Seppä, M. and Hämäläinen, M. (2005). Visualizing human brain surface from T_1-weighted MR images using texture-mapped triangle meshes. NeuroImage, 26: 1-12.

NeuroImage

www.elsevier.com/locate/ynimg
NeuroImage 26 (2005) 1 – 12

Visualizing human brain surface from T_1-weighted MR images using texture-mapped triangle meshes

Mika Seppä[a,*] and Matti Hämäläinen[a,b]

[a]Brain Research Unit, Low Temperature Laboratory, Helsinki University of Technology, FIN-02015 HUT, Espoo, Finland
[b]MGH/MIT/HMS Athinoula A. Martinos Center for Biomedical Imaging, Charlestown, USA

Received 12 November 2003; revised 30 September 2004; accepted 21 January 2005
Available online 3 March 2005

We describe a novel method for visualizing brain surface from anatomical magnetic resonance images (MRIs). The method utilizes standard 2D texture mapping capabilities of OpenGL graphics language. It combines the benefits of volume rendering and triangle-mesh rendering, allowing fast and realistic-looking brain surface visualizations. Consequently, relatively low-resolution triangle meshes can be used while the texture images provide the necessary details. The mapping is optimized to provide good texture-image resolution for the triangles with respect to their original sizes in the 3D MRI volume. The actual 2D texture images are generated by depth integration from the original MRI data. Our method adapts to anisotropic voxel sizes without any need to interpolate the volume data into cubic voxels, and it is very well suited for visualizing brain anatomy from standard T_1-weighted MR images. Furthermore, other OpenGL objects and techniques can be easily combined, for example, to use cut planes, to show other surfaces and objects, and to visualize functional data in addition to the anatomical information.

Keywords: Brain surface visualization; Volume rendering; Triangle-mesh rendering; Texture mapping

Introduction

Visualization of brain anatomy plays an important role for interpretation of functional brain data. Both the surface and inside structures of the brain need to be visualized. Nowadays, the most common source of anatomical data is magnetic resonance imaging (MRI) that provides good spatial resolution and is not harmful to the subject. In clinical applications, other 3D-imaging modalities, such as computed tomography (CT), are also common.

Brain surface visualization can be carried out with two fundamentally different methods: volume rendering or surface-based rendering, both of which have their benefits and drawbacks. Volume rendering is performed using the volumetric 3D data as such and it typically produces realistic-looking brain surface images. It can be applied directly to the unprocessed volumetric images so that volume element (voxel) opacity depends on the voxel values. Preprocessing steps, such as segmentation and tissue classification, can be also included. Volume rendering is computationally a very intensive process and it has been studied extensively with many different approaches. The four most common basic techniques are ray-casting (Levoy, 1988), splatting (Westover, 1990), shear-warp factorization (Lacroute and Levoy, 1994), and hardware-based 3D texture mapping (Cabral et al., 1994). An evaluation and comparison of these methods can be found in Meissner et al. (2000) and a more general survey of volume visualization algorithms in Elvins (1992). The 3D texture mapping capabilities and large on-board memories even in modern consumer-level PC graphics boards have made the 3D texture-mapping-based techniques very attractive and widely available (Rezk-Salama et al., 2000). Although volume rendering can be used for visualizing opaque surfaces, its capabilities are best suited for visualizing transparent volumes.

Surface-based rendering methods require segmentation of the volume of interest to define the surface being visualized. Despite several existing possibilities for surface representations, by far the most common approach is to employ triangle meshes. The segmented 3D surface is tessellated with triangles and the resulting triangle mesh is rendered as a representation of the 3D object. Meshes consisting of several thousands of triangles can be rendered quickly with current display hardware, thereby allowing interactive visualization. The drawback of this method for brain visualization is the poor visual quality of solid colored surfaces compared with the results achievable by volume rendering (see Fig. 1). Furthermore, the segmentation and the surface tessellation steps are difficult when the brain sulci need to be triangulated (Fischl et al., 2001). For solid colored triangle surfaces of the brain, order of 100,000 triangles are necessary to show surface details.

* Corresponding author. Brain Research Unit, Low Temperature Laboratory, Helsinki University of Technology, P.O. Box 2200, FIN-02015 HUT, Espoo, Finland. Fax: +358 9 4512969.
E-mail address: Mika.Seppa@hut.fi (M. Seppä).
Available online on ScienceDirect (www.sciencedirect.com).

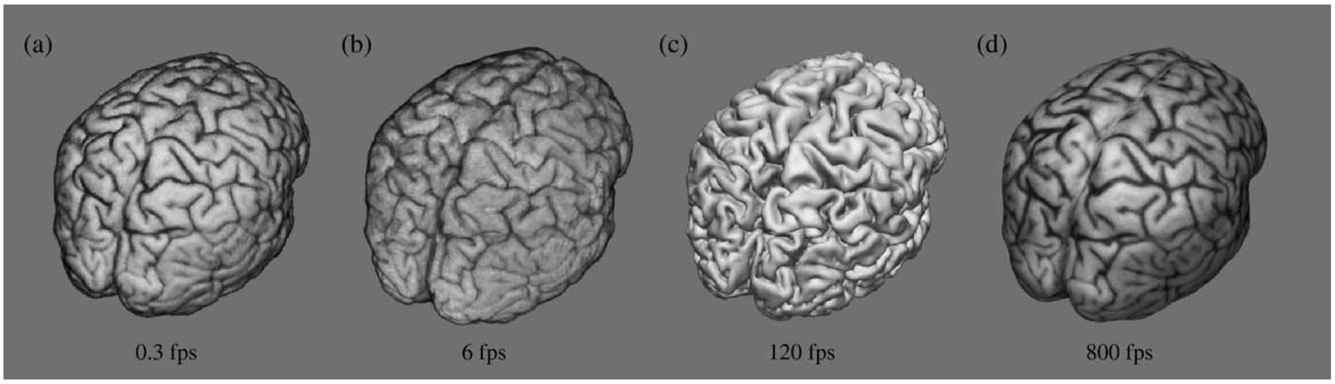

Fig. 1. Examples of different rendering methods showing the image quality and the refresh rates in frames-per-seconds (fps) for a window of 512 × 512 pixels using 3.00-GHz Pentium 4 Linux platform with NVidia Quadro4 980 XGL graphics board. First two images use volume rendering: ray-casting with lighting calculations and depth integration for surface colors (a) and hardware-based 3D texture mapping without lighting (b). Last two images use surface-based rendering with triangles meshes: plain colored mesh with 95 879 triangles (c) and texture-mapped mesh of our method with 12 224 triangles (d), both with lighting.

Fig. 1 shows example renderings of different methods along with typical refresh rates.

Some hybrid methods have been proposed to mix volume data with other rendering methods. Levoy (1990) studied a combination of polygon rendering and ray-casting for volume data and methods for combining hardware-accelerated volume and polygon rendering have been presented by Kreeger and Kaufman (1999). For volume visualization methods based on 3D texture mapping (Cabral et al., 1994), the rendering of polygon models can be easily included with the help of depth buffer. Hybrid visualization techniques, combining volume rendering and point rendering, have been also reported (Ma et al., 2002; Wilson et al., 2002). These methods reduce the rendered volume data resolution and fill-in details with point rendering.

The method described in this paper uses normal 2D texture mapping techniques (Blinn and Newell, 1976; Heckbert, 1986) to add surface details on the rendered brain triangle mesh. Consequently, we can achieve visual quality of volume-rendered brain surface while maintaining the speed of triangle-mesh rendering. The need to segment and triangulate the brain surface into sulci is avoided as well. Furthermore, the triangle size can be increased reducing the number of triangles even further while the visible surface details are preserved with texture images. Only about 10,000–20,000 triangles are needed for the textured brain surface model.

Methods

In this section, we present our method of creating texture-mapped triangle meshes from the MRI data. Although none of the basic ideas here is strictly new, the combination of these already known steps produces an efficient and very practical visualization solution. To our knowledge, the idea of relating directly the 3D volume data resolution and 2D texture image resolution and optimizing the mapping on the basis of this (Eq. (2)) is also novel. Depth integration (Bomans et al., 1990) method has been previously used for volume rendering of T_1-weighted brain MRI data by ray-casting. However, its potential for creating texture map images for brain surface triangle meshes has not been reported before.

