CONSTRUCTION OF RESPONSIVE WEB SERVICE FOR SMOOTH RENDERING OF LARGE SSC DATASET AND THE CORRESPONDING PREPROCESSOR FOR SOURCE DATA

Yueqian Xu

Mentor #1: Martijn Meijers
Mentor #2: Peter van Oosterom
Mentor #3: Timothy Kol (Computer Graphics)
Introduction

- Space Scale Cube (SSC) model: vario-scale geographical data structure. Non-redundant geometric data for different level of details.

- Viewport acts as a camera; only chunks intersecting with viewpoint will be transferred to GPU.

- Develop a web service to reveal geometry change against massive user actions with large dataset.
Problem statement:
• Web service pursues fluent performance and fast responsiveness
• Large datasets: 9km x 9km dataset > 40 MB (one chunk)
• Limited bandwidth → Long data transmission time
• Poor decoding capability (parse text based data) of Javascript

Ultimate goal:
Implement a web-based service along with its preprocessor that:
• Performs well with large datasets;
• Enables fast and smart data transmissions;
• Eliminates decoding time through direct GPU uploads;
• Minimizes the number of HTTP requests by reusing memory slots.

(a) Concept of smart data fetching, anti-reloading, and reusing memory slots
Research questions

What is the architecture of web service? What are the possible data format and serialization method?

Preprocessing:
• Is binary format a possible arrangement? How should the text-based source files be formatted?
• What is the size change after octree dividing (different thresholds & allocation of triangles)?

Prototype development:
• How should the octree structure be reflected in Javascript?
• How to define a viewport bounding box and update it regarding user actions?
• What is the dynamic and light schema that prevents repeated loading and allows reuse of memory against heavy user actions?
Related works – WebGL rendering

**GL Shader Language**
- Vertex shader (manipulate vertex position)
- Fragment shader (assign color)
- Call drawArray

**WebGL coordinate system**
- Coordinates in all three axes go from -1.0 to +1.0
- Z for depth testing

**Near Z plane**
- Everything above it will be cut away
- Move near z plane from the top downwards, map scale changes are revealed.
Related works – Main memory vs. GPU memory

- Upload data to GPU memory from outside.
- Rendering is fast after data transmission.
- Data transfer is relatively slow.
Source data

OBJ File

<table>
<thead>
<tr>
<th>v</th>
<th>x coordinate</th>
<th>y coordinate</th>
<th>z coordinate</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>93851.3255</td>
<td>463551.399</td>
<td>378</td>
</tr>
<tr>
<td>v</td>
<td>93848.358512</td>
<td>463548.100973</td>
<td>378</td>
</tr>
<tr>
<td>v</td>
<td>93853.1826667</td>
<td>463553.491</td>
<td>378</td>
</tr>
<tr>
<td>g</td>
<td>1001706</td>
<td>13000</td>
<td>437 506</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>f</th>
<th>Vertex index 1</th>
<th>Vertex index 2</th>
<th>Vertex index 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>114803</td>
<td>114802</td>
<td>114801</td>
</tr>
<tr>
<td>f</td>
<td>114801</td>
<td>114804</td>
<td>114803</td>
</tr>
</tbody>
</table>

| g   | 1001704       | 12400          | 435 452       |

OBJ file content

Color information

<table>
<thead>
<tr>
<th>Class id</th>
<th>Red Value</th>
<th>Green Value</th>
<th>Blue Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>13000</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
</tbody>
</table>

Dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of triangles</th>
<th>Scope (minx, minY, maxX, maxY) (RD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smooth sample</td>
<td>136</td>
<td>(-0.993582, 0, 0, 1)</td>
</tr>
<tr>
<td>Leiden</td>
<td>10,125</td>
<td>(93500, 463500, 94100, 464100)</td>
</tr>
<tr>
<td>9x9</td>
<td>3090.8k</td>
<td>(182000, 308000, 191000, 317000)</td>
</tr>
</tbody>
</table>

Three experimental datasets
Data preprocessing concept

**OBJ File**
- Vertex coordinates: x, y, z
- Object information: Oid, class_id, min_lifespan, max_lifespan, vertex id, index of vertices composing the triangle

