Blast
A Binary Large Structured Transmission Format for the Web

Jan Sutter[*] DFKI Saarland University
Kristian Sons[†] DFKI Saarland University
Philipp Slusallek[‡] DFKI Saarland University Intel VCI

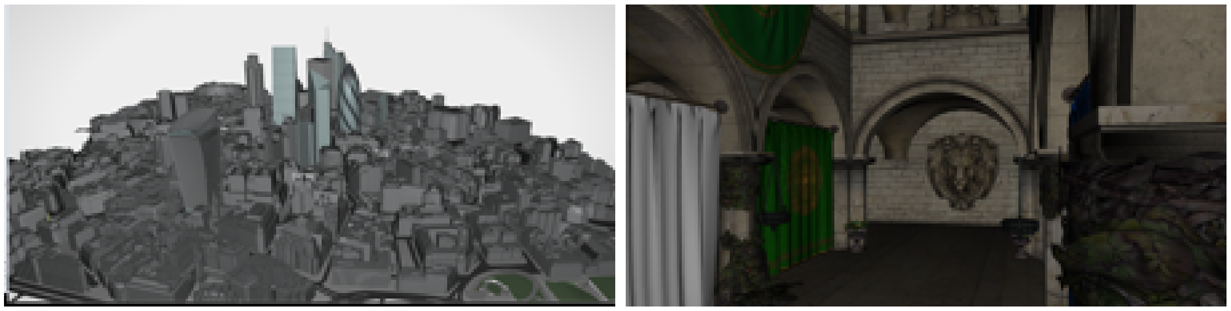

Figure 1: *Left: "London City" (Model courtesy of Vertex Modelling). It consists of 4067k triangular faces has no textures and an overall size of 279 MiB. Right: The "Atrium Sponza Palace" (Model courtesy of Crytec). It consists of 394 meshes, 36 textures and 279k triangular faces with an overall size of 81 MiB. Both rendered in XML3D and streamed with Blast in a single request using the Open3DGC mesh encoder. On a 6 Mbit/s connection the complete transmission took 31.7 seconds ("London City") and 89.5 seconds ("Atrium Sponza Palace").*

Abstract

Recent advances in Web technology, especially real-time 3D content using WebGL, require an efficient way to transfer binary data. Images, audio and video have respective HTML tags and accompanying data formats that transparently handle binary transmission and decompression. 3D data, on the other hand, has to be handled explicitly by the client application. In contrast to images, audio and video, 3D data is inhomogeneous and neither common formats nor compression algorithms have been established for the Web.

Despite the many existing formats for binary transmission of 3D data none has been able to provide a general binary format for *all* kinds of 3D data including meshes, textures, animations, and materials. Existing formats are domain-specific and fixed on a certain set of input data and thus too specific to handle other types of data.

Blast is a general container format for structured binary transmission on the Web that can be used for all types of 3D scene data. Instead of defining a fixed set of encodings and compression algorithms Blast exploits the code on demand paradigm to provide a simple yet powerful encoder-agnostic basis to leverage existing domain-specific solutions and compression techniques. Because streaming is of primary importance for a good user experience Blast is designed on the basis of self-contained chunks to enable JavaScript clients to utilize Web Workers for parallel decoding and to provide early feedback to the user.

CR Categories: I.3.0 [Computer Graphics]: General—;

Keywords: binary transmission, streaming 3d, webgl

1 Introduction

WebGL [Khronos Group 2011a] paved the way for real-time 3D visualizations embedded in websites. Both, imperative [Cabello 2010] and declarative [Web3D Consortium 2008; Sons et al. 2010], solutions exist that enable web developers with little knowledge of 3D to use WebGL. All these solutions face the challenge of transferring large structured and inhomogeneous binary data.

JavaScript client applications have two ways to communicate with a server, XMLHttpRequest (XHR) [W3C 2014b] and Web Sockets [Fette and Melnikov 2011]. The latter allows the client and server to stream data bidirectional but requires a binary protocol understood by both ends. However, no general protocol for the transmission of 3D data has been established yet. XMLHttpRequest, on the other hand, uses the HTTP protocol. It is the typical mechanism used to resolve external resources. It supports binary requests based on Array Buffers [Khronos Group 2011b] but has currently no support for streaming binary data [W3C 2014b].

The transmission of 3D data to the client is usually a two step approach. First, the scene structure is sent to the client. Declarative approaches such as XML3D [Sons et al. 2010] and X3DOM [Behr et al. 2012] provide the structure as part of the HTML document, while other solutions such as threejs [Cabello 2010] use glTF [Robinet et al. 2012] or similar JSON based descriptions. This scene structure contains references to all external resources of the scene, including meshes, images for textures, material descriptions

*e-mail:jan.sutter@dfki.de
†e-mail:kristian.sons@dfki.de
‡e-mail:philipp.slusallek@dfki.de

and animations. All external resources are then resolved on the client-side.

