Cloud and mobile Web-based graphics and visualization

Haim Levkowitz
Computer Science Department
University of Massachusetts Lowell
Lowell, MA, USA
haim@cs.uml.edu

Curran Kelleher
Computer Science Department
University of Massachusetts Lowell
Lowell, MA, USA
ckellehe@cs.uml.edu

***Abstract*—Cloud computing is rapidly becoming one of the most prevailing computing platforms. At the same time, the Web browser has become an application platform. *Mobile+Cloud,* the combination of mobile devices and cloud-based computing is changing how users produce, consume, and use computing resources. With the introduction and penetration of HTML5, and, in particular, its visual capabilities in the form of the Canvas element, the implementation of high-quality browser-based graphics has become a reality. Indeed, WebGL offers capabilities comparable to the traditional (desktop) OpenGL utilizing in-browser computing resources. It is now feasible to have high-performance graphics and visualization "in your palm," utilizing a mobile device's browser as the graphics platform as well as the front end interface and the display. In the near future, graphics' "heavy lifting" on a cloud-based platform, coupled with a mobile client will afford high-performance graphics for most users most of the time. We argue that this will become the most common platform for computer graphics and visualization in the not-too-distant future. We further argue that such platforms will democratize the use of advanced graphics and visualization and will revolutionize analysis and display of the growing amount of data we all face every day.**

The goals of this survey are to make the reader familiar with the underlying technologies that make this possible, including (but not limited to) cloud-based computing, mobile computing, their combination, HTML5 and the Canvas element, the WebGL and other graphics libraries, and general Web-based graphics and visualization.

***Keywords*-cloud computing; computer graphics; visual analytics; visual data mining; visualization; Web-based graphics; Web-based visualization;**

I. Introduction

We all know the history of computing: very large (but not very capable) computers were located in special rooms to which access was limited. Punched paper tapes gave way to punched cards, then "dumb" terminals led the way to time-sharing. The personal computer started a long "democratization" process by providing almost anybody some compute power on their desks, and then on their laps, or in their briefcase. The Internet provided connectivity, and the World-Wide Web, easy access to information any time any place, as long as Internet connectivity is available. Recent penetration of "smart" mobile devices have extended that reach to a person's pocket or purse.

"Visual computing" [1] has been following a somewhat parallel path. However, due to visual computing's heavy performance requirements, that reach has had its limitations. Even though graphics hardware — like all other computing equipment — have become more powerful, less costly, and more portable, limitations have kept the need for special-purpose hardware to handle more than trivially-sized applications. For example, "big data" is now everywhere, but it has so far not been easy to handle such large amounts of data on general purpose, consumer-grade equipment using consumer-oriented software.

With the advent of recent new technologies, such as HTML5 and its Canvas element, WebGL, and cloud-computing, a compute environment has emerged, in which users can get "closer" to their visual computing needs, and can utilize their general purpose equipment, including their mobile devices, to view and interact with large amounts of data in visual form.

We observe that traditional, paper-based publications are rapidly giving way to electronic formats. We predict that within a relatively short period of time, most paper-based newspapers will disappear, to be replaced by electronic publications. Electronic books have been the strongest growth segment in the book publishing industry. These new platforms offer far reaching capabilities. For example, if you compare the paper version and the tablet version of any modern magazine offering both (e.g., Wired, Popular Mechanics, or your favorite choice), you will find a much richer experience on the tablet version. Unlike the static nature of paper, the electronic medium allows for a dynamic interactive presentation. Imagine a data-driven story: A very contentious topic in the USA and Europe has been the economy. Some opine that austerity is the only way out of a recession; other advocate more government stimulus spending. Each side offers conjectures and projections. The reader is asked to take their figures and projected outcomes at face value. But the current electronic media provide us with the capabilities to offer a reader an interactive "what if?" scenario, in which she can manipulate projected values (e.g., debt/GDP ratio, inflation rates, interest rates) and observe expected outcomes for growth, debt, deficit,

[1] We use the general term "visual computing" to refer to the aggregate of computer graphics, vision, imaging, and visualization.

978-0-7695-4830-2/12 $26.00 © 2012 IEEE
DOI 10.1109/SIBGRAPI-T.2012.12

and unemployment figures, all from the comfort of her easy chair, nursing a tablet.

Clearly, this requires access to true and current data, and the computational abilities to process, analyze, and present them, and to do so in response to the users interactive manipulation.

Does this sound futuristic, utopian? It is not! As we demonstrate later, more and more of that necessary data is becoming available for the general public's consumption. And, as we further show, hardware and software are rapidly moving "towards us," making these goals more realistic and attainable. This paper surveys the technologies and tools that exist today that make this scenario possible. The technologies we survey here offer a substantial amount of in-browser processing and interaction capabilities. Cloud-computing can offer additional "horsepower" where the in-browser, client-side processing capacity is not sufficient.

