

3D Graphics on the Web: a Survey

Alun Evans*, Marco Romeo, Arash Bahrehmand, Javi Agenjo, Josep Blat

Interactive Technologies Group, Universitat Pompeu Fabra, Barcelona, Spain

Abstract

In recent years, 3D graphics has become an increasingly important part of the multimedia web experience. Following on from the advent of the X3D standard and the definition of a declarative approach to presenting 3D graphics on the web, the rise of WebGL has allowed lower level access to graphics hardware of ever increasing power. In parallel, remote rendering techniques permit streaming of high-quality 3D graphics onto a wide range of devices, and recent years have also seen much research on methods of content delivery for web-based 3D applications. All this development is reflected in the increasing number of application fields for the 3D web. In this paper, we reflect this activity by presenting the first survey of the state of the art in the field. We review every major approach to produce real-time 3D graphics rendering in the browser, briefly summarise the approaches for remote rendering of 3D graphics, before surveying complementary research on data compression methods, and notable application fields. We conclude by assessing the impact and popularity of the 3D web, reviewing the past and looking to the future.

Keywords: 3D, Graphics, Web, Survey, Internet, Browser, WebGL, X3D, X3DOM, Three.JS, Rendering, Meshes

1. Introduction

The Web has come a long way since its humble beginnings. Initially existing as a means of sharing remote pages of static text via the internet, bound together by a new markup language, HTML, the release of the Mosaic browser allowed users to 'browse' remote 'web-pages' which featured a mixture of both text and images [1]. Following the release of the Netscape Navigator and Microsoft's Internet Explorer in the mid nineteen nineties, development in web-technology exploded. Cascading Style Sheets (CSS), Secure Sockets Layer (SSL), cookies, Java and Javascript, Adobe Flash, XML, and AJAX (to name but a few of the enabling technologies); all were to be incorporated into the web model during the following years. By the turn of the millennium, the web was full of professionally designed pages, rich in visual content - recovering fully and to an unexpected extent both of initial goals of the web: browsing and editing content.

Yet full interactive multimedia content was invariably heavily reliant on Adobe Flash. As a proprietary system, Adobe had free reign to define the functionalities of its browser plugin [2], without necessarily making any reference to the series of standards that were being gradually undertaken and adopted by the World Wide Web Consortium (W3C). Adobe's closed system allowed developers to embed interactive audio, video, and

2D animations into a Flash executable file, which was then executed by the browser-plugin installed on the user machine, almost completely bypassing the browser itself. Flash-based applications typically required (relatively) long download times, and this fact, coupled with the later lack of availability for Apple's iOS platform, meant that Flash lost some of its initial popularity [3].

Meanwhile, the potential for creating interactive, multimedia rich web pages, using open standards methods, grew during the early part of the new millennium. Firstly, Scalable Vector Graphics (SVG) was introduced into browsers, allowing complex 2D drawing in a manner which fit in with the existing style of HTML. Then the *canvas* element was introduced, also allowing 2D drawing, but differing from SVG in being controlled via Javascript. First introduced by Apple as part of the WebKit framework, canvas later became incorporated into the draft HTML5 standard [4] (along with SVG). HTML5 is (being) specifically designed to adapt HTML in a way such that it can be used to make web *applications* i.e. dynamic interactive pages which are rich in multimedia content.

This consistent expansion of the scope, power, and complexity of the web has meant that technology previously reserved for custom software (or even hardware) is now being used via the web. One example of such technology is 3D graphics. While efforts have been made to implement real time 3D graphics over the internet since the mid-nineties (see Section 2 below), recent years have seen a rapid growth in its availability and distribution. This paper presents a survey of the current state of the art in real-time 3D graphics on the web, covering rendering techniques, scene description methods, 3D specific data delivery, and relevant application fields.

*Corresponding author

Email addresses: alun.evans@upf.edu (Alun Evans), marco.romeo@upf.edu (Marco Romeo), arash.bahrehmand@upf.edu (Arash Bahrehmand), javi.agenjo@upf.edu (Javi Agenjo), josep.blat@upf.edu (Josep Blat)

URL: <http://gti.upf.edu> (Josep Blat)

Why a survey into 3D web graphics? We see the first of its kind timely due to a new maturity in the subject. With all modern versions of major browsers now supporting plugin-free access to dedicated graphics hardware, and the ever-increasing power of this hardware, web-based 3D graphics applications finally have the opportunity to become a ubiquitous part of the web environment, accessible to everyone. Our goal in this paper is to both provide historical context in addition to assessing the current state of the art, which we attempt to do by reviewing the academic literature, assessing trends in the commercial sector, and drawing on our direct experience in using 3D web technology. Our hope is that the results of our survey will allow future developers and researchers to have a more complete understanding of the field as they proceed with their endeavours.

For the purposes of this survey, we define 3D graphics to be the use of 3D geometric data (usually through Cartesian coordinates) to perform certain calculations (for example, changes of form, animation, collision detection etc.) and to create 2D images suitable for display on a standard computer screen or monitor. The term *rendering* describes the process of converting the 3D data into a 2D image on a screen. Rendering techniques vary greatly in terms of their complexity, speed, photorealism and application. Photorealism is usually judged according to the realism of the 3D shape being rendered, and also how that shape is shaded (a synonym, in graphics terms, for coloured) with respect to light sources in the scene. Complex shading techniques, such as ray-tracing and radiosity, produce 2D images which are more photorealistic, at the cost of increased calculation time. By reducing the complexity of the algorithms used to simulate the effect of light in the scene, rendering time can be shortened to enable applications where the 2D image is updated at framerates fast enough to be undetectable (or barely detectable) to the human eye. One factor common to most 3D graphics is the use of perspective projection, where the projected size of any 3D object onto a 2D image is inversely proportional to its distance from the eye. The use of perspective projection, homogenous coordinates, and heavy use of 3D vectors in Cartesian coordinates to represent 3D objects, means that most 3D graphics applications make extensive use of matrix mathematics to both simplify and speed up computation.

The requirement to perform calculations on multiple data points (either for calculating the projection of 3D data points, or for calculating the colour of each pixel in a rendered 2D image) has led to development of specific class of hardware, the Graphics Processing Unit (GPU) which is designed to process several operations in parallel. The introduction of the modern GPU heralded an unprecedented reduction in the time taken to render 3D graphics in widespread systems, thus allowing more complex and photorealistic techniques to be applied in real-time. To harness the power of the GPU, access is usually facilitated via a low-level API such as OpenGL or Direct3D. OpenGL bindings exist for most major programming languages and most major platforms; whereas Direct3D is typically restricted to Microsoft platforms (Windows or Microsoft games consoles).

In this survey we repeatedly make reference to several basic terms. Although all of these will be familiar to the graphics community, for clarity we define them here. A real-time 3D

Scene features one or more objects, which can be represented in an implicit manner (such as using NURBS) or, more commonly, as polygonal *Meshe*s. Object appearance (colour, reflectivity, transparency etc.) is described by an associated *Material*. Materials frequently define colours according to *textures*: arrays of colour values usually loaded from 2D image files. These image files, along with any other file which is loaded into the scene (for example, to specify the structure or the behaviour of a Mesh), are frequently called *Assets*. A Scene also contains a viewpoint or *camera*, as well as one or more *light* sources, and may be organised into structure called a *Scene Graph*, where the contents of the scene are organised logically (e.g. imposing a hierarchy) and spatially. Scene Graphs are frequently used in 3D rendering engines to group scene objects together such that different rendering processes may be applied to different groups. The components of a *fixed pipeline* programme use algorithms predefined by the 3D programming API to calculate the position and colour of the assets in the Scene, depending on the point of view of the camera and location of any light sources. A *programmable pipeline* approach requires the programmer to carry out such calculations manually in a *shader*, which is a separate programme compiled at run-time and executed on the GPU. It follows therefore, that a programmable pipeline approach requires the programmer to have deeper understanding of the underlying mathematics, while also allowing a much finer level of control of the appearance of the final scene.

The remainder of this survey is organised as follows. We first review the state of the art in web based 3D rendering, including historical approaches (Section 2), before presenting an overview of the field of remote rendering (Section 3) and how it applies to the 3D web. We then review techniques of data compression and content delivery (Section 4), of vital importance to any client-server based scenario, but especially relevant to 3D applications where file sizes are typically large. Following this we present a selection of standout applications, grouped by application field, which have contributed to the state of the art (Section 5). Finally, Section 6 discusses the rise in the popularity of 3D web and concludes the paper.

2. Browser-based Rendering

Jankowski et al [5] classify browser-based 3D rendering (i.e. where the client browser/machine executes the rendering process) into declarative and imperative techniques, creating a matrix which classifies 2D and 3D graphics according to the different paradigms. Figure 1 is an adaptation of their classification. For example, it is possible to draw 2D graphics within the browser using both SVG and the HTML5 canvas element: SVG [6] is an XML-based file format for two dimensional vector graphics - drawing and image filter effects are coded in a declarative manner. By contrast, similar drawing and image processing can be achieved by using the `<canvas>` element in an imperative way using Javascript. Jankowski et al extend this declarative/imperative distinction into the world of 3D browser-based graphics, referencing some of the approaches which we discuss below.

Approach	Requires Plugin	Part of HTML Document	DOM Integration	Inbuilt Scene Graph	Customisable Pipeline*	Standards-based
X3D	yes	no	no	yes	no	yes
X3DOM	no	yes	yes	yes	no	yes
XML3D	no	yes	yes	yes	no	partial
CSS Transforms*	no	partial	yes	no	no	yes
O3D	no longer	no	no	yes	no	partial [†]
Three.JS	no	no	no	yes	yes	partial [†]
Stage3D (Flash)	yes	no	no	no	yes	no
Java	yes	no	no	yes [‡]	yes [‡]	no
Silverlight	yes	no	no	no	yes	no
Native WebGL	no	no	no	no	yes	yes
Unity3D [✱]	yes	no	no	yes	no	no

Table 1: Approaches to browser-based 3D rendering, classified according to level of declarative behaviour. * A customisable pipeline is one where the programmer has low-level control over the render targets, render order, scene graph, materials and associated shaders. † While CSS transforms are not true rendering per se, they are included for completeness. ‡ Both O3D and Three.JS can render via WebGL, which is standards based. ✱ Only certain Java libraries and APIs have these functionalities. ✱ Unity3D is included as a (popular) example of proprietary games engines (see section 2.12)




	2D	3D
Declarative Scene Graph Part of HTML Document DOM Integration CSS/Events		Declarative 3D for the Web Architecture Community Group 
Imperative Procedural API Drawing Context Flexible	<canvas>	

Figure 1: Declarative vs Imperative approaches to web-based graphics (adapted from [5])

The declarative/imperative distinction for web-based 3D graphics is useful, particularly as it allows comparison to the equivalent 2D cases. Nevertheless, when considering the broad spectrum of browser-based 3D rendering techniques, it becomes difficult to impose such a strict classification. For example, many of the approaches surveyed in this section are based on programming languages which follow an imperative paradigm, yet some of them also make heavy use of a *Scene Graph*, which could be considered a declarative construct. Table 1 shows a comparative summary of the browser-based 3D rendering approaches surveyed in this paper, particularly from the point of view of their classification as declarative or imperative. An approach can be said to be more declarative than imperative if:

- It is part of the HTML document (i.e. it uses a text format to declare content without direct context)
- It is capable of integrating with the Document Object Model (DOM)
- It features a scene graph
- It does not allow overloading of the rendering pipeline

- It has a high level of platform interoperability

Other important characteristics, such as the requirement for plugins, and whether the approach is standards based or not, are also listed in the table. The remainder of this section surveys each of these technologies in turn.

2.1. VRML, X3D and ISO Standards

In 1994, David Raggett, in parallel with Mark Pesce and Tony Parisi called for development the Virtual Reality Modeling Language (VRML) file format to describe a 3D scene [7], and the format (in its second version) became an ISO standard (ISO/IEC 14772-1:1997) in 1997. VRML97, as it became known, uses text to specify the both the contents and appearance of a scene (i.e. associating a mesh with a particular material) but does not allow the specification of more advanced 3D concepts such as NURBS (implicit surfaces) or complex animation. Shortly after the definition of VRML97, and in order to protect it as an open standard, the Web3D consortium was formed, as a cross-section of businesses, government agencies, academic institutes and individuals. Several applications and browser plugins were developed to enable the display of VRML scenes in the browser, such as Cosmo Player, WorldView, VRMLView, and Blaxxun Contact. In many respects, these applications formed the first efforts to bring 3D graphics to the internet.

Despite these efforts, support for the format was intermittent [7], and in 2004 VRML was replaced by X3D (ISO/IEC 19775/19776/19777). While designed to be backward compatible with VRML, it provides more advanced APIs, additional data encoding formats, stricter conformance, and a componentized architecture [8]. The most immediate difference to VRML is the syntax: while X3D still supports the traditional VRML 'C-like' syntax, it also adds support for binary and XML formats. The latter is important as it is a standard format widely used in web-technologies, and thus brings X3D closer to the web. X3D supports a multi-parent scene-graph and a runtime, event and behaviour model; as well as a dataflow system which

integrates sensor, interpolation and visualisation nodes. All this permits the description of animated scenes in a declarative manner and without using imperative scripting techniques. X3D is designed to be used in both web and non-web applications; in that sense, it can be said more precisely that X3D is concerned with *internet-based* 3D as opposed to purely web-based. As such, an X3D enabled application (whether standalone software, or browser plugin) is required to view and interact with X3D scenes. Web browser integration involves the browser holding the scene internally, allowing the plugin developer to control and update content via a *Scene Access Interface*.

