Animation on the Web: A Survey

Amit L Ahire*, Alun Evans, Josep Blat
Interactive Technologies Group, Universitat Pompeu Fabra, Barcelona, Spain

Abstract

The main motivation of this paper is to provide a current state and a brief overview of animation on the web. Computer animation is used in many fields and it has seen a lot of development in the recent years. With the widespread use of WebGL and the age of powerful modern hardware available on small devices, 3D rendering on the browser is now becoming commonplace. Computer Animation can be described as the rendering of objects on screen, which can change shape and properties with respect to time. There are many approaches to rendering animation on the web, but none of them yet provide a coherent approach in terms of transmission, compression and handling of the animation data on the client side (browser). And if computer animation has to become more accessible over the web, these challenges need to be addressed in the same "minimalistic manner (requirement wise)" as every other multimedia content has been addressed on the web. We aim to provide an overview of the current state of the art, while commenting on the shortcomings pertaining to current formats/approaches and discuss some of the upcoming standards and trends which can help with the current implementation.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: Real Time Animation, Streaming, Compression, WebGL, Web, Animation, Survey

1 Introduction

1.1 What is Animation?

Computer animation is rendering of sequence of images on the screen, or can be perceived as an intuitive way of showing visualization to the user. It can be either interactive (Character animation), illustrative (Simulations) or entertaining (Motion Cinema, CGI). When concerned with high end realistic outputs, as in animation movies, the heavy duty rendering is typically carried out on render farms, where the end product is in the form of a video file which can be easily encoded and transmitted with different array of algorithms and techniques available. The purpose of this paper is not survey these techniques; we are concerned with real time rendering, based on the underlying transmitted mesh data, on the browser, to the array of output terminals available today in consumer space.

*e-mail: amit.ahire@upf.edu

Animation of complex objects are represented as either a set of points, mathematical equations or polygonal surface meshes. Animation of an object can be thought of as applying a set of geometric transformations or Interpolations from one set of key frames to another. The polygonal surface meshes, which are most commonly used data set type for animating an object, should primarily contain the following types of data to facilitate real time rendering: 1) Surface representation of the object with minimal error range and 2) A mechanism to provide generation or fetching of object properties (such as mesh vertex positions, triangle shape, texture, normal etc.) which can help in the subsequent animation process.

Animation as a process can be of different types depending on how the data is handled and what sort of calculations are applied to it:

- **Animation of Static Objects:** Any 3D object, irrespective of any other force that may be acting upon it, can be animated; either by applying various transformations to the camera or by numerically integrating the object properties over time based on some parameters. This is a very basic kind on animation that can be achieved without any complementing data from the object. Rigid bodies are the most common physical representation systems, here the object can be defined as one consisting of a shape in space, and having a volume. This volume is fixed and it doesn't change or deform in shape over time. They can exhibit physical properties as moving, rotating, experiencing and exerting physical forces.

- **Animation of Deformable Objects:** Deformability can be defined as change in shape or volume due to application of external forces. Deformable objects have their own set of physical properties and they exhibit complex motion that needs to be taken into consideration while animating. Some applications and examples are: Articulated Bodies (Character Animation), Elastic Objects (Cloth, Hair etc.), Flexible characters (fluids etc.), Procedural (height maps, crowds etc.) and skinning.

The principal contribution of this paper is that it forms the first survey in the state of the art into animation on the web; stating the specific challenges that web-based animation faces, reviewing the efforts made so far to address them, and discussing as to where the future may lie.

1.2 Overview Of The Challenges

Animation on the web as compared to normal animation comes with its own set of challenges and requirements. Regular animation maybe carried out by accessing a sequence of 3D models or by virtue of some sort of physic-based methods, which are employed to create a time based state of the animation model called key-frames.

For successful animation, there should be sufficient control points or governing rules to create key-frames. Using some form of

extrapolation or interpolation techniques, subsequent frames are generated and rendered, in a time-elapsed manner, onto the screen.

Computation and rendering of complex 3D objects is processor intensive, and level of detail (LOD) techniques are sometimes utilized in order to expedite the process. LOD techniques increase or decrease the complexity of the 3D object depending on the distance of the object from the camera. In online rendering, with a modern web browser as our canvas (rendering context) the paramount information is to have all the required data files for computing and rendering in order to reproduce the animation on the client end. But animation data can be voluminous and require a lot of space, which is not desired on a sandboxed environment as in a browser, hence efficient compression algorithms need to be applied on the base animation file, so that it can be transmitted as fast as possible and can be rebuilt with minimal information. Employing various LOD techniques can further reduce the data for transmission. And these transmitted data can be sent in a structured manner to reduce the computation time on the client side. JavaScript is mostly used to process data and rendering.

There are lots of research for compression and transmission of static 3D objects [Avilés and Morán 2008; Rossignac 2004] and dynamic 3D [Müller et al. 2005; Kao et al. 2010; Cao et al. 2014] objects and some of these techniques can be extended for animation files. But there is lack of uniformity among the various techniques used as well as among various technological standards on the web [Jones 2011; KhronosGroup 2014]. Hence, one method could decimate the whole 3D object and employ a spatial data structure in order to traverse through the structure for different LOD, or another would split the 3D object in a coherent manner and transmit in a structured manner. It all boils down to the particular requirement or the format of the existing 3D object. There is a need for a schema-less, encoding agnostic, structured or compatible format which could support 3D files without any dependency.

2 Problems and Challenges

2.1 Transmission Of Data

Animation of a 3D object or scene requires transmitting extra data, beyond that which is required to render the scene statically. Depending on the context, this extra data may dwarf the original static data source. Transmission of data via the internet uses basic and well established protocols such as TCP and UDP. At the lowest level, the basic problem associated with these networking protocols resolve around latency (TCP), accuracy (UDP) and bandwidth (both). Latency, in this context, can be perceived as the time needed to fully refine the 3D object since the time of application start up, and the time to refine the model based on the user interaction and different point of views. It is one of the most critical issues that a low powered client device can have, but considering that nowadays the so-called “low powered” mobile devices are equipped with so much processing power that the negative impact on the interactivity of the application mostly depends on network bandwidth and how the transmission is structured.

