A Fast, Space-Efficient Algorithm for the Approximation of Images by an Optimal Sum of Gaussians

Jeffrey Childs Cheng-Chang Lu Jerry Potter

Department of Mathematics and Computer Science
Kent State University

Abstract

Gaussian decomposition of images leads to many promising applications in computer graphics. Gaussian representations can be used for image smoothing, motion analysis, and feature selection for image recognition. Furthermore, image construction from a Gaussian representation is fast, since the Gaussians only need to be added together. The most optimal algorithms [3, 6, 7] minimize the number of Gaussians needed for decomposition, but they involve nonlinear least-squares approximations, e.g. the use of the Marquardt algorithm [10]. This presents a problem, since, in the Marquardt algorithm, enormous amounts of computations are required and the resulting matrices use a lot of space. In this work, a method is offered, which we call the Quickstep method, that substantially reduces the number of computations and the amount of space used. Unlike the Marquardt algorithm, each iteration has linear time complexity in the number of variables and no Jacobian or Hessian matrices are formed. Yet, Quickstep produces optimal results, similar to those produced by the Marquardt algorithm.

Key words: Gaussian approximations, geometric algorithms, surface fitting.

1 Background

In the Gaussian decomposition of an image, we wish to find a minimum number of Gaussians whose sum approximates the image with a certain tolerance. Each Gaussian G_i has the form:

$$G_i(X,Y) = A_i e^{-[(X-x_i)^2+(Y-y_i)^2]/2\sigma_i^2} \qquad (1)$$

where (X,Y) are pixel coordinates, A_i is the amplitude of G_i, (x_i, y_i) is the mean (or position) of G_i and σ_i is the standard deviation (or scale) of G_i. Thus, an image can be represented by an array of these four parameters.

Gaussian representations have many promising applications. Gaussian parameters are promising for image transmission, since the Gaussians only need to be added together to reconstruct the image. In image transmission, they can be used for low to high resolution display, where the largest Gaussians are transmitted, added and displayed first, forming the basic structures of the image, while the smaller Gaussians are being transmitted. The smaller Gaussians, when added, fill in the details of the image. Hence, this type of representation allows simultaneous transmission and display from low to high resolution.

For motion analysis applications, Gaussian representation of objects can simplify computation, because a moving object can be simulated by adjusting the Gaussian parameters which represent it. In addition, Gaussians may be used to represent features of images, needed for image matching and recognition.

Once a Gaussian representation of an image is found, the Gaussians can be used, from then on, in a variety of ways. Finding the Gaussian representation, however, is not a trivial problem, and literature in the Gaussian decomposition of images is rather scarce. The greatest advances in optimality were made by Goshtasby and O'Neill [6, 7]. However, their method involves the Marquardt algorithm [10], a nonlinear least-squares approximation algorithm which requires enormous amounts of computations for practical image problems.

A further improvement in optimality was made by Childs *et al.* [3], by bounding Gaussian parameters intrinsically. In their work, which also uses the Marquardt algorithm, intrinsic boundaries were applied

to 1D signals to reduce the number of Gaussians used to represent them. Intrinsic boundaries for position are useful in this work, also, but those for amplitude and standard deviation are not used. The boundaries for position are set beyond the edges of a square image, at one quarter of the distance of one dimension, in pixels, of the image. In other work on Gaussian decomposition, Ben-Arie and Rao developed two very fast algorithms [1]. However, these methods are focused on speed rather than optimality. In contrast, this work attempts to improve the speed without sacrificing optimality. In the appendix of their work, as well as in a comment by Ferreira [5] on Wiener's earlier work, it was shown that any practical signal can be decomposed into a sum of Gaussians, given a tolerance.

In this work, a novel algorithm is introduced, which we call the Quickstep algorithm. It takes a global optimization approach to the number of Gaussians as originated by Goshtasby and O'Neill [6], yet it has a lower time complexity than the Marquardt algorithm.

2 Introduction

If we let a function f be defined as a sum of m Gaussians, such that

$$f(X,Y)=\sum_{i=1}^{m} A_i e^{-[(X-x_i)^2+(Y-y_i)^2]/2\sigma_i^2} \qquad (2)$$

where the parameters are as defined in Equation 1, our goal is to find a function f with minimum m such that the signal is approximated by the sum of m Gaussians, i.e.

$$\max\{|P(X,Y)_j - f(X,Y)_j| : j=1,2,3,...,N\} < \varepsilon \qquad (3)$$

where N is the number of pixels, $P(X,Y)_j$ is the data value at the j^{th} pixel, $f(X,Y)_j$ is the value of the function of Equation 2 at the j^{th} pixel, ε is a prescribed tolerance, and pixel j has coordinates (X,Y).