2.2. X3DOM

In an attempt to extend browser support for X3D, in 2009 Behr et al [9], introduced X3DOM. X3DOM provides native browser, plugin-free (where possible) and independent 3D capabilities, and is designed to integrate closely with standard web techniques such as AJAX. It maintains the declarative nature of X3D and is, in effect, an attempt to code X3D directly into a web page, without the need for a browser plugin. X3DOM is defined as a front-end and back-end system, where a connector component transforms a scene defined in the front-end (the DOM) to the backend (X3D). The connector then takes responsibility for synchronising any changes between the two ends of the model. X3DOM defines an XML namespace [10], to enable an `<x3d>` tag (and all of its children) to be used in a standard HTML or XHTML page. An example of the code required to draw a scene featuring a sample mesh is shown below, starting from the initial `<body>` tag of an XHTML file (vertex data of the object is redacted for brevity). The result of this code, when loaded in a web browser, is shown in Figure 2:

```
<body>
  <X3D
    xmlns="http://www.web3d.org/specifications/x3d-namespace"
    width="400px" height="400px">
    <Scene DEF='scene'>
      <Viewpoint position='0 0 300'
        orientation="0 0 0 1" />
      <Background skyColor='0.6 0.6 0.6' />
      <Transform translation='0 10 0' >
        <Shape>
          <Appearance DEF='App_0'>
            <Material diffuseColor="0 1 0" shininess='0.15625' />
          </Appearance>
          <IndexedFaceSet creaseAngle='4' coordIndex='
            [indexed vertex coordinates for model]
            ' />
          </IndexedFaceSet>
        </Shape>
      </Transform>
    </Scene>
  </X3D>
  <script type="text/javascript" src="x3dom.js"></script>
</body>
```

This code snippet uses hierarchical declaration to create a 3D `<scene>`. A viewpoint position and orientation is set, as is the background colour. A `<transform>` tag defines translation, rotation and scaling transforms which apply to the tag contents (in this case all the content defined within it should be translated by 10 units in the y-axis). The `<shape>` tag defines an

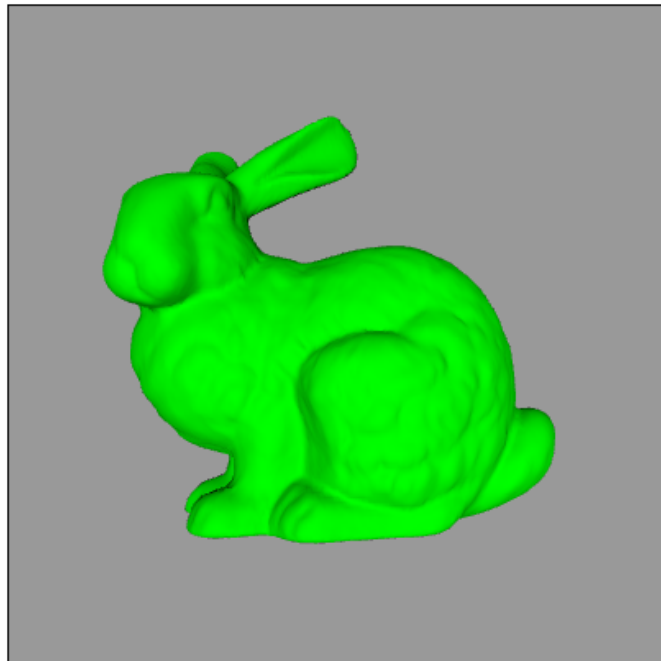


Figure 2: Screenshot taken from a Mozilla Firefox browser window executing the X3DOM code shown above.

object in the scene, within which the objects properties (such as the mesh and appearance) are defined in parallel. The mesh in this scene is defined as a set of indexed faces; no normal vectors are supplied so X3DOM calculates them to enable lighting effects.

For rendering the scene X3DOM defines a fallback model [11], trying to initialise preferred renderers first, and using alternative technology if required. Currently the fallback model first checks whether the user application is capable of rendering X3D code natively, and is so, passes the contents of the `<x3d>` tag to the X3D rendering engine defined by the host application. If native X3D rendering is not detected, or an X3D browser plugin is not installed, the application tries to create a WebGL rendering context, and build the X3D scene graph on top of that. With the partial integration of WebGL into Microsoft Internet Explorer 11 [12], the majority of modern browsers now support browser based 3D rendering via WebGL (see below). If WebGL is not supported, X3DOM tries to render with Stage 3D (see Section 2.10 below) before finally presenting a non-interactive image or video if no 3D rendering is possible.

X3DOM is designed to integrate with the DOM and thus can be modified in the same manner as any DOM object. For example, if the programmer wished to modify the position of the shape at runtime (for example, in response to user input), the translation attribute of the `<transform>` tag can be modified at runtime using Javascript, and the scene will update accordingly.

3D models can be loaded into X3D/X3DOM by defining vertex geometry as in the example above, or by importing an X3D (or VRML) scene defined in a separate file. This scene file can be coded manually or, for complex objects and scenes, it can be created using custom export plugins for 3D digital

content creation (DCC) packages (such as Blender, Autodesk Maya, and Autodesk 3D Studio Max). Basic shading (such as diffuse or specular shading) is supported in a declarative manner. Schwenk et al [13] [14] added a declarative shader to X3D, enabling advanced 3D effects and lighting (such as multi texturing, reflections and other lighting effects), this is now partially supported in X3DOM. Custom shaders are supported via the ComposedShader node; this allows the programmer to write their own shader code as long as uniform data is restricted to a supported set and naming conventions. In 2011, Behr et al extended their framework with various functionalities [15]. Several camera navigation methods were introduced, defining a *Viewpoint* node (seen in the above example) and several different navigation patterns (definition of custom camera navigation is also supported via manipulation of the equivalent DOM node, for example, via Javascript). Also introduced was simple object animation, either via CSS transforms and animations (see below) or by the X3D Interpolators.

2.3. XML3D

XML3D [16] is similar to X3DOM, in that it is an extension to HTML designed to support interactive 3D graphics in the browser, requiring the use of XHTML. Its stated goal is to try to find the minimum set of additions that fully support interactive 3D content as an integral part of 2D/3D web documents. It takes a similar high-level approach to 3D programming as X3DOM: both use declarative language to define a 3D scene, and both expose all elements of the 3D to the DOM (with the benefits of DOM manipulation that arise thereafter). The central difference between the two approaches is that X3DOM grew out of an attempt to natively embed an existing framework (X3D) into a browser context, whereas XML3D proposes to extend the existing properties of HTML wherever possible, embedding 3D content into a web page with the maximum reuse of existing features. The sample XML3D code below draws a similar scene to the X3D code above (for brevity, standard html code such as the body and head tags are excluded, along with lines to load XML3D libraries and the data for the mesh):

```
<xml3d id="BunnyXML3D" activeView="#defaultView"
class="xml3d" style="width: 400px; height: 400px;" >

<defs id="mainDef">
  <transform id="t_Light" rotation="20 22 92 190"
    scale="1.0 1.0 1.0" translation="400 100 600">
  </transform>
  <transform id="t_Bunny" rotation="0.0 0.0 0.0 0"
    scale="1.0 1.0 1.0" translation="0.0 0.0 0.0">
  </transform>

<data id="mesh_bunny">
  <float3 name="position">[vertex coordinates for model]
  </float3>
  <float3 name="normal">[normals for model]
  </float3>
  <int name="index">[indices for model]</int>
</data>

<lightshader id="ls_Spot"
  script="urn:xml3d:lightshader:point">
  <bool name="castShadow">true</bool>
  <float3 name="attenuation">1.0 0.03 0.00</float3>
```

```
    <float3 name="intensity">1.0 1.0 1.0</float3>
  </lightshader>
  <shader id="Material" script="urn:xml3d:shader:phong">
    <float name="ambientIntensity">0.0</float>
    <float3 name="diffuseColor">0.4 0.12 0.18</float3>
    <float3 name="specularColor">0.5 0.5 0.5</float3>
    <float name="shininess">0.2</float>
  </shader>
</defs>

<view id="defaultView" position="0 0 300"></view>

<group shader="#Material" transform="#t_Bunny">
  <mesh src="#mesh_bunny_noMat" type="triangles"/>
</group>

<group transform="#t_Light">
  <light shader="#ls_Spot"></light>
</group>

</xml3d>
```

In this example, a `<defs>` tag allows the definition of various transforms, the data for the mesh to be displayed, and shaders and their uniforms, along with a light (position and intensity), and the eye view position. Transforms are defined as a tag, as in X3DOM, although they could be specified using CSS transforms (see below) for those browsers that support them, fulfilling the authors' goal to reuse existing features as much as possible. Shaders can also be defined using a fixed number of CSS properties (again, with the goal of extending existing functionality). Once all the definitions are finished, the viewpoint is defined, and mesh and light added to the scene. In this case, the 3D mesh is defined in line, but 3D objects can also be loaded from an external file with the mesh data - stored, for example, in a JSON, XML or binary file.

XML3D also introduces another level of abstraction, seen in the example above, by using *Data Containers*, defining a `<data>` tag which wraps the primitive data [17]. This definition of a declarative data containers allows the piping of the data to processing modules (see Section 2.4 below) which in turn permits the complex geometry transformations required for dynamic meshes and certain graphical effects.

XML3D rendering is implemented both in WebGL and also as a native component for the Mozilla browser framework (supporting Firefox) and Webkit-based browsers (supporting Google Chrome and Apple Safari). The native component was apparently implemented in order to avoid any performance issue with the Javascript/WebGL implementation (specifically, slow Javascript parsing of the DOM, and limitations of the OpenGL ES 2.0 specification).

2.4. Xflow

The needs of interactive 3D graphics applications go beyond simply drawing and shading meshes. Techniques such as character animation, physical material simulation, and rendering complex effects such as smoke or fire typically require a considerable amount of runtime data processing. X3DOM, as previously mentioned, passes such issues to the X3D backend and synchronises the results to the DOM front end, yet this option is hamstrung by Javascript's slow parsing of the DOM, and

furthermore is not available to alternative approaches such as XML3D. Thus, in 2010 Klein et al. proposed a declarative solution to intensive data processing on the web, called Xflow [17]. Xflow is a system for exposing system hardware in a declarative manner and allowing dataflow programming. Examples of 3D graphics applications for this technique are modifying vertex data for animated or otherwise dynamic meshes, animation of shader parameters, and image processing and post-processing.

In essence, Xflow can be thought of as a data processing extension to XML3D. Xflow is integrated into XML3D and provides a crucial extension by defining a `compute` attribute for defined data containers. This attribute is used to provide processing operators to the data defined within a container (for example, for skeletal animation).

2.5. CSS 3D Transforms

CSS transforms are a part of the W3C specification which allow elements styled with CSS code to be transformed in two or three dimensional space. Originally developed by Apple as part of the WebKit framework [18] (and thus functional in Safari and Chrome browsers), these transforms are now fully supported by Mozilla based browsers and almost fully supported in Internet Explorer.

3D transforms work by initially setting a perspective to the scene (using `transform: perspective(value);` or more simply `perspective: value;`). Once set, standard 3D transforms such as translation, rotation and scale can all be applied in three axes. By using CSS *Transitions* [19], simple animation between different states can be achieved. Object shading must be specified manually by defining colour or texture information (as an image) as with standard CSS.

CSS 3D transforms provide a very fast, easy to understand method of coding simple 3D effects into a webpage. One interesting aspect of their use is that existing 3D content, such as a DOM element (for example, a canvas) which features 3D graphics rendered by other techniques, can be further transformed using CSS 3D. Nevertheless, the lack of true lighting and shading capabilities means CSS 3D is very limited in what can be achieved with more powerful declarative solutions such as X3DOM or XML3D.

2.6. Collada

Collada [20] is a declarative file format used for describing a 3D scene, and while it does not feature any form of runtime or event model, its popularity as an interchange format for web based 3D graphics means that it is included in this section for completeness. Initially developed by Sony and Intel, it is now managed by the Khronos Group (see WebGL section below) and is an open standard with ISO/PAS 17506. Apart from describing basic object parameters such as shape and appearance, Collada also stores information about animation and physics. Superficially, it is similar to X3D, in that both define XML schemas for describing 3D scenes, and were specifically designed for transfer of digital assets in a standardised manner. X3D scenes, however, are designed to be rendered with specific X3D capable software (such as browser browser plugins



Figure 3: Screenshot from the Barcelona World Race browser-based MMO game [23], rendered using O3D (reproduced with permission)

or X3DOM), whereas Collada is designed purely to be a format for data interchange, and is agnostic as to the rendering engine used to draw the scene. In 2013, the Collada working group announced the project to define and create the *glTF* format [21], which is designed to better match WebGL processing requirements (for example, using typed arrays, and storing geometry and texture assets in binary format).

