

# Discrete Object Detection and Motion Registration Based on a Data Management Approach

Hans Hinterberger

Institute of Scientific Computing, ETH Zürich  
CH-8092 Zürich, Switzerland  
e-mail: hinterbe@inf.ethz.ch

Bettina Bauer-Messmer

Institute of Atmospheric Science, ETH Zürich  
CH-8093 Zürich, Switzerland

## Abstract

*When scientific data sets can be interpreted visually they are typically managed as pictures and consequently stored as large collections of bitmaps. Valuable information contained in images is often not exploited, however, simply because the data is not processed further. Common reasons for this are that access to information in image collections is notoriously difficult and that interesting applications often depend on supplementary data with incompatible formats.*

*If such data sets are treated as higher-dimensional point data instead of byte streams and managed with a suitable multidimensional file structure, then it is possible to transform "fuzzy" objects into  $n$ -dimensional solids. Several benefits result: content based access becomes possible, the potential for data compression without loss of relevant information exists and additional information can readily be incorporated simply by increasing the file structure's dimensionality.*

*This paper describes how this approach has been successfully applied to detect and track storm cells in weather radar and various satellite images. The key is to parametrize the data for efficient access based on several different image attributes.*

## 1 Introduction

With scientific data, heterogeneity comes in many flavors. Data about storms in our atmosphere, for example, originate from radar stations, satellites, ground measurements, sounding equipment, aircraft measurements, GIS terrain representations, and hail damage reports. Each of these sources generates data that is evaluated individually, often in real time. In combination, this type of data could provide novel and interesting information and many atmospheric phenomena are therefore investigated by joining archives with data from different observations. The resulting data management task is anything but easy, however, because radar data is incompatible with infrared satellite data which in turn is incompatible even with satellite data from the visible channel. For a solution to be practical, it must be based on a data repre-

sentation that can accommodate the different types of information under one hat.

The need for more flexible *representational structures* is not new, it has been identified as a main issue in scientific database management before, particularly in [2], where the authors argue that existing data models are inadequate for science data needs.

### 1.1 Discrete objects in raster data

Radar technology has been employed to detect severe weather conditions since World War II and, depending on the type of radar being used, densities of cloud droplets, rain drops and hail stones are now being routinely recorded<sup>1</sup>. The measured reflectivities, interpreted as precipitation intensities, are discretized based on a regular geographic grid and made available as two- or three-dimensional bitmaps. To simplify the discussion we restrict examples to two-dimensional image data, extensions to higher dimensions are straightforward.

Cell-like patterns that emerge when visualizing different precipitation intensities are particularly interesting when their structural *evolution* and spatial *propagation* can be investigated. The Swiss Meteorological Institute issues images depicting the precipitation observed over the entire country every ten minutes, thereby providing an information base to investigate the behavior of storm cells, including their movement with respect to the underlying geographic region. This information can also be useful when deciding for which localities severe weather warnings must be announced.

Figures 1 and 2 show weather radar data displayed in different forms. The original objects visible in the left image of Fig. 1 are rather "fuzzy": they vary substantially in size and have frazzled outlines. More importantly though, Figs. 1 and 2 illustrate that interesting storm-related information can readily be extracted from radar images if: a) the data is reduced to *subimages* and b) the data is extended to include *sequences* of subimages. All that is needed is a flexible representational structure to efficiently manage the modified image data.

<sup>1</sup> e.g. <http://www.wral-tv.com/weather/doppler5000/>

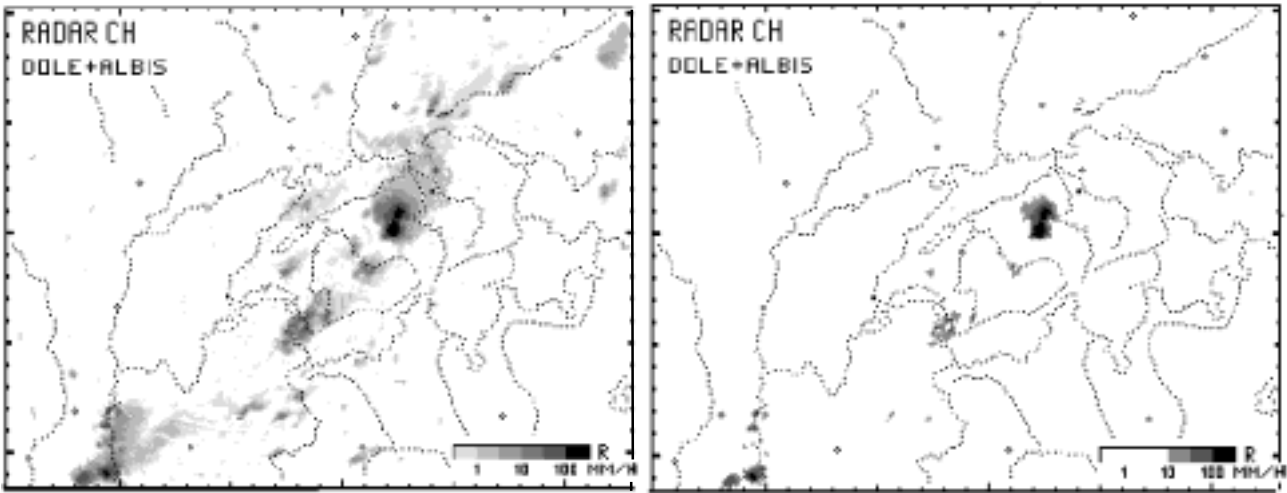


Figure 1. Left: Example of an individual weather radar image as issued by the Swiss Meteorological Institute. Precipitation intensities are divided into six ranges on a logarithmic scale. Values from each range are coded with a different color. Dotted lines mark geopolitical boundaries, major rivers and lakes. Right: A display of only the interesting parts (the storm cells) of the image at the left, reconstructed after the image data were stored in a dynamic four-dimensional file structure. The pixels shown were selected using a range query restricting precipitation intensities to the three highest ranges ( $\geq 10$  mm/h). These individual storm cells are discrete objects that must be detected in an analysis of the image at the left.