Let us define a vector $\boldsymbol{b}$ of length n, composed of Gaussian parameters, such that

$$\boldsymbol{b}^{[n]} = (A_1, \alpha_1, \beta_1, \sigma_1, A_2, \alpha_2, \beta_2, \sigma_2, ..., A_m, \alpha_m, \beta_m, \sigma_m). \qquad (4)$$

where α_i and β_i are the intrinsic boundary counterparts to x_i and y_i, respectively [3]. The Marquardt algorithm attempts to make corrections to $\boldsymbol{b}$ to meet the prescribed tolerance by combining Newton's method [8] with the gradient method [4]. The gradient method determines the direction of correction, while the Newton method determines the amount of correction.

In every iteration of the Marquardt algorithm, a partial derivative of function f is taken with respect to every component of the $\boldsymbol{b}$ vector, at every point (X,Y), to form a Jacobian matrix of dimensions N x n. The transpose of the Jacobian is multiplied by the Jacobian to form a Hessian matrix. The Hessian is used, along with the gradient vector to solve a linear system of equations. The solution, with some modification, is used as the correction to the $\boldsymbol{b}$ vector to form a better fit between the sum of the Gaussians and the image being decomposed. A more detailed description is found in Marquardt's paper [10].

Hence, two major problems with the Marquardt algorithm are that (1) the number of computations within an iteration can consume much time and (2) the size of the matrices can consume much space for image decomposition. For example, in a small 16 x 16 image that may involve as many as 50 Gaussians, the Jacobian matrix would have over 50 thousand entries, making matrix multiplication time-consuming. Furthermore, if the reasonable method of Gaussian elimination is used to solve the linear system of equations, it would involve over eight million calculations. It would clearly be impractical to use the Marquardt algorithm for even larger images involving more Gaussians.

The decomposition method involves (for both Marquardt and Quickstep) selecting an initial Gaussian for the $\boldsymbol{b}$ vector first, and then letting the algorithm iterate, adjusting the $\boldsymbol{b}$ vector to minimize the error between the Gaussian and the image. If the tolerance ε is not met (see Inequality 3), another Gaussian is selected, concatenated to the $\boldsymbol{b}$ vector, and the algorithm runs again, refining all Gaussians in the set further. This continues until the tolerance is finally met. If many Gaussians are added to the set at once, there will probably not be a good initial fit between the Gaussian mass and the image, and it is more difficult for the algorithm to find an optimal fit, due to the high energy of the Gaussian mass. When the Gaussians are allowed to settle one by one, they have lower energy. This method was used in previous work [3, 6, 7] for optimality.

This problem is classified as a large-scale nonlinear least-squares approximation problem, for which the

Marquardt algorithm is not considered suitable for the reasons above. Current large-scale methods [2, 11] focus on approximating the Hessian matrix by other faster methods than matrix multiplication. For our application, however, we still find this unacceptable, as the formation of the Hessian matrix still requires at least $O(n^2)$ time, and the linear system of equations, which takes $O(n^3)$ time, does not benefit. It will be shown in the results section, that even if the formation of the Hessian matrix were instantaneous, the Quickstep method is still faster for practical image problems.

The contribution of this work is to provide a method which reduces the number of computations significantly, yet produces results that are similar in optimality to those produced by the Marquardt algorithm. The Quickstep algorithm is linear in the number of Gaussian parameters on each iteration. Each iteration takes a crude step towards the solution, requiring more iterations. However, each iteration is very fast so that the overall time is reduced substantially. The Quickstep method also has the advantage that it does not require large amounts of space to reach the solution, as the formation of the Jacobian matrix, the formation of the Hessian matrix, and solving the linear system of equations are all eliminated.

3 The Quickstep Algorithm

In practically any nonlinear least-squares algorithm, the initial selection of parameters is an important issue. A recent method for selecting initial parameters, which has proven quite successful, is the selection method used by Childs *et al.* [3]. It has been shown in their work, that for a signal composed of a set of Gaussians, a smaller set of Gaussians can often be found to approximate the signal, using this Gaussian selection method. Therefore, this selection method is also used in this work. To avoid computation inaccuracy, however, the selected standard deviation is no smaller than 0.4 and no larger than 100.0. The initial parameters, when determined, are concatenated to the $\boldsymbol{b}$ vector of Equation 4.

Quickstep makes use of the idea that these Gaussians can be truncated. Since a Gaussian decays so rapidly as the distance from its position increases, it makes sense to eliminate the insignificant computations at a certain distance from its position. The criterion that is used here is not the standard deviation, but the function value of the Gaussian, because this function value should be considerably less than ε (see Inequality 3). Therefore, the effective distance ξ from the position of the Gaussian is defined as

$$\xi = \left\lceil \sqrt{2\sigma^2 \ln \frac{|A|}{\delta}} \right\rceil \qquad (5)$$

where δ is an absolute function value tolerance. This equation is easily derived from the 1D Gaussian function, setting its function value to δ and solving for $\xi = |X - x|$. The ceiling is used here since pixels have integral coordinates. For ease of implementation, computations are confined to a square of dimensions 2ξ instead of a circle of radius ξ. The notation ESA is used to refer to this "effective square area" of the Gaussian. The choice of δ for Equation 5 can be critical to how well this truncation method works. If δ is too small, no gain may be realized from using it. If δ is too large, it will prevent the algorithm from using a significant part of the Gaussian, increasing the number of Gaussians required. A good choice for δ seems to be $\delta = 0.05\varepsilon$, determined experimentally.