The starting point for our method is a triangle mesh of the brain surface and the corresponding MR volumetric image. As the method does not place any special requirements for the triangle mesh or for the MRI volume, the segmentation and triangulation steps are only briefly explained. The rest of the procedure consists of four fundamental steps shown in Fig. 2. In the subsections below, we first explain the basic idea of our method followed by the description of the procedure steps in detail. The last two subsections focus on the adjustment of method parameters.

For visualization, we use the industry-standard OpenGL graphics library, which provides fast rendering of triangle meshes as compatible display hardware is readily available nowadays for different platforms, ranging from high-end workstations to low-end PCs and laptops. OpenGL is also operating system (OS) independent, unlike some other graphic libraries provided by hardware or OS manufacturers. Please note that the method described in this paper is not limited to OpenGL alone but can be used with any graphics applications programming interface (API) supporting texture-mapped triangles.

Basic idea

The basic idea of our approach is to take a triangle mesh with relatively few triangles and to map 2D texture (raster) images onto

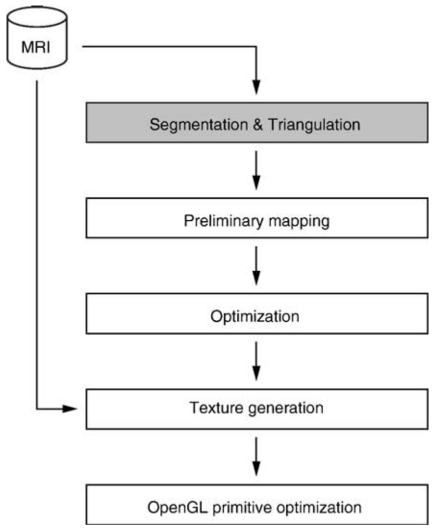

Fig. 2. Flow diagram of the procedure. The gray box, segmentation and triangulation of MRI data, is a pre-requisite for the method and the white boxes show the actual method steps.

it. The layouts of the texture images are optimized and the image data generated individually for each mesh. The visualization of such texture-mapped meshes provides better surface details while still benefiting from fast triangle mesh rendition.

The way how the images are wrapped onto the triangle mesh can be alternatively thought as how the 3D mesh is opened onto 2D texture images. Hence, the wrapping is closely related to techniques used for unfolding the brain and creating flat maps (Fischl et al., 1999), with the difference that the 2D images in our method are not visualized as such. Thus, the mesh surface can be divided into several parts and basically each triangle could be even mapped onto its own texture image. However, in texture mapping, the seams of the parts are potential sources for texture discontinuities and thus their number should be kept at minimum. On the other hand, the larger are the pieces, the larger is the average distortion when a curved surface is mapped onto a plane.

We decided to divide the triangle mesh into three parts that are separately mapped onto three texture map images. One of these parts resembles the surface of a cylinder and wraps around the mesh. The wrapping leads to fewer seams, allows more efficient use of the texture-map image area, and facilitates faster rendering as there are less discontinuities for OpenGL primitive optimization. Other two parts are the top and bottom caps attached to this imaginary cylinder.

The mesh is mapped onto 2D texture images in two stages. First, a preliminary mapping is created by distributing the original triangles evenly onto a unit sphere and dividing it then into three parts. This results in a rather even division with smooth borders and also produces a preliminary mapping where triangle overlap is already at minimum. Next, these mappings are optimized to provide texture image resolution that is comparable to the resolution of details that can be extracted from the volumetric data. After that, texture-map raster images are generated by depth integration from the original MRI data. The last step is optimization of OpenGL primitives that creates triangle strips and triangle fans for the mesh, speeding up per-vertex processing.

Surface triangulation

As mentioned before, our method does not pose any special requirements for the triangle mesh or for the MRI volume. Specifically, the voxel size can be different along different axes. Thus, the slice separation can differ from the in-slice resolution without the need for re-slicing the volume. The triangles of the mesh do not need to be closely equilateral, although our triangulation method produces such triangles. Furthermore, only the surface of the brain is needed where the sulci are closed (see the texture-less images of Fig. 7). Thus, the difficulty of segmentation and triangulation of cortical mantle into sulci is avoided. As the exact implementations of the segmentation and triangulation steps are not crucial for the method, we explain our tools only briefly.

Subject's T_1-weighted MR images are segmented for brain surface. We use 3D region-growing operation (Fuchs et al., 1993) with thresholding followed by morphological operations that close the sulci and smooth the surface. In our software, these and several other 3D image processing commands can be interactively applied to the volumetric data. Manual adjustment of segmentation mask is also possible. For a comprehensive review of MRI brain segmentation techniques, see for example Suri et al. (2002a,b).

Next, the brain surface is triangulated. Our algorithm is derived from a patching process of Wagner et al. (1995). The patch centers become vertices of the mesh and the neighboring patches are connected which creates edges and triangles. This method produces almost equilateral triangles with a side length close to a specified value (patch size). This procedure is implemented as one 3D image processing operator in our software. As mentioned, any other triangulation method can be employed, as long as it produces connected and oriented triangles (with normals pointing out) that represent a closed surface.

Importantly, with this new visualization method, the triangle sizes can be increased while the texture mapping still provides accurate surface details. This way, the number of triangles used can be reduced. As will be noted in the Results section, excess number of triangles slows down the rendering speed without any benefit in the visual accuracy. Thus, some common triangulation methods producing large number of small triangles, such as the marching-cubes algorithm (Lorensen and Cline, 1987), would need triangle mesh decimation to produce efficient meshes for our method.

Preliminary mapping

In the first phase of our method, we create a preliminary mapping where the 3D triangle mesh is divided into parts that are opened on top of 2D texture images. This step is performed by mapping the mesh first onto a sphere that is then partitioned. The spherical parametrization of meshes has many uses, such as compression (Schröder and Sweldens, 1995), morphing (Alexa, 2002), re-meshing (Praun and Hoppe, 2003), and shape analysis (Funkhouser et al., 2003). Different parametrization methods have been developed that employ for example conformal approximations (Haker et al., 2000), spring-like relaxation processes (Alexa, 2002), spectral graph theory (Gotsman et al., 2003), and minimization of stretch metrics (Praun and Hoppe, 2003).

For our method, the spherical parametrization is an intermediate step that helps mesh division into parts and provides a preliminary mapping. The important properties are good areal distribution of triangles, minimal amount of overlapping triangles, and speed. Thus, we ended up using an iterative process that optimizes an error function based on triangles' areas on the sphere. It starts by projecting the vertices of the 3D mesh onto a unit sphere whose origin is inside the mesh. The origin is used as the projection center and its exact location is not of great concern for the method as the locations of the projected vertices are optimized next. We select the origin of the largest sphere that can be fully enclosed inside the mesh.

Next, the projected vertices are moved on the unit sphere to minimize the error function

$$E_{\text{sphere}} = \sum_{i=1}^{N} \left[w_i - 4\pi \left(\alpha \frac{a_i}{A} + (1-\alpha) \frac{1}{N} \right) \right]^2 \tag{1}$$

where w_i is the solid angle subtended by the ith triangle on the unit sphere at the origin, a_i is the area of that triangle in the original 3D mesh, $A = \sum a_i$ is the total surface area of the original mesh, N is the number of triangles, and α is a parameter determining the optimal target. Using this error function, the optimal target can be selected linearly between equal area (equal solid angle) for each triangle (α = 0) and area proportional to the triangle's original size (α = 1). We use signed values for w_i so that triangles whose surface normal points inwards on the sphere have negative solid angles. As the target term of the error function is always positive, those triangles produce large errors which helps to straighten out the projected mapping on the unit sphere and to reduce overlaps associated with flipped triangles.

In the minimization, we employ the Levenberg–Marquardt method (Marquardt, 1963) and use several hierarchy levels for the mesh (multi-resolution technique). This way, the spatial distribution of error across the mesh is smoothed faster. Higher-level meshes are constructed from the lower-level meshes by the half-edge-collapse technique (Hormann et al., 1999).

Next, the mesh on the sphere is divided into three parts (see Fig. 3) by determining where the center point of a triangle (mean of its vertices) falls. If the center is above the 45 degree of latitude, the triangle belongs to the top cap. If it is below the −45 degree of latitude, the triangle goes to the bottom cap. Otherwise, it belongs to the middle band. Afterwards, triangles that have only one neighbor out of three in the same part are transferred to the neighboring part. This smoothes the edges of the mesh parts.

The top and bottom caps are preliminarily mapped onto square texture images and the middle band onto a texture image whose width is four times its height. This initial size is selected, because the height of that part on the sphere is 90° and the width 360°. This texture map is repeated in horizontal direction so that the coordinates of the texture map wrap around. To each part, one layer of border triangles is added (see Fig. 4). These triangles are marked to be special border triangles and they are handled differently from normal triangles in several steps. Their function is to prevent border overlap and also to help the generation of texture map data.

