Construction of the Web Service for Smooth and Fast Rendering of Large SSC Dataset and the Preprocessing of Source Data

MSc Geomatics Thesis P3

Yueqian Xu

April 2, 2017

1 Research questions

1.1 Objectives

The object of the research is to develop a web service for smooth vario-scale map rendering during zooming and panning of large dataset.

1.2 Research questions

Main research question:
What is the architecture within the prototype that allow client to conduct repetitive user interactions and how does it reflect on the format of input data?

1. Server side:

    - In the existing OBJ files, vertices and triangles can be distinguished by the starting character of each line. However, it has already been proved that progressively comparing and splitting strings (decoding) of a static file is slow under Javascript environment. How should the static files be formatted? Is binary file a feasible format under this circumstance?
    - How should the original dataset be structured and serialized so that it can be directly loaded into GPU after binary encoding?
    - During the octree dividing, what is the affiliation of a triangle with the chunks it is intersecting with? What will the size change before and after the binary formatting if the one triangle belongs to all intersecting chunks?
    - What will be difference between the preprocessed data size using ordinary octree and other grouping methods?

2. Client side:

    - What is the limitation of the web service? What will be the maximum size of packages for one rendering process?
    - If a user repetitively zooming in/out during a short period, will there be overload? How to store loaded chunks in buffer? How to update buffer data and verex number without the unloading of all chunks that were requested by previous render request?
    - If there is gap between the package(s) required before and after zooming, how can the animated frames be generated? By loading all chunks in between?

2 Methodology

The research framework contains two segments: (1) data preprocessing at server side Figure 2.1 and, (2)client development.

2.1 Data preprocessing

The original OBJ file is composed as shown in Table 2.1. A line starts with a capital "F" followed by object id (int) and the number of faces of the object (int) indicates the beginning of an object. The following lines represent the triangles that form the object; each line contains three integers which are references to the vertices. The SSC dataset is then divided into eight octants; any triangle that intersecting with more than one octant belongs to all the octants it intersects. For example, as shown in Figure 2.2, the large triangles at the top of the model, they belong to both chunk 1 and 2. An individual file is then generated for every chunk. The content of the chunk file is shown in Table 2.2. Each line is composed by three vertices of a triangle; the vertex reference is followed by a color id. The color information can be obtained by Oid & 15; it returns an indicator to the color list.

During the preprocessing, the coordinates of the upper right and lower left vertices of bounding box of each chunk are detected and outputted to a new binary file. The bounding box file contains eight lines; the first 24 bytes of each line represent the bounding box of the corresponding octant produced after the first division. Take the bandwidth into consideration, assume that most PC users have a bandwith at 3-5 MB per second; the file size of each chunk should be limited (around 3 MB in this case); therefore, the initial chunk file should be restricted to around 1 MB (the final file size will be 3 times larger than the initial chunk file after binary compiling hence the initial chunk file size should be one third of the 3 MB bandwidth limitation).
After the first division, the chunks which are over 1 MB should be subdivided. The bounding box coordinates of the newly generated chunks are appended to the line which represents the root octant. Recursively subdivide the dataset until all leaf octants are smaller than 1 MB. Finally, write all leaf chunks to binary file separately; the content of the binary file is shown in Table 2.3. The binary file contains one line which is formed by x, y, z coordinates and their R, G, B value; one followed by another, without any white space or end of line: [x1 y1 z1 R G B x2 y2 z2 R G B. . .].

