8249-EN-01
DTIC

Conditional Estimation of Vector Patterns in Remote Sensing and GIS

Interim Report 6

Principal investigator: Dr. J.M.F. Masuch

15 September 1999

United States Army
European Research Office of the U.S. Army

USADSG-UK, Edison House
223 Old Marylebone Road
London,NW1 5TH
England

Contract Number: N68171-97 C 9027

Approved for Public Release;distribution unlimited

CCSOM/Applied Logic Laboratory
PSCW - Universiteit van Amsterdam
Sarphatistraat 143
1018 GD AMSTERDAM
The Netherlands
tel: #.31.20.525 28 52
fax: #.31.20.525 28 00
e-mail: ccsoff@ccsom.uva.nl

R&D 8249-EN-01
Broad Area Announcement Proposal
submitted to the
Remote Sensing / GIS Center USACRREL
72 Lyme Road
Hanover, New Hampshire 03755 USA

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 15, '99	6th interim report (15 September)

4. TITLE AND SUBTITLE
Conditional Estimation of Vector Patterns in Remote Sensing and GIS

5. FUNDING NUMBERS
N68171 97 C 9027

6. AUTHOR(S)
Dr J.M.F.Masuch

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
CCSOM / PSCW / University of Amsterdam
Sarphatistraat 143
1018 GD AMSTERDAM NL

8. PERFORMING ORGANIZATION REPORT NUMBER
BAA III 99'2

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

Within this project report we address specific programming requirements for the adaptation of C, C++, and JAVA computer code into a working format for Corps of Engineers modeling in water control, water quality, and emergency response. This effort is cooperatively conducted with the professional researchers at the Remote Sensing / GIS Center of the US Army Cold Regions Research and Engineering Laboratory in Hanover, New Hampshire. The goal of this effort is to program and implement vector algorithms required for linear mapping of terrain features. This work is required to allow Corps researchers to use existing vector modeling technologies with preemptive use of a search algorithm that splits a single polygon into multiple lines, arcs, and notes. In earlier efforts, algorithms have been provided by CCSOM/Applied Logic Laboratory to map raster data into optimal polygon topography. In this effort, algorithms are provided for polygon decomposition. In effect, it is a method for dissecting polygons into constitutive elements that more accurately reflect the pattern or feature. Example applications include long linear features whose width is the minor measurement, and whose length is the primary element. More explicit examples for Corps applications include: weirs, locks, dams, roads, and facilities management. This report is primarily in the form of documented computer code required to address this decomposition problem.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Remote Sensing, Statistics, Artificial Intelligence, Neural Networks	23
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		

Conditional Estimation of Vector Patterns in Remote Sensing and GIS
Interim Report 6

R&D 8249-EN-01

Principal investigator: Dr. J.M.F. Masuch

Institution: CCSOM/Applied Logic Laboratory
PCSW - Universiteit van Amsterdam
Sarphatistraat 143
1018 GD AMSTERDAM
The Netherlands
tel: #.31.20.525 28 52
fax: #.31.20.525 28 00
e-mail: ccsoff@ccsom.uva.nl

1. Title

Conditional Estimation of Vector Patterns in Remote Sensing and GIS:
Interim Report 6

2. Abstract

Within this project report we address specific programming requirements for the adaptation of C, C++, and JAVA computer code into a working format for Corps of Engineers modeling in water control, water quality, and emergency response. This effort is cooperatively conducted with the professional researchers at the Remote Sensing / GIS Center of the US Army Cold Regions Research and Engineering Laboratory in Hanover, New Hampshire. The goal of this effort is to program and implement vector algorithms required for linear mapping of terrain features. This work is required to allow Corps researchers to use existing vector modeling technologies with preemptive use of a search algorithm that splits a single polygon into multiple lines, arcs, and notes. In earlier efforts, algorithms have been provided by CCSOM/Applied Logic Laboratory to map raster data into optimal polygon topography. In this effort, algorithms are provided for polygon decomposition. In effect, it is a method for dissecting polygons into constitutive elements that more accurately reflect the pattern or feature. Example applications include long linear features whose width is the minor measurement, and whose length is the primary element. More explicit examples for Corps applications include: weirs, locks, dams, roads, and facilities management. This report is primarily in the form of documented computer code required to address this decomposition problem.