**Color List**
- Class_id
- RGB Value

**Node: struct**
- Recursively subdivide
- If length of Node_data > size limit
  - Normal Octree Dividing
  - Lower Node
    - For each node
      - If length of Node_data > size limit
        - BBox File of Node (txt file): Bounding box of 8 leaf nodes
  - If length of Node_data <= size limit
    - Binary File: ArrayBuffer
Dividing methods: Octree

- Order
- Node id = binary file name
- Threshold: <500KB & max 4 levels

Node content:

<table>
<thead>
<tr>
<th>Root Node</th>
<th>Leaf Node (lowest)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ Node.level: 0</td>
<td>+ Node.level: int</td>
</tr>
<tr>
<td>+ Node.id: &quot;0&quot;</td>
<td>+ Node.id: string</td>
</tr>
<tr>
<td>+ Node.data: vector of floats</td>
<td>+ Node.data: vector of floats</td>
</tr>
<tr>
<td>+ Node.bbox: vector of floats</td>
<td>+ Node.bbox: vector of floats</td>
</tr>
<tr>
<td>+ Node.children: vector of 8 nodes</td>
<td>+ Node.children: [ ]</td>
</tr>
</tbody>
</table>
Allocation of triangles

- If a triangle intersects with more than one chunk BBox, it will be added into all chunks it is intersecting with → cause redundancy.
- Avoid missing geometry at chunk boundaries.

6 disjoint cases
Preprocessing results

Binary formatted source data:
• One triangle $\rightarrow$ 72 bytes

<table>
<thead>
<tr>
<th>x1</th>
<th>y1</th>
<th>z1</th>
<th>R</th>
<th>G</th>
<th>B</th>
<th>x2</th>
<th>y2</th>
<th>z2</th>
<th>R</th>
<th>G</th>
<th>B</th>
<th>x3</th>
<th>y3</th>
<th>z3</th>
<th>R</th>
<th>G</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>0.3</td>
<td>0.5</td>
<td>1.0</td>
<td>0</td>
<td>0.5</td>
<td>0.8</td>
<td>0.4</td>
<td>0.2</td>
<td>1</td>
<td>0</td>
<td>0.5</td>
<td>0.7</td>
<td>0.3</td>
<td>0.5</td>
<td>1</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>12 bytes</td>
<td>12 bytes</td>
<td>12 bytes</td>
<td>12 bytes</td>
<td>12 bytes</td>
<td>12 bytes</td>
<td>12 bytes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) A slice of the binary file and the size in byte

File size:

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Size (MB)</th>
<th>Chunks</th>
<th>$&lt;$ threshold</th>
<th>$&gt;$ threshold</th>
<th>$&lt;$ 50KB Max (KB)</th>
<th>Min (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>One chunk</td>
<td>40</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$&lt;$ 500KB (no lifespan)</td>
<td>43.9</td>
<td>400</td>
<td>400</td>
<td>0</td>
<td>130</td>
<td>484</td>
</tr>
<tr>
<td>$&lt;$ 500KB (with lifespan)</td>
<td>(9.75%)</td>
<td>(100%)</td>
<td>(0%)</td>
<td>(32.5%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>239</td>
<td>1135</td>
<td>1123</td>
<td>12</td>
<td>72</td>
<td>526</td>
<td>25</td>
</tr>
</tbody>
</table>

(b) Comparison of chunk size of 9x9 dataset
Node structure at client side

<table>
<thead>
<tr>
<th>New node</th>
<th>Type</th>
<th>tree._root</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ BBox = [ ]</td>
<td>List of floats</td>
<td>[-1, 0, 0, 0, 1, 0.885833]</td>
</tr>
<tr>
<td>+ depTogo = null</td>
<td>0 or 1</td>
<td>1</td>
</tr>
<tr>
<td>+ intersecting = false</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>+ loaded = false</td>
<td>Boolean</td>
<td>false</td>
</tr>
<tr>
<td>+ timestamp = [ ]</td>
<td>int</td>
<td></td>
</tr>
<tr>
<td>+ numVertex = [ ]</td>
<td>Int</td>
<td>300</td>
</tr>
<tr>
<td>+ BufferObject = [ ]</td>
<td>Buffer Object</td>
<td>gl.createBuffer()</td>
</tr>
<tr>
<td>+ Children = [ ]</td>
<td>List of child nodes</td>
<td>[rootNode00, rootNode01, ... , rootNode07]</td>
</tr>
</tbody>
</table>