Two solutions exist for handling external references using XHR. First, requesting each resource separately and transferring it using JSON, XML, binary XHR [Robinet et al. 2012] or domain-specific binary formats [Geelnard 2009; Lavoué et al. 2013; Bischoff and Kobbelt 2002; Limper et al. 2013]. Second, aggregating multiple resources together in an XML document using fragment identifiers [Berners-Lee et al. 2005] to address subordinate resources. The former allows the application to provide early feedback to the user as soon as a resource is received and, moreover, to utilize unstructured binary domain-specific transmission formats. This approach however, results in an enormous amount of requests for large 3D scenes. Using XML for structured transmission, on the other hand, reduces the number of requests but has no support for binary data and requires the whole document to be available before user feedback is possible. The Web, thus, is in need of a binary transmission format for 3D data that allows for domain-specific compression techniques and aggregation of multiple resources equally.

As outlined in [Sons and Slusallek 2013] every binary transmission format for the Web has to be designed to reduce the number of requests, handle network characteristics and facilitate client applications in providing a good user experience.

Bandwidth and latency are the two major properties of network connections. The former, even though steadily increasing, is still a bottleneck for large data transmissions. In particular mobile connections still suffer from low bandwidth. Furthermore, connection characteristics can change as mobile devices move between cells and even fast mobile connections often have contractual limits on their overall traffic. Bandwidth limitations can be handled using compression techniques to reduce the overall size of a transmission. For the remainder of the paper we will use encoding as an umbrella term for binary encoding including compression.

Current domain-specific formats allow for compression and binary transmission but lack the internal structure to transfer more than a single resources of a specific type. Thus, the number of requests these formats require increase with the number of resources within a scene. Then latency becomes the limiting factor if bandwidth is sufficient but the communication overhead carried by each request outweighs the actual transmission time.

As a result any transmission format for binary data has to be able to deal with bandwidth limitations using compression but also has to provide means to transfer multiple resources in a single request. More importantly, failing to do so inevitably affects the user experience. Internet users are accustomed to responsive web applications and sensitive for delays. Even 10 seconds of wait time result in a serious degradation of the user experience [Nielsen 1993]. It is, thus, crucial to provide not only continues feedback but early interaction possibilities with the system even though some transmission may still be in progress. To facilitate this the format has to be streamable and must provide early access to usable chunks that can be handled in parallel to the main thread of the application.

2 Requirements

We derive the following set of requirements for binary transmissions as a consequence of these factors. They are based on those stated in [Sons and Slusallek 2013] and [Doboš et al. 2013].

Efficient Efficient handling in JavaScript is important and a reason why JSON has become so prominent as transmission format. Handling binary data in JavaScript is only efficiently possible if the format facilitates the use of Typed Arrays. They are also required as the input for APIs such as WebGL, WebCL [Khronos Group 2014], or River Trail [Herhut et al. 2012] and the first choice for any data processing in JavaScript [Klein et al. 2012]. Additionally, Typed Array data can be transfered with zero copying overhead between a Web Worker [W3C 2012] and the main thread.

Structured Requesting each resource individually can induce high latency. A binary format therefore has to be able to send multiple resources in a single request similar to XML. The client, thus, needs the ability to address individual resources inside the transmission using fragment identifiers.

REST Compatible Even though not a direct consequence of the key factors we require a transmission format for 3D data to be compatible with web-services following the REST paradigm [Fielding 2000]. 3D content in the browser is in the transition from a conventional static file based approach to more sophisticated service oriented approaches [Doboš et al. 2013; Arnaud 2011]. REST APIs do not explicitly indicate the transmission format as part of the URL but use content-negotiation to determine the best suitable delivery format. To enable content-negotiation it is important that all possible formats use the same syntax to address subordinate resources based on fragment identifiers. The binary transmission format should support the established fragment identifier syntax of XML.

Schemaless The format should neither impose a special schema on the data nor should it only support a specific set of attributes of certain 3D scene relevant parts. Existing domain-specific formats are not only unstructured but impose a particular signature on the input data. Even though, mesh data accounts for the majority of a 3D scene's data and are, thus, the most important part a binary transmission format has to handle, the contribution of animations, textures, and materials to the overall size of a scene should not be underestimated. A binary format for 3D data therefore has to be able to transport this data equally well. Moreover, the ability to handle the very inhomogeneous nature of material properties and input data implies a generic approach no existing binary format provides yet.

Encoding-Agnostic Data encoding is important to minimize the size of the transported data and to overcome bandwidth limitations. The format, however, should not dictate any encodings or compression algorithms. Generic encoders such as gzip achieve reasonably good compression ratios on many types of data, especially textual data containing lots of redundancy. To obtain the best possible compression ratios with minimal amount of processing time the algorithm has to be specifically designed for the data. For a 3D transfer format this implies that encoders must be definable on a per-data basis.

Streamable Streamability is a direct consequence of the requirement to send multiple resources at once and the necessity to provide early and continues feedback to support a good user experience. Without streaming, a single request containing all resources of a 3D scene would be required to be received entirely before data processing can start, ruining the user experience.

Based on these requirements we propose *Blast*, a streamable, encoder-agnostic container format for binary data transmissions on the web.

3 Blast

As a container format Blast supports both, a single binary resource per request and sending multiple resources in a structured transmission similar to XML. Moreover, Blast is streamable. Since streaming is currently not supported by XHR [W3C 2014b], Blast is designed to work with the upcoming Streams API [W3C 2014a], which allows for XHR based binary streaming. Additionally, Blast can be streamed using Web Sockets [Fette and Melnikov 2011].

In fulfillment of the stated requirements for a binary transmission format Blast is build around the following key concepts:

Generic Approach Even though designed specifically with 3D data in mind, Blast does not impose any restriction on the transported data. Blast is typeless by design (see Section 3.1).