Optimization of the mapping

After the preliminary phase, the triangles mapped onto the 2D texture images are optimized for size and shape. The original mesh is first converted into voxel coordinates so that all measured distances will be in voxels. Analogously, the mapped mesh uses pixels as coordinates. When converting the original triangle mesh to voxel coordinates, we introduce a scaling factor β. The optimal size of a mapped triangle would be such that its side lengths in pixels are equal to the side lengths of the scaled original triangle in voxels. The rationale for the use of these units is that the resolution of the original MR image determines the details that can be visualized on the texture images. The effect of the scaling factor β is discussed in a later subsection.

We want to emphasize again that the voxels of the MRI data do not need to be cubic. In particular, the inter-slice distance can be different from the pixel dimensions in slice without any need to re-

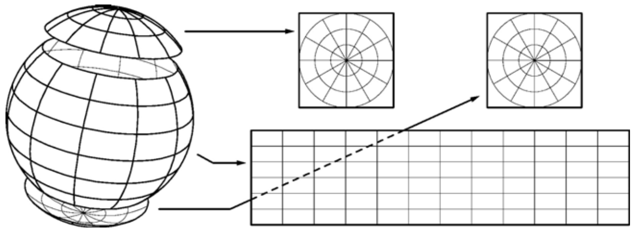

Fig. 3. Dividing and opening the unit sphere for preliminary mapping.

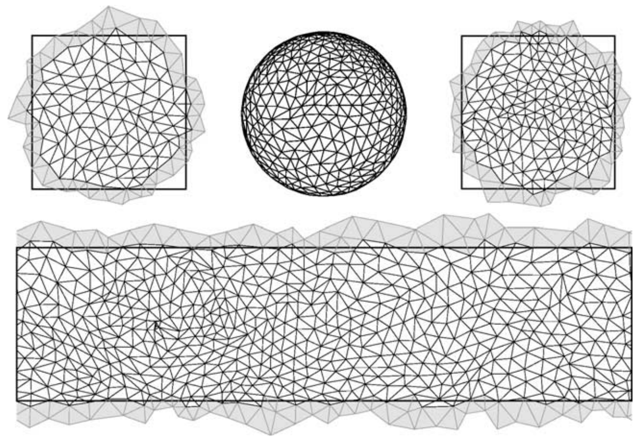

Fig. 4. Spherized sample mesh and preliminary mapping for it. The added border triangles are shown with gray shading.

slice the set to cubic voxels. By using the voxels and pixels as units of measurement, we make sure that the resolution a triangle obtains from texture-map image is comparable to the resolution of information we can extract from the MR images.

Low-distortion parametrization of triangulated surfaces is widely studied in computer graphics. In addition to texture mapping, it has applications in operations such as multi-resolution analysis and re-meshing (Alliez et al., 2003; Eck et al., 1995; Lee et al., 1998), surface fitting (Floater, 1997), digital geometry processing (Guskov et al., 1999), texture synthesis (Turk, 2001), and mesh compression (Gu et al., 2002). Many popular parametrization methods are based on assigning spring-like forces to triangles' edges in the mesh (Eck et al., 1995; Floater, 1997; Maillot et al., 1993) and finding an equilibrium. Furthermore, use of conformal maps have been investigated (Gu and Yau, 2003; Haker et al., 2000; Hurdal et al., 1999; Lévy et al., 2002) in applications where minimization of angle distortion is important.

A very useful and intuitive base for the triangle's distortion cost function is the Jacobian $\mathbf{J}_i$ of the affine transformation from 2D texture coordinates to the 3D coordinates of the original triangle T_i (Sander et al., 2001). Specifically, the two singular values $s_{i,\max} \geq s_{i,\min}$ of the matrix $\mathbf{J}_i$ express the triangle's deformation as scalings in two orthogonal directions. Sander et al. (2001) also introduce the distortion norms $L^2(T_i) = \sqrt{(s_{i,\max}^2 + s_{i,\min}^2)/2}$ and $L^\infty(T_i) = s_{i,\max}$. Other proposed distortion measures are $s_{i,\max}/s_{i,\min}$ (Hormann and Greiner, 2000), $\max\{s_{i,\max},\ 1/s_{i,\min}\}$ (Sorkine et al., 2002), and $s_{i,\max}s_{i,\min}$ (Khodakovsky et al., 2003).

These distortion measures are typically used for filling an area of predetermined size with triangles. The boundary vertices are either assumed fixed or normalization is used for the global scale. We, however, want the mapping to evolve freely and to find an optimal size for the texture images with respect to the 3D data resolution. Thus, both triangle stretching and compression has to be penalized and the distortion measure should obtain minimum with $s_{i,\max} = s_{i,\min} = 1$. Only one of the above measures ($\max\{s_{i,\max},\ 1/s_{i,\min}\}$) fulfills these requirements.

We furthermore require that scaling in one direction has no effect on the orthogonal direction. The optimum value 1 and the error induced by a deviation from this optimum should be independent of the other singular value. This is important to us as resolution lost or gained in one direction cannot be compensated by changing the resolution in the orthogonal direction. Since none of the above distortion measures has this property, we ended up using the following novel measure:

$$E_{\text{tx}}(T_i) = s_{i,\max} + 1/s_{i,\max} + s_{i,\min} + 1/s_{i,\min} - 4 \tag{2}$$

where $E_{\text{tx}}(T_i)$ is the distortion error (on the texture map) for triangle T_i and constant 4 is subtracted for convenience to make $E_{\text{tx}}(T_i) = 0$ for the optimal case. This measure fulfills the requirements mentioned.

For each mesh part, we iteratively minimize the total error $E_{\text{tot}} = \sum \lambda_i E_{\text{tx}}(T_i)$ where λ_i is the weighting factor for triangle T_i. The special border triangles added in the preliminary mapping phase have $\lambda_i = 1/4$ weight while other triangles have $\lambda_i = 1$. For vertex $p_k^{(\ell)}$ of the mesh part (kth vertex, iteration step ℓ), the gradient vector $\boldsymbol{G}_k^{(\ell)}$ and Hessian matrix $\mathbf{H}_k^{(\ell)}$ of E_{tot} are calculated considering other vertices constant (i.e., only triangles adjacent to $p_k^{(\ell)}$ contribute). We evaluate a Levenberg–Marquardt-type (Marquardt, 1963) iteration step

$$p_k^{(\ell+1)} = p_k^{(\ell)} - \left(\mathbf{H}_k^{(\ell)} + \sigma \mathbf{I}\right)^{-1} \boldsymbol{G}_k^{(\ell)} \tag{3}$$

where $\mathbf{I}$ is the identity matrix and $\sigma \geq 0$ the dampening factor. With $\sigma = 0$, we jump to the minimum of the local second-degree Taylor expansion of the error function and with increasing values of σ the step approaches deepest descent method with decreasing step length. The dampening factor σ is modified like in the Levenberg–Marquardt method, depending whether the considered step decreased the error or not.

The triangles are stored as ordered sets of three vertices which allows us to determine the direction they are facing. Overlapping triangles in the interior of the mesh necessarily produce flipped triangles. The optimization algorithm counts the number of triangles facing the opposite way than the majority and prefers this count over the actual error E_{tot}. Thus, the error is allowed to increase if that makes the number of flipped triangles to decrease. Furthermore, the count is never allowed to increase even if this would result in a lower total error.

To speed up the propagation of error and thus to reduce the number of iterations needed, we again used different hierarchy levels for the mesh (Hormann et al., 1999). The optimization starts for a simplified mesh and progresses back to more accurate levels and finally to the full original mesh part. The optimization ends when the relative change in the error drops below a predetermined level and all triangles are facing the same way.

The vertex positions are not limited to be within the texture boundaries while they are being optimized. The texture sizes are adjusted at the end so that all vertices fall inside the boundaries. Furthermore, a simple one-dimensional search is accomplished for the two cap parts to find out whether a smaller texture map size could be used if the mapping was rotated. For the wrapped texture map direction, all the vertices moved out of one side come in from the opposite side. The width of that texture image is adjusted during the optimization as it affects the outcome. We periodically test to increase and decrease the width to find the optimal value. All texture sizes are limited to powers of two due to OpenGL texture size restrictions. Fig. 5 shows the result of the optimization for the sample mesh overlaid on the texture images. The special border triangles are not shown although they have been used for texture generation.