(a) Node content and data type

```javascript
Node.prototype.addChild = function(BBox, depTogo) {
    var child = new Node(BBox, depTogo);
    this.children.push(child);
};
```

```javascript
function addLevel(ParentNode, Child_BBoxes){
    for (var i = 0; i < Child_BBoxes.length; i++) {
        ParentNode.addChild(Child_BBoxes[i], Child_BBoxes[i][5]);
    }
}
```

(b) Pseudo code to construct tree at client side

```javascript
var rootNode0 = tree._root;
var box0 = [ [-0.5, 0.0, -0.5, 0.442917, 1], [-1.0, 0.0, -0.5, 0.5, 0.442917, 0], [-0.5, 0.5, 0, -0.1, 0.442917, 1], [-1, 0.5, 0, -0.5, 1, 0.442917, 1], [-0.5, 0.0, 0.442917, -0.5, 0.5, 0.885833, 0], [-1, 0.0, 0.442917, -0.5, 0.5, 0.885833, 0], [-0.5, 0.5, 0.442917, -0.5, 1, 0.885833, 1], [-1, 0.5, 0.442917, -0.5, 1, 0.885833, 1] ];
addLevel(rootNode0, box0);
```

(c) A parent node automatically generated during preprocessing
Intersection testing function

Function is called after each user action
Modified LoadChunk function

Compared with old schema, the modified one:
• takes longer to finish loading a chunk;
• stores chunk data directly in GPU memory;
• ArrayBuffer objects cause no main memory usage.
Modified RenderChunk function

Compared with old schema, the modified one:
• doesn’t communicate with outside;
• hence, the rendering is fast and light;
• causes only GPU memory usage.
User actions

• Pan

- Manipulate vertex position
- New x coordinate = old coordinate - xoffst (dragged distance)

• Zoom

- Visual enlargement coordinate * zoom
- Near z plane position = 1/zoom
**Update user action parameters**

**Mouse Movements**

**Pan**
- **Press left key**
  - Set `drag = true`
  - `old page X = e.pageX`
  - `old page Y = e.pageY`
  - `original location X = current loc X`
  - `original location Y = current loc Y`

- **Drag**
  - Fetch `e.pageX` and `e.pageY` at every pan step
  - Update current x & y location at every pan step
  - Update viewport BBox at every pan step

- **Release left key**
  - Set `drag = false`
  - Call intersection test function

**Zoom**
- **Mouse Wheel**
  - Update Zoom at every zoom step
  - Update viewport BBox at every pan step
  - Call intersection test function

**Offset_X & Offset_Y & Zoom**

Every frame

**Vertex shader**

'\[\text{gl\_Position} = \text{viewmatrix} \times \text{vec4(zoom} \times \text{vec3(extent, extent, 1.0)} \times (\text{vertPosition} - \text{vec3(\text{offsetX, offsetY, 0)}), 1.0);\]'

**Manipulate vertex position**

offsetX = origLocX + panStepSize * (1.0 / mouseZoom) * (e.pageX - oldPageX);
offsetY = origLocY + panStepSize * (1.0 / mouseZoom) * (e.pageY - oldPageY);
Viewport bounding box

Viewport BBox is defined by VP centroid & radius
It only relates to normalized source data
Radius = 0.5/zoom
Centroid = (offset_\(X\), offset_\(Y\))