2.7. Javascript access to graphics hardware

In contrast to declarative or functional programming, the paradigm of imperative programming describes computation as a series of statements which change a programme's state. It could be argued that most computer programming must eventually reduce to imperative code, in the sense that most low level hardware programming (for example with Assembly language) is imperative. The traditional language for executing non-declarative code in the web browser is Javascript, which has elements of the imperative, object-orientated, and functional programming paradigms. According to Tony Russell, the Chair of the WebGL working group, one of the biggest contributing factors to rise of imperative 3D web programming are the huge performance increases in the Javascript Virtual Machine, allowing fast control and manipulation of thousands of vertices in 3D space, in every drawn frame [22].

Although it is possible to create a web-based software rendering algorithm using SVG [24] or the HTML5 Canvas, towards the end of the first decade of the 21st century efforts were being made to allow imperative programming access to dedicated graphics hardware. Principal among these was Google's O3D library [25] [26]. Developed as a cross-platform plugin for all major browsers and operating systems, O3D originally provided a Javascript API to enable access to its plugin code (written in C/C++), which in turn allowed programming of the graphics hardware, either via Direct3D or OpenGL (the decision was hidden from the final user). In order to popularise the technology and to act as tutorials, Google published an extensive suite of demo web applications made using O3D [25]. Perhaps one of the most visible applications of the technology, in terms of number of users, was the official MMO game for the Barcelona World Race sailing regatta, which used O3D to power its 3D component (see Figure 3). The game developers

noted [23] that O3D support for multi pass rendering (important for post-processing effects such as shadows, reflections and weather) were critical in the decision to use the API ahead of other options such as the nascent Stage 3D from Adobe (see below). This conclusion was in agreement with Sanchez et al. [27], who concluding that O3D demonstrated better performance than X3DOM, both in terms of rendering performance (where frustum culling was possible) and animation. With the advent of WebGL, the O3D plugin was discontinued, and the API was ported to use the WebGL renderer; non-rendering elements of O3D (for example, the Scene-Graph and Events API) are still accessible and usable.

Besides O3D, other efforts were made to allow access to the GPU via Javascript. Canvas3D was a Firefox plugin from Mozilla which allowed the creation of an OpenGL context within a HTML5 canvas element, and was essentially the precursor to WebGL (see below). Canvas3D JS [28] [29] was a mid-layer API designed to simplify its use, and was later adapted to make use of WebGL (see below). Opera Software (creators of the Opera web-browser) also released a similar plugin [30] which allowed access to OpenGL calls via the HTML5 canvas.

All of the above efforts use some form of browser plugin, usually programmed in C or C++, which essentially acts as a Javascript wrapper for accessing the graphics hardware via OpenGL or Direct3D. By 2009, the need to standardise methods for accessing the GPU via the browser had become clear. Thus, the Khronos Group (the not-for-profit organisation responsible for, among other things, the OpenGL specification) started the WebGL working group, and with input from Mozilla, Apple, Google, Opera and others, released the version 1.0 of WebGL in 2011 [31].

2.8. WebGL & Associated Libraries

Khronos describes WebGL thus:

WebGL is a cross-platform, royalty-free web standard for a low-level 3D graphics API based on OpenGL ES 2.0, exposed through the HTML5 Canvas element as Document Object Model interfaces [31].

OpenGL ES ("Embedded Systems") 2.0 is an adaptation of the standard OpenGL API, designed specifically for devices with more limited computing power, such as mobile phones or tablets. WebGL is designed to use, and be used in conjunction with, standard web technology; thus while the 3D component of a web page is drawn with the WebGL API via Javascript, the page itself is built with standard HTML.

WebGL is purposefully built to be a lean, reasonably low level API - indeed, the two principal declarative methods mentioned above (X3DOM and X3D) both use or have used WebGL to some extent in their implementations. WebGL is targeted at the experienced graphics programmer, who has a good knowledge of core graphics concepts such as matrix and vector mathematics, shading, and preferably organisation of 3D scenes with Scene Graphs [22]. The process required to create a simple box is very similar to that for any OpenGL ES 2.0 renderer: create a WebGL context (in this case, on a HTML5 Canvas element); define and load a vertex and a fragment shader and bind any

uniform data; bind data buffers and pass data (for vertices, and optionally normals, colours and texture coordinates); control camera and perspective using standard model-view-projection matrices; and finally draw.

The Khronos group provides several helper Javascript files which assist in the setup and debugging of WebGL applications, which are contained within many of the public demos publicly available on the Khronos site [32]; nevertheless, as mentioned above, WebGL is clearly aimed at the more experienced graphics programmer. Directly comparing it to the declarative techniques is not particularly worthwhile, as they target different sections of the developer/artist communities.

A detailed analysis of WebGL is beyond the scope of this paper, and the reader is directed to several recent books [22] [33] [34]. However, it is interesting to introduce several Javascript libraries, whose goal is to abstract WebGL inner workings and produce higher level code, which have proliferated as a result of WebGL's steep learning curve. One of the first of these libraries to appear was SpiderGL [35] [36]. Its original version consists of five libraries: *GL*, which abstracts core WebGL functionalities; *MESH*, defining and rendering 3D meshes; *ASYNC*, to load content asynchronously; *UI*, to draw the user interface within the GL context; and *SPACE*, a series of mathematics and geometry utilities. Despite abstracting many WebGL functions, programming an application in SpiderGL still requires knowledge of 3D concepts (more than those required to use XML3D, for example) and requires at least some basic knowledge of OpenGL. After its initial impact as the first comprehensive abstraction of WebGL, SpiderGL entered a period of extensive refactoring, from which it emerged with several lower level improvements [36], such as custom extensions and better integration with other libraries.

LightGL is a low-level wrapper which abstracts many of the more code intensive WebGL functionalities, while still requiring shader programming and matrix manipulation. Agenjo et al. [37] modify LightGL to use the popular GLMatrix library [38] to create a tiered API with several layers of abstraction. OSG.JS [39] is a WebGL framework which aims to mimic an OpenSceneGraph [40] approach to 3D development. Other libraries of note are SceneJS [41], PhiloGL [42], and GLGE [43]; the latter providing a declarative method of programming a 3D scene, much like X3DOM or XML3D.

2.9. Three.JS

Perhaps the most famous library/API for web-based 3D graphics is ThreeJS [44] [45]. Although originally developed in ActionScript (see below), it is now an open-source Javascript library which enables high-level programming of browser-based 3D scenes, such as that shown in Figure 4. Its modular structure means that several different rendering engines (WebGL, Canvas and SVG) have been developed to render scene content, and the library can be (and is being) extended in a distributed manner by several dozen individual contributors. Three.JS features a scene graph, several types of camera and navigation modes, several pre-programmed shaders and materials (and the ability to program custom shaders), Level-of-Detail mesh loading and



Figure 4: Car visualisation running in Mozilla Firefox, created using ThreeJS by Plus 360 Degrees [46] (reproduced with permission)

rendering, and an animation component allowing skeletal and morph-target animation.

The sample code below demonstrates the steps required to load a mesh with a simple material, similar to the previous two code examples (code is Javascript/JQuery, and HTML setup left out for brevity):

```
var WIDTH = 400, HEIGHT = 400;
var $container = $('#container');
var scene = new THREE.Scene();

var renderer = new THREE.WebGLRenderer();
renderer.setSize(WIDTH, HEIGHT);
$container.append(renderer.domElement);

var VIEW_ANGLE = 45, ASPECT = WIDTH / HEIGHT;
var NEAR = 0.1, FAR = 10000;
var camera = new
    THREE.PerspectiveCamera(VIEW_ANGLE, ASPECT, NEAR, FAR);
camera.position.z = 300;

var sphereMaterial = new THREE.MeshLambertMaterial(
{
    color: 0xCC0000
});

var pointLight = new THREE.PointLight( 0xFFFFFF );
pointLight.position.x = 10;
pointLight.position.y = 50;
pointLight.position.z = 130;

var loader = new THREE.ObjectLoader( );
loader.load( 'bunny.obj', function ( object ) {
    object.traverse( function ( child ) {
        if ( child instanceof THREE.Mesh ) {
            child.material = sphereMaterial;
        }
    } );
    scene.add( object );
} );

scene.add(camera);
scene.add(pointLight);
```

```
animate();
function animate() {
    requestAnimationFrame(animate);
    renderer.render(scene, camera);
}
```

Beyond the obvious changes in language syntax, the Three.JS approach is not dissimilar to those of X3D and XML3D described above, particularly XML3D. A Material, a Mesh, a Camera and Light must all be defined by being added to the scene. In this example the mesh is loaded from an external file (using the ubiquitous Wavefront Object format) via an asynchronous loading method. Once the mesh file has been downloaded and the mesh object created, it is assigned a material and added to the scene using a callback function. This ability to call functions is a major difference in comparison with declarative examples above, and is seen again in the final few lines, which are specific instructions to request a new animation (or drawing) frame from the browser, and render. Without this final imperative part, the code would download the mesh object correctly and add it to the scene, but nothing would appear in the viewport, as there would have been no command to render the scene executed after the object was downloaded.

2.10. Stage 3D

As mentioned above, Adobe's Flash plugin is a proprietary system which allows multimedia content to run inside a web page which has the Flash plugin enabled. Although there were initial attempts to embed 3D graphics into Flash (such as the now disappeared Papervision [9]) these relied on software rendering techniques which did not allow access to the GPU. *Stage 3D* is Adobe's proprietary 3D engine [47], with the key difference being that it allows Flash and AIR applications to draw hardware accelerated 3D graphics. Stage 3D applications are written in ActionScript, an object-oriented language developed to write Flash-based applications. Stage3D is marketed as a medium-low language for 3D graphics programming, allowing platform independent programming of applications that are fully compatible with existing Flash libraries. It is quite low level in that a sample application must deal directly with vertex and index buffers, shaders, and the Model/View/Projection matrices common to many 3D applications, and there is no built-in scene graph. The application code is compiled against relatively high level libraries, allowing drawing to contexts which allow seamless merging with 2D Flash, and/or Flash Video contents; this means that many low-level hardware aspects are hidden from the programmer by proprietary libraries. Shaders in Stage 3D are written in Adobe Graphics Assembly Language (AGAL), a very low level assembly language, which makes writing shaders for Stage 3D a more laborious task compared with writing shaders in a higher level language such as GLSL or HLSL, which are the shading languages for OpenGL and Direct3D, respectively. Adobe has made efforts to develop a higher level shader creation package called Pixel Bender, though as of 2013, development of this tool appears to have stalled.

2.10.1. Silverlight

Microsoft Silverlight is an API for developing web-based applications, not dissimilar to Adobe Flash. It facilitates the creation of interactive multimedia applications and their distribution via web, with client side execution depending on a browser plugin which the user must install. Silverlight applications are created using Microsoft's .NET framework. Version 3 of Silverlight introduced basic 3D transforms (similar to modern CSS 3D transforms), but the current version (version 5) now features a full programmable graphics pipeline, allowing access to the GPU and shader programming.

2.11. Java

The Java platform, developed initially by Sun Microsystems before its merger with Oracle, is now an established part of modern computing. From a web perspective, the ability to launch a *Java applet* (a small application which is executed within the Java Virtual Machine) within the browser, was one of the earliest ways to programme more computation-expensive visualisations [48]. In particular, it is possible to allow Java applets to access the GPU which, prior to the advent of WebGL, was one of the earlier methods of accessing hardware acceleration from a web page without relying on a custom browser plugin.

In 1998, the Java3D API was released to facilitate 3D development with Java [49]. Java3D features a full scene graph and is able to render using Direct3D or OpenGL. Development on Java3D was abandoned in 2008 as Sun Microsystems moved to push its new JavaFX platform, though development on Java3D has been restarted by the JogAmp community [50]. For lower level access (i.e. wrapping OpenGL functions directly) other libraries exist such as JOGL [50] or LWJGL [51]. The latter is the basis of the popular multiplatform game, Minecraft, whose well documented success demonstrates that a Java based approach to web 3D graphics can be a viable option for many developers.

2.12. Proprietary Videogame Engines

The video game industry has long been a showcase for the latest graphics technology (and a driving force for the constant performance increases and lower prices of GPUs), and it is not surprising that there have been commercial efforts to make 3D video games available via the browser. However, the historical lack of full cross-browser support for WebGL (only recently overcome with the release of Microsoft Internet Explorer 11), and the natural desire of companies to target the widest possible user base, has meant that WebGL-based 3D gaming is still in its infancy (see also Section 5.3).

