DEVELOPMENT OF RADAR ALTIMETRY DATA PROCESSING IN THE OCEANIC COASTAL ZONE

ESA/ESRIN Contract No. 21201/08/I-LG – CCN 3 (Phase2)

EWP1 – Deliverable D1.5

Coastal Mask Utility

Scott Gleason

VERSION 1.0, 30 April 2010

National Oceanography Centre, European Way, Southampton, SO14 3ZH, United Kingdom

Tel: +44 (0)23 80596404 Fax: +44 (0)23 80596400 www.noc.ac.uk

Code COASTALT2-D15-10 **Edition** 1.0 **Date** 30 April 2010

Client European Space Agency **Final User** -

	Name	**Signature**	**Date**
Written by	Scott Gleason		30 April 2010

DISTRIBUTION	**Affiliation**
Jérôme Benveniste, Salvatore Dinardo, Bruno Lucas	ESA
Paolo Cipollini, Helen Snaith, Phil Woodworth	NOC
Stefano Vignudelli	CNR
Jesus Gomez-Enri	U Cadiz
Marco Caparrini, Cristina Martin	STARLAB
Joana Fernandes, Alexandra Nunes,	U Porto
Susana Barbosa	U Lisbon

Revision History

Issue	Date	Change
1.0	30 April 2010	Initial Release

TABLE OF CONTENTS

1. What is the Coastal Mask Tool

The Coastal Mask Tool is the combination of a bash shell script which executes a set of commands from the Generic Mapping Tools (GMT) and a custom C program that are used together to generate coastal mask binary files for use in the Coastalt Processor.

1.1. Development Tools Needed

- The Generic Mapping Tools (GMT). This is a open source set of mapping tools widely used in the remote sensing community, available at http://gmt.soest.hawaii.edu/. For the examples shown below the full resolution coast data base was used.

Note: to execute GMT commands the executable must be included in the system PATH variable, as an example for the Mac the following needs to be initialized from the console,

```
export
PATH=/Users/scott/Personal/AppsMac/GMT/GMT4.3.1/bin:$PATH
```

- The GNU Compiler Collection. A publicly available compiler set with is available for all platforms. On Mac OS it is included as part of the Xcode development environment. However, the C program used to generate the final coastal mask could be compiled with any standard C compiler.

2. The GMT bash script

There are 5 GMT commands that need to be executed to generate the two input files needed for the C program. The example below is for the entire globe. A bash shell script is run using the sh command, for example,
sh world_final.sh

First, the data from the GMT coastal data base must be extracted for the desired region and placed in an intermediate file. This is done using the GMT pscoast command as follows (See GMT web site for detailed command breakdowns),

```
pscoast -R0.0/360.0/-80.0/80.0 -JM6i -W -P -B5g5 -Df -M
> world.txt
```

-R indicates the region of the globe. For the projection used her (Mercator) it works better if you cut out the poles. Experimenting with different projections that go all the way to the poles is left as an exercise for users :-).

Parameters: longitude low/longitude high/latitude low/latitude high

-J The map projection. Currently Mercator.

-D the coastal data base used. ((**f**)ull, (**h**)igh, (**i**)ntermediate, (**l**)ow, and (**c**)rude). The better the resolution the slower it runs.

Next, a radius is drawn around every coastal point using the grdmask command as follows,

```
grdmask world.txt -R0.0/360.0/-80.0/80.0 -S200k -I0.1 -
N0/0/1 -M -Gtemp_mask.grd
```

-S This is the diameter of the circle drawn around each point, in this case 200 km.

-I this is the resolution of the grid written to the temp grid file. It sets the resolution of the final coastal mask, in this case 0.1 degrees. Be careful with finer resolutions for it will slow the processing down significantly.

Next, the grd file must be converted to a txt file that the C program can read using the grd2xyz command,

```
grd2xyz temp_mask.grd > temp_mask.xyz
```

Next, A grided land mask must be generated over the same region at the same resolution using the grdlandmask command,

```
grdlandmask -R0.0/360.0/-80.0/80.0 -Df -I0.1 -N0/1/1/1/1 -
Gland_mask.grd
```

-N ocean/land/lake/island/pond flags. In this case, no ocean, but consider everything else in the land map.

Next, the grd file must be converted to a txt file that the C program can read using the grd2xyz command,

```
grd2xyz land_mask.grd > land_mask.xyz
```

Summary: At this point we have two output text file which the C program can read. The first is a 0.1 degree sub-sampled grid with a 200 km circle drawn around every coast point from the full database. The second is a 0.1 degree sub-sampled grid of all land points over the same region. The C program then subtracts the land grid from the "coastal circles" grid and formats the output into a binary map readable by the Coastalt processor.

