Fig. 14). During the image

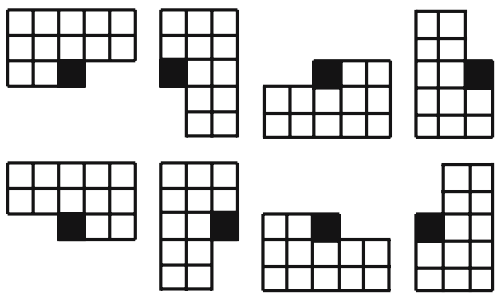


Fig. 14. Eight different orientations of the 5×5 best-fit/causal kernel. The reference point is the filled square

analysis phase (refer to Sect. 5.1, step 4.1), all eight of these kernels are used to analyze the source image and eight kD-trees are generated. In the reconstruction phase (refer to Sect. 5.1, step 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 a texture that is synthesized in a scanline order fashion. Then all possible holes would be filled if the top left 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 we have at least a non-missing subimage that fits the neighborhood kernel. We think this assumption is reasonable, since the neighborhood kernel is usually 5×5 or a similar size. However, we usually need more non-missing pixels to get a better result in practice.

Another advantage of using 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. A kernel of arbitrary shape is more powerful and can exploit the all non-missing pixel information. However, this cannot fit the kD-tree search algorithm since you cannot query a vector by changing its size and structure. Furthermore, it is not practical to generate all kD-trees corresponding to all patterns and store them in memory, because the total number of kernels of arbitrary shape is the sum of the binomial function. Some methods [18, 50] use a kernel of arbitrary shape 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 Wei and Levoy [77]. Both texture synthesis and image inpainting treat small scratches and thin areas well, i.e., text regions. Limitations of texture synthesis and image inpainting arise when large areas are missing. This point is also stressed in Drori et al.[18], where the texture reconstruction from real objects is investigated. In this problem, some continuous missing area – rather than small scratch or blotches – usually exists. Because of this, we need a method to reconstruct large structures and small details.

Our algorithm is controlled by two different parameters: the number of DCT subbands (κ), from which the low-frequency image L_κ is computed (refer to Sect. 5.2), and the number of levels $(n+1)$ in the Gaussian decomposition of the high-frequency image (refer to Sect. 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 Sect. 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 $n = 3$.

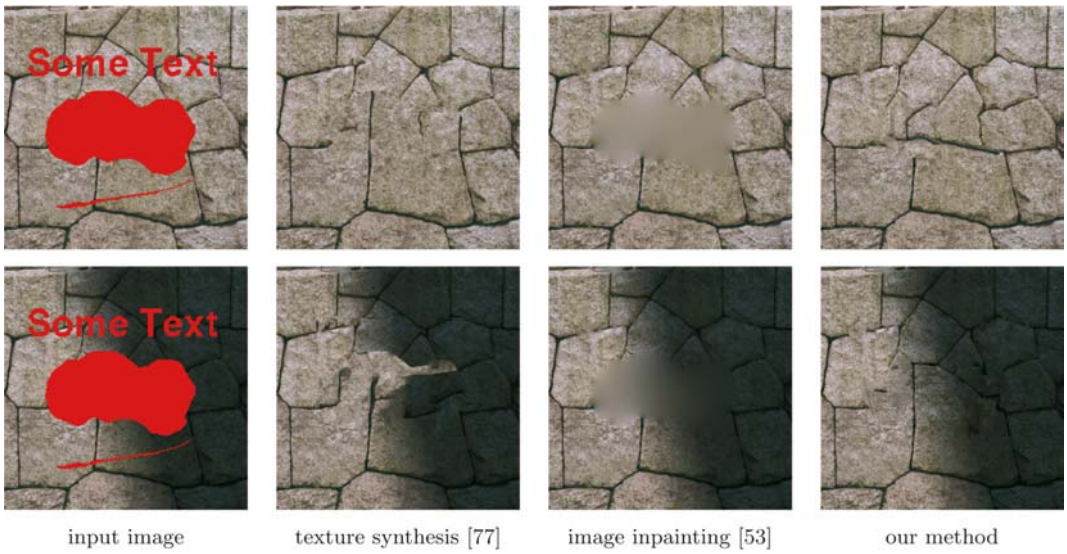


Fig. 15. Comparison between texture synthesis, image inpainting, and our method for input images with texture (top row) and with texture and additional intensity gradient (bottom row). Each row, from left to right: input image (damaged areas are masked out); resulting images from texture synthesis [77],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