We survey the following technologies and their applications:

- HTML5, the cross-platform application stack of the future;
- Canvas, the visual workhorse of HTML5;
- WebGL, the technology bringing 3D graphics to the Web;
- various 2D- and 3D-graphics libraries;
- the fundamentals of cloud computing;
- the fundamentals of mobile applications development;
- *Mobile+Cloud* computing in general, and their particular applications to visual computing;
- the Semantic Web and its role in the democratization of data;
- several examples of the penetration of Mobile+Cloud applications in support of democratized data visualization and analysis.

II. A brief history of Computer Graphics and the World-Wide Web

The reader is probably well-versed in the history and fundamentals of computer graphics, the Internet, and the World-Wide Web. We provide this brief (and incomplete) history as a refresher for context purposes. For more details, the reader is referred to the references provided and, e.g., [1], [2], [3].

The early days of computing were limited to alpha-numeric text (initially in upper-case only). In 1963, Ivan Sutherland literally gave birth to the field of interactive computer graphics with the publication of his MIT Ph.D. dissertation, *Sketchpad,* an interactive system on a vector graphics display, utilizing a light-pen as its input device [4]. Further development during the 1960s included Bresenham's raster algorithms for drawing line segments, circles, ellipses, and other conic sections [5], [6]; Coons's and Bézier's parametric surfaces and computer-aided geometric design [1], [7], [8], [9], [10], [11], [12], [13]; Appel's hidden surface removal [14] and Appel's and Crow's shadow algorithms [15], [16]; Doug Engelbart's invention of the mouse at Xerox PARC (where several other revolutionary inventions were made); and the founding of Evans & Sutherland Corp., which built flight simulators on raster graphics.

During the 1970s Gouraud [17], [18] and Phong [19] developed rendering and a reflection model. Xerox PARC developed a paint program. Catmull developed parametric patch rendering, the depth-buffer (better known as the z-buffer) algorithm and texture mapping [20]. Whitted developed recursive ray-tracing, which became a standard for photorealism [21]. Apple's first computers launched commercially the personal computing. Arcade games Pong and Pac Man became popular. In 1974, the first SIGGRAPH conference on computer graphics opened, becoming the main event for presenting innovations in interactive computer graphics, computer animations, as well as other early visual computing innovations.

In the early 1980s Fournier, Fussell, and Carpenter started exploring fractals in computer graphics [22]. Adobe Systems was founded, and introduced to the market the page-layout language Postscript and the image editing software Photoshop. Animators started aiming at character animation, and video arcade games became very popular.

In 1980 Disney's TRON was the first live-action movie that included more than 20 minutes of computer animation. In 1981 IBM introduced the first IBM PC, (utilizing the 16 bit Intel 8088 CPU chip). Around the same time, the movie Raiders of the Lost Ark won an Academy Award ("Oscar") for visual effects.

In 1982 The Genesis Effect, created by Industrial Light and Magic (ILM) for the Startrek II movie was the first all-computer-animated visual effects sequence shot for a film.

In 1984 Pixar was founded, originally as a developer and manufacturer of special-purpose, high-performance graphics hardware (often referred to as "the box"). Pixar abandoned hardware in favor of software, introduced *RenderMan,* a software package and an API for network-distributed rendering of complex three dimensional views. RenderMan was designed to utilize a "render farm" of many client computers. Those clients do not require 3D graphics cards, but may take advantage of available cards [23], [24]. Pixar would later become much more well-known for its blockbuster, fully-computer-animated movies.

In 1985 the movie The Last Starfighter was the first live-action feature film that had realistic computer animation of highly-detailed models.

In 1989 the movie The Abyss was the first movie that included 3D character animation that could be described as "convincing."

At the end of the 1980s the World-Wide Web was invented by Tim Berners-Lee, who developed the Hypertext Transfer Protocol (HTTP), the Hypertext Markup Language (HTML), the first Web server, and the first Web browser.

During the 1990s Shaded raster graphics began to be featured in films. Computers started to include 24-bit raster display and hardware support for more functions that were previously only supported in software, such as Gouraud shading. The first graphical Web browser, Mosaic (later becoming Netscape) launched the popular revolution of the World-Wide Web. In 1995 Pixar released Toy Story, the first feature-length, entirely computer-generated film.

During the 2000s graphic software became more capable and much more accessible to a more general class of users.

PC display capabilities continued to advance, including the support of real-time texture mapping. Graphics input and output devices, such as scanners, printers, and cameras became affordable and penetrated the consumer market. 3D modeling advanced to be able to capture human faces and their expressions, hair, water, and other aspects that had previously been difficult to render.