(a) Viewport Bounding Box

\[
\begin{align*}
\text{minVPX} &= \text{offsetX} - \frac{1.0}{\text{mouseZoom}}/2; \\
\text{maxVPX} &= \text{offsetX} + \frac{1.0}{\text{mouseZoom}}/2; \\
\text{minVPY} &= \text{offsetY} - \frac{1.0}{\text{mouseZoom}}/2; \\
\text{maxVPY} &= \text{offsetY} + \frac{1.0}{\text{mouseZoom}}/2;
\end{align*}
\]
(b) Update viewport bounding box
Prototype performance

- Reveal geometry change

(a) Z value = 0.02998

(b) Z value = 0.02848 (zoom step = 0.95)

(c) Obvious gradual change (zoom step = 0.95)
Prototype performance

• Accuracy

(a) Coordinates obtained by prototype

(b) Online map for validation (Adapted from EPSG (2017))

• No repetitive loading of chunks

(c) ArrayBuffer objects of Leiden dataset

<table>
<thead>
<tr>
<th>Total Count</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0%</td>
</tr>
<tr>
<td>8</td>
<td>0%</td>
</tr>
<tr>
<td>564</td>
<td>2%</td>
</tr>
<tr>
<td>7,246</td>
<td>22%</td>
</tr>
<tr>
<td>10,185</td>
<td>30%</td>
</tr>
</tbody>
</table>

No more than 8 ArrayBuffer objects; hence, no repeated loading.
Time consumption – modified schema

(a) A typical workflow of intersection testing, loading, and rendering (load one chunk: 50ms)

(b) Time consumption for pure tree traversal and rendering (less than 10ms)

<table>
<thead>
<tr>
<th>Modified schema</th>
<th>Old schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Function</strong></td>
<td>% of time</td>
</tr>
<tr>
<td>Gecko</td>
<td>45.9</td>
</tr>
<tr>
<td>Graphics</td>
<td>33.8</td>
</tr>
<tr>
<td>RenderChunk</td>
<td>5.5</td>
</tr>
<tr>
<td>Tools</td>
<td>3.6</td>
</tr>
<tr>
<td>loadChunk</td>
<td>2.3</td>
</tr>
</tbody>
</table>

(c) Most time-consuming calls during a complete performance recording
Time consumption – local server

(a) Javascript frame chart during 2581ms to 5632ms (Modified schema: low fps due to loading of chunks and data transmission to GPU)

(b) Time consumption for pure tree traversal and rendering (Modified schema: less than 10ms)

(c) Lags caused by data transmission result in low fps (old schema)
Time consumption – remote server

- Significantly affected by network condition; especially when zooming out to the top of the model.

(a) Relative low fps due to delay of data transmission through network (modified program with network at 6MB/s)

(b) Relative higher (modified program with network at 9MB/s)
# Memory consumption

## Main memory usage (old schema):

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Average fps</th>
<th>Memory at loading (MB)</th>
<th>after traversal (MB)</th>
<th>ArrayBuffer (MB)</th>
<th>Tree Structure (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample data</td>
<td>57.1</td>
<td>2.3</td>
<td>4.3</td>
<td>0.01 (0%)</td>
<td>0</td>
</tr>
<tr>
<td>Leiden</td>
<td>58.9</td>
<td>4.84</td>
<td>5.58</td>
<td>0.9 (17%)</td>
<td>0.03</td>
</tr>
<tr>
<td>9km x 9km</td>
<td>39.0</td>
<td>5.96</td>
<td>238</td>
<td>233 (98%)</td>
<td>0.79</td>
</tr>
</tbody>
</table>

(a) General performance of three datasets at different states (old schema)

## Main memory usage (Modified schema):

<table>
<thead>
<tr>
<th>Stages</th>
<th>Main memory use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right after heavy user actions</td>
<td>120.34 MB</td>
</tr>
<tr>
<td>Idle the browser for 10 seconds</td>
<td>10.25 MB</td>
</tr>
<tr>
<td>Idle the browser for another 1 minute</td>
<td>6.7 MB</td>
</tr>
</tbody>
</table>

(b) Main memory use of the modified program at different stages
Unloading from GPU memory

(a) Unloading function

- Call unload function every 15 seconds
- Call LoadChunk function for unloaded chunks if they are once again required
- Increases CPU computation
GPU memory consumption – with unloading

- **Stage 1:** GPU memory use at the initial loading of the page
  - **169MB**

- **Stage 2:** GPU memory use after traversing through the dataset
  - **462MB**

- **Stage 3:** GPU memory use after unloading
  - **130MB**
Future work

• 475% volume up caused by the duplication due to the lifespan; is there a better way to deal with it?