Generation of texture-map images

After the optimization, the actual 2D texture images are generated. The color of a pixel is determined in the following way. First, a triangle is found on the texture image into which the center of the pixel belongs to. Next, the barycentric coordinates inside that triangle are calculated for the point. By using these coordinates, the corresponding location is found from the original triangle in 3D and a normal vector is interpolated from the normal vectors of the triangle's vertices. The gray-scale color assigned for the pixel is obtained by depth integration (Bomans et al., 1990) from the T_1-weighted MR images. The integration starts from the inverse mapped location and proceeds inwards, anti-parallel to the interpolated normal vector. The width, height, and depth of the integration volume are adjustable parameters whose effects are discussed in a later subsection. We use uniform weighting for integration and thus the result is simply a gray-scale average inside the integration volume.

A pixel whose center is outside a particular triangle (but close to its edge) can still affect the color that is rendered into this triangle. This is especially apparent if linear interpolation and mip-mapping (Woo et al., 1997) are used for rendering the triangles. The continuous mapping in the texture-map images makes sure that the local neighborhood of a pixel contains the

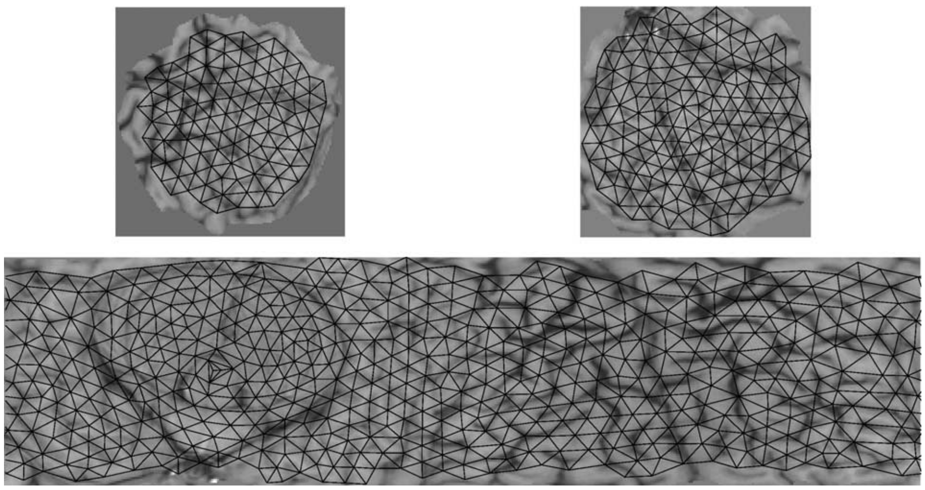
Fig. 5. Optimized mapping and the generated texture images for the sample mesh. Edge lines for the added border triangles are not shown for clarity reasons.

true neighborhood for the original data. Furthermore, the extra border triangles are used for generating texture map data to avoid any edge effects and to prevent the seams to become visible when the mesh parts are joined. The extra border triangle replaces the missing edge triangle that belongs to the neighboring part. Thus, the mapping and the texture image is continuous to even outside of the actual triangle mesh part. Pixels not inside any of the triangles can be given a color that is an average of the other pixels. Fig. 5 shows texture-map images for our sample mesh with the optimized mesh overlaid (extra border triangle edges are not shown). The aspects of aliasing and texture seams are considered further with the adjustment of texture generation parameters.

Optimization of the OpenGL primitives

As a final step for fast rendering, the triangles in the three separate parts are arranged into triangle strips and triangle fans to reduce the amount of per-vertex data that are transferred to OpenGL and preprocessed (Evans et al., 1996). Our simple method minimizes the number of separate triangles (Akeley et al., 1990) and results in more than 50% savings in the amount of transferred vertex data.

Adjustment of voxel–pixel correspondence ratio

The voxel–pixel correspondence ratio is determined by the parameter β used for scaling the original triangle mesh when converting it to voxel units. This parameter defines how many pixels, in the optimal case, should be dedicated for a unit voxel distance. The value for the optimal correspondence ratio depends on the texture generation method (depth integration in our case) and the method parameters.

For a moment, let us consider simple point-like re-sampling (zero integration volume). Then, the β parameter can be thought to effectively define the sampling rate for the texture map image with respect to the sampling rate of the MRI set. The original sampling of the MR images defines the highest possible spatial frequency present in the data. If the original signal is fully reconstructed and then re-sampled, the new sampling frequency can be the same without any loss of information. The full reconstruction can be done with sinc interpolation as sinc function is the Fourier transform of the ideal low-pass filter (Gonzalez and Woods, 1992). Any other interpolation will have a non-ideal frequency response.

Sinc interpolation is not widely used because the calculation is very time consuming. Instead, the most common method is linear interpolation which is fast to compute. For a comparison of other widely used interpolators, see for example Lehmann et al. (1999, 2001). As OpenGL uses linear (or alternatively nearest neighbor) interpolation when it rasterizes texture-mapped triangles onto screen, sinc interpolation cannot be used throughout the whole texturing process and the ideal β = 1 correspondence ratio is not necessarily enough. Although the frequency characteristics of the volume integration are difficult to estimate, the procedure basically takes an average through the integration volume and thus has a smoothening (i.e., low-pass filtering) effect. On the other hand, volume integration also has potential to generate more details (i.e., higher frequencies) as the direction of depth integration might change very rapidly if local surface curvature is high.

Because the frequency response of linear interpolation goes first time to zero at two times the frequency compared with that for sinc interpolation, value β = 2 seems reasonable. Smaller values might cause visible aliasing as the frequency response of the linear interpolation is still at 40% level for the cut-off frequency of the ideal low-pass filter. On the other hand, we could use even higher values for β to avoid aliasing from the first few side lopes that have peak values of 4.7% and 1.6% (1st and 2nd side lope, respectively). However, the consumption of texture-map memory grows as a function of β^2.

Due to the considerations above, we decided to use value β = 2. This should produce only minimal amount of aliasing provided that the optimized mapping does not stretch texture image excessively and that the local surface curvature is low. Both of these assumptions hold for the brain surface in general, and only cylindrical structures (i.e., spine) cause larger optimization errors (see Results section). For the brain surface used for our method (sulci closed), the spine is also the only location where surface curvature is high compared with the texture-map pixel size.

Our experiments (see Fig. 6) confirm the above considerations. With β = 1, clear telltale traces of linear interpolation (performed by OpenGL) can be seen on closeups: saw-like edges in diagonal lines (upper arrow) and loss of detail (smearing of a vein, lower arrow). With β = 2, such artifacts are mostly gone and β = 3 does not seem to provide any further details.

Adjustment of texture generation parameters

As described previously, our texture images are created by depth integration from the original T_1-weighted MRI data. The parameter affecting the outcome most is the integration depth, as is illustrated in Fig. 7. Shallow integration depths produce higher accuracy of surface details but the images are also noisy and they can be affected by small errors in surface segmentation and triangulation. Larger integration depths smooth out the information

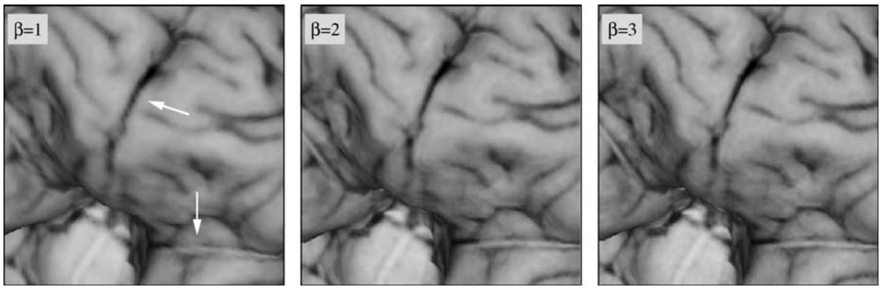

Fig. 6. Comparison for the effect of the β parameter values. Arrows point out some structures that show the linear interpolation artifacts (saw-like edges) and loss of detail with β = 1 but are gone with higher values.