Despite dramatic improvements in recent years, mobile network connections still largely suffer from low bandwidth, and moreover these characteristic change with mobility and geographic location. Therefore, it is important that the data is transmitted in a structured manner and there is minimum possible interaction between server and client. As Sutter et al. [2014] points out, current formats allow for compression and binary transmission but lack any internal structure for transferring more than a single resource of a specific type. Thus, the number of requests to the server increases with the number of resources within a scene or object. In which case, if bandwidth is sufficient, latency becomes the limiting factor.

The transmission of 3D data is usually a two-step approach. First, the object structure is sent to the client. Second, the handling and processing of the data at the client end. The 3D data format for the web can be categorized as either text-based or in binary format. Formats encoded as text are human readable, thus their file size is bulkier. But the native HTTP’s GZIP compression does an excellent job of compressing it. A typical 3D scene or object structure(s) contains references to all the external resources such as meshes, images, textures, and animation data. The XMLHttpRequest (XHR) API is used to request each resource separately and they are transmitted using JSON, XML, binary XHR or any of the domain specific formats.

Another transmission data format suggested by Behr et al. [2012], is to use 2D images to encode the arrays representing the coordinates, normals and other attributes. This would allow us to use various audio-video encoding and compression techniques, and on the client side the data can be passed as texture in WebGL to the GPU, bypassing all the computation. But this can cause some problem in some older devices.

2.2 Compression & Preprocessing

The large amount of data which needs to transmitted for animation thus benefits from some level of compression and preprocessing, in order to reduce the time taken to transfer the data. However, as pointed out by Evans [2014], the balance between compression and speed is delicate, as complex compression can take time to decompress within a browser context, which is required to use interpreted JavaScript to do the work (and, thus, is slower). The balance between compression and bandwidth has been demonstrated effectively by Limper [2013].

Every mesh and its animation data is defined by: *Geometry* (Vertex Positions), *Connectivity* (graph of triangles) [Avilés and Morán 2008; Alliez and Desbrun 2001; Lee 2013] and *texture*/*color data*. On a broad scale, Compression can be divided into two groups:

Single Rate Compression: encodes and decodes the 3D object as a whole, thus we see the entire 3D object only after the data is decoded. These algorithms work by removing the redundancy in the dataset, while making no assumptions about its complexity, regularity or uniformity. Although these methods can achieve really good compression ratios, they are unsuitable for the network [Rossignac 2004]. For readers interested, we direct them to these comprehensive survey papers as a starting point [Alliez and Gotsman 2005; Avilés and Morán 2008; Ewiner 2005]

Progressive Compression: Progressive compression of 3D meshes uses the concept of refinement over a period of time. The idea is to transmit the crude approximation first, then predict intermediate vertices and either transmit the residues for the prediction or generate refinement of the mesh progressively. During decoding the connectivity of geometry is reconstructed incrementally from the transmission stream. The main advantage here is the minimal wait time for the user as intermediate states are transmitted through the network. Hoppe [1996] proposed progressive meshes (PM) in his seminal paper: a progressive mesh is obtained from a simplified base mesh and a series of vertex

splits (*vsplits*), which are used to refine the base mesh into a more refined and detailed mesh. Hoppe et al. took this further and developed view-dependent progressive meshes (VPM) where features contributing to the visible part is given priority for transmission. [Hoppe 1997; To et al. 2001; Yang et al. 2004]. Garland et al. proposed [Garland and Heckbert 1997] a method based on simplifying the vertex pair in the mesh iteratively, which was accurate and had a very low computation time. The simplified edge sequence is determined by the quadric error metrics (QEM) – a quadric measure which is a matrix that represents the sum of the squared distances from the vertex to the planes of neighboring triangles, is associated with each vertex.

Geometry videos [Briceño et al. 2003] offer a different approach, based on the geometry images of [Gu et al. 2002]. The mesh is first parameterized, then re-sampled over a regular grid in space. The result is a 2D image with three channels, for x, y and z coordinates. The mesh geometry is stored or transmitted simply as an image stream, which can be compressed using existing video compression methods.

2.3 Rendering

In this section, we briefly survey the principal rendering context for web-based animation. WebGL has been used for real time 3D visualization over the web, which is an imperative approach to 3D web graphics [Jankowski et al. 2013; Coughlin 2014]. It uses the HTML5 canvas element and JavaScript to expose low level graphics API which is based on OpenGL ES 2.0 (a restricted version of the OpenGL API, originally designed for embedded systems, but capable of compiling vertex and fragment shaders). During initialization of the WebGL context, the shader code is compiled and copied onto the video card memory to be executed on the GPU. 3D data is fetched over the Internet using any of the data-interchange formats (See section 3.4). These fetched data are stored in *ArrayBuffers,* with the recent TypedArray specification [Herman and Russell 2013; Ecma International 2013], which allows for low level manipulation in JavaScript and moreover, they are perfect for parallel decoding in the GPU, as data can be copied with zero overhead between the web worker [World Wide Web Consortium 2013] and the main thread.

Indexed 16-bit arrays: The current limitations are mainly in terms of the current specification of the rendering API. In comparison with the standard OpenGL specification, the *glDrawElements()* function used for mesh rendering only allows `GL_UNSIGNED_BYTE` and `GL_UNSIGNED_SHORT` types for index array, limiting it to a maximum of 65536 vertices in an array. To render larger meshes, the mesh needs to be split into sub-meshes with number of indices not exceeding this limit.

Large Memory Footprint Of Declarative Languages: As Sons [2010] pointed out, that both WebKit and Mozilla represent the text within an HTML element as native "text nodes" within the DOM data. As a result both currently store the full text representations of all vertex data in the DOM, which increases the memory footprint.

DOM *onProgress* Event: While using the Progressive Image-based transmission method, due to the periodical notification of the *onProgress* event, there could be cases when only approximations of the sub images are calculated. Some web browsers employ a better approach, where the *onProgress* event wouldn't be time based but would only be triggered after completion of some event such as loading up of one sub image.