Fig. 16. Effect of the value of κ. 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 a low-frequency part. An optimal κ is between the extremes. We use a coherence parameter [34] of 10

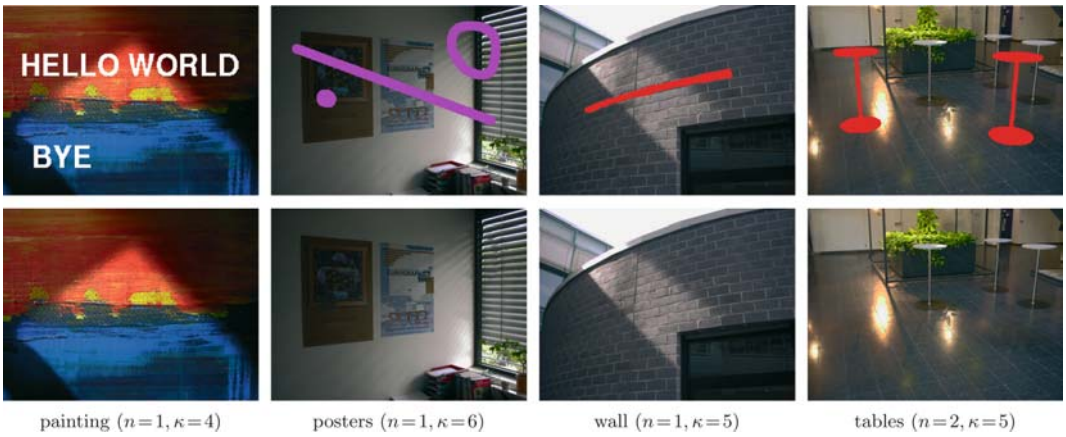


Fig. 17. Top row: input images with masked areas. Bottom row: restored images (see also Sect. 6). The parameter κ (= number of DCT subbands used to compute the low-frequency image L_κ) has been chosen automatically according to our autocorrelation metric (see Sect. 5.2). n is the highest multiresolution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

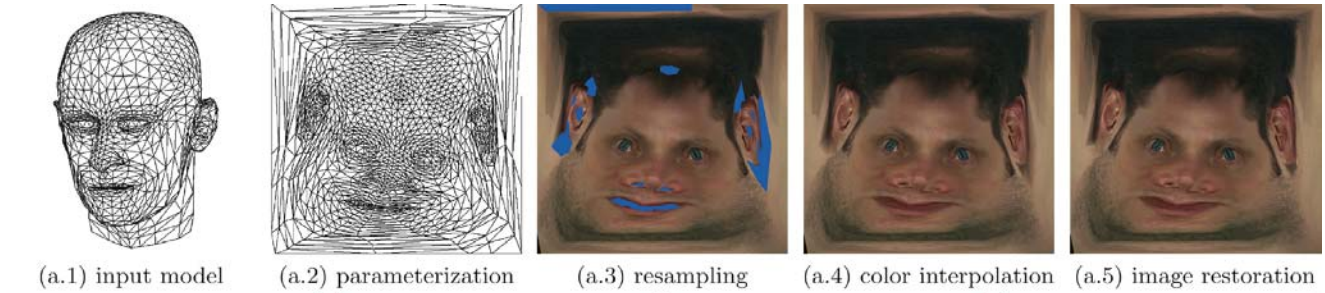
Figure 17 shows some results obtained with our method. Each input image is shown with its mask applied. For the purpose of illustration, the color of each mask has been chosen to differ significantly from the content of the input image. We found that the restored images look plausible in general. In some cases, we obtained results that looked surprisingly good. One example is the table image (Fig. 17, right column), where 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 multiresolution 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 multiresolution texture synthesis is more favorably.