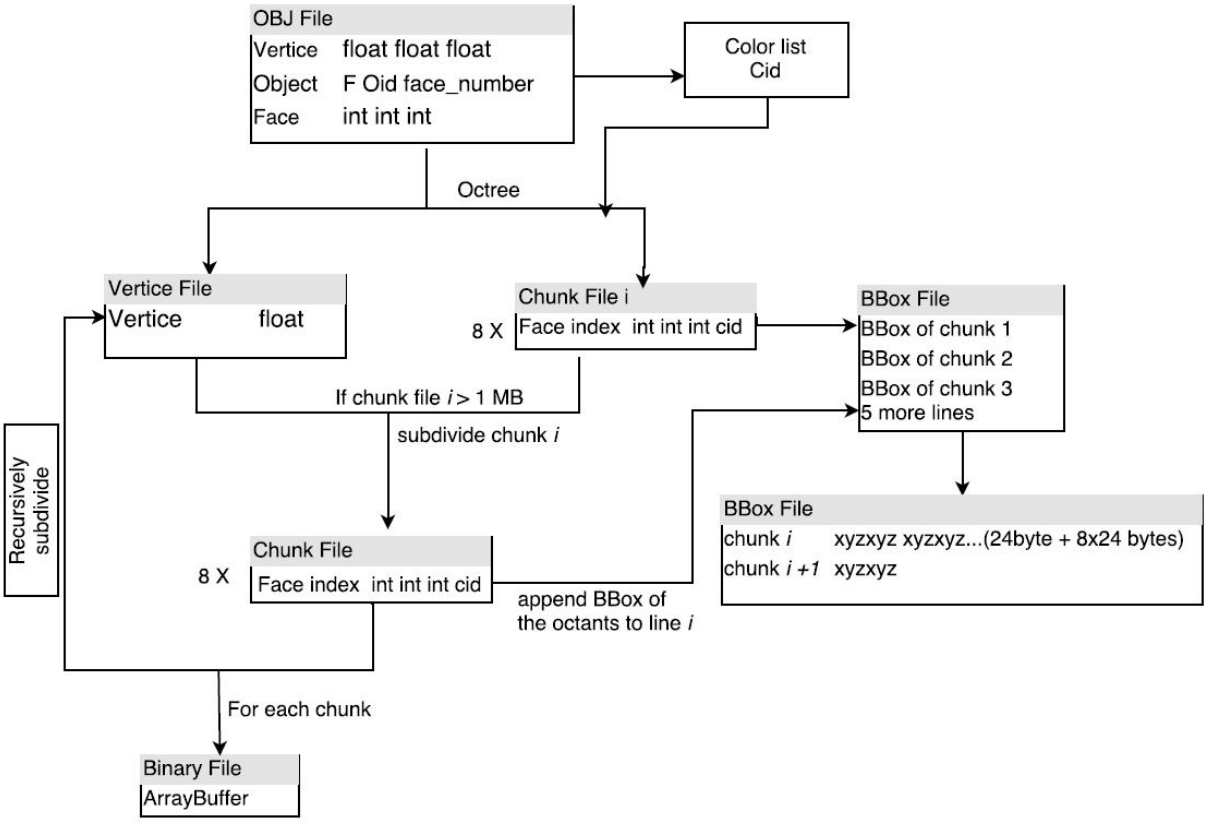
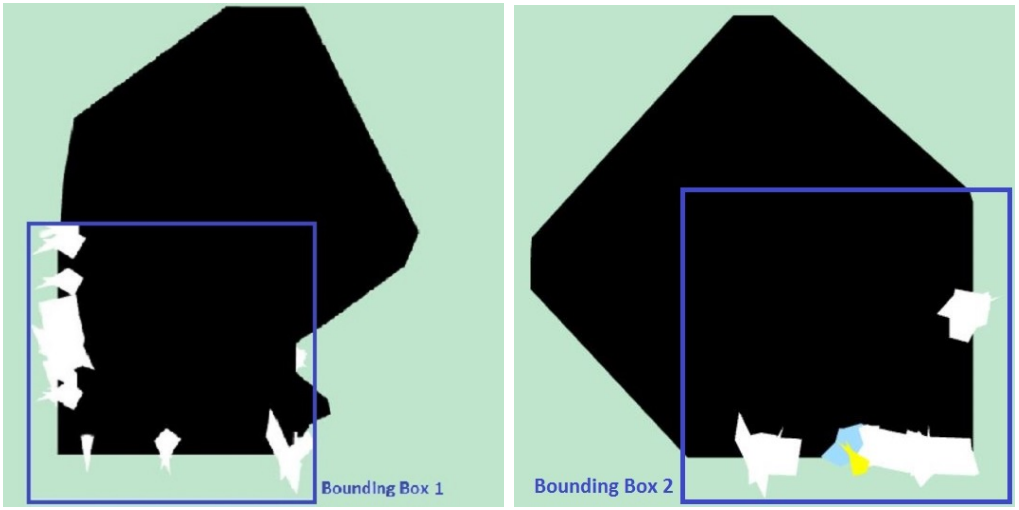


Figure 2.1: The flowchart of data preprocess

v	93851.3255	463551.399	378
v	93848.358512	463548.100973	378
v	93853.1826667	463553.491	378
...			
F	16099670	42	
114803	114802	114801	
114801	114804	114803	
114807	114805	114806	
...			
F	16072661	4	
87399	87398	87397	
87397	87400	87399	
87401	87399	87400	
...			

Table 1: Original OBJ file



(a) Triangles intersecting with chunk 1 *(b) Triangles intersecting with chunk 1*

Figure 2.2: Larger triangle intersecting with more than one chunk

v1	v2	v3	color id
3194	1277	1280	7
1280	34	3194	7
1899	752	5369	9
...			