During the last two decades, the World-Wide Web has progressively transformed from not-much-more than an electronic bulletin board to a full-fledged application development and deployment environment.

Around 1981-82 Silicon Graphics, Inc. (SGI) was founded by Jim Clark and several former Stanford students of his. SGI's initial market was 3D graphics display terminals. Their first systems were based on the *Geometry Engine* that Clark and Marc Hannah had developed at Stanford University. The Geometry Engine was the first hardware (VLSI) implementation of a graphics *geometry pipeline,* specialized hardware that accelerated the geometric computations needed to display images of three-dimensional scenes. Access to SGI's high performance 3D graphics subsystems was originally done through a proprietary API, the *IRIS Graphics Language (IRIS GL).* In 1992, SGI revamped its proprietary GL and created the *OpenGL* API and licensed it to competitors at a low price. Subsequently, SGI created the *OpenGL Architecture Review Board,* an industry-wide consortium to maintain the OpenGL standard. This led to the ability to write cross-platform graphics programs. OpenGL has since been the primary real-time 3D graphics standard that is portable across several operating systems and platforms. *OpenGL-ES* ("Embedded Systems") runs on many types of mobile devices and embedded systems. *WebGL,* one of the primary technologies surveyed in this paper, is a derivative of OpenGL-ES to a Web-based graphics environment.

III. HTML5

"Developers of software for the World-Wide Web say the new HTML5 standard is revolutionizing the way the Web evolves, works, and is used. It is simplifying the work of programmers, harmonizing access to diverse devices and applications, and giving users amazing new capabilities, they say." [25]

HTML5 is to Mobile+Cloud as Java is to desktop computing: a cross-platform application-building technology.

While HTML5 sounds like just a new version of the HTML markup language itself, several standards are wrapped into it: DOM, the Document Object Model for accessing and manipulating HTML documents; CSS or Cascading Style Sheets, to define the presentation and appearance of a document; and JavaScript, which is rapidly becoming one of the most used scripting languages. But even more so, the term HTML5 is often used as an inclusive collection of specific APIs, which enable the utilization of client-side resources, such as location-based services, client-side graphics, client local storage, and audiovisual capabilities, including sound-, camera-, photo- and video handling and presentation.

The World-Wide Web Consortium (W3C) oversees the development of HTML5, which is a central piece of its *Open Web Platform,* the combination of the markup language itself and the technologies that are associated with it, including its graphics capabilities (which are the focus of this survey). The W3C governs the HTML5 standards, but vendors implement the standards independently. Vendors have the freedom to innovate, and these innovations often become W3C standards. For example, Apple unilaterally introduced the Canvas element in 2004, way before it found its way into the latest HTML standard definitions. Because of this dynamic "push and pull" between vendors and standards, implementations of the standards are continually increasing in quality and the evolution of the technology is not the responsibility of any single entity. Building applications with HTML5 avoids vendor lock-in, and affords compatibility across most desktop browsers and mobile devices (when developers observe certain rules and constraints). [26], [27], [28]

IV. Web-based Visual Computing

The primary W3C graphics technologies today are HTML5 Canvas, WebGL, and Scalable Vector Graphics (SVG). Graphics and visualization libraries have been built upon these standards, providing developers higher levels of abstraction for working with interactive graphics.

HTML5 Canvas is an immediate-mode graphics API for the Web [29]. The concepts behind Canvas are not fundamentally new; rather, they represent the introduction of well-known 2D graphics techniques into the world of HTML and JavaScript. Many similar APIs have come before Canvas, such as Cairo, Java2D, graphics libraries of the .NET platform, QT, and GTK. Because these types of frameworks give developers full access to the graphical display, they allow developers to build entire applications from the ground up with custom designed looks-and-feels and graphical behavior. Now this capability has come to the Web, and to the mobile devices that are increasingly being used to access it and view its contents. For this reason, some have said that the Canvas is the single most powerful HTML5 element [30].

WebGL is a JavaScript API for rendering 2D and 3D graphics within supporting Web browsers without the need for any external assistive technologies, such as plug-ins. It is based on OpenGL ES 2.0, a simplified version of the standard graphics library OpenGL that provides immediate-mode 3D graphics capability. No concept of WebGL is fundamentally new, but rather it represents the introduction of hardware-accelerated 3D graphics into the world of HTML and JavaScript. WebGL brings known graphics capabilities to the Web, including rendering 3D scenes, lighting, textures, and definition of shaders. Scene graph libraries, such as *Three.js* [31], provide implementations of retained-mode 3D graphics [32]. Physics libraries like Box2D.js provide physics simulation [33].

Scalable Vector Graphics (SVG) is a DOM-based W3C standard for retained-mode vector graphics [34]. When using SVG, developers specify a 2D scene graph by manipulating the DOM, and the SVG implementation is responsible for

rendering the scene to a bitmap for display whenever updates occur. *Data Driven Documents* (*D3.js,* [35]) is a notable library based on the use of SVG for developing interactive visualizations.