3 Survey of Current Approaches

3.1 Transmission over the Web

Computer animation files exported from 3D modeling applications are huge in size, for transmission over the web, let us define the process, whereby a lightweight HTML pages holds the DOM structure and embedded script code and all the other external data such as mesh, vertices, animation data, and images are separately requested over the HTTP protocol. Here, we compare some of the current solutions and methods :[Doboš et al. 2013; Sutter et al. 2014]

Binary transmission: A very simple and common approach which is used in XML3D [Sons et al. 2010; Sons 2013], X3DOM [Stamoulias et al. 2014; X3DOM], glTF [KHRONOS GROUP 2012] and Three.js[Cabello 2010]. The data being transmitted is encoded in textual representation (e.g. XML or JSON) and the external data is requested using a XHR request separately.

Image Based: Here, the mesh data is encoded into images, which presents us with the opportunity to effectively use the underutilized power of the GPU on the modern systems. Encoding geometry information inside 2D images is a very promising prospect. Image decoding is done natively in the browser, decoding vertex data to the GPU totally bypasses any JavaScript based decoding. Sequential Image Geometry (SIG) [Behr et al. 2012] extends this approach further by adding functionality for quantization and progressive loading.

Domain Specific: There are many solutions for streaming static 3D meshes, But a 3D object not just consists of a mesh, but also contains other information such as texture, normals etc. Hence, it can be hard for any methods to be extended to all kinds of 3D data. Solutions like OpenCTM [Geelnard 2009] and WebGL-Loader [Chun 2011; Blume et al. 2011] are lossless and efficient binary representations. Both formats are converted to TypedArray, which needs to be processed on the client side. Compared to image based GPU solution, it can have a slower decoding speed.

3.2 Compression and Preprocessing

Level of Detail: Another issue is the reduction of quality and detail of the mesh in interactive applications. LOD concepts such as Progressive meshes [Hoppe 1996] help render according to the scene details either by mesh simplification process or by exploiting progressive transmission and decompression schemes.

Mesh Level of Detail: Representation of objects based on their distance from the camera with lower LOD for distant object and higher LOD for nearer object, the interactivity and rendering of the 3D scenes can be improved a lot. According to [Savoye and Meyer 2008], there are no algorithms that excel at simplifying animated models and in achieving high fidelity of the original mesh. Either there are quantization metric measures for surface simplification and preserving the topology using *Quadric Error Metric* (QEM) by [Garland and Heckbert 1997]; or the most popular real-time triangle reduction for polygon manifolds, introduced by Hoppe et al., *Progressive Meshes* [1996]. Ensuring the best quality of a mesh simplification requires huge computational costs, and sometimes for no significant visual improvement.

Animation Level of Detail: Simplification techniques work well for static meshes but not so much for deformable objects. *Deformation Sensitive Decimation* (DSD) by Mohr and Gleicher

[2003] was based on the idea of QEM [Garland and Heckbert 1997], for measuring the contraction cost of edges. Pilgrim et al. [2006] present *Progressive Skinning* – A view and pose independent methods for automatic skeleton simplification for the methods given in [DeCoro and Rusinkiewicz 2005]. The method uses an edge collapsed contraction with QEM and a linear skinning weight update rule. Kavan et al. introduced in [Kavan et al. 2008] a novel representation of virtual characters called Polypostors with 2D polygonal impostors. Mukai and Kuriyama introduce the idea of motion level of detail in [Mukai and Kuriyama 2007] using an LOD control method of motion synthesis with a multilinear model.

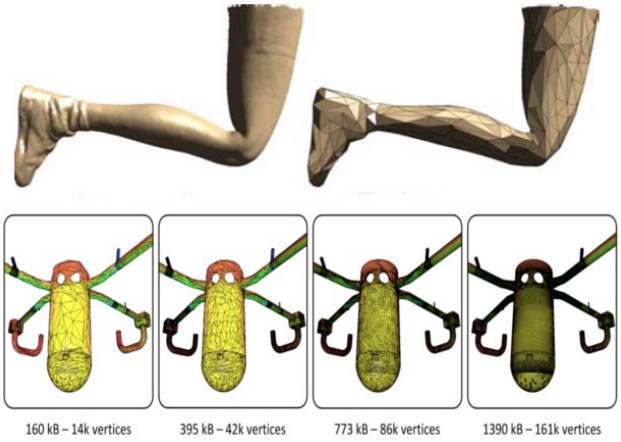

Figure 1: *Depending of various factors, mesh can be decimated to various levels. Images reference from* [DeCoro and Rusinkiewicz 2005] *and* [Lavoué et al. 2013]

3.3 Rendering Animation

Animation in the WebGL context can employ many different concepts such as: 1) programmatically updating properties of the visual object each time through the run loop; 2) Using key frame animation, intermediate frames are generated via interpolation; 3) Using morph targets to deform geometry by blending among a set of distinct shapes. 4) Using skinning to deform geometry based on the underlying skeleton. And 5) Implementation of some/all of the above using GPU based methods, where the deformation of vertices and fragment values are computed on the GPU in the shaders.

For rendering progressive animation in WebGL, the animation should be reconstructed first after the minimal transmission is done, a coarse representation is rendered on the screen. Then as more data is downloaded, and more information becomes accessible, the compressed animation is refined. Mostly these refinement methods can be classified into 1) Connectivity updating - which changes the feature shapes of an animation, and 2) Geometry updating – which changes the complexity of the motion.

In real-time rendering, 2D images are used to store textures that are applied and interpolated during the shading process to flat surfaces in 3D space. Similarly, Normal maps and displacement maps give the illusion of more structured details. For 3D data transmission, textures can be used by using the quantization threshold [Rodríguez et al. 2013; Maglo et al. 2012; Schwartz et al. 2011] which represents a factor by which to quantize the geometry, which can be transmitted separately in a JSON file. Once the client side application knows of the factor, the decompression of the PNG takes place to retrieve the chunk of relevant data into a WebGL texture. Behr et al. present a detailed explanation on various techniques involving 2D images for data transmission. [Behr et al. 2012]

Using VBO in WebGL: Declarative languages provide types for data entries in the structure which can be directly mapped to an OpenGL Vertex Buffer Objects (VBOs). These set of vertex attributes can then be packed into Vertex Array Objects (VAOs) (available in WebGL 1.0 through `OES_vertex_array_object` extension) and called in a single API call to reduce the state change overhead. In comparison, when data is transmitted in binary or any other domain specific manner, different subsets of 3D data is sent to the shader to be rendered. To draw such highly frequent changing data in the VBOs, they need to be constantly created and/or updated and the data needs to be passed from main memory to client-side index array which proves to be a huge performance bottleneck for WebGL. One approach to reduce the vertex buffer switches, is to have all the mesh data in a single image (either a texture atlas or map) and create one VBO on the JavaScript layer and bind it with this single image geometry and have all the other data directly uploaded as textures in the GPU. This is similar to the technique used in [Behr et al. 2012], [Schwartz et al. 2011], [Englert et al. 2014] and [Stein et al. 2014]. This allows the GPU to store more mesh data, and calculate all the decoding and vertex coordination on the fly.

