WebGL Earth

Petr Sloup[*]
Faculty of Informatics
Masaryk University in Brno
Botanická 68a, 602 00 Brno, Czech Republic
sloup@mail.muni.cz

Abstract

This work describes design and implementation of a new three-dimensional virtual globe that works in the web browsers without the need to install any plug-in, addon, or extension. Implementation itself is realized in JavaScript language with WebGL (*Web-based Graphics Library*) extension that is available in most of the modern web browsers (Mozilla Firefox 4, Google Chrome 10, latest Safari, or Opera 11.50). The key part of the implementation is appropriate texture management and rendering method. This work analyzes existing solutions, evaluates their usability in the specified environment, and proposes new, better suitable structures and algorithms. The main result of this work is an *open-source* application that renders three-dimensional virtual globe inside a web browser in real-time.

Categories and Subject Descriptors

I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Graphics data structures and data types*; I.3.3 [**Computer Graphics**]: Picture/Image Generation—*Display algorithms*

Keywords

HTML5, WebGL, GLSL, Virtual globe, GIS, Mercator projection, Virtual Textures, ClipMapping

1. Introduction

Many web pages need to present some data to the visitors on a map and sometimes even on a 3D model of the Earth (elevation data, 3D buildings, etc.). At the the time of starting this project, the only way to achieve such 3D visualization was by using *closed-source* Google Earth API, which requires the installation of a browser extension.

The goal of the *WebGL Earth* project is to provide an alternative *open-source* solution implemented in JavaScript and readily available in all WebGL-enabled browsers. The result of this work is a platform-independent 3D application.

WebGL is a JavaScript extension defined by [1] (sometimes considered to be part of HTML5) that extends the capability of web browser to be able to generate three-dimensional graphics. It is based on OpenGL ES 2.0 and allows the web developers to control low-level graphics rendering process and even program GPUs with GLSL shaders [2].

The most technically challenging part of the visualization process is the handling of very large textures (real data consists of up to 2 billion × 2 billion pixels[1]).

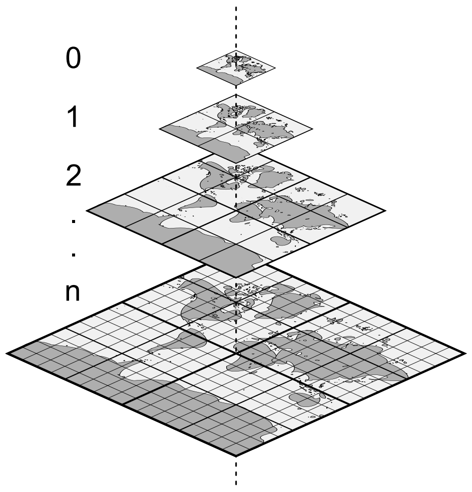

Figure 1: Mercator tile pyramid

Map data itself are usually divided into smaller images (usually with the resolution of 256 × 256 px) to provide granularity and support sophisticated data-management algorithms. On top of this data, we can easily build the so-called "tile pyramid" (figure 1), where each layer consists of four times more tiles than the previous one while covering the same area. This means that each level has twice as much detail in each dimension as the previous level.

Such prepared *tilesets* are freely available from various projects such as *OpenStreetMaps* (up to level 18) or *Bing Maps* (including aerial imagery).

[*]Bachelor degree study programme in field of Applied Informatics. Bachelor's thesis supervisor: Petr Přidal, Faculty of Informatics, Masaryk University in Brno.

Sloup, P. WebGL Earth. Information Sciences and Technologies Bulletin of the ACM Slovakia, Special Section on the ACM Student Project of the Year 2011 Competition, Vol. 3, No. 4 (2011) 53-54

[1]With 23 zoom levels and 256×256 px tiles — $(2^{23} \times 256)^2$

2. Real-time texturing methods

To deal with the large amount of data, we had to implement some tile management system, that would take care of dynamic tile streaming, caching, and real-time lookup in a GLSL shader program during rendering.

Because of really large amount of data, JavaScript performance properties and the fact that WebGL is missing some advanced features available in standard OpenGL, existing methods (such as Virtual Texturing (as described by Mittring [3]) or ClipMapping [4]) proved to be impossible or not optimal to use. We therefore designed new, derived methods that are more adequate for our needs.

