1996

A Peak Finding Strategy for Multi- Dimensional Scalar Fields with Applications to X-ray Crystallography

Calin R. Costian

Ioana M. Boier Martin

Dan C. Marinescu

Mikhail J. Atallah
Purdue University, mja@cs.purdue.edu

Report Number:
96-038

A PEAK FINDING STRATEGY FOR MULTI-DIMENSIONAL SCALAR FIELDS WITH APPLICATIONS TO X-RAY CRYSTALLOGRAPHY

Calin R. Costian
Ioana M. Boier Martin
Dan C. Marinescu
Mikhail J. Atallah

Department of Computer Science
Purdue University
West Lafayette. IN 47907

CSD TR-96-038
June 1996

A Peak Finding Strategy for Multi-dimensional Scalar Fields with Applications to X-ray Crystallography

Calin R. Costian, Ioana M. Boier Martin, Dan C. Marinescu, Mikhail J. Atallah
Computer Sciences Department
Purdue University, West Lafayette, IN 47907

Abstract

This paper describes a *dimension-reduction (DR)* strategy for locating peaks in multi-dimensional scalar fields. Algorithms based on the DR-strategy are used in molecular structure determination by means of X-ray crystallography.

1 Overview

Scientific and simulation experiments often produce multi-dimensional scalar values. The collection of all these values constitutes a *scalar field* over the volume sampled. A problem encountered in analyzing such data is to define and locate *peaks*, that is, regions with high values of the scalar property surrounded by comparatively lower regions that can be considered background. A peak is characterized by its *zone of influence* (the region of the space that contains all the points composing the peak) and a *summit* (one of the points with the highest scalar value in the peak). The output of a peak locating procedure can give either the precise zones of influence of each peak as sets of points, or a sphere that comprises the peak, centered at the geometrical center of the peak.

This information can then be used as input to a visualization system which may display the peaks in various representations. Two problems related to the determination of the structure of biomacromolecules using X-ray crystallography are described here. The first is to identify reflections on X-ray diffraction images and the second is to locate ridges in 3-D electron density maps used in the structure refinement process.

A biomacromolecule consists of a large number of atoms with fixed positions. Information about the 3D atomic structure of such molecules can be gathered by means of Electron Microscopy, Nuclear Magnetic Resonance, and X-ray crystallography. X-ray crystallography is the method of choice for the study of large macromolecules like viruses because the wavelength of the radiation is of the same order as the inter-atomic distances and high resolution X-ray diffraction images can be used to locate the individual atoms.

The electrons of the atoms in a crystal scatter the X-rays producing a diffraction pattern which can be recorded on photographic emulsion or by CCD detectors. A crystal is a periodic

arrangement of atoms, ions and molecules and therefore, the electron density within such a crystal is also periodic and it can be expressed as the sum of a series of sinusoidal waves. A structure is known when the amplitudes and phases of the sinusoidal waves that make up the electron density function are all known. The amplitudes of the waves are proportional to the square root of the intensities of the recorded diffraction pattern (also known as optical densities). Such diffraction patterns usually contain a fair amount of noise and to identify true diffraction spots from the noisy background, one must be able to locate regions of high optical density in these patterns. The list of peaks together is used to create a list of reflections, including their intensities, the error in estimating them, and the coordinates of each reflection in the reciprocal space (Miller indices).

The phases cannot be directly measured from the X-ray diffraction pattern and instead must be determined using other techniques. After all the amplitudes and phases are known, the electron density function can be computed by means of Fourier synthesis. Regions of high electron density correspond to positions of atoms or groups of atoms within the structure. A 3-D version of the peak-location algorithm can be used to identify such regions in a high resolution electron density map.

The problem of locating peaks in a scalar field requires more than a purely algorithmic approach as the zone of influence of a peak and the notion of a peak itself are not easily quantifiable and are largely dependent on the type of data. Some regions may be considered to be peaks for some applications and may appear to be background for others. The attribute that best characterizes a peak is not its height relative to the surrounding background, but rather, its shape. Thus we have incorporated heuristic techniques in our peak finding approaches, allowing the user to fine tune the search parameters according to the type of image frame that is being explored.

A number of strategies for locating the peaks and determining the zone of influence of each peak are possible. For instance, Kabsch [1] considers a box of fixed dimensions that is centered successively around each pixel of the frame, trying to identify the cases when the box is centered on a peak based on a "mask" that quantifies the approximate shape of a generic peak. This approach can be time-consuming and has the disadvantage of requiring that most of the peaks have a certain approximate size and shape that needs to be known beforehand. Kim [4] divides the peak location process into two phases: a rapid, coarse search that is performed by skipping entire lines and columns of the image frame, followed by a fine step search around the located maxima. This approach seems to be faster, but also yields poorer results, as many peaks can be missed in the process.

Another possibility [6] is to sort all the points in the data set in decreasing order of their scalar values and then to proceed by inspecting the point with the highest value and its neighborhood. After selecting a contiguous area around that point such that all the points in that region satisfy a certain criterion, e.g. their values are above a certain *threshold*, we mark it as a peak, eliminate all the points in the peak set from the sorted list, and restart the procedure for the next unvisited highest point.