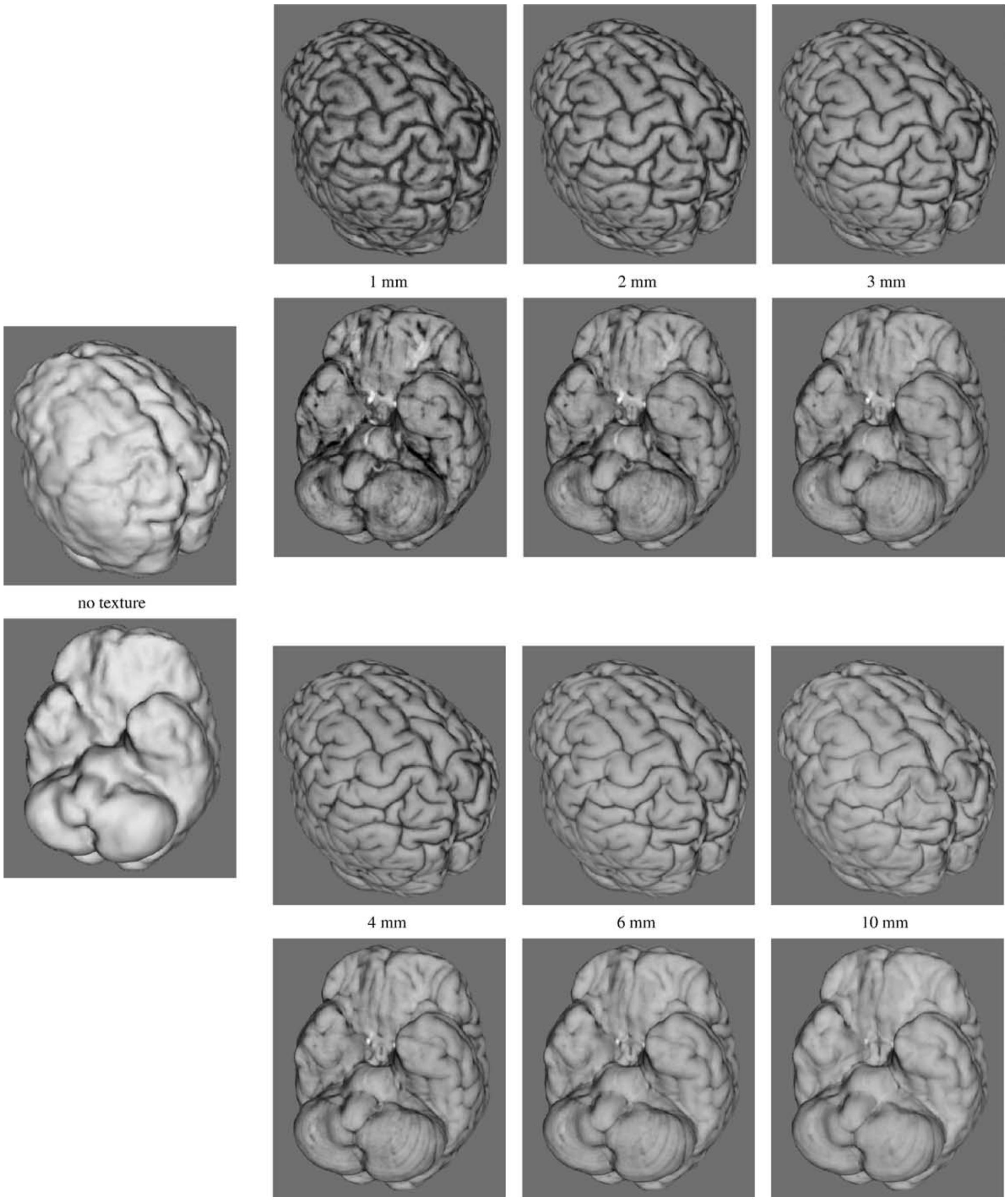

Fig. 7. Effect of the integration depth on generation of texture maps. Image pair on the left shows the triangle mesh without texture maps to give an idea of the surface details that the mesh follows. Rest of the images show the same mesh with texture maps applied. Only the integration depth is changed between different images.

through larger volume and show less details. Thus, the selection for integration depth depends on the application and all depths from 1 to 6 mm seem to be usable. We suggest 3 mm as a reasonable default value.

Selection for the base of the integration volume also affects the outcome. Two different approaches, absolute and relative, are possible, both with their own benefits. An absolute base fixes the base size to be constant in absolute units. The size can be determined from voxel dimensions but after that it is the same for every texture-map pixel generated. Benefit of this approach is that every pixel uses the same integration volume, no matter how the respective mapped triangle is deformed. Thus, pixels close to the seam of one part have necessarily similar colors as those at the respective position of another texture-map part. The added extra border triangles help here to generate texture images across the seams in a continuous manner. In different mesh parts, exactly the same data are re-sampled for the edges but possibly with slightly different resolution as the stretch along the edge is not necessarily the same. In practice, this still guarantees that no visible seams are produced when the mesh parts are joined. The drawback of this approach is that fixed base size cannot compensate for the compression or stretching of the triangles. Thus, aliasing and possibly loss of details may occur if the integration base size is smaller than the separation of adjacent pixels. On the other hand, increasing base size causes integration over larger volume and thus smoothes the details.

In the relative approach, the pixel size on the texture image is inverse mapped back to the original mesh and MR images. Scaled version of this inverse-mapped pixel is then used as the base for the integration. This way, the compression or stretching of the triangle in different directions is taken into account, and in general an asymmetric integration base is produced. Thus, the integration volume changes from triangle to triangle and no details are lost as the integration volume adapts to the triangle scaling. Texture aliasing can be also avoided with this adaptive re-sampling by keeping the integration base size equal to or larger than the inverse mapped pixel. As a drawback, the triangles generally have different scalings on the edges of different parts and thus a slightly different integration volumes are used for them. In principle, this could produce visible seams in the texture-map colors when the mesh parts are joined.

In practice, we have mostly used the relative approach with unit scaling, meaning that the inverse-mapped pixel is used for the integration base as such. We have not been able to observe the seams in the texture-mapped brain surfaces even while knowing their exact location. When we increased the scaling to three for test purposes (integration base area nine times the inverse-mapped pixel area), the borders were seen in some cases when high-contrast surface details were present.

To calculate the volume integration result for each pixel, we sample through the specific volume with a given resolution. At these sample locations, the gray-scale value is calculated by trilinear interpolation from the original MRI data. For the integration resolution, we select half of the smallest dimension of the MRI voxel.

Results

While developing and testing the method, we have used it with several different T_1-weighted MRI volumes with in-slice resolution of about 1 mm and inter-slice distance varying between 1 mm and 1.4 mm. The volumes were segmented for the brain and the surface was tessellated with triangle meshes of varying sizes. These meshes along with the original MRI volumes were used for creating texture-mapped meshes for our method. For the numerical results provided here, we selected our highest resolution T_1-weighted MRI volume with 1 mm × 1 mm × 1 mm voxel size.

In the following subsections, we consider the preprocessing times (i.e., generation of the texture-mapped meshes), the rendering speeds, the quality of the rendered surfaces, and the effect of using anisotropic voxels.

Preprocessing times

Fig. 8 shows example preprocessing times for different mesh sizes on a 3.00-GHz Intel Pentium 4 workstation. The preliminary mapping, optimization, and texture generation are explored in more detail below as they are the parts we have contributed to. The OpenGL optimization uses a method directly from the literature.

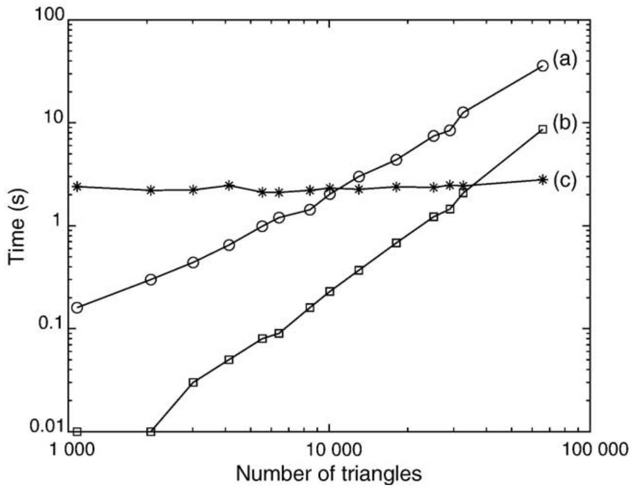

Fig. 8. Preprocessing times for different mesh sizes. Curve (a) shows the time for preliminary mapping and optimization together, curve (b) the time for OpenGL primitive optimization, and curve (c) the time for texture map generation.