2.1 TileBuffer with binary lookup

The main problem of Virtual Texturing is it's low scalability. Instead of having an enormous lookup table where most of the space is empty, we tried a reverse approach. We introduced *metadata* that describes the content of each "slot" of the buffer (in context of this method called the *TileBuffer*).

The metadata are small enough to be repeatedly sent into the vertex shader. By going through the metadata, the shader determines in what slot (if any) the needed tile is stored. We can easily reduce the complexity of the tile lookup operation to logarithmic by presorting the metadata and performing the *binary search.*

This method is usable to solve the specified problem, but the amount of instructions executed in the vertex shader increases with TileBuffer size and with 8×8 tiles the rendering already takes too long on low-end devices.

2.2 ClipStack

ClipMapping proved to be an interesting method of very large texture management. By introducing several modifications, improvements and simplifications we get a new method, suitable for implementation in JavaScript and WebGL, that we call the *ClipStack*.

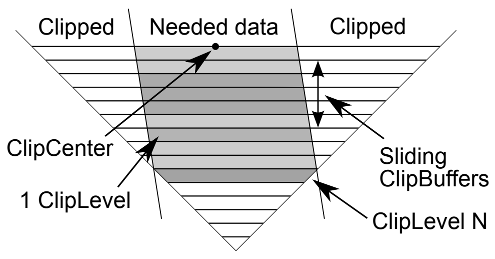

Figure 2: ClipStack

A ClipStack is a collection of *ClipLevels* (figure 2). Each ClipLevel represents a continuous subset of one level of the tile pyramid.

Although having all the same size, each ClipLevel covers twice as large an area as the previous one. It would be an unnecessary waste of computational power to fall back too many times – the shader would get too complex, too much data would have to be streamed, and individual executions would potentially diverge too much. We limit the number of *active* ClipLevels that are actually used during single frame rendering to a constant value. We use 3 active ClipLevels with the resolution of 2048×2048 px (8×8 tiles, 256×256 px each) in our implementation. It proved to be enough to cover a very large area making it really hard to see "uncovered" geometry even inside viewports with a resolution close to 1920×1080 px.

Full description of the ClipStack and it's implementation details and additional optimization techniques can be found in the bachelor's thesis [5].

3. Implementation

In order to visualize the Earth, we also need properly optimized geometry onto which to map the texture data. To speed up the rendering process, we need polygon edges to be aligned with tile edges. It would be impossible to create and store a triangle mesh that would cover the whole Earth at once.

Therefore, we developed a specialized structure called the *Segmented Plane*, which represents constant amount of geometry data. These data are dynamically warped to form the needed subset of Earth's surface. This way, we effectively moved geometry generation problem from CPU to GPU's shader units.

4. Conclusions

Practical results of this work were released as an open-source project called *WebGL Earth*[2][3] and it already has contributors from both academical (*Universitat Politècnica de València* – Spain) and commercial areas (*CampToCamp S.A.* – France and Switzerland; *BIMserver.org* – Netherlands; *National Oceanic and Atmospheric Administration (NOAA)* – USA).

Principles described in this work were also already extended to allow for free-look camera and even 3D terrain rendering.

Important part of the project is also the WebGL Earth JavaScript API, which enables web developers to easily include virtual globe into their own websites.

References

[1] Khronos WebGL Working Group. WebGL Specification.

[2] Khronos Group Inc. The OpenGL ES Shading Language.

[3] M. Mittring and Crytek GmbH. Advanced Virtual Texture Topics. In *ACM SIGGRAPH 2008 classes*, SIGGRAPH '08, pages 23–51, New York, NY, USA, 2008. ACM.

[4] C. C. Tanner, C. J. Migdal, and M. T. Jones. The Clipmap: A Virtual Mipmap. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 151–158, New York, NY, USA, 1998. ACM.

[5] P. Sloup. WebGL Earth. Bachelor's thesis, Faculty of Informatics, Masaryk University, 2011.

[2]Live demo: `http://www.webglearth.com/`

[3]Developer info: `http://www.webglearth.org/`