Code On Demand To support custom encoders on a per data basis Blast uses a code on demand [Fuggetta et al. 1998] approach to supply clients with the necessary decoding procedure. (see Section 3.2).

Identification and Addressability Structured transmission requires means to address resources within a transmission. To be compatible with XML Blast uses fragment identifiers based on a path description language to address resources in a transmission (see Section 3.3).

Chunked Based Streaming Binary streaming in Blast is achieved using self-contained chunks that can be independently processed in parallel (see Section 3.5).

Transparent Decoding Blast serves as a transparent layer for the transmission and decoding such that client and server can simply exchange their data structures disregarding any transmission details (see Section 3.6).

3.1 Generic Container

Blast is a generic container format and can be used for the binary transmission of any key-value data structure containing any types of data. Blast is type-agnostic and only handles byte data. The byte data generated by an encoder is an atomic value in a Blast transmission and will be passed as-is to the respective decoder. It can be a flat key-value structure describing a single mesh, a collection of such mesh definitions, a single vertex buffer of a mesh, or a complex hierarchical key-value structure. Every Blast transmission contains an arbitrary number of these values.

Each value addressable by the Blast addressing mechanism (see Section 3.3) can be encoded individually. Blast transports all necessary information for the reconstruction of the original key-value data structure.

Since Blast transmissions do only transport byte data the endianness of the encoded data is the responsibility of the sender and the used encoders that have to handle multi-byte data properly. Endianness of a Blast transmission should be specified during content-negotiation as a parameter of the media-type, e.g. `application/blast;endianness=little`.

3.2 Custom Encoders – Code on Demand

Code on demand is a term that describes technologies that transfer source code to be executed on a remote end. All websites that use JavaScript use the code on demand idiom whenever they send JavaScript to the client.

Custom encoders are an important requirement for a general container format. Encoders in Blast can operate on any part of the key-value structure sent. There is only one requirement: they have to produce binary data that allow for a decoding procedure to reconstruct the original data. This encoded binary data is then handled as a value and the information where this value was located in the overall key-value structure is specified using a path expression (see Section 3.3). Additionally, for each value a unique identification of the corresponding decoding procedure has to be provided using a URL. We will show in Section 4 that for 3D data containing meshes as values the overhead induced by these URLs is negligible.

The client can implement a decoder utilizing technologies such as Xflow [Klein et al. 2012], WebCL [Khronos Group 2014], RiverTrail [Herhut et al. 2012] or native functionality and use the URL for identification purposes only. However, in case the client does not have an implementation available it can use the URL to request one on demand. This allows the use of any encoder without requiring the client to implement a decoding procedure (see Section 3.6).

The implementation requested on demand has to conform to a certain interface to be usable by the client. It has to expose a function `decode` taking two parameters, the encoded bytes and the endianness of the transmission. For this approach to function properly it is the responsibility of the server to ensure that a suitable implementation is available at that specific URL. Content-negotiation can further be used to serve more than a single implementation. Depending on the client implementations for different languages can be made accessible.

In contrast to the code on demand that web applications use, the approach in Blast may result in the execution of JavaScript from domains unrelated to the current web application. This imposes a potential security threat, even though JavaScript is already sand-boxed in the browser. Potentially malicious code can harm the client application by manipulating or reading sensible information from the DOM or flooding the console. To mitigate this potential we propose to exploit the secure execution context of Web Workers [W3C 2012]. These worker threads, already proposed for parallel decoding, have no access to any sensible part of a web application and communication with the main thread is limited to a single channel, the message queue. Running worker threads can be monitored and watch dogs can be deployed to terminate them if necessary.

3.3 Structured Transmission

Structured transmission requires the client to be able to address individual resources inside the transmission for further processing. XML uses fragment identifiers. Other formats require the client to explicitly know the enclosing structure to extract individual resources as they do not provide a standardized addressing mechanism.

In Blast the structure of a transmission is specified using *JPath* expressions. Blast transmission are always structured and JPaths are used to identify the original position of the encoded value inside the original key-value data structure. JPath expressions only define the structure between values, the structure inside the values is the responsibility of the encoder and decoder. The differentiation between a single resource and a collection of independent resources is determined by the server by generating the necessary fragment identifiers on the URLs.

JPath expressions are slash delimited property paths, or key paths respectively. JPath is inspired by JSONPath [Goessner 2007] and JSONPointer [Bryan and Nottingham 2013] and borrows its syntax from the latter, which is compatible with the XML fragment identifier syntax.

Consider the following key-value map representing a scene in JavaScript:

```
{
  rootNode: {
    children: [{
      name: "Sample Node",
      mesh: 4
    }]
  }
  meshes: [... { // an array of meshes
    name: "firstMesh"
    position: new Float32Array([...]),
    normal: new Float32Array([...]),
    texcoord: new Float32Array([...]),
    index: new Uint16Array([...]),
  } ...]
  //... possible other properties
}
```

The JPath expression `/meshes` would specify the entire array of meshes. `/rootNode/children/0/name` would specify the string `"Sample Node"`. JPath allows for both, property notation and bracket notation to specify array elements, for instance the entire first mesh of the array can be specified either using `/meshes/0` or `/meshes/[0]`. The property notation is used for compatibility with the XML fragment identifier syntax, while the bracket notation is used to explicitly indicate that the type of the parent element is an array. This is important for the transparent decoding that Blast provides (see Section 3.6).