Currently, our implementation is rather experimental: no optimizations have been performed, and the timings require the gathering of quite a lot of statistical data. All timings were collected on a 1.7 GHz Pentium 4 PC and are given for an input image size of 600×450 pixels. The time required 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

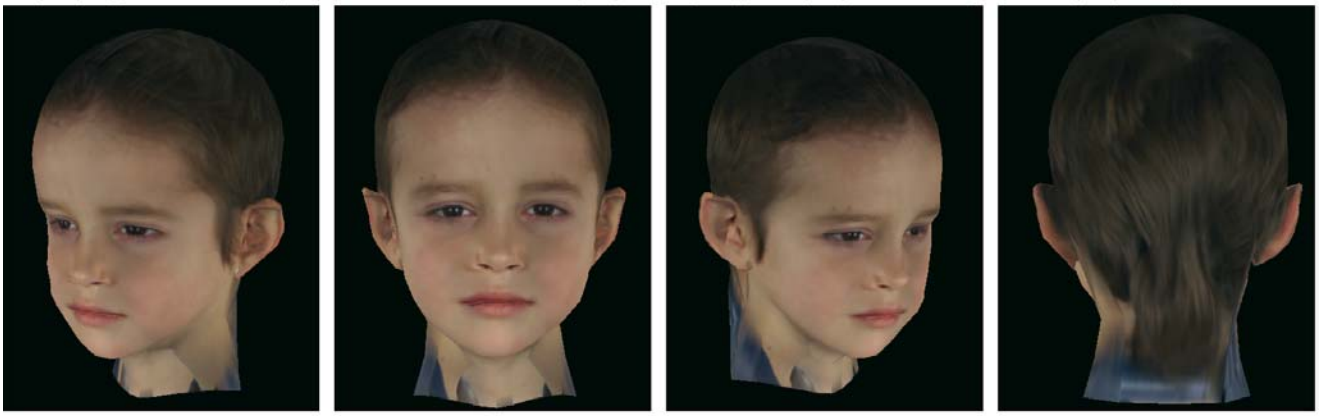


(a.1) input model (a.2) parameterization (a.3) resampling (a.4) color interpolation (a.5) image restoration

(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 $k = 1.5$ and 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 criteria given in Sect. 5

with photographs. In this example, we process 3D range scan data according to the method presented by Kähler et al. [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 Eq. 2, where the view vector is directed to see the front face. Resampling results are found in the same figure (Figs. 18 (a.3) and (b.3)). Blue pixels indicate that the surface regions for which the input photographs do not provide any information, 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 Sect. 4, while in Figs. (a.5) and (b.5), the missing parts are reconstructed using the proposed image restoration techniques as described in Sect. 5. The frequency decomposition parameters are $\kappa = 4$ in (a.5) and $\kappa = 5$ in (b.5). These κs are calculated with the criteria as described in Sect. 5.2. In Figs. 18 (a.6) and (b.6), the texture-mapped 3D models are presented. We use five photographs as the input – namely, the front, left, right, back, and upper front view. An example of front, left, and right inputs is given in Fig. 9. If we render the reconstructed model from a view that 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 Figs. (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 Figs. 18 (a.3) and (b.3). Although these regions have no input information, both methods can reproduce colors. In addition, the image restoration method reconstructs some texture.