In the Quickstep algorithm, the gradient method [4] is used to determine the direction of correction for the $\boldsymbol{b}$ vector. Therefore, in an iteration of the Quickstep algorithm, the gradient vector $\boldsymbol{g}$ is created as

$$\boldsymbol{g}^{[n]} = \sum_{j \in ESA_\ell} \left[P(X,Y)_j - f(X,Y)_j \right] \frac{\partial f(X,Y,b_\ell)_j}{\partial b_\ell},$$

$$\ell = 1,2,\ldots,n. \qquad (6)$$

where f and P are as defined in Equation 2 and Inequality 3.

The choice of scale is a very important problem. Without a properly chosen scale, the number of Gaussians to decompose a problem will increase sharply. The Quickstep method uses the diagonal of the Hessian that would be formed in the Marquardt algorithm to determine a scale, since this provides some important information without increasing the time complexity of an iteration. In Marquardt's original paper, the standard deviations of the parameters are used [10]. However, in Quickstep, when the diagonal Hessian is scaled, it becomes the identity matrix. Hence, the gradient vector is the solution. Since the scale is applied to both the gradient vector and the solution, it can be applied to the gradient vector twice. Hence, in Quickstep, the scale is used only on the gradient vector, and the variances of the parameters are used instead of the standard deviations, which avoids an expensive call to a square root

function. Hence, a scale vector $\boldsymbol{s}$ is created as

$$\boldsymbol{s}^{[n]} = \sum_{j \in ESA_\ell} \left(\frac{\partial f(X,Y,b_\ell)_j}{\partial b_\ell} \right)^2, \quad \ell = 1,2,3,...,n$$

which should be computed at the same time as Equation 6, in order to prevent storing or recalculating the derivatives. The scale is then applied to the gradient vector $\boldsymbol{g}$:

$$\hat{g}_\ell = \frac{g_\ell}{s_\ell} \qquad \ell = 1,2,\ldots,n$$

The refinement correction to the $\boldsymbol{b}$ vector is now computed and stored to a trial vector $\boldsymbol{t}$:

$$t_\ell = b_\ell + \frac{\hat{g}_\ell}{\eta} \qquad \ell = 1,2,\ldots,n \tag{7}$$

where

$$\eta = 1 + \frac{\lambda}{v}$$

where λ is initially 0.01 when the Quickstep algorithm starts up for a set of Gaussians, and v is more or less an arbitrary value, set to 10.0 in this work.

The trial vector $\boldsymbol{t}$ is processed a little more, depending on the type of element. Nothing is done to amplitude parameters. However, position parameters are normalized to equivalent values between 0 and 2π (for the sine function [3]); otherwise, their valid adjustments can be quite large, leading to inaccuracy in computations. In addition, the standard deviation is set to its absolute value (since it may become negative in the algorithm) between 0.3 and 100p, where p is the number of pixels in one dimension of a square image. If the standard deviation adjusts outside these boundaries, it is set to the appropriate boundary value.

The next step is to calculate a sum-of-Gaussians matrix of the same size as the image or image section being decomposed (only the ESA's of the Gaussians need to be used here). This matrix is used as function f to calculate the sum-of-squares error Φ:

$$\Phi_{(k)} = \sum_{j=1}^{N} \left[P(X,Y)_j - f(X,Y)_j \right]^2 \tag{8}$$

where P was defined earlier in Inequality 3, and $\Phi_{(k)}$ denotes the error Φ on the k^{th} iteration of Quickstep. If $\Phi_{(k)} < \Phi_{(k-1)}$, then $\boldsymbol{t}$ becomes the new $\boldsymbol{b}$ vector, the new value λ is set as λ / v, and the Quickstep iteration repeats. If $\Phi_{(k)}$ is not lower, however, the search for a trial vector continues until Φ is lowered (assuming that Φ is not at the minimum error). If the search must continue for a suitable trial vector, the algorithm proceeds to adjust λ as in Marquardt's paper [10].

The test for convergence and other stopping conditions, as used in Marquardt's paper [10], are used as stopping conditions for Quickstep also. However, a stopping condition which needs further elaboration is when the number of iterations has reached a certain limit. Optimality seems to level off when 40 iterations has been reached. Many iterations, in both Quickstep and Marquardt, are idle iterations, in which insignificant decreases to error are made without any benefit in optimality, and putting a limit on the number of iterations removes many of the idle ones.