Naturally, the time needed for the preliminary mapping and the optimization increases with the number of triangles used (see Fig. 8). The computational costs for one evaluation of the error functions in both cases are linearly dependent on the number of triangles (i.e., O(N)). As the surface tessellation gets finer, the sampled local surface curvature increases and unfolding to plane inevitably produces larger errors, on average. Furthermore, the propagation of error across the mesh slows down as the error from one triangle mediates only to the neighboring ones. Thus, the optimization needs more error function evaluations and the total time is larger than expected from a simple linear dependency.

A crucial factor to speed up the preliminary mapping and the optimization steps was the use of different hierarchy levels for the meshes which speeded up the propagation of error and thus reduced the number of error function evaluations. The effect grew larger as the mesh size increased, and almost a ten-fold speed-up was obtained for large meshes (>50,000 triangles).

Cylindrical structures proved to be difficult and very time consuming to be flattened on a plane. Fig. 9 shows an example of brain surface mesh where a long protrusion is created by segmenting and tessellating also the brain stem and part of the spine. The enlargement on the right shows this cylindrical structure opened on a plane.

As the base of the structure is opened on the plane, all further segments must map inside the region as we progress towards the tip. This is the only way not to make the triangles overlap. As a result, all further segments will have less and less area available. The optimization algorithm progresses very slowly as only minute changes in those vertex positions can be made without producing flipped and overlapping triangles. The use of different hierarchy levels does not speed up this portion as only triangles starting from the tip of the thin protrusion can be removed between different levels. Triangles are removed by vertex collapse operation (Hormann et al., 1999) and on the side of a narrow structure that would lead to a cut.

The computing time used for the preliminary mapping and optimization depends on the size of the mesh and whether those cylindrical structures are present or not. Tests on a 3.00-GHz Intel Pentium 4 workstation gave times from 1 s (6000 triangles) up to

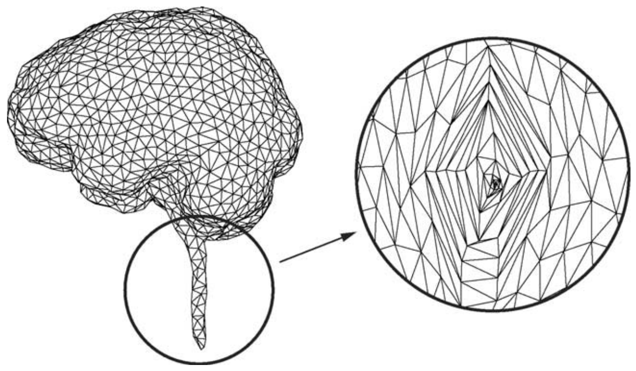
Fig. 9. Example of a cylindrical protrusion causing problems to open it onto 2D plane.

5 s (18,000 triangles) for recommended mesh sizes (Fig. 8). With a finer mesh and when also the brain stem and part of the spine are tessellated, the computation can easily take 1–5 min to end up with a satisfactory result.

The time needed for texture-map generation depends on the size of the texture-map image, on the size of the integration volume, and on the integration resolution. As can be seen from Fig. 8 (curve c), the computing time is almost independent of the triangle mesh size. Although all the triangles are traversed through, the time for that is negligible compared with the time for volume integration. A typical total computing time ranges from 1 to 10 s depending mainly on depth integration parameters (and texture map sizes). The sample times of Fig. 8 are 2–3 s, as run on a 3.00-GHz Intel Pentium 4 platform with integration depth of 3 mm, integration resolution of 0.5 mm, and using relative base size.

Rendering speed

The rendering speed of our method depends mainly on the number of triangles in the texture-mapped mesh. The choice for the triangle mesh size also affects the quality of the resulting image which is explored in the next subsection. Using more triangles produces more accurate surface representation but it also increases the rendering time. The optimal trade-off between these two goals depends on the application, viewport (window) size, and the OpenGL hardware present.

Table 1 shows rendering speeds for five different mesh sizes (from 5542 to 217,733 triangles) run on three different platforms using two viewport sizes. The view was set to barely enclose the minimal bounding sphere of each mesh inside the viewport. The times shown are averages of 360 renderings where the mesh was rotated around its vertical axis with 1° steps. The tests were run both for plain colored and textured meshes, except in the last case where the rendering speed of larger amount of plain-colored triangles was investigated. That mesh was created by segmenting and triangulating the cortex into sulci and no texture maps were created for it.

Rendering quality

Like mentioned above, the size of the triangle mesh affects also the quality of the resulting renderings. Higher amount of triangles allows finer surface tessellation and the geometry of the brain can be reproduced more accurately. Which size produces “good” quality visualizations is always ultimately a subjective opinion. To form a general idea of usable mesh sizes, we inspected several texture-mapped meshes of different sizes created from different MRI volumes. These meshes were rendered with many viewport (window) sizes and the resulting visualizations were inspected while interactively rotating them.

Our tests indicated that meshes with triangle side lengths of 10 mm down to 5 mm (1500–6000 triangles) are suitable for window sizes up to 512 × 512 pixels and possibly when no devoted OpenGL hardware is available. Closer inspection, especially with interactive rotation of the mesh, still reveals that the surface is fairly smooth with texture images placed on top of it. Triangle side lengths of 5 mm down to 3 mm (6000–20,000 triangles) were suitable for full screen (1280 × 1024 resolution) visualizations. Tests with stereo visualization showed that 3 mm side length was necessary for those cases because users were able to see the true 3D surface shape. Side lengths smaller than 3 mm did not produce any particular visual benefit but increased greatly the amount of triangles and thus also the rendering time. For example, 1.5 mm side length produces about 70,000 triangles for the brain surface with sulci closed.

Non-cubic voxels

We also explored the visual difference for non-cubic versus cubic voxels. In principle, that should not give any difference

Table 1
Rendering speed in frames-per-second (fps) for HP Visualize C3600 workstation (552 MHz CPU) with Visualize-fx4 graphics accelerator (HP-fx4), 700-MHz Athlon-based Linux platform with NVidia GeForce4 MX 440-SE graphics board (GeForce4), and 3.00-GHz Pentium 4 Linux platform with NVidia Quadro4 980 XGL graphics board (Quadro4)

Triangles	Edge (mm)	Viewport	HP-fx4		GeForce4		Quadro4	
			Plain	Tex.	Plain	Tex.	Plain	Tex.
5 542	5.2	512 × 512	420	240	480	440	2300	2000
		1024 × 1024	250	130	160	150	680	640
12 990	3.5	512 × 512	190	120	370	350	1400	1300
		1024 × 1024	160	90	140	140	610	580
25 134	2.5	512 × 512	100	63	270	250	820	780
		1024 × 1024	98	56	130	120	510	480
65 892	1.6	512 × 512	38	24	130	130	340	320
		1024 × 1024	38	23	88	84	290	280
217 733	1.5	512 × 512	10	–	45	–	110	–
		1024 × 1024	10	–	38	–	100	–

For each platform, we give results for plain-colored mesh rendering (Plain) and for texture-mapped mesh rendering (Tex.). Second column (Edge) provides the average side length of the triangle mesh.

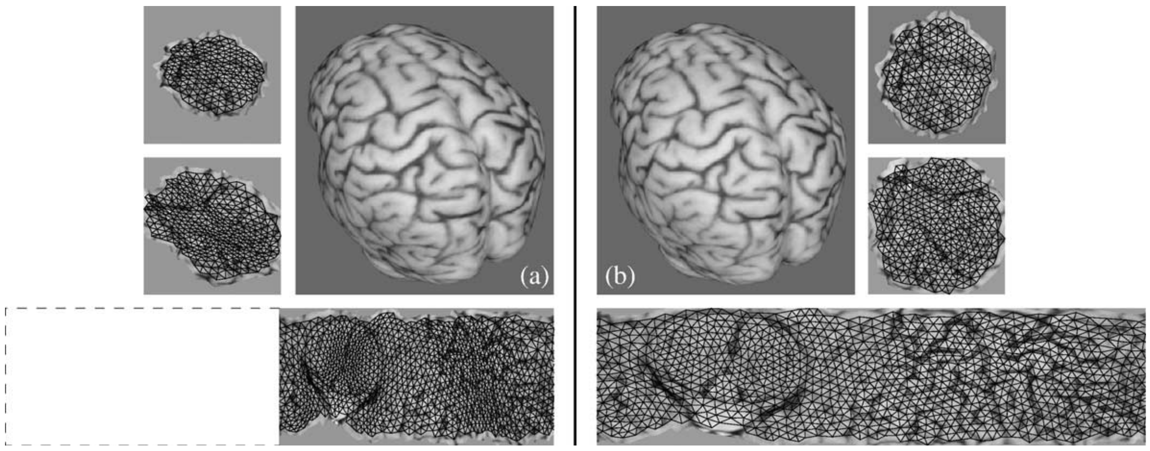

Fig. 10. Comparison of optimized mappings, textures, and resulting visualizations. Left-side images (a) are for an MRI set with slice thickness twice the in-slice pixel size. Right-side images (b) are for an MRI set interpolated to cubic voxels. Dashed line shows the saving in texture map area.

because voxels and pixels are used as comparable units of measurement. This leads to the optimization of the texture map resolution in accordance to the volume data resolution. For the images presented in Fig. 10, we took the example MRI set (voxel size 1 mm × 1 mm × 1 mm) and re-sliced it to 2 mm slice thickness by averaging every two slices. From this down-sampled MRI set, we created yet another set by linearly interpolating new slices so that we obtained again cubic voxels. These two new sets simulated a situation where the MR images would be acquired with thicker slices and would be then interpolated to cubic voxels.