We could impose additional conditions, e.g. on the diameter of a peak, or required that the total number of points belonging to a peak should be within a given range, and discard peaks that do not satisfy these conditions. We stop when all the remaining points in the sorted list are below the threshold and consider them part of the background. The

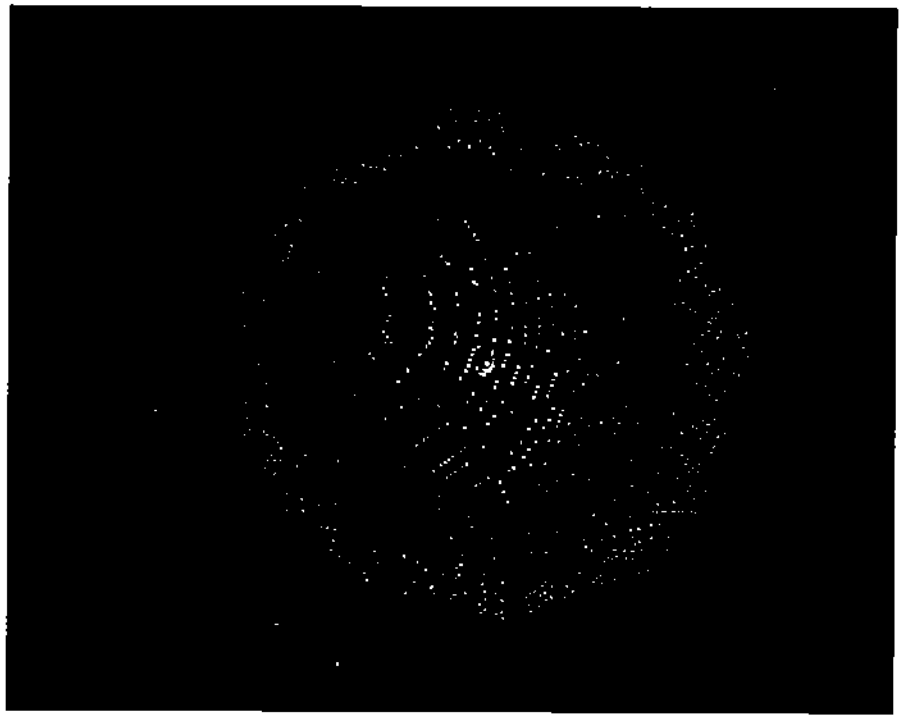


Figure 1: 2-D map (Fuji frame) with a large overall variation of the background. Black corresponds to the lowest optical density values and white to the highest.

determination of this threshold value is complicated by the fact that although the background does not vary greatly *locally*, it can vary greatly across the entire image. As a consequence, the above mentioned threshold depends on the location where it needs to be evaluated. For example, Figure 1 shows an entire frame where the various gray levels correspond to different levels of the scalar values (black corresponding to the lowest values and white to the highest) and we can see the significant increase in the background as we go from the periphery of the image towards the center.

One possible approach is to determine first the background at each point and then proceed with the search (such an algorithm for background evaluation is presented in [3]). The problem is that the whole process can be rather time- and space-consuming.

A better solution is to divide the entire image in regions where the background is almost constant, and apply the procedure in each of these regions separately starting with sorting only the pixels within a region, and ending with locating the peaks in that region. Once this partition is done, it is relatively easy to determine a value for the threshold for each region. For instance in every region, the background accounts for most of the points. The average of all scalar values in a given region is slightly larger than the background since the peak areas are significantly higher than the background, but they have relatively few points. This average can be used as the threshold for the whole region, eventually corrected by a fixed *threshold adjustment.*

Another issue is how to search "around" a peak. Two well-known searching strategies are commonly used: depth-first and breadth-first (see Appendix B). The efficiency of the computation can be affected by the possibility of "going in the wrong direction". The

feature that distinguishes a peak from a background region is the shape, the "sharpness" that characterizes how abruptly the scalar values decrease from the top of the peak towards its boundaries. While the problem of quantifying the sharpness of a peak is not difficult in the unidimensional case, its difficulty can increase substantially in higher dimensions.

In the next section we present an algorithm that reduces the peak location problem to simpler subproblems not by dividing it into regions of the same dimension, but by reducing the dimension of the problem itself, and we discuss the advantages of this approach.

2 Locating Reflections in Optical Density Frames

In this section we present the Dimension Reduction algorithm for locating peaks (also called *reflections* in crystallography) on 2-D optical density frames, whose typical dimensions are approximately 2000 by 2000 pixels. We discuss algorithms for finding unidimensional peaks, correlating them into 2-D peaks and some implementation details.

The idea of the algorithm is to split the 2-D map into rows, find the peak areas in each row (each such area will be represented by a segment) and then combine the segments lying on adjacent rows to form contiguous 2-D peak areas. This approach reduces a 2-D peak search to the problem of unidimensional peaks (segments) that can be located more easily and with greater accuracy. The algorithm scales up well for 3-D peak finding problems. The pseudocode for the relevant routines is presented in Appendix A.

The following subsections present the strategies for finding unidimensional peaks, correlating them into 2-D peaks, tuning the parameters that monitor the process of accepting or rejecting peaks (according to their size, shape, etc.) and some implementation details.

2.1 The Unidimensional Problem

The strategy is to start with the point on the current row having the maximum scalar value and visit the points at its right and left, determining whether that point is part of a unidimensional peak (segment), and if so, what are the left and right boundaries of that segment. Afterwards, find the highest point among all unvisited points on the row and restart the procedure, until the whole row has been visited. The segments returned are non-overlapping.

To define the set of points belonging to a certain peak we can go in one direction (say, to the right) until the current scalar value falls below a threshold, and that would be the rightmost limit of the peak. This approach could accept as a peak a segment that looks more like an almost flat background (see Figure 2a). Figure 2b shows a real peak whose area is poorly determined (a lot of background is included) due to the way the threshold is set. Finally, in Figure 2c shows a low peak that was ignored because the threshold was set above it.

Hence, when defining a peak we must take into account not only its "height" but also its shape. We introduce a parameter to characterize the slope of the peak called sdp (*sharp decrease percentage*), defined as the the difference between the scalar values of two adjacent points.