There is a stipulation in setting such a limit, however. Although the limit removes many idle iterations, it is difficult to predict when idle iterations eventually lead to significant iterations. In such cases, the algorithm "breaks new ground", that is, finds a steep path downhill. If the algorithm "breaks new ground", the count of iterations is restarted at 0, since such a steep path can lead to significant progress. New ground is considered to have been broken when $\Phi_{(k)} < 0.8\Phi_{(k-1)}$, determined experimentally.

The Quickstep algorithm assures convergence if no iteration limit is set, for three reasons: (1) the gradient method is used, (2) a trial vector is not accepted unless it leads to lower error, and (3) if no trial vector is ever accepted, eventually η reaches a number so large that, due to Equation 7, the trial vector approaches the $\boldsymbol{b}$ vector until the convergence test [10] is met.

The Quickstep algorithm uses the gradient method, like most nonlinear least-squares algorithms, but the Newton component [8] is reduced. The Newton component determines the amount of correction to the $\boldsymbol{b}$ vector very well, but it is responsible for the enormous amount of calculations in one iteration. In this work, a way is provided to quickly determine the correction size, which, though crude compared to the Newton correction size, still makes decent progress towards the solution. The result is that Quickstep requires more iterations to reach the solution, but an iteration of Quickstep is so fast that the overall time is dramatically reduced. Using the Quickstep method, the complexity

of each iteration is reduced to $O(N + n\rho)$, where ρ is the average number of pixels in an ESA of a Gaussian.

It may be worth commenting that truncation of Gaussians can also be applied successfully to the Marquardt algorithm to increase the speed. In particular this does much to reduce the time for the matrix multiplication, since only the intersections of the ESAs need to be multiplied. However, the system of linear equations does not benefit from such truncation in the Marquardt algorithm, and therefore, the complexity of an iteration is not reduced as in Quickstep.

When the standard deviation of a Gaussian becomes the imposed minimum 0.3, the Gaussian can be used to approximate a pixel without having any significant effect on other pixels. Thus, a 16 x 16 image section, in the worst case, requires 256 "mutually exclusive" Gaussians for approximation. Though this type of approximation is undesirable, it shows that arbitrarily high frequencies can be approximated discretely. It also shows that noise in images will be included in the approximation. It is possible to use Gaussian decomposition for image smoothing, by eliminating Gaussians with small standard deviations, but this will remove some detail as well.

4 Results

The results in this section are limited to digital images. The use of the algorithm for continuous images would require that the image be first represented by a function, which is the goal of Gaussian decomposition in the first place (see Equation 2 and Inequality 3).

In this section, the performances of Quickstep and Marquardt will be compared. Therefore, the implementation of the Marquardt decomposition algorithm is described next. In this implementation, intrinsic boundaries are used as in Quickstep, for optimality, and the selection of initial parameters is the same as in Quickstep. The same adjustments on trial parameters (see previous section), are used to ensure accurate computations. In the Jacobian matrix multiplication, we make use of the fact that the resulting Hessian is symmetric, and therefore, only roughly half of the entries need to be computed. The system of linear equations was solved using Gaussian elimination with maximal column pivoting. The stopping conditions are the same as for Quickstep, except that the iteration limit is set at 10 for Marquardt and 40 for Quickstep (the reason for this will be explained later). However, if "new ground is broken", the iteration count is reset to 0 (see previous section). The tolerance ε is set at 10.0 for both methods, which usually produces a good SNR rms; most of the points in the Gaussian approximation are usually well under the tolerance when the algorithms complete. The rest of the Marquardt algorithm is implemented as in Marquardt's original paper [10]. All experiments in this section were conducted on a 300 MHz PC.

In an initial experiment involving 30 random 16 x 16 images, Marquardt attained a total of 558 Gaussians for the image sections, requiring a total time of 5150 seconds, while Quickstep attained a total of 563 Gaussians in a total of 168 seconds. Each case usually had a different number of Gaussians for both Marquardt and Quickstep. Quickstep had increased the number of Gaussians in 9 of the cases, but decreased the number of Gaussians in 10 cases.

As mentioned earlier, in comparisons of Marquardt with Quickstep, the iteration limit is set at 10 for Marquardt, while it is set at 40 for Quickstep. The reason for setting an iteration limit in the first place is that it removes idle iterations which produce insignificant decreases in the error, and most of these occur in the later iterations. Since Quickstep requires more iterations to reach the solution, its iteration limit cannot be set to 10 without degradation of optimal performance. This is unfortunate because the lower the iteration limit is set, the faster the algorithm runs. For example, if the iteration limit is set at 10 for Quickstep, it requires a total of only 49 seconds instead of 168 seconds; however, the total number of Gaussians increases substantially to 596, an average increase of 1.1 Gaussians per test case. It is worth mentioning also that if the iteration limit is set at 40 for Marquardt, its total number of Gaussians only decreases by 3 to 555, yet its total time increases to a painful 15932 seconds. For these reasons, we feel that these different iteration limit settings allow for a fair comparison of the algorithms.