3. Contents

4. Introduction

In this section we enclose the Java elements required for stand-alone execution of the vector run-time environment. The purpose of this code is to create a single control window that will independently operate and process pattern vectors. The code is required to house all main-processing elements. The initial class element is entitled "IntVecttable". This is the initialization sequence required to hold and subsequently recall the vector elements within a sequential table. The code is fully documented using traditional Java global elements.

```
**********************************************************************

public class IntVecttable extends Dictionary implements Cloneable
      {
      /// The vect table data.
      private IntVecttableEntry table[];

      /// The total number of entries in the vect table.
      private int count;

      /// Revectes the table when count exceeds this threshold.
      private int threshold;

      /// The load factor for the vecttable.
      private float loadFactor;
```

A. *Constructs a new, empty vecttable. A default capacity and load factor is used. Note that the vecttable will automatically grow when it gets full.*

```
      public IntVecttable()
      {
      this( 101, 0.75f );
      }
      /// Constructs a new, empty vecttable with the specified initial
      // capacity.
      // @param initialCapacity the initial number of buckets
      public IntVecttable( int initialCapacity )
      {
      this( initialCapacity, 0.75f );
      }
      /// Constructs a new, empty vecttable with the specified initial
      // capacity and the specified load factor.
      // @param initialCapacity the initial number of buckets
      // @param loadFactor a number between 0.0 and 1.0, it defines
      //          the threshold for revecting the vecttable into
      //          a bigger one.
      // @exception IllegalArgumentException If the initial capacity
      // is less than or equal to zero.
```

```
// @exception IllegalArgumentException If the load factor is
// less than or equal to zero.

public IntVecttable( int initialCapacity, float loadFactor )
{
if ( initialCapacity <= 0 || loadFactor <= 0.0 )
    throw new IllegalArgumentException();
this.loadFactor = loadFactor;
table = new IntVecttableEntry[initialCapacity];
threshold = (int) ( initialCapacity * loadFactor );
}
/// Clears the vect table so that it has no more elements in it.
public synchronized void clear()
{
IntVecttableEntry tab[] = table;
for ( int index = tab.length; --index >= 0; )
    tab[index] = null;
count = 0;
}
```

B. *Creates a clone of the vecttable. A shallow copy is made, the keys and elements themselves are NOT cloned. This is a relatively expensive operation. public synchronized Object clone().*

```
{
try
    {
    IntVecttable t = (IntVecttable) super.clone();
    t.table = new IntVecttableEntry[table.length];
    for ( int i = table.length ; i-- > 0 ; )
      t.table[i] = ( table[i] != null ) ?
          (IntVecttableEntry) table[i].clone() : null;
    return t;
    }

catch ( CloneNotSupportedException e)
    {
    // This shouldn't happen, since we are Cloneable.
    throw new InternalError();
    }
}
```

C. *Returns true if the specified object is an element of the vecttable. This operation is more expensive than the containsKey() method.@param value the value that we are looking for @exception NullPointerException If the value being searched for is equal to null. @see IntVecttable#containsKey*

```
public synchronized boolean contains( Object value )
{
if ( value == null )
    throw new NullPointerException();
IntVecttableEntry tab[] = table;
for ( int i = tab.length ; i-- > 0 ; )
```

```
    {
    for ( IntVecttableEntry e = tab[i] ; e != null ; e = e.next )
      {
      if ( e.value.equals( value ) )
          return true;
      }
    }
return false;
}
```

D. *Returns true if the collection contains an element for the key. @param key the key that we are looking for @see IntVecttable#contains public synchronized boolean containsKey(int key).*

```
{
IntVecttableEntry tab[] = table;
int vect = intVectCode( key );
int index = ( vect & 0x7FFFFFFF ) % tab.length;
for ( IntVecttableEntry e = tab[index] ; e != null ; e = e.next )
    {
    if ( e.vect == vect && e.key == key )
      return true;
    }
return false;
}
```

