A New Vector Data Compression Approach for WebGIS

LI Yunjin, ZHONG Ershun
Institute of Geographic Sciences and Natural Resources Research, Chinese Academy of Sciences, 11A Datun Road, Chaoyang District, Beijing 100101, China

© Wuhan University and Springer-Verlag Berlin Heidelberg 2011

Abstract High compression ratio, high decoding performance, and progressive data transmission are the most important requirements of vector data compression algorithms for WebGIS. To meet these requirements, we present a new compression approach. This paper begins with the generation of multiscale data by converting float coordinates to integer coordinates. It is proved that the distance between the converted point and the original point on screen is within 2 pixels, and therefore, our approach is suitable for the visualization of vector data on the client side. Integer coordinates are passed to an Integer Wavelet Transformer, and the high-frequency coefficients produced by the transformer are encoded by Canonical Huffman codes. The experimental results on river data and road data demonstrate the effectiveness of the proposed approach: compression ratio can reach 10% for river data and 20% for road data, respectively. We conclude that more attention needs be paid to correlation between curves that contain a few points.

Keywords vector data compression; WebGIS; progressive data transmission

CLC number P208

Introduction

Mapping is time-consuming, and it is carried out by the server side in traditional WebGIS.[1] Raster maps, which are divided into tiles, are often generated in advance to reduce the response time of map service. However, if we change the style of the map, the old tiles cannot be used anymore. Although traditional WebGIS is simple and efficient, it lacks flexibility. Vector data can be rendered effectively nowadays by the client-side using Rich Internet Application techniques, such as SilverLight and Flex. However, delivering large volumes of spatial datasets over narrow bandwidths is difficult. There are at least two solutions to overcome this difficulty: to deliver vector datasets at increasing levels of detail from the server side to the client side and to compress the vector datasets before they are delivered.

The compression of vector data can be achieved through polygonal approximation or through the quantization of coordinates. Douglas-Peucker algorithm[2] is often used to approximate curves. However, it fails to keep topology consistent, and it is a suboptimal algorithm. To obtain an optimal result, many methods have been introduced,[3-6] and they fall into three categories: min-# method, which finds the minimal number of vertices under specified distortion; min-rate method, which finds the minimal bit rate solution under specified distortion; and min-ε method, which finds the minimal distortion under a specified count of points. Both min-# and min-rate can be
found by determining the shortest path in a weighted graph. Vertices are connected only if the approximation error is within the given threshold. It takes \(O(n^3)\) time to build the weighted graph. Subsequently, Chan and Chen proposed an \(O(n^2)\) algorithm.\(^3\) The min-\(\epsilon\) problem has optimal substructure and overlapping subproblems, and thus, we can use a dynamic programming algorithm to find the optimal solution. To reduce the calculation, Perez and Vidal proposed the incremental form of two error measures.\(^6\) They constructed the weighted graph in \(O(mn^2)\) time, where \(m\) stands for the count of approximate edges. In some research studies,\(^4, 5\) time complexity has been reduced to \(O(w^2n^2/m)\), where \(w\) stands for search bandwidth.

Fibonacci, Huffman, Markov (FHM)\(^7\) is a chain coding-based\(^8\) algorithm designed for signature compression. Given a signature, the FHM algorithm uses a dictionary of line segments with fixed slopes and lengths. However, the dictionary used in FHM is static and fails to consider the distribution of data. Shekhar et al.\(^9\) used the K-means approach to build a dynamic dictionary, and the distortion was controlled by the count of clusters. Yang et al.\(^10\) built a dynamic dictionary through TIN generalization, and the distortion depended on the distance threshold.

The selection of compression method depends not only on the data type but also on the software and hardware environment. We focus our research on the application in WebGIS with the following features:

1) Vector data is delivered in a progressive way. Rough maps, which are generated through map generalization, are transmitted first and then are refined in the following steps.

2) Decompression efficiency is critical to users. Vector data is compressed on the server side and then transmitted to the client side. Compressed vector data is decompressed on the client side and then rendered dynamically.

A new method is proposed based on the above considerations. First, we divide the original dataset into subdatasets with different view scales, and at the same time, the coordinate components are converted from float to integer. We then apply Integer Wavelet Transform (IWT) and Canonical Huffman codes to the converted coordinates.