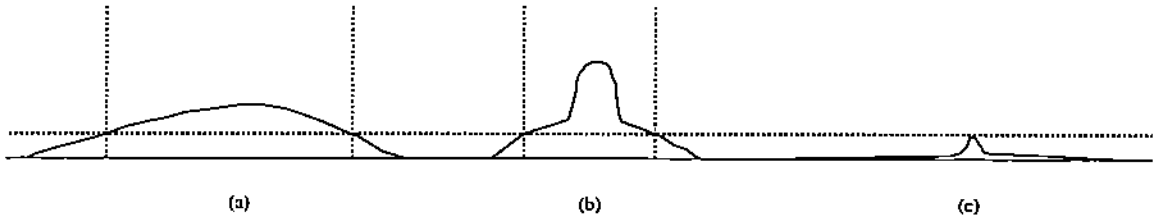


Figure 2: Examples of poor peak identification. The dotted line corresponds to the threshold. In (a) we see a flat background identified as a peak, (b) shows a peak whose zone of influence includes a lot of background, and in (c) we can see a perfectly legitimate peak that is missed because the background surrounding it is much lower than the threshold.

In other words, if P_1, P_2 are two adjacent points (P_1 being at the left of P_2) of scalar values `od1`, `od2` respectively and `od1 > od2`, then the decrease from P_1 to P_2 is considered sharp if and only if:

$$\texttt{od1 - od2} > \texttt{od1} * \frac{\texttt{sdp}}{100}$$

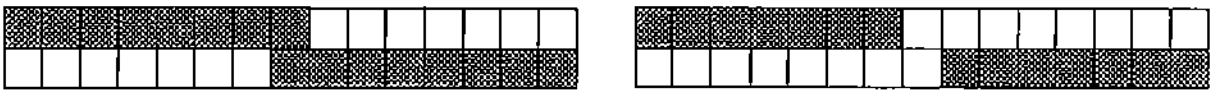
We can now define a better algorithm for exploring the right side (and similarly the left side) of a local maximum: we go to the right starting from the maximum and we record the moments of sharp decrease according to the above condition. We stop when we either hit an already visited point or the optical density starts to increase, as the peak area cannot continue beyond such a point. Then, to eliminate the trailing background, we go back to the last position where we had a sharp decrease, and set that point to be the rightmost boundary of the peak. If we didn't have a sharp decrease at all, then that area is part of the background and the starting maximum cannot be part of a peak (even if the left side looks like a peak, we cannot consider a peak a region that looks like a peak on the left side and like background on the right side) so we discard it.

If we had a sharp decrease at the right (and thus set the rightmost boundary of the peak), we proceed in a similar way at the left of the current maximum. If we don't find any sharp decrease there, we discard the whole area we just visited and continue with the next unvisited maximum on the current line. Otherwise, we have found a leftmost boundary of the segment (which will be the leftmost sharp decrease starting from the current maximum) and we have thus defined a peak segment that we record in our `current_row_segms` array.

The algorithm described above does indeed locate unidimensional peaks based solely on their shape, and thus presents the risk of accepting peaks that are very low and small, and which should be part of the background. To avoid such small variations of the background to be considered peaks, we introduce a second parameter called `sdc` (*sharp decrease cutoff*) that adds a new condition for considering a decrease to be sharp: in the notation used above, this condition can be written as

$$\texttt{od1 - od2} > \texttt{sdc + min\_od}$$

where `min_od` is the minimum scalar value on the current row. In other words, the decrease can be considered sharp only if it is greater than `sdc` units above the "local" minimum (of the current row).



(a) (b)

Figure 3: Two criteria for considering segments correlated. In (a) we see two segments that have one common point, while (b) shows two segments that are off by one point, but that can be still considered correlated if the `peak_offset` parameter is set to 1.

Since we might also want to reject 2-D peaks that are too small or too large, we introduce two more parameters called `maxps` and `minps` (maximum and minimum peak size respectively). The parameter `maxps` can be also used in the unidimensional peak finding phase. If we find a segment larger than `maxps` we can discard it right away and reduce the number of segments. On the other hand, `minps` can only be used after the whole 2-D peak area has been determined, since small segments can be part of a larger two-dimensional peak. The pseudocode for this algorithm is presented in Figure 10.

Finally we sort the segments in increasing order of their endpoints (recall that there are no overlapping segments), since this will be useful in the second phase of correlating the adjacent rows to form the 2-D peaks.

In summary, there are four parameters to control the determination of 1-D peaks: `sdp`, `sdc`, `maxps`, and `minps` and by adjusting them one can adapt the algorithm to different types of data.

2.2 Correlating Adjacent Rows

The procedure presented in Figure 11 takes two adjacent rows and correlates segments identified by the algorithm described in the previous section.

Two segments on adjacent rows are said to be correlated if they have at least one common point (see Figure 3a). We could tighten this condition by requiring them to have two or more points in common, but in practice this can result in many missed peaks. On the other hand, we can relax the condition by allowing the two segments to be off by a small number of points, given by a predefined parameter called `peak_offset`. For example, if `peak_offset` is 1, the segments can be off by 1 and they are still correlated (like in Figure 3b).

Each 2-D peak will be seen as a group of correlated segments. We group the correlated segments in the following way: a group is started by a single segment S that has not been correlated with any segment from the previous row. This segment will be the "head" of the group that will ultimately represent a 2-D peak. On the next row, if we find a segment S' that is correlated with S, we add it to S's group, and S' becomes the "tail" (the last segment) of the group. Then on the next group, if we find a segment S" that is correlated with S', we add it to the group in which S' lies (which is S's group) and S" becomes the tail. This procedure continues until there are no more segments that can be added to this group (i.e.

no segment on the current row can be correlated with the tail of the group). At that moment the group is defined and we add it to our list of 2-D peaks.

