On Distributed, Probabilistic Algorithms for Computer Graphics

Eugene Fiume
Marc Ouellette

Department of Computer Science
University of Toronto
10 King's College Road
Toronto, Ontario, Canada
M5S 1A4

Abstract

When attempting to solve a multi-variate optimisation problem, it is often a wise strategy to sacrifice a locally-optimal solution in the hope of finding a better global solution. Algorithms that solve optimisation problems in this manner are sometimes called *hill climbing* algorithms. Recently, several new hill-climbing approaches have been proposed and have gained in popularity for solving special classes of optimisation problems. These approaches are probabilistic in nature. We introduce one particular approach called *simulated annealing*, and show how a distributed version of it can be used to solve a basic problem in computer graphics and image processing: colour quantisation.

Résumé

Lorsqu'on essaie de résoudre un problème en fonction de plusieurs variables, il est souvent nécessaire de sacrifier une solution localement optimale, de manière à pouvoir trouver une meilleure solution globale. On classifie parfois ce genre d'algorithmes, appliqué à des problèmes d'optimisation, comme étant de type *hill climbing*. Récemment, plusieurs méthodes d'*hill climbing* ont été introduites pour résoudre certains problèmes d'optimisation. Ces méthodes, dites aléatoires, ont connu un regain de popularité. Nous introduisons ici une approche particulière appelée *simulated annealing*. Une application répartie de cette technique sera utilisée pour résoudre un problème de base en infographie, soit celui de *colour quantisation*.

1 Hill-Climbing Algorithms

Not every problem has an efficient solution. Many problems are provably intractable, or suspected of being so. Consequently, there will always be room for approximate, heuristic algorithms. A class of such algorithms based on probabilistic "hill climbing" have recently been suggested as being useful for solving a variety of optimisation problems [KiGV83]. Probabilistic algorithms have been applied to a number of problems in image processing, VLSI placement, partitioning, routing, and packing. Under the circumstances, it would be worthwhile to consider the possible applications of probabilistic approaches to the solution of some difficult problems in computer graphics. In this paper, we shall motivate the use of probabilistic algorithms in computer graphics, and discuss the application of one such algorithm to a well-known problem in computer graphics, colour quantisation. A probabilistic solution to the "reverse iterated function system" problem has also recently appeared [LeGa88].

One particularly colourful probabilistic approach is called *simulated annealing*, and is based on an appealing analogy from statistical thermodynamics. Imagine that a substance is in a liquid state at a high temperature, and that at some sufficiently-low temperature, the liquid solidifies (i.e., freezes) into a crystal. As the substance in liquid form is slowly cooled, the overall activity of its constituent atoms follows suit. However, the energy level of any given atom may be significantly higher or lower than the mean energy level, and in fact the level of some atoms may even increase as the liquid is cooled. This becomes increasingly improbable as the temperature continues to diminish. If the temperature diminishes very slowly in a sequence of discrete steps, an equilibrium will be achieved for each temperature, and the substance will eventually crystallise at the lowest possible energy level. This annealing process was directly simulated by Metropolis *et al.* quite some time ago [MRRT53].

The metaphor of simulated annealing has attracted many researchers interested in new approaches to solving optimisation problems [Kirk84]. Suppose we have a function $E : \mathbf{R}^n \rightarrow \mathbf{R}$ to minimise. Our goal will be to arrive at a minimum value for E. To to so, we shall attempt to anneal E, and the analogy to physical annealing will be obvious. We begin the process at a fairly-high temperature $T = T_0$, and some initial configuration $x \in \mathbf{R}^n$ for E. We compute a new configuration $x' \in \mathbf{R}^n$ and look at $\Delta E = E(x') - E(x)$. If $\Delta E \leq 0$, we accept x' as the new configuration. Otherwise, we accept x' with probability $P(T, \Delta E)$. The occasional acceptance of x' when the value of E is increased

permits hill climbing to occur. This process iterates at a given temperature T until it is felt that some equilibrium has been reached, at which point, T is decreased. The process then continues for the new value of T, until T falls below a minimum threshold temperature T_1, upon which we output the current configuration, x.