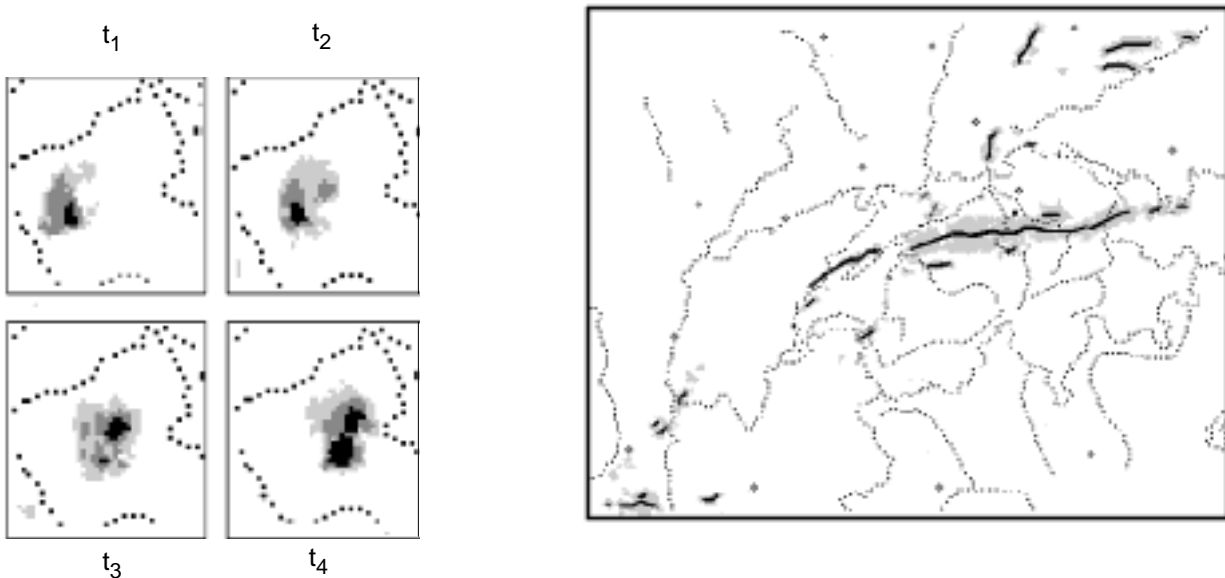


Figure 2. Sequences of subimages yield information about the dynamic behavior of the visualized objects. In the left picture one can observe how a storm cell changes its shape over time. In the right picture the location of the storm cell's centroid during a one hour interval is visualized to show its spatial propagation as it makes its way from the foot of the Bernese Jura towards Liechtenstein. The cell depicted at time  $t_4$  above can be seen in a larger context at the right in Fig. 1.

Researchers investigating conditions that give rise to hail storms are currently working with images as shown at the left in Fig. 1, from which storm cells are isolated *manually*. Automating this procedure can yield substantial savings in time and also provide greater flexibility for further processing of this information. To provide the ne-

cessary software we have chosen an approach that requires solutions to the following two problems:

- 1) An algorithm must be found with which all pixels belonging to an arbitrarily shaped cluster can be identified and stored for further processing. To this end

we notice two important properties of weather radar images:

- a) The environment in which the objects are to be detected is not cluttered (no overlaps). Objects in these images can therefore be recognized based on global features such as *area* or *volume*.
  - b) The objects are *discrete* in the sense that they can be represented as sets of adjacent pixels. Given one pixel of a subimage, its remaining pixels can readily be determined with suitable neighborhood relations.
- 2) It must be possible to extract subimages from sequences of images efficiently. Given the potentially large number of images, it is impractical to process the data as bitmaps for two reasons:
- a) Only a small fraction of an image is used for processing.
  - b) Data must be accessed based on at least 5 attributes simultaneously (x-, y-coordinates, intensity, time, and subimage identifier).

In Sections 2 and 3 we introduce a conceptual framework to address these constraints.

## 1.2 Discrete object representation based on solids

Whenever objects in an uncluttered image environment are composed of simple geometric shapes, it can be advantageous to describe them based on volume rather than outlines, particularly if an object's outlines are difficult to characterize. If a two-dimensional image is suitably tiled, its graphical objects can be discretized and represented as sets of *adjoining 2-boxes*.

The following terminology illustrates the idea in general ([7]).

***n*-box:** An *n*-box in  $\mathfrak{R}^n$  is a subset of the  $\mathfrak{R}$  of the form  $[a_1, b_1] \times [a_2, b_2] \times \dots \times [a_n, b_n]$  where  $b_i > a_i$ , for  $i = 1, 2, \dots, n$ .

***n*-cube:** An *n*-box is called an *n*-cube when  $b_i - a_i = b_j - a_j$  for all  $i, j$  with  $1 \leq i \leq j \leq n$ . A 2-box is called a *rectangle* and a 2-cube is called a *square*.

Note that the sides of an *n*-box are required to be parallel to the Cartesian coordinate axes in  $\mathfrak{R}^n$ .

**adjoining:** Two *n*-boxes are said to be adjoining if they share a corner, line, face or hyperface.

***n*-solid:** A set of adjoining *n*-boxes.

Interpreting each image pixel as a square tile, one can treat subimages as *2-solids*. Extensions to higher dimensions are straightforward.

Using this terminology, an arbitrarily shaped cell in a weather radar image becomes a 2-solid, formed by pixels of a given color (or colors). The coordinates of the individual 2-cubes are a basis to determine a cell's geographic position and the number of 2-cubes relates to a cell's size. Tracking a cell's movement is facilitated by labeling the 2-cube located at the cell's centroid and record its positional change over time (see Section 4). An important aspect of this approach is that identifying and measuring objects can now be reduced to *searching* and *counting*.

## 2 Interpreting bitmapped information as multidimensional point data

Weather radar data has traditionally been stored as large collections of bitmaps, with a separate log kept to record interesting image sequences. If, in the course of a new inquiry, certain characteristics of an image which are not part of the log are deemed to be interesting, then such a log can become useless and the entire archive must be processed again. Also, most physical phenomena appear only in a small part of an image, consequently most of the archive is just a (useless) burden.

Finding collections of small objects in large multidimensional data sets can be time consuming, especially when much irrelevant data must be processed during searching. A storage structure that only manages pixels representing pertinent signals is therefore a first step in the right direction. This section discusses how this can be done.

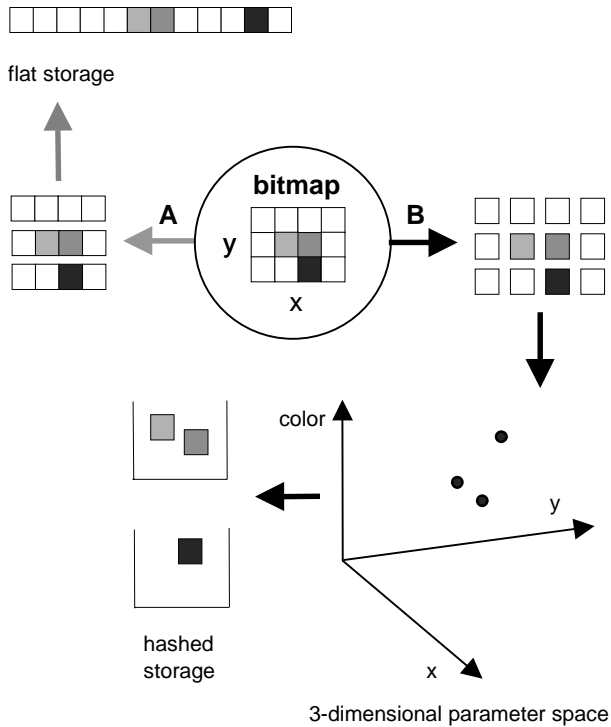
### 2.1 Hashing a bitmap

For image processing applications dealing with images of constant size it is natural to store the data using the pixels's positions as array indices. As a consequence, only subimage extractions based on location can be executed efficiently. If a subimage must be found based on image content or the image's position in an image sequence, as listed under 2) and 3) in Section 1.1, one has to cope with inefficient *comparison searching*, to find a pixel of a given color, for example.

Retrieval of a specific subimage from a sequence is only fast, if a four-dimensional range query based on the coordinates of two diagonally opposite corners of a cut-out, a pixel color range, and image identifiers, can be executed efficiently.