The pseudocode for this procedure is presented in Figure 11. The parameters of the procedure are: the current row, the arrays containing the segments found on the current row and on the previous row (`current_segms` and `prev_segms` respectively), and the number of elements in each of these arrays (`nc` and `nprev` respectively). Since `Find1DPeaks()` always returns arrays of segments on the current row that are ordered, we can speed up the correlation process by having two pointers (a segment called csegm on the current row and a segment psegm on the previous row) that advance in a coupled way and reveal the pairs of correlated segments on the two rows.

2.3 Implementation and Results on 2-D Scalar Fields

In this section we present the results obtained with the dimension-reduction algorithm on two-dimensional maps as part of the ODG graphics package [5].

Some criteria to compare the algorithms are the ability to locate peaks correctly (recognizing the real peaks and eliminating "false peaks"), the resolution (how well the algorithm can distinguish between two peaks that are very close to each other) and the speed.

Table 1 shows the comparison between the three algorithms for three areas of different sizes (small, medium and large). The depth-first and breadth-first algorithms produce identical results as far as the number and shape of peaks located (since regardless of the traversal method, the contiguous area around a point that is above a given threshold is unique), and also their execution times vary slightly. Depth-first seems to be a little faster for small and medium sized areas, while breadth-first seems to be faster for large areas where peaks are to be located. This is due to the fact that in the current implementation, the recursion taking place in depth-first incurs the function calling overhead that is not present in the queue manipulation of the breadth-first search routine.

The dimension-reduction is by far the fastest. Moreover, even though in some cases the number of peaks determined can be larger in the case of depth/breadth-first search than for the dimension-reduction algorithm, many of these extra peaks are actually "false peaks" (portions of background that happen to be higher than the background around them). Also, some of the peaks around the center of the frame were lost in the case of the depth/breadth-first search due to the significant variation of the background in that area that lowered the threshold and induced a dramatic but artificial increase of the dimensions of the peaks determined (over the maximum limit imposed on the size of all peaks). In all cases shown below, the constraints on the sizes of the peaks being located have been set to the same values: minimum size = 3 and maximum size = 12. Also, in these particular cases we have found the following settings to produce good results.

For the depth/breadth-first algorithms, the threshold adjustment can be set[1] to 50 and the search box size can be set[2] to 30. For the dimension-reduction algorithm, the slope cutoff

[1]This value is added to the average of the peaks in a search box to obtain a better value for the threshold.

[2]If the search box is too small then the peaks get easily fragmented, if it is too large then sorting the points in the box becomes time consuming.

Searching Method	Number of Points	Number of Peaks Found	Execution Time
Depth-First	30420	181	3.1 secs
Breadth-First	30420	181	2.6 secs
Dimension-Reduction	30420	192	2.5 secs
Depth-First	572000	1784	40.2 secs
Breadth-First	572000	1784	40.8 secs
Dimension-Reduction	572000	2166	7.1 secs
Depth-First	5230000	10686	450.2 secs
Breadth-First	5230000	10686	419.4 secs
Dimension-Reduction	5230000	9925	87.9 secs

Table 1: A comparison of the DR algorithm with the DF and BF search algorithms, on three areas of a FUJI frame having different sizes: small (about 3×10^4), medium (5×10^5) and large (5×10^6). In the first two cases, the DR method locates more peaks than the other two algorithms, while for the large area, even though the DF and BF algorithms report more peaks that have been located, many of them turn out to be "false peaks", portions of the background that happen to be higher than the background around them. DR is faster than the DF/BF methods in all three cases.

can be set[3] to 50, while the slope percentage can be set[4] to 12.

Regarding the resolution of the DR algorithm, for the films we experimented with, setting the constant `peak_offset` to 1 (see section 2.2) seems to give the best results (it allows the program to distinguish between close peaks and at the same time ensures that "thin", elongated peaks are not missed). Also, in the depth/breadth-first search algorithms, increasing the size of the search box has a dramatic effect in increasing the execution time: for example, doubling the box size in the cases presented below would increase the execution time more than five times.

It is not difficult to see that for a portion of an image frame with nr rows and nc columns, the time complexity of the DF/BF algorithms is $\mathcal{O}(nr\, nc \log np)$ where np is the number of points in a block (box), while for the DR algorithm it is $\mathcal{O}(nr\, nc^2)$, the nc^2 coming from the repeated search for the maximum on the current row within Find1DPeaks().

The reason why the DR algorithm turns out to be still faster than the DF/BF methods is that the *expected* time complexity of Find1DPeaks() is not quadratic, but rather linear ($\mathcal{O}(nc)$) due to the fact that most of the time, the number of unidimensional peaks found on a row is small compared to nc, the total number of points on that row. Thus, the expected time complexity for the DR algorithm is $\mathcal{O}(nr\, nc)$, which is linear in the total number of points.

Figures 4 and 5 show the peaks located using the depth-first search and dimension-

[3]The slope cutoff represents the minimum amount in the decrease of a unidimensional peak necessary for it to be considered a sharp decrease.

[4]The slope percentage is the minimum percentage in the decrease of a unidimensional peak necessary for it to be considered a sharp decrease.