3.4 Structure of a Blast Transmission

Every Blast transmission consist of two major building blocks, the preamble and a sequence of chunks terminated by the null chunk. The entire structure of Blast, as depicted in Figure 2, is designed to be streamable, to facilitate the parallel decoding on client-side, and to reduce management overhead on both ends.

The preamble, shown in Figure 3, is the very first part of every Blast transmission and carries information for all following chunks. Its first 4 bytes (ASCII BLST) are for identification purposes and detection of the endianness of the stream. Following are 4 bytes to be interpreted as an unsigned integer that indicate the overall size of the preamble. The next 2 bytes each interpreted as an unsigned byte specify major and minor version number of the Blast stream format. The last part is a NUL terminated ASCII string for the URL that identifies the decoding procedure necessary to reconstruct the header information of each of the following chunks. Because this URL can be arbitrary long in theory the size of the preamble is not limited and its overall size has to be specified.

3.5 Streaming – Chunked Transmission

Blast can be used in three ways. First, sending a single resource per request. Second, sending multiple resources in a structured way within a single request to reduce the overall latency. It can transfer multiple binary resources ranging from a single mesh to all data of a scene in a single request. More importantly, Blast can be streamed.

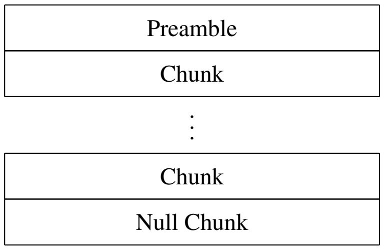

Figure 2: *General structure of a Blast stream.*

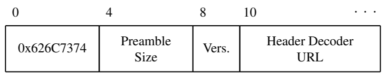

Figure 3: *The preamble of a Blast stream. Size indications are given in bytes.*

This enables a client to request all resources of a scene in a single request while still providing early user feedback and interaction possibilities. This flexibility is achieved using a chunk based approach (see Figure 4). Chunked container formats are widely used for multimedia data [Murray and VanRyper 1994; Matroska 2005; Electronic Arts 1985].

Chunks in Blast can transport any number of values. As long as XHR does not support streaming, a single chunk can be used to transfer a single resource or multiple resources. Web Socket connections, on the other hand, can use multiple chunks for streaming. The number of values, i.e. the size, of each chunk can be determined by the server depending on connection properties and the structure of the data.

Instead of using a single description for the structure information of the entire data, Blast uses self-contained chunks. While a single description for the structure may allow for more efficient encoding of the structure information it imposes a specific order on the data transported and, moreover, makes it necessary that all data is already available to define the structure.

In Blast each chunk consists of two parts: a header and payload. Chunks in Blast are always self-contained and independently processable. They never transport only parts of the data necessary for reconstruction nor do they depend on other chunks in a stream. All information necessary is contained inside the chunk itself or was received in the preamble.

This does not only enable parallel decoding using Web Workers, but allows the receiver to discard already processed chunks immediately. Parallel decoding is important to not block the main thread during CPU intensive decoding. More importantly, autonomous chunks reduce management overhead, enable the remote end to send chunks as soon as possible or in any order the client requests, and to cache and reorder them during later requests.

Each chunk starts with 4 bytes to be interpreted as an unsigned integer to indicate its overall size. This allows the client to reserve the necessary memory for the chunk at once avoiding costly reallocations during processing. Following are 4 bytes to specify the size of the chunk header. The required header contains information about the structure of the data transported in the payload. It is an array of key-value maps, one for each value transported by this chunk. This header is encoded and has to be decoded using the procedure indi-

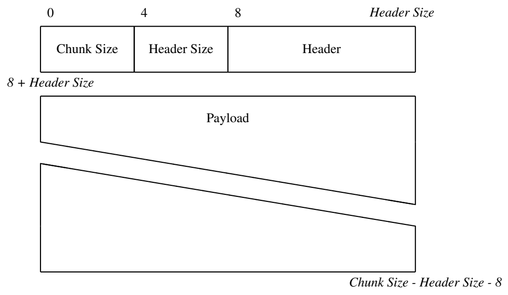

Figure 4: *The structure of a Blast chunk. Size indications are given in bytes.*

cated in the preamble of the stream. The remainder of the chunk is its transported payload.

Each chunk header contains structure information about the values encoded in the payload. Since Blast is designed to allow every value to be encoded in a user definable way every entry inside the header contains a unique identification of the necessary decoding procedure to reconstruct the original data. Each entry additionally contains a JPath expression that uniquely identifies the value's original position in the key-value data structure sent. The following values are transported in a header entry:

offset The offset in bytes into the chunk's payload at which this value is located.

size The size in bytes of the encoded data.

path The JPath expression that uniquely identifies the value (see Section 3.3).

metadata An optional key-value map that can hold any information of any size and can be used to transport additional information about a value. This meta information can, for instance, be used to specify additional semantics or application specific type information if necessary.

decoding specification The decoding specification indicates how the bytes specified by offset and size have to be decoded. This specification is always a URL that the client can use to uniquely identify the decoding procedure or to request an implementation on demand.