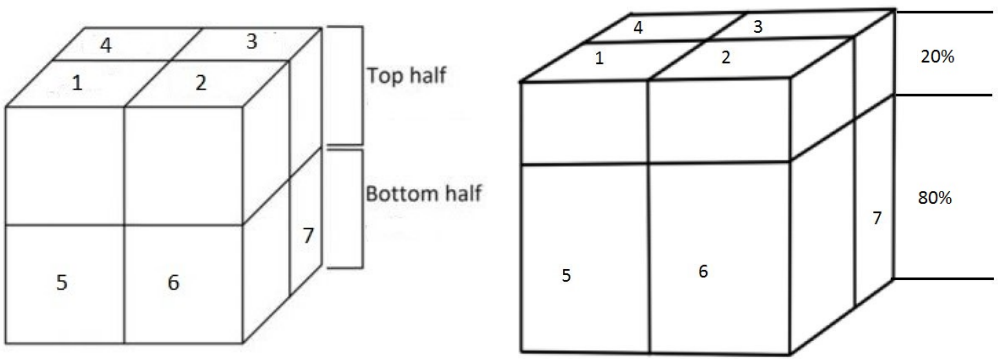
Table 2: Chunk file content

x1	y1	z1	R	G	B
93851.3	463551.4	378	1	0.5	0
12 bytes			12 bytes		

Table 3: Binary file content

2.2 Octree

It is found that the size of the chunks in the top half is much larger than those in the bottom half if the model is evenly divided into eight octants. It is mainly caused by the plural affiliation of a large triangle with different chunks. Another division method is then tested, as shown in Figure 2.3, the model is evenly divided in the horizontal direction but unevenly divided in the vertical direction. The top part is redefined as the part over the 1:4 plane.
Another dividing method is applied to the dataset as well; Figure 2.4 shows a separate binary file for triangles intersecting with more than one chunk. By using this dividing method, duplicated rendering of those triangles can be avoided. The comparison of file size among three dividing methods is listed in Table 2.4. It can be indicated that the wide gap between the file size of upper and lower chunks is not caused by the plural affiliated triangles. The output file sizes are more balanced using the second dividing method (1:4 octree).



(a) Evenly divided octree *(b) 1:4 divided octree*

Figure 2.3: Octree model

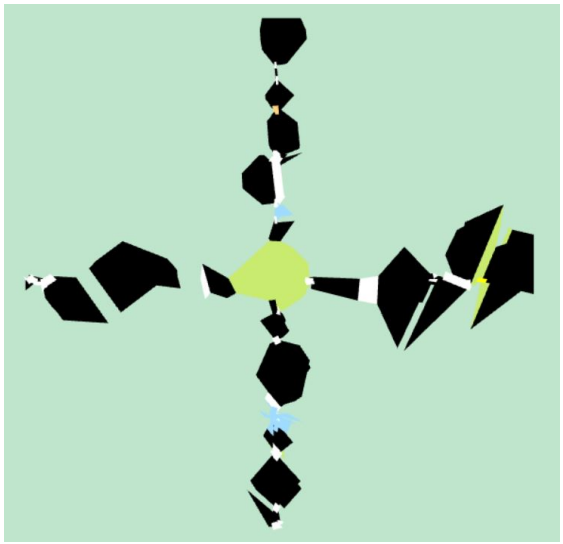


Figure 2.4: A sepatare file for triangles belonging to multiple chunks

Chunk	Size (1:1 octree)(KB)	Size (1:1 octree)	Size (1:4 octree)
1	2940	2660	1827
2	2953	2664	1739
3	4155	3875	2340
4	3858	3587	2640
5	208	186	1320
6	175	157	1389
7	463	408	2278
8	777	718	2010
upper intersecting		557	
lower intersecting		75	
Total	15529	14887	15543

Table 4: Comparison of bianry file size

2.3 JavaScript side

The process before the scene is rendered at the client GPU is shown in Figure 2.5. Once the client passes a request for specific chunk(s), the corresponding binary file(s) will be loaded by an XMLHttpRequest. The response to the XMLHttpRequest is an ArrayBuffer from which the attributes needed for rendering such as the number of vertices and the buffer size can be acquired. Push the ArrayBuffer to the buffer list then it can be used for other XMLHttpRequest. Add number of vertices and the buffer size of this chunk to the total vertices number and total buffer size respectively. By knowing the buffer size, an empty buffer with correct size is generated. Data is given to the empty buffer by BufferSubData function, with an offset equals to the total buffer size. Therefore, the buffer to be rendered and the number of vertices to be render are extended after each XMLHttpRequest is conducted. If all required chunks are loaded and ready to be rendered, the buffer will be send to the client GPU along with the total number of vertices.