3.4 How existing formats and/or libraries deal with Animation

3D graphics on the web features various file formats and libraries, each with its own advantages and disadvantages. In this paper, we will not focus on the general functionality and working aspect of these. There are many surveys available which do a fine job explaining them [Evans et al. 2014; Coughlin 2014]. Here, we will only focus on how these file formats or libraries deal with animation.

X3D: X3D is a royalty free ISO standard defining an XML based file format for representing Web3D computer graphics and was first accepted as an ISO standard in 2004. X3D is designed to deliver a lightweight and balanced web-based application, which offers easy to use and develop interactive real-time 3D content. The X3D standard supports a wide number of advanced 3D graphics functionalities, including key frame animations, humanoid animations (H-Anim) etc. [Stamoulias et al. 2014]. X3D consists of nodes which handle various types of jobs within the framework, In order to handle key frame animations it goes through the following process: The *TimeSensor* node is used to control animation as time passes. An *OrientationInterpolator* node sits between the *TimeSensor* node and the *Transform* node and turns the events into vectors. Based on the update time step, the object properties are updated in order to achieve the animation. [X3D; H-Anim]. In [Schilbach 2014], a framework for animation is introduced based on the existing model but it is mainly aimed at generating animated visualization to conduct quantitative experiments and not on real time realistic animation.

X3DOM & X3DOM Binary Geometry (BG): X3DOM allows X3D documents to be easily embedded directly into web applications by integrating X3D into the DOM [Behr et al. 2010; Stamoulias et al. 2014]. X3DOM operates as a connector which is responsible for the synchronization of the browser frontends and the X3D backend, by monitoring DOM updates in the X3D

code. The raw binary encoding of indices and vertex attributes are considered, along with lightweight structure description in the human readable format by packing them in a grid format such as in images and video. It is mainly possible because of the *TypedArray* [Ecma International 2013] specification of JavaScript, a recent addition to JavaScript to handle the binary data efficiently. *TypedArray* are like a slab of memory, more like how the array works in C, which allows downloading and manipulation of the binary data directly in a web page using JavaScript, and they can be transferred with zero copying overhead between a Web Worker [World Wide Web Consortium 2013] and the main thread and onto the GPU, thus minimalizing the CPU-GPU overhead. It does support animation and interaction to some degree without the low level flexibility. [X3DOM]

XML3D: XML3D [Sons et al. 2010; Sons 2010] is a minimal extension to HTML5 for interactive 3D content. It defines a small set of new elements to describe a scene graph with 3D geometry, surface shading, and lighting. In addition, it features the declaration of generic data structures as well as the declarative language. *Xflow* [Klein et al. 2013] is used to perform vertex and images processing on the data. XML3D supports a wide range of 3D related features, such as skinned mesh animations and Augmented Reality [Klein 2013] without adding a large amount of new DOM elements.[XML3D]

glTF: glTF is created from a Collada [KHRONOS GROUP] digital asset exchange (.dae) files which became an ISO standard in 2013. Collada is widely supported as an export file type option across many types of 3D content creation software. Every glTF asset is comprised of various files, which is embedded in 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. Using the JSON format for the scene hierarchy is practical because it is much more easily parsed than XML and is also more compact so will take less time to download as well. Textures are simply PNG, JPEG etc., hence there is no need for further modification as all the major browsers come with these decoders. The mesh binary data, is raw binary data meant to be passed directly into buffers. Optional Open3DGC encoding is available [Mammou 2013] which is designed for fast decoding in JavaScript or C++ using arithmetic algorithms.

A common approach in XML3D, X3DOM and glTF are unstructured binary XHR using an additional document for structure information.

Sequential Image Geometry (SIG): Here, instead of transforming and resampling of the original mesh to a regular structure, it utilizes an image file structure to store unlinked vertex data and indices. The vertex arrays are split into 8-bit chunks of decreasing relevance that are distributed as a sequence of images. The approach supports quantization and progressive loading as long as the transmission is in correct order. [Behr et al. 2012]

OpenCTM : OpenCTM is an open binary format for 3D mesh compression [Geelnard 2009]. It provides a compact representation of 3D triangle meshes using the entropy-reduction technique with the state-of-the-art Lempel-Ziv-Markov chain algorithm (LZMA [7Zip]) also offering good compression rates, while still providing a relatively fast decompression. OpenCTM is a mainly a lossless compression technique but also allows to use a lossy compression in order to improve the compression ratio. [Chávez et al. 2013; Limper et al. 2013]

WebGL-Loader (Or UTF-8 Compression): The UTF-8 compression or also known as the *WebGL-Loader* is a JavaScript library for compact mesh transmission. [Blume et al. 2011; Cozzi and Riccio 2012; Chun 2011], which was developed for *Google Body* project - a browser based human anatomy project. The UTF-8 compression is a lossy technique. It improves compression by predicting the normal direction from neighborhood positions of incident triangles, also performs a vertex cache optimization on the index list [Forsyth 2006]. Instead of a simple delta encoding, a more advanced parallelogram prediction is used for the attributes, it predicts the next vertex position by constructing a parallelogram with the last three vertices of the triangle strip. The normals are predicted using the cross product of the edges of every triangle. The data is encoded using the UTF-8 character set: 16-bit values taking 1-3 bytes per character. So, the lower the values, the lesser memory the algorithm requires for encoding a mesh. And the native GZIP implementation of the browser, provides a fast decompression.