The choice of probability function $P(T, \Delta E)$ is important. It must be such that when T is large with respect to ΔE, P is fairly close to 1, and that as T decreases, P decreases. Intuitively this corresponds to the physical situation in that when the temperature is high, a particle x could take on, with non-negligible probability, a very large range of energy values. As the temperature decreases, this range statistically decreases. In keeping with the analogy, P is chosen such that at early stages of the annealing process, it is quite likely that a new configuration x' will be chosen, even if it increases the value of E; as the process continues, such x' will be rejected increasingly often. A particularly good choice for P is

$$P(T, \Delta E) = e^{-\Delta E/T}.$$

Many such functions are possible, but perhaps surprisingly, this function appears to be superior in the sense that it meets the above end conditions, and in that it allows the system to converge more quickly than all other currently-known functions. Moreover, it happens to be the precise analogue of the probability function describing the expected behaviour of particles in real physical systems [Donn87; NaSS85].

Another important issue is the choice of *annealing schedule*, that is, the rate at which to decrease the temperature, and the amount of time to spend at each discrete temperature. The choice of this schedule tends to be somewhat *ad hoc*. What is important about the schedule, however, is to ensure that the temperature does not decrease too quickly, for this would tend to trap the simulation in a local minimum.

It has been observed by Donnett that simulated annealing and its variants are useful for solving a particular class of problems, namely those in which there exist many acceptable nearly-optimal solutions, and that the difference between nearly- and absolutely-optimal solutions is negligible. Even with these restrictions, the class of applicable problems appears to be quite large, and several problems in computer graphics fall into this category.

To reduce a problem to an instance of an annealing problem, we must come up with a function E that, when minimised, gives the solution for our original problem. Often the choice of E is quite obvious, but the choice of annealing schedule and initial and final temperatures is not.

A general annealing schema for minimising a function E is as follows. Let T_0 be the starting temperature, and let T_1 be the final temperature. The algorithm schema is given in Figure 1. A useful reformulation of the schema is Donnett's, in which the program transformations given in Figure 2 are made [Donn87]. The values k_0, k_1, k_2 are constants, chosen essentially by trial and error, depending on the specific problem to be solved. This provides a useful set of parameters with which to customise the annealing schedule to the problem at hand. We also found that an "$\vee$" in place of the "$\wedge$" in "**not** at equilibrium" to be useful.

```
generate an initial configuration x
T ← T_0
accepts ← rejects ← 0
while (not frozen)
    while (not at equilibrium)
        generate new configuration x'
        ΔE = E(x') − E(x)
        if (ΔE ≤ 0)
            x ← x'
            accepts ← accepts + 1
        else
            choose random ∈ [0,1]
            if (random < e^{−ΔE/T})
                x ← x'
                accepts ← accepts + 1
            else
                rejects ← rejects + 1
    accepts ← rejects ← 0
    update T
```

Figure 1: *Simulated Annealing Schema.*

not frozen $\equiv T \geq T_1$
not at equilibrium $\equiv$ *attempts* $< k_0$ $\wedge$ *rejects* $< k_1$
update $T \equiv T \leftarrow T \times k_2$

Figure 2: *Refinement of annealing schema.*

One final observation worth making at this juncture is that simulated annealing is not likely to be a particularly fast way of finding an optimum. However, it lends itself quite nicely to some kinds of parallel implementations. As a concrete example of the approach we shall show how the problem of colour quantisation can be cast into an instance of an annealing problem, and we shall describe a parallel implementation of it.