Unity is a cross-platform game engine featuring advanced 3D graphics capabilities [52]. Compared to the other technologies described thus far, Unity is much higher level, aiming to be an application for creating video games (and, thus, more than simply a 3D engine for the web). Nevertheless, Unity's ease of use in creating a 3D scene and exporting it to a webpage, and popularity among the public web (225 million claimed installs of the web-player plugin, as of 2013 [53]) merits its inclusion in

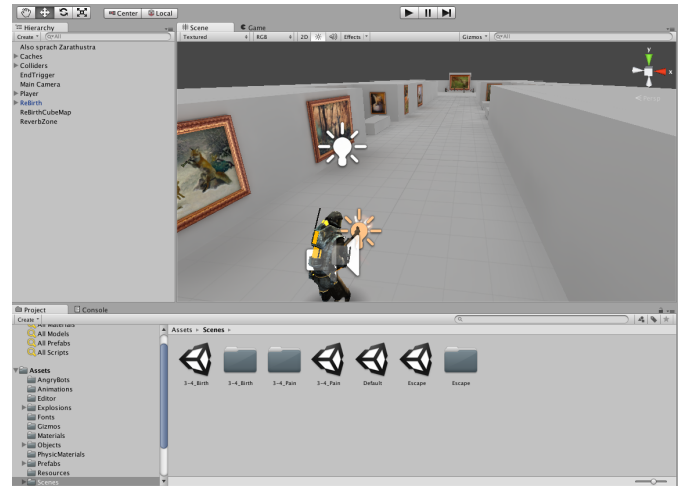


Figure 5: The Unity IDE and game engine has become a popular way to quickly embed 3D graphics into a web-browser

this paper. Unity's technology is split into two applications - an Integrated Development Environment (IDE) (see Figure 5) for the creation of a scene/game, and a Player plugin/standalone application which allows Unity scenes/applications developed with the IDE to be executed on a target platform. At its most basic level, creating a 3D scene for the web in Unity involves dragging and dropping 3D assets into a viewport, optionally setting material, light and camera settings, and then exporting the scene to a custom file format. The Unity web-browser plugin (available for all major browsers) reads this file and display the scene in the browser window. Beyond the drag and drop capabilities of the IDE, a Unity scene can be modified (or even created) entirely in code, either via Javascript, C# or Boo. Unity also allows exporting applications to other platforms, including mobile platforms such as iOS and Android.

Other companies have made short-term efforts to present video games as browser-based experience; for example Epic Games' Unreal Engine has in the past featured a web player component and now has a HTML5/WebGL-based demo available [54]; while the game Battlefield Heroes [55] from DICE is a purely browser based experience which requires downloading a plugin.

3. Remote Rendering

The concept of remote rendering involves the server-based generation of data, which is then passed to a client for visualisation. Much of the research in this field does not apply directly to web-based 3D graphics, in that it focuses on client-server rendering systems where the client is usually a non-web application. However, the increasing use of the web-browser as a 3D graphics client (as demonstrated in this paper) means that we consider it appropriate to include a brief survey of remote rendering techniques.

Commercially, remote rendering of 3D graphics for video game purposes has been exploited by at least two startup companies, Onlive [56] and Gaikai [57], the latter being purchased

by Sony in 2012 [58] for \$380 million. From a research perspective, it is possible to roughly classify the different approaches to remote rendering into three areas: Graphics Commands, Pixels, and Primitives or Vectors. A fourth server-based process, that of parsing, segmenting and/or simplifying 3D objects, also falls under the remote rendering definition, but is not included in this section as it is reviewed in detail in section 4 below.

3.1. Graphics Commands

Low-level draw calls to the server GPU are intercepted and passed to the client [59], which then renders and displays the final image. This technique has been adapted by Glander et al. [60] for parallel WebGL visualisation of existing 3D desktop applications, using AJAX to synchronise the two rendering applications. Furthermore, there are efforts being made to directly convert C/C++ code (via LLVM bytecode) to Javascript [61].

3.2. Pixels

The server renders the image and passes it directly to the client for display [62] [63] [64]. This basic method of remote rendering can be viewed as a generic data transfer issue, essentially sampling the remote system graphics buffer and sending it to the client as a video stream [65]. Several optimisations can be made to this technique, such as synchronising a high-end server render with a low-end client render [66], selectively transmitting pixels [67], or optimising the video encoding to take advantage of GPU-rendered scenes [68].

3.3. Primitives or Vectors

Feature extraction techniques are used on the server to obtain vectors to be passed to the client to render, either in 2D [69] or 3D [70]. The advantage of this method is that it that client devices which do not have any native 3D capabilities can render 3D objects from the passed vector data, as the costly 3D transformations are being executed by the server.

3.4. Combined Techniques

Finally there have been efforts which combine several of these techniques, such as the parallel client-server image-based rendering by Yoon [71], or the Games@Large platform [72]. The latter captures the rendering instructions of an application at run-time, and sends changes in that scene to the remote client, which is rendering the scene locally. If the client is not capable of rendering the scene locally, a video stream is sent instead.

As scientific datasets become ever larger, the challenge of viewing and interacting with them locally has increased, and there has been a move to storing the data on central parallel clusters, and interacting with data via remote clients. ParaView [73] is a multi platform, open source data analysis and visualisation application, built on top of VTK and extended to support parallel cluster rendering. Recently it has been extended to enable 3D visualisation via a web-based context, called ParaViewWeb [74], which allows a user to access a ParaView rendering cluster from within a web page.

One concrete application of remote rendering is in collaborative visualisation, as the same scene must be rendered in

real-time to multiple users, via either a client-server model, a peer-to-peer model, or a hybrid of the two. The Resource-Aware Visualisation Environment (RAVE) [75] was created in order to demonstrate whether web services were capable of supporting collaborative visualisation (not dissimilar to the Games@Large approach). It uses a Java applet on the client to determine the client's rendering capabilities - less powerful clients receive a video-feed from a remotely rendered scene, whereas more powerful clients receive the polygonal dataset to render it locally. The system was extended to support the X3D format with ShareX3D [76], which was the first implementation of a collaborative 3D viewer based on HTTP communication. A more comprehensive survey of collaborative visualisation systems, including some applications to the 3D web, is presented in [77].

4. Data Compression & Delivery

For the majority of the techniques for browser-based 3D graphics described above, the 3D data are represented by polygon (usually triangle) meshes, composed of vertices and faces. Such data, when describing an object in great detail, can be relatively large in size if unoptimised. For example, a laser scan of a large 3D object will result in a file which is hundreds of megabytes in size. Large file size has serious performance implications (both in terms of parsing and rendering the data); to offset/reduce these implications, there has been considerable research effort made into mesh optimisation techniques [78] [79], and mesh segmentation [80]. However, many mesh compression methods are highly geared towards dealing with specific type of data, and are not designed to handle arbitrary meshes including normal and texture information i.e. meshes for the web [81].

4.1. Compression & Optimization

Progressive Meshes (PM), proposed originally in 1996 by Hoppe [82], allow continuous, progressive refinement of a polygonal mesh during data transmission over a network. PM works by initially transferring a coarse mesh from server to client, then progressively refining the mesh using a stream of vertex-split operations, until the high resolution mesh has been recreated. Limper et al. [81] summarise reported decode times and compression performance for the original PM technique and several of its subsequent refinements. They found that Hoppe's 1998 implementation of Progressive Meshes [83] still provides the fastest decompression time, despite several attempts to improve upon it, and not taking into account the improvements in hardware capabilities since 1998. On the other hand, the compression factor (i.e. how many bits each vertex occupies) is an order of magnitude less for more modern techniques - for example, that of Maglo et al. [84]. The conclusion therefore is that mesh compression research over the last decade has focused more on improving pure compression (rate-distortion) performance, and less on its speed. With this in mind, and combining the conclusion of both Limper et al. [81] and Lavoué et al. [85], it is possible to draw a series of requirements for mesh compression

for web-based 3D rendering, separate from the requirements of more general mesh compression:

Rate distortion versus decompression speed Mesh compression techniques for web-based 3D graphics present a special case where both decompression speed and download speed contribute to the overall performance of a given technique. An algorithm which achieves very high compression rates (for example, that of Valette et al [86]) may not be suitable for a web-based system, in that the decompression rates are comparatively slow [81]. This problem is particularly relevant currently, where despite increasing network bandwidth (allowing rapid data transfer), there has been a parallel increase in low-power mobile devices which may struggle to execute complex decompression algorithms.

Browser-based decompression On the other hand, plugin-free browser-based 3D may necessitate decompression using Javascript. For all the improvement in Javascript execution time in recent years, it is still slower than native code [87]. Given that much of the literature on progressive meshes reports results with native code, and despite possibilities to speed up decompression using the GPU techniques, the requirement to implement the code with Javascript is not to be underestimated.

Multiple scene objects A typical 3D scene for web-based rendering may feature several objects of varying topological complexity and size. It may also feature modification or animation of these objects. Classic Progressive Meshes algorithms work well with regularly-sampled, closed surfaces, and as a result may not function correctly or efficiently in many use-cases for the 3D web - or at the very least, some form of pre-processing step is required to split and classify the meshes before compression.

Various data sources for a single object Basic PM does not take into account vertex properties other than position. While vertex position is naturally the most important component of any mesh (as it describes the shape), a mesh object may store several other components which are important for display, such as normal vectors, texture coordinates and colour values.

In recent years, however, there have been several research efforts to address these four points and create a viable method for mesh compression more suitable to web-based contexts. Tian and AlRegib [88] present a method to compress texture resolution in parallel with mesh vertex resolution. The authors note that geometric inaccuracies in the compressed mesh may be either enhanced or dissimulated by errors in the compressed texture; however, refining first the mesh and then the texture is not efficient, as either a full resolution mesh with a coarse texture or a full resolution texture with a coarse mesh will not generally provide good visualisation for the textured model. Thus, they propose a progressive solution which attempt to refine both the mesh and the texture in the same bit-stream, by proposing a bit-allocation framework. More research on bit-allocation for mesh

compression has been carried out by King and Rossignac [89], who use a shape-complexity measure to optimise distortion for a given bit rate; and by Payan and Antonini [90] who use a wavelet based method; although neither of these techniques are adapted for progressive transmission [91].

While several methods for encoding colour information exist (for example, Ahn et al. [92], who encode indices of each vertex in a colour mapping table; or Yoon et al. [93] who introduce a predictive method based on geometry information), only in recent years have efforts been made for progressive meshes. Cirio et al [94] propose a technique that allows any property or set of properties (such as vertex colours or normals) to drive a compression algorithm based on kd-trees. Lee et al. [91] propose a progressive algorithm which is based on the valence-driven progressive connectivity encoding from Alliez and Desbrun [95], and this technique is further optimised for the web by Lavou et al. [85]. A mesh is encoded by progressive steps of decimation and cleansing, and colour components are quantised adaptively according to level of details. Lavou et al [85] also present several implementational details specific to decompression using Javascript, such as the use of Array Buffers and the minimisation of garbage collection, as it was found that the latter process used blocked performance in unacceptable ways. Gobbetti et al. [96] convert input meshes to equally sized quad patches, each one storing vertex, colour and normal information, and which are then stored in an image format. This allows to use the atlas images for multi-resolution, and fast rendering using simple mip-map operations. Limper et al. [97] present a progressive encoding scheme for general triangle soups, using a hierarchy of quantisation to reorder the original primitive data into nested levels of detail.

Mesh compression faces further problems when the chosen rendering framework relies on declarative markup, such as X3D/X3DOM or XML3D, as one of advantages of those frameworks (the ability to describe a scene with clear, human-readable code) causes difficulties when loading scenes with very large meshes - parsing text files of hundreds of megabytes is effectively impossible for web browsers (although applying HTTP's GZIP compression can greatly reduce file sizes). Behr et al. [98] counter this by taking advantage of Javascript Typed Arrays to introduce a *BinaryGeometry* component to X3DOM, allowing the scene to be described in a declarative manner, but the content geometry to be passed to the client as raw binary data. Once the data has been downloaded, it can be passed immediately to the GPU memory, which effectively eliminates the relatively slow Javascript parsing of the data. Transmission speed of binary data can be reduced by compressing it, for example using the OpenCTM format [99], which is designed specifically to compress mesh data. The downside of using compression for web-based methods is, as always, the associated decompression cost in the Javascript layer.

In 2011, Google launched the *Google Body* project [100], an interactive browser-based visualisation of the human body. The development of this work also resulted in the creation of *WebGL-Loader* [101], a minimalistic Javascript library for compact 3D mesh transmission. It takes full advantage of in-built browser features, such as support for GZIP and the UTF-8 file

format, to attempt to enable fast decompression of mesh data in Javascript, using predictive techniques. Limper et al. [81] conducted a case-study to compare WebGL loader with OpenCTM [99], X3DOM's Binary Geometry [98] and standard X3D. Their results demonstrated the balance between file-transmission size and decompression rate; at low bandwidths, techniques which minimised file size (such as CTM compression) provided better results, while at high bandwidths, the speed of transfer of Binary Geometry to the GPU ensured best performance. While this initial result is not surprising, in the mobile context the result was different: the decreased power of the hardware meant that any technique which relied on heavy decompression suffered in comparison, even at high bandwidths.

4.2. Selective Transmission of Geometry

In 1996, Schmalstieg and Gervautz [102] proposed an early approach to optimising the transmission of geometry for distributed environments, arguing that the network transmission rate is the bottleneck in such a system (perhaps as true in the current day as it was in 1996). The authors proposed a system of demand-driven geometry transmission as a method for efficient transmission of geometric models, which was later expanded by Hesina [103]. A server stores data for all the objects (and their positions) for a given scene. Client viewer applications can make server requests only for the objects that are in the client *area of interest*. Thus, if geometry can be delivered from the server to the client "just in time", there is no need to transfer the entire scene geometry to the client.

While not directly related to web-based 3D graphics, there has been considerable research effort made into selective transmission of the 3D data for online virtual worlds and Massive Multiplayer Online (MMO) games, such as World of Warcraft [104] and Eve Online [105]. Such systems feature dedicated client-server systems which also employ Peer-to-Peer technology for the transfer of 3D assets. For further information, we direct the interested reader to a recent comprehensive survey paper by Yahyavi [106].