The flexibility required for fast multidimensional range queries can be obtained by breaking up the bitmap's flat structure with a suitable parametrization and *hashing* its content into data buckets which are accessible through a multidimensional index structure. From the possible choices that exist, mapping the data into a *Cartesian product space* has the advantage, that multidimensional spatial data structures can be used. Figure 3 illustrates the

difference between this and the array-based approach with a simplifying diagram.



**Figure 3.** Two different ways to store bitmapped data. Path "A" illustrates linearization of a two-dimensional array into a flat file. Path "B" illustrates the parametrization of a bitmap such that information-carrying pixels can be mapped into a higher-dimensional point space and hashed into storage buckets. "Empty" pixels can thus be ignored.

When evaluating file structures to store weather radar images that have been transformed into higher-dimensional point data, as shown in Fig. 3, it is helpful to first summarize the data's characteristics and the application's requirements:

- 1) The data are *dynamic* (every 10 minutes a new image).
- 2) The data are *multidimensional* and *extensible* (attributes created as a result of processing the data can be used to increase their dimensionality).
- 3) Many of the data's attribute domains are inherently *ordered* (e.g. geographic locality, time).

- 4) The application requires that data can be accessed *symmetrically*, i.e. searching must be equally fast over many attributes.

The gridfile, briefly reviewed in the next Section, represents a storage method that accommodates these requirements.

### 3 Managing multidimensional point data

From the many data structures introduced to manage multidimensional data we chose the *gridfile* ([6]) because it dynamically organizes multidimensional ordered search spaces, preserves each attribute domain's order, and grows and shrinks gracefully in response to insertions and deletions.

#### 3.1 Concepts and terminology

The gridfile has been designed to ensure efficient access to data stored on disk by collecting neighboring data points into *buckets*. The following two principles guided the design of the gridfile.

*Two disk access requirement.* A fully specified query must retrieve a single record in at most two disk accesses: The first to the correct portion of the directory, the second to the correct data bucket.

*Efficient range queries with respect to all key attributes.* The storage structure should preserve the order defined on each attribute domain as much as possible, so that records that are "close" with respect to the domain of any key attribute are likely to be stored in the same physical storage block.

The gridfile manages its data by *automatically* partitioning the embedding space from which the data are drawn in a grid-like fashion, so that each interval boundary cuts the entire search space in two, as shown in Fig. 4. Adaptation of the grid's granularity to the data's density distribution happens during insertion and deletion. The grid partition is modified one interval boundary at a time which reduces updates to one-dimensional changes: the partition of a single attribute domain is modified either by splitting one of its scale intervals in two, or by merging two adjacent scale intervals into one.

This grid partition divides the record space into rectangular shaped blocks, called *grid blocks*. For reasons of access efficiency, all records in one grid block are stored in the same bucket. Several grid blocks can share a bucket to avoid arbitrarily low memory utilization. Such sets of grid blocks are called *bucket regions* and form multidimensional, box-shaped rectangles. These convex bucket regions are pairwise disjoint and the set of all bucket regions  $\{B_i\}$  spans the record space.

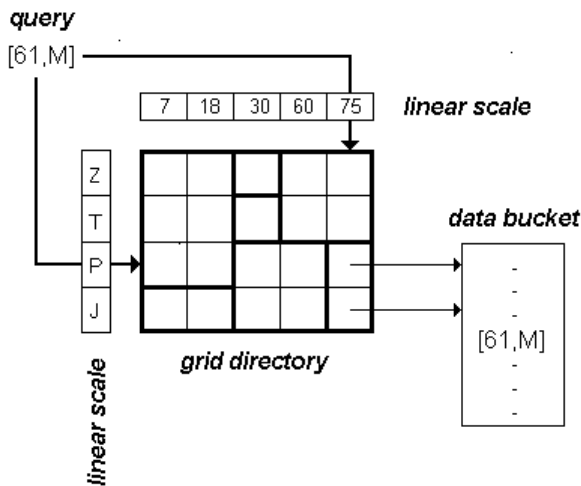


Figure 4. The gridfile's concepts applied to hypothetical two-dimensional data. Retrieval of a single record starts at the linear scales which store the interval boundaries ( $i$ ) of the grid partition. It continues in the appropriate part of the grid directory, and terminates in the correct data bucket. Heavy lines indicate boundaries of regions whose data points are all stored in the same bucket.

A Find for a fully specified query, such as Find [61, M] in Fig. 4, is executed as follows. The attribute value 61 is converted into interval index 5 through a search in the scale drawn horizontally, and M is located in the interval with index 2 in the scale drawn vertically. These linear scales are stored in central memory; thus the conversion of attribute value to interval index requires no time in a model where only the number of disk accesses counts. The interval indices 5 and 2 provide direct access to the correct element of the grid directory, where the bucket address is located.

### 3.2 A linearly growing grid directory and high average bucket occupancy

At the core of the gridfile is the *grid directory* whose purpose is to dynamically maintain the correspondence between grid blocks and data buckets. A grid directory, managing data with  $k$  key-attributes, basically consists of  $k$  one-dimensional arrays called linear scales and a dynamic  $k$ -dimensional grid array whose elements (pointers to data buckets) are in one-to-one correspondence with the grid blocks of the partition introduced by the scales. The grid directory maintains symmetrically a dynamic partition on the space of all key-values in order to support efficient range queries with respect to all key-attributes. In addition to treating all key-attributes alike, the grid directory adapts its structure automatically to the data's distribution in order to maintain an average bucket occupancy of around 50% over the entire life of the file.

Methods to organize the grid array fall into two broad categories: those organizing grid blocks and those organizing bucket regions. Directories that manage individual grid blocks carry the potential for superlinear growth and for this reason hierarchical structures are often introduced to keep fine tilings local to the region where they are needed. Directories that manage bucket regions on the other hand *grow linearly* with the number of data buckets, regardless of the tiling that is imposed on the record space, because they maintain one directory entry of constant size for each data bucket.

The price for predictable directory growth, however, is unpredictable directory access because a grid block can no longer be used to calculate a directory's array address, it must be used as argument to search a list of bucket regions instead (see below). It also means that an originally defined performance goal, the two disk access principle, must be suspended. This is no serious drawback because: 1) Large central memories are now common even with personal computers, 2) the region directory is comparatively small and its growth rate can be controlled with the size of the data buckets. It is therefore possible to maintain grid directories in central memory even for large files, reducing disk accesses to data bucket retrieval, i.e. one disk access for fully specified queries!

#### The region directory

By definition, each bucket region can be uniquely identified with two diagonally opposite *vertices*, formed by the intersection of  $k$  scale interval boundaries such that no interval boundary is common to both vertices. Given a grid block's or a bucket region's  $2k$  interval boundaries, those associated with the small boundary in each key-attribute intersect at the *near vertex*, all others at the *far vertex*. A given grid block is included in a bucket region  $B_j$  if and only if the coordinates of the grid block's near vertex are smaller than those of the far vertex of  $B_j$  and the coordinates of its far vertex are larger than those of the near vertex of  $B_j$  (a bucket region is never smaller than a grid block). The *region directory* (introduced in [4]) takes advantage of this fact to organize the correspondence between grid blocks and data buckets. Figure 5 illustrates how the data bucket for a given grid block is found with the region directory.