Three.js: Three.js has becomes a very popular option on the web, it provides good performance and good abstraction making it easier to use than dealing with low level initialization. While dealing with animation, it deals with skeletal animation, morph shapes and key frames [Cabello 2010; 2012], It allows "baked" animation i.e. where vertices per frame are available in the file, similar to how morph target or blend shapes work, and also blending and interpolation among the key frames. It supports many types of blending (additive, multiplying and subtractive) and uses Catmull-rom spline method to interpolate among given frames [Cabello 2011a; Cabello 2011b] Catmull-rom spline are means of representing a curve, by specifying a series of points at interval along the curve and then using a mathematical function that then calculates the additional points along the way.

Figure 2: *An example of animation in three.js showcasing blending and morphing* [Cabello 2010].

Three.js supports animation by taking in a JSON, which consists of parameters such as position, rotation, scale, time etc. The library itself provides several scripts for exporting and/or converting popular 3D formats (such as format discussed above) into a three.js compatible JSON. Currently the library only handles mesh models, non-manifold and other data formats of the objects are not supported. For key frame animation, the weights associated with vertices must be mentioned for proper representation, and it supports multiple animations for a single object. [Ray 2014]

Babylon.js: Babylon.js is another open source framework, supported by Microsoft. While Three.js attempts to bring wide range of animation features, Babylon.js takes a more target specific approach and focuses more on being a game engine

equipped with collision detection and a physics engine. In terms of 3D object animation, it supports key frame animation and skeletal animation [Microsoft 2013], and also importing scenes from various 3D formats such as OBJ and FBX. A skeleton is specified by providing the bone information, weights and influences associated with each vertex position.

SpiderGL : SpiderGL is a 3D graphics library for real time rendering [Di Benedetto et al. 2010], it provides a data structure and algorithms to ease development and similarly aims at providing an abstraction for developers to use the underlying graphical capability of WebGL. It supports features such as asynchronous content loading etc. But it doesn't seem to support animation as such.

4 Future Trends Or Discussion

4.1 Transmission Of Data

The Web is in need of a binary transmission format for 3D data that allows for domain-specific or at least a general standard of compression technique for animation data. As outlined in [Sons 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. All approaches discussed here address only a subset of the challenges that need to be solved for a general, common, and efficient transmission of 3D data. With regards to 3D asset transmission, a prominent problem is the lack of progressive streaming of all relevant mesh and texture data, with a minimal number of HTTP requests. Furthermore, there is still no established format for a joined, interleaved transmission of geometry data and texture data.

[Limper et al. 2014a] aims to minimize the number of HTTP requests, by progressively transmitting an arbitrary number of mesh data chunks within a single SRC file. This file contains the reference to all the resources of data and progressively downloads them.

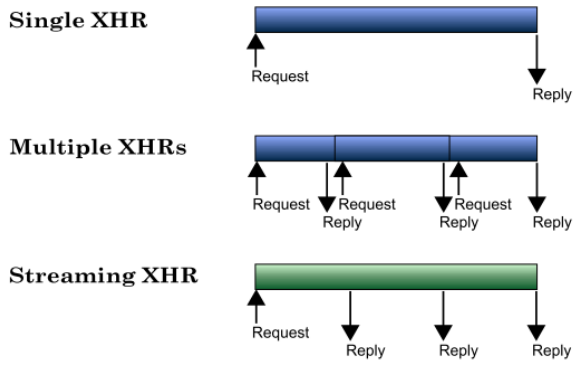

Figure 3*: Multiple XHRs to server to download resources is not good., Idealy streaming XHR should consist of only one request.Image reference from* [Limper et al. 2014b]

Another approach to solving the bandwidth problem is to use caching strategies; caching is necessary to improve the performance by avoiding redundant data transfer. Caching prevents sending large amount of data, but it requires bi-directional communication which is not available HTTP protocol. *WebSockets* can be used, which could improve latency as it supports a full duplex communication channel. Web Sockets allow the client and server to stream data bi-directionally but requires a protocol understood by both ends. However, no general protocol for the transmission of 3D data has been established yet. To improve upon the lack of progressive downloads, there is already W3C draft in the works for the next standards for HTTP 2.0 and Streams API, which will bring progressive downloads to its feature list.

A more challenging problem worthy of further investigation is 1-D streaming [Lee 2013], which combines the geometry and connectivity operators into the more general refining operators. To achieve different progressive representations for animation, other progressive methods such as valance-driven can be compared.

Gobbetti et al. [Gobbetti and Marton 2012] proposed a method that also uses image-based mesh description format. It resamples the model data in order to build a tight atlas parametrization of the mesh geometry. This enables them to use the atlas images also for multi-resolution transmission and rendering via simple mipmap operations.

Similar to X3DOM's BinaryGeometry node, Lee et al. [2010] propose to reduce the size of binary mesh data for efficient storage and transmission, using a straightforward local quantization scheme. They argue that geometry compression for mobile graphics requires a careful choice of the compression method in order to maintain interactive decompression rates.

4.2 Compression and Preprocessing

Lavoué et al. [2013] state some of the issues that are overlooked by the scientific community: Existing compression techniques give more importance to compression ratio, where as in online transmission, improving quality of level of detail is more important than being clever and gaining a few extra bits. Few recent researchers have taken LOD into serious consideration. [Peng et al. 2010; Lee 2013]

Perfect level of detail management requires efficient handling of the 3D object attributes such as color, texture and other information regarding the data and animation. They need to be progressively maintained and processed as LOD of geometry and connectivity is handled. Some example of recent techniques are [Lee 2013; Limper et al. 2014a; Schwartz et al. 2011; Behr et al. 2012].

Wen et al. [Wen 2014] presents a similarity aware 3D model reduction method, which searches for similar component in the 3D model and reuse them through the construction of Lightweight Scene Graph (LSG), It doesn't require any decompression at the client end and required LOD is obtained with help of instance rendering.

The objective of progressive compression is to speed up transmission over the network, but if decompression takes as long on the client side, then there is no point in applying fancy techniques and data structures. It needs to rely on simple yet effective algorithm, which could easily transcribe itself onto JavaScript and WebGL.