V. HTML5 Canvas

```
<html>
<body>
  <canvas id="myCanvas" width="200" height="200" />
  <script>
    var canvas = document.getElementById("myCanvas");
    var ctx = canvas.getContext("2d");
    ctx.fillStyle = "red";
    ctx.fillRect(0, 0, 200, 200);
    ctx.fillStyle = "green";
    ctx.fillRect(50, 50, 100, 100);
    ctx.fillStyle = "blue";
    ctx.fillRect(75, 75, 50, 50);
  </script>
</body>
</html>
```

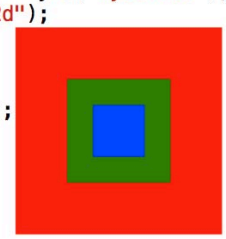

Fig. 1. Example code that draws colored rectangles using HTML5 Canvas and the resulting graphic.

The HTML5 Canvas API allows developers to insert a rectangular bitmap as an element in an HTML page and access it through a JavaScript API. The Canvas bitmap is essentially an array of colored pixels, and the API provides functions for manipulating the bitmap using well-known 2D graphics concepts and techniques. Figure 1 shows an example code that draws colored rectangles using HTML5 Canvas and the resulting graphic. The Canvas API was originally defined and implemented by Apple in 2004, and was subsequently proposed as a standard. Today Canvas is implemented in all major browsers, and polyfills, such as ExplorerCanvas and Google Chrome Frame, support backward compatibility with older browsers. In this section we outline the capabilities of Canvas in detail.

Like many other 2D graphics frameworks, Canvas uses a stateful context for determining properties of graphical elements drawn. The Canvas 2D context includes the following variables [30]:

- `canvas` — a reference to the containing Canvas element;
- `fillStyle` — the color, gradient, or repeated image pattern used for filling shapes;
- `strokeStyle` — the color, gradient, or repeated image pattern used for stroking (tracing the outline of) shapes;
- `font` — the font used by `fillText` and `strokeText`;
- `globalAlpha` — a global alpha value between 0 (transparent) and 1 (opaque) applied to all drawing operations;
- `globalCompositeOperation` — the operation used when compositing graphical layers;
- `lineCap` — the form used for line endings. Valid values are:
  - `butt` — draws the ending as square and exactly matched to the end point;
  - `round` — draws a rounded ending, as a circle centered on the end point;
  - `square` — draws the ending as a square whose center is the end point;
- `lineWidth` — the width (in pixels) of lines drawn;
- `lineJoin` — the way line joins are drawn. Valid values are:
  - `bevel` — a flat connecting face is drawn;
  - `round` — the join is rounded;
  - `miter` — the lines extend to meet at a point;
- `miterLimit` — the maximum extent that lines will extend to meet one another when `lineJoin` is set to `miter`;
- `shadowBlur` — the amount shadows are blurred;
- `shadowColor` — the color of shadows;
- `shadowOffsetX`, `shadowOffsetY` — the vector (in pixels) by which shadows are shifted;
- `textAlign` — the horizontal alignment of text. Valid values are “start,” “end,” “left,” “right,” or “center”;
- `textBaseline` — the vertical alignment of text. Valid values are “top,” “hanging,” “middle,” “alphabetic,” “ideographic,” or “bottom” [36].

Once the context variables are set, paths that use their values can be drawn to the canvas. Functions are provided for pushing these context variables onto a stack (`context.save`) and popping them off the stack (`context.restore`). The Canvas element provides functions for drawing lines, polygons, arcs, and text. Paths can be filled with solid colors, gradients (both linear and radial), or repeated image patterns. Bézier curves are also supported. Styles can be set for line caps and joins. A comprehensive text drawing API is provided.

The Canvas API exposes the pixels of the bitmap directly, allowing the construction of filters, such as blur, invert, and emboss. This functionality also enables implementation of computer vision algorithms. Image data can be exported and potentially uploaded to a server. A compositing API that supports composition of multiple bitmaps using various techniques is also provided. Shadows are supported, but carry a severe performance penalty when used. External image data and frames from an HTML5 Video element can be loaded and drawn on the Canvas.

The Canvas API alone is not enough to build interactive graphics applications. Mouse and keyboard events (and multi-touch events from mobile devices) must be accessed through JavaScript and used to drive graphical manipulations. A special function, called `requestAnimationFrame`, has been introduced for implementing smooth animations. This is a better alternative to using `setInterval` for an animation loop because it provides timing of animation frames that are synchronized with the refresh rate of the graphics hardware, and implementations can provide additional optimizations (e.g., not executing animations when the page is not visible). As JavaScript is single threaded, intensive computations that would slow the animation loop (such as physics or image processing) can be offloaded to other threads using Web

Workers [37], [38]. Using WebSockets [39], a W3C standard for bidirectional communication between servers and clients, real-time multi-user scenarios can be brought into Canvas-based applications. When used in conjunction with these features, Canvas provides a basis for constructing full-featured interactive graphics applications.