A special chunk in every Blast stream is the last one, the null chunk. Since every Blast transmission can contain an arbitrary and unspecified number of chunks this null chunk is used to signalize the end of the stream. It is a chunk without payload and thus without header information and is 4 byte in size as a chunks overall size includes the 4 bytes specifying that size.

3.6 Transparent Decoding

The dichotomy between a resource's internal structure and the format in which the resource is transfered is an important aspect of the Web. Resources have a structure, e.g. meshes have buffers, materials have parameters and shaders. Content-negotiation defines the process in which a client and a server determine the used transmission format. This format, however, does not necessarily reflect that inherent structure of the data sent. A client's main interest is the data not the format in which it is transfered. For the client the transmission format is only of importance because it has to deserialize the actual data. One reason for JSON's prominence is that deserialization is natively supported in JavaScript and that encoding and compression using the HTTP intrinsic gzip compressor [Fielding et al. 1999] is handled transparently.

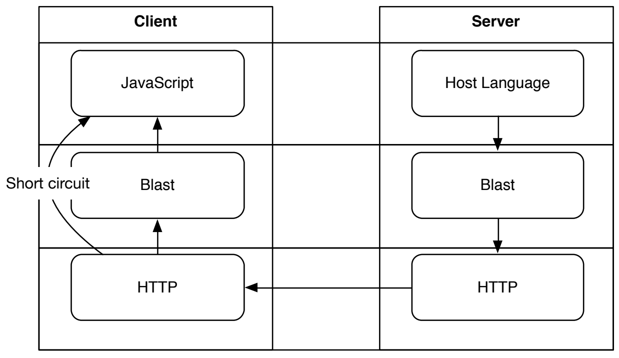

Figure 5: *Blast serves as a transparent decoding and transmission layer while still providing the client ways to short circuit.*

As a consequence of the code on demand approach together with the unique identification and structure specification that a JPath identifier provides Blast can be used as a transparent layer for data transmission (see Figure 5). This layer can parse incoming chunks, extract all transported values, download the necessary decoding functionality and reconstruct the sent object using the value's JPath expressions. The precise specification of a JPath expression enables the reconstruction of the complete structure of the original key-value map. The differentiation in JPath regarding array indices using bracket notation allows the receiver to recursively create non existing parent values along the path. As soon as the final chunk is received the Blast layer can inform the client that the object is completely reconstructed. Moreover, because decoders are specified uniquely using URLs the client can always identify a decoder and short-circuit the transparent decoding layer to use its own implementation.

The transparent decoding layer provided by Blast enables the client application and the server to disregard transmission details of the exchanged data structure. Moreover, it is now possible for the server to choose the encoding based on connection properties and utilize encoders that are unknown to the client. Intensive compression can outweigh transmission time on high bandwidth connections and increase latency while mobile connections may require such compression because of bandwidth and traffic limitations. Without the code on demand approach choosing an encoder based on these properties is only possible if the client has the required decoding procedure implemented.

4 Results

For the evaluation and benchmarking of Blast we integrated a server-side implementation into XML3DRepo [Doboš et al. 2013] and a client-side implementation into XML3D [Sons et al. 2010]. Furthermore, we integrated two existing encoders, OpenCTM [Geelnard 2009] and Open3DGC [Mamou 2013].

The flexibility of Blast allows requesting scenes in three different ways: individual requests per resource, aggregated requests of multiple resources, and streaming of *all* resources within a single request. We conducted benchmarks with respect to the transmission time of these usages simulating a broadband connection with 6 Mbit/s download and 512 Kbit/s upload bandwidth and a 1 Gbit/s local area network using a software bandwidth shaper. The bandwidths of the broadband connection are chosen to be conservative in regard to the average available bandwidth of todays connections.

The benchmark results for the transmission of two large scenes are shown are shown in Table 1: The "Atrium Sponza Palace" scene from Crytek[1] and the "London City" from Vertex Modelling[2] (see Figure 1).

The benchmarks show that the influence of the required number of requests is less significant for a 6Mbit/s connection: the limiting factor is the bandwidth itself, thus, the encoding becomes important. In contrast to a high bandwidth connection the reduction in the overall transmission time by aggregating and streaming resources is less significant. However, being able to stream all resources is important: it provides full control over the order of transmitted resources. The time for the first mesh in the "Atrium Sponza Palace"

[1]http://www.crytek.com/cryengine/cryengine3/downloads
[2]http://www.vertexmodelling.co.uk/site/

	Atrium Sponza Palace (81 MiB)				London City (279 MiB)			
			Transmission Time [s]				Transmission Time [s]	
	#Req.	Size [MiB]	6 Mbit/s	1 Gbit/s	#Req.	Size [MiB]	6 Mbit/s	1 Gbit/s
Individual								
JSON	418	72.2	104.7 (0.2, 84.6)	3.8 (0.02, 0.89)	328	136.2	194.9 (0.14)	4.4 (0.08)
OpenCTM		67.3	97.8 (0.04, 83.6)	3.7 (0.06, 0.84)		59.7	86.8 (0.09)	3.5 (0.04)
Blast (OpenCTM)		67.4	97.9 (0.05, 83.9)	3.7 (0.06, 0.85)		59.7	87.8 (0.10)	3.5 (0.04)
Blast (Open3DGC)		63.2	92.0 (0.02, 82.8)	3.6 (0.04, 0.81)		22.3	34.3 (0.05)	2.9 (0.02)
Aggregated								
XML	103	72.1	102.5 (1.2, 87.3)	1.2 (0.09, 0.93)	73	136.2	193.4 (0.27)	1.8 (0.12)
Blast (OpenCTM)	84	67.3	97.1 (0.9, 83.8)	1.0 (0.07, 0.79)		59.6	86.7 (0.23)	1.5 (0.10)
Blast (Open3DGC)		63.1	91.3 (0.8, 77.4)	1.0 (0.07, 0.74)		22.3	33.1 (0.13)	0.8 (0.09)
Streamed								
Blast (Open3DGC)	1	63.1	89.5 (0.06)	0.6 (0.03)	1	22.3	31.7 (0.06)	0.2 (0.03)

