9B.9 CHALLENGES OF DISPLAYING DYNAMIC WEATHER CONTENT IN INTERACTIVE MAPPING SOLUTIONS

Scott Ganson[1] and JT Johnson[1]
[1]Weather Decision Technologies, Norman, Oklahoma

1. Introduction

In recent years, interactive mapping such as Google Maps/Earth, Microsoft Virtual Earth and numerous others have become increasingly popular. The idea of combining various data with these mapping solutions has created a "buzz" in the Internet world as developers create new dynamic and immersive views of their world around them. A natural dataset to combine with relevant location information is weather.

However, this combination does not come without challenges. Much, if not all, of the weather information that enthusiasts have sought for years on the Internet has been in text, simple image, or complex binary formats. To create weather content for the interactive mapping world that is both informative and intriguing the data often needs to have both high temporal and spatial resolution as well as be dynamic and intuitive.

Thus, the first challenge: Overcoming large amounts of data transfer. Most mapping platforms have opened tunnels into their infrastructure via web-based API's. However, these API's do not yet offer the flexibility and efficiency necessary for displaying highly dynamic and voluminous content.

The second challenge is overcoming typically "clunky" meteorological formats and placing them in a typical browser-based format that can be quickly transferred without sacrificing data integrity.

Lastly, a final challenge to overcome is using multiple meteorological datasets to create interactive map products which are meaningful to users. No longer does just a map of color-filled temperature contours mean as much to people. In the Web 2.0 world, it is necessary to assist in everyday decision making processes for the general public. We will specifically address these challenges throughout this paper.

2. Challenges

a. Large dataset transfers

As most in the meteorological community know, large datasets are often necessary to get a clear picture of atmospheric conditions at any given time or location. Examples of these types of datasets include point data, gridded data, or raster data. Specifically, the amount of data in each of these formats presents a developer with very different yet challenging dilemmas.

* Corresponding author address: Scott Ganson, Weather Decision Technologies, 3100 Monitor Ave. Suite 280, Norman, OK 73072, sganson@wdtinc.com

One of the most prevalent issues that will immediately grind production to a halt is limited bandwidth and its expense. Therefore, it becomes necessary in most production environments to make the datasets manageable for file transfer without affecting the integrity of the original dataset.

Although there are numerous approaches to solve this problem, this paper will only address a few of them. The first approach, known as "progressive disclosure," "allows one to plot a large number of points (e.g. there are as many as 2,500+ [US] METARS reporting in a one-hour cycle) without cluttering up the large-scale map. " [3]. As this definition implies, the benefits to progressive disclosure also help the user. The question still remains as to how this process can help limit bandwidth and keep data transfer to a minimum in a production environment. By using a combination of progressive disclosure along with Web 2.0 techniques such as Asynchronous JavaScript And Xml (AJAX) [1] and JavaScript Object Notation (JSON) [2], a developer can build an application to use a client's geo-referenced information (such as a location) to develop a location-based query which only returns a small subset of data relevant to a user's specific location at a given time. By doing this, the developer creates the illusion of having a complete dataset loaded on a client's machine by transferring small "chunks" of data based on client side events inside a dynamic mapping application.

Another useful technique is a tiled solution, however this technique is more complex and difficult to implement. One huge advantage to tiled data transfer is that tiles are the backbone to mapping solutions available on the Internet. Tiles are used to piece together, one small chunk at a time, a larger image which is seamlessly composed of the smaller tiles. Therefore, to tile meteorological data is a natural extension of the mapping technologies.

A tiled solution is best used with raster or even gridded data such as radar, satellite or weather model data. One relatively simple tiling method is an "on-demand" approach, where tiles are created from a "master" data set on the fly; built upon a location-based query from a client-side application. Another tiling method is pre-processing the tiles to be served. This method can be computationally quite a bit more expensive, and very hard to accomplish with highly dynamic data.