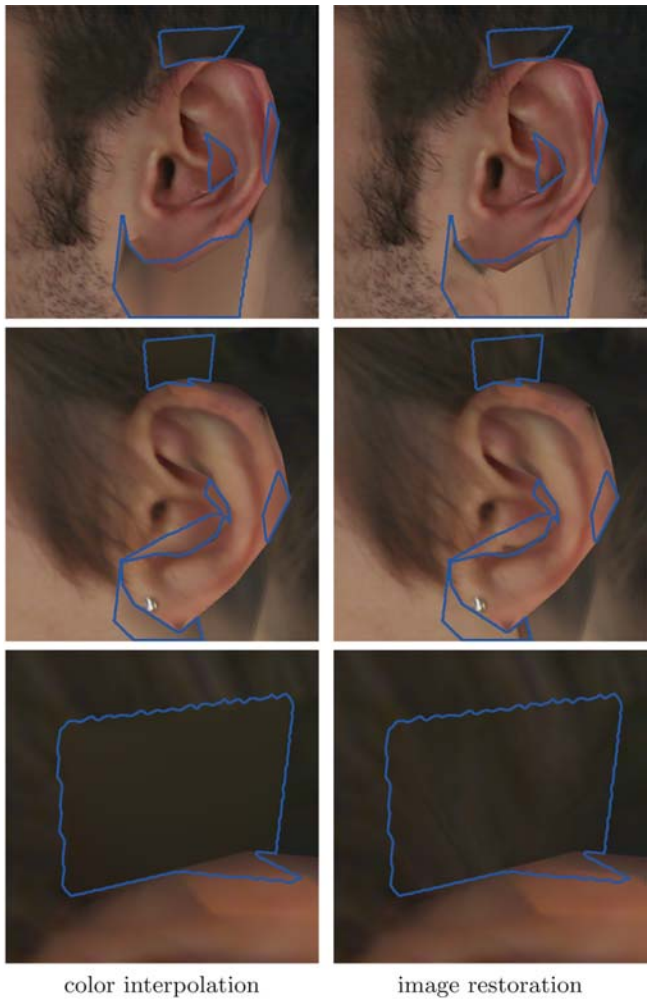


Fig. 19. Comparison of the color interpolation method and the image restoration method. Inside the areas delineated by the blue lines, there is no texture information. The bottom row is zoomed-in view of the upper part of the ear in the middle row. (see (a.3) and (b.3) in Fig. 18)

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 subtasks: parameterization, texture combination, and texture restoration.

For the parameterization, we introduced two signal terms for the geometry stretch energy method: the *visual importance* term and the *triangle shape* term. The visual importance term defines the importance of the texture for efficient use of the texture area. The triangle shape term is for alleviating the parameter crack problem of L^2 geometric stretch parameterization.

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

Texture restoration is needed because sometimes we cannot find pixel information on input photographs by registration error or occlusion. In our case, taking some more pictures to fill in such missing pixel information might not be a good solution, because the human face is a dynamic object, and it is hard to reproduce exactly the same face.

Therefore, introducing more pictures may add to the error. Moreover, a face has view-dependent reflection component (e.g., specular reflection component); this makes it difficult to control the illumination condition.

To solve this problem, we introduced a color interpolation method, which exploits the 3D mesh topology to fill in the missing pixels, and an image restoration method, which combines image inpainting with texture synthesis by using 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:

- **Parameterization:** We gave two kinds of signals. Each signal parameter is decided experimentally. A promising research direction would be to find optimal parameter values to fine-tune the signals.
- **Texture combination:** We solved the boundary problem by the multiresolution spline technique. This method connects discontinuity regions with a 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.
- **Texture 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 that 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 Zhang et al. [83], a user can prescribe feature information (a *texton* mask), a rotation vector, and a transition function. Such user-defined guidance may improve the reconstruction quality and also give some freedom to the user to control the results.

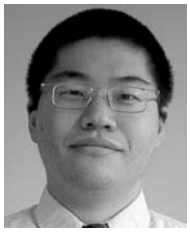
Acknowledgement This work has been partially funded by the Max Planck Center for Visual Computing and Communication.