Table 1: *Benchmark for the transmission of the "Atrium Sponza Palace" (left) and the "London City" (right) regarding a 6 Mbit/s and a 1 Gbit/s connection. All measurements were conducted 10 times and the median values are given. Except for OpenCTM transmissions, which already deploys an LZMA compression, all formats were further gzip compressed. Transmission times in parentheses indicates the times until the first mesh was received. For the "Atrium Sponza Palace", where textures contribute to the major part of the size, two numbers are given: the first indicates the minimum time, the second the maximum time until the first mesh was received. The used mesh encoder for the Blast transmissions is given in parentheses. For the aggregated resources scenario five meshes are sent within a single request. Since XML does not support binary data textures are still requested separately. Concurrent requests were limited to two following the HTTP/1.1 standard [Fielding et al. 1999]. In general streaming Blast using Open3DGC results in the best performance for all benchmarked settings.*

scene clearly varies for all transmission types, except for streaming. This variation in time is a result of the browser's internal order and scheduling of the requests. There is no guarantee, for instance that meshes are always received before the textures without particular application logic. Blast, on the other hand, allows the server to control the order of the resources.

In contrast to the low bandwidth connection, the benchmark simulating a 1 Gbit/s connection clearly shows the impact the number of requests have on the overall transmission time. Aggregation and streaming result in a significant reduction in the overall time.

Blast, because of its flexible design, can be used in all three scenarios. Its encoder-agnostic design enables the use of existing domain-specific encoders to reduce the size of the transmission. The overhead of Blast for the transmission of individual resources per request is negligible in comparison to domain-specific formats such as OpenCTM. However, while these format lack the ability to transfer multiple resource in a single request, Blast provides this functionality still taking advantage of their compression abilities. The reduction in the number of required requests by aggregating resources considerably improves the transmission time for high bandwidth connections.

Because of the inability to stream binary data using XHR resource aggregation is of great importance. However, using Web Sockets to stream Blast further reduces the required request to a single one still providing the same early feedback and user experience as individual requests for each resource. In general streaming Blast in combination with Open3DGC achieved the best performance in all tested settings. Furthermore, the server has full control over the order of the resources, which allows for a controlled transmission. This can be used to send meshes depending on the client's view into the scene or to interleave meshes and their respective textures to further increase the early visual quality of the rendering.

5 Related Work

The list of existing approaches regarding binary data transmission is huge. Many different formats exist in scientific literature and else where. We will discuss three categories of current approaches, why they only address a subset of the stated requirements and how they differ to Blast yet are not mutually exclusive.

5.1 Document based

Textual transmission formats such as UTF-8 JSON [Crockford 2006] and XML [W3C 2008] are widely used. Both are not bound to any set of predefined attributes and allow for arbitrary structured content. As textual formats none supports embedding binary data directly without using base64 [Josefsson 2006] encoded binary strings. Even though this would allow for a binary transmission inside a structured document base64 encoding necessarily increases the size of the original data and increases decoding time on the client.

JSON and XML both have binary representations. One very common binary JSON format is BSON [10Gen, Inc. 2013]. Developed as the main storage format for mongoDB, BSON is driven by the design requirement for fast traversal. BSON provides data types not natively found in the JSON specification [Crockford 2006], in particular binary blobs. These blobs are the application's responsibility and can be utilized to transport arbitrary binary data inside a BSON document. Its inherent JSON like structure, however, imposes difficulties for streaming. Additional BSON has no support for custom and data specific encoders.

Several competing standards for binary representations of XML exist, for instance XML Metadata Interchange (XMI) [Object Management Group 2013] and FastInfoset (FI) [Telecommunication Standardization Secor of ITU 2005]. The latter is also used for binary encoding of X3D documents [Web3D Consortium 2011]. Binary XML representations utilize dictionary compression for element and attribute names. XMI additionally deploys a deflate compression on the data. FI, on the other hand, allows for custom encodings and domain-specific compression methods similar to Blast. This custom encoding support, however, uses a special schema for identification that does not allow for code on demand approach. Clients always have to provide a decoding routine for any specified encoder.

5.2 Domain-Specific

Numerous solutions for streaming 3D meshes exist [Lavoué et al. 2013; Bischoff and Kobbelt 2002; Limper et al. 2013; Hoppe 1996]. Meshes, however, are only one aspect that has to be handled for a generic transmission format and their domain-specific design makes them not applicable to other kinds of 3D data. Moreover, most of these approaches can exploit Blast chunks for transmission and their implementation has to be done on the application level. Their usage is, thus, orthogonal to Blast.