We used both of the new MRI sets and a mesh created for the original 1 mm^3 voxel set to create optimized mappings and texture images. Fig. 10 shows the result. Left side (a) shows the case with 2-mm slices and right side (b) the case interpolated from that to cubic voxels. On the left side, the optimized triangles are more or less compressed in one direction compared with the right-side images. Only the few triangles in-plane with the MRI set slices are not compressed and the triangles perpendicular to the slice plane orientation are compressed most as the resolution of the MRI data is lower in that direction. The optimal texture sizes were 256 × 256 for all square texture maps, 512 × 256 for the wrapped texture map in case (a), and 1024 × 256 for the wrapped texture map in case (b).

The large panels in Fig. 10 show the resulting surface visualizations with integration depth of 4 mm. As expected, practically no differences can be observed between these images. The linear texture interpolation performed by the OpenGL system produces equivalent results when compared with linear interpolation of the original MRI data. Thus, it is best to use the acquired MR images as such and benefit from the savings in both runtime memory for texture generation and in texture map memory of the OpenGL rendering.

Discussion

The method presented in this paper allows fast and realistic-looking visualizations of the brain surface. As compared to plain-colored triangle meshes, the use of texture mapping allows us to increase the triangles' side length which leads to a smaller number of triangles. The count is reduced even further as segmentation and tessellation into brain sulci is avoided. As shown, the time needed for preprocessing in our method is about 10 s in typical cases for recommended mesh sizes. Higher-resolution meshes take 10–60 s and problem cases (i.e., spine included) 1–5 min. All this needs to be calculated only once per mesh and the results can be stored for later use. Furthermore, as the complicated process of segmenting and tessellating the cortical mantle into sulci is not necessary, the preprocessing time required for our method can be easily saved in that phase.

Due to the reduced number of triangles, the rendering of the texture-mapped mesh of our method outperforms even the solid colored triangle meshes, not to mention volume rendering techniques. Still, the quality of visualization for brain surface details is excellent and on par with the volume rendering (see Fig. 1 for comparison). Tessellation into brain sulci with fine mesh is necessary for the solid colored triangles to show enough surface details (last row of Table 1). On the other hand, for our method, textured triangles with side lengths of 2.5–3.5 mm are enough for even full screen visualizations. As can be seen from the table, using smaller number of textured triangles instead of large number of solid-colored triangles provides 3- to 10-fold increase in speed depending on the platform and viewport size. For volume rendering, the speed depends on many things like the selected method, exact method parameters, volume size, and viewport size. Typical frame rates for visualizing brain MRI volumes fall into 0.1–10 fps (see Fig. 1 for examples).

Although texture-mapped triangle meshes are widely used in computer graphics, to our knowledge, they have not been previously applied for visualizing brain surface anatomy from MR images. Texture-mapped meshes require the suitable texture images and our key contribution was to notice the applicability of volume rendering techniques for texture generation. This allows us to create textured surfaces with visual quality of volume renderings. Like mentioned, the individual steps of our method are not new but they result in a novel visualization solution. Our further contributions are in the preliminary mapping and optimization phases. The error function for the spherical parametrization (Eq. (1)) is new and especially suitable for the specific requirements of our method. In the final optimization of the mapping, the error function (Eq. (2)) is also novel. Along with the new idea of relating pixels and voxels as units of measurement, this allows us to optimize the mapping according to the resolution of data available. Typically, the mapping has been optimized to minimize the geometrical distortion of the triangles.

We made many choices when building up the method, some of which can be criticized. Our way of dividing the mesh into three parts is not necessarily the best one. It seems suitable for typical brain surfaces, but some other ways may be better for different kinds of tessellated surfaces. Furthermore, the use of wrapped texture map has drawbacks as it effectively inflicts a static one-direction scaling for all the triangles in that part. The benefit is the reduction in the number of seams and the better optimization of OpenGL primitives, but these effects may be considered negligible. For easy implementation of this method, the possibility for wrapped texture maps can be discarded as that only complicates the implementation.

Cylindrical protrusions (such as spine) were difficult to open to 2D plane and to optimize. This difficulty could be possibly circumvented by an algorithm detecting those structures and separating them into their own parts. Furthermore, if the amount of seams is not an issue, methods that allow tearing of the flattened mesh to bound the distortions (Sorkine et al., 2002) could be used. Nevertheless, in typical brain-research applications, the spine can be left out and thus we did not address this problem in detail.

The error function (2) can be easily changed so that the terms associated with loss of surface details ($s_{i,\max}$ and $s_{i,\min}$) are weighted differently from those increasing with wasted texture-map area ($1/s_{i,\max}$ and $1/s_{i,\min}$). These factors are not necessarily equally desirable, but in some cases the gain on surface details can be greatly preferred over the wasted texture map area. However, for typical brain meshes, the compression/stretching error is very low on average and smoothly distributed so that this is not of much concern. Only the cylindrical protrusions cause local concentrations of error as they necessarily distort the triangles (Fig. 9).

We believe that the method described here proves to be very efficient for brain surface visualization. Nowadays, even laptop computers have graphic cards capable for fast OpenGL rendering with texture-mapped triangles. The texture-mapped mesh object can be easily combined with other OpenGL objects and methods to produce more complex renderings. For example, the realistic looking brain surface can be accompanied with solid-colored triangle-mesh objects visualizing scalp and skull surfaces, all this cut with standard OpenGL cut planes to show inside details with anatomical and functional data. If these kind of complex scenes are rendered into stereo buffers for a large screen, the rendering speed and the quality of individual scene components become crucial. With the method introduced here, interactive frame rates can be maintained even with widely available consumer-level graphics cards.

Acknowledgments

The work of Mika Seppä was supported by a grant from the Foundation of Vilho, Yrjö, and Kalle Väisälä (Finnish Academy of Sciences and Letters). Matti Hämäläinen was supported by the MIND institute (USA). We thank Prof. Riitta Hari and Dr. Kimmo Uutela for their comments on the manuscript.

References

Akeley, K., Haeberli, P., Burns, D., 1990. The tomesh.c program. C Program on SGI Developer's Toolbox CD.

Alexa, M., 2002. Recent advances in mesh morphing. Comput. Graph. Forum 21, 173–196.

Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., Desbrun, M., 2003. Anisotropic polygonal remeshing. ACM Trans. Graph 22, 485–493.

Blinn, J.F., Newell, M.E., 1976. Texture and reflection in computer generated images. Commun. ACM 19, 542–547.

Bomans, M., Höhne, K.-H., Tiede, U., Riemer, M., 1990. 3-D segmentation of MR images of the head for 3-D display. IEEE Trans. Med. Imag. 9, 177–183.

Cabral, B., Cam, N., Foran, J., 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In: Kaufman, A., Krüger, W. (Eds.), Proc. Vol. Vis. '94. ACM Press, New York, USA, pp. 91–98.

Eck, M., DeRose, T., Duchamp, T., Hoppe, H., Lounsbery, M., Stuetzle, W., 1995. Multiresolution Analysis of Arbitrary Meshes. Proc. SIGGRAPH '95. ACM Press, New York, USA, pp. 173–182.

Elvins, T.T., 1992. A survey of algorithms for volume visualization. Comput. Graph. 26, 194–201.

Evans, F., Skiena, S., Varshney, A., 1996. Optimizing triangle strips for fast rendering. In: Yagel, R., Nielson, G.M. (Eds.), Proc. Vis. '96. IEEE Computer Society and ACM, San Francisco, CA, USA, pp. 319–326.