Data structures used to represent a region directory can be simple, e.g. a linear list of bucket region vertices, lexicographically ordered, say, by near vertex as shown in Fig. 6. The query depicted in Fig. 4 (scale indices 5 and 2) translates to a grid block with near vertex [4,1] and far vertex [5,2]. The bucket region for this grid block has vertices [4,0] and [5,2] and any record satisfying the query will consequently be found in the bucket with address  $a_3$  (Fig. 6).

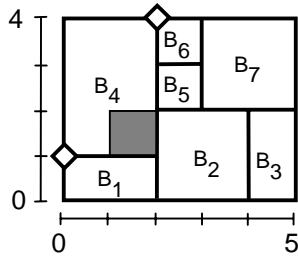


Figure 5. Graphical representation of the bucket regions of the gridfile illustrated in Fig. 4. The grid block shown (gray square) falls between the near and far vertex (diamonds) of bucket region  $B_4$ .

<u>bucket region</u>	<u>near vertex</u>	<u>far vertex</u>	<u>bucket address</u>	<u>% bucket occupancy</u>
$B_1$	[0, 0]	[2, 1]	$a_1$	40
$B_4$	[0, 1]	[2, 4]	$a_4$	55
$B_2$	[2, 0]	[4, 2]	$a_2$	80
$B_5$	[2, 2]	[3, 3]	$a_5$	65
$B_6$	[2, 3]	[3, 4]	$a_6$	20
$B_7$	[3, 2]	[5, 4]	$a_7$	60
$B_3$	[4, 0]	[5, 2]	$a_3$	20

Figure 6. A region directory for the gridfile shown in Fig. 4 organized as linear list, ordered by near vertex. Near and far vertices are represented as search space coordinate vectors [x-coordinate, y-coordinate]. A grid block of the grid directory falls into one of the regions delineated by the near and the far vertices.

### Compacting data buckets

A common characteristic of many dynamic file structures is their unpredictable memory utilization. This may or may not be a cause for concern, depending on whether efficient update operations or efficient memory utilization is important. For this reason, we distinguish between "active" and "archival" gridfiles. An active gridfile is subject to insertions and deletions, with an average bucket occupancy of around 50%. Archival gridfiles are no longer modified dynamically so that their data structures can be adjusted for better memory utilization. To this end we notice that the contents of data buckets can be joined pairwise whenever their combined number of records does not exceed the gridfile's bucket capacity. In this way, the average bucket occupancy can easily be doubled and often improves to close to 100%.

To illustrate, consider the hypothetical region directory shown in Fig. 6 which represents a gridfile with an average bucket occupancy of 48%. The contents of the bucket that stores records from bucket region  $B_1$  can be moved to the empty part of the bucket assigned to bucket

region  $B_4$ , raising the occupancy of the latter to 95%. The bucket associated with bucket region  $B_1$  is then discarded, but the records it contained can still be found if its address in the region directory is replaced with the address of the bucket corresponding to bucket region  $B_4$ . Figure 7 shows the region directory of Fig. 6 after all buckets have been processed in this manner, yielding a gridfile with 85% average bucket occupancy.

<u>bucket region</u>	<u>near vertex</u>	<u>far vertex</u>	<u>bucket address</u>	<u>% bucket occupancy</u>
$B_1$	[0, 0]	[2, 1]	$a_4$	
$B_4$	[0, 1]	[2, 4]	$a_4$	95
$B_2$	[2, 0]	[4, 2]	$a_2$	100
$B_5$	[2, 2]	[3, 3]	$a_5$	85
$B_6$	[2, 3]	[3, 4]	$a_2$	
$B_7$	[3, 2]	[5, 4]	$a_7$	60
$B_3$	[4, 0]	[5, 2]	$a_5$	

Figure 7. The region directory of Fig. 6 after the buckets of the gridfile have been joined pairwise to improve memory utilization. Note that the size of the directory and the partition represented by bucket regions remains unchanged.

Compacting data buckets in this manner while a gridfile is still active is a topic for further research.

### 3.3 Overhead incurred when storing image data with a gridfile

Before a radar image can be stored as higher-dimensional point data, its bitmap's content must be parametrized to fit the records of a gridfile. The following type declaration illustrates how this can be done with four pixel-attributes (from the 285 x 255 bitmap of a radar image).

```

TYPE
Pixel =
RECORD
  x-coord: WORD; {key 1: pixel's horizontal
                  position}
  y-coord: BYTE; {key 2: pixel's vertical
                  position}
  intensity: BYTE; {key 3: precipitation
                   intensity, color coded}
  time: WORD; {key 4: time at which intensity
                was observed}
END;
```

This data structure allows direct access to a hail storm's radar echo simply by assigning the highest precipitation intensity to search key 3. The corresponding records also show where the storm was observed (search keys 1 and 2) and at what time (search key 4).

To evaluate actual storage requirements, data from 144 bitmaps, representing images recorded during a 24

hour period with heavy precipitation were inserted into a gridfile. Each data bucket has a capacity of 100 entries of type `Pixel`, a directory page manages 500 data buckets. Table 1 summarizes characteristic properties of the resulting gridfile’s data structures.

Number of images	145 (incl. geopolitical outline)
Size of each bitmap	32 kByte (pixel depth: 4 bits)
Number of data items	573005 (inf. carrying pixels)
Number of data buckets	8938
Size of data bucket file	5.45 MByte
Number of directory entries	8938
Size of directory file	184 kByte
Size of scale file	2.7 kByte
Average bucket occupancy	64% (active gridfile)
Search space partition:	111 x 111 x 111 x 4 scale intervals = 5'470'524 grid blocks

**Table 1. Characteristics of a four-dimensional gridfile storing information carrying pixels of weather radar images which were issued during a 24 hour period. For comparison, the 144 bitmaps of the original images occupy 4.6 MByte (uncompressed). Note that on a day without precipitation the gridfile – contrary to a bitmap – would be empty.**

An individual record of type `Pixel` takes up 12 times as much storage as a single, 4 bit deep pixel. Consequently, the 64% bucket occupancy in Table 1 means that the gridfile will require more storage than the equivalent bitmaps only when on average more than 5.2% of the pixels carry relevant information. The images shown in Fig. 1 cover an area of approx. 240'000 km<sup>2</sup>, but only an area of approx. 1000 km<sup>2</sup> is observed when hail storms are investigated. A typical gridfile can therefore accommodate more than twelve storms at a time before it grows larger than the original bitmaps; which is more than enough.

We deliberately chose an extreme example weather-wise with images having an average of 6.5% information-carrying pixels to illustrate that even in the worst case (w.r.t this application) we are at par, storagewise, with uncompressed bitmaps. Only uncompressed bitmaps are considered for this comparison because the information that is being observed is not readily accessible in compressed images. If archival gridfiles (approx. 90% avg. bucket occupancy) are used or if precipitation is limited to a small geographical area, the gridfile outperforms bitmaps by a large margin.

Savings in storage space become particularly important when one considers technically more advanced applications, because: a) the ratio of information carrying to "empty" pixels grows exponentially in the number of dimensions ("curse of dimensionality") and b) doppler radar stations generate data in the order of ½ to 1 GByte per day.

Information carrying pixels stored in a gridfile can be accessed with queries expressed as convex regions in the four-dimensional space spanned by the search-key attribute domains. These queries can now be utilized to selectively visualize only those parts of the images that are relevant to a particular question.