The pre-processing tiling method tends to lend itself more logically to lower temporal resolution data (i.e. climate data). This is mainly due to the fact that depending on how zoomed in on the data the mapping solution allows, many

tiles may have to be created. In fact, the number of tiles is a power curve. For example, at the highest (most zoomed OUT level) only 1 tile may be necessary. However, as you zoom in to the next level that number increases whereby the total image is composed of 2x2 tiles, or 4 total tiles. Then as you zoom in to the next level that number again is doubled to 4x4, and zooming again yields 8x8. It can quickly be seen how the limitations to pre-processing the tile generation process can yield itself an impossible solution. If one has to make a tiled map which consists of 16384x16384 tiles, that becomes a huge computing resource drain to recreate every hour or worse yet every 5 minutes for something as dynamic as radar images.

zoom level	dimensions	total number of tiles
1	1	1
2	2	4
3	4	16
4	8	64
5	16	256
6	32	1024
7	64	4096
8	128	16384
9	256	65536
10	512	262144
11	1024	1048576
12	2048	4194304
13	4096	16777216
14	8192	67108864
15	16384	268435456
16	32768	1073741824
17	65536	4294967296

Table 1. Example of the number of tiles derived by using a tiling method.

However, as computing becomes cheaper, there is obviously room for this solution to become more viable for highly dynamic datasets.

On-demand tiling solutions are a logical approach to datasets which do not lend themselves as easily to a pre-generated solution due to the fact that computing resources are only used when necessary (i.e. requested by the user). Most mapping API's will allow a developer to point to specific URL's using a Cartesian coordinate system in the x,y directions coupled with a zoom level coordinate (which acts similar to a z-axis) and overlay the image found at the target URL onto the map. Using dynamic page generation such as PHP, CGI, or Ruby can allow a developer the opportunity to create the tiles on the fly using the information passed to the dynamic script as URL arguments. To further enhance the end-user experience, another process can be added to disk-cache the generated tiles for faster repetitive retrieval by multiple users in rapid succession.

Extending mapping solutions to include meteorological raster data can yield such results as pictured in Figure 2. These tiling solutions are extremely effective in reducing overhead associated with bandwidth due to the fact that the resulting tiles transferred are generally on the order of ~10KB instead of > 1MB for a large high resolution image which would produce the same result at high zoom levels. The other added benefit to tiles is that the end-user experience again is greatly enhanced due to the loading of numerous bits of small data chunks opposed to waiting on one large dataset of which the user may only interrogate or look at 10% of the total amount of data transferred.

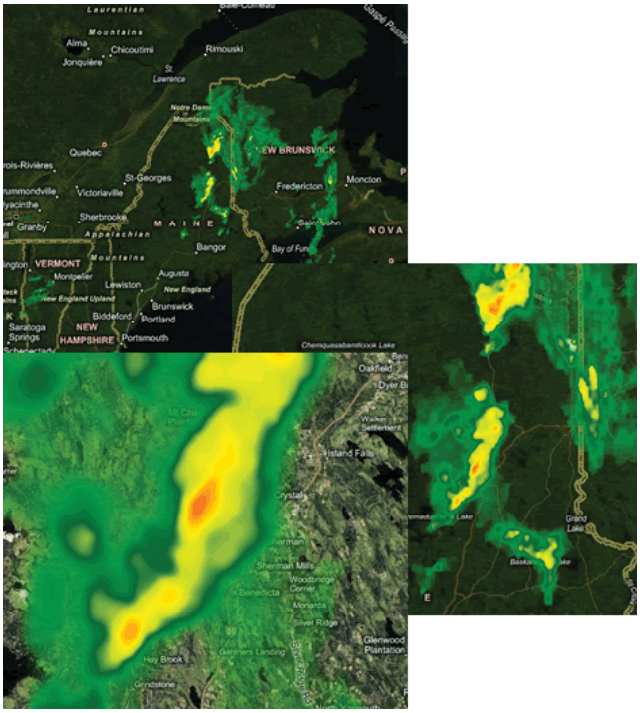

Figure 2. Example of tiled data at progressively zoomed in levels.

b. File format issues

Another challenge facing developers of client-side applications which utilize these mapping solutions is choosing a file format to transfer data to the application from the server. The issue is that most native meteorological formats such as Grib files, flat text files, NIDS, NetCDF, etc. , are either not browser compatible, or bloated with information not necessary for the end-user application to function. This leaves the developer with a decision to make. The first being that the programmer could write a decoder for the application which takes the raw meteorological format, decodes

it, and displays it in a manner which is understandable by the user. Secondly, the developer could make use of existing technologies such as SQL, XML, object notation, etc, and do pre-processing to massage the dataset into these formats. Or finally, the dataset could be manipulated into a proprietary format either created by the developer or institution which has created the client-side application.

