New MapServer Raster Features

Frank Warmerdam
warmerdam@pobox.com
MUM
June 2004

Intro

Focusing on:

- GDAL raster render (from raster input)
- Raster output options.

NOTE: Options being discussed only work via the GDAL renderer

Overview

- RGB to 8-bit via color cube
- Dithering to 8-bit
- OUTPUTFORMAT
- Raw Data Mode
- Image Reprojection
- Non-8bit Image Scaling
- Non-8bit Image Classification
- Raster Query

RGB to 8bit via Color Cube

- Need to support 24bit RGB input images, and 32bit RGBA input images.
- Need to produce 8bit pseudocolored outputs as gif or png.
- Method needs to be **fast**.

Color Cube Cont.

Approach:

- Pre-allocate 175 colors in “ cube” .
- Select nearest color in RGB space.
- 5 levels red / 7 levels green / 5 levels blue

- Preallocate 32 greyscale levels.
- Grey pixels use 32 level greyscale ramp.

Color Cube Cont.

RGB -> Index Calculation (per output pixel):

- 3 integer divisions.
- 2 integer multiplications
- 3 integer additions
- 1 table lookup.

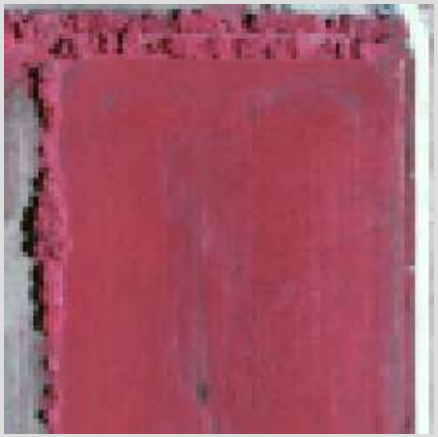
FAST!

Color Cube Cont.

Downside:

– patchy color in slowing changing area

24bit:



color cube:

Color Cube Cont.

Also note:

- Color cube will re-use existing colors.
- Color cube allocation uses hardcoded “ COLOR_MATCH_THRESHOLD” of 1.
- 175+32 (205) is a lot of colors to tie up!
- If color cube allocation fails bad image results.
- No user control over color cube size.

Dithering to 8bit

- Uses same color cube.
- Also uses all other already allocated colors.
- Uses Floyd-Steinberg error diffusion.
- Essentially uses a mix of colors over a region to average out color error.

- Does **not** use optimal color table generation.

Dithering Cont.

Downsides:

- Slower (more computation per pixel)
- Result doesn't compress as well.
- Broken on Windows till early this week!

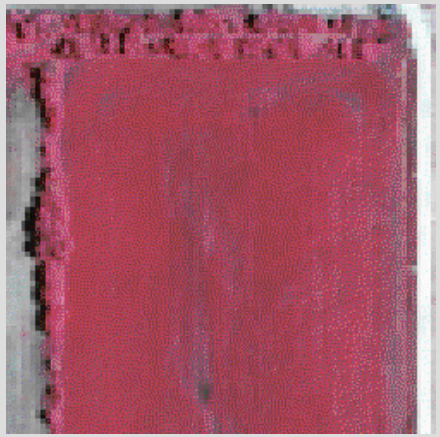
Dithering Cont.

Comparison:

24bit:

Color cube:

Dithered:



OUTPUTFORMAT

Need:

- Flexible control of MapServer output formats
- Ability to control 8bit/24bit/rawmode.
- Ability to select GDAL formats.
- Ability to pass “ options” for output format.
- Ability to provide extra info like mimetype.

OUTPUTFORMAT Cont.

Approach:

- New mapfile OUTPUTFORMAT section.
- Declare many output formats.
- Internally define “ standard” output formats.
- Re-engineer all map/legend/refmap/scale code to use outputFormatObj

OUTPUTFORMAT Cont.

```
OUTPUTFORMAT
    NAME gif
    MIMETYPE "image/gif"
    EXTENSION "gif"
    DRIVER "GD/GIF"
    FORMATOPTION "INTERLACE=ON"
END

OUTPUTFORMAT
    NAME jpeg
    MIMETYPE "image/jpeg"
    EXTENSION "jpg"
    DRIVER "GD/JPEG"
    IMAGEMODE RGB
    FORMATOPTION "QUALITY=80"
END
```

OUTPUTFORMAT Cont.

```
OUTPUTFORMAT
   NAME geotiff_rgb
   MIMETYPE "image/tiff"
   DRIVER "GDAL/GTiff"
   IMAGEMODE RGB
   FORMATOPTION "TILED=YES"
 END

 OUTPUTFORMAT
   NAME geotiff
   MIMETYPE "image/tiff"
   DRIVER "GDAL/GTiff"
   IMAGEMODE PC256
 END
```

OUTPUTFORMAT Cont.

Used for:

- GD formats (gif/png/wbmp/jpeg)
- GDAL formats (ie. GeoTIFF, .img)
- PDF
- Flash (swf)
- Imagemap

OUTPUTFORMAT Cont.

IMAGEMODE:

- PC256: Traditional 256 color mode.
- RGB: 24bit RGB output (ie. Jpeg, png24)
- RGBA: RGB + alpha (transparency)
- BYTE/INT16/FLOAT32: Raw mode...

Raw Data Mode

Original Need:

- Ability to produce 16bit DEM data from MS.