### 3.4 Comparison with other approaches

In the United States, weather radar data is analyzed using primarily SCIT [9] and TITAN [1], two different storm analysis packages originally designed for real time analyses. Using this software, storm cells are identified and tracked during the storm’s entire life. With these systems it is possible to archive the entire set of original bitmap data or some calculated storm properties such as centroids or cell size. In order to work with this archived information, the complete series of bitmaps must be restored. This typically adds up to a few hundred megabytes of data. Consequently, not many stormy days can be processed at one time.

A multidimensional parametrization such as the one described in the previous section allows the storage of storm data without unnecessary overhead: all non-storm related data is left out. Furthermore, all storm related data is fully accessible in its archived form, without the need for restoration. Table 2 summarizes some advantages and disadvantages of the different approaches.

## 4 Object detection and motion registration

Now that radar images are stored in a gridfile, fast direct access to parts of a discrete object of interest (a single pixel of a storm cell) is possible. Next we proceed to isolate entire objects in an image and capture their movements in space and time.

### 4.1 Object detection and object storage

The method we introduce has the following components:

- 1) *Raw data*: a set of bitmaps from a time-ordered sequence of images.
- 2) *Orderpreserving, m-d storage structures*: a four- and a five-dimensional gridfile.
- 3) *Object detection algorithm*: to find all adjoining 2-cubes and organize them into 2-solids, regardless of the involved cell’s shape or location.
- 4) *Motion registration algorithm*: to determine which 2-solids represent the same cell in two consecutive images.

Storage technique	Advantages	Disadvantages
entire bitmaps	<ul style="list-style-type: none"> <li>no information lost</li> <li>further processing possible</li> <li>no preprocessing needed</li> </ul>	<ul style="list-style-type: none"> <li>large storage required</li> <li>processing of only a few days at a time</li> </ul>
calculated properties	<ul style="list-style-type: none"> <li>storage requirement negligible</li> </ul>	<ul style="list-style-type: none"> <li>preprocessing required (calculation of the properties such as centroids, cell size)</li> <li>further processing not possible</li> <li>no access to original storm data</li> </ul>
multidimensional point data	<ul style="list-style-type: none"> <li>significant storage reduction</li> <li>fast access to both original data and calculated properties</li> <li>different properties can be combined in a single query</li> <li>further processing possible</li> </ul>	<ul style="list-style-type: none"> <li>preprocessing required (parametrization and insertion in gridfile)</li> </ul>

Table 2. Comparison of different methods to detect and track storm cells.

The data's organization plays a central role with this approach because a) climatologists investigating severe weather conditions often work with storm data from image *archives* and b) the practicality of our method depends on the availability of dynamic and symmetric four and five-dimensional *file structures*.

Storms appear in a weather radar image in the form of small cells, made up of pixels whose color represents a high precipitation intensity, say  $I_{max}$ , surrounded mostly by pixels covering the area whose precipitation falls into the next lower intensity range  $I_{max-1}$ .

The procedure to isolate storm cells can be summarized as follows:

- 1) Select those images in the archive which fall into the specified time period.
- 2) In these images select all pixels with values falling into the ranges  $I_{max-1}$  and  $I_{max}$ .
- 3) Organize these pixels into 2-solids (without considering their value).
- 4) For each 2-solid determine its centroid before storing the corresponding cell.

Figure 8 illustrates what is being observed and how these objects can change over time. The cells that are of interest do not only change their position, shape, and size over time but are also likely to split or merge.

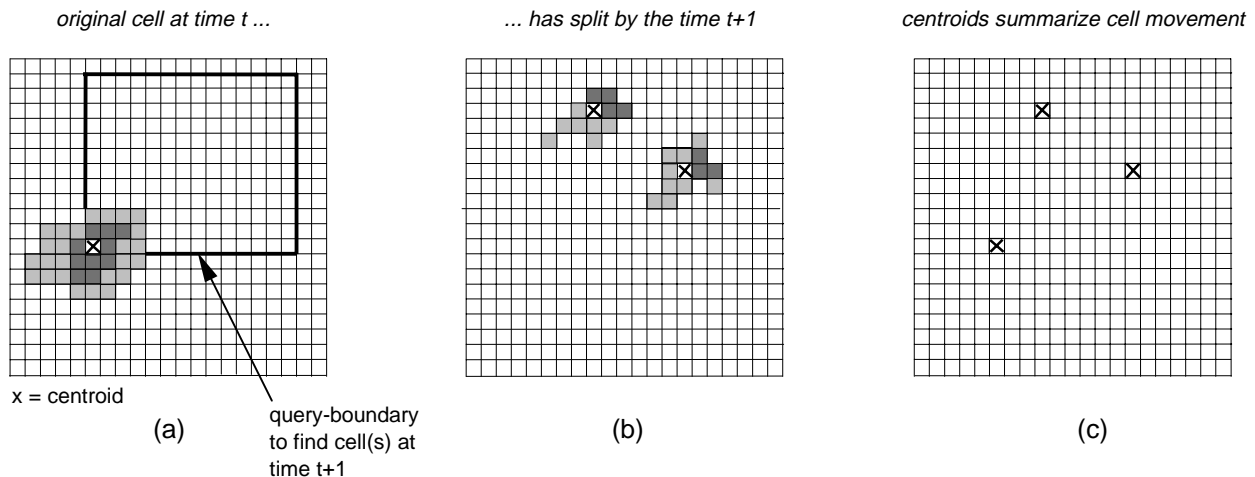


Figure 8. Graphical representation of subimages (each square = 1 pixel) with identical cutout domains depicting a cell moving from the area shown in (a) to the region shown in (b). Even though the pixels in subimages (a) and (b) represent the same cell at two different points in time, interpreting them as 2-solids will yield three individual but related objects. The heavy outline in diagram (a) indicates a search region that will find the cell's successor(s) at time  $t+1$  for a given motion vector. In (c) only the centroids of the cells in (a) and (b) are shown.

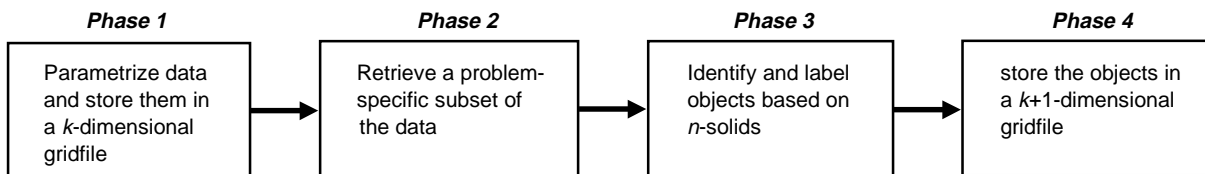


Figure 9. The four steps carried out to isolate objects in a raster image. Note how the gain in information translates into an increase in a data structure's dimensionality.

At the data management level, the procedure to detect and store discrete objects can be divided into four phases as shown graphically in Fig. 9.

#### Phase 1: Providing initial conditions

Store all information-carrying pixels of all images generated during the time period under investigation in a four-dimensional gridfile, assigning search-key attributes as shown in Section 3.3. Note that individual images can be reconstructed based on appropriate range queries over the gridfile, with the time at which the precipitation to be visualized was recorded assigned to key 4.