4.3 Rendering

JavaScript has evolved tremendously in the last few years, from being the scripting language of the client-side Web to a full-fledged comprehensive language used in almost any application area. It runs on a sandbox environment on a browser but now with the help of frameworks such as node.js or Rhino, it is used as a standalone language. One of the biggest drawbacks is its slow execution speed when complied just-in-time (JIT). In an attempt to bring JavaScript execution speed closer to native code, asm.js was developed, which enables a strict subset of JavaScript language; stripping if off of some of the features, allows for performance improvement such as ahead-of-time optimization and hence speeding up the execution speed. Another strategy has been used to enable the developer to bring their native(C/C++) legacy code to the browser, Google Native Client is one such sandboxing technology which would allow safely running native codes from a web browser. Emscripten [Zakai 2011] is another such attempt, it's a transcompiler – it takes bit code as input and emits JavaScript code (in asm.js).

Parallelism in JavaScript: Client side processing in JavaScript when dealing with large amount of data can stall the performance, as JavaScript is single-threaded. WebWorkers were introduced in HTML5 and help in parallel decoding by taking care of some task in the background and then passing the data to the main thread through message passing. Apart from WebWorkers, other technologies that aim to bring parallelism by passing the computation onto the GPU are : River Trail [Herhut et al. 2013] and WebCL [KhronosGroup 2012] . River Trail provides a ParallelArray data structure with primitive operations that operate on it and offload the computation to the GPU and the underlying OpenCL implementation. WebCL exposes a subset of OpenCL 1.1 through JavaScript and allow application to harness the power of the GPU.

WebGL 2.0: The WebGL 2.0 specification has been under development for two years and is approaching release. WebGL 2.0 is equivalent to OpenGL ES 3.0, although many of the current specifications are available as extension packages of WebGL 1.0. Some of the major features include occlusion queries – which tries to reduce the rendering load on the graphical system by eliminating objects from the rendering pipeline which are hidden by other objects; transform feedback – capturing primitives generated by the vertex shader and recording data into a *BufferObjects*, thus allowing to preserve and resubmit data multiple time; Multiple render targets – allows a single draw call to write out to multiple targets (texture or renderBuffer); and , vertex array objects (VAO) – VAO allows us to store the vertex/index binding information for set of objects in a easy to manage manner.

Stein et al. [2014] demonstrate using spatial data structure on the client end using the latest features of WebGL 2.0 to improve upon the shortcomings of the current 3D web environment. They render an interactive visualization of large 3D data sets, by employing small bounding volume hierarchies to accelerate visibility determination. Although they support only static scenes, they efficiently handle and render huge amounts of data sets into the scene.

Texture Compression: Texture compression can drastically reduce the amount of memory consumption in a 3D scene. It can also be helpful in transmission. Texture compression support of WebGL allows for the direct upload of the compressed textures onto the GPU without the need of any processing step. There are many WebGL extensions to support various different formats. Texture can also contain other data which can be helpful for rendering, all the data required for rendering is stored in 2D textures, as currently this is what is supported by WebGL, although 3D textures are proposed to be supposed in WebGL 2.0 specification.

5 Conclusion

We have presented an overview of the state of animation over the web and the open challenges present in the context. It is clear that WebGL as graphical platform and web browsers as a common viewport present seamless prospects for graphics over the web. With the new working draft of these standards, any or all barriers for real time rendering are fast diminishing. But it is also clear that there is a need for a common, structured, encoding-agnostic and stream able standard for animated content over the web.

Most of the current techniques are built upon a few seminal works which have approached the problem in diverse ways. For example, one general approach tries to decimate the mesh data based on the connectivity, geometry, and using a spatial data structure for traversal. While other approaches are more geared towards efficient rendering on the client side. Moreover, there is surprisingly little work done on bringing real time animation on the web, most of it has been focused on transmitting static 3D mesh objects.

Nonetheless, there is a lot of research done recently concerning WebGL and rendering objects on the web [Congote et al. 2011; Schilbach 2014; Stein et al. 2014; Stamoulias et al. 2014]. WebGL 2.0 promises to bring many more features to the table, and there are several interesting avenues that need to be explored, such as using the GPU to parallelize the computing process (on the browser), protecting intellectual data post-transmission, and efficient reproduction of the animation data. Fortunately, the landscape on the web is soon catching up with the state of the art and promises to bring exciting new features with broadening horizons.

Acknowledgement

This work was supported by the European Commission, H2020 KRISTINA project, and by the Spanish EEE (TIN2011-28308-C03-03) project.

References

7ZIP. Lempel-Ziv-Markov Chain Algorithm (LZMA). .

ALLIEZ, P. AND DESBRUN, M. 2001. Progressive Compression for Lossless Transmission of Triangle Meshes. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 195–202.

ALLIEZ, P. AND GOTSMAN, C. 2005. Recent advances in compression of 3D meshes. *Advances in Multiresolution for Geometric Modelling*, 1–25.

AVILÉS, M. AND MORÁN, F. 2008. Static 3D triangle mesh compression overview. *Proceedings - International Conference on Image Processing, ICIP 2*, 2684–2687.

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. *Web3D 3D technologies for the World Wide Web 1*, 17–26. http://dl.acm.org/citation.cfm?id=2338717.

BEHR, J., JUNG, Y., KEIL, J., ET AL. 2010. A scalable architecture for the HTML5/X3D integration model X3DOM.

Proceedings of the 15th International Conference on Web 3D Technology - Web3D '10, ACM Press, 185.

DI BENEDETTO, M., PONCHIO, F., GANOVELLI, F., AND SCOPIGNO, R. 2010. SpiderGL. *Proceedings of the 15th International Conference on Web 3D Technology - Web3D '10*, ACM Press, 165.

BLUME, A., CHUN, W., KOGAN, D., ET AL. 2011. Google Body. *ACM SIGGRAPH 2011 Talks on - SIGGRAPH '11*, ACM Press, 1.

BRICEÑO, H.M., SANDER, P. V., MCMILLAN, L., GORTLER, S., AND HOPPE, H. 2003. Geometry videos: a new representation for 3D animations. 136–146.

CABELLO, R. 2010. ThreeJS. .

CABELLO, R. 2011a. Three.js | Animation.js. .

CABELLO, R. 2011b. Three.js | Animation Docs. .

CAO, C., HOU, Q., AND ZHOU, K. 2014. Displaced dynamic expression regression for real-time facial tracking and animation. *ACM Transactions on Graphics 33*, 4, 1–10.