3. Running the C Program

The C Program “BinaryMap” can be compiled into an executable using the GCC C compiler with the following line,

gcc -o BinaryMap BinaryMap.c

First, The input_parameters.txt file must be modified to include the parameters used to generate the GMT input files above. The format of the input_parameters.txt file is

^A lat low/lat high/lon low/lon high/resolution

For the above example this would be
^A -80 80 0 360 0.1

Next, run the C program with the following line,

./BinaryMap
The coastal mask file will generated in the file **geomask.out**

4. How to Generate Your Own Coastal Mask

- Change pscoast, grdmask and grdlandmask –R parameters to your location (must be square). Experiment with lower resolution using the –D flag.
- Change the coastal distance using the –S parameter in the grdmask command.
- Change the mask resolution using the –I parameter in the grdmask and grdlandmask commands.
- Run the shell script to generate the 2 text files.
- Change the fitting_parameters.txt file to match the latitude and longitude lows and highs and resolution used in GMT.
- The new coastal mask file will be called geomask.out and can be renamed and used by configuring the Coastalt Processor coastal mask input flag.

4.1. Example. Creating a 50 km 0.01 degree coastal mask for Italy

The file italy.sh contains the GMT commands for generating the necessary input files. For this example the GMT script takes a minute or so to finish. The C program takes a couple seconds to run.

- All –R parameters changed to 7/19/35/47
- -S parameter in grdmask command changed to -S50k
- All –I parameters changed to 0.01
- Input_parameters.txt file changed to
  ^A 35 47 7 19 0.01

4.2. Testing the Example. Creating a 50 km 0.01 degree coastal mask for Italy

After the italy.sh script and BinaryMap programs are run, the geomask.out file should contain a 0.01 degree resolution 50 km coastal map between 35 and 47 deg latitude and between 7 and 19 deg longitude.

A test C program has been written to do this, and a simple GMT shell script can be used to plot the coastal mask. The C program is contained in the file test_coastal_mask.c and can be compiled with the line,

```
gcc -o test_coastal_mask test_coastal_mask.c
```

and executed with the line,

```
./test_coastal_mask
```

When you are prompted for "Resolution" this is the resolution the test program scans within the range of the coastal mask for generating a temporary output file used in the GMT plotting script. Normally this is the same as the coastal map resolution but can be finer or coarser if desired (using 0.1 deg generates a less accurate but more manageable ps file to view). The test program will generate the file GMT_xyz_map.txt

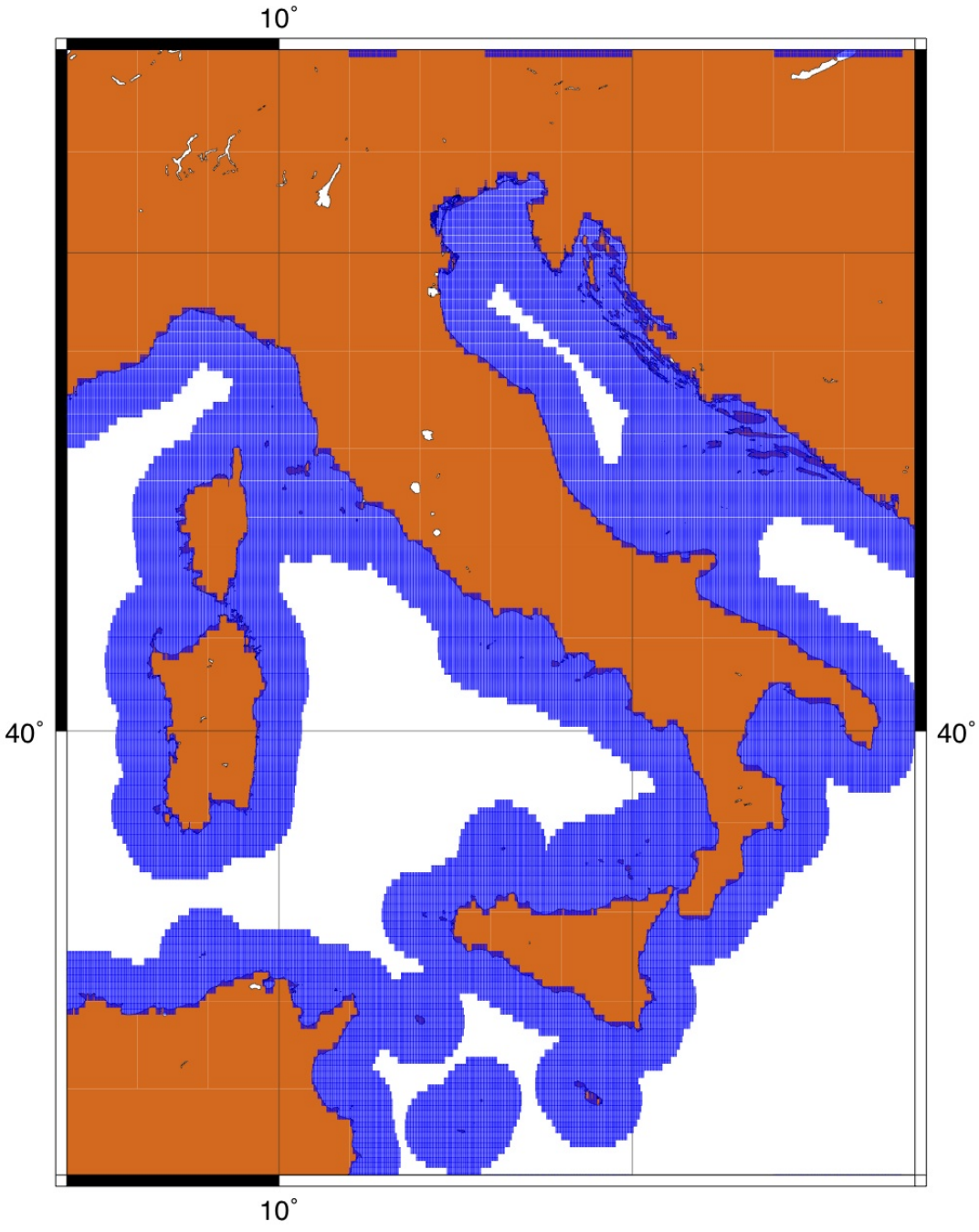
This file is read by the shell script generate_test_italy_map.sh which can be executed with the line,

```
sh generate_test_italy_map.sh
```