WCS Needs:

- Preserve original image values.
- Support variety of data types (8/16/float)
- Support multispectral (any number of bands)

Raw Data Mode Cont.

Approach:

- New BYTE, INT16, FLOAT32 image modes
- ImageObj extended to hold non-GD image arrays.
- GDAL raster renderer updated to support producing these types.
- Added BAND_COUNT FORMATOPTION to control number of bands generated.

Raw Mode Cont.

Caveats:

- Vector layers not rendered in rawmode.
- Non-GDAL raster drivers don't work.
- Ideally should support additional image types.
- BAND_COUNT not defaulted from the source file. Must be set explicitly if not 1.
- Some work is post MapServer 4.2 for WCS.

Image Reprojection

Need:

- Support for on the fly image reprojection as is done for vector data.

Image Projection Cont.

Approach:

- Added module to reproject an image in GDAL renderer.
- Uses existing GDAL renderer to “ pre-render” higher resolution image in original projection to resample from.

Image Reprojection Cont.

Caveats:

- Bugs in past with transparency (all fixed?)
- Only “ nearest neighbour” resampling.
- Problems estimating the proper input area in “ world mapping” cases.
- Significant performance overhead (doubles render time)
- Doesn't support resample via GCPs or other non-projection models.

Non-8bit Data Scaling

Need:

- Ability to render non-8bit data sources (ie. DEM, 16bit satellite images, science data)
- Ability to control scaling of values to 8bit (0 to 255) flexibly.

Scaling Cont.

Approach:

- Add scaling options to GDAL data loader.
- All scaling done before any render code kicks in
- Render gets 8bit image.
- PROCESSING “ SCALE=<min>,<max>”
- PROCESSING “ SCALE=AUTO”
  - Dynamic stretch based on actual data loaded.

Scaling Cont.

Caveats:

- Renderer doesn't know original values.
- Classification was screwed up.
- OFFSITE doesn't work well.
- NODATA values don't work.
- >256 entry color tables don't work.

Scaling Cont.

Example:

```
LAYER
  NAME raster_l
  TYPE raster
  DATA irvine.pix
  PROCESSING "BANDS=10"
  PROCESSING "SCALE=40,883”
  STATUS default
END
```

Scaling Cont.

Additional Notes:

- For RGB images, scaling can be controlled for each band explicitly using SCALE_1, SCALE_2, and SCALE_3.
- Can be applied to 8 bit images too for constrast stretching!
- If not used for 16bit images, all values over 255 are truncated to 255.

Non-8bit Classification

Need:

- Ability to classify 16bit or floating point raster
- [pixel] should be substituted with the original pixel value.
- To be fast – can't evaluate an expression for each pixel processed!

Classification Cont.

Approach:

- Prescale image data to 16bit (up to 65536 buckets).
- Compute classification result for each bucket.
- Use lookup to apply to image as rendered.
- Provide PROCESSING directives to control scaling.
- For 16bit data no scaling needs to be done.

Classification Cont.

Caveats:

- Floating point data is **still** not exactly classified.
- For floating point data, the .map file needs explicit control of the scaling and buckets.
- If there are a lot of buckets, computing the lookup table may still be expensive.
- NODATA and OFFSITE still don't work well.

Classification Cont.

```
LAYER
  NAME grid1
  TYPE raster
  STATUS default
  DATA data/float.tif
  PROCESSING "NODATA=-6"
  PROCESSING "SCALE=-100.5,100.5"
  PROCESSING "SCALE_BUCKETS=201"
  CLASS
    NAME "red"
    EXPRESSION ([pixel] < -3)
    COLOR 255 0 0
  END
  CLASS
    NAME "green"
    EXPRESSION ([pixel] >= -3 and [pixel] < 3)
    COLOR 0 255 0
  END
  ...
END
```

Raster Query

Need:

- Query support for raster layers.
- Ability to find the real underlying raster value for science data.
- Smooth integration into existing query approach.

Raster Query Cont.

Approach:

- Implement queryByPoint(), queryByRect() and queryByShape() for rasters.
- Query result turned into a set of point shapeObjs available via layer API.
- Each point has attributes for pixel values, class name, and red/green/blue color.
- Point location is center of pixel.
- RASTER_QUERY_MAX_RESULT processing option to limit result set size.

Raster Query

- MapScript can be used to get query results using normal getResult(), getShape() calls
- Templates work normally.
- LAYER needs a TEMPLATE to be query enabled (like vector layers)
- Implemented in MapServer 4.3 (CVS)
- Docs only in Wiki for now.
- Be wary of large result sets!

Raster Query Cont.

```
map = mapscript.Map('rquery.map')
layer = map.getLayer(0)
pnt = mapscript.Point()
pnt.x = 440780
pnt.y = 3751260

layer.queryByPoint(map,pnt,mapscript.MS_MULTIPLE,180.0)
layer.open()
for i in range(1000):
    result = layer.getResult( i )
    if result is None:
        break

    s = layer.getShape(result.shapeindex,result.tileindex)
    for i in range(layer.numitems):
        print '%s: %s' % (layer.getItem(i), s.getValue(i))

layer.close()
```

Raster Query Cont.

Template File:

```
Pixel:<br>
  values=[values]<br>
  value_0=[value_0]<br>
  value_1=[value_1]<br>
  value_2=[value_2]<br>
  RGB = [red],[green],[blue]<p>
  Class = [class]<br>
```