#### Phase 2: Reducing the working set

Hail storms are typically local and of short duration, but often "embedded" in an extended rainy period covering a comparatively large geographical area. Under these conditions, a small amount of interesting data is obscured by large amounts of non-relevant information. All pixels representing particular precipitation intensities at a given point in time are retrieved with queries in the key-domains 3 and 4. In this way the data set is reduced substantially by extracting subimages as shown in Fig. 1 (image at left vs. image at right), containing only information about storm cells observed during the specified time period. It is now easy to visualize data showing how a cell changes its shape over time (Fig. 2, left).

#### Phase 3: Object detection

Individual cells are readily identified by treating them as 2-solids. The following recursive algorithm "grows" completely into a 2-solid of any shape, to capture all its 2-cubes (pixels), starting with an arbitrary 2-cube. The following program segment (pseudo Pascal notation) illustrates its application.

```

VAR
  data: array [x-interval, y-interval] of BYTE;
  time, cell_number, cell_size, r, s: WORD;

PROCEDURE object_detection(x, y: WORD;
  VAR count: INTEGER);

BEGIN
  {insert current 2-cube in the five-dimensional
  gridfile}
  Insert(x,y,data[x,y],time,cell_number);
  INC(count); {count is the 2-solid's number

```

```

of 2-cubes}
data[x,y]:= 0; {prevent further processing
of this 2-cube}
{inspect all adjoining 2-cubes recursively}
IF data[x,y+1] > 0
  THEN object_detection(x,y+1, count);
IF data[x,y-1] > 0
  THEN object_detection(x,y-1, count);
IF data[x+1,y] > 0
  THEN object_detection(x+1,y, count);
IF data[x-1,y] > 0
  THEN object_detection(x-1,y, count);
IF data[x+1,y+1] > 0
  THEN object_detection(x+1,y+1, count);
IF data[x+1,y-1] > 0
  THEN object_detection(x+1,y-1, count);
IF data[x-1,y+1] > 0
  THEN object_detection(x-1,y+1, count);
IF data[x-1,y-1] > 0
  THEN object_detection(x-1,y-1, count);
END; {object_detection}

{main program}
BEGIN
  {from a given subimage assign all pixels
  representing specific intensities to the
  working array "data"}
  cell_number:= 1;
  cell_size:= 0;
  WHILE {there is at least 1 element in data
  with value} > 0 DO
  BEGIN
    {find all adjoining 2-cubes, r and s are the
    row and column indices of a pixel}
    object_detection(r,s,cell_size);
    INC(cell_number);
  END;
END.

```

In order to find the storm cells in a subimage all relevant pixels are retrieved from the four-dimensional gridfile and temporarily stored in a two-dimensional array (variable data above). Starting with an arbitrary array element whose value is greater than zero, the cells can be identified by repeatedly calling procedure object\_detection until all elements of the array data are set to zero. From a given 2-cube the algorithm proceeds to an adjoining 2-cube if its pixel value is greater than zero and after processing sets its value to zero (termination condition for the recursion).

#### Phase 4: Object storage

For efficiency reasons, it must be possible to address each cell individually. To this end pixels are assigned an additional search-key attribute, identifying the cell to which they belong (variable `cell_number` declared before procedure `object_detection` listed in Phase 3). Records of the corresponding gridfile are of the following type:

```
TYPE
Cell_Pixel =
RECORD
  x-coord: WORD; {key 1: pixel's horizontal
                  position}
  y-coord: BYTE; {key 2: pixel's vertical
                  position}
  intensity: BYTE; {key 3: precipitation
                   intensity, color coded}
  time: WORD; {key 4: time at which intensity
                was observed}
  cell: WORD {key 5: cell-identifier for
               this pixel}
END;
```

Limiting a data file to only contain pixels belonging to a storm cell drastically reduces its size, whereas the additional dimension only slightly increases a data set's complexity.

## 4.2 Motion registration

Data representations of individual storm cells are useful in many ways: they can lead to insights into the way storms build up, for instance, or they can form the basis for algorithms to track and forecast the route of a storm cell in real time (severe weather warnings).

The critical part when tracking a cell, observed at time  $t_i$ , is finding out which cell it corresponds to in the image taken at time  $t_{i+1}$ . Difficulties arise because new cells can appear, existing cells can split (Fig. 8), merge or disappear. For each cell there are five possible outcomes:

- 1) A cell at  $t_{i+1}$  has no predecessor at  $t_i$  (a new cell came into existence).
- 2) A cell at  $t_i$  has no successor at  $t_{i+1}$  (an existing cell disappeared).
- 3) A cell at  $t_i$  has exactly one successor at  $t_{i+1}$  (translation of a cell).
- 4) A cell at  $t_i$  has more than one successor at  $t_{i+1}$  (the cell split into several parts).
- 5) A cell at  $t_{i+1}$  has more than one predecessor at  $t_i$  (several cells merged into one cell).

### Reducing a cell to a point

Since a cell's shape undergoes constant deformation, it is more convenient to track the movements of its centroid ([5]). Most storm cells have a roughly circular or elliptic shape so that the following simple centroid calculation yields good results: find the median pixels when

counting a) in row major order and b) in column major order; the centroid is at the intersection of the row containing the median found in a) and the column containing the median found in b).

To track centroids we apply a method similar to the one used in TITAN [1], SCIT [9] and the current NEXRAD algorithm [8]: for each centroid at time  $t_i$  a query region which is likely to contain its successor at time  $t_{i+1}$  is determined (heavy outline in Fig. 8 (a)). The extent of the region is given by a predetermined search radius and a forecasted motion vector which can either be the most recent motion vector, or be calculated as a weighted average of several recent motion vectors. A centroid at time  $t_{i+1}$  is considered to belong to the cell that prompted the query at time  $t_i$  if it falls inside the search radius and consequently inside the query region.

In a first step, pixels considered to be part of a severe storm, observed during the time period from  $t_{start}$  to  $t_{stop}$ , are retrieved from the gridfile. They must have values representing high precipitation intensities ( $I_{max}$ ,  $I_{max-1}$ ).

In the second step, the centroids of all cells observed at time  $t_i$  ( $t_{start} \leq t_i \leq t_{stop}$ ) are calculated. Individual motion vectors can then be established based on a centroid's position at time  $t_i$  and  $t_{i-1}$  and combined into directed graphs, restricted by the five possible outcomes of a cell transformation listed above. Starting with a sequence of images as shown in Fig. 1 left, this procedure leads to images depicting storm tracks as shown in Fig. 2 right.

Dividing motion registration into the two steps of storm cell identification and cell matching allows one to compare different tracking algorithms for a given set of centroids, and also to evaluate a given tracking algorithm with different sets of centroids representing the same storm.

## 4.3 Data management's role

The gridfile's fast range query facility and its dynamic capabilities (insertion and deletion) can be exploited to capture a cell's spatial propagation as follows ( $\phi$  = "don't care" search key value).