• The tree structure of the 9x9 dataset is 0.79MB; it could be > 6MB for a 20x20 dataset. Is it possible to split tree structure script into multiple scripts, load a particular part only when it is requested?

• Geometry changes are subtle that are easily being skipped over with a large zoom step. Is there a way to magnify the change either within source data or during rendering? For example, generate an animation.

• Different unloading methods; e.g. based on distance or times of requests.

• Balance the use of main memory and GPU memory.

• After unloading, will be GPU memory be fragmented? Does that affect the performance?
Prototype
Thank you for your attention
Questions
Supplementary slides
Happens when the horizontal splitting plane intersects with the lifespan of a triangle

<table>
<thead>
<tr>
<th>Chunk</th>
<th>Size (without lifespan) (kb)</th>
<th>Size (with lifespan) (kb)</th>
<th>Size (one chunk) (kb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>104</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>83</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>142</td>
<td>141</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>125</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>79</td>
<td>114 (44% up)</td>
<td>729</td>
</tr>
<tr>
<td>05</td>
<td>70</td>
<td>100 (42% up)</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>100</td>
<td>140 (40% up)</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>87</td>
<td>126 (45% up)</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>790 (8% up)</td>
<td>930 (28% up)</td>
<td></td>
</tr>
</tbody>
</table>

Comparison of chunk size divided with/without lifespan (Leiden dataset)
Example of load-render workflow

- **Main memory (Node00)**
  - intersecting = True
  - loaded = True
  - Temporal tribuffer = 00.bin

- **Server**
  - Load Chunk 00

- **Main memory (Node01)**
  - intersecting = False
  - loaded = False

- **GPU memory**
  - One buffer object
  - Buffer data 00.bin
  - Node00 rendered

- **Main memory (Node00)**
  - intersecting = True
  - loaded = True

- **Server**
  - Load Chunk 01

- **Main memory (Node01)**
  - intersecting = True
  - loaded = True
  - Temporal tribuffer = 01.bin

- **GPU memory**
  - Two buffer objects
  - Buffer data 00.bin 01.bin
  - Node00 rendered
  - Node01 rendered

- **Main memory (Node00)**
  - intersecting = False
  - loaded = False

- **Server**
  - No loading of chunks

- **Main memory (Node01)**
  - intersecting = True
  - loaded = True

- **GPU memory**
  - Two buffer objects
  - Buffer data 00.bin 01.bin
  - Node01 rendered
Separate file for triangles intersecting with multiple chunks → avoid redundancy

<table>
<thead>
<tr>
<th>Chunk</th>
<th>Separate files</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Intersecting 01</td>
</tr>
<tr>
<td>1</td>
<td>Intersecting 13</td>
</tr>
<tr>
<td>2</td>
<td>Intersecting 23</td>
</tr>
<tr>
<td>3</td>
<td>Intersecting 02</td>
</tr>
<tr>
<td>4</td>
<td>Intersecting 45</td>
</tr>
<tr>
<td>5</td>
<td>Intersecting 67</td>
</tr>
<tr>
<td>6</td>
<td>Intersecting 57</td>
</tr>
<tr>
<td>7</td>
<td>Intersecting 46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Intersecting triangles</th>
<th>Size (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>In upper half</td>
<td>357</td>
</tr>
<tr>
<td>In lower half</td>
<td>275</td>
</tr>
<tr>
<td>Total SSC</td>
<td>14487</td>
</tr>
</tbody>
</table>
Memory consumption

Locality of reference
- Temporal locality
- Spatial locality

Garbage collection (GC)
- Automatic memory management system for Javascript
- Non reachable objects
- Reachable objects