References

1. Arya S, Mount DM, Netanyahu NS, Silverman R, Wu AY (1998) An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. J ACM 46(6):891–923
2. Ashikhmin M (2001) Synthesizing natural textures. In: Proceedings of the 2001 ACM Symposium on Interactive 3D Graphics, March 19–21 2001, pp 217–226
3. Ballester C, Bertalmio M, Caselles V, Sapiro G, Verdera J (2001) Filling-in by joint interpolation of vector fields and gray levels. IEEE Trans Image Process 10(8):1200–1211
4. Balmelli L, Taubin G, Bernardini F (2002) Space-optimized texture maps. Eurographics 21(3):411–420
5. Bar-Joseph Z, El-Yaniv R, Lischinski D, Werman M (2001) Texture mixing and texture movie synthesis using statistical learning. IEEE Trans Visual Comput Graph 7(2):120–135
6. Bennis C, Vézien J-M, Iglésias G (1991) Piecewise surface flattening for non-distorted texture mapping. In: Computer Graphics, SIGGRAPH '91 Conference Proceedings, pp 237–246
7. Bertalmio M, Sapiro G, Caselles V, Ballester C (2000) Image inpainting. In: Computer Graphics, SIGGRAPH '00 Conference Proceedings, pp 417–424
8. Blinn JF (1978) Simulation of wrinkled surfaces. In: Computer Graphics, SIGGRAPH '78 Conference Proceedings, pp 12:286–292
9. Blinn JF, Newell ME (1976) Texture and reflection in computer generated images. Commun ACM 19(10):542–547
10. Brooks S, Dodgson N (2002) Self-similarity based texture editing. ACM Trans Graph, pp 653–656
11. Burt PJ, Adelson EH (1983) A multi-resolution spline with application to image mosaics. ACM Trans Graph 2(4):217–236
12. Cignoni P, Montani C, Rocchini C, Scopigno R, Tarini M (1999) Preserving attribute values on simplified meshes by resampling detail textures. Vis Comput 15(10):519–539
13. Cohen MF, Shade J, Hiller S, Deussen O (2003) Wang tiles for image and texture generation. ACM Trans Graph 22(3):287–294
14. De Bonet JS (1997) Multiresolution sampling procedure for analysis and synthesis of texture images. In: Computer Graphics, SIGGRAPH '97 Conference Proceedings, pp 361–368
15. Debevec PE, Hawkins T, Tchou C, Duiker H-P, Sarokin W, Sagar M (2000) Acquiring the reflectance field of a human face. In: Computer Graphics, SIGGRAPH '00 Conference Proceedings, pp 145–156
16. Desbrun M, Meyer M, Alliez P (2002) Intrinsic parameterizations of surface meshes. In: Eurographics 2002 Conference Proceedings 21(3):209–218
17. Desbrun M, Meyer M, Schröder P, Barr AH (1999) Implicit fairing of irregular meshes using diffusion and curvature flow. In: Computer Graphics, SIGGRAPH '99 Conference Proceedings, pp 33:317–324
18. Drori I, Cohen-Or D, Yeshurun H (2003) Fragment-based image completion. ACM Trans Graph 22(3):303–312
19. Duchamp T, Certain A, DeRose A, Stuetzle W (1997) Hierarchical computation of PL harmonic embeddings. Technical report, University of Washington
20. Eck M, DeRose T, Duchamp T, Hoppe H, Lounsbery M, Stuetzle W (1995) Multiresolution analysis of arbitrary meshes. In: Computer Graphics, SIGGRAPH '95 Conference Proceedings, pp 173–182
21. Efros A, Leung T (1999) Texture synthesis by non-parametric sampling, pp 1033–1038
22. Efros AA, Freeman WT (2001) Image quilting for texture synthesis and transfer. In: Computer Graphics, SIGGRAPH '01 Conference Proceedings, pp 341–346
23. Fleischer K, Laidlaw D, Currin B, Barr A (1995) Cellular texture generation. In: Computer Graphics, SIGGRAPH '95 Conference Proceedings, pp 239–248
24. Floater MS (1997) Parametrization and smooth approximation of surface triangulations. Comput Aided Geom Des 14:231–250
25. Floater MS (2003) Mean value coordinates. Comput Aided Geom Des 20(1):19–27
26. Floater MS, Hormann K (2004) Surface parameterization: a tutorial and survey. In: Advances on multiresolution in geometric