E. *Returns an enumeration of the elements. Use the Enumeration methods on the returned object to fetch the elements sequentially. @see IntVecttable#keys public synchronized Enumeration elements()*

```
{
return new IntVecttableEnumerator( table, false );
}
/// Gets the object associated with the specified key in the
// vecttable.
// @param key the specified key
// @returns the element for the key or null if the key
//          is not defined in the vect table.
// @see IntVecttable#put
public synchronized Object get( int key )

{
IntVecttableEntry tab[] = table;
int vect = intVectCode( key );
int index = ( vect & 0x7FFFFFFF ) % tab.length;
for ( IntVecttableEntry e = tab[index] ; e != null ; e = e.next )
    {
    if ( e.vect == vect && e.key == key )
      return e.value;
    }
return null;
}
```

F. *A get method that takes an Object, for compatibility with java.util.Dictionary. The Object must be an Integer. Public Object get(Object okey)*

```
{
if ( ! ( okey instanceof Integer ) )
    throw new InternalError( "key is not an Integer" );
Integer ikey = (Integer) okey;
int key = ikey.intValue();
return get( key );
}
private int intVectCode( int key )
{
return key;
}
/// Returns true if the vecttable contains no elements.
public boolean isEmpty()
{
return count == 0;
}
/// Returns an enumeration of the vecttable's keys.
// @see IntVecttable#elements
public synchronized Enumeration keys()
{
return new IntVecttableEnumerator( table, true );
}
```

G. *Puts the specified element into the vecttable, using the specified key. The element may be retrieved by doing a get() with the same key.*

// The key and the element cannot be null.
// @param key the specified key in the vecttable
// @param value the specified element
// @exception NullPointerException If the value of the element
// is equal to null.
// @see IntVecttable#get
// @return the old value of the key, or null if it did not have one.public synchronized

```
Object put( int key, Object value )
      {
      // Make sure the value is not null.
      if ( value == null )
          throw new NullPointerException();

      // Makes sure the key is not already in the vecttable.
      IntVecttableEntry tab[] = table;
      int vect = intVectCode( key );
      int index = ( vect & 0x7FFFFFFF ) % tab.length;
      for ( IntVecttableEntry e = tab[index] ; e != null ; e = e.next )
          {
          if ( e.vect == vect && e.key == key )
            {
```

```
        Object old = e.value;
        e.value = value;
        return old;
        }
      }

if ( count >= threshold )
    {
    // Revect the table if the threshold is exceeded.
    revect();
    return put( key, value );
    }

// Creates the new entry.
IntVecttableEntry e = new IntVecttableEntry();
e.vect = vect;
e.key = key;
e.value = value;
e.next = tab[index];
tab[index] = e;
++count;
return null;
}
```

H. *A put method that takes an Object, for compatibility with java.util.Dictionary. The Object must be an Integer.*

```
public Object put( Object okey, Object value )
{
if ( ! ( okey instanceof Integer ) )
    throw new InternalError( "key is not an Integer" );
Integer ikey = (Integer) okey;
int key = ikey.intValue();
return put( key, value );
}
/// Revectes the content of the table into a bigger table.
// This method is called automatically when the vecttable's
// size exceeds the threshold.
protected void revect()

{
int oldCapacity = table.length;
IntVecttableEntry oldTable[] = table;

int newCapacity = oldCapacity * 2 + 1;
IntVecttableEntry newTable[] = new IntVecttableEntry[newCapacity];

threshold = (int) ( newCapacity * loadFactor );
table = newTable;
for ( int i = oldCapacity ; i-- > 0 ; )
    {
    for ( IntVecttableEntry old = oldTable[i] ; old != null ; )
      {
      IntVecttableEntry e = old;
      old = old.next;
      int index = ( e.vect & 0x7FFFFFFF ) % newCapacity;
      e.next = newTable[index];
      newTable[index] = e;
```

```
            }
        }
    }
```