CHÁVEZ, G., ÁVILA, F., AND ROCKWOOD, A. 2013. Lightweight Visualization for High-Quality Materials on WebGL. *Proceedings of the 18th International Conference on 3D Web Technology*, 109–116.

CHUN, W. 2011. WebGL-Loader. https://code.google.com/p/webgl-loader/.

CONGOTE, J., SEGURA, A., KABONGO, L., MORENO, A., POSADA, J., AND RUIZ, O. 2011. Interactive visualization of volumetric data with WebGL in real-time. *Proceedings of the 16th International Conference on 3D Web Technology - Web3D '11*, ACM Press, 137.

COUGHLIN, B. 2014. 3D for the Modern Web - Declarative3D and glTF. http://mason.gmu.edu/~bcoughl2/cs752/.

COZZI, P. AND RICCIO, C. 2012. OpenGL Insights. 712. http://www.seas.upenn.edu/~pcozzi/OpenGLInsights/OpenGLInsights-WebGLModelsEndToEnd.pdf.

DECORO, C. AND RUSINKIEWICZ, S. 2005. Pose-independent simplification of articulated meshes. *Proceedings of the 2005 symposium on Interactive 3D graphics and games - SI3D '05*, ACM Press, 17.

DOBOŠ, J., SONS, K., RUBINSTEIN, D., SLUSALLEK, P., AND STEED, A. 2013. XML3DRepo. *Proceedings of the 18th International Conference on 3D Web Technology - Web3D '13*, ACM Press, 47.

ECMA INTERNATIONAL. 2013. ECMAScript Language Specification, 6th Edition, Draft Revision 14. http://wiki.ecmascript.org/doku.php?id=harmony:specification_drafts.

ENGLERT, M., JUNG, Y., KLOMANN, M., ETZOLD, J., AND GRIMM, P. 2014. Instant texture transmission using bandwidth-optimized progressive interlacing images. *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies - Web3D '14*, ACM Press, 144–144.

EVANS, A., ROMEO, M., BAHREHMAND, A., AGENJO, J., AND BLAT, J. 2014. 3D graphics on the web: A survey. *Computers & Graphics 41*, 43–61.

EWINER, T.H.L. 2005. GEncode : Geometry – driven compression in arbitrary dimension and co – dimension 1 Introduction 2 Basic concepts. 1–8.

FORSYTH, T. 2006. Linear-Speed Vertex Cache Optimisation. https://home.comcast.net/~tom_forsyth/papers/fast_vert_cache_opt.html.

GARLAND, M. AND HECKBERT, P.S. 1997. Surface simplification using quadric error metrics. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, ACM Press, 209–216.

GEELNARD, M. 2009. OpenCTM Mesh Compression Format. http://openctm.sourceforge.net/.

GOBBETTI, E. AND MARTON, F. 2012. Adaptive quad patches: an adaptive regular structure for web distribution and adaptive rendering of 3D models. *Web3D 3D technologies for the World Wide Web*, 9–16.

GU, X., GORTLER, S.J., AND HOPPE, H. 2002. Geometry images. *ACM Transactions on Graphics 21*, 3, 355–355–361–361.

H-ANIM. Humanoid Animation. http://www.h-anim.org.

HERHUT, S., HUDSON, R.L., SHPEISMAN, T., AND SREERAM, J. 2013. River trail. *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications - OOPSLA '13*, 729–744.

HERMAN, D. AND RUSSELL, K. 2013. Typed Array Specification. http://www.khronos.org/registry/typedarray/specs/latest/.

HOPPE, H. 1996. Progressive meshes. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*, ACM Press, 99–108.

HOPPE, H. 1997. View-dependent refinement of progressive meshes. *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, ACM Press, 189–198.

JANKOWSKI, J., RESSLER, S., SONS, K., JUNG, Y., BEHR, J., AND SLUSALLEK, P. 2013. Declarative integration of interactive 3D graphics into the world-wide web. *Proceedings of the 18th International Conference on 3D Web Technology - Web3D '13*, ACM Press, 39.

JONES, B. 2011. TojiCode: Compressed Textures in WebGL. http://blog.tojicode.com/2011/12/compressed-textures-in-webgl.html.

KAO, C.K., JONG, B.S., AND LIN, T.W. 2010. Representing progressive dynamic 3D meshes and applications. *Proceedings - Pacific Conference on Computer Graphics and Applications*, 5–13.

KAVAN, L., DOBBYN, S., COLLINS, S., ŽÁRA, J., AND O'SULLIVAN, C. 2008. Polypostors. *Proceedings of the 2008 symposium on Interactive 3D graphics and games - SI3D '08*, ACM Press, 149.

KHRONOS GROUP, 2012. 2012. glTF - the runtime asset format for WebGL, OpenGL ES, and OpenGL. https://github.com/KhronosGroup/glTF.

KHRONOS GROUP, 2013. COLLADA.org. https://collada.org/.

KHRONOSGROUP. 2012. WebCL, Heterogeneous parallel computing in HTML5 web browsers. http://www.khronos.org/webcl/.

KHRONOSGROUP. 2014. WEBGL_compressed_texture_s3tc Extension Specification. *The Khronos Group*. http://www.khronos.org/registry/webgl/extensions/WEBGL_compressed_texture_s3tc/.

KLEIN, F. 2013. Declarative AR in the Web with XML3D and Xflow XML3D. 1–8. http://www.perey.com/ARStandards/[Klein-Slusallek]xflow_9th_AR_St_Meeting.pdf.

KLEIN, F., SONS, K., RUBINSTEIN, D., AND SLUSALLEK, P. 2013. XML3D and Xflow: Combining declarative 3D for the Web with generic data flows. *IEEE Computer Graphics and Applications 33*, 5, 38–47.

LAVOUÉ, G., CHEVALIER, L., AND DUPONT, F. 2013. Streaming compressed 3D data on the web using JavaScript and WebGL. *Proceedings of the 18th International Conference on 3D Web Technology - Web3D '13*, 19. http://dl.acm.org/citation.cfm?id=2466533.2466539.

LEE, J., CHOE, S., AND LEE, S. 2010. Mesh geometry compression for mobile graphics. *2010 7th IEEE Consumer Communications and Networking Conference, CCNC 2010*, 301–305.

