Streaming textures with progressive JPEG

Stefan Wagner

June 21, 2012

1 Introduction

WebGL is capable of rendering complex scenes and high poly models. Those complex scenes are usually using many 3D models and textures. A single level of a 3D game can take up to over 200 MB. All this data has to be transferred to the client before he can even start the game. With existing compression methods for 3D models it's possible to reduce the size of the model data. For a typical 3D game model like shown in figure 1, the size of the RAW vertex attributes can be reduced from 795 KB (if send uncompressed as OBJ file) down to 37 KB (if using the MG2 compression method from OpenCTM). See table 2 for further comparisons of the compression formats OpenCTM [3] and WebGL Loader [1].

2 Streaming

Using one of those compression formats reduces the size of the model data down to a reasonable amount. But a typical 3D model not only consists out of vertex attributes but also has some materials assigned. The model in figure 1 consists out of 6 different materials. Each material has one diffuse texture. All materials combined take up 395 KB. The diffuse textures from the materials are already compressed with JPEG. It would be possible to further decrease the file size by decreasing the resolution or quality of the JPEG images but this would also decrease the quality overall. Another option left, is to stream those textures and show a progressive preview while streaming. JPEG supports a progressive mode, in which data is compressed in multiple passes of progressively higher detail. This is intended for large images which will be displayed while downloading over a slow connection, allowing a reasonable preview after receiving only a portion of the data [5]. As the *Image* element of JavaScript doesn't support the *onProgress* event we can't simply listening for the progress event. The image has to be transfered with a XHR request which supports this event. The mime type of the request has to be overwritten to allow binary data transport. The binary data of the image has to be encoded with base64 and applied to a texture. While this approach is working in Firefox 12, for some reasons (maybe because of the introduction of libjpeg-turbo) it doesn't work in Chrome 19. See code example 1 for this approach. The result can be seen here.

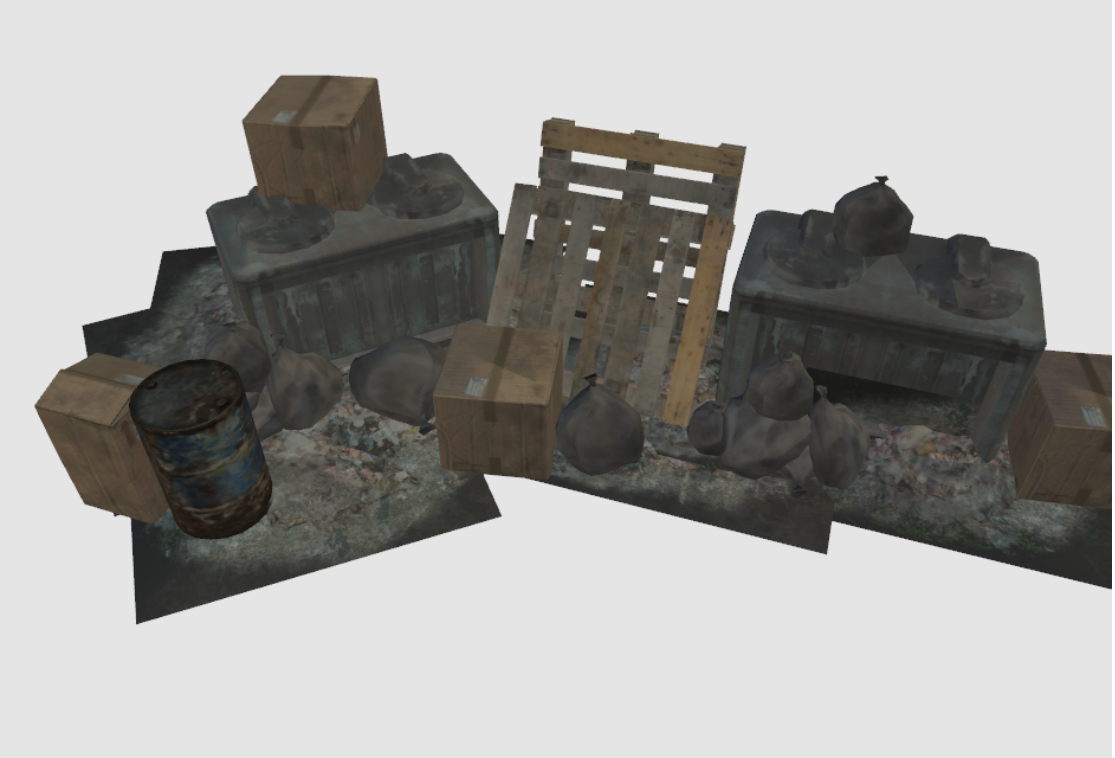

Figure 1: Trash in a corner of a urban city. This 3D model is taken from the Cryengine FreeSDK [2] (Copyright Crytek)

Format	RAW	GZIP
OBJ	795 KB	171 KB
JSON	771 KB	185 KB
BINARY	200 KB	88 KB
OpenCTM MG1	91 KB	91 KB
OpenCTM MG2	37 KB	37 KB
WebGL Loader	68 KB	46 KB

Figure 2: Comparison of the compression formats OpenCTM and WebGL Loader applied at the model from figure 1. Note that OpenCTM always compresses the files with LZMA hence the same file size for RAW and GZIP.

```
xhr.addEventListener("progress", updateProgress, false);
xhr.overrideMimeType('text/plain; charset=x-user-defined');
xhr.open('GET', src);
xhr.send();

...

function updateProgress(e) {
  var t = (xhr.mozResponseArrayBuffer || xhr.mozResponse || xhr.
      responseArrayBuffer || xhr.response);
  image.onload = function() {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE,
        image);
  };
  image.src = "" + base64Encode(t);
}
```

Listing 1: Downloads a JPEG image and listen for the *onProgress* event. If this event occurs the JPEG image gets encoded as base64 and copied over to a WebGL texture.