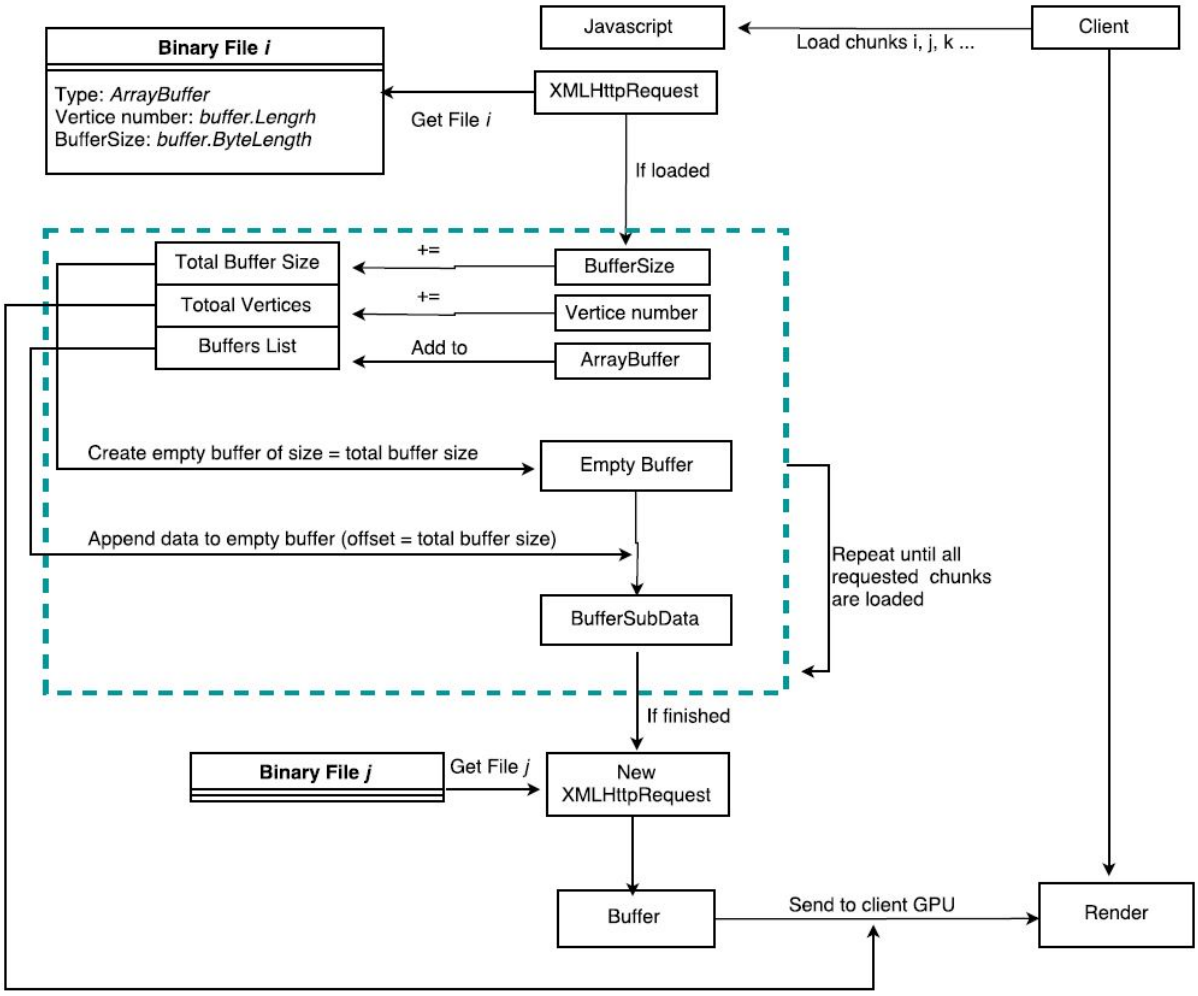


Figure 2.5: The process at JavaScript side before the scene is rendered

3 Time planning

3.1 Activities

It sets up a series of activities that are needed to achieve the research objectives. Literature study of the SSC model and the development of a testing prototype are done before P2. Some important dates are listed in Table 4. The schedule between P3 and P4 are listed in Table 6. Here listed some important date:

Date	Activity
3 April	P3
13 April	Final application dates for P4
11-24 May	P4
24 May	Final application dates for P5 (the draft should be finished before)
26 June–7 July	P5 (report should be finalized before)

Table 5: Important dates

Activity	date
Dividing data into chunks based on Octree	Done
Format data into binary and viewport position determination	Done
Load and unload multiple packages at client side	Done
Pass mouse events after one rendering to Javascript side	10-17 April
Try to store unwanted chunks in memory once they are loaded instead of reloading them when requested again	18-25 April
Generate slices inbetween two interactions	1-7 May
Enrich user interactions and final report	after P4

Table 6: Activities after P3