LEE, P.-F. 2013. Progressive Animation Sequences. *2013 10th International Conference Computer Graphics, Imaging and Visualization*, 11–16.

LIMPER, M., THÖNER, M., BEHR, J., AND FELLNER, D.W. 2014a. SRC - a streamable format for generalized web-based 3D data transmission. *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies - Web3D '14*, 35–43. http://dl.acm.org/citation.cfm?id=2628588.2628589.

LIMPER, M., THÖNER, M., BEHR, J., AND FELLNER, D.W. 2014b. SRC - a streamable format for generalized web-based 3D data transmission. *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies - Web3D '14*, 35–43.

LIMPER, M., WAGNER, S., STEIN, C., JUNG, Y., AND STORK, A. 2013. Fast delivery of 3D web content. *Proceedings of the 18th International Conference on 3D Web Technology - Web3D '13*, ACM Press, 11.

MAGLO, A., COURBET, C., ALLIEZ, P., AND HUDELOT, C. 2012. Progressive compression of manifold polygon meshes. *Computers & Graphics 36*, 5, 349–359.

MAMMOU, K. 2013. Open 3D Graphics Compression. https://github.com/KhronosGroup/glTF/wiki/Open-3D-Graphics-Compression.

MICROSOFT. 2013. Babylon.js. .

MOHR, A. AND GLEICHER, M. 2003. *Deformation sensitive decimation.* .

MUKAI, T. AND KURIYAMA, S. 2007. Multilinear Motion Synthesis with Level-of-Detail Controls. *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, IEEE, 9–17.

MÜLLER, K., SMOLIC, A., KAUTZNER, M., EISERT, P., AND WIEGAND, T. 2005. Predictive compression of dynamic 3D meshes. *Proceedings - International Conference on Image Processing, ICIP 1*, 621–624.

PENG, J., HUANG, Y., KUO, C.C.J., ECKSTEIN, I., AND GOPI, M. 2010. Feature oriented progressive lossless mesh coding. *Computer Graphics Forum 29*, 2029–2038. http://www.ics.uci.edu/~gopi/PAPERS/PG10.pdf.

PILGRIM, S.J., AGUADO, A., MITCHELL, K., AND STEED, A. 2006. Progressive skinning for video game character animations. *ACM SIGGRAPH 2006 Sketches on - SIGGRAPH '06*, ACM Press, 114.

RAY, A. 2014. Exporting models from 3dsMax to ThreeJS. .

RODRÍGUEZ, M.B., GOBBETTI, E., MARTON, F., AND TINTI, A. 2013. Coarse-grained multiresolution structures for mobile exploration of gigantic surface models. *SIGGRAPH Asia 2013 Symposium on Mobile Graphics and Interactive Applications on - SA '13*, ACM Press, 1–6.

ROSSIGNAC, J. 2004. Compressing Volumes and Animations (Tutorial Notes). .

SAVOYE, Y. AND MEYER, A. 2008. Multi-layer level of detail for character animation. *Proceedings of the Workshop on Virtual Reality Interaction and Physical Simulation - VRIPHYS*.

SCHILBACH, J. 2014. An Event-Based Framework for Animations in X3D. 89–97.

SCHWARTZ, C., RUITERS, R., WEINMANN, M., AND KLEIN, R. 2011. WebGL-based Streaming and Presentation Framework for Bidirectional Texture Functions. *The 12th International Symposium on Virtual Reality Archeology and Cultural Heritage VAST 2011*, 113–120. http://diglib.eg.org/EG/DL/WS/VAST/VAST11/113-120.pdf.

SONS, K. 2010. XML3D: Declarative and interactive 3D graphics as extension to HTML5. http://www.xml3d.org/wp-content/uploads/2010/11/XML3D-TPAC-ks-2010.pdf.

SONS, K. 2013. Towards a 3D transmission format for the Web. 1–7. http://www.perey.com/ARStandards/[Klein]3dtf-position-paper_Ninth_AR_Standards_Meeting.pdf.

SONS, K., KLEIN, F., RUBINSTEIN, D., BYELOZYOROV, S., AND SLUSALLEK, P. 2010. XML3D: Interactive 3D Graphics for the Web. *Proceedings of the 15th International Conference on Web 3D Technology 1*, 175–184.

STAMOULIAS, A., MALAMOS, A.G., ZAMPOGLOU, M., AND BRUTZMAN, D. 2014. Enhancing X3DOM declarative 3D with rigid body physics support. *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies - Web3D '14*, ACM Press, 99–107.

STEIN, C., LIMPER, M., AND KUIJPER, A. 2014. Spatial data structures for accelerated 3D visibility computation to enable large model visualization on the web. *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies - Web3D '14*, ACM Press, 53–61.

SUTTER, J., SONS, K., AND SLUSALLEK, P. 2014. Blast: A Binary Large Structured Transmission Format for the Web. *Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies - Web3D '14*, 45–52.

THREE.JS. 2012. https://github.com/mrdoob/three.js/wiki/Features.

TO, D., LAU, R.W.H., AND GREEN, M. 2001. An Adaptive Multiresolution Method for Progressive Model Transmission. *Presence: Teleoperators and Virtual Environments 10*, 1, 62–74.

WEN, L. 2014. LPM : Lightweight Progressive Meshes Towards Smooth Transmission of Web3D Media over Internet. 95–103.

WORLD WIDE WEB CONSORTIUM. 2013. Web Workers. http://dev.w3.org/html5/workers/.

X3D. X3D/Canvas : Animation and Interactivty. http://www2.it.nuigalway.ie/~sredfern/CT404/05.pdf.

X3DOM. X3DOM Documentation: Tutorials. http://doc.x3dom.org/tutorials/.

XML3D. Tutorial « Animation : XML3D.ORG. http://xml3d.org/tutorial/#Animations.

YANG, S., KIM, C.-S., AND KUO, C.-C.J. 2004. A Progressive View-Dependent Technique for Interactive 3-D Mesh Transmission. *IEEE Transactions on Circuits and Systems for Video Technology 14*, 11, 1249–1264.

ZAKAI, A. 2011. Emscripten. *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion - SPLASH '11*, 301.