Current large-scale methods still try to speed up by approximating a Hessian matrix without doing a matrix multiplication [2, 11]. The Marquardt algorithm can also be sped up by using Gaussian truncation , done in our earlier experiments, which has a large effect on the matrix multiplication time. However, these methods do nothing to speed up an accurate solution to the linear system of equations. The total time spent in solving linear systems of equations in the Marquardt algorithm for the initial experiment was 470 seconds, about 2.8 times higher than the total time for Quickstep.

In a second more extensive experiment, the Lena standard image was divided into 16 x 16 sections and completely decomposed yielding 256 test cases of a variety of image sections, including high and low con-

(a)

(b)

(c)

(d)

Figure 1: (a). The original Lena image. (b). Lena constructed with Gaussians, decomposed with the Marquardt algorithm in 16 x 16 sections. (c). Lena constructed with Gaussians, decomposed with the Quickstep algorithm in 16 x 16 sections. (d). Lena constructed with Gaussians, decomposed with the Quickstep algorithm in 32 x 32 sections (shows a reduced blocking effect).

	Marquart	Quickstep	Quickstep
section size	16 x 16	16 x 16	32 x 32
iteration limit	10	40	20
total Gaussians	4617	4693	4332
maximum Gaussians in a section	61	62	197
total time for decomposition	95448 s	1561 s	3516 s
trial acceptance percentage	44.9%	47.7%	46.5%
average iterations per section	151	611	1234
SNR rms	30.7	29.8	32.3

Table 1: Comparison of decomposition methods.

trast, detailed and smooth. Figure 1a shows the original Lena image, while Figures 1b and 1c show the results for Marquardt and Quickstep, respectively. Table 1 shows how the decomposition methods compare. Although Table 1 shows slightly more Gaussians for Quickstep than Marquardt, Quickstep had a smaller number of Gaussians than Marquardt in 77 sections,

with an average decrease of 1.7 Gaussians, while having a larger number of Gaussians in 89 sections, with an average increase of 2.3 Gaussians. Note, in Table 1,that the average trial acceptance percentage shows that Quickstep has a good ability to come up with trial vectors which reduce the error. Note, however, that Quickstep takes four times as many iterations to reach roughly the same optimal solution as Marquardt (to be expected from the iteration limits). It is apparent that while Quickstep takes steps to reduce the error, the steps it takes are rather crude compared to Marquardt. In spite of this increase in iterations, a striking feature of Quickstep is that it was sixty times faster in decomposing the image, giving rise to the name "Quickstep". For those sections which require few Gaussians, Quickstep and Marquardt had nearly the same speeds, but for those image sections requiring 50 or more Gaussians, Quickstep often achieved a speed 100 times faster than Marquardt, to be expected from the differences in the time complexities. Finally, the quality of results, SNR rms, is similar between the algorithms.

Note that there is a substantial blocking effect in Figures 1b and 1c, occurring mainly in the smooth regions of the image. This effect is largely due to our visual systems, which can perceive slight distortions in smooth regions much more easily than in detailed regions. Figure 1d uses 32 x 32 blocks in the Quickstep method, reducing the blocking effect. The price paid for this reduction, of course, is the increase in time shown in Table 1, in spite of the fact that the iteration limit was set lower at 20. The results in Figure 1d were produced for demonstration purposes only; the Marquardt algorithm was not used for this, since it would be hundreds, if not thousands, of times slower.

In spite of the reduction in the blocking effect in Figure 1d, note that some visual distortions still exist in the smooth areas. One way to improve the image quality would be to lower the tolerance ε (see Inequality 3), which was set at 10 gray scales for the results of Figure 1. This would, however, increase the number of Gaussians. Perhaps a better approach would be to decrease the tolerance for only smooth regions. It also appears that the tolerance can be increased for the detailed regions, providing a balancing effect in the number of Gaussians. This is to be a subject for future research.

Figure 2 shows a 128 x 128 BRDF image [9], which was interesting because of its smoothness and points of brightness. Figure 2a shows the original image while Figure 2b shows the Quickstep decomposition using a tolerance of 10 gray scales with the entire image block size. Note that the Gaussians