HTML5 is rapidly being deployed now but is not a finished standard. Adoption levels vary. For example, there is currently no single HTML5 standard for video compression (codec), streaming protocols, or digital rights management (DRM). Until recently it was almost taken for granted that Adobe's Flash would define de-facto video standards. However, Apple's rejection of Flash for iOS has completely changed the map. Apple has vigorously implemented and promoted the use of HTML5 as an in-browser video standard. This has fractured Flash's status, especially after Adobe announced it would not deliver a mobile-capable version. In addition, at this time both Microsoft and Google have taken slightly different approaches toward video delivery. More generally, browsers coverage and support of HTML5 varies. (Users can test their browser's HTML5 support at [40].) "But the individual specifications are at different maturity levels and will become standards at different times." [25]

VI. 2D Graphics

Two-dimensional graphics on a Web browser can be supported by a number of libraries and tools.

A. 2D Graphics Libraries

Processing.js is a library for immediate-mode graphics that uses HTML5 Canvas [41], based on the original Java-based Processing project [42]. The Processing language parser and the Processing graphics API implementation are the two major components of Processing.js. The Processing language parser transforms source code written in the Java-like Processing language into JavaScript. This enables developers to execute Processing programs in an HTML5 environment with little or no code modification. Processing.js implements the original Processing graphics API using HTML5 Canvas for 2D features and WebGL for 3D features. In practice, many developers prefer to use the Processing API from JavaScript rather than write their software in the Processing language. This is because the introduction of another language into a project introduces complexity, and because Processing programs are difficult to debug.

Paper.js is a scene graph library for 2D graphics that uses HTML5 Canvas [43]. Paper.js provides a vector graphics scene graph similar to that of Adobe Illustrator. The Paper.js API is inspired by, and mostly compatible with Adobe Scriptographer, a JavaScript scripting plugin for Adobe Illustrator, created by the developers of Paper.js. Paper.js provides support for common graphics primitives, groups, layers, paths (Bézier curves) with outline drawing, mouse and keyboard interaction, working with raster images, and vector geometry operations. The main contribution of Paper.js is that it brings the vector graphics model of Adobe Illustrator to the Web, giving developers a straightforward cross-browser retained-mode 2D graphics API.

Data Driven Documents (D^3) is a JavaScript library written by Mike Bostock for 2D Web-based interactive data visualization [35]. D^3 solves the problem of performing Document Object Model (DOM) manipulation based on data. D^3 uses a declarative approach leveraging functional programming techniques, so developers can write concise statements to manipulate the DOM based on data rather than writing verbose and convoluted data transformation code using the raw DOM API. D^3 does not introduce its own graphics API or scene graph model, but rather provides developers a powerful tool for leveraging Web standards such as SVG and CSS. D^3 is the successor to ProtoVis [44], a visualization library that did introduce its own graphics vocabulary. Figure 2 shows an example applications of the Data Driven Documents (D3.js) library demonstrating the capabilities of SVG (Scalable Vector Graphics).

VII. 3D Graphics

Three-dimensional graphics on a Web browser can be supported by a number of libraries and tools.

A. WebGL

As mentioned above, WebGL (Web Graphics Library), is a JavaScript API for rendering 2D and 3D graphics within supporting Web browsers without the need for any external assistive technologies, such as plug-ins. It is based on OpenGL ES 2.0, a simplified version of the standard graphics library OpenGL that provides immediate-mode 3D graphics capability.

OpenGL was devised for desktop graphics. OpenGL ES 2.0 is an OpenGL subset, designed for embedded systems, including mobile devices. WebGL is based on, but not identical to OpenGL ES 2.0. The differences between WebGL and OpenGL ES 2.0 have been documented in [45].

WebGL brings known graphics capabilities to the Web. WebGL has been integrated into Web browser standard elements, providing graphics processing unit (GPU) acceleration of all typical graphics functionality, but carried out via the Webpage's Canvas.

WebGL elements are introduced into a Web page just like any other HTML elements and can be mixed with them.

To write a WebGL program, a developer needs to write JavaScript control code as well as rendering (also referred to as *shader*) code to be executed on the GPU.

The WebGL Working Group was initiated in 2009 by Khronos Group, a non-profit technology consortium. Apple, Google, Mozilla, Opera, and others were among the first to participate [46], [47], [48], [49].