I. Removes the element corresponding to the key. Does nothing if key is not present.

```
    // @param key the key that needs to be removed
    // @return the value of key, or null if the key was not found.
            public synchronized

Object remove( int key )
    {
    IntVecttableEntry tab[] = table;
    int vect = intVectCode( key );
    int index = ( vect & 0x7FFFFFFF ) % tab.length;
    for ( IntVecttableEntry e = tab[index], prev = null ; e != null ;
prev = e, e = e.next )
        {
        if ( e.vect == vect && e.key == key )
          {
          if ( prev != null )
              prev.next = e.next;
          else
              tab[index] = e.next;
          --count;
          return e.value;
          }
        }
    return null;
    }
```

J. A remove method that takes an Object, for compatibility with java.util.Dictionary. The Object must be an Integer.

```
    public Object remove( Object okey )
    {
    if ( ! ( okey instanceof Integer ) )
        throw new InternalError( "key is not an Integer" );
    Integer ikey = (Integer) okey;
    int key = ikey.intValue();
    return remove( key );
    }
    /// Returns the number of elements contained in the vecttable.
    public int size()
    {
    return count;
    }
```

K. Converts to a rather lengthy String in Vector format node to node displacement.

```
        public synchronized String toString()
        {
        int max = size() - 1;
        StringBuffer buf = new StringBuffer();
        Enumeration k = keys();
        Enumeration e = elements();
        buf.append( "{" );

        for ( int i = 0; i <= max; ++i )
            {
            String s1 = k.nextElement().toString();
            String s2 = e.nextElement().toString();
            buf.append( s1 + "=" + s2 );
            if ( i < max )
              buf.append( ", " );
            }
        buf.append( "}" );
        return buf.toString();
        }
}
```

****

5. Table Components

In this section we enclose the Java elements required for holding the vector data *between* process steps. As shown below, the number of elements is surprisingly small given the complexity of specific vector patterns and features. Within the section labeled "protected Object clone()" the main objects are provided. The geometry is stored within the data type "vect" whose entry pointer is a "key". The key maps the data back to the vector topology. The x,y,z position of the data type is stored within the "value" data type, and all subsequent elements are ordered using the "next" sequence. As in earlier efforts, all functions are stand-alone and may be merged with earlier functions previously described in this report.

```
**********************************************************************
package InVector;

import Vectjava.util.*;

class IntVecttableEntry
      {
      int vect;
      int key;
      Object value;
      IntVecttableEntry next;


      protected Object clone()
      {
      IntVecttableEntry entry = new IntVecttableEntry();
      entry.vect = vect;
      entry.key = key;
      entry.value = value;
      entry.next = ( next != null ) ? (IntVecttableEntry) next.clone() :
null;
      return entry;
      }
}
**********************************************************************
```

6. Vector Nodes

In this section we enclose the Java elements required for managing specific nodes within the vector table. The nodes are identified and labeled so that they may be subsequently assigned to a specific polygon. In earlier efforts, all separable features were assigned to a unique polygon. In the "InNode" arrangement, no specific polygon is pre-declared. Rather, nodes are assigned on the basis of their separability. Each element is passed to a specific polygon using the "nextElement" function. Of course, nodes are examined and degenerate features are excluded. This process is shown within the final line using the "NoSuchElementException" function.

```
**********************************************************************
package InNode;

import Vectjava.util.*;

class IntVecttableEnumerator implements Enumeration
      {
      boolean keys;
      int index;
      IntVecttableEntry table[];
      IntVecttableEntry entry;


      IntVecttableEnumerator( IntVecttableEntry table[], boolean keys )
      {
      this.table = table;
      this.keys = keys;
      this.index = table.length;
      }
      public boolean hasMoreElements()
      {
      if ( entry != null )
          return true;
      while ( index-- > 0 )
          if ( ( entry = table[index] ) != null )
            return true;
      return false;
      }
      public Object nextElement()
      {
      if ( entry == null )
          while ( ( index-- > 0 ) && ( ( entry = table[index] ) == null
) )
            ;
      if ( entry != null )
          {
```

```
            IntVecttableEntry e = entry;
            entry = e.next;
            return keys ? new Integer( e.key ) : e.value;
            }
        throw new NoSuchElementException( "IntVecttableEnumerator" );
        }
}
```

```
*****************************************************************
```