modelling. Springer, Berlin Heidelberg New York, pp 259–284
27. Foley J, van Dam A, Feiner S, Hughes J (1992) Computer graphics principles and practice, 2nd edn. Addison-Wesley, Boston
28. Garber DD (1981) Computational models for texture analysis and texture synthesis. Dissertation, University of Southern California
29. Greene N (1986) Environment mapping and other applications of world projection. IEEE Comput Graph Appl 6(11):21–29
30. Guskov I, Vidimce K, Sweldens W, Schröder P (2000) Normal meshes. In: Computer Graphics, SIGGRAPH '00 Conference Proceedings, pp 95–102
31. Haker S, Angenent S, Tannenbaum A, Kikinis R, Sapiro G, Halle M (2000) Conformal surface parameterization for texture mapping. IEEE Trans Visual Comput Graph 6(2):181–189
32. Harrison P (2001) A non-hierarchical procedure for re-synthesis of complex textures. In: WSCG 2001 Conference Proceedings, Plyen, Czech Republic, pp 190–197
33. Heeger DJ, Bergen JR (1995) Pyramid-based texture analysis/synthesis. In: Computer Graphics, SIGGRAPH '95 Conference Proceedings, pp 229–238
34. Hertzmann A, Jacobs CE, Oliver N, Curless B, Salesin DH (2001) Image analogies. In: Computer Graphics, SIGGRAPH '01 Conference Proceedings, pp 327–340
35. Hirani AN, Totsuka T (1996) Combining frequency and spatial domain information for fast interactive image noise removal. In: Computer Graphics (SIGGRAPH '96 Conference Proceedings), pp 269–276
36. Hoppe H (1996) Progressive meshes. In: Computer Graphics, SIGGRAPH '96 Conference Proceedings, pp 99–108
37. Hormann K, Greiner G (2000) Mips: an efficient global parametrization method. In: Laurent PJ, Sablonnière P, Schumaker LL (eds), Curve and surface design: Saint-Malo 1999. Vanderbilt University Press, Nashville pp 153–162.
38. Igarashi T, Cosgrove D (2001) Adaptive unwrapping for interactive texture painting. In: Proceedings of the 2001 Symposium on Interactive 3D graphics, pp 209–216
39. Igehy H, Pereira L (1997) Image replacement through texture synthesis. In: Proceedings of the IEEE International Conference on Image Processing 3:186–189
40. Kähler K, Haber J, Yamauchi H, Seidel H-P (2002) Head shop: generating animated head models with anatomical structure. In: Proceedings of the 2002 ACM SIGGRAPH Symposium on Computer Animation, San Antonio, USA, pp 55–64
41. Khodakovsky A, Litke N, Schröder P (2003) Globally smooth parameterizations with low distortion. ACM Trans Graph 22(3):350–357
42. Lee AWF, Sweldens W, Schröder P, Cowsar L, Dobkin D (1998) Maps: Multiresolution adaptive parameterization of surfaces. In: Computer Graphics, SIGGRAPH '98 Conference Proceedings, pp 95–104
43. Lee W-S, Magnenat-Thalmann N (2000) Fast head modeling for animation. Image Vis Comput 18(4):355–364
44. Lévy B (2001) Constrained texture mapping for polygonal meshes. In: Computer Graphics, SIGGRAPH '01 Conference Proceedings, pp 417–424
45. Lévy B, Petitjean S, Ray N, Maillot J (2002) Least squares conformal maps for automatic texture atlas generation. ACM Trans Graph 21(3):362–371
46. Liang L, Liu C, Xu Y-Q, Guo B, Shum H-Y (2001) Real-time texture synthesis by patch-based sampling. In: Computer Graphics, SIGGRAPH '01 Conference Proceeding 20(3):127–150
47. Maillot J, Yahia H, Verroust A (1993) Interactive texture mapping. In: Computer Graphics, SIGGRAPH '93 Conference Proceedings, pp 27–34
48. Marschner SR, Westin SH, Lafortune EPF, Torrance KE, Greenberg DP (1999) Image-based BRDF measurement including human skin. In: Proceedings of the 10th Eurographics Workshop on Rendering, pp 131–144
49. Miyata K (1990) A method of generating stone wall patterns. In: Computer Graphics, SIGGRAPH '90 Conference Proceedings, pp 387–394
50. Nealen A, Alexa M (2003) Hybrid texture synthesis. In: Proceedings of the 13th Eurographics workshop on Rendering, pp 97–105
51. Neugebauer PJ, Klein K (1999) Texturing 3D models of real world objects from multiple unregistered photographic views. Eurographics 18(3):C245–C256
52. Oh BM, Chen M, Dorsey J, Durand F (2001) Image-based modeling and photo editing. In: Computer Graphics, SIGGRAPH '01 Conference Proceedings, pp 433–442
53. Oliveira MM, Bowen B, McKenna R, Chang YS (2001) Fast digital image inpainting. In: Proceedings of the International Conference on Visualization, Imaging and Image Processing (VIIP 2001), pp 261–266
54. Pérez P, Gangnet M, Blake A (2003) Poisson image editing. ACM Trans Graph 22(3):313–318
55. Perona P, Malik J (1990) Scale-space and edge detection using anisotropic diffusion. IEEE Trans Pattern Anal Mach Intell PAMI-12(7):629–639
56. Pighin F, Hecker J, Lischinski D, Szeliski R, Salesin DH (1998) Synthesizing realistic facial expressions from photographs. In: Computer Graphics, SIGGRAPH '98 Conference Proceedings, pp 75–84
57. Piponi D, Borshukov G (2000) Seamless texture mapping of subdivision surfaces by model pelting and texture blending. In: Computer Graphics, SIGGRAPH '00 Conference Proceedings, pp 471–478
58. Popat K, Picard RW (1993) Novel cluster-based probability model for texture synthesis, classification, and compression. In: Proceedings of SPIE Visual Communications and Image Processing 2094:756–768
59. Praun E, Hoppe H (2003) Spherical parametrization and remeshing. ACM Trans Graph
60. Rocchini C, Cignoni P, Montani C, Scopigno R (1999) Multiple textures stitching and blending on 3D objects. In: Proceedings of the 10th Eurographics Workshop on Rendering, pp 127–138
61. Saint-Marc P, Chen JS, Medioni G (1991) Adaptive smoothing: a general tool for early vision. IEEE Trans Pattern Anal Mach Intell 13(6):514–529
62. Sander PV, Snyder J, Gortler SJ, Hoppe H (2001) Texture mapping progressive meshes. In: Computer Graphics, SIGGRAPH '01 Conference Proceedings, pp 409–416
63. Sander PV, Gortler SJ, Snyder J, Hoppe H (2002) Signal-specialized parametrization. In: Proceedings of the 13th Eurographics Workshop on Rendering, 26–28 June 2002, pp 87–98
64. Shannon C (1948) A mathematical theory of communication. Bell Syst Tech J 27(3):379–423
65. Sheffer A, de Sturler E (2000) Parameterization of faceted surfaces for meshing using angle based flattening. Eng Comput 17(3):326–337
66. Sheffer A, Hart JC (2002) Seamster: inconspicuous low-distortion texture seam layout. In: Proceedings of IEEE Visualization '02, pp 291–298
67. Simoncelli E, Portilla J (1998) Texture characterization via joint statistics of wavelet coefficient magnitudes. In: Fifth IEEE International Conference on Image Processing, Volume I, Chicago, 4–7 October 1998, pp 62–66
68. Smith SM, Brady JM (1995) SUSAN – A new approach to low level image processing. Technical Report TR95SMS1c, Defence Research Agency, Chertsey, Surrey, UK
69. Soler C, Cani M-P, Angelidis A (2002) Hierarchical pattern mapping. ACM Trans Graph, pp 673–680
70. Sorkine O, Cohen-Or D, Goldenthal R, Lischinski D (2002) Bounded-distortion piecewise mesh parameterization. In: Proceedings of IEEE Visualization '02, pp 355–362
71. Soucy M, Godin G, Rioux M (1996) A texture-mapping approach for the compression of colored 3D triangulations. Vis Comput 12(10):503–514
72. Szummer M, Picard RW (1996) Temporal texture modeling. In: ICIP, Lausanne, Switzerland, volume 3, pp 823–826
73. Tarini M, Yamauchi H, Haber J, Seidel H-P (2002) Texturing faces. In: Proceedings of Graphics Interface 2002, Calgary, Canada, pp 89–98
74. Tomasi C, Manduchi R (1983) Bilateral filtering for gray and color images. In: ICCV-98, 4–7 January 1998, pp 839–846