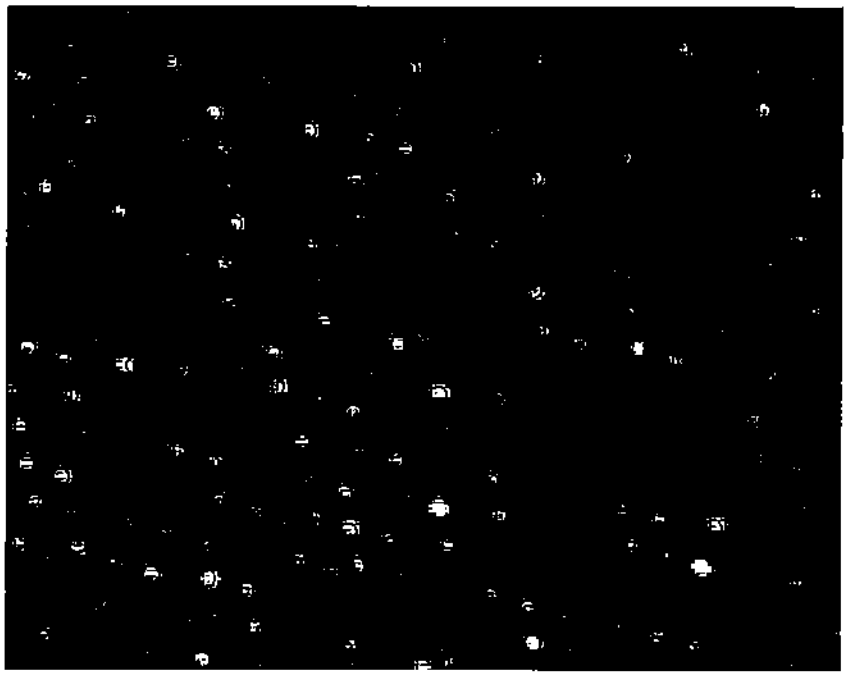


Figure 4: Peaks located with the depth-first search algorithm. The located peaks have been marked by small circles that delimit the zone of influence of each peak. Note that some of the peaks are located with poor accuracy.

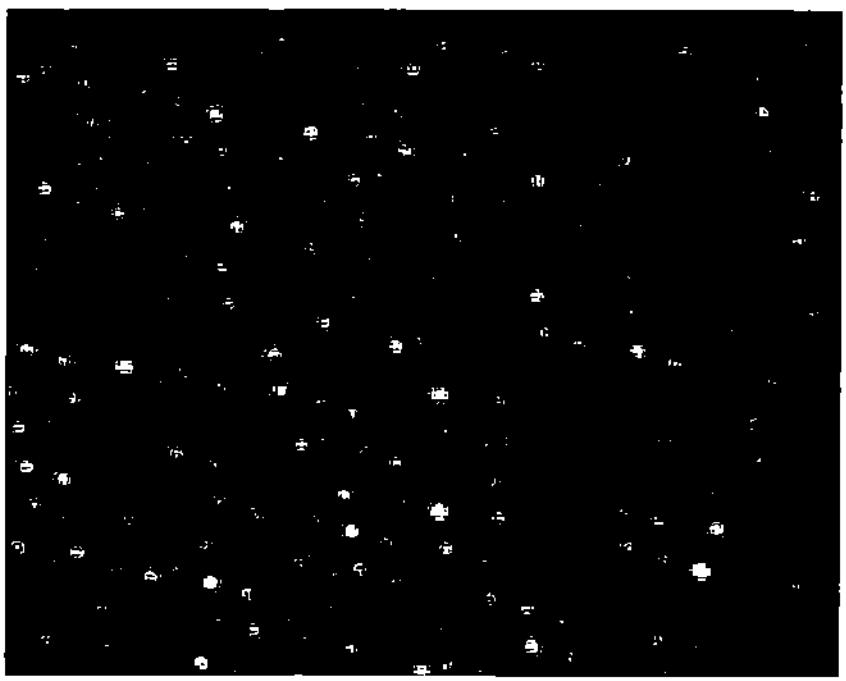


Figure 5: Peaks located with the dimension-reduction algorithm. Note the improvement over the DF algorithm in the accuracy of defining the zone of influence of each peak.

reduction algorithms respectively, on the first study case presented in Table 1 (corresponding to a small sized area of a Fuji image frame). The located peaks have been marked by small circles that delimit the zone of influence of each peak. From the two figures we can see not only that the DR method located more peaks, but also that the zone of influence of each peak is more accurately delimited by the DR method compared to the DF search.

3 Location of Peaks in 3-D Electron Density Maps

This section examines the extension of the 2-D peak finding algorithm to three dimensions for finding regions of high electron density on crystallographic 3-D maps of macromolecules. The basic idea is to split the 3-D map into planes, find the peak areas in each plane using the `Find2DPeaks()` routine and then combine the planes lying in adjacent planes to form contiguous 3-D peak areas. The pseudocode for the main routine for finding 3-D peaks as well as for the routine correlating two consecutive planes is sketched in Figure 12.

Figures 6 and 7 show two representations of the peaks located in the same 3-D electron density map of a Ross-River virus [8]. The volume examined has $55 \times 55 \times 55$ points, and the settings of the parameters of the peak search are:

```
Slope Cutoff = 10
Slope Percentage = 7
Minimum Peak Size = 5
Maximum Peak Size = 50
Minimum 2-D Overlap = 5
```

The execution time was 0.769 seconds. The slope cutoff and slope percentage have the same significance as in the unidimensional DR algorithm, the minimum and maximum peak size give the acceptable range of the diameter of a peak, and the minimum 2-D overlap specifies the minimum number of points that two 2-D peaks on consecutive planes need to have in common to be considered correlated (part of the same 3-D peak).

Figure 6 is a continuous scale 2-D representation of a Z-section at the top of the volume, while Figure 7 is a 3-D isosurface representation of the whole volume. Both images have been obtained using the Tonitza visualization package [9].

Acknowledgements

This work was partially supprted by the National Science Foundation grants CCR-9119388, CCR-9202807, BIR-9301210 and MCB-9527131, and by a grant from Intel Corporation and from the California Institute of Technology. We would also like to thank to Michael Rossmann and Robert Lynch for the valuable suggestions and the help given us in finalizing this paper.