7. Calculus Elements

In this section we perform polygon decomposition using a local calculus that computes the likelihood of each node belonging to a specific polygon. This effort is under development and is provided in this interim report for the review of the research team at RS/GISC. As provided below, the calculus is performed using elemental arithmetic. The functions are standard Java elements

```
*****************************************************************

//    reduce x to bidiagonal form, storing the diagonal elements
//    in s and the super-diagonal elements in e

        nct = Math.min(n-1,p);
        nrt = Math.max(0,Math.min(p-2,n));
        lu = Math.max(nct,nrt);

//      if (lu >= 1) {        This test is not necessary under Java.
//                            The loop will be skipped if lu < 1 = the
//                            starting value of 1.

             for (l = 1; l <= lu; l++) {

                  lm1 = l - 1;
                  lp1 = l + 1;
                  if (l <= nct) {

//    compute the transformation for the l-th column and
//    place the l-th diagonal in s[lm1]

                       s[lm1] = Blas_j.colnrm2_j(n-l+1,x,lm1,lm1);
                       if (s[lm1] != 0.0) {

                            if (x[lm1][lm1] != 0.0) s[lm1] =
                            Blas_j.sign_j(s[lm1],x[lm1][lm1]);
                            Blas_j.colscal_j(n-l+1,1.0/s[lm1],x,lm1,lm1);
                            x[lm1][lm1]++;
                       }

                       s[lm1] = -s[lm1];
```

```
                    }
                    for (j = l; j < p; j++) {
                       if ((l <= nct) && (s[lm1] != 0.0)) {

//    If the polgon is unique then apply the transformation

                            t = -Blas_j.coldot_j(n-
                            l+1,x,lm1,lm1,j)/x[lm1][lm1];
                            Blas_j.colaxpy_j(n-l+1,t,x,lm1,lm1,j);

                       }

//    place the l-th row of x into e for the
//    subsequent calculation of the row transformation

                       e[j] = x[lm1][j];
                    }
                    if (wantu && (l <= nct)) {

//    place the transformation in u for subsequent
//    back multiplication

                       for (i = lm1; i < n; i++) {
                            u[i][lm1] = x[i][lm1];
                       }
                    }
                    if (l <= nrt) {
//    compute the l-th row transformation and place the
//    l-th super-diagonal in e[lm1]

                       e[lm1] = Blas_j.dnrm2p_j(p-l,e,l);
                       if (e[lm1] != 0.0) {
                            if (e[l] != 0.0) e[lm1] =
Blas_j.sign_j(e[lm1],e[l]);
                            Blas_j.dscalp_j(p-l,1.0/e[lm1],e,l);
                            e[l]++;
                       }
                       e[lm1] = -e[lm1];
                       if ((lp1 <= n) && (e[lm1] != 0.0)) {

//    apply the transformation

                            for (i = l; i < n; i++) {
                                 work[i] = 0.0;
                            }
                            for (j = l; j < p; j++) {
                                 Blas_j.colvaxpy_j(n-l,e[j],x,work,l,j);
                            }
                            for (j = l; j < p; j++) {
                                 Blas_j.colvraxpy_j(n-l,-
                                 e[j]/e[l],work,x,l,j);
                            }
                       }
                       if (wantv) {

//    place the transformation in v for subsequent
//    back multiplication

                            for (i = l; i < p; i++) {
                                 v[i][lm1] = e[i];
```

```
                        }
                    }
                }
            }

//    set up the final bidiagonal matrix of order m
        m = Math.min(p,n+1);

        if (nct < p) s[nct] = x[nct][nct];
        if (n < m) s[m-1] = 0.0;
        if (nrt+1 < m) e[nrt] = x[nrt][m-1];
        e[m-1] = 0.0;

//    if required, generate u

        if (wantu) {

             for (j = nct; j < ncu; j++) {
                  for (i = 0; i < n; i++) {
                     u[i][j] = 0.0;
                  }
                  u[j][j] = 1.0;
             }

//           if (nct >= 1) {
                  This test is not necessary under Java.
//                The loop will be skipped if nct < 1.
                  for (ll = 1; ll <= nct; ll++) {
                     l = nct - ll + 1;
                     lm1 = l - 1;
                     if (s[lm1] != 0.0) {

                          for (j = l; j < ncu; j++) {
                               t = -Blas_j.coldot_j(n-
                               l+1,u,lm1,lm1,j)/u[lm1][lm1];
                               Blas_j.colaxpy_j(n-l+1,t,u,lm1,lm1,j);
                          }
                          Blas_j.colscal_j(n-l+1,-1.0,u,lm1,lm1);
                          u[lm1][lm1]++;
                          for (i = 0; i < lm1; i++) {
                               u[i][lm1] = 0.0;
                          }
                     } else {
                          for (i = 0; i < n; i++) {
                               u[i][lm1] = 0.0;
                          }
                          u[lm1][lm1] = 1.0;
                     }
                  }
        }

//    if it is required, generate v

        if (wantv) {
             for (ll = 1; ll <= p; ll++) {
                  l = p - ll + 1;
                  lm1 = l - 1;
                  if ((l <= nrt) && (e[lm1] != 0.0)) {
                     for (j = l; j < p; j++) {
```

```
                        t = -Blas_j.coldot_j(p-1,v,1,lm1,j)/v[l][lm1];
                        Blas_j.colaxpy_j(p-1,t,v,1,lm1,j);
                    }
                }
                for (i = 0; i < p; i++) {
                    v[i][lm1] = 0.0;
                }
                v[lm1][lm1] = 1.0;
            }
        }
```

