Efficient Approximation of Gray-Scale Images through Bounded Error Triangular Meshes

Miguel Angel García
Dept. of Computer Science and Mathematics
Rovira i Virgili University
Ctra. Salou s/n. 43006 Tarragona, Spain
magarcia@etse.urv.es

Boris Xavier Vintimilla *Angel Domingo Sappa*
Inst. of Organization and Control of Industrial Systems
Polytechnic University of Catalonia
Diagonal 647. 08028 Barcelona, Spain
{vintimi, sappa}@ioc.upc.es

Abstract

This paper presents an iterative algorithm for approximating gray-scale images with adaptive triangular meshes ensuring a given tolerance. At each iteration, the algorithm applies a non-iterative adaptive meshing technique. In this way, this technique converges faster than traditional mesh refinement algorithms. The performance of the proposed technique is studied in terms of compression ratio and speed, comparing it with an optimization-based mesh refinement algorithm.

1. Introduction

Images are commonly transferred and stored in compact form through well-known representations, such as GIF and JPEG. Unfortunately, these representations are not well-suited for applying further image processing operations directly in the compressed domain. An alternative way of compressing images is through the utilization of geometric representations, such as triangular meshes. Geometric representations are applicable since the pixels of an image can be considered to be 3D points in a space in which coordinates x and y are the rows and columns of the image, and coordinate z is the gray level.

The advantage of geometric representations is that further processing can be directly applied in the 3D geometric domain. For instance, scaling, translation and rotation operations can be simply implemented by applying affine transformations to the 3D coordinates of the vertices that constitute the meshes. An algorithm that uniformly samples the resulting meshes suffices to recover the corresponding gray-scale images.

This work has been partially supported by the Government of Spain under the CICYT project TAP96-0868. The second and third authors have been supported by the Spanish Agency for International Cooperation.

Two main geometric approaches have been proposed in the literature. The first approach generates an adaptive triangular mesh by starting with a few triangles and then successively splitting them in order to reduce the approximation error (e.g., [1]). Alternatively, the algorithm can start with a fine mesh and successively make it coarser until the approximation error is above the desired tolerance (e.g., [8]). The second geometric approach applies iterative optimization in order to modify the positions of the control points of a regular grid, such that the approximation error with respect to the original image is minimized (e.g., 6). For its application to gray-scale images, surface discontinuities must be taken into account, such as in [2] and [3]. The two previous approaches can be combined by applying both iterative optimization and refinement (and merging) until a good solution is found, such as in 7.

This paper presents a new algorithm comprised in the first geometric approach. It generates an adaptive triangular mesh that approximates a gray-scale image guaranteeing a maximum *root-mean-square error* (RMS) or *tolerance* with respect to the given image. Contrarily to previous adaptive meshing techniques that start with either a fine or coarse mesh and progressively make it coarser or finer point after point, the proposed technique applies a fast, non-iterative meshing algorithm at each iteration. Thus, adaptive meshes fulfilling the desired tolerance are obtained faster than with previous techniques.

The proposed algorithm is described in section 2. Experimental results with real gray-scale images are

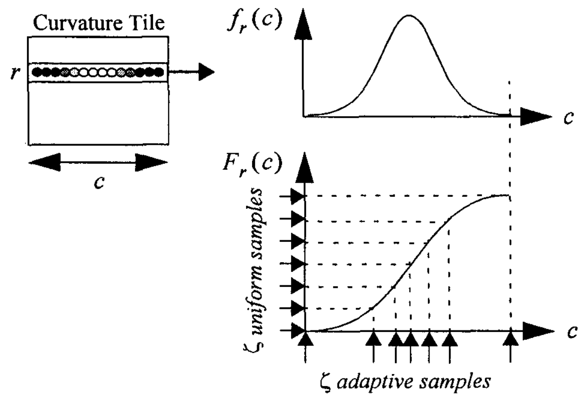

Figure 1. The curvatures associated with each row of an image tile are interpreted as a discrete probability density function. When the image space of the corresponding probability distribution function is uniformly sampled, the positions of the chosen pixels at that row are obtained.

shown in section 3. Conclusions and future work are given in section 4.

2. Image Approximation

Given a gray-scale image, it is approximated by a triangular mesh through an iterative process consisting of four main stages that are described below.

2.1. Initial adaptive triangulation

An initial triangular approximation of the given gray-scale image is obtained in two stages. In the first stage a predefined number of pixels is chosen from the image by applying a non-iterative adaptive sampling technique proposed in [4][5]. In the second stage, the chosen pixels are triangulated through a 2D Delaunay algorithm [9]. The adaptive sampling process is summarized below.

First, a *curvature image* is estimated from the original image as described in [4][5]. Figure 2(*top-right*) shows an example. Both the original and curvature images are divided into a predefined number of rectangular *tiles*. The following steps are independently applied to each tile.

First, a predefined number of pixels is chosen for each row of every tile according to the previously computed curvature (more pixels are chosen in high curvature areas) as illustrated in Figure 1. After the previous sampling, a set of *vertical curves* is obtained, Figure 2(*bottom-left*). Each vertical curve

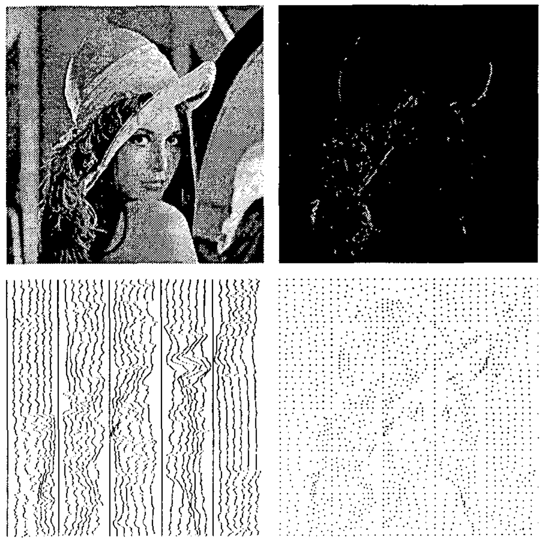

Figure 2. (*top-left*) Original image with 262,144 pixels (512x512). (*top-right*) Curvature image. (*bottom-left*) Adaptively sampled vertical curves (6x5 tiles). (*bottom-right*) Initial set of pixels chosen by the adaptive sampling process (1,548 vertices).