The feasibility of writing a decoder for a client-side application seems at first logical. However, upon further examination there seems to be compelling evidence to the contrary. To address one reason why this may not be possible is to look at the platform on which the application is being developed. There exists a very large difference between a client-side application which is built upon operating system framework compiled from a low-level language such as C, C++, or others and an application which is designed to use browser-based technologies. While the former has a scope which is completely able to lend itself to handling standard meteorological formats, the latter is much better suited to markup languages, flat text files, or simple binary formats. To design a browser-based application to use input data such as NIDS or netCDF would be a tremendous undertaking and could possibly rapidly drain resources on a client-side computer when in use. Therefore this paper will focus on the challenges a developer may face when designing these browser-based applications.

To have a responsive application which is both intuitive and a positive end-user experience, it is very important to first and foremost reduce to a bare minimum the amount of time a user spends loading data into the application. This necessitates the use of small manageable file formats. With the exception of images, this thought process can be applied to most meteorological data sets. The mapping platforms which have been mentioned, with the exception of Google Earth which uses Keyhole Markup Language (KML) format, have API's which are available in JavaScript (a standard browser based scripting language). Therefore, it becomes important to have a method to actually utilize the file format which is chosen. The options which have been discussed are limited to SQL, XML, and object notation, which in JavaScript, can be shortened to JSON.

The most complicated of these "formats" is to use SQL or Standard Query Language in conjunction with XML or JSON. Unfortunately, this method cannot be done solely with JavaScript. This procedure uses a combination of JavaScript and a server-side scripting language such as PHP. The developer can use a JavaScript application to pass HTTP request variables to a PHP script to define and/or refine search criteria for returning information which is applicable to the end-user. The server-side script can then decode the result of a SQL query either directly into JavaScript commands, or better yet into a format which can be used natively by JavaScript. This trick is widely used in the Web 2.0 world as a convenient way of data transfer without performing a "hard refresh" which causes an entire web page to reload.

Similar to the tile serving technique, it is entirely possible to pre-process the above steps by creating, for example, an XML file which contains all the latest surface observations in the CONUS. Of course, this technique is not without its downside. The tradeoff here, is the fact that this entire file would need to be transferred to the client computer which takes up valuable loading time and bandwidth.

KML format offers an interesting inspiration for trying to find a happy medium for this problem. The idea to break apart one large file into geographic regions could greatly increase user-performance while decreasing bandwidth usage. By breaking the file into regions, the application could easy download the relevant XML file with data pertaining to the location the user is currently interrogating in the application.

The last format which needs to be discussed is objection notation, or more commonly in JavaScript referred to as JSON (JavaScript Object Notation). JSON is an interchange format, much like XML except it has the advantage of being what developers call "lightweight." This means there is not a lot of bloating of the file through syntax rules as there can be with XML. By treating SQL output as "objects," a developer can utilize JSON to create a file of variables or expressions which can be used directly in JavaScript applications without the overhead of parsing. Below is an example of a standard ASOS observation and how one could easily fit it into a JSON-like syntax.

```
var currentsData =
{
    id: 1,
    conditions:
    [
      {
        stn: 'KOUN',
        datetime: '2008-01-09 21:13:00',
        temp: '57.2',
        dewp: '28.4',
        lat: '35.217',
        lon: '-97.45',
        name: 'Norman/Westheimer',
        state: 'OK',
        country: 'US',
        dir: '170',
        dir_eng: 'S',
        speed_mps: '8.2',
        gust_mps: '11.3',
        slp_mb: '',
        weather: '',
        cld_cover: 'Clear'
      },
    ]
  };
```

This is just one simple example of what can be done with this format. Most meteorological data, with the exception of raster data, is easily massaged into this format for simple transfer and use by browser-based applications.

c. Meaningful products

While the intent of this paper is to discuss technical challenges associated with interactive mapping applications, it is also important to emphasize that these applications are no longer going to survive as just simple weather maps like those found in standard newspapers. In today's world, people are looking to the Internet not only for information, but decision support, risk management, and increased efficiency in daily activities. That being said, the challenge to us as meteorologists is to create content that does more than just deliver information.

The first way one can do this is to attempt to have applications "tell a story." Meaning, "How does the weather affect people and the daily decisions they make?" Desire for applications to lend themselves to this kind of information is increasingly evident in the popularity of social networking and location based services. Tying weather to these kinds of information sources is a logical next step, as well as an important one in continuing to protect the life and property of those around us.