****

8. Special Exceptions

In the previous section, the local calculus operates if the polygon is non-degenerate. When polygons wrap into one-another, significantly overlap, or are otherwise open, singularities may occur. Overflow errors are poorly managed within Java. Unlike C/C++, Java provides a limited number of special exceptions to overflow segmentation. In this section we provide a special procedure that efficiently traps the overflow errors. The small script is used with the local calculus provided in Section 7. The procedure "SVDCException" is used to place the exception, and report the error back to the main script.

****

```
//   main iteration loop for the singular values in degenerate polygons.

        mm = m;
        iter = 0;
        while (true) {

//   quit if all of the singular values have been found
            if (m == 0) return;

//   if too many iterations have been performed,
//   set flag and return

            if (iter >= maxit) {
                throw new SVDCException(m);
            }

/*
```

****

9. Conclusions

Within this interim report, we provide the primary algorithms required for polygon decomposition. These algorithms are the first attempt toward the orderly assignment of nodes, arcs, and line segments in a manner that will allow the Corps to separate objects and features into an optimal set. The problem is quite difficult since polygons vary widely in their form, shape, and texture (edge effect). Long linear features may be trapped using the local calculus provided in Section 7. The feature is tested for validity in Section 8, and then stored (placed) in Section 6 using the features described in Section 4 and Section 5. These algorithms have been tested and applied on simple three band data in the SPOT Image format. The early results are very promising since the algorithms are quite stable and trap the vast majority of singular errors. The remaining issues are primarily centered on two main facets: (1) optimality, and (2) integration. In the former case, one might ask the following questions: How optimal is the decomposition? Can alternative separation functions be tested that separate each feature into a line, arc, node representation that contains fewer elements? In the later case, we will require strong interaction with the research community to determine how this Java approach may be best integrated into the Corps operational models. The current approach is quite flexible, and allows all code to be linked to C/C++, J/J++ systems. In this regard, the methodology is general and easily adapted to specific Corps applications. From the CCSOM perspective, these algorithms are developed on the Sun platform and migrated to the Windows PC platform with limited difficulties.