Fischl, B., Liu, A., Dale, A.M., 2001. Automated manifold surgery: constructing geometrically accurate and topologically correct models of the human cerebral cortex. IEEE Trans. Med. Imag. 20, 70–80.

Fischl, B., Sereno, M.I., Dale, A.M., 1999. Cortical surface-based analysis: II. Inflation, flattening, and a surface-based coordinate system. NeuroImage 9, 195–207.

Floater, M.S., 1997. Parametrization and smooth approximation of surface triangulations. Comput. Aided Geom. Des. 14, 231–250.

Fuchs, M., Wagner, M., Wishmann, H.-A., Ottenberg, K., Dössel, O., 1993. Possibilities of functional brain imaging using a combination of MEG and MRT. Oscillatory Event Related Brain Dynamics. Plenum Press, New York, pp. 435–457.

Funkhouser, T., Min, P., Kazhdan, M., Chen, J., Halderman, A., Dobkin, D., 2003. A search engine for 3D models. ACM Trans. Graph. 22, 83–105.

Gonzalez, R.C., Woods, R.E., 1992. Digital Image Processing. Addison Wesley, Reading, MA.

Gotsman, C., Gu, X., Sheffer, A., 2003. Fundamentals of spherical parameterization for 3D meshes. ACM Trans. Graph. 22, 358–363.

Gu, X., Yau, S.-T., 2003. Global conformal surface parametrization. Proc. Euro-graphics/ACM SIGGRAPH Symp. Geom. Proc. Eurographics Association, Aire-la-Ville, Switzerland, pp. 127–137.

Gu, X., Gortler, S.J., Hoppe, H., 2002. Geometry images. Proc. SIGGRAPH '02. ACM Press, New York, USA, pp. 355–361.

Guskov, I., Sweldens, W., Schröder, P., 1999. Multiresolution signal processing for meshes. Proc. SIGGRAPH '99. ACM Press, New York, USA, pp. 325–334.

Haker, S., Angenent, S., Tannenbaum, A., Kikinis, R., Sapiro, G., Halle, M., 2000. Conformal surface parametrization for texture mapping. IEEE Trans. Vis. Comput. Graph. 6, 181–189.

Heckbert, P.S., 1986. Survey of texture mapping. IEEE Comput. Graph. Appl. 6, 56–67.

Hormann, K., Greiner, G., 2000. MIPS: an efficient global parametrization method. In: Laurent, P.-J., Sablonniére, P., Schumaker, L.L. (Eds.), Curve and Surface Design: Saint-malo 1999. Vanderbilt Univ. Press, Nashville, USA, pp. 153–162.

Hormann, K., Greiner, G., Campagna, S., 1999. Hierarchical parametrization of triangulated surfaces. In: Girod, B., Niemann, H., Seidel, H.-P. (Eds.), Proc. Vision, Model., Vis. '99. Infix, Erlangen, Germany, pp. 219–226.

Hurdal, M.K., Bowers, P.L., Stephenson, K., Sumners, D.L., Rehm, K., Schaper, K., Rottenberg, D.A., 1999. Quasi-conformally at mapping the human cerebellum. Proc. 2nd Int. Conf. on Med. Image Comput. and Computer-Assisted Interv. Springer-Verlag, London, UK, pp. 279–286.

Khodakovsky, A., Litke, N., Schröder, P., 2003. Globally smooth parameterizations with low distortion. ACM Trans. Graph. 22, 350–357.

Kreeger, K., Kaufman, A., 1999. Hybrid volume and polygon rendering with cube hardware. Proc. ACM SIGGRAPH/EUROGRAPHICS Workshop Graph. Hardware. ACM Press, New York, USA, pp. 15–24.

Lacroute, P., Levoy, M., 1994. Fast volume rendering using a shear-warp factorization of the viewing transformation. Comput. Graph. 28, 451–458.

Lee, A.W.F., Sweldens, W., Schröder, P., Cowsar, L., Dobkin, D., 1998. Maps: multiresolution adaptive parameterization of surfaces. Proc. SIGGRAPH '98. ACM Press, New York, USA, pp. 95–104.

Lehmann, T.M., Gönner, C., Spitzer, K., 1999. Survey: interpolation methods in medical image processing. IEEE Trans. Med. Imag. 18, 1049–1075.

Lehmann, T.M., Gönner, C., Spitzer, K., 2001. Addendum: B-spline interpolation in medical image processing. IEEE Trans. Med. Imag. 20, 660–665.

Levoy, M., 1988. Display of surfaces from volume data. IEEE Comput. Graph. Appl. 8, 29–37.

Levoy, M., 1990. A hybrid ray tracer for rendering polygon and volume data. IEEE Comput. Graph. Appl. 10, 33–40.

Lévy, B., Petitjean, S., Ray, N., Maillot, J., 2002. Least squares conformal maps for automatic texture atlas generation. Proc. SIGGRAPH '02. ACM Press, New York, USA, pp. 362–371.

Lorensen, W.E., Cline, H.E., 1987. Marching cubes: a high resolution 3D surface construction algorithm. Comput. Graph. 21, 163–169.

Ma, K.-L., Schussman, G., Wilson, B., Ko, K., Qiang, J., Ryne, R., 2002. Advanced visualization technology for terascale particle accelerator simulations. Proc. ACM/IEEE Supercomputing '02. IEEE Computer Society Press, Los Alamitos, USA, pp. 1–11.

Maillot, J., Yahia, H., Verroust, A., 1993. Interactive texture mapping. Proc. SIGGRAPH '93. ACM Press, New York, USA, pp. 27–34.

Marquardt, D.W., 1963. An algorithm for least-squares estimation of nonlinear parameters. J. Soc. Ind. Appl. Math. 11, 431–441.

Meissner, M., Huang, J., Bartz, D., Mueller, K., Crawfis, R., 2000. A practical evaluation of popular volume rendering algorithms. Proc. IEEE Symp. Vol. Vis. '00. ACM Press, New York, USA, pp. 81–90.

Praun, E., Hoppe, H., 2003. Spherical parametrization and remeshing. ACM Trans. Graph. 22, 340–349.

Rezk-Salama, C., Engel, K., Bauer, M., Greiner, G., Ertl, T., 2000. Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization. In: Spencer, S.N. (Ed.), Proc. ACM SIGGRAPH/EUROGRAPHICS Workshop Graph. Hardware. ACM Press, New York, USA, pp. 109–118.

Sander, P.V., Snyder, J., Gortler, S.J., Hoppe, H., 2001. Texture mapping progressive meshes. Proc. SIGGRAPH '01. ACM Press, New York, USA, pp. 409–416.

Schröder, P., Sweldens, W., 1995. Spherical wavelets: efficiently representing functions on the sphere. Proc. SIGGRAPH '95. ACM Press, New York, USA, pp. 161–172.

Sorkine, O., Cohen-Or, D., Goldenthal, R., Lischinski, D., 2002. Bounded-distortion piece-wise mesh parametrization. Proc. Vis. '02. IEEE Computer Society, Washington, USA, pp. 255–362.

Suri, J.S., Singh, S., Reden, L., 2002a. Computer vision and pattern recognition techniques for 2-D and 3-D MR cerebral cortical segmentation (part I): a state-of-the-art review. Pattern Anal. Appl. 5, 46–76.

Suri, J.S., Singh, S., Reden, L., 2002b. Fusion of region and boundary/surface-based computer vision and pattern recognition techniques for 2-D and 3-D MR cerebral cortical segmentation (part-II): a state-of-the-art review. Pattern Anal. Appl. 5, 77–98.

Turk, G., 2001. Texture synthesis on surfaces. Proc. SIGGRAPH '01. ACM Press, New York, USA, pp. 347–354.

Wagner, M., Fuchs, M., Wischmann, H.-A., Otternberg, K., Dössel, O., 1995. Cortex segmentation from 3D MR images for MEG reconstructions. Biomagnetism: Fundamental Research and Clinical Applications. IOS Press, Amsterdam, pp. 433–438.

Westover, L., 1990. Footprint evaluation for volume rendering. Comput. Graph. 24, 367–376.

Wilson, B., Ma, K.-L., McCormick, P.S., 2002. A hardware-assisted hybrid rendering technique for interactive volume visualization. Proc. IEEE Symp. Vol. Vis. Graph. '02. IEEE Press, Piscataway, USA, pp. 123–130.

Woo, M., Neider, J., Davis, T., 1997. OpenGL programming guide. Addison Wesley Developers Press, Reading, Massachusetts, pp. 338–344. Chapt. 9.