The Open Compressed Triangle Mesh file format (OpenCTM) [Geelnard 2009] is a domain-specific encoding and compression format specifically design to handle triangulated 3D mesh data. OpenCTM can achieve high compression rates but cannot be used to transfer multiple meshes and is not applicable to other kinds of 3D data. However, OpenCTM, as shown in our results, can be leveraged in Blast to efficiently compress triangular mesh data.

Encoding geometry information inside images is an approach that allows for a very efficient reconstruction on the GPU. Moreover, image decoding is done natively in the browser, circumventing any JavaScript based decoding for the image data. Sequential Image Geometry (SIG) [Behr et al. 2012] extends this approach further. Every vertex attribute is encoded in chunks of 8 bit. These chunks are distributed into sequences of images ordered by relevance to allow simple quantization by omitting less relevant images. Progressive loading can also be achieved if images of a sequence are received in descending order of relevance. One of the issues of SIG is the large number of required requests, i.e. at least one request per vertex attribute (8 bit quantization).

Similar to OpenCTM, Blast is able to exploit the efficiency of SIG while reducing the necessary requests to a single one and furthermore providing a way to send all images in a specific order to allow progressive refinement.

5.3 JSON and Binary XHR

A common approach used, for instance, in XML3D [Sons et al. 2010], X3DOM [Behr et al. 2009], and glTF [Robinet et al. 2012] are unstructured binary XMLHttpRequests using an additional document for structure information. This approach, despite its common usage, increases the number of requests by at least a factor of two as structure and data are separate resources.

glTF [Robinet et al. 2012] is the suggested approach of the Khronos Group to standardize a JSON based 3D scene description format. Every glTF asset is comprised of at least two files, a JSON based description that references multiple binary files. The JSON description contains all information necessary to extract the embedded information inside the binary files.

glTF defines external binary containers for meshes, textures and materials, however, are currently only referenced by URL or embedded as a base64 encoded string. There is no solution towards reducing the overhead in requesting each of these resources in separate request. Compression in glTF is defined using an extension mechanism comparable to those used for functionality in OpenGL and WebGL. Clients either support an extension, in that case they must have an implementation of the decompression algorithm, or they cannot read the file. This approach would allow for different encoders and compressors but it requires the client to support all of those extensions. The current draft standard only specifies the Open 3D Graphics Compression (Open3DGC) extension [Mamou 2013].

Compared to Blast, glTF follows a different design rational and intention. As an asset format glTF aims to provide a standardized format for 3D scenes. Blast and glTF can be combined to provide a standardized 3D scene format and an efficient streamable binary transmission format for all external resources.

6 Conclusion and Future Work

We have proposed Blast, a general container format for binary data transmission for the Web. Though we focused on its ability to transport 3D mesh data it is general enough to be used for all kinds of 3D data. Blast can leverage existing mesh encodings, e.g. Open3DGC and OpenCTM to achieve high compression rates. The code on demand approach and the transparent decoding enables the server to employ the best possible encoding on the data without burdening the client with decoding and transmission details. Blast's streaming design focused around chunked based transmission allows a server to adapt to different features of a network connection by changing the number of values per chunk or the used encoders. The benchmarks have shown that the flexibility of Blast comes with negligible overhead and that its design does not obsoletes existing efforts but provides a solid basis to overcome their inherent limitations while still taking advantage of their strengths. Blast can reduce the number of requests required to a single one while still providing the same early feedback compared to a single request per resource approach.

Based on Blast we can now start the evaluation of advanced requests based on spatial details of a scene including view-dependent and occlusion-dependent requests. In current approaches network characteristics are typically not incorporated into the decision and negotiation of the used encoding or how a resource is transfered. The code on demand approach and the flexible chunked encoding of Blast enables the incorporation of heuristics about connection bandwidth, round trip time and user agent information in the choice of encoding, chunk size and resource order.

Acknowledgments

The research presented in this paper has been generously supported by the Intel Visual Computing Institute, FI-CONTENT, FI-WARE/FI-Core, and FITMAN, which are part of the EU Future Internet PPP program, and the EU FP7 project VERVE.

References

10GEN, INC., 2013. BSON-Binary JSON specification. http://bsonspec.org/spec.html.

ARNAUD, R., 2011. rest3d 3D for the REST of us. http://rest3d.wordpress.com.

BEHR, J., ESCHLER, P., JUNG, Y., AND ZÖLLNER, M. 2009. X3DOM: A DOM-based HTML5/X3D Integration Model. In *Proceedings of the 14th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '09, 127–135.

BEHR, J., JUNG, Y., FRANKE, T., AND STURM, T. 2012. Using Images and Explicit Binary Container for Efficient and Incremental Delivery of Declarative 3D Scenes on the Web. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '12, 17–25.

BERNERS-LEE, T., FIELDING, R., AND MASINTER, L., 2005. RFC 3986: Uniform Resource Identifier (URI): Generic Syntax, Jan. http://tools.ietf.org/html/rfc3986.

BISCHOFF, S., AND KOBBELT, L. 2002. Streaming 3D geometry data over lossy communication channels. In *Multimedia and*

Expo, 2002. ICME '02. Proceedings. 2002 IEEE International Conference on, vol. 1, 361–364 vol.1.