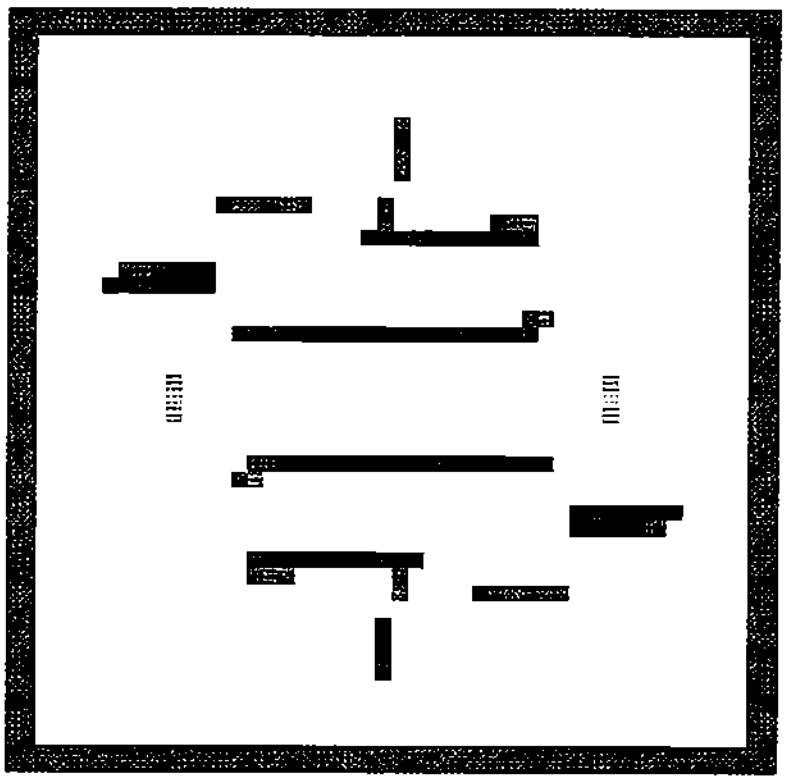


Figure 6: Continuous scale 2-D section of the peaks located in a 3-D volume of electron density.

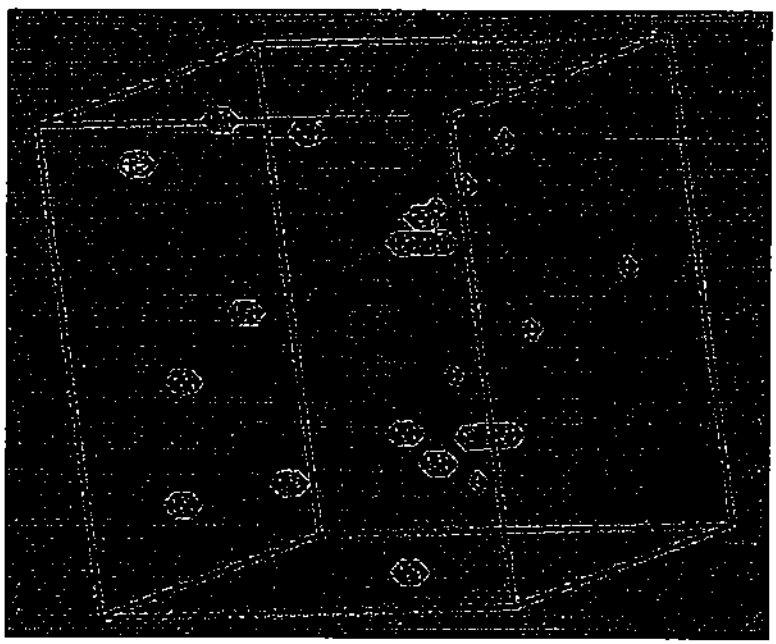


Figure 7: 3-D isosurfaces of the peaks located in a 3-D volume of electron density.

References

References

[1] Kabsch, W., A Pattern-Recognition Procedure for Scanning Oscillation Films, *Journal of Applied Crystallography*, vol. 10, pp. 426-429, (1977).

[2] Rossmann, M.G., Processing Oscillation Diffraction Data for Very Large Unit Cells with an Automatic Convolution Technique and Profile Fitting, *Journal of Applied Crystallography*, vol.12, pp. 225-238, (1978).

[3] Kabsch, W., Evaluation of Single-Crystal X-ray Diffraction Data from a Position-Sensitive Detector, *Journal of Applied Crystallography*, vol. 21, pp. 916-924, (1988).

[4] Kim, S., Auto-Indexing Oscillation Photographs, *Journal of Applied Crystallography*, vol. 22, pp. 53-60, (1989).

[5] Costian, C.R., The Optical Density Graphics Package Web Page at http://www.cs.purdue.edu/research/sb/fp/ODG/odg.html

[6] Rossmann, M.G., Determining the Intensity of Bragg Reflections from Oscillation Photographs, *Methods in Enzymology*, vol. 114, pp. 237-280, Academic Press, (1985).

[7] Watson, D.F., Contouring – A Guide to the Analysis and Display of Spatial Data, Pergamon Press, New York, (1992).

[8] Cheng, R.C. & al., Nucleocapsid and Glycoprotein Organization in an Enveloped Virus, *Cell*, vol. 80, pp. 621-630, (1995).

[9] Boier Martin, I.M, Marinescu, D.C, Graphics for Macromolecular Crystallography and Electron Microscopy, submitted for publication, (1995).

Appendix A

In this appendix we present the pseudocode for the DR algorithm. We start with the main routine for finding 2-D peaks and then show the routines for finding 1-D and also correlating two adjacent rows. At the end we show the pseudocode for the DR algorithm in the 3-D case.