2 Colour Quantisation

The *colour quantisation* problem can be defined as follows. We are given an input image of dimensions $m \times n$ pixels such that each pixel $P_{ij} \in [0, K], P_{ij} \in \mathbf{N}$ encodes an (r, g, b) colour value, and we are given a value $k \ll K$. In a typical case, $K = 2^{24}$, encoding 8-bit values for each of red, green, and blue, and $k = 256$. The output of the algorithm is a k-element set $C \subset [0, K]$ and an $m \times n$ pixel image such that each output pixel $Q_{ij} \in C$. The set C, which is often called a *colour map*, is chosen such that the function

$$E = \sum_i \sum_j \epsilon(P_{ij}, Q_{ij})$$

is minimised, where ϵ is an error metric on a colour space. As is often noted (but rarely implemented), the best choice for ϵ would be one that takes perceptual considerations into account. Most often, it is simply the euclidean distance metric on the red, green, and blue components of the colour. That is, if P_{ij} denotes colour (r_0, g_0, b_0) and Q_{ij} denotes (r_1, g_1, b_1), then

$$\epsilon(P_{ij}, Q_{ij}) = \sqrt{(r_0 - r_1)^2 + (g_0 - g_1)^2 + (b_0 - b_1)^2}.$$

All algorithms using this metric are therefore heuristic to some degree. The square root operation is of course not essential in practice. In effect, the colour map induces k volumes or *partitions* of the colour space such that each volume i encloses the points in the colour space that are closest, with respect to ϵ, to a specific colour map entry $c_i \in C$.

Observe that the solution to the above problem is, in general, very difficult. Several effective heuristic algorithms have been devised to give an approximate solution to this problem [Heck82; GePu88; WaWP88]. These algorithms typically approximate a local minimum in the solution space, subject to several reasonable criteria. It is natural to attempt to find the best minimum possible, within reason. One line of attack would be to solve something resembling a large least-squares problem in order to find the appropriate set C, although formulation of the problem in this manner is not easy. This is moreover likely to be unbearably slow. On the other hand, the problem is clearly a good candidate for simulated annealing, and it would be instructive to consider solving it in this manner. In fact, it is sufficiently straightforward to derive a good parallel implementation of it from the outset. We now present an outline of our approach.

Clearly, the objective function to be minimised by annealing is the cost function E given above in terms of an error metric ϵ. Suppose we have a set of $N \geq 1$ processors with which to work and with an $n \times m$ input image I as described above. We also allow an initial colour map C to be given as input if desired. If not, a default initial colour map is chosen (the default being either the 3-3-2 or popularity partitions [Heck82]). We first determine the set or *histogram* H of distinct colours in I, and give each processor i a distinct partition $H_i \subset H$ consisting of $h = |H|/N$ colours. The processors co-operate by evaluating E for their respective image partitions with respect to the current colour map. The algorithm is depicted in Figure 3. The manner in which parallelism is exploited was deliberately kept as simple as possible, for reasons discussed below.

The data structures we employed were extremely basic, since our goal was simply to determine whether or not a probabilistic approach would work at all. A "production" version of the implementation should make use of more advanced search structures such as those described in [Heck82; GePu88; WaWP88]. Each histogram entry (i.e., unique colour in the original image) records the original colour a, its multiplicity in the image, the index of the colour map entry denoting colour b that is currently closest to a, and

```
basic set up, get input image, create H
set up or get initial colour map C
partition H into N parts H_1, ..., H_N
distribute C, H_i to client i
for each colour c ∈ H_i each client i
    finds closest c' ∈ C

T ← T_0
accepts ← rejects ← 0
while (not frozen)
    while (not at equilibrium)
        determine c ∈ C to be replaced
        choose a new colour c'
        C' = C − {c} ∪ {c'}
        client i computes ΔE_i(C', C)
        cost change ΔE ← Σ_i ΔE_i(C', C)
        if (ΔE ≤ 0)
            C ← C' for all clients
            each client i updates H_i
            accepts ← accepts + 1
        else
            choose random ∈ [0,1]
            if (random < e^{-ΔE/T})
                C ← C' for all clients
                each client i updates H_i
                accepts ← accepts + 1
            else
                rejects ← rejects + 1
    accepts ← rejects ← 0
    update T
```