5. Applications

Despite the clear growth of 3D graphics applications across multiple platforms in the last two decades, the initial approaches to bring this growth to the internet have either stalled or did not gain traction [9]. In an attempt to research the reasons behind this, Jankowski [107] surveyed the different tasks and actions carried out by users when viewing web pages, and also when interacting with 3D content. The result is a taxonomy of 3D web use, which demonstrates that there are very few actions which are shared between what might be considered 'Web Tasks' (e.g. 'Click on Hyperlink', 'Find on Page') and '3D tasks' ('Spatial Navigation', 'Object Selection'). This lack of shared tasks leads to the conclusion that switching between textual (hypertext) and spatial (3D graphics) spaces can be potentially disorientating to the user, and any interface should take this separation into account. This research eventually resulted in [108], which presents a hybrid 2D/3D interface, designed to avoid confusing the user.

Nevertheless, Mouton et al [77] argue that web applications have major benefits with respect to desktop applications, for two main reasons:

- Web browsers are available for all major platforms, including mobile devices. This means that cross-platform compatibility is almost guaranteed, and there is no need to spend resources in developing for different platforms.
- Application deployment is much more straightforward, as a web application typically does not require the user to install or update any software or libraries (other than the web browser).

These advantages, combined with a greater understanding of how users interact with web-based 3D content [107], have spurred application development in the field. In this section we present an overview of the different application fields for 3D web graphics, which we have grouped into different areas: data visualisation and medical applications, digital content creation, video games, e-learning, and engineering, architecture and cultural heritage.

The overview does attempt to be exhaustive, but rather to provide the reader with an understanding of the breadth and depth of applications that have been, and are currently being, developed using 3D web technology.

5.1. Data Visualisation and Medical Applications

3D graphics have long been used as a technique for data visualisation [109], and the introduction of a 3D context to the web is now facilitating access to the field. A remote rendering example for visualizing scientific information already mentioned is ParaViewWeb [74]. Yet as Marion and Jomier point out [110], a downside of many of these frameworks (from a web point-of-view) is that many require a customised client setup, either with plugins or external applications. Thus, [110] propose a system which uses Three.JS and WebSockets to counter these issues. The WebSockets specification [111] introduces a full-duplex socket connection created in Javascript, designed to allow real-time synchronisation between multiple browsers. Marion et al extend their work in [112] and compare a WebGL/WebSocket implementation of a molecular visualisation with an identical ParaViewWeb scene; their results suggesting better performance of the WebGL version. Other approaches to visualisation of molecular structures are presented by Zollo et al. [113], who use X3DOM to enable users to interact in real-time with molecular models built in X3D, and molecular visualisation in WebGL developed by Callieri et al [114] (see Figure 6).

Limberger et al [115] use WebGL to visualise source code repositories using software maps (see Figure 7), which link 3D tree-maps, software system structure and performance indicators, used in software engineering processes. Software map visualisations typically feature large number of vertices, each of which requires linking to several attributes in order to precisely control the visualisation. The authors thus propose a customised structure for vertex arrays that allows rapid drawing of

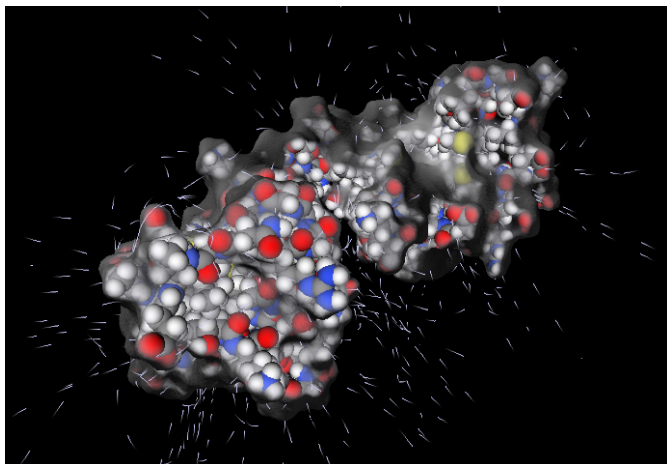


Figure 6: Superposition of molecular structure and underlying atomic structure in WebGL (reproduced with permission from [114])

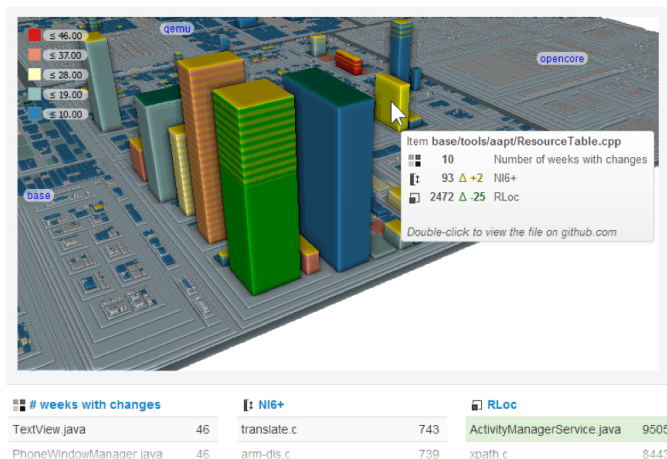


Figure 7: WebGL used to visualise software maps from Web-based source code repositories (reproduced with permission from [115])

geometry and avoids issues arising from WebGL current 16-bit limitation for its vertex index buffers.

Medical science and education have also benefitted by increasing accessibility to 3D software, through the 3D visualization of medical data. According to [116], visualization techniques in medicine fall in two main categories: surface extraction and volume rendering. The result of the surface representations, in both techniques, can be stored in a 3D format such as X3D and then rendered in a web real-time fashion in by taking advantages of 3D web recent developments. Warrick and Funnell [117] are the pioneers of using network technologies in medical education, by rendering the surface of the anatomical representation stored in a VRML file. In [118] animation capabilities in the form of rotational and slicing movements were added to the learning modules. In addition, the authors tried to address issues under low bandwidth through using lattices as an extension for X3D, representing high quality surface shape with minimal data. [119] propose an approach for modelling of developmental anatomy and pathology that provides users with a UI for a narrative showing different stages of anatomical de-

velopment. Moreover, the user is given some basic control such as pause, rewind, and fast forward.

A new mobile learning tool [120] provides users with an interactive 3D visualization of medical imaging to teach anatomy and manual therapy. The authors implement a new volume rendering method specialized for mobile devices that defines the color of each pixel through a ray casting method. Jacinto et al [121] bring the medical application field into the modern HTML5/WebGL paradigm with an application that allows real-time visualisation and segmentation of medical images, using a client-server approach. Data is processed using VTK [122] on the server side, with segmentation of medical images being converted into 3D surfaces and sent to the client to be rendered using Three.JS. Mani and Li [123] present a surgical training system built with X3D and WebGL, allowing real-time updates from trainees and experienced surgeons. Congote et al. [124] use WebGL for volume rendering of scientific data, using a ray-casting method to visualise both medical data and meteorological data.

5.2. Digital Content Creation

A clear application of web-based 3D is the creation, editing, and revision of 3D assets which are destined for other applications or productions. There have been several academic efforts at proposing and creating solutions for digital content creation [125] [126], annotation [127] [128], and scene and programme creation [37] [129]; yet with the continual success of digital animation and video game production, it is perhaps no surprise that the commercial sector is at the forefront for using the 3D web in a production environment. The digital production industry has now become global involving many companies working on different aspects of a production from any place around the globe. Modern digital production deals with 3D assets on a daily basis, and there are now 3D applications on the web which are being used to ease the production pipeline of these digital media.

One of the most important tasks for such production is creating tools that are able to properly access 3D assets, review them and properly annotate any change. Such tools may be applied to varying facets of the authoring process, such as modeling, shading or even animation and lighting/rendering. The success of such tools is encapsulated by Tweak Software's RV [130]. RV is a desktop offline tool that allows users to review and annotate still frames, image sequences and 3D contents using OpenGL technology. As an industry tool it includes a feature that allows users to share the RV workspace through the internet in order to collaborate on the same data and watch the results of any change in real time even if in different places in the world.

With the increased maturity of web-based technology for creating advanced user interfaces, there are several web applications attempting to replicate RV's success in a web-based environment. Sketchfab [131] is a web tool and community, developed with WebGL, for 3D modeling and texturing, and is designed to allow 3D artists to share their creations seamlessly via the web. Commonly, 3D models are showcased through pre-rendered turnaround videos or still images but with tools like



Figure 8: Screenshot of the Sketchfab 3D web application [131]

the Sketchfab it is possible to share the results in a 3D web context, so that the viewer can interact with the scene, rotating the camera and even interacting with the content. Additional features such as the ability to write comments and the possibility to view the model in three different combined ways (wireframe, shaded and textured) make it suitable for use as a review tool for media production companies. It could be enhanced with a sketching tool, allowing reviewers to rapidly add corrections or notes, and to relate a comment to a particular part of the geometry.

Clara.io [132] is an online tool (currently in Beta) for the creation and sharing of 3D content. It features a suite of modelling tools for mesh creation, and also allows key-frame animation and annotation of meshes for sharing and collaborative creation.

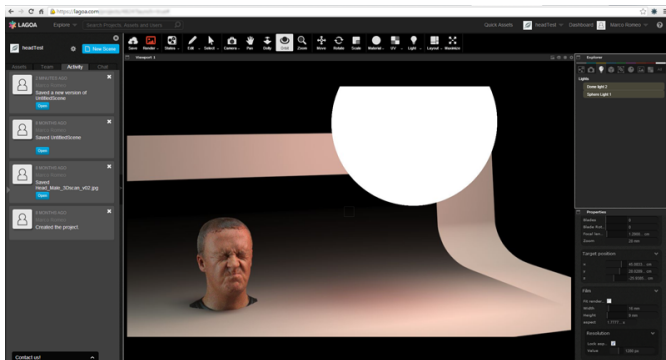


Figure 9: Screenshot of the Lagoa 3D web application [133]

Lagoa [133] is a "web based platform for photoreal 3D visualization and rendering" developed using WebGL. It offers the possibility to work on 3D scenes through the web browser in a collaborative way by creating workgroups and simultaneously interacting with the contents and commenting them with other members. It enables users to upload assets directly from their computer or access them from the cloud asset manager that Lagoa offers. This asset manager is also based on workgroups and allows users to see only the scenes they are permitted to work on, partially mimicking the industry standard tool for cloud asset management, Shotgun [134]. All the accessible assets can be placed, moved around and previewed in 3D space.

The tool can also manage lights in real time for rapid development of the scenes and includes a state-of-the-art photorealistic rendering engine. The latter uses all the information provided by the users in the scene and makes all the computation on servers provided by Lagoa itself. In this way, the user is not bounded by computational constraints of his/her computer and can render rapidly even on lower end hardware. The rendering is performed using progressive refinement global illumination algorithms that shows the user a gradually enhanced version of the entire scene (opposite to standard renderers which show image tiles only when completed). This permits creative users (e.g. a lighting artist, or the director of photography) to start commenting on the work even before rendering is finished.

3DTin [135] is a tool for rapid creation of simple 3D geometries, which was recently purchased by Lagoa. Unfortunately, the resulting mesh is very basic and too simple to be used in general production, but the tool demonstrates that there is an effort to work in this direction.

Autodesk, the leader in 3D authoring software development, has also shown interest in developing 3D applications for the web and is working on different solutions. 123Design [136] is a modelling tool similar to 3DTin but, more than just creating simple geometries out of primitives, it also allows users to import their own models and modify them. It requires a plugin to be installed and does not use WebGL. Despite Autodesk backing, 123Design does not yet achieve the quality required for general industry use. However, Autodesk has announced a platform for hosting servers running their applications, allowing users to connect and interact with the applications through an internet connection. This solution would unleash all the power of current high-end Autodesk products, together with the possibilities of working on the cloud. Although this solution would not involve direct rendering of 3D graphics on the web, it evidence of the interest in building 3D applications for the industry, running through the web and on the cloud.

5.3. Games

It is beyond doubt that video games have contributed to the very cutting edge of computer graphics since their invention, with several annual conferences (such as the Games Developers Conference) bringing together companies, researchers and individuals in the field. Videogames are also indelibly associated with the web thanks to the ubiquity of Adobe Flash, which has provided a platform for many 2D-based games. However, the contribution of the games field to the 3D web is less obvious, as many popular online games, such as World of Warcraft [104] or Eve Online [105] use custom, platform specific clients. The rise of the free-to-play model has seen some efforts to produce 3D games within the browser [55] [137]; there have been some open-source efforts to port older 3D games to WebGL [138]; and Epic Games have created a demo of their Unreal Engine working with WebGL [54]. Perhaps the greatest breakthrough in the enabling of 3D gaming via the web so far has been by Unity [52] [53], discussed in detail above. With the rise of web-enabled 3D digital content creation tools (mentioned above), it is perhaps only a matter of time before more

commercial attention is paid to 3D games running natively in the browser.

5.4. E-learning

It is acknowledged that students only fully absorb the learning material once they have applied it in practice; yet learning through Virtual Reality (VR) provides a simulated learning experience [139] which attempts to enhance learning by joining theory and practice. The application of 3D virtual platforms on the web has become an increasingly popular research topic since Wickens analyzed the advantages of learning in VR environment [140]. He stated that VR can be defined based on five main concepts: 3D perspective, real-time rendering, closed loop interaction, inside-out perspective, and enhanced sensory feedback. At the same time, [141] assessed the merits and faults of VR's conceptual and technical future. Due to the special software and hardware requirements of VR systems, the development of these types of systems have a high cost associated with them [142], while in the past few years, we have witnessed the creation and proliferation of 3D virtual frameworks through the web that can be used by lower end computers.