Note that to save space we keep track of the segments (peaks) that we found on the current row in an array of segments `current_row_segms` and we keep the segments we found on the previous row in an array called `previous_row_segms`. As we move to the next row, these arrays get updated accordingly. Also, the variables `n_current_segms` and `n_previous_segms` keep track of how many elements are currently in these two arrays.

```
procedure Find2DPeaks( image frame )
  global list of 2-D peaks := initially empty;
  n_previous_segms = 0;   /* there was no previous row explored initially */
  for  row := 1  to  NROWS  do

    /* find the segments and their number in the current row */

    n_current_segms := Find1DPeaks( row, current_row_segms );

    /* correlate them with the segments from the previous row */

    correlate_with_prev_row( row, current_row_segms, n_current_segms,
      previous_row_segms, n_previous_segms );

    switch the two arrays, so that now previous_row_segms will contain
      what current_row_segms contained, and current_row_segms can be again
      initialized with the values from the next row;
  endfor
  display global list of 2-D peaks;
endproc
```

Figure 8: Pseudocode for the Dimension-Reduction 2-D peaks finding algorithm. The procedure returns a linked list with all the 2-D peaks found, where each 2-D peak is in turn a linked list of segments (1-D peaks) composing its zone of influence, and each segment is a structure containing information about the row on which it is placed, its endpoints, etc. (see the procedure Find1DPeaks() below).

```
integer function Find1DPeaks( row, current_row_segms )
  nsegms := 0;
  min_od = the minimum scalar value on the current row;
  while (there are unvisited points on the current row) do
    P := one of the points of maximum value on the current row;
    mark P as visited;
    if (P was visited or is next to a visited area or is at the
     endpoints of the row) then
      continue;   /* go to the next point */
    endif

    if (P is lower than sdc + min_od) then
      break;   /* exit loop, as the whole rest of the line will be lower */
    endif

    visit the right side of P by recording every sharp decrease and
      stopping when the scalar values start to increase or when we
      hit a point that was already visited;
    if (there was no sharp decrease) then
      continue;   /* discard current peak */
    else
      record the right end of the segment as the point where the
        scalar values last decreased sharply;
    endif

    visit the left side of P by recording every sharp decrease and
      stopping when the scalar values start to increase or when we
      hit a point that was already visited;
    if (there was no sharp decrease) then
      continue;   /* discard current segment */
    else
      record the left end of the segment as the point where the
        scalar values last decreased sharply;
    endif

    if (the size of the segment is larger than maxps) then
      continue;   /* discard current segment */
    else
      register the segment in current_row_segms and increment nsegms;
    endif
  endwhile  /* the whole row has been visited */

  sort segments in current_row_segms in increasing order of their position;
  return nsegms;
endproc
```

Figure 9: Find1DPeaks() fills in the array current_row_segms with the 1-D peaks (segments) found on the current row and returns the number of segments found. Each segment is a structure containing information about the row on which it is placed, the coordinates of its endpoints, and the coordinate of its point of maximum.

```
procedure correlate_with_prev_row( row, current_segms, nc, prev_segms, nprev )
  if (nprev > 0) then   /* there have been segments on the previous row */
    csegm := current_segms[1];  /* current segment on the current row */
    psegm := prev_segms[1];     /* current segment on the previous row */
    i := 1;
    j := 1;
    while (i <= nc and j <= nprev) do
      if (csegm is entirely at the left of psegm) then
        i := i + 1;
        if (i > nc) then  /* current row is done: no more correlation */
          break;  /* exit the while loop */
        else
          csegm := current_segms[i];  /* advance on the current row */
        endif
      else if (csegm is entirely at the right of psegm) then
        j := j + 1;
        if (j > nprev) then   /* finished previous row */
          break;  /* exit the while loop */
        else
          psegm := prev_segms[j];  /* advance on the previous row */
        endif
      else  /* psegm and csegm are correlated */
        add csegm to the group to which psegm belongs;
        mark csegm that it has been correlated with another segment;
        mark psegm also that it has been correlated with another segment;
        i := i + 1;
        j := j + 1;
        if (i > nc or j > nprev) then
          break;  /* exit the while loop */
        endif
        csegm := current_segms[i];  /* advance on both rows */
        psegm := prev_segms[j];
      endif
    endwhile
  endif

  now all the segments that are left unmarked in current_segms are the ones who
    haven't been correlated with any segment from the previous row; so mark
    them as the heads of new groups of segments to be formed later on;
  also, the segments that were not marked in prev_segms were not correlated
    with any segment from the current row, so they are the last segments
    in their respective groups; take their groups and mark them as finished
    (add them to the global list of 2-D peak regions);

endproc
```

Figure 10: Pseudocode for correlating two consecutive rows in a 2-D scalar field. The segments located with Find1DPeaks() are correlated. A segment on the current row correlated with a segment on the previous row is placed in the same linked list, indicating that they both belong to the same 2-D peak, i.e. are part of its zone of influence.

```
procedure Find3DPeaks( array_in, nx, ny, nz, peak_list )
  initialize the peak_list to the empty list;
  initialize a list called previous_2d_peaks to the empty list;
  for  plane := 1  to  nz  do
    /* find the current_2d_peaks list for the current plane */
    current_2d_peaks := find_2d_peaks( array_in, nx, ny, nz, plane );
    /* correlate the two consecutive planes and add the identified 3-D peaks
       to the peak_list */
    correlate_with_prev_plane( array_in, nx, ny, nz, current_2d_peaks,
      previous_2d_peaks, plane );
    switch the two lists, so that now current_2d_peaks becomes
      previous_2d_peaks, and previous_2d_peaks can be reused
  endfor
endproc


procedure correlate_with_prev_plane( array_in, nx, ny, nz, current_2d_peaks,
                                     previous_2d_peaks, plane );
  /* current_2d_peaks contains all 2-D peaks on the current plane, while
     previous_2d_peaks contains all the 2-D peaks on the previous plane */

  for each peak2d in the previous_2d_peaks list do
    find the peak in the current_2d_peaks list who has the largest number of
      points in common with peak2d;
    if (the number of common points is smaller than a preset parameter) then
      continue;   /* go to endfor */
    else   /* the two 2-D peaks are correlated */
      make a link between them so that now they are part of the same 3-D peak;
    endif
  endfor
  for each peak on the previous_2d_peaks list that has not been matched do
    make it the tail (last 2-D component) of the 3-D peak in which it lies;
    register that 3-D peak;  /* compute its diameter, add it to the list */
  endfor                     /* of 3-D peaks, etc. */
  for each peak on the current_2d_peaks list that has not been matched do
    make it the head of a new 3-D peak;
  endfor
endproc
```

Figure 11: Pseudocode for locating peaks in 3-D scalar fields using the DR approach. A 2-D peak is a linked list of 1-D peaks, a 3-D peak is a linked list of 2-D peaks and the routine Find3DPeaks() returns a linked list with all 3-D peaks found.