Figure 3: *Distributed colour quantisation algorithm.*

the error between a and b. Each colour map entry contains the current colour it represents, the number of pixels, h, in H which are currently being mapped to it, and the cumulative error, e. The statement

determine $c \in C$ to be replaced

in the algorithm above amounts to choosing the colour map entry that has maximal average cumulative error, $a = \frac{e}{h}$. If h is zero, then a is deemed arbitrarily large (as would be expected).

The statement

choose a new colour c'

involves "jittering" each of the red, green, and blue components of c by a signed, uniformly-distributed random variable.

3 Implementation

3.1 Environment and Parameters

As with the other approaches to colour quantisation, the colours in the input image were prequantised to five bits of precision. This allowed us to focus on perceptually noticeable changes when choosing new colour map entries. The jitter range for each colour component was chosen to be $\pm[0,7]$. Since each colour component has a value of [0,31] after prequantisation, a jittered colour may deviate from the original colour by about 25%.

We have successfully implemented the above algorithm in the C programming language on a network of Sun 3 workstations running UNIX.[1] The communication mechanism among processors was via sockets.

The values we used for most of the parameters are debatable, and we relied on trial and error to set them. This, we feel, is one shortcoming of the approach, in that it is difficult to devise scientific ways of determining good values for these parameters. Nevertheless, the default values that we shall now discuss worked fairly well. Of course, a user can override default values at run time.

The initial temperature is the most difficult value to determine. Our criterion was that, since we wished to allow arbitrary colour maps to be given as input, the initial temperature should reflect some level of "excitation" of the system with respect to the input map. For this reason, we set the initial temperature to the average cost per red, green, or blue component of the colour map entry. That is, if the initial colour map is $C = \{c_1, c_2, ..., c_k\}$, then the initial global cost is

$$g = \sum_i \sum_j \epsilon(P_{ij}, \text{closest}(P_{ij}, C)),$$

where closest(P, C) is the closest colour map entry in C to colour P. The initial starting temperature is then set to

$$T_0 = g/k/3.$$

Other parameters were set as follows.

final temperature: $T_1 = 1.0$.
equilibrium acceptance threshold: $k_0 = 20$.
equilibrium rejection threshold: $k_1 = 50$.
temperature scale : $k_2 = 0.8$.

For other applications, the ratio k_0/k_1 is usually smaller, but we found that the system tended to cool too quickly if k_0 was too small, especially if the "V" formulation of equilibrium was used–see above).

3.2 Results

The results are in several ways surprising, and may in time lead to a deeper understanding of the nature of the problem (read: the authors aren't sure why the algorithm works as well as it does). The graphs given below are with respect to one particularly tricky image to quantise (Figure 8). It is a good test case because it contains a large number of distinct colours after prequantisation (on the order of 6,000), it contains several transparent objects which act as coloured filters for the refracted texture, it contains several regions of nearly constant shade, and it contains our old friend and primate, the mandrill, as a subcase. We tested the algorithm out on many images, with similar results.

Many of the papers discussing probabilistic methods mention that the running time of the programs is quite high. Our results did not entirely support this. We found that the approach converged to a decent local minimum quite quickly. Convergence occurred regardless of the starting colour map. The program was tested using four different input colour maps:

1. the popularity partition: the 256 most commonly-occurring colours in the image.
2. the colours arising from retaining only the 3 most significant bits of the red and green components, and the 2 most significant bits of the blue component. This provides a fairly uniform scattering of colour map entries throughout the colour space.
3. a completely red colour map (ignoring green and blue).
4. the output of the "variance-based" colour quantisation algorithm of Wan *et al.* [WaWP88] (courtesy of Craig Kolb, Yale University via USENET).