The output map is a postscript file called **test.ps**

This will plot the coastal map on a background of the coastal map region. (Italy in this case).

The resulting map is shown below.


10°
40°
40°
10°
GMT 2010 Apr 30 12:08:51
Italy 50kout,1kin coastal map

5. Format of the Coastal Map File

Header:

The first 4 entries in the file are doubles (64 bits) defining the lat/lon box of the map. IN other words, only points within this box will be defined.

Low Latitude (degrees, 8 bytes, 64 bits), range -90 to 90, default -80
High Latitude (degrees, 8 bytes, 64 bits), range -90 to 90, default 80
Low Longitude (degrees, 8 bytes, 64 bits), range 0 to 360, default 0
High Longitude (degrees, 8 bytes, 64 bits) , range 0 to 360, default 360

The next entry is a double defining both the lat and lon resolution

Following are 8 bytes of filler in the pattern 0xa5a5a5a5a5a5a5a5

Resolution (degrees, 8 bytes, 64 bits) , default 0.1

Summary

Parameter	Size (bytes)	default
Low Latitude	8	-80
High Latitude	8	80
Low Longitude	8	0
High Longitude	8	360
Resolution	8	0.1
fill	4	0xa5a5a5a5
fill	4	0xa5a5a5a5

Note: 0 degrees latitude is the equator, 0 degrees longitude is a the greenwich meridian

Total Header bytes = 48

Latitude Longitude Bit Map

0 = point IS NOT processed, 1 = point IS processed

Starting at Low Longitude the entire range of latitudes at the defined resolution is bit packed. The same is done for the next longitude (at a Resolution step) until all longitudes are completed.

To calculate the file offset for a given **latitude** and **longitude** apply the following formula

// size of the header

```
start_offset = 48

// calculate entire latitude and longitude sizes
lat_size = ( High Latitude - High Latitude) / Resolution;
lon_column_size = (High Longitude - Low Longitude) / Resolution;

// calculate relative offsets of the desired lat and lon point
tempd1 = (High Latitude – latitude) / Resolution;
tempd2 = (longitude – Low Longitude) / Resolution;

// sequence based on repeated latitude column
tempd3 = lat_size*tempd2 + tempd1;
byte_offset = (unsigned long) floor(tempd3/8);
tempd4 = fmod(tempd3,8);
bit_offset = (unsigned int) floor(tempd4 + 0.5);

if(bit_offset == 8){
   byte_offset++;
   bit_offset = 0;
}
```

The resulting byte_offset and bit_offset will contain a 1 or 0 corresponding to the latitude and longitude point specified.

```
// index into the file to the appropriate byte
fseek(indata_file,byte_offset+start_offset,SEEK_SET);

// then retrieve the desired bit from the read byte
aa = fread((void*) &tempc, 1, 1, indata_file);

// is the bit in this byte 1 or 0
value = ((1 << (int) bit_offset) && (tempc));
```

[STG – Its probably easier to write a little program that reads an lat,lon text file and sets all the bits in an existing mask]. I have test routines that virtually do this already (i.e. they find the byte and bit but do not change it)]

Example

Low Latitude = 10
High Latitude = 20
Low Longitude = 30
High Longitude = 40
Resolution = 1.0

After the header each bit holds a 2D latitude/longitude pair shown below as (lat,lon)

byte 0 [bits: (10,30), (11,30), (12,30) ... (17,30)]
byte 1 [bits: (18,30), (19,30), (20,30), (10,31) ... (14,31)]
byte 2 [bits: (15,31), (16,31), (17,31), (18,31), (19,31), (20,31), (10,32), (11,32)]