BRYAN, P., AND NOTTINGHAM, M., 2013. RFC 6901: JavaScript Object Notation (JSON) Pointer, Apr. https://tools.ietf.org/html/rfc6901.

CABELLO, R., 2010. three.js - JavaScript 3D library. http://threejs.org/.

CROCKFORD, D., 2006. RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON), July. https://tools.ietf.org/html/rfc4627.

DOBOŠ, J., SONS, K., RUBINSTEIN, D., SLUSALLEK, P., AND STEED, A. 2013. XML3DRepo: A REST API for Version Controlled 3D Assets on the Web. In *Proceedings of the 18th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '13, 47–55.

ELECTRONIC ARTS, 1985. Electronic Arts Interchange File Format.

FETTE, I., AND MELNIKOV, A., 2011. RFC 6455: The WebSocket Protocol, Dec. https://tools.ietf.org/html/rfc6455.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T., 1999. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, June. http://tools.ietf.org/html/rfc2616.

FIELDING, R. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Phd thesis, University of California.

FUGGETTA, A., PICCO, G., AND VIGNA, G. 1998. Understanding code mobility. *Software Engineering, IEEE Transactions on 24*, 5 (May), 342–361.

GEELNARD, M., 2009. Open Compressed Triangle Mesh file format, Dec. http://openctm.sourceforge.net.

GOESSNER, S., 2007. JSONPath - XPath for JSON, Feb. http://goessner.net/articles/JsonPath/.

HERHUT, S., HUDSON, R. L., SHPEISMAN, T., AND SREERAM, J. 2012. Parallel Programming for the Web. In *Presented as part of the 4th USENIX Workshop on Hot Topics in Parallelism*, USENIX, Berkeley, CA.

HOPPE, H. 1996. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, 99–108.

JOSEFSSON, S., 2006. RFC 4648: The Base16, Base32, and Base64 Data Encodings, Oct. http://tools.ietf.org/html/rfc4648.

KHRONOS GROUP, 2011. OpenGL ES 2.0 for the Web, Mar. http://www.khronos.org/webgl/.

KHRONOS GROUP, 2011. Typed Array Specification, Feb. https://www.khronos.org/registry/typedarray/specs/latest/.

KHRONOS GROUP, 2014. WebCL, Mar. http://www.khronos.org/registry/webcl/specs/1.0.0/.

KLEIN, F., SONS, K., JOHN, S., RUBINSTEIN, D., SLUSALLEK, P., AND BYELOZYOROV, S. 2012. Xflow: Declarative Data Processing for the Web. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '12, 37–45.

LAVOUÉ, G., CHEVALIER, L., AND DUPONT, F. 2013. Streaming Compressed 3D Data on the Web Using JavaScript and WebGL. In *Proceedings of the 18th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '13, 19–27.

LIMPER, M., JUNG, Y., BEHR, J., AND ALEXA, M. 2013. The POP Buffer: Rapid Progressive Clustering by Geometry Quantization. *Computer Graphics Forum 32*, 7, 197–206.

MAMOU, K., 2013. Open 3D Graphics Compression (Open3DGC). https://github.com/amd/rest3d/tree/master/server/o3dgc.

MATROSKA, 2005. Matroška Media Container. http://matroska.org/technical/specs/index.html.

MURRAY, J., AND VANRYPER, W. 1994. *Encyclopedia of graphics file formats*. O'Reilly Software Series. O'Reilly & Associates, Inc.

NIELSEN, J. 1993. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

OBJECT MANAGEMENT GROUP. 2013. OMG MOF 2 XMI Mapping Specification. http://www.omg.org/spec/XMI/.

ROBINET, F., COZZI, P., ARNAUD, R., AND PARISI, T., 2012. glTF - the runtime asset format for WebGL, OpenGL ES, and OpenGL, Feb. http://gltf.gl.

SONS, K., AND SLUSALLEK, P. 2013. Towards a 3D transmission format for the Web. Tech. rep., 9th International AR Standards Community Meeting, May.

SONS, K., KLEIN, F., RUBINSTEIN, D., BYELOZYOROV, S., AND SLUSALLEK, P. 2010. XML3D: Interactive 3D Graphics for the Web. In *Proceedings of the 15th International Conference on Web 3D Technology*, ACM, New York, NY, USA, Web3D '10, 175–184.

TELECOMMUNICATION STANDARDIZATION SECOR OF ITU. 2005. Information technology – Generic applications of ASN.1: Fast infoset. http://www.itu.int/ITU-T/recommendations/rec.aspx?rec=X.891.

W3C, 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition), Nov. http://www.w3.org/TR/2008/REC-xml-20081126/.

W3C, 2012. Web Workers - W3C Candidate Recommendation, May. http://www.w3.org/TR/workers/.

W3C, 2014. Streams API. http://www.w3.org/TR/streams-api/.

W3C, 2014. XMLHttpRequest Level 1 - W3C Working Draft, Jan. http://www.w3.org/TR/XMLHttpRequest/.

WEB3D CONSORTIUM, 2008. Extensible 3D (X3D). http://web3d.org/x3d/specifications/.

WEB3D CONSORTIUM, 2011. Extensible 3D (X3D) encodings – Part 3: Compressed binary encoding. http://www.web3d.org/files/specifications/19776-3/V3.2/index.html.