Several libraries have been introduced for WebGL development. Among them are WebGLU (the first WebGL library made publicly available), GLGE, C3DL, CopperLicht, GLOW, SpiderGL, PhiloGL, gwt-g3d, SceneJS, X3DOM, Oak3D, Processing.js, Three.js, KickJS, OSGJS, XB PointStream, CubicVR.js, A3 (Aerotwist), Jax, ANGLE, XTK — The X

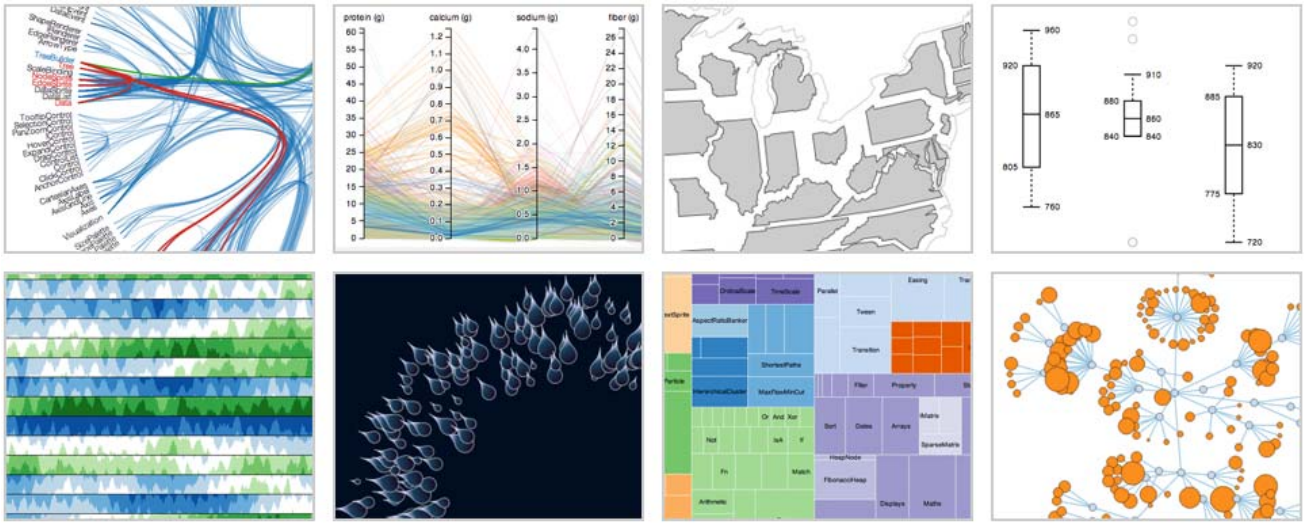

Fig. 2. Example applications of the Data Driven Documents (D3.js) library [35] demonstrating the capabilities of SVG (Scalable Vector Graphics).

Toolkit (WebGL for Scientific Visualization), EZvideo-Web, and Zlatnaspirala.

We refrain from describing all these various libraries. Instead, we discuss just a few, and refer the reader to the literature for details on the other ones.

B. 3D Graphics Libraries

We have described Processing.js in Section VI-A.

Three.js is a JavaScript library for 3D graphics that abstracts the rendering layer, allowing developers to write the same code for rendering 3D graphics using either WebGL, HTML5 Canvas, or SVG [31]. Three.js introduces standard 3D graphics abstractions, such as a scene graph, cameras, meshes, materials, textures, lighting models, common objects (e.g., cubes and spheres), support for loading 3D models in the Collada file format, and more. A wealth of examples using Three.js are published on the Web. A physics engine has been created for Three.js called *Physijs* [50], built on Ammo.js (a direct port of the Bullet physics engine to JavaScript) [51]. Figure 3 shows an example applications of the Three.js library demonstrating the capabilities of WebGL.

VIII. Mobile App Development and Deployment

An integral part of the Mobile+Cloud vision, includes mobile applications (*"mobile apps"* in the current lingo). While mobile apps provide general functionality on mobile devices, such as smart phones and tablets, they stand to play an increasing role in what is the focus of this survey: visual computing that is executed on Web browsers, in collaboration with cloud-based data repositories and servers. For this reason, we find it important to understand the basics of mobile apps development and deployment.

A. Developing apps

HTML5 is increasingly serving as a common "write once, run anywhere" platform for developing mobile applications. Rather than write an Android native application using Java, then port the application to iOS using Objective-C, then port it again to Windows Mobile using Microsoft's tools, one can now write applications once using HTML5 (utilizing some mobile-specific compatibility tricks) and deploy them as native applications to all of these platforms using tools such as PhoneGap [52] or Appcellerator-Titanium [53] . While not without its own shortcomings and problems, this cross-platform capability is a powerful force driving HTML5 toward being the most widely used application development approach.