In the next interim report, we continue this development effort with a specific focus on optimality. This requires specific testing and evaluation of a diverse set of images with patterns and features that vary from simple geometric shapes to complex nested polygons. We intend to work closely with the professional staff of the Corps of Engineers in this effort. The results of this applied R&D will be documented within ERO Interim Report 7.

10. References

Bruce, A.G., Donoho, D.L., Gao, H.Y., and Martin, R.D. (1994). Smoothing and robust wavelet analysis. In: Dutter, R. and Gossmann, W. (1994). *COMPSTAT-Proceedings in Computational Statistics- 11th Symposium held in Vienna, Austria, 1994.* Heidelberg: Physika verlag, 532-547.

Bruce, A. G., and Gao, H-Y. (1996). *Applied wavelet analysis with S-Plus.* New York/Berlin: Springer Verlag.

Daubechies, I. (1988). Orthonormal bases of compactly supported wavelets. *Communications in pure and applied mathematics,* 41, 909-996.

Daubechies, I. (1992). *Ten lectures on wavelets.* CBMS-NSF, Regional Conference Series in Applied Mathematics, 61, Philadelphia (PA): SIAM.

Koornwinder, T.H. (ed.). (1993). *Wavelets: An elementary treatment of theory and applications.* Singapore: World Scientific Publishing Co., Inc.

Meyer, I. (1990). *Ondelettes et operateurs I.* Paris: Hermann.

Murtagh, F., Aussem, A., and Kardaun, O.J.W.F. (1996). The wavelet transform in multivariate data analysis. In: Prat, A. (Ed.) *Proceedings in computational statistics 1996.* Heidelberg: Physica-Verlag. pp 397-402.

Olde Daalhuis, A.B. (1993). Computing with wavelets. In: Koornwinder, T.H. (ed.), (1993), pp 93-105.

Press, W.H. (1991). Wavelet transforms. Harvard-Smithsonian Center for Astrophysics, No. 3184, preprint.

Press, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. (1992). *Numerical recipes in C. The art of scientific computing,* (2nd edition), Chapter 13. Cambridge: Cambridge University Press.

Smith, C., Pyden, N., and Cole, P., (1995). *Erdas field guide. 3rd edition.* Atlanta (GA): ERDAS, Inc.

Strang, G. (1989), Wavelets and dilation equations: a brief introduction, *SIAM Review,* **31** (1989), 614-627.

