Progressive Streaming of Compressed 3D Graphics in a Web Browser

Guillaume Lavoué*
Université de Lyon, CNRS

Laurent Chevalier
VELVET

Florent Dupont
Université de Lyon, CNRS

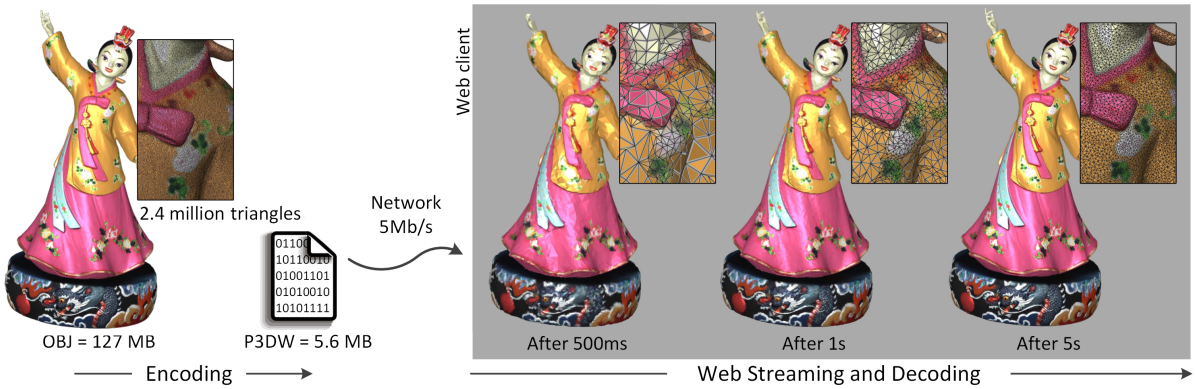

Figure 1: *On the server side, our compression algorithm transforms the triangle mesh into a very compact P3DW file, which can then be streamed, decompressed and visualized progressively on a Web browser. There is no latency even for huge meshes and low bandwidth.*

1 Introduction

The introduction of the WebGL API for rendering 3D graphics within the browser has boosted the development of 3D Web applications. However, delivering 3D Web content without latency remains a challenging issue, not yet solved. An efficient way to remove the latency is to compress the 3D content in a way that allows streaming and progressive decoding (i.e. by generating levels of detail). Such kind of progressive approach also makes it possible to adapt the data to different networks and client hardware. A huge number of mesh compression algorithms have been introduced by the scientific community these last 20 years; unfortunately as highlighted in [Limper PG2013; Lavoue Web3D2013], they are more suited to classical desktop applications than to lightweight Web-based environments. In particular, most of these techniques have ignored a critical factor which is the decoding complexity.

In this context, we introduce a solution for fast progressive streaming and visualization of compressed 3D graphics on the Web. Our approach is based on two main features: (1) a dedicated progressive compression algorithm especially suited to Web-based environments. It produces a compact binary compressed format which allows very fast transmission as well as progressive decoding with levels of detail. (2) a plugin-free solution for streaming, decoding and visualization by the Web browser, which relies on an optimized parallel JavaScript/WebGL implementation.

Our system allows instantaneous interactive visualization by providing a good approximation of the 3D models in a few milliseconds even for huge data and low-bandwidth channels. Experiments and comparison with concurrent solutions for 3D web content delivery demonstrate its excellent results in terms of latency, adaptability and quality of user experience.

*e-mail:glavoue@liris.cnrs.fr

2 Overview

Encoding. The main idea of our progressive compression algorithm is to represent the 3D mesh by a simple coarse model (low resolution) followed by a refinement sequence enabling its incremental refinement until the highest resolution. Our encoding process is based on iterative simplifications. At each simplification step a set of vertices are removed and the data necessary for their insertion, at the decoding, are recorded; these data contain connectivity (vertex valences), geometry (vertex coordinates quantized and predicted) and other attributes (e.g. colors, normals). These data are then fed to an entropic coder. Special care has been taken to minimize the decoding complexity.

Decoding. The decoding is fully written in JavaScript and WebGL; basically, three processes run in parallel: (1) The *binary reader* which decodes the connectivity, geometry and attribute information in a streamed way. (2) The *LoD decompressor* which constructs the next level of details (LoD), starting from an already decompressed LoD and using the decoded information. This component computes the necessary geometrical and topological operations; for this task we introduced a JavaScript HalfEdge data structure. (3) The rendering and user interaction management. Our implementation is multi-threaded by relying on *Web Workers* and *Array Buffers*, as well as asynchronous rendering. Special care has been taken to minimize the use of the garbage collection which is critical in JavaScript.

3 Results

Firstly, our approach provides excellent compression rates, better than existing Web-based tools (OpenCTM, Google WebGL-Loader). Secondly, compared to these tools, our approach allows for progressive decoding and therefore makes possible rapid visualization of coarse versions of the 3D data. Thirdly, The JavaScript decoding is very fast (7 seconds to decompress the Happy Buddha model - 1 million faces). These advantages make our approach particularly efficient for mid-low bandwidth scenarios of 3D Web visualization. It improves on our previous work [Lavoue Web3D2013] both in terms of compression rate ($\times 2$) and decoding time ($\times 3$). The accompanying video provides comparisons with OpenCTM, WebGL-Loader and the recent POP Buffer [Limper PG2013].