75. Turk G (1991) Generating textures on arbitrary surfaces using reaction-diffusion. In: Computer Graphics, SIGGRAPH '91 Conference Proceedings, pp 289–298
76. Turk G (2001) Texture synthesis on surfaces. In: Computer Graphics, SIGGRAPH '01 Conference Proceedings, pp 347–354
77. Wei L-Y, Levoy M (2000) Fast texture synthesis using tree-structured vector quantization. In: Computer Graphics, SIGGRAPH '00 Conference Proceedings, pp 479–488
78. Wei L-Y, Levoy M (2001) Texture synthesis over arbitrary manifold surfaces. In: Computer Graphics, SIGGRAPH '01 Conference Proceedings, pp 355–360
79. Yamauchi H, Haber J, Seidel H-P (2003) Image restoration using multiresolution texture synthesis and image inpainting. In: Proceedings of Computer Graphics International, Tokyo, Japan, pp 120–125
80. Ying L, Hertzmann A, Biermann H, Zorin D (2001) Texture and shape synthesis on surfaces. In: Eurographics rendering workshop, London, United Kingdom, June 25–27, pp 301–312
81. Zayer R, Rössl C, Seidel H-P (2004) Variations of angle based flattening. In: Advances in multiresolution for geometric modeling. Springer, Berlin Heidelberg New York
82. Zelinka S, Garland M (2002) Towards real-time texture synthesis with the jump map. In: Proceedings of the 13th Eurographics Workshop on Rendering, pp 101–107
83. Zhang J, Zhou K, Velho L, Guo B, Shum H-Y (2003) Synthesis of progressively-variant textures on arbitrary surfaces. ACM Trans Graph 22(3):295–302