Figures 4-7 depict the current global cost as a function of the number of iterations (i.e., potentially new choices for the colour map). Figure 4 uses initial colour map 1, Figure 5 uses colour map 2, and Figures 6-7 use colour map 4. The test image was of resolution 256 × 256 for Figures

[1] Sun is a trademark of Sun Microsystems. UNIX is a trademark of AT&T Bell Laboratories.

4-6, and of resolution 900 × 900 in Figure 7. The result of using colourmap 3 is, apart from scale, almost identical to colour map 1, and is omitted. The global cost (and therefore initial starting temperature) is much different for each of the test cases, the initial cost being highest for the 3-3-2 partition and lowest for the variance-based partition. Also note that the scale is different for each figure. The cost is essentially halved in Figures 4 and 5, but the improvement in Figures 6-7 is a hard-earned but noticeable 6-9%. Moreover, in Figures 4 and 5, the greatest improvement in cost occurs in the first quarter of the program's execution. Also notice the remarkable amount of hill-climbing evident in the early stages of Figures 6-7. As expected, the frequency of hill-climbing quickly falls off as the system cools. The amount of hill-climbing is less pronounced in the other figures. This is probably because the system cooled too quickly to allow substantial hill climbing; the results, however, were quite acceptable at this stage. We found that the popularity partition was often a better initial map than the minimum-variance partition, in that the system converged more quickly.

It is very difficult to devise ways of visualising the differences between algorithms and among images, particularly since a great deal of noise is added in the photographic and reproduction stages of publication. For this reason, we have devised a less-direct visualisation scheme. Figure 8 is a black and white representation of a test input image. The remaining figures represent quantisation error. The error displayed for each pixel constitutes the scaled, euclidean distance between the (r, g, b) component of the original and mapped images. Thus the whiter the image appears, the greater the error. Figures 9 and 10 illustrate the error under the popularity mapping and the 3-3-2 mapping, respectively. Figures 11 and 12 depict the error after using the colour-variance algorithm of Wan *et al.* and after annealing, respectively. The output of the annealing algorithm was similar with each initial partition. In each case, we ran the program until its overall error was significantly less than that produced by the colour-variance algorithm. For a 900 × 900 image, this required about one hour of CPU time on a single Sun 3/280 when started with the popularity partition; when started with the output of the colour-variance algorithm, about one hour was required for a 5% change, and three hours were needed for a 9% change. This suggests, of course, that the colour-variance algorithm is quite good. The differences between the colour-variance pictures and those produced by simulated annealing were small but noticeable (on a monitor, but not in a photograph). About two hours of CPU time were required when started with the red colour map or 3-3-2 partition. These CPU figures are given only as ballpark figures, since our (linear) data structures are quite primitive, and these timing results could certainly be improved.

It is interesting to compare Figures 11 and 12. Although from these figures it is not possible to tell which has done a better job of quantisation (to do so is impossible from the colour photographs we produced, but as stated earlier, it is certainly possible when staring at a monitor), the aims of the algorithms are decidedly different. The colour-variance algorithm is willing to admit some fairly large discrepancies, whereas our algorithm focussed on minimising global error by minimising average error per partition. Consequently, there are fewer completely dark areas in Figure 12, but there are also fewer very bright areas (note particularly the errors in the mandrills' eyes).

The results of parallelism are not particularly interesting. The algorithm demonstrates an almost perfect speed up as the number of processors increase. After about 4 processors, communication overhead overwhelms the additional processing gains. This suggests that either communication bandwidths be increased, or that a solution based on tightly-coupled parallel processors be found. We are currently examining the latter possibility.