Initially, the growth of 3D multi-user virtual worlds (MUVE) faced the challenge of motivating learners to utilize this kind of 3D capability to learn courses with greater precision. In light of recent developments of web capabilities (as surveyed in this paper), this particular type of e-learning has received a great deal of attention as a compelling means of resolving traditional teaching issues. One of the first attempts [143] in this area was launched by Linden Labs with Second Life [144], originally a 3D virtual world presented as an educational framework for simulating social interactions. The programming language of Second Life was Linden Scripting Language. However, this application suffers from two main drawbacks: poor performance due to network latency and the prohibition of sharing the content outside of the virtual environment. Later, OpenSim [145] expanded on this concept, improving the quality of the learning environment and making it free and open rather than proprietary. In addition, it allowed user-created content to be exported in a portable format called OAR and shared within the community. The advantages and disadvantages of these two systems are specifically discussed in [146] [143]. Second Life has itself become a platform for research, with several authors using it as a base for e-learning research [147] [148]; and now it has a web-based client, bringing it inline with the current trends for browser-based 3D.

[149] analysed the application of Second Life as a 3D platform on the web that offers potential as a tourism educational tool by providing interactive experiences for students. In this study, Self-Determination Theory (SDT) [150] [151] is applied as a metric to recognise significant factors which influence student learning motivations in 3D world on the web. The results revealed that a positive emotional state had a positive and significant impact on students' intrinsic motivation when learning in a 3D virtual world.

Di Cerbo et al [152] make specific efforts to integrate a 3D web interface into an avatar-based e-learning platform. They conclude that the addition of high-performance, plugin-free 3D

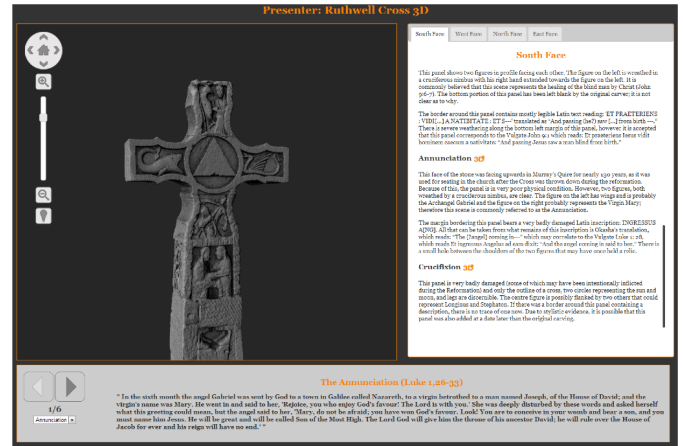


Figure 10: Screenshot of the Ruthwell Cross and associated narrative (reproduced with permission from [153])

graphics into their platform allowed a complete and meaningful interaction with the e-learning services, benefitting the user experience.

5.5. Geography, Architecture and Virtual Heritage

Geography and architecture are natural application fields of 3D technology, and this is reflected in related research in the web context. Over et al. [154] provide a summary of efforts made to use OpenStreetMap (OSM) data as a basis for creating 3D city models, which are then viewed in a web-based context. Many of the developments in this field [155][156][157] create and use 3D data stored in the CityGML format [158] an XML-based open format and information model for the representation of urban objects. 3DNSITE [159] streams large hybrid data (georeferenced point clouds and photographs) to client hardware, with the goal of assisting crisis managers and first responders to familiarise themselves with an environment either during an emergency or for training purposes. Lamberti [160] use web-based 3D to visualise real-world LED-based street lighting. Geographical Information Systems (GIS) (such as Google Earth) are increasingly being augmented with 3D data [161], and there have been several efforts to aid in the streaming and visualisation of such data [162] [163].

The rise of Virtual Heritage (VH) represents the equivalent rise in the ability of technology to digitise real 3D objects (in this case, of cultural value), and display them in a virtual environment. Thus VH has been an important application of internet based 3D graphics and has been since the days of VRML [164] [165] [166]. Efforts at using modern, browser based declarative 3D techniques in VH are presented in [167]. Manferdini and Remondino [128] outline a method to semantically segment 3D models in order to facilitate annotation, and demonstrate the new possibilities with architectural and archaeological heritage structures. Callieri et al [153] demonstrate how 3D visualisation for VH can be combined in a narrative way with standard HTML to create narrative experiences and provide greater understanding of the artwork or cultural object in question (see Figure 10). Other modern efforts for web-based

virtual heritage have involved adaptation of systems specifically for mobile contexts [168] [169].

Finally, Autodesk has recently worked in conjunction with the Smithsonian Foundation to deliver the Smithsonian X 3D [170], a WebGL virtual heritage tool for the online visualization of 3D scans. It has many features common to 3D tools, such as the choice between wireframe or shaded view, textures, and advanced real time materials, but it also provides the possibility to alter the lighting of the scene in a very direct and easy way by defining light sources around the surface of a virtual dome.

6. Discussion

In 2010, Ortiz [26] stated that there were several major hurdles which needed to be overcome before "the 3D web can truly flower":

- The requirement for web browsers to use **plugins** in order to view and interact with 3D content; not only because plugin-installation is a barrier to installation, but because plugins are prone to cause browser crashes and other problems.
- The reliance on plugins also damages **cross-platform compatibility** and contributes to the inability of 3D to work on all browsers and operating systems.
- The lack of **standardisation** may lead to the web becoming a tangled mess of incompatible formats and technologies, forcing the developers to create multiple versions of the same technology.
- The long **authoring times** for 3D content on the web are a barrier to entry, both for web developers and end users.
- Proponents are not designing online 3D technology **for the average user** (which, it is claimed, led to the perceived failure of VRML)
- The rise in use of **low-power devices** means that running 3D content on the client side is increasingly difficult

Referring to the work surveyed in this paper, we can now attempt to conclude whether any progress has been made on any of these points, and whether any new issues have arisen.

Firstly, it is clear that the need to use plugins for web-based 3D content is now diminishing. The release of WebGL, now supported by all major browsers, means that developers can access graphics hardware acceleration via the browser, without requiring the user to install a third party plugin. This flexibility is reflected in the number of libraries which abstract and add functionality to the core WebGL specification, chief of which is Three.JS. The removal of plugins means that cross-platform compatibility is also greatly enhanced. The only current remaining stumbling block to the general acceptance of WebGL is the continuing reluctance of Apple to enable official support on its Mobile Safari browser (the default browser on all the companies internet capable mobile devices, such as the iPhone and

Keyword	GitHub stars	ACM Digital Library	IEEEExplore
X3DOM (X3D)	143 (n/a)	71 (547)	9 (127)
Three.JS	13616	14	1

Table 2: Popularity of X3D and Three.JS in different online platforms. Github is an online source code repository used for collaboratively creating software projects, 'stars' are can be given by users to a particular project (maximum one star per project per registered user). The ACM and IEEE digital libraries are databases of academic papers published in journals and affiliated conferences for these two institutions.

iPad), despite clear evidence that the functionality is present (in both hardware and software) [171].

Efforts towards a standard for web-based 3D graphics have achieved mixed results. Table 2 summarises some very basic statistics on the relative popularity of X3D/X3DOM (proposed standard) and Three.JS (non-standard library built on the WebGL). Three comparison measures are used: (a) the number of 'stars' (recommendations by users) the projects have received on the popular GitHub online repository; and (b) and (c) the number of returned hits for published papers when searching on two major academic portals, the ACM Digital Library and the IEEEExplore Digital Library respectively. While not rigorous in terms of assessment, they show a clear difference in popularity between the academic community on one hand and the wider development community on the other. The X3D standard, and its younger, browser-integrated cousin, X3DOM, are featured in dozens of academic articles, whereas Three.JS is barely mentioned in any. The Github stars show that Three.JS is clearly much more popular among developers than X3DOM. This presents a quandary for the standards community, as while there has clearly been considerable research effort put into creating a standard for declarative 3D, most developer community attention has been paid to an open source library which has grown in a less formal way (not forgetting that both Three.JS and X3DOM depend heavily on WebGL, itself a standard of the Khronos group). The appearance of XML3D muddies the water yet further for 3D web standards - while it shares many of the overall goals of the declarative 3D paradigm favoured by X3D/X3DOM, its existence somewhat undermines the latter's goal to become a web standard. Nevertheless, the community is making efforts to tread carefully through this minefield, with Jankowski et al [5] proposing a common PolyFill layer for all declarative 3D approaches, and the development of interoperability tools such as those by Berthelot et al. [172]. Through this open dialogue and collaboration, then, there is hope that Ortiz' [26] "tangled mess of standards" can yet be avoided.

Our survey on Digital Content Creation for the 3D web (see Section 5.2) has demonstrated that there is now real effort into the task of democratising the creation of 3D content. Both from academic field (see Figure 11) and the commercial fields, there are now several tools and interfaces that are bringing the power of 3D content and scene creation within a web context, partially removing the need for installation of large and/or expensive desktop software (though we have yet to see the release of a fully-featured 3D modelling tool).



Figure 11: WebGLStudio is an example of how web 3D tools are being used to democratise the process of 3D scene creation (reproduced with permission from [37])

It is clear that the rise of low-power devices, such as mobiles phones and tablets, is not presenting a large barrier to adoption of the 3D web. Modern mobile devices can execute 3D content in the browser, and Section 3 references several efforts made to offset the downsides of a mobile context, particularly regarding processing power and bandwidth consumption.

Our general conclusion from this survey is that the world of web-based 3D graphics is vibrant and exciting, both in the academic and wider developer communities. Each passing year brings further developments which are shared online, in the general academic press and conferences, and in specific gatherings such as the International Conference on 3D Web Technology, which in 2013 was staged for the 18th time.

Acknowledgements

The authors would like to acknowledge the support of the IMPART Project, funded by the ICT - 7th Framework Program from the European Commission (<http://impart.upf.edu>).

References

- [1] Schatz BR, Hardin JB. NCSA Mosaic and the World Wide Web: Global Hypermedia Protocols for the Internet. Science (New York, NY) 1994;265(5174):895–901.
- [2] Curtis H. Flash Web Design: The Art of Motion Graphics. New Riders Publishing; 2000. ISBN 0735708967.
- [3] Nielsen J. Designing Web Usability. New Riders; 1999. ISBN 156205810X.
- [4] W3C . HTML5 Specification. 2009. URL: <http://www.w3.org/TR/html5/>.
- [5] Jankowski J, Ressler S, Jung Y, Behr J, Slusallek P. Declarative Integration of Interactive 3D Graphics into the World-Wide Web: Principles, Current Approaches, and Research Agenda. In: Proceedings 18th International Conference on 3D Web Technology (Web3D'13). 2013, p. 39–45.
- [6] W3C . Scalable Vector Graphics. 2001. URL: <http://www.w3.org/Graphics/SVG/>.
- [7] Geroimenko V, Chen C. Visualizing Information Using SVG and X3D. Springer; 2004. ISBN 1852337907.
- [8] Web3D . X3D. 2013. URL: http://www.web3d.org/x3d/specifications/x3d_specification.html.
- [9] Behr J, Eschler P, Jung Y, Zöllner M. X3DOM: a DOM-based HTML5/X3D integration model. Proceedings of the 14th International Conference on 3D Web Technology 2009;:127–36.
- [10] W3C . Namespaces in XML. 1998. URL: <http://www.w3.org/TR/REC-xml-names/>.
- [11] Behr J, Jung Y, Keil J, Drevensek T. A scalable architecture for the HTML5/X3D integration model X3DOM. In: Proceedings of the 15th International Conference on 3D Web Technology. ISBN 9781450302098; 2010, p. 185–94.
- [12] Microsoft . WebGL. 2013. URL: <http://msdn.microsoft.com/en-us/library/ie/bg182648%2528v%2529.aspx>.
- [13] Schwenk K, Jung Y, Behr J, Fellner DW. A modern declarative surface shader for X3D. In: Proceedings of the 15th International Conference on Web 3D Technology - Web3D '10. 2010, p. 7.
- [14] Schwenk K, Jung Y, Voß G, Sturm T, Behr J. CommonSurfaceShader revisited: improvements and experiences. In: Proceedings of the 17th International Conference on 3D Web Technology. ISBN 1450314325; 2012, p. 93–6.
- [15] Behr J, Jung Y, Drevensek T, Aderhold A. Dynamic and interactive aspects of X3DOM. In: Proceedings of the 16th International Conference on 3D Web Technology - Web3D '11. 2011, p. 81.
- [16] Sons K, Klein F, Rubinstein D, Byelozorov S, Slusallek P. XML3D. In: Proceedings of the 15th International Conference on Web 3D Technology - Web3D '10. 2010, p. 175.
- [17] Klein F, Sons K, Rubinstein D, Slusallek P. XML3D and Xflow: Combining Declarative 3D for the Web with Generic Data Flows. IEEE Computer Graphics and Applications 2013;33(5):38–47. doi:10.1109/MCG.2013.67.
- [18] W3C . CSS 3D Transforms. 2012. URL: <http://www.w3.org/TR/css3-transforms/>.
- [19] W3C . CSS Transitions. 2009. URL: <http://www.w3.org/TR/css3-transitions/>.
- [20] Arnaud R, Barnes M. COLLADA: sailing the gulf of 3D digital content creation. CRC Press; 2006. ISBN 1568812876. URL: <http://www.lavoisier.fr/livre/notice.asp?ouvrage=1828050>.
- [21] Khronos . glTF. 2013.
- [22] Parisi T. WebGL: Up and Running. O'Reilly Media; 2012. ISBN 144932357X.
- [23] Evans A, Agenjo J, Abadia J, Balaguer M, Romeo M, Pacheco D, et al. Combining educational MMO games with real sporting events. 2011.
- [24] Tautenhahn L. SVG-VML-3D. 2002. URL: <http://www.lutano.net/svgvml3d/>.
- [25] Google . O3D. 2008. URL: <https://code.google.com/p/o3d/>.
- [26] Ortiz Jr. S. Is 3D Finally Ready for the Web? Computer 2010;43(1):14–6.
- [27] Sánchez JR, Oyarzun D, Díaz R. Study of 3D web technologies for industrial applications. In: Proceedings of the 17th International Conference on 3D Web Technology - Web3D '12. 2012, p. 184.
- [28] C3DL . Canvas 3D JS. 2008. URL: <http://www.c3dl.org/>.
- [29] Leung C, Salga A, Smith A. Canvas 3D JS library. In: Proceedings of the 2008 Conference on Future Play Research, Play, Share - Future Play '08. ISBN 9781605582184; 2008, p. 274.
- [30] Johansson T. Taking the canvas to another dimension. 2008. URL: <http://my.opera.com/timjoh/blog/2007/11/13/taking-the-canvas-to-another-dimension>.
- [31] Khronos . WebGL Specification. 2011. URL: <http://www.khronos.org/registry/webgl/specs/latest/1.0/>.
- [32] Khronos . WebGL Demo Repository. 2013. URL: http://www.khronos.org/webgl/wiki/Demo_Repository.
- [33] Cantor D, Jones B. WebGL Beginner's Guide. Packt Publishing; 2012. ISBN 184969172X.
- [34] Matsuda K, Lea R. WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL (OpenGL). Addison-Wesley Professional; 2013. ISBN 0321902920.
- [35] Di Benedetto M, Ponchio F, Ganovelli F, Scopigno R. SpiderGL. In: Proceedings of the 15th International Conference on Web 3D Technology - Web3D '10. ISBN 9781450302098; 2010, p. 165. doi:10.1145/1836049.1836075.
- [36] Di Benedetto M, Ganovelli F, Banterle F. Features and Design Choices in SpiderGL. In: Cozzi P, Riccio C, editors. OpenGL Insights. CRC