## 1 Type conversion

Single-precision and double-precision float data types are used to store geo-coordinates, and thus, each geo-coordinate component of a point requires four or even eight bytes of storage. However, higher data precision does not ensure a better rendering quality. Using an integer or a short type instead of a float type can reduce data volume. Type conversion is similar to the process of rendering vector data because both need to translate, scale, and round geo-coordinates. Although the loss of precision will be brought by type conversion, the loss may have little influence on the rendering quality. For example, if the size of view window is 1024×768 and we use a 10-bit integer to represent one component of a point, the rendering result based on the integer data is equal to that based on float point data.

Choosing a mapping range is the key to type conversion. Factors affecting range selection include boundary of original data, view scale, and screen resolution. Let \(dx\) denote the difference between the maximum and minimal value of \(x\), and let \(dy\) denote the difference between the maximum and minimal value of \(y\). \(s\) stands for view scale, and \(r\) stands for screen resolution of which the unit is dots per inch (dpi). As one inch is equal to 2.54 cm, the mapping range can be calculated as follows:

\[
\text{range} = \text{max}(dx, dy) \times s \times 100 / 2.54 \times r
\]

where the max function returns the maximum value of \(dx\) and \(dy\), and one integer coordinate component costs \(I\) bits of storage:

\[
I = \left\lceil \log_2 \text{range} \right\rceil
\]

The rendering result based on \(I\)-bits integer coordinates is the same as the rendering result based on the original data. For instance, if \(dx\) is 5200 km, \(dy\) is 5500 km, \(s\) is 1:4000000, and \(r\) is equal to 96 dpi, we can determine that the bit length of an integer coordinate component is 13.

Let \((x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}})\) be the minimal bounding box of the original data. The ratio of integer coordinate range and float coordinate range is then

\[
\text{ratio} = 2^I / \text{max}(x_{\text{max}} - x_{\text{min}}, y_{\text{max}} - y_{\text{min}})
\]

We can now convert the type from float to integer
as follows:

\[
\begin{align*}
    x' &= \left( x - x_{\min} \right) \times \text{ratio} \\
    y' &= \left( y - y_{\min} \right) \times \text{ratio}
\end{align*}
\]  

(4)

Two or more adjacent points may overlap after type conversion, and these points satisfy

\[x'_{i+1} = x'_{i}, \quad y'_{i+1} = y'_{i} \]

(5)

Removing these points will not affect the rendering result.

According to Eqs. (1) and (2), the bit length of the integer coordinate component will increase along with the increase of the three related factors. However, the values of the three factors will not increase unrestrained. As a short integer needs 16 bits to store in most computers and is easy to be dealt with, we use a 16-bit integer to represent one integer coordinate component. If the theoretical length calculated by Eq. (1) is greater than 16, a division of the original data is required. For example, dividing the original data with 2×2 grids will decrease the range of the coordinate component to 1/2 of the original data, hence reducing the length by 1 bit.

2 Progressive data model of curves

Fixed scales model is used frequently in map browsing. Data sources of the model can be classified into two types: data with different resolution and data generated by map generalization or interpolation algorithms. Douglas-Peucker, a typical map generalization algorithm, can be used to generate progressive data. It selects points that satisfy the distance requirement and deletes others. Note that the point selection of this algorithm does not take the view scale into account. Using different distance thresholds to represent different view scales is a possible solution. Type conversion can also be used to generate multiscale data. First, we calculate the bit length of the integer coordinate component according to Eqs. (1) and (2) and then convert the original data to different versions with different bit lengths. This process is simple but leads to waste of storage because of the large amount of redundancy among the data with different scales. Moreover, the bit lengths of the different levels are different, thus causing difficulties in the subsequent data processing.

2.1 Progressive data generation algorithm

Storing and transmitting vector data in a progressive way will reduce not only the storage cost but also the waiting time of users. A curve is divided into two parts: approximate curve and supplement points. Approximate curve is a coarse version of the original curve. Supplement points are the complement of points on the approximate curve. Approximate curves are transmitted at the minimal view scale, whereas the supplement points are transmitted with the increment of view scale. The approximate curve and supplement points are constructed as follows:

Step 1 Calculate the mapping range and length of one integer coordinate component according to the maximum view scale and convert the type of coordinate components to integer. Remove the redundant points overlapping the neighbor points.