1) Native apps: Native apps have capabilities not available to Web pages due to some limitations, such as browser sandboxing. These capabilities include:

- Access to the device's file system;
- native operating system features (e.g., launching sub-applications with Android intents);
- hardware accelerated graphics;
- accelerometer;
- camera;
- compass;
- geolocation;
- notifications with alerts, sound, or vibration;
- access to the user's contacts;
- access to the revenue-generating app marketplaces, such as the Apple Store or Google Play (formerly the Android Market).

There is a high cost associated with developing applications with native tools for multiple mobile platforms. Imagine a company chooses to develop an Android application using the Java programming language and Android tooling. The app becomes a huge success and makes the company millions. Now the company wants to release a version that will run on iPhones and iPads (iOS devices). In order to get their application to work on iOS devices, the company must invest many hours of development time to re-implement the appli-

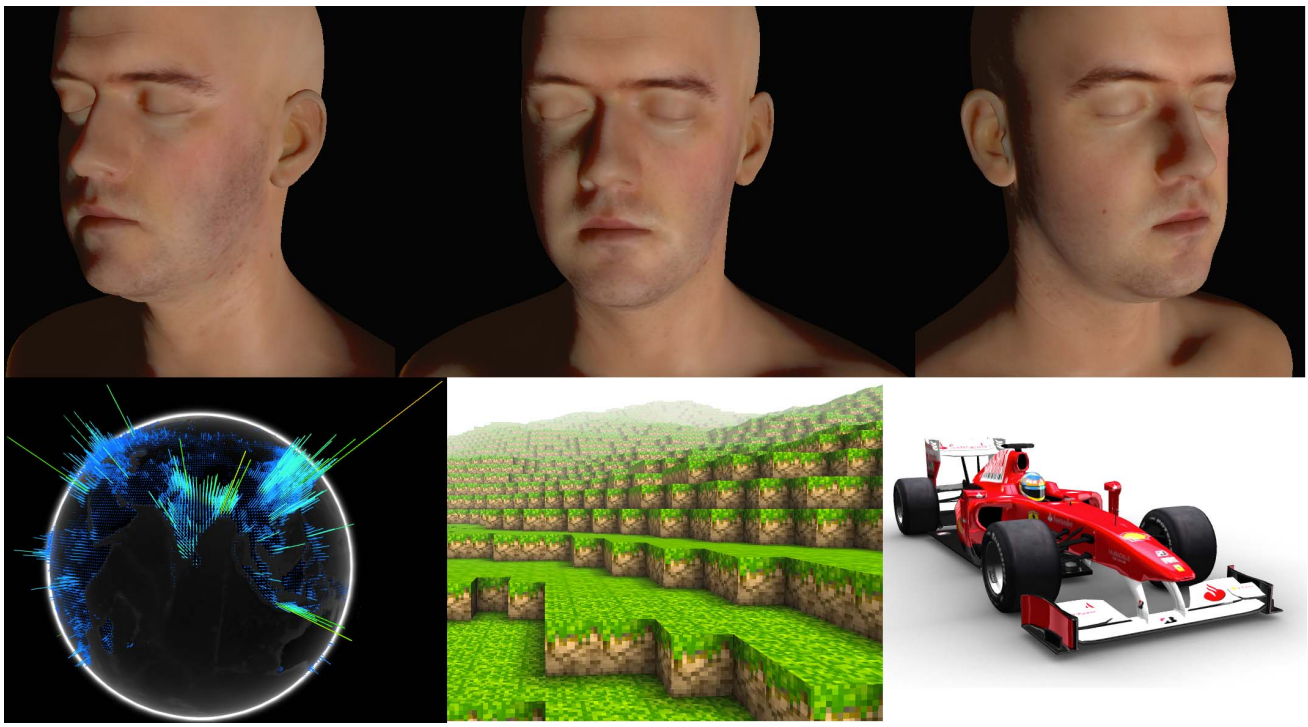

Fig. 3. Example applications of the Three.js library [31] demonstrating the capabilities of WebGL.

cation using Objective-C and Apple developer tooling. For every other mobile platform the company wants to support, they will need to invest huge amounts of developer time for learning proprietary app development tools and porting the code to different programming languages.

2) HTML5 apps: In recent years, HTML5 has emerged as a viable cross-platform solution. When an application is developed in HTML5, it can be deployed as a

- Web page for desktop browsers;
- Web page for mobile browsers;
- native application for a multitude of mobile platforms.

This means that if a developer originally developed their application using HTML5, they could reach a much larger audience (and thus make a much larger profit) with a lower cost of application development and deployment.