4 Discussion

Several phenomona have shown up in our results which bear discussion. First, the choice of initial colour map strongly affects the rate of convergence. To some extent this is to be expected. We were surprised, however, by how much more slowly the system converged when given the 3-3-2 partition over the popularity partition. (The slow rate of convergence when given an already good colour map was of course entirely unsurprising.) A possible explanation is that a uniform scattering of colour map entries tends to isolate many of them (i.e., the cardinality of many partitions is quite small). Random jitter, in this case, would not be sufficiently "directed" toward a single solution (or several closely-related ones).

Second, the initial rate of convergence for the popularity and 3-3-2 partitions is, in absolute terms, quite high, the comment above notwithstanding. Presumably, the explanation is that there is much room for improvement with these initial partitions.

Third, our naive use of parallelism was not as fruitful as expected. The main difficulty with this global optimisation problem (and one that might indeed be generally true) is that it is resistant to true divide-and-conquer approaches. We were not able develop a parallel algorithm which synthesises a global solution from several smaller subproblems. We were thus forced to cast the algorithm into a simple client/server scheme. An ongoing problem of interest is to develop more creative approaches to the parallel decomposition of this problem.

We have demonstrated the use of a distributed, probabilistic algorithm for a nontrivial problem in computer graphics. We now hope to further refine our approach to this problem, and to attack several other problems in computer graphics, including colour gamut mapping, and global illumination, using probabilistic techniques.

Acknowledgements

The comments by the reviewers of this paper are much appreciated. We wish to acknowledge the financial support

of NSERC in the form of a operating grant and University Research Fellowship for the first author, and a postgraduate scholarship for the second author. We are also grateful to the Province of Ontario's Information Technology Research Centre for additional funding.

References

[Donn87] Donnett, J.G., "Simulated annealing and code partitioning for distributed multimicroprocessors", Technical Report 87-194, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada.

[GePu88] Gervautz, M., and W. Purgathofer, "A simple method for colour quantization: octree quantization" *Proceedings of Computer Graphics International '88* (May 1988), Springer-Verlag, 219-231.

[Heck82] Heckbert, P., "Colour image quantization for frame buffer display", *ACM SIGGRAPH '82 Proceedings*, also published as *ACM Computer Graphics 16,* 3 (July 1982), 297-307.

[KiGV83] Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing", *Science 220,*, 13 (May 1983), 671-680.

[Kirk84] Kirkpatrick, S., "Optimization by simulated annealing: quantitative studies", *Journal of Statistical Physics 34,* 5/6 (March 1984), 975-986.

[LeGa88] Levy-Vehel, J., and A. Gagalowicz, "Fractal approximation of 2-D objects", *Proceedings of EUROGRAPHICS '88* (Sept. 1988), Elsevier Science Publishers (North-Holland, Amsterdam), 297-311.

[MRRT53] Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equations of state calculations by fast computing machines", *Journal of Chemical Physics 21* (1953), 1087-1091.

[NaSS85] Nahar, S., S. Sahni, and E. Shragowitz, "Experiments with simulated annealing", *Proceedings of the 22nd Design Automation Conference* (1985), 748-752.

[WaWP88] Wan, S.J., K.M. Wong, and P. Prusinkiewicz, "An algorithm for multidimensional data clustering", *ACM Transactions on Mathematical Software 14,* 2, (June 1988), 153-162.

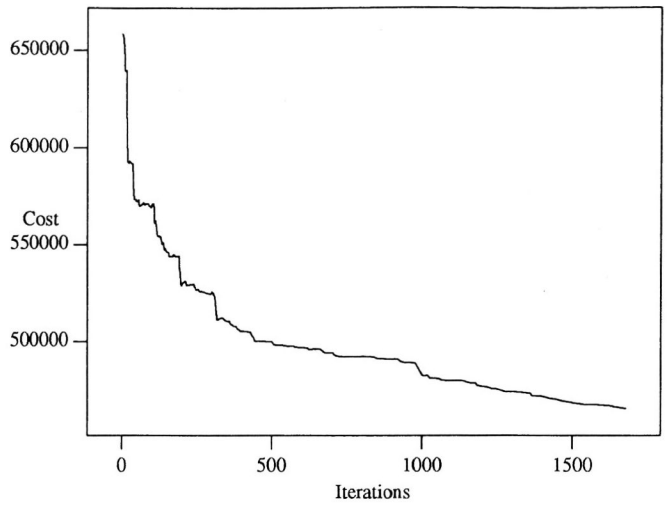

Figure 4: *Cost plot.* Initial map: popularity.