Step 2 Transform the data generated in Step 1 to the data with the minimal view scale. All the redundant points under the minimal view scale belong to the supplement point set. The remaining points belong to the approximate curve. Record the curve ID and the index of point. Set the minimal view scale to −1 for each supplement point.

Step 3 For each scale between the maximum and the minimal view scale, do the following:

For each point in the supplement point set whose minimal view scale is equal to −1, check whether this point overlaps with the previous point under the current view scale. If the point does not overlap the previous point, set the minimal view scale of the point to the current view scale.

End.

Each curve is represented as an approximate curve plus supplement points through the above steps. A record of a supplement point consists of a coordinate, a curve ID, a node index, and a minimal view scale. The coordinate of points are calculated under the maximum view scale. If the current view scale is equal to \( s_{\text{cur}} \) (greater than the minimal view scale), data to be transmitted should include the approximate curves and the supplement points whose minimal view scales are smaller than \( s_{\text{cur}} \). The curve with maximum view scale will be reconstructed and converted to the current view scale in the further steps.
The supplement points of the different curves can be organized by the minimal view scale. All the supplement points with the same minimal view scale belong to the same subdataset. While discrete points in subdatasets can be compressed by offsetting and encoding, the approximate curve can be compressed by IWT, which will be discussed in Section 3.

2.2 Mapping errors

The process of type conversion has the same effect as the process of rendering vector data when they are processed at a maximum view scale, thus the rendering results of the original data and the converted data are identical. However, when the view scale is smaller than the maximum view scale, the rendering results may be different. We define the distance from the original point to its converted point on screen as the mapping error, which is denoted by \( \Delta \).

Apparently, the point obtained from the inverse transform can be represented by Eqs. (9) and (10),

\[
\begin{align*}
\Delta x &= \frac{1}{2} \left[ (x-x_{\text{min}}) \times \text{ratio} \right] / b^r - \left[ (x-x_{\text{min}}) \times \text{ratio} / b^r \right] \\
\Delta y &= \frac{1}{2} \left[ (y-y_{\text{min}}) \times \text{ratio} \right] / b^r - \left[ (y-y_{\text{min}}) \times \text{ratio} / b^r \right]
\end{align*}
\]

where \( b^r \) is the zoom ratio, where \( b > 1 \), and \( n \) denote the times of zoom operation from the maximum view scale.

If \( b \) is an integer, we define \( A = (x-x_{\text{min}}) \times \text{ratio}, B = b^r \), and Eq. (7) can be simplified to

\[
\Delta x = \left[ \frac{A}{B} \right] - \left[ \frac{A}{B} \right]
\]

If \( b \) is a decimal, we define \( A = z + e(z \in \mathbb{Z}, 0 < e < 1) \),

\[
z = fB + g(f \in \mathbb{Z}, 0 < g < B)
\]

and Eq. (7) can be simplified to

\[
\Delta x = \left[ \frac{z}{B} \right] - \left[ \frac{z + e}{B} \right] = f - \left[ \frac{f + g + e}{B} \right]
\]

As \( 0 < g + e < 2B \), we have \( |\Delta x| \in (0, 1) \). The mapping error is zero if the zoom ratio is an integer number, and the mapping error does not exceed \( \sqrt{2} \) pixels if the zoom ratio is a decimal.

In addition, integer data may be used to calculate the distance between two points. To calculate distance, the coordinates should be transformed from the pixel-coordinate system to the geo-coordinate system. Apparently, the point obtained from the inverse transform is not equal to the original one, and the difference is affected not only by the boundary of data but also by the bit length of each integer coordinate component.