3. WDT's Solution

One major area of work done by WDT has been to take these challenges head on, overcome them, and successfully create applications which utilize this new area of technology. In recent months, work to create a radar "Tile Server" system has recently reached its beta phase. As previously mentioned, the main challenge with tiling technology is the limitations of computing power. However, WDT has managed to overcome this hurdle and prove not only the concept, but the successful fabrication of an end-user application based upon the technology.

In order to accomplish this task, WDT needed to develop a way to have tiles, in mass quantities, be all but immediately available to a requesting application. To do this, it was decided that an on-demand approach would have too much lag due to reasons involving Input of a CONUS NetCDF file and the output of a high-resolution much smaller domain PNG image. Due to the fact that reading the NetCDF file was computationally intensive, disk monopolizing, and unable to be memory cached for simultaneous access by multiple tiling applications, it meant that repeating this process every time a tile was requested would be impossible. To further add difficulty to the process was the issue that writing PNG images was computationally expensive due to compression algorithms which are inherent in the image format. Therefore, yielding on-demand tile generation not a viable option for this particular application.

Due to these complications the decision was made to head in a direction of pre-generation of tiles. This allowed for one nation-wide NetCDF file to be read just once for all the tiles to be created; solving one of the main challenges to tile generation. The next compelling reason to use pre-generation was because this would allow tiles to sit on the disk in a format which was easily interpreted by the browser, thus making load time of the tile relatively small. However, the main challenge now associated with pre-generation was first, raw computational power, and second disk I/O.

To overcome computational challenges, new library extensions were written with brand new mathematical classes which executed as close to bit-level as possible. By doing this, the developers were able to cut down on much of the overhead of using standard development libraries. A next step in the process was to analyze the entire NetCDF file once and flag only areas in the file which would need to be further broken down into tiles. This process was done by looking at a CONUS composite reflectivity mosaic and only flagging areas with echoes and not areas of no data. To determine if this step was actually a sound computational investment a case study was done on the Enterprise, AL case on March, 1 2007 at 2000Z. When the study was completed, it was found that the results were staggering. Not only was it more than necessary to analyze the data prior to actually beginning the tiling process, the pre-generation scheme itself would fail without it due to reasons which will be discussed later in the paper.

# of Zooms	Tile size (in pixels)	# of Before Analyzing	# of Tiles after Analyzing
1	1024	128	15
2	512	512	61
3	256	2,048	246
4	128	8,192	983
5	64	32,768	3,932
6	32	131,072	15,729
7	16	524,288	62,915
8	8	2,097,152	251,658
9	4	8,388,608	1,006,633

Table 2. Comparison of the number of tiles at a given zoom level prior to and after analyzing a NetCDF file.

After completing the study on a case with widespread convection over the Eastern U.S. as well as large areas of stratiform precipitation in the Northwestern U.S., it was found that on a case as "bad" as the Enterprise case, roughly only 12% of the CONUS domain was actually found to have some sort of echo associated with it. This

allowed the amount of tiles that needed to be created to be greatly reduced.

The next challenge associated with pre-generation was to actually complete the process in <5 minutes, or the average time that WDT creates a new nation-wide composite mosaic. This posed a whole new set of hurdles to overcome. Physically, a single disk at 15,000 rpm could not write the 8,388,608 tiles necessary in less than 5 minutes. Therefore, a new research area was spawned. How can WDT mimic ~8 million tiles with significantly less I/O?

Inspiration for solving this dilemma was found in compression algorithms for representing large files with the fewest bytes possible. By breaking down the entire NetCDF file into the core components which comprise it, (in this case the unique reflectivity values in the file), it was found that all of the roughly 8 million tiles could be represented by a relatively small number of "building blocks." By only creating unique tiles and fabricating them together to re-create a much larger picture, the team discovered that the entire NetCDF file could actually be generated using a significantly smaller number of tiles than originally expected, as Table 3 illustrates.

# of Zooms	Tile size (in pixels)	# of Before Analyzing	# of Tiles after Analyzing	WDT Compression algorithm at ~7%
1	1024	128	15	1
2	512	512	61	4
3	256	2,048	246	17
4	128	8,192	983	69
5	64	32,768	3,932	275
6	32	131,072	15,729	1,101
7	16	524,288	62,915	4,404
8	8	2,097,152	251,658	17,616
9	4	8,388,608	1,006,633	70,464

Table 3. Example of the amount of tiles required after WDT compression algorithm is applied.