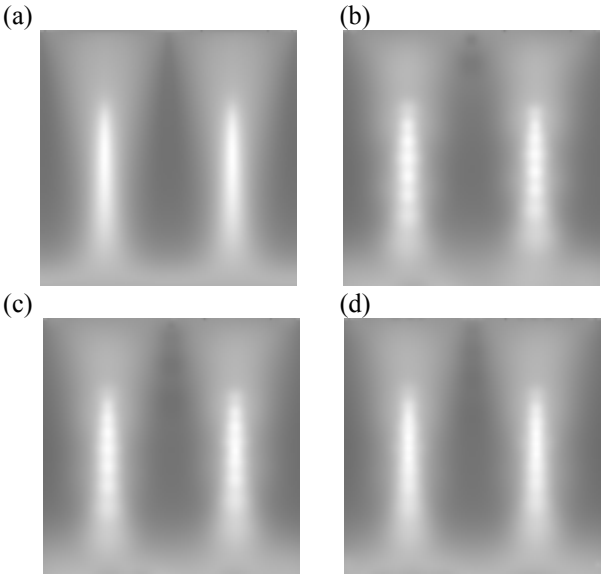

Figure 2: (a) Original BRDF image. (b)-(d). Constructed with Quickstep Gaussians at different tolerances. The solution for part (b) used a tolerance of 10 gray scales for 39 Gaussians, an SNR rms of 62.9, and a time of 325 s. The solution for part (c) used a tolerance of 8 for 46 Gaussians, an SNR rms of 93.2, and a time of 502 s. The solution for part (d) used a tolerance of 5 for 67 Gaussians, an SNR rms of 125.7, and a time of 850 s.

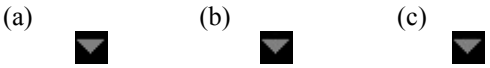

Figure 3: (a) Original solid triangle. (b) Marquardt solution, 26 Gaussians in 279 seconds. (c) Quickstep solution, 22 Gaussians in 4 seconds.

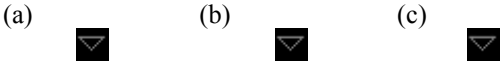

Figure 4: (a) Original line triangle. (b) Marquardt solution, 18 Gaussians in 183 seconds. (c) Quickstep solution, 20 Gaussians in 3 seconds.

Figure 5: (a) Original checkerboard. (b) Marquardt solution, 43 Gaussians in 1194 seconds. (c) Quickstep solution, 42 Gaussians in 20 seconds.

A	x	y	σ
42.274208	12.203599	6.583468	7.200311
43.400757	12.005547	6.820785	7.372244
70.441765	11.396811	8.395815	10.301986
266.132111	7.506141	1.496234	0.570001
-163.423981	11.431159	7.452995	8.088238
318.471039	1.500715	7.505594	0.523438
105.653259	9.611252	7.463912	1.520774
182.185608	7.993632	9.074813	1.588626
141.002945	15.503941	1.484623	1.454900
176.595016	1.502229	15.495743	0.845721
174.587067	15.493904	9.499632	0.893617
313.620148	11.507056	15.504231	0.523847
455.373016	11.532457	5.467762	0.416167
511.010590	13.470234	3.530351	0.400589
381.775269	11.592936	7.508440	0.394871
465.260345	5.475596	11.524047	0.422260
508.425995	3.507012	13.492288	0.423976
441.056396	5.423644	9.497160	0.390403
431.503571	3.502404	7.502584	0.452811
396.376526	13.446114	9.497159	0.442155
329.314117	3.506291	5.498511	0.515609
317.646271	5.494154	3.509284	0.522677
417.074951	7.498525	11.584785	0.391672
418.942352	9.497097	13.497495	0.458514
320.169495	11.502256	13.491319	0.524116
302.156281	13.483729	11.508123	0.537596
315.037018	7.505332	3.510144	0.524875
358.022034	9.490659	5.400075	0.404291
-212.223663	7.749088	7.731223	0.668982
-213.701141	9.200408	9.267549	0.695086
418.416382	1.501172	13.449910	0.432953
381.748230	15.498572	3.603220	0.386795
-58.661896	6.072452	7.875461	0.715204
329.980164	1.496699	5.494276	0.510028
370.436798	13.395384	1.502107	0.390808
358.394714	3.556432	15.498575	0.460700
-63.836678	9.177217	11.012353	0.681923
314.962189	5.487666	1.494551	0.521280
308.281555	9.492308	15.499116	0.533067
288.189209	15.502576	11.565727	0.508811
-134.192871	15.503010	8.184283	0.300000
-53.650463	7.808944	5.910851	0.627661
-60.838818	10.877752	9.168189	0.699867

Table 2: The Gaussians in the Marquardt solution used to form the checkerboard of Figure 5b. A tolerance of 10 gray scales is used, so a summation that is slightly negative should be set to 0 for image reproduction.