- Press. ISBN 978-1439893760; 2012, p. 583–604.
- [37] Agenjo J, Evans A, Blat J. WebGLStudio. In: Proceedings of the 18th International Conference on 3D Web Technology - Web3D '13. ISBN 9781450321334; 2013, p. 79.
 - [38] GLMatrix. glMatrix v2.2.0. 2013. URL: <http://glmatrix.net/>.
 - [39] Pinson C. OSGJS. 2013. URL: <http://osgjs.org/>.
 - [40] Osfield R, Burns D. Open scene graph. 2004. URL: <http://www.openscenegraph.org/>.
 - [41] Kay L. Scene.js. 2010. URL: <http://www.scenejs.org/>.
 - [42] Belmonte N. PhiloGL. 2013. URL: <http://www.senchalabs.org/philogl/>.
 - [43] Brunt P. GLGE.org. 2010. URL: <http://www.glge.org/>.
 - [44] Cabello R. Three.js. 2010. URL: <http://threejs.org/>.
 - [45] Dirksen J. Learning Three.js: The JavaScript 3D Library for WebGL. Packt Publishing; 2013. ISBN 1782166289.
 - [46] Degrees P. Car Visualizer. 2013. URL: <http://carvisualizer.plus360degrees.com/threejs/>.
 - [47] Adobe. Stage 3D. 2011. URL: <http://www.adobe.com/devnet/flashplayer/stage3d.html>.
 - [48] Boese ES. An Introduction to Programming with Java Applets. Jones & Bartlett Learning; 2009. ISBN 0763754609.
 - [49] Oracle. Java3D. 2008. URL: <https://java3d.java.net/>.
 - [50] JogAmp. JogAmp. 2013. URL: <http://jogamp.org/>.
 - [51] LWJGL. LWJGL - Lightweight Java Game Library. 2013. URL: <http://lwjgl.org/>.
 - [52] Unity. Unity3D. 2013. URL: <http://www.unity3d.com>.
 - [53] Unity. Web Player Statistics. 2013.
 - [54] Epic Games. Unreal Engine. 2013. URL: <http://www.unrealengine.com/html5/>.
 - [55] DICE. Battlefield Heroes. 2013. URL: <http://www.battlefieldheroes.com/>.
 - [56] OnLive. OnLive. 2013. URL: <http://www.onlive.com/>.
 - [57] Gaikai. Gaikai. 2013. URL: <http://www.gaikai.com>.
 - [58] Engadget. Sony buys Gaikai cloud gaming service for \$380 million. 2012. URL: <http://www.engadget.com/2012/07/02/sony-buys-gaikai/>.
 - [59] Humphreys G, Eldridge M, Buck I, Stoll G. WireGL: a scalable graphics system for clusters. Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH) 2001;:129–40.
 - [60] Glander T, Moreno A, Aristizabal M, Congote J, Posada J, Garcia-Alonso A, et al. ReWeb3D. In: Proceedings of the 18th International Conference on 3D Web Technology - Web3D '13. 2013, p. 147.
 - [61] Emscripten. Emscripten. 2014. URL: <http://emscripten.org>.
 - [62] Lamberti F, Zunino C, Sanna A, Fiume A, Maniezzo M. An accelerated remote graphics architecture for PDAS. In: Proceeding of the eighth international conference on 3D web technology - Web3D '03. 2003, p. 55.
 - [63] Noimark Y, Cohen-Or D. Streaming scenes to MPEG-4 video-enabled devices. Computer Graphics and Applications, ... 2003;.
 - [64] Tizon N, Moreno C, Cernea M, Preda M. MPEG-4-based adaptive remote rendering for video games. In: Proceedings of the 16th International Conference on 3D Web Technology - Web3D '11. 2011, p. 45.
 - [65] Richardson T, Stafford-Fraser Q, Wood KR, Hopper A. Virtual network computing. Internet Computing, IEEE 1998;2(1):33–8.
 - [66] Levoy M. Polygon-assisted JPEG and MPEG compression of synthetic images. In: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH '95. s; 1995, p. 21–8.
 - [67] Mann Y, CohenOr D. Selective pixel transmission for navigating in remote virtual environments. Computer Graphics Forum 1997;.
 - [68] Fechteler P, Eisert P. Depth map enhanced macroblock partitioning for H. 264 video coding of computer graphics content. In: Image Processing (ICIP), 2009 16th IEEE International Conference on. ISBN 1424456533; 2009, p. 3441–4.
 - [69] Diepstraten J, Gorke M, Ertl T. Remote line rendering for mobile devices. In: Computer Graphics International. 2004, p. 454–61.
 - [70] Quillet JC, Thomas G, Granier X, Guittion P, Marvie JE. Using expressive rendering for remote visualization of large city models. In: Proceedings of the eleventh international conference on 3D web technology - Web3D '06. 2006, p. 27.
 - [71] Yoon I, Neumann U. Web-Based Remote Rendering with IBRAC (Image-Based Rendering Acceleration and Compression). Computer Graphics Forum 2000;19(3):321–30.
 - [72] Jurgelionis A, Fechteler P, Eisert P, Bellotti F, David H, Laulajainen JP, et al. Platform for Distributed 3D Gaming. International Journal of Computer Games Technology 2009;2009:1–15.
 - [73] Squillacote A. The Paraview Guide. Kitware, Inc.; 2008. ISBN 1930934211.
 - [74] Jourdain S, Ayachit U, Geveci B. Paraviewweb, a web framework for 3d visualization and data processing. In: IADIS International Conference on Web Virtual Reality and Three-Dimensional Worlds; vol. 7. 2010, p. 1.
 - [75] Grimstead IJ, Avis NJ, Walker DW. RAVE: the resourceaware visualization environment. Concurrency and Computation: Practice and Experience 2009;21(4):415–48.
 - [76] Jourdain S, Forest J, Mouton C, Nouailhas B, Moniot G, Kolb F, et al. ShareX3D, a scientific collaborative 3D viewer over HTTP. In: Proceedings of the 13th international symposium on 3D web technology. ISBN 1605582131; 2008, p. 35–41.
 - [77] Mouton C, Sons K, Grimstead I. Collaborative visualization: current systems and future trends. In: Proceedings of the 16th International Conference on 3D Web Technology. ACM. ISBN 1450307744; 2011, p. 101–10.
 - [78] Peng J, Kim C, Kuo CJ. Technologies for 3D mesh compression: A survey. Journal of Visual Communication and Image ... 2005;.
 - [79] Alliez P, Gotsman C. Recent advances in compression of 3D meshes. In: Advances in Multiresolution for Geometric Modelling. Springer Berlin Heidelberg. ISBN 978-3-540-21462-5; 2005, p. 3–26.
 - [80] Shamir A. A survey on mesh segmentation techniques. Computer graphics forum 2008;27(6):1539–56.
 - [81] Limper M, Wagner S, Stein C, Jung Y, Stork A. Fast delivery of 3D web content: a case study. In: Proceedings of the 18th International Conference on 3D Web Technology. ISBN 145032133X; 2013, p. 11–7.
 - [82] Hoppe H. Progressive meshes. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques SIGGRAPH 1996;:99–108.
 - [83] Hoppe H. Efficient implementation of progressive meshes. Computers & Graphics 1998;22(1):27–36.
 - [84] Maglo A, Courbet C, Alliez P, Hudelot C. Progressive compression of manifold polygon meshes. Computers & Graphics 2012;36(5):349–59.
 - [85] Lavoué G, Chevalier L, Dupont F. Streaming Compressed 3D Data on the Web using JavaScript and WebGL. In: ACM International Conference on 3D Web Technology (Web3D), San Sebastian, Spain. 2013, p. 19–27.
 - [86] Valette S, Chaine R, Prost R. Progressive lossless mesh compression via incremental parametric refinement. Computer Graphics Forum 2009;28(5):1301–10.
 - [87] Charland A, Leroux B. Mobile application development: web vs. native. Communications of the ACM 2011;54(5):49–53.
 - [88] Tian D, AlRegib G. BateX3: Bit allocation for progressive transmission of textured 3-d models. Circuits and Systems for Video Technology, IEEE Transactions on 2008;18(1):23–35.
 - [89] King D, Rossignac J. Optimal bit allocation in compressed 3D models. Computational Geometry 1999;14(1):91–118.
 - [90] Payan F, Antonini M. An efficient bit allocation for compressing normal meshes with an error-driven quantization. Computer Aided Geometric Design 2005;22(5):466–86.
 - [91] Lee H, Lavoué G, Dupont F. Rate-distortion optimization for progressive compression of 3D mesh with color attributes. The Visual Computer 2012;28(2):137–53.
 - [92] Ahn JH, Kim CS, Ho YS. Predictive compression of geometry, color and normal data of 3-D mesh models. Circuits and Systems for Video Technology, IEEE Transactions on 2006;16(2):291–9.
 - [93] Yoon YS, Kim SY, Ho YS. Color data coding for three-dimensional mesh models considering connectivity and geometry information. In: Multimedia and Expo, 2006 IEEE International Conference on. IEEE; 2006, p. 253–6.
 - [94] Cirio G, Lavoué G, Dupont F. A Framework for Data-driven Progressive Mesh Compression. In: GRAPP. 2010, p. 5–12.
 - [95] Alliez P, Desbrun M. Progressive compression for lossless transmission of triangle meshes. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. ACM. ISBN 158113374X; 2001, p. 195–202.
 - [96] Gobbetti E, Marton F. Adaptive quad patches: an adaptive regular struc-