Applications developed with HTML5 can "look-and-feel" like native apps. Though this is possible, it does not come automatically. Developers must use a number of tricks in order to achieve the look and feel of a native mobile app [54]. These include:

- Detect the display resolution and render an appropriately-scaled version of the application. Applications often require several different layouts for different display resolutions, or if possible, the user interface elements are simply scaled to match the display.
- Detect when device orientation changes (i.e., it is turned sideways) and render the application accordingly. The elements of the user interface often must have a different layout for different orientations. Care must be taken to ensure both work well.
- Hide browser-specific visual components. Measures can be taken to hide the mobile browser's navigation bar and other distracting visual elements. If these are hidden, the user only sees the application, in full screen.
- Use a matching icon. When a Web page is saved as a shortcut to the desktop of a mobile device, it is represented as an icon that can be specified by the page. If an icon of the correct resolution is properly configured, the icon that appears on the mobile desktop will look no different than an icon for a native application.
- Use local storage. If the HTML5 local storage API is used, the application can continue functioning normally (read and write its state) even when Internet connectivity is interrupted. Even with cloud-driven applications, local storage can be a temporary holding place for changes to be synchronized with a cloud service when a connection is re-established. This can make the difference between an app that is usable at all times and one that has limited usability.

By taking these measures, developers can deploy HTML apps as Web pages and have them look and feel like native apps. However, the app is still a Web page, so it cannot be offered or sold in an app marketplace. In the next section, we discuss how HTML5 apps can be deployed as native apps.

B. Cross platform development tools and environments

Most mobile development platforms have a user interface component available that can display a Web page within a native app. This means that it is possible to build native apps that simply load a full screen view of a Web page, which can itself be a full-featured HTML5 app. In this way, HTML5 apps can be deployed as native apps.

In addition to the ability to build a native mobile app that displays a Web page, it is also possible to modify the runtime of that Web page such that arbitrary functions in the native app are available to JavaScript code on the loaded Web page. This means that arbitrary functionality available to native apps can be exposed via a JavaScript API to Web pages. In this way, HTML5 apps can gain all the functionality available to native apps.

Thus the pattern for deploying HTML5 apps as native apps is as follows:

- Build a native app that displays a Web page;
- Expose native functionality to the Web page via a JavaScript API;
- Build the HTML5 app as a Web page embedded within a native app, taking advantage of native functionality via the JavaScript API.

1) PhoneGap: Because the pattern for transforming HTML5 apps into native mobile apps is straightforward, it can be automated. Because mobile device capabilities are similar across platforms, the JavaScript API to native functionality can be standardized. The PhoneGap project automates the transformation of HTML5 apps into native mobile apps, and specifies a single JavaScript API for accessing native capabilities across many mobile platforms [52]. Using PhoneGap, developers can author HTML5 apps once, then derive native apps for the following platforms in a single automated step:

- Android;
- iOS;
- Windows Mobile;
- WebOS;
- Symbian;
- Bada.

Once a native app is created, the author can go through the typical steps for deploying the app to revenue-generating app marketplaces, such as Apple's App Store and Google Play.

2) Appcellerator Titanium: Appcellerator Titanium [53] is a comprehensive mobile app development platform based on JavaScript APIs and HTML5. Appcellerator includes:

- A software SDK for developing mobile apps;
- An Eclipse-based IDE equipped with platform-specific tooling;
- A suite of cloud-based services to speed app development;
- An app analytics service.

The Titanium Mobile SDK provides developers the means to write an app once in JavaScript and automatically deploy the app to numerous mobile platforms (as native apps as well as mobile Web apps) using a process similar to that of PhoneGap. The Appcellerator development environment, Titanium Studio, is an Eclipse-based IDE that offers platform-specific functionality for developing, testing, and deploying mobile apps. Appcellerator Cloud Services provide scalable back end features that are common to many apps and would normally need to be implemented by app developers. These features include user management, logins, photo uploads, push notifications, and status updates. The Appcellerator Analytics service aids app developers in collecting and analyzing data about users, sessions, and actions taken within apps.

C. Deploying apps

So far we have covered how to build an HTML5-based application and package it as a native app for various platforms. In this section we will survey the means by which authors can get their app "out there" and generate revenue. As of June 2012, 51.8% of smart phone owners had a device running Android, and 34.3% had a device running iOS [55]. Therefore the corresponding app marketplaces, Google Play ("Android Market") and the iOS App Store, are the two native app marketplaces that reach the most users.

1) Android Market and Google Play: The Android Market is Google's app marketplace for Android. App developers can publish their apps in the Android Market, and the process of payment and deployment to client devices is managed by the platform. Android Market has evolved into a new service called Google Play, which features a new interface, new app discoverability features (such as a recommendation system), and sells not only apps but also music, books, and movies. Publishing apps to Google Play is an automated process that is free for developers. This mean that after having developed an Android app, authors can follow a straightforward app preparation and Web-based publishing process [56], and their app will appear in the marketplace