For Chrome it's necessary to render the image into a 2D Canvas element. A typed array can be read out from this canvas element and later be copied to a WebGL texture. The relevant code snipped is listed in code example 2. The live example can be seen here.

Using this approach it's already possible to stream progressive JPEGs, but the approach also introduces stuttering each time a JPEG image gets decoded. This problem occurs because all operations are executed in the main thread of the application. Even applying the texture in small 128x128 chunks for each frame doesn't remove this stuttering. The decoding of a 2k JPEG image using the *Image* element takes around 50 ms, this is too much to keep a constant frame rate of 60 fps.

Web workers allow one or more JavaScript threads running in the background. They do not have direct access to the DOM and have to communicate by message passing. As the *Image* element is part of the DOM it's not possible to directly decode JPEG images using the *Image* element in a web worker. Instead, the decoding of the JPEG images has to be done completely in JavaScript without using DOM objects. It would be possible to write a custom library that supports the decoding of progressive JPEGs but it's faster to use the project Emscripten [6].

Emscripten converts C to JavaScript code using a LLVM bridge. With this tool it's possible to convert the reference implementation library of the Independent JPEG Group [4]. The converted library can then be used in a web worker. The code example 3 shows the web worker which decodes the JPEG image to a Typed Array and pass it back to the application. A drawback of this approach is the decoding performance of the converted library. It takes around four seconds to decode a 2048x2048 JPEG image. That's 80 times slower compared to the 50 ms of the native JPEG decoder. But as the decoding

```
image.onload = function() {
  var canvas2d = document.getElementById('canvas2d');
  canvas2d.width = 2048;
  canvas2d.height = 2048;
  var ctx2d = canvas2d.getContext('2d');
  ctx2d.drawImage(image, 0, 0);

  //copy content to UInt8Array
  var canvaspixelarray = ctx2d.getImageData(0, 0, canvas2d.width, canvas2d.
      height).data;
  uploadTextureData = new Uint8Array(canvaspixelarray.length);
  for (var i=0;i<canvaspixelarray.length;++i) {
    textureData[i] = canvaspixelarray[i];
  }

  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texImage2D (gl.TEXTURE_2D, 0, gl.RGBA, canvas2d.width, canvas2d.height
      , 0, gl.RGBA, gl.UNSIGNED_BYTE, textureData);
};
image.src = "" + base64Encode(t);
```

Listing 2: Writes the content of the base64 encoded JPEG image into a 2D canvas. After that the content of the canvas gets written into an UInt8Array which is later uploaded to a WebGL Texture.

process doesn't run in the main thread of the application, the whole application feels overall faster without stuttering. The converted library has a file size of 85 KB (GZIP) which has to be downloaded before the application can decode JPEGs in a web worker. This approach is therefore only useful if the application uses many large JPEGs. The live example can be seen here.

3 Summary

The streaming of progressive JPEG only makes sense if the decoder is faster as the communication link and if there are many large JPEGs to be transfered. With the web worker approach the decoding time for one 2k JPEG image takes around four seconds. If the bandwidth of the communication link is capable of downloading the JPEG in less than four seconds it's better to just download the image without a progressive preview. Further enhancements of the JavaScript engines could speed up the decompression process. Also it could be possible to compile another JPEG decoding library like libjpeg-turbo which performs faster as the reference library. If the stuttering is acceptable for the certain use case (e.c. the scene is mostly static) the approach using the canvas element could also be used. The decoding of a 2k JPEG takes around 50 ms using the native approach. This should be faster than most common connection speeds.

```
importScripts('djpeg.js');
onmessage = function(event) {
  var data = readJpeg(event.data);
  var b = new ArrayBuffer(2048*2048*4);
    var v1 = new Uint8Array(b);
    var i = 0, j = 18;
    while (i < 2048*2048*4 && j < (2048*2048*3)+18) {
        v1[i] = data[j + 2]; // R
        v1[i + 1] = data[j + 1]; // G
        v1[i + 2] = data[j]; // B
        v1[i + 3] = 255; // A
        // Next pixel
        i += 4;
        j += 3;
    }
  postMessage(v1);
};
```

Listing 3: This web worker uses the with Emscripten converted library of libjpeg to parse a progressive 2048x2048 JPEG. The JPEG image gets decoded into a typed array and passed back to the main application.

References

[1] Won Chun. Webgl loader - simple, fast, and compact mesh compression for webgl. Technical report, Google, 2012.

[2] Crytek. Cryengine 3 free sdk. Technical report, Crytek, 2012.

[3] Marcus Geelnard. Openctm - the open compressed triangle mesh file format. Technical report, Marcus Geelnard, 2012.

[4] Independent JPEG Group. libjpeg. Technical report, Independent JPEG Group, 2012.

[5] Wikipedia. Jpeg. Technical report, Wikipedia, 2012.

[6] Alon Zakai. Emscripten. Technical report, Mozilla, 2012.