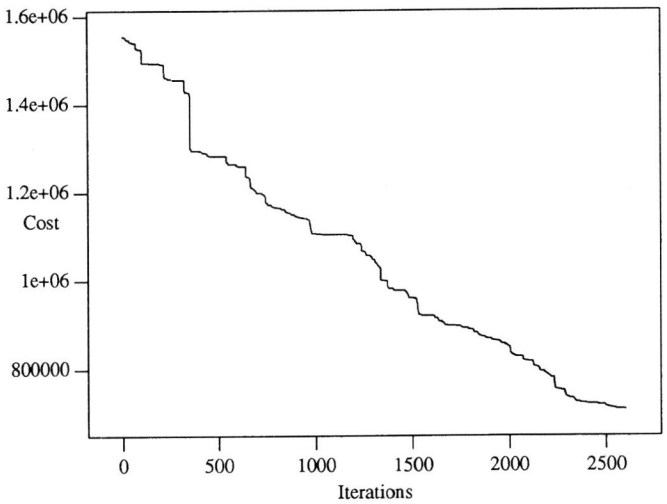

Figure 5: *Cost plot.* Initial map: 3-3-2.

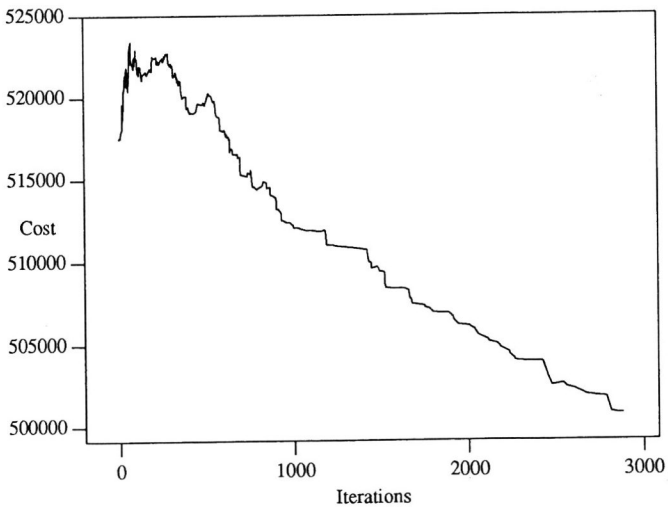

Figure 6: *Cost plot.* Initial map: output of variance algorithm.

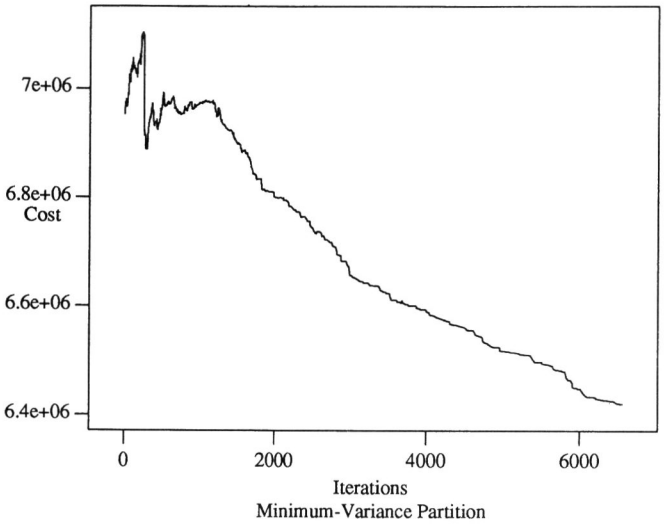

Figure 7: *Cost plot.* Initial map: output of variance algorithm. 900 × 900 image.