- ture for web distribution and adaptive rendering of 3D models. In: Proceedings of the 17th International Conference on 3D Web Technology. 2012, p. 9–16.
- [97] Limper M, Jung Y, Behr J, Alexa M. The POP Buffer: Rapid Progressive Clustering by Geometry Quantization. *Computer Graphics Forum* 2013;32(7):197–206.
- [98] Behr J, Jung Y, Franke T, Sturm T. 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. 2012, p. 17–26.
- [99] Geelnard M. OpenCTM, the Open Compressed Triangle Mesh file format. 2010. URL: <http://openctm.sourceforge.net/>.
- [100] Blume A, Chun W, Kogan D, Kokkevis V, Weber N, Petterson RW, et al. Google body: 3d human anatomy in the browser. In: ACM SIGGRAPH 2011 Talks. ACM. ISBN 1450309747; 2011, p. 19.
- [101] Chun W. WebGL models: End-to-End. In: Cozzi P, Riccio C, editors. *OpenGL Insights*. CRC Press. ISBN 1439893764; 2012, p. 431–52.
- [102] Schmalstieg D, Gervautz M. DemandDriven Geometry Transmission for Distributed Virtual Environments. *Computer Graphics Forum* 1996;15(3):421–32.
- [103] Hesina G, Schmalstieg D. A network architecture for remote rendering. In: Proceedings. 2nd International Workshop on Distributed Interactive Simulation and Real-Time Applications (Cat. No.98EX191). 1998, p. 88–91.
- [104] Blizzard . *World of Warcraft*. 2013. URL: <http://www.worldofwarcraft.com/>.
- [105] CCP. Eve Online. 2013. URL: <http://www.eveonline.com/>.
- [106] Yahyavi A, Kemme B. Peer-to-peer architectures for massively multiplayer online games: A survey. *ACM Computing Surveys* 2013;1:8022980.
- [107] Jankowski J, Hachet M. A Survey of Interaction Techniques for Interactive 3D Environments. In: *Eurographics 2013-State of the Art Reports*. ISBN 1017-4656; 2012, p. 65–93.
- [108] Jankowski J, Decker S. A dual-mode user interface for accessing 3D content on the world wide web. In: Proceedings of the 21st international conference on World Wide Web - WWW '12. 2012, p. 1047.
- [109] Weiskopf D. GPU-Based Interactive Visualization Techniques (Mathematics and Visualization). Springer; 2006. ISBN 3540332626.
- [110] Marion C, Jomier J. Real-time collaborative scientific WebGL visualization with WebSocket. Proceedings of the 17th International Conference on 3D Web Technology 2012;:47–50.
- [111] W3C . *WebSocket Specification*. 2009. URL: <http://www.w3.org/TR/websockets/>.
- [112] Marion C, Pouderoux J, Jomier J, Jourdain S, Hanwell M, Ayachit U. A Hybrid Visualization System for Molecular Models. In: Proceedings of the 18th International Conference on 3D Web Technology. Web3D '13; ACM. ISBN 978-1-4503-2133-4; 2013, p. 117–20. doi:10.1145/2466533.2466558.
- [113] Zollo F, Caprini L, Gervasi O, Costantini A. X3DMMS. In: Proceedings of the 16th International Conference on 3D Web Technology - Web3D '11. ISBN 9781450307741; 2011, p. 129.
- [114] Callieri M, Andrei RM, Di Benedetto M, Zoppè M, Scopigno R. Visualization methods for molecular studies on the web platform. In: Proceedings of the 15th International Conference on Web 3D Technology - Web3D '10. 2010, p. 117.
- [115] Limberger D, Tr J. Interactive Software Maps for Web-Based Source Code Analysis. In: Proceedings 18th International Conference on 3D Web Technology. ISBN 9781450321334; 2013, p. 91–8.
- [116] John NW. The impact of Web3D technologies on medical education and training. *Computers & Education* 2007;49(1):19–31.
- [117] Warrick PA, Funnell WRJ. A VRML-based anatomical visualization tool for medical education. *Information Technology in Biomedicine, IEEE Transactions on* 1998;2(2):55–61.
- [118] Wakita A, Hayashi T, Kanai T, Chiyokura H. Using lattice for web-based medical applications. In: Proceedings of the sixth international conference on 3D Web technology. ISBN 1581133391; 2001, p. 29–34.
- [119] Brenton H, Hernandez J, Bello F, Strutton P, Firth T, Darzi A. Web based delivery of 3D developmental anatomy. In: Proceedings of the LET-WEB3D 2004 workshop on Web3D technologies. 2004, p. 53–7.
- [120] Noguera JM, Jiménez JJ, Osuna-Pérez MC. Development and evaluation of a 3D mobile application for learning manual therapy in the physiotherapy laboratory. *Computers & Education* 2013;69:96–108.
- [121] Jacinto H, Kéchichian R, Desvignes M, Prost R, Valette S. A Web Interface for 3D Visualization and Interactive Segmentation of Medical Images. In: Proceedings of the 17th International Conference on 3D Web Technology. Web3D '12. ISBN 978-1-4503-1432-9; 2012, p. 51–8. doi:10.1145/2338714.2338722.
- [122] Schroeder W, Martin K, Lorensen B. *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, 4th Edition. Kitware; 2006. ISBN 193093419X.
- [123] Mani G, Li W. 3D web based surgical training through comparative analysis. In: Proceedings of the 18th International Conference on 3D Web Technology - Web3D '13. 2013, p. 83.
- [124] Congote J, Segura A, Kabongo L, Moreno A, Posada J, Ruiz O. Interactive visualization of volumetric data with WebGL in real-time. In: Proceedings of the 16th International Conference on 3D Web Technology - Web3D '11. 2011, p. 137.
- [125] Doboš J, Steed A. 3D Revision Control Framework. In: Proceedings of the 17th International Conference on 3D Web Technology. Web3D '12; New York, NY, USA: ACM. ISBN 978-1-4503-1432-9; 2012, p. 121–9.
- [126] Ulbrich C, Lehmann C. A DCC pipeline for native 3D graphics in browsers. In: Proceedings of the 17th International Conference on 3D Web Technology - Web3D '12. ACM. ISBN 9781450314329; 2012, p. 175. doi:10.1145/2338714.2338744.
- [127] Lehmann C. Annotating 3D Content in Interactive , Virtual Worlds. In: Proceedings 18th International Conference on 3D Web Technology. ISBN 9781450321334; 2013, p. 67–70.
- [128] Manfredini AM, Remondino F. Reality-based 3D modeling, segmentation and web-based visualization. In: *Digital Heritage*. Springer. ISBN 3642168728; 2010, p. 110–24.
- [129] Abadía J, Evans A, Gonzales E, Gonzales S, Soto D, Fort S, et al. Assisted animated production creation and programme generation. In: Proceedings of the International Conference on Advances in Computer Entertainment Technology ACE 09. ACM Press. ISBN 9781605588643; 2009, p. 207. URL: <http://portal.acm.org/citation.cfm?doid=1690388.1690423>. doi:10.1145/1690388.1690423.
- [130] TweakSoftware . *RV*. 2013. URL: <http://www.tweaksoftware.com>.
- [131] SketchFab . *SketchFab*. 2013. URL: <http://sketchfab.com/>.
- [132] Exocortex . *Clara.io*. 2013. URL: <http://clara.io/>.
- [133] TeamUp.Technologies . *Lagoa*. 2013. URL: <http://home.lagoa.com/>.
- [134] Shotgun.Software . *Shotgun*. 2013. URL: <http://www.shotgunsoftware.com/>.
- [135] TeamUp.Technologies . *3DTin*. 2013. URL: <http://www.3dtin.com/>.
- [136] Autodesk . *123Design*. 2013. URL: <http://www.123dapp.com/design>.
- [137] Id Software . *Quake Live*. 2007.
- [138] Cromwell R, Webber J. *Quake 2 HTML5 Port*. 2010. URL: <https://code.google.com/p/quake2-gwt-port/>.
- [139] Herrington J, Oliver R. Critical characteristics of situated learning: Implications for the instructional design of multimedia. In: *Proceeding ASCILITE 1995*. 1995, p. 253–62.
- [140] Wickens CD. Virtual reality and education. In: *IEEE International Conference on Systems, Man and Cybernetics*. IEEE. ISBN 0780307208; 1992, p. 842–7.
- [141] Helsel S. Virtual Reality and Education. *Educational Technology* 1992;32(5):38–42.
- [142] Chittaro L, Ranon R. Web3D technologies in learning, education and training: Motivations, issues, opportunities. *Computers & Education* 2007;49(1):3–18.
- [143] Allison C, Miller A, Oliver I, Michaelson R, Tiropanis T. The Web in education. *Computer Networks* 2012;56(18):3811–24.
- [144] Linden.Labs . *Second Life*. 2003. URL: <http://www.secondlife.com>.
- [145] OpenSimulator . *OpenSim*. 2013. URL: <http://opensimulator.org>.
- [146] Allison C, Miller A, Sturgeon T, Nicoll JR, Perera I. Educationally enhanced virtual worlds. In: *Frontiers in Education Conference (FIE)*, 2010 IEEE. IEEE. ISBN 1424462614; 2010, p. T4F–1.

- [147] De Lucia A, Francese R, Passero I, Tortora G. Development and evaluation of a system enhancing Second Life to support synchronous rolebased collaborative learning. *Software: Practice and Experience* 2009;39(12):1025–54.
- [148] Zhang Q, Marksby N, Heim S. A case study of communication and social interactions in learning in second life. In: *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*. IEEE. ISBN 142445509X; 2010, p. 1–9.
- [149] Huang YC, Backman SJ, Chang LL, Backman KF, McGuire FA. Experiencing student learning and tourism training in a 3D virtual world: An exploratory study. *Journal of Hospitality, Leisure, Sport & Tourism Education* 2013;13:190–201.
- [150] Deci EL, Ryan RM. *Self-Determination*. Wiley Online Library; 1985. ISBN 0470479213.
- [151] Ryan RM, Deci EL. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American psychologist* 2000;55(1):68.
- [152] Di Cerbo F, Doderio G, Papaleo L. Integrating a Web3D interface into an e-learning platform. In: *Proceedings of the 15th International Conference on Web 3D Technology - Web3D '10*. ISBN 9781450302098; 2010, p. 83.
- [153] Callieri M, Leoni C, Dellepiane M, Scopigno R. Artworks narrating a story: a modular framework for the integrated presentation of three-dimensional and textual contents. In: *ACM WEB3D - 18th International Conference on 3D Web Technology*. ACM; 2013, p. 167–75.
- [154] Over M, Schilling A, Neubauer S, Zipf A. Generating web-based 3D City Models from OpenStreetMap: The current situation in Germany. *Computers, Environment and Urban Systems* 2010;34(6):496–507.
- [155] Rainer J, Goetz M. Towards Interactive 3D City Models on the Web. *International Journal of 3D Information Modelling* 2012;1(3).
- [156] Christen M, Nebiker S, Loesch B. Web-Based Large-Scale 3D-Geovisualisation Using WebGL: The OpenWebGlobe Project. *International Journal of 3-D Information Modeling (IJ3DIM)* 2012;1(3):16–25.
- [157] Gesquière G, Manin A. 3D Visualization of Urban Data Based on CityGML with WebGL. *International Journal of 3-D Information Modeling (IJ3DIM)* 2012;1(3):1–15.
- [158] Kolbe T, Gröger G, Plümer L. CityGML: Interoperable Access to 3D City Models. In: Oosterom P, Zlatanova S, Fendel E, editors. *Geoinformation for Disaster Management SE - 63*. Springer Berlin Heidelberg. ISBN 978-3-540-24988-7; 2005, p. 883–99.
- [159] Pintore G, Gobbetti E, Ganovelli F, Brivio P. 3DNSITE: A networked interactive 3D visualization system to simplify location awareness in crisis management. In: *Proceedings of the 17th International Conference on 3D Web Technology*. ISBN 1450314325; 2012, p. 59–67.
- [160] Lamberti F, Sanna A, Henao Ramirez EA. Web-based 3D visualization for intelligent street lighting. In: *Proceedings of the 16th International Conference on 3D Web Technology*. ISBN 1450307744; 2011, p. 151–4.
- [161] Rakkolainen I, Vainio T. A 3D city info for mobile users. *Computers & Graphics* 2001;25(4):619–25.
- [162] Cellier F, Gandoin PM, Chaîne R, Barbier-Accary A, Akkouche S. Simplification and streaming of GIS terrain for web clients. In: *Proceedings of the 17th International Conference on 3D Web Technology*. ACM. ISBN 1450314325; 2012, p. 73–81.
- [163] Prieto In, Izkara JL. Visualization of 3D city models on mobile devices. In: *Proceedings of the 17th International Conference on 3D Web Technology*. ISBN 1450314325; 2012, p. 101–4.
- [164] Walczak K, Cellary W, White M. Virtual museum exhibitions. *Computer* 2006;39(3):93–5.
- [165] Patel M, White M, Walczak K, Sayd P. Digitisation to Presentation: Building Virtual Museum Exhibitions. *Vision, Video and Graphics* 2003;.
- [166] Wojciechowski R, Walczak K, White M, Cellary W. Building virtual and augmented reality museum exhibitions. In: *Proceedings of the ninth international conference on 3D Web technology*. ISBN 1581138458; 2004, p. 135–44.
- [167] Jung Y, Behr J, Graf H. X3DOM as Carrier of the Virtual Heritage. In: *International Society for Photogrammetry and Remote Sensing (ISPRS), Proceedings of the 4th ISPRS International Workshop 3D-ARCH*. 2011, p. 475–82.
- [168] Michaelis N, Jung Y, Behr J. Virtual Heritage to Go. In: *Proceedings of the 17th International Conference on 3D Web Technology*. Web3D '12; ACM. ISBN 978-1-4503-1432-9; 2012, p. 113–6.
- [169] Rodríguez MB, Gobbetti E, Marton F, Tinti A. Compression-domain seamless multiresolution visualization of gigantic triangle meshes on mobile devices. In: *Proceedings of the 18th International Conference on 3D Web Technology - Web3D '13*. ACM Press; 2013, p. 99.
- [170] SmithsonianX3D . Smithsonian X 3D. 2013. URL: <http://3d.si.edu/>.
- [171] Benin A, Leone GR, Cosi P. A 3D talking head for mobile devices based on unofficial iOS WebGL support. In: *Proceedings of the 17th International Conference on 3D Web Technology*. ACM. ISBN 1450314325; 2012, p. 117–20.
- [172] Berthelot RB, Royan J, Duval T, Arnaldi B. Scene graph adapter. In: *Proceedings of the 16th International Conference on 3D Web Technology - Web3D '11*. New York, New York, USA: ACM Press; 2011, p. 21.