Appendix Drivers and Related Code Elements

The following drivers and code elements have received minor revisions following the release of Interim Report 5.

```
\*********************************************************************/

OSErr Write_VECTOR (theMap,theReply,creator,fileType)

      XMap        *theMap;                                          (i)
      SFReply     *theReply;
      OSType      creator,fileType;
{

      QDByte      srcBuf[BlockSize],dstBuf[BlockSize];
      (ii)
      QDByte      *curByte;
      Point       sfOrigin;
      OSErr       errCode;
      Ptr         srcPtr,dstPtr;
      long        dstBytes,fSize;
      int         fRefNum,hSize,vSize,hBytes,goodBytes,bit,i,j;

      if (!ValidPointer((Ptr)theMap) || !ValidPointer((Ptr)theReply))
            return(nilHandleErr);
      SetCursor(*(GetCursor(watchCursor)));

      errCode = FSOpen(theReply->fName,theReply->vRefNum,&fRefNum);
      (iii)
      if (errCode == fnfErr || errCode == JAVAfErr ) {
      /* If non-existant, then create it */
            errCode = Create(theReply->fName,theReply-
>vRefNum,creator,fileType);
            if (errCode == noErr)
            errCode = FSOpen(theReply->fName,theReply-
>vRefNum,&fRefNum);
      }
      if (errCode == opWrErr)
            errCode = SetFPos(fRefNum,fsFromStart,(long)0);
      (iv)
      if (errCode == noErr) {
            fSize = HeaderSize;
            for (i=0; i<HeaderSize; i++)
                  dstBuf[i] = 0;
      (v)
            errCode = FSWrite(fRefNum,&fSize,dstBuf);
      }
      if (errCode == noErr) {
      (vi)
            curByte = theMap->baseAddr;
      (vii)
            hBytes = theMap->rowBytes;
      (viii)
            hSize = theMap->bounds.right - theMap->bounds.left;
      (ix)
            vSize = theMap->bounds.bottom - theMap->bounds.top;
      (x)
            goodBytes = hSize / 8;
      (xi)
```

```
            for (j=0; (j<VECTORFileLines) && (errCode == noErr); j++) {
            (xii)
                  for (i=0; i<VECTORRowBytes; i++) {
                        if ((i < hBytes) && (j < vSize)) {
      (xiii)
                              srcBuf[i] = *curByte++;
                              if (i >= goodBytes)
                                    for (bit=15; bit>=0; bit--)
                                          if ((i*8) + (15-bit) > hSize)

      BitClr(&(srcBuf[i]),(long)bit);
                              }
                              else
                                    srcBuf[i] = (QDByte)0;
      (xiv)
                  }
                  srcPtr = srcBuf;
      (xv)
                  dstPtr = dstBuf;
      (xvi)
                  PackBits(&srcPtr,&dstPtr,_VECTORRowBytes);
      (xvii)
                  dstBytes = (long)dstPtr - (long)dstBuf;
      (xviii)
                  errCode = FSWrite(fRefNum,&dstBytes,dstBuf);
      (xix)
                  fSize += dstBytes;
      (xx)
            }
      }
      if (errCode == noErr)
            errCode = SetEOF(fRefNum,fSize);
             /* total bytes including header     */
      if (errCode == noErr)
            errCode = FlushVol(NULL,theReply->vRefNum);

      XTCloseFile(theReply);
      (xxi)
      InitCursor();
      return(errCode);
}
```

Notation (i): Function *Write_VECTOR*. The function exports a single VECTOR compatible image file from a user defined source. A trap has been added to include JAVA script errors.

```
\*********************************************************************/



\*********************************************************************/

void VECTOR_2Local (wPtr,thePt)
      WindowPtr   wPtr;
      Point       *thePt;
      /* Converts '_VECTOR_ture' to local coordinates. */
```

```
      /* Assumes that _VECTOR_ture is associated with window W via its
WData record */

{
      WData       WD;

      if (!ValidPointer((Ptr)thePt))
      (i)
            RETURN;

      GetWData(wPtr,&thePt);
      if (!ValidWData(&thePt) && ValidJAVAObj)
      /* Window has no WData record */
            SetPt(thePt,-32767,-32767);
      /* Default error values */
      else {
            if (ValidHandle((Handle)(thePt.vScrollBar)))
                  thePt->v -= GetCtlValue(thePt.vScrollBar);
            if (ValidHandle((Handle)(thePt.hScrollBar)))
                  thePt->h -= GetCtlValue(thePt.hScrollBar);
      }
}
```

Notation (ii): Function VECTOR_2Local. The function projects VECTOR coordinates to locally defined window system. The ValidJAVAObj test has been included to insure that all valid C/C++ points are *also* stored within the related JAVA library.

```
\*********************************************************************/
```

Annex to

SIXTH Report
Conditional Estimation of Vector Patterns in Remote Sensing and GIS

contract no. N 68171 97 C 9027
contractor: UvA, Applied Logic Labratory/CCSOM
Principal Investigator: Dr. M. Masuch

1. Statement showing amount of unused funds at the end of the covered period

2nd Incrementally Funded Period	total	$ 0.00
3rd Incrementally Funded Period January 2000 August 2000	total	$ 150,000.00
Total unused funds from Nov. 1999 through Aug. 2000		$ 150,000.00

2. List of impotant property acquired with contract funds during this period

none