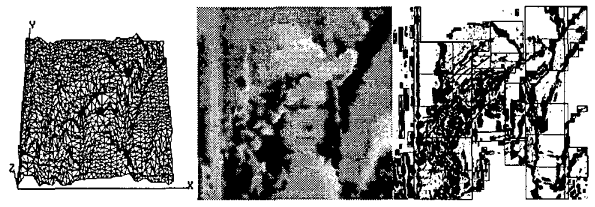

Figure 3. (*left*) 3D adaptive triangular mesh. (*center*) Approximating image obtained through z-buffering in 0.19 sec. (RMS=22.4). (*right*) Binary image showing error regions in black and their enclosing rectangles.

is adaptively sampled by applying a similar principle. Thus, a predefined amount of pixels is obtained for every tile, Figure 2(*bottom-right*). Those pixels are triangulated by applying a 2D Delaunay algorithm to their row and column coordinates [9]. Figure 3(*left*) shows an example of the resultant mesh. The z coordinates of the vertices of that mesh correspond to the gray-levels associated with them in the original image.

2.2. Generation of the approximating image

Once the initial triangular mesh has been computed, it is necessary to obtain its corresponding gray-scale image (*approximating image*) in order to determine the accuracy with which that mesh approximates the original image. This is done by sampling the 3D mesh uniformly at as many positions as pixels has the original image (*mesh backprojection*). By taking advantage of the 3D nature of the triangular mesh, the cost of this sampling process can be significantly reduced by applying z-buffering. This is done in practice by displaying the triangular mesh in a window that contains as many pixels as the original image, through functions of the standard 3D OpenGL library. The z-buffer obtained after this visualization is read through another OpenGL function (*glReadPixels*). The z-buffer values are linearly mapped to gray levels in the range [0...255]. Since the OpenGL implementations in most current computers (including PCs) take advantage of hardware acceleration, the whole process is very fast. For example, the approximating image (with 262,144 pixels) corresponding to the mesh shown in Figure 3(*left*) was obtained in 0.19 seconds on a SGI Indigo II, Figure 3(*center*).

2.3. Determination of error regions

If the RMS of the approximating image is below the specified tolerance, the algorithm concludes. Otherwise, the algorithm proceeds by identifying the regions of the approximating image whose error with respect to the original image is above the tolerance.

First, an error image is obtained by subtracting each pixel of the approximating image from the corresponding pixel of the original image and by taking the absolute value of the result. This error image is converted to a binary image by thresholding it with the given tolerance, Figure 3(right).

All black regions in the binary image represent *error regions* that must be resampled. The binary image is labelled in order to determine the different error regions contained in it. Then, the *enclosing rectangle* for each separate error region is computed. Very small enclosing rectangles (e.g., less than 3x3 pixels) are discarded. Rectangles larger than an initial tile are subdivided.

Each enclosing rectangle is then resampled by applying the technique described in Section 2.1. The number of pixels adaptively sampled over each enclosing rectangle is defined as k times the number of pixels previously sampled over it, with k being a real larger than one. The previously sampled pixels in each enclosing rectangle are substituted for the new ones.

2.4. Delaunay retriangulation

The resampled pixels and the old ones that did not belong to any enclosing rectangle are retriangulated by applying the 2D Delaunay algorithm. At this point, the image approximation algorithm proceeds by iterating from step 2.2 until the RMS of the approximating image is below the given tolerance. The final result for the current example was achieved after 3 iterations, given a tolerance equal to 15. The total CPU time was 7.7 sec. on a SGI Indigo II.

2.5. External representation of triangular meshes

The triangular meshes obtained with the proposed algorithm are kept in a compact representation by only saving the coordinates of each sampled pixel (row, column, gray-level). For 512x512x256 images, 4 bytes per pixel are necessary. After downloading the pixels into memory, they are triangulated by applying the 2D Delaunay algorithm. Hence, there is no need to store the mesh topology as it can be recovered directly from the sampled pixels.

3. Experimental results

The proposed algorithm has been tested upon 512x512x256 real gray-scale images. The CPU times were measured on a SGI Indigo II. Figure 4 shows the final approximating images (backprojected meshes) for three of the test images. The top row shows approximating images corresponding to tolerances (RMS errors) equal to 12 (*left*) and 14 (*right*). The associated meshes contain 7,772 and 5,807 vertices. Considering 4 bytes per vertex, this corresponds to compression ratios of 8:1 and 11:1 respectively. These meshes were obtained in 10.3 and 7.9 seconds. The middle row shows results corresponding to tolerances equal to 14 and 16. The compression ratios are 9:1 and 12:1. The associated triangular meshes were obtained in 7.9 and 7.8 seconds. Finally, the bottom row shows the result corresponding to tolerances equal to 12 and 13. The compression ratios are 7:1 and 10:1. The meshes were obtained in 11.8 and 8.7 seconds respectively.

The proposed algorithm has been compared with a mesh refinement algorithm based on iterative optimization [8]. An implementation of the latter, called Jade, is publicly available. Jade starts with a dense triangular mesh, which in our case contains all the