3 Integer wavelet transform

IWT can be used to reduce the redundancy between points on the same curve. High- and low-frequency coefficients are produced by IWT, and most high-frequency coefficients are small numbers. When we apply IWT repeatedly to the low-frequency coefficients, the original data can be transformed to data consisting of only small integers. IWT is reversible, and thus, data can be reconstructed without any loss. For example, the length of high-frequency filter is 5, and the length of low-frequency filter is 3. Given a data vector of \( N \) integers \( x_i \), we define \( k = N/2 \), if \( N \) is even. The transform process can be summarized as

\[
y_{2i+1} = \begin{cases} 
    x_{2i+1} - \left( \frac{x_{2i+2}}{2} \right) & i = 0, 1, \ldots, k-2 \\
    x_{2i+1} - x_{2i} & i = k-1
\end{cases} \tag{9}
\]

\[
y_{2i} = \begin{cases} 
    x_{2i} + \left( \frac{y_{2i+1}}{2} \right) & i = 0 \\
    x_{2i} + \left( \frac{y_{2i+1} + y_{2i+2}}{4} \right) & i = 1, 2, \ldots, k-1
\end{cases} \tag{10}
\]

and the inverse transform can be summarized as

\[
z_{2i+1} = \begin{cases} 
    y_{2i} - \left( \frac{y_{2i+1}}{2} \right) & i = 0 \\
    y_{2i} - \left( \frac{y_{2i+1} + y_{2i+2}}{4} \right) & i = 1, 2, \ldots, k-1
\end{cases} \tag{11}
\]

\[
z_{2i} = \begin{cases} 
    y_{2i} + \left( \frac{z_{2i+1}}{2} \right) & i = 0 \\
    y_{2i} + \left( \frac{z_{2i-1} + z_{2i+2}}{4} \right) & i = 1, 2, \ldots, k-2
\end{cases} \tag{12}
\]

4 Canonical Huffman coding

The coordinates are converted to low-frequency
and high-frequency coefficients. Most values of low-frequency coefficients are big integers, whereas most values of high-frequency coefficients are small integers that can be encoded through a variable-length encoder. Short codes are assigned to integers that occur often, and long codes are assigned to integers that occur seldom. The Huffman method is simple, efficient, and produces the best codes for individual data symbols. However, the only case where it produces ideal variable-size codes is when the symbols have probabilities of occurrence that are the negative powers of 2. This is because the Huffman method assigns a code with an integer number of bits to each symbol in the alphabet. Arithmetic coding overcomes the problem of assigning integer codes to the individual symbols by assigning one code to the entire input file. While an error in a single bit prevents the bits that follow from being correctly decoded by an arithmetic decoder; Huffman codes tend to resynchronize quickly, thus localize damage. An arithmetic decoder also uses the frequencies of dictionary entries during decoding. We choose the Canonical Huffman coding for our application because it costs less storage for extra information decoding than Huffman coding, and the bit stream can be decoded efficiently. For example, the Huffman code tree shown in Fig. 1 can be represented in the following preorder form: \(11\) (0,13) (0,11) (0,9) (1,C) (1,D) (1,A) (0,17) (1,B) (0,29) (0,23) (1,F) (0,27) (1,G) (1,H) (1,E). It costs 14 bytes to store the address. Canonical Huffman coding writes the number of codes with a specified length to bit stream. The extra information for decoding can be represented as follows: 0 2 A B 3 C D E 1 F 2 G H. It costs only 5 bytes to store the count. Identifying the length of a code by reading and examining the input bits one by one is easy for the decoder because of the way the codes are constructed. Once the length is determined, the symbol can be found in one step.

5 Experimental studies

We use two datasets in our experiments. Dataset 1, which comes from the National Geomatics Center of China, comprises river data from levels 1-3 with a scale of 1:4000000. It contains 2143 records, with each record containing about 64 points on the average. Dataset 2, which comes from the US Census Bureau, comprises road lines in Florida. It contains 32387 records, with each record containing about 3 points on the average.