As the Table 3 shows, on average, WDT was able to further lower the number of tiles by a staggering 93% by using this process. This now allowed the tile generation process to successfully be able to write the adequate number of tiles to disk while leaving repeated tiles or blank tiles to be all pointed to a representative tile that was already generated. Thus leaving the opportunity to develop an end-user application which utilized the mapping solutions freely available as well as combining WDT's now pre-generated radar tiles for an un-paralleled end-user experience.

4. Future Work

An additional, but anticipated issue, which has been encountered in the tiling of gridded data such as NetCDF data, is resolution of the data at extremely high zoom levels. Performing a zoom to this degree can leave the data appearing "pixilated" (Figure 3) instead of smooth and pleasing to the eye.

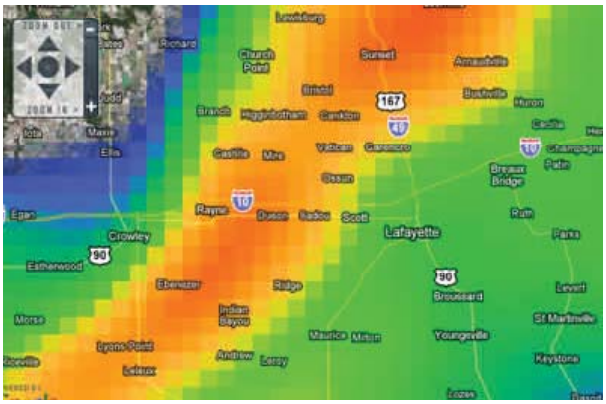

Figure 3. Pixelized radar image due to gridded data.

For that reason, WDT's future mission which is already underway is to represent gridded radar data such as NetCDF and convert it into vector data, then perform a process similar to what was detailed in the Section 3. By converting the grid into a vectorized form, it gives the user the ability to zoom infinitely far in to a point and not see pixelization because data is filled via linear interpolation in between points in the vectored data. An example of a hypothetical result from this process is seen in Figure 4.

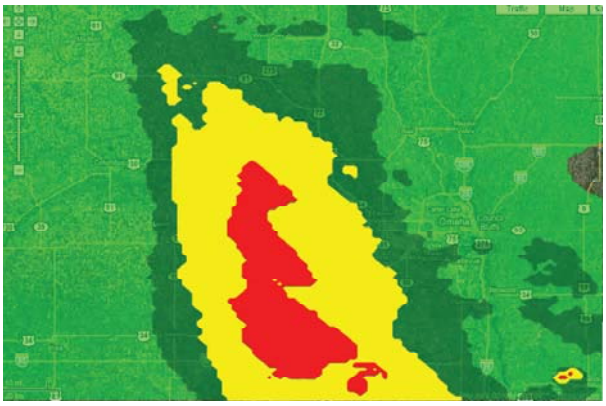

Figure 4. Hypothetical vectorized radar image.

5. Summary

In reality, there are an infinite number of ways which meteorological data can be visualized. While this paper explains methods which are operationally used, it does not necessarily mean better ways do not exist. The main

goals of browser-based applications, regardless of how the goals are accomplished should be:

1. Ensure necessary information in conveyed to the end-user in an intuitive manner.
2. Create a positive end-user experience by utilizing techniques to minimize load time and increase application responsiveness.
3. Assist in decision making and not just information disclosure.

It is the mission of the meteorological software developer to meet the challenges discussed in the paper and to overcome them. While only a few solutions have been introduced to some very broad challenges, there are still a limitless number of both which will exist as datasets become larger, higher in temporal and spatial resolution, and the users of such data become more connected and widespread.

6. References

1. AJAX: For information:
   http://www.w3schools.com/ajax/default.asp

2. JSON: For information:
   http://json.org/

3. Roberts, William F., Nicholas D. Gebauer, and Leigh K. Cheatwood, 2007,A ONENOAA CONCEPT PROTOTYPE FOR DATA VIEWING NOAA RESEACH 23th Int. Conf. on Interactive Information Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology, San Antonio,TX, Amer. Meteor. Soc., 9A.4.