HITOSHI YAMAUCHI is a research associate at the Max-Planck-Institut Informatik in Saarbrücken, Germany. He received his PhD (1997) in Information Science from Tohoku University, Japan. His research interests include parallel global illumination, human face modeling, image reconstruction, parameterization, and segmentation.

HENDRIK LENSCH is currently a visiting assistant professor with Marc Levoy at Stanford University's Computer Graphics Lab, USA, where he leads the research group "General Appearance Acquisition" of the Max Planck Center for Visual Computing and Communication (Saarbrücken/Stanford). He studied computer science at the University of Erlangen and the Royal Institute of Technology (KTH) in Stockholm, receiving his diploma in 1999. He worked as a PhD student and research associate at Hans-Peter Seidel's computer graphics group at the Max-Planck-Institut für Informatik in Saarbrücken, Germany. In 2003, he received his doctorate from Saarland University. His research interests include 3D appearance acquisition, BRDF reconstruction, and image-based rendering.

JÖRG HABER is a senior researcher at the Max-Planck-Institut Informatik in Saarbrücken, Germany. He received his Master's (1994) and PhD (1999) degrees in Mathematics from the Technische Universität München, Germany. During the last eight years he did research in various fields of computer graphics and image processing, including global illumination and real-time rendering techniques, physics-based simulation, scattered data approximation, and lossy image compression. For the last couple of years, his major research interests concentrate on modeling, animation, and rendering of human faces.

HANS-PETER SEIDEL is the scientific director and chair of the computer graphics group at the Max-Planck-Institut (MPI) Informatik and a professor of computer science at the University of Saarbruecken, Germany. The Saarbruecken computer graphics group was established in 1999 and currently consists of about 40 researchers.
He has published some 200 technical papers in the field and has lectured widely on these topics. He has received grants from a wide range of organizations, including the German National Science Foundation (DFG), the European Community (EU), NATO, and the German-Israel Foundation (GIF).
In 2003 Seidel was awarded the 'Leibniz Preis', the most prestigious German research award, from the German Research Foundation (DFG). Seidel is the first computer graphics researcher to receive this award. In 2004 he was selected as founding chair of the Eurographics Awards Programme.