A	x	y	σ
-138.774292	1.533950	1.532053	1.135911
-144.186066	15.474975	15.534304	1.752688
153.853256	9.906965	9.721567	15.381313
-226.461365	7.516838	15.506108	0.696704
-169.637299	1.508797	9.492138	0.976500
-127.458023	7.247541	7.248353	1.094320
-164.261551	9.473604	9.484006	1.845971
-258.263092	9.489571	1.491194	0.592394
-233.087234	15.504033	7.516286	0.683021
-280.434479	7.506627	13.478838	0.556137
-276.227295	5.489450	13.484420	0.590671
-265.548615	9.494089	11.634647	0.420271
-268.916412	3.532081	11.522730	0.590058
-266.030731	3.611399	9.485515	0.533130
-284.022095	5.370478	7.518034	0.481777
-318.762756	11.554947	11.554553	0.464486
-368.327332	13.454527	13.452782	0.442399
-261.482300	5.456026	5.456465	0.584261
-345.473083	3.530949	3.530430	0.485007
-271.530792	7.518799	5.366188	0.493460
-263.843109	9.481397	3.522405	0.603679
-269.478210	11.517205	3.518836	0.596757
-277.096405	13.483739	5.491963	0.594079
-301.062653	13.482814	7.507129	0.536386
-251.241989	11.640913	9.494258	0.430138
226.632263	7.685505	9.437407	0.475072
208.890335	9.425779	7.682054	0.502574
-301.843292	3.607900	1.504779	0.449477
-272.315552	15.494983	13.369817	0.420588
-289.918518	1.505110	3.611056	0.456856
-280.464020	11.514410	1.492916	0.556433
-252.680649	13.359729	15.491117	0.425132
-267.772858	5.464641	15.510127	0.573313
-268.120422	15.511187	5.466878	0.578068
-257.250610	1.495238	11.602804	0.522109
110.602249	1.484623	7.847948	0.378529
79.193634	11.295532	7.749356	0.510608
79.568695	7.753181	11.298998	0.527986
55.679623	6.479113	9.261699	0.437128
39.449394	9.199624	6.398310	0.464254
-18.759287	14.871377	10.357703	0.783934
-16.998495	10.361329	14.967628	0.722080

Table 3: The Gaussians in the Quickstep solution used to form the checkerboard of Figure 5c. For image reproduction, Gaussians should be truncated using Equation 5, with δ set to 0.5. A tolerance of 10 gray scales is used, so a summation that is slightly negative should be set to 0.

line up in the bright regions, producing a noticeable rounding effect. In this case, 10 gray scales is probably not an acceptable tolerance. Figures 2c and 2d show the results when tolerances of 8 and 5 are used, respectively. This was also done for demonstration purposes; Marquardt was not used for this because of the enormous amount of time it would take on a 128 x 128 block size.

Finally, Figures 3 through 5 show comparisons between Marquardt and Quickstep on some 16 x 16 artificial images. From these few results, we can conclude that Quickstep will sometimes attain smaller Gaussian solutions for artificial images. Tables 2 and 3 show the Gaussian solutions achieved by Marquardt and Quickstep, respectively, for the checkerboard image in Figure 5. In viewing these Gaussian solutions, one should keep two things in mind: (1) Gaussians with negative amplitudes will partially cancel Gaussians with positive amplitudes in summation, producing special surfaces, and (2) since the Gaussian approximations are used only for the discrete pixels, the Gaussians can do "whatever they want" between the pixels; it is not unusual for thin Gaussians to peak very high between pixels, with only its lower points being used in the discrete approximation. Note that, although the two solutions differ by only one Gaussian, the two Gaussian solutions are quite different. One suspects that there may be many such solutions. In a work by Childs *et al.* [3] it was shown that, for a signal composed of Gaussians, the same Gaussian solution is often not recovered. In fact, quite often, a smaller Gaussian solution is found.

Care should be taken to be taken to follow the directions under Tables 2 and 3, if the reader wishes to reproduce the results.

5 Conclusions

In Gaussian decomposition of images, Quickstep offers a much faster method to achieve optimal Gaussian representations than other methods currently available. The optimal Gaussian decomposition of images, previously considered a problem for supercomputers, is now within the grasp of the PC. This is due to a large reduction in the time complexity of an iteration. Newton's method can achieve very good steps towards the solution. In contrast, Quickstep takes rather crude steps towards the solution, but it takes them so quickly that the overall time is dramatically reduced.

Quickstep is based largely on the gradient method, but incorporates many components of the Marquardt algorithm to avoid the long convergence times characteristic of the gradient method. By using an appropriate scale and trying for larger step sizes, it is possible to reach the solution much faster.

Quickstep not only saves time, it also saves a great deal of memory space. Hessian matrices, in an application like this, can have up to hundreds of thousands of entries. Intermediate matrices are also often formed when solving a linear system of equations.

Quickstep also has the advantage of having less roundoff error for very large problems than Marquardt. One problem in solving large linear systems of equations is in the accumulation of roundoff error, which leads to inaccuracies of the solution and poor convergence. Quickstep, in comparison, can be expected to perform basically the same for even larger problems.

Gaussian representations have promising applications in computer graphics. Although it takes time to find a Gaussian representation of an image, once a Gaussian representation is achieved, the Gaussians can be added together quickly to reconstruct the image. Gaussians can be transmitted one at a time, from the most significant Gaussian to the least significant. The Gaussians that have been transmitted can be added to start constructing the image while the other ones are being transmitted. This allows low-to-high resolution display of images across the Internet, resulting in faster image recognition. Truncated Gaussians can be used to construct images even faster, because they only need to be added to the image in the truncated area. This is an important speedup since many Gaussians that fill in details of an image are small. Furthermore, moving images can be handled more easily, since it is only necessary to adjust the parameters, especially position parameters, of the Gaussians which represent them. Gaussians also make very good features for the coarse structures of an image, aiding in image recognition.