In Experiment 1, the maximum view scale of Dataset 1 is set to 1:500000, and it is set to 1:5000 for Dataset 2. Each coordinate component costs 16 bits of storage; the zoom ratio is 2.0. Each dataset is divided into seven sub-datasets with a fixed scale. The integer coordinate component ranges from 0-1024 at the minimal view scale. This means that the minimal view scale for Dataset 1 is 1:32000000, whereas the minimal view scale for Dataset 2 is 1:320000. The percentages of points in each sub-datasets are shown in Table 1. The subdatasets of Dataset 2 contains more than 90% of points belonging to Level 1. This result can be explained by the fact that the curves of Dataset 2 only contain about 3 points on the average, but the start and end points of each curve should not be removed from the approximate curve that belongs to Level 1.

| Level | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|
| Dataset 1 | 19.4 | 14.1 | 21.2 | 24.0 | 15.7 | 5.0 | 0.7 |
| Dataset 2 | 90.0 | 6.5 | 2.7 | 0.7 | 0.1 | 0 | 0 |

To study the efficiency of IWT in compressing approximate curves, we compare it with another approach that stores the first point and coordinate increments instead of coordinates. The input data of Experiment 2 is the approximate curves of Dataset 1. The offset distribution of coordinates is shown in Fig. 2. More than 99% of the values of \(\Delta x\) are within the arrangement of \([-27, 27]\) with entropy of 4.76, and more than 99% of the values of \(\Delta y\) are within the arrangement of \([-22, 23]\) with entropy of 4.47. The distribution of high-frequency coefficients is shown in Fig. 3. The entropy of \(x\) component is 4.43 and that of \(y\) is 4.18. Therefore, IWT is more effective. Finally, the high-frequency coefficients are passed on to the
Canonical Huffman encoder. The average length of codes for $x$ is 4.79 and that for $y$ is 4.70. The entropies of $x$ and $y$ before the transform are 14.97 and 14.31, respectively.

In Experiment 3, IWT is applied to the approximate curves of the two datasets, and the high-frequency coefficients are then encoded by the Canonical Huffman algorithm. The compression ratio for Dataset 1 is 7.52% and that for Dataset 2 is 16.62%. Dataset 2 is mainly composed of road lines containing less than four points on the average; hence, the redundancy between the points of a curve makes less contribution to the compression ratio. More attention should thus be paid to the correlation between curves.

6 Conclusion

Current vector data compression approaches pay little attention to progressive data transmission. Thus, we proposed a progressive representation of curve using type conversion. A curve is divided into an approximate curve and several supplement points with different view scales. As the converted points are close to the original points and the mapping error is less than 2 pixels, our approach is suitable for vector data visualization in WebGIS. Experiments on the two kinds of data show that the compression ratio of curves with more points is smaller than that of the curves with fewer points. Hence, we suggest that more attention should be paid to the correlation between curves having a small size of points.

References

[1] Wang H, Yu Z, Zeng W, et al. (2009) The research on the algorithm of spatial data cache in network geographic information service[J]. Acta Geodaetica et Cartographica Sinica, 38(04): 348-355

[2] Douglas D H, Peucker T K (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature[J]. The Canadian Cartographer, 10(2): 11 eta2-122

[3] Chan W S, Chin F (1992) Approximation of polygonal curves with minimum number of line segments[J]. Lecture Notes in Computer Science, 650: 378-387

[4] Kolesnikov A, Fr Nti P (2003) Reduced-search dynamic programming for approximation of polygonal curves[J]. Pattern Recognition Letters, 24(14): 2243-2254

[5] Kolesnikov A, Fr Nti P (2005) Data reduction of large vector graphics[J]. Pattern Recognition, 38(3): 381-394

[6] Perez J C, Vidal E (1994) Optimum polygonal approximation of digitized curves[J]. Pattern Recognition Letters, 15(8): 743-750

[7] Salomon D, Motta G, Bryant D (2007) Data compression: the complete reference[M]. New York Inc.: Springer-Verlag

[8] Freeman H (1974) Computer processing of line-drawing images[J]. ACM Computing Surveys (CSUR), 6(1): 57-97

[9] Shekhar S, Huang Y, Djugash J, et al. (2002) Vector map compression: a clustering approach[M]. McLean, Virginia, USA: ACM Press

[10] Yang Bisheng, Li Qingquan (2009) Efficient compression of vector data map based on a clustering model[J]. Geo-Spatial Information Science, 12(1): 13-17

[11] Hirschberg D S, Lelewer D A (1990) Efficient decoding of prefix codes[J]. Communications of the ACM, 33(4): 449-459