Figure 8: *Black and white version of test image.*

Figure 9: *Quantisation error of popularity partition.*

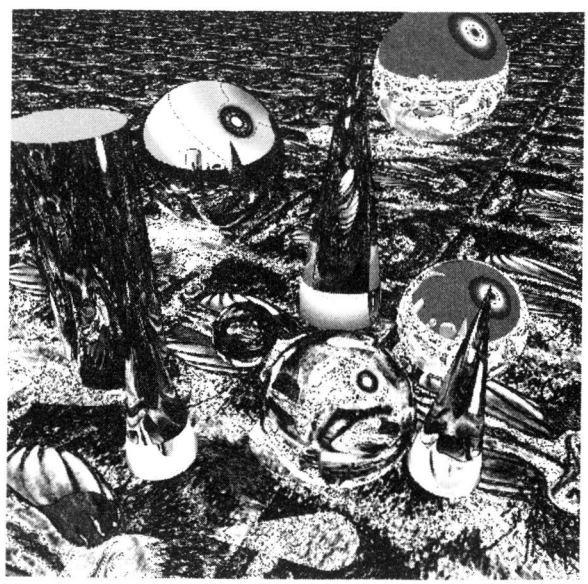

Figure 10: *Quantisation error of 3-3-2 partition.*

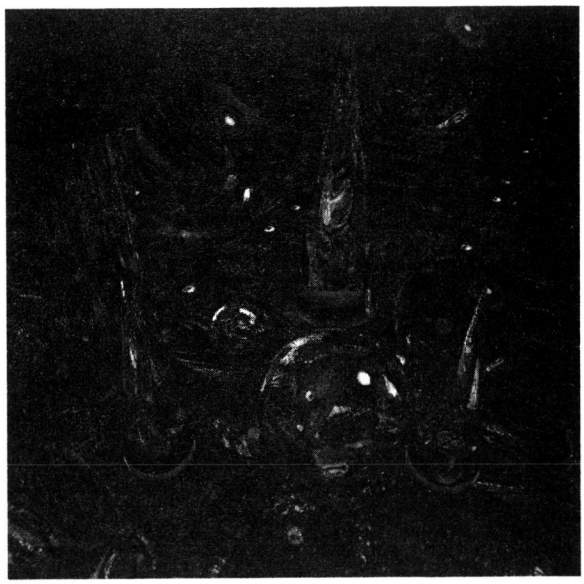

Figure 11: *Quantisation error made by variance algorithm.*

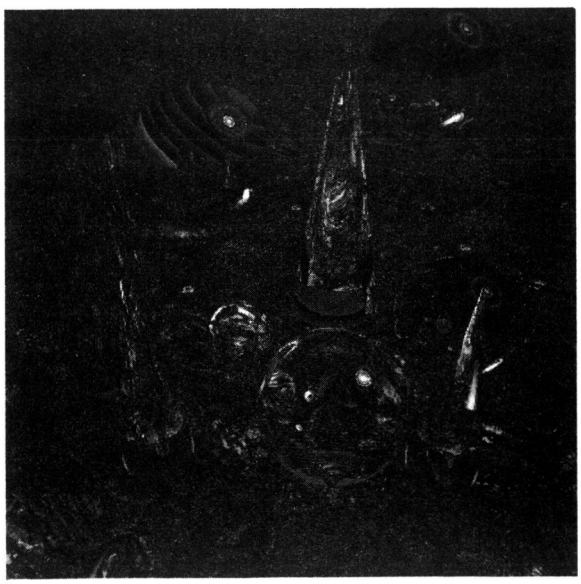

Figure 12: *Quantisation error made by annealing algorithm.*