Practitioners are usually most interested in three areas: optimality, time, and quality of results. Regarding optimality, Marquardt may have a slight, though subjective, edge over Quickstep. However, in any event, it is justifiable to classify Quickstep as an optimal algorithm, since it can often achieve smaller Gaussian solutions than Marquardt. In fact, from the results in this section, Quickstep achieves a smaller solution than Marquardt about 30% of the time (Marquardt achieves a smaller solution than Quickstep about 34% of the time). Therefore, for those practitioners where optimality is most important (and time is not), it is recommended to use both Marquardt

and Quickstep, and take the smaller Gaussian result of the two. The Gaussian solution can also be reduced by increasing the tolerance ε (see Inequality 3), to the point where the quality level is just acceptable.

Regarding time, Quickstep is unquestionably faster for large problems. However, one may want to consider the Marquardt algorithm if one has access to a parallel computer. The iterations cannot be put in parallel, and Marquardt takes less iterations to achieve a solution. It is currently not known how a Marquardt iteration will compare to a Quickstep iteration on a parallel machine. If Quickstep practitioners care little about optimality, the time can be further reduced by (1) decreasing the iteration limit and (2) increasing δ (see Equation 5). However, one should use care in adjusting these parameters; eventually, the algorithm will slow because of the sheer increase in the number of Gaussians.

Regarding quality, Marquardt and Quickstep are very close. For those practitioners for which quality is much more important than the number of Gaussians, the tolerance ε can be lowered. However, one should note that it is never necessary to set the tolerance below 0.5. When the tolerance is set at 0.5, the summation of the Gaussians result for a pixel can be rounded to produce the exact integral pixel value. Hence, such a Gaussian representation is lossless. However, one should also note that the number of Gaussians should increase substantially from a tolerance of 1.0 to a tolerance of 0.5, due to quantization effects. Using a tolerance of 0.5, even a very smooth image will have a blocked effect at the pixel level, due to its integral pixel values.

Quickstep is still in its early stages and there are many possibilities for improving the time further. Future research efforts should focus on reducing the time to form a sum-of-Gaussians matrix, as this process accounts for more than half of the time for the current implementation of Quickstep with truncation. This is presumably because the formation of the initial vectors does not need to be repeated when a trial vector fails, but the calculation of the sum-of-Gaussians matrix does repeat to check the error of other trial vectors. Reduction of the number of iterations, possibly by using a dynamic iteration limit, is another possibility for decreasing the time involved.

References

[1] J. Ben-Arie and K. R. Rao. Nonorthogonal signal representation by Gaussians and Gabor functions. In *IEEE Transactions on Circuits and Systems, Part II: Analog and Digital Signal Processing*, Vol. 42, pages 402-13, June, 1995.

[2] A. R. Conn, N. I. M. Gould, and P. L. Toint. A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds. In *SIAM J. Numer. Anal.*, Vol. 28, No. 2, pages 545-72, April, 1991.

[3] J. Childs, C. C. Lu, and J. Potter. Intrinsic boundaries in Gaussian decomposition. In *Proceedings of IASTED*, pages 64-8, 1999.

[4] H. B. Curry. The method of steepest descent for non-linear minimization problems. In *Quarterly of Applied Mathematics*, pages 258-61, 1944.

[5] P. J. S. G. Ferreira. A comment on the approximation of signals by Gaussian functions. In *IEEE Transactions on Circuits and Systems, Part II: Analog and Digital Signal Processing*, Vol. 45, No. 2, pages 250-1, February, 1998.

[6] A. Goshtasby and W. D. O'Neill. Curve fitting by a sum of Gaussians. In *CVGIP: Graphical Models and Image Processing*, Vol. 56, No. 4, pages 281-84, July, 1994.

[7] A. Goshtasby and W. D. O'Neill. Surface fitting to scattered data by a sum of Gaussians. In *Computer Aided Geometric Design*, Vol. 10, pages 143-56, 1993.

[8] H. O. Hartley. The modified Gauss-Newton method for fitting of nonlinear regression functions by least-squares. In *Technometrics*, Vol. 3, No. 2, pages 269-80, 1961.

[9] J. Kautz and M. McCool. Interactive rendering with arbitrary BRDFs using separable decompositions. In *10th Eurographics Workshop on Rendering,* Springer-Verlag, Rendering Techniques '99, pages 247-60, 1999.

[10] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. In *J. Soc. Indust. Appl. Math.*, Vol. 11, No. 2, pages 431-41, 1962.

[11] P. L. Toint. On large scale nonlinear least squares calculations. In *SIAM J. Sci. Stat. Comput.*, Vol. 8, No. 3, pages 416-35, May, 1987.