Appendix B

We outline the depth-first and breadth-first search algorithms for peak finding. In the 2-D case we have the four possible search directions in the following order: North, East, South, West. For the 3-D case we add two more directions: Above and Below.

The current point in the depth-first search starts as the highest one from the sorted list of all the points in the block. The peak set (region) initially contains only this point. For the current point we apply the following search procedure: we try to go in the North and if that neighboring point has not been visited and is above the threshold, we add it to our peak set and it becomes the current point to which we apply again this same search procedure. If that point was below the threshold, we try to go East and if that point has not yet been visited and is above the threshold, we add it to our peak set and it becomes the current point to which we apply again the above search procedure from the beginning. If this one was also visited or below the threshold, we try going South and then West, and if none of the points is eligible to become the next current point, we go back to the previous current point and try a new direction if there is any left (if not, go back to the previous current point, and so on). The search for determining the current peak stops when there are no more directions left to take from the initial point. At this moment, the next unvisited highest point is picked up from the sorted list. If its value is above the threshold, restart the search procedure to find a new peak region, otherwise stop and go to the next block. The depth-first search algorithm is illustrated by the pseudocode in Figure 13.

The breadth-first search starts also with the current point being the highest one from the sorted list with all the points in the block. The peak set is initially empty. We also have a FIFO queue that initially contains the current point. We always take the current point to be the first element in the queue. We remove it from the queue and add it to the peak set. We visit the current point by exploring its 8 direct neighbors by going around it (say, clockwise) and adding to the queue the points above the threshold that have not been previously visited. Then the current point is set to be the first element in the queue (which is at this moment removed from the queue and added to the peak set), and the search procedure is repeated. The algorithm stops when the queue is empty, in which case we mark the peak set that we have found and pick the next unvisited highest point to restart the whole algorithm for finding a new peak region. If this point has a value that is below the threshold, we move on to the next block. The breadth-first search algorithm is illustrated by the pseudocode in Figure 14.

Note that the procedures `DepthFirstPeakLocation()` and `BreadthFirstPeakLocation()` are almost identical, except for the procedures used for the actual search.

Care must be taken not to discard or incorrectly split peaks that happen to lie on the frontiers of two or more adjacent blocks, by defining the blocks in such a way that they overlap, and eliminating the peaks that were counted more than once due to this overlap.

```
procedure DepthFirstPeakLocation( image frame )
  partition entire image frame into blocks;
  list of peak regions := initially empty;
  while (there is any block left) do
    threshold := average of all scalar values in the block;
    make a sorted list with all the points in the block above the threshold
      in decreasing order of their values;
    while (the first unvisited element P in the list is above the threshold) do
      peak_set := the empty set;
      visit_point_depth_first( P, threshold, peak_set );
      add the current peak_set to the list of all peak regions;
    endwhile
  endwhile
  display list of peak regions;
endproc


procedure visit_point_depth_first( P, threshold, peak_set )
  dir[1] := North;
  dir[2] := East;
  dir[3] := South;
  dir[4] := West;
  mark P as visited;    /* we know that when this routine is called, P is */
  add it to the peak_set;   /* always not visited and above the threshold */
  for  i := 1  to  4  do
    P1 := neighbor of P in direction dir[i];
    if (P1 not visited and P1 above threshold) then
      visit_point_depth_first( P1, threshold, peak_set );
    else
      mark P1 as visited;
    endif
  endfor
endproc
```

Figure 12: The Depth-First Peak Location Algorithm. The information about each 2-D peak is stored as an array holding the coordinates of its composing points. A linked list of all 2-D peaks found is returned.

```
procedure BreadthFirstPeakLocation( image frame )
  partition entire image frame into blocks;
  list of peak regions := initially empty;
  while (there is any block left) do
    threshold := average of all scalar values in the block;
    make a sorted list with all the points in the block above the threshold
      in decreasing order of their values;
    while (the first unvisited element P in the list is above the threshold) do
      peak_set := the empty set;
      visit_point_breadth_first( P, threshold, peak_set );
      add the current peak_set to the list of all peak regions;
    endwhile
  endwhile
  display list of peak regions;
endproc


procedure visit_point_breadth_first( P, threshold, peak_set )
  dir[1] := North;
  dir[2] := East;
  dir[3] := South;
  dir[4] := West;
  queue := initially contains only the "root" point P;
  while (queue is not empty) do
    P := first element in the queue;
    remove P from the queue;
    if (P was visited) then
      continue;
    endif
    mark P as visited;
    if (P below threshold) then
      continue;
    endif
    add P to the peak_set;
    for  i := 1  to  4  do
      P1 := neighbor of P in direction dir[i];
      if (P1 not visited and P1 above threshold) then
        add P1 to the queue;
      else
        mark P1 as visited;
      endif
    endfor
  endwhile
endproc
```

Figure 13: The Breadth-First Peak Location Algorithm. The data structures are identical to the ones in the Depth-First Search method.