- 1) **Retrieve** the centroid of the cell with the smallest cell identifier: `Find(x-coordinate =  $\phi$ , y-coordinate =  $\phi$ , intensity = centroid, time =  $t_{start}$ , cell_ident = smallest)`. Call this pixel *current centroid*. If no cell with identifier `smallest` exists, continue with  $t_{start} := t_{start+1}$ .
- 2) Based on the x- and y-coordinates of *current centroid* and the criteria that delimit its forecasted movement, issue a **range query** (heavy outline in Fig. 8 (a)) to retrieve those pixels that could represent this centroid's position at time  $t_{start+1}$ : `Find(x-`

coordinates = x-range, y-coordinates = y-range, intensity = centroid, time =  $t_{start+1}$ , cell\_ident = min). Call these pixels *future centroid*.

- 3) Replace the cell\_ident value of all *future centroid* from 2) with the cell\_ident value of *current centroid* from 1) and call them *new current centroid*. **Insert** every *new current centroid* in this gridfile and **delete** all *future centroid*. Take any *new current centroid*, make it the *current centroid* and continue with step 2) until time =  $t_{stop}$  or the range query in 2) returns empty. Repeat until no *new current centroid* is left.
- 4) Let the next larger cell identifier become the smallest cell identifier and continue with step 1) until all cells are processed.

Before this routine starts, every centroid has its unique cell identifier. When this routine finishes, centroids can only be distinguished by combining their cell identifier with the time at which the cell was observed. In other words, centroids that mark the path of a cell lie on a hyperplane in the 5-dimensional search space of the gridfile. The entire path of cell  $c_i$  can now be retrieved with the range query Find(x-coordinate =  $\phi$ , y-coordinate =  $\phi$ , intensity = centroid, time =  $\phi$ , cell\_ident =  $c_i$ ).

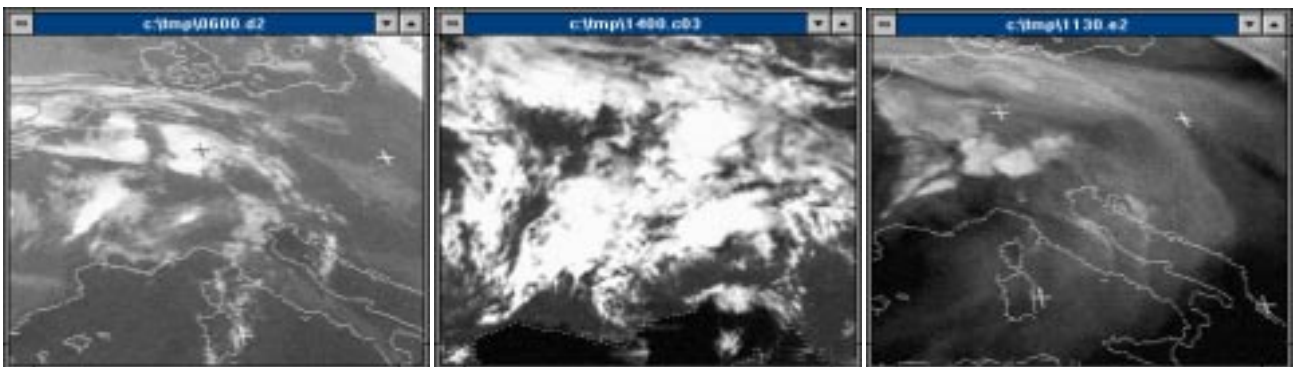
Complicating conditions, such as what to do when the paths of two cells cross, have been omitted to simplify the discussion.

## 5 Discrete objects in heterogeneous data sets

The following typical query illustrates the data diversity with which a meteorologist has to cope: *Find areas where the air is warm and moist (ground measurements) and growing clouds can be observed (satellite images) on days when the probability for severe storms is high (stability indices calculated from sounding data)*. Searching such diverse sources is impractical when the data are in their original format. Furthermore, each measuring device provides an individual temporal and spatial representation of the data. Again, suitable parametrizations can lead to practical solutions with the advantage that the data need not be interpolated to a common grid, they only have to be represented in a *common coordinate system*.

In a first step we wanted to be able to compare the tracks of storm cells as they appear in weather radar images with tracks observable from satellites and for this reason extended the method introduced in Section 4 accordingly ([3]). Figure 10 shows instances of the three types of Meteosat images that were included. Coast lines are indicated with dotted lines (the Mediterranean region is readily recognized).

First, the images are georeferenced, i.e. corrected (bilinear interpolation) to compensate for the different viewing angles and reduced to the same geographic area covering Switzerland and surroundings ( $44.8^\circ$  ..  $48.8^\circ$  N,  $3,6^\circ$  ..  $10,6^\circ$  E), such that each pixel represents an area of approx.  $1.7 \text{ km} \times 1.7 \text{ km}$ . Figure 11 shows georeferenced and size-adjusted infrared and water vapor images.



(a)

(b)

(c)

Figure 10. Three different types of Meteosat images representing (a) the infrared spectrum, one image every hour, (b) the visible spectral band, one image every hour, and (c) water vapor concentrations, one image every six hours, over the south-central part of Europe. One pixel covers approx.  $5 \text{ km} \times 8 \text{ km}$  in images (a) and (b) and approx.  $2.6 \text{ km} \times 4 \text{ km}$  in image (c). All images have a size of 640 kByte.

Once the bitmaps represent comparable information, cells are extracted, using a threshold technique, and inserted in the five-dimensional gridfile. We are then in a position to access in a single data structure discrete objects representing related atmospheric phenomena that have been observed by four different instruments. By formulating the appropriate query it is just as easy to extract sequences of weather radar subimages as shown in Fig. 2 as it is to display sequences of cells that appear both in infrared satellite images and weather radar images at the same time, as shown in Fig. 12.

## 6 Summary

This paper introduces a novel approach to store bit-mapped data, allowing fast symmetric access to several different attributes in addition to a pixel's coordinates, so that operations based on *image content* become practical. Since only information carrying pixels are stored we have a method that incorporates the possibility to *compress* images without loss of relevant data. This type of compression is attractive because images can subsequently be processed without prior decompression.

With representative examples we illustrated that it can be practical to treat objects in images based on volume rather than outlines and showed that structures, embedded in a sequence of images, can readily be isolated simply by combining objects that appear in individual images with a single multidimensional, dynamic data

structure. The method we presented is *universal* in the sense that it is applicable whenever an event can be described with discrete objects in an uncluttered environment. Furthermore, *heterogeneous* data sets can readily be combined if they are suitably parametrized.

The choice of a particular data storage method limits the ways in which data are processed (by definition of the term *data structure*). As a consequence, it can be useful to critically examine traditional storage structures for scientific data to find out if they could be better and more efficiently exploited with other representational structures. Our example illustrates this phenomenon for the case where the storage methods chosen are tailored to efficiently manipulate entire raster images but do not support the domain scientist who needs access to specific information visible in these images.

First experiences gained with a prototype implementation on a PC convince us of the method's practicality and usefulness. The efficiency of cell tracking based on archived data has been improved by orders of magnitude: it can now be done in a matter of minutes instead of days. Accessing information based on arbitrary choices of geographic location or precipitation intensities, for example, is now just a matter of issuing the appropriate (range) queries. A future project will extend the approach described and investigate the consequences and benefits of combining satellite and weather radar images with ground measurement data.

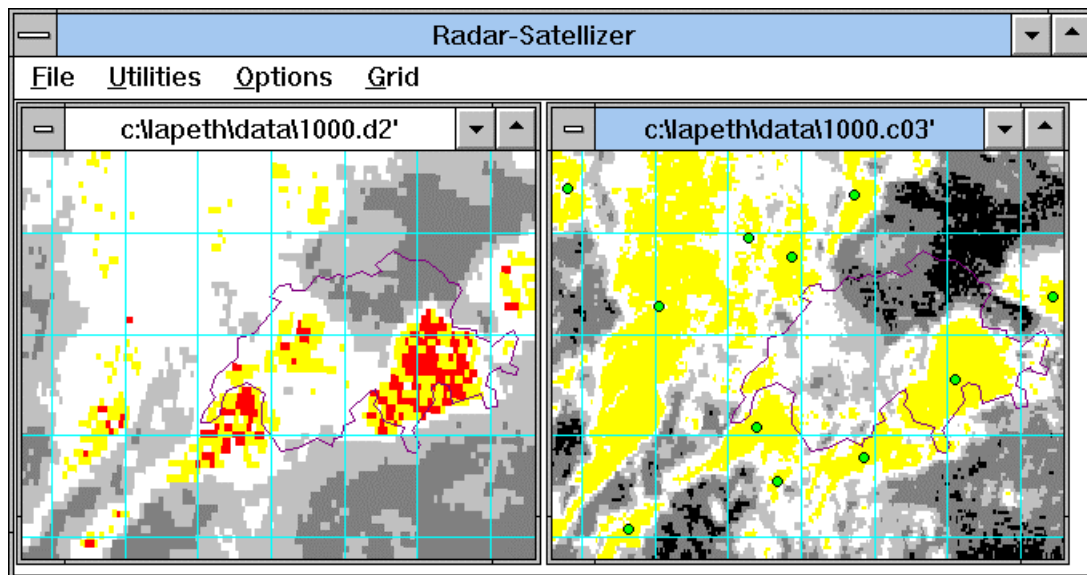


Figure 11. The picture at the left shows a georeferenced and size-adjusted infrared satellite image. High intensity cells are shown in dark grey. The geographic outline represents the borders of Switzerland. The picture at the right shows a georeferenced and size-adjusted water vapor satellite image with small circles marking cell centroids. The data come from Meteosat images captured on 10. Aug. 1994, 10 am.

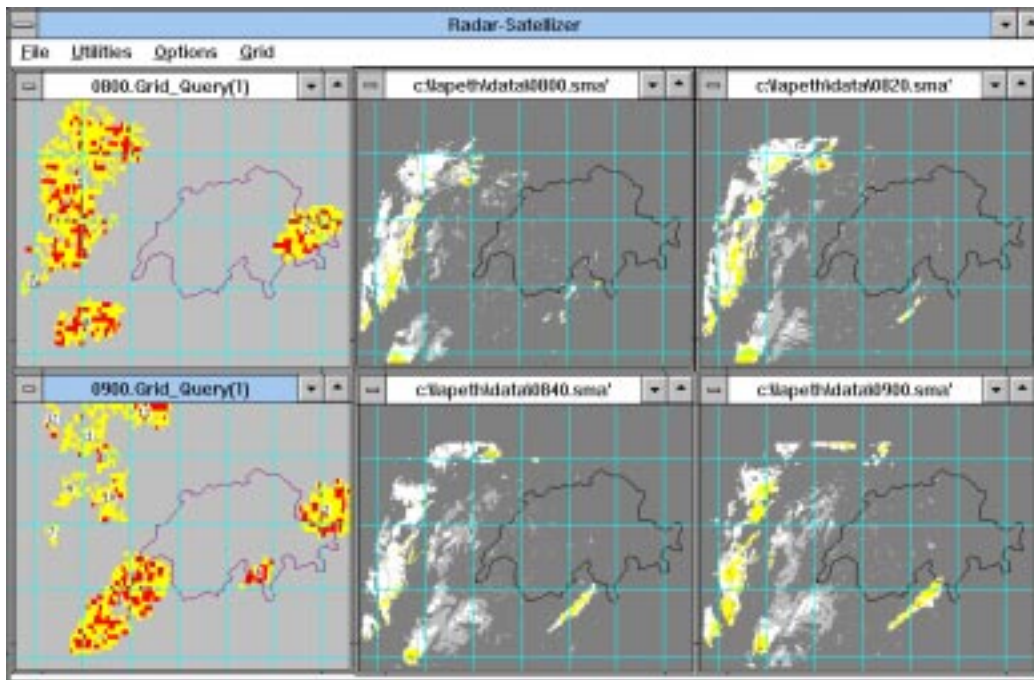


Figure 12. When cells that were extracted from infrared satellite images (Fig. 10, left) are stored in a five-dimensional gridfile, it is possible to retrieve subimage sequences with simple range queries (panels at left). A second range query instantly produces weather radar images for comparison (panels at center and to the right). Cell no. 2 in the satellite image is absent in the radar data because the alps obstruct the radar signal. Precipitation patterns were observed between 8 and 9 am on 10. Aug. 1994.

## References

- [1] Dixon, M. & Wiener, G.: TITAN – Thunderstorm identification, tracking, analysis and nowcasting – a radar based methodology. *Journal of Atmospheric and Oceanic Technology*, Vol. 10, No. 6, 785 - 794. 1993.
- [2] French, J.C., Jones, A.K., Pfaltz, J.L.: *Scientific Database Management*, Report of the Invitational NSF Workshop on Scientific Database Management, Charlottesville VA. Report No. TR-90-21, Dept. of Computer Science, University of Virginia, Charlottesville, VA 22903, 1990.
- [3] Hendseth, A.: *Entwurf und Implementation einer Dateistruktur für die gemeinsame Verwaltung von Satelliten- und Radarbildern des LAPETH*. Diploma thesis, Institute of Scientific Computing, ETH Zürich, Switzerland, 1995.
- [4] Hinterberger, H.: *Data Density: A Powerful Abstraction to Manage and Analyze Multivariate Data*. Dissertation ETH No. 8330, 1987.
- [5] Messmer, B.: *Entwicklung neuer Methoden für die Verwaltung und Visualisierung von Wetter-Radar-daten*. Diploma thesis, Institute of Scientific Computing, ETH Zürich, Switzerland, 1993.
- [6] Nievergelt, J., Hinterberger, H., Sevcik, K.: The Grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, Vol. 9, No. 1, pp. 35-45, 1984.
- [7] Trotter, W.T., Jr.: *Tiling bounded open sets with squares that touch the boundary*. In Goodman, J.E., Litwak, E., Malkevitch, J., Pollack, R. (eds.), *Discrete Geometry and Convexity*, Annals of the New York Academy of Sciences, Vol. 440, pp. 304-322, 1985.
- [8] *NEXRAD Algorithm Report*, U.S. Dept. of Commerce, 1985.
- [9] Witt, A., and Johnson, J.T.: *An enhanced storm cell identification and tracking algorithm*. Preprints Conference on Radar Meteorology, Norman, OK, 1993.

## Acknowledgements

This project was supported in part by the Swiss National Science Foundation under Grant No. 20-36395.92. We are grateful to A. Hendseth for designing and implementing the software required to handle satellite images. Weather radar and Meteosat images were available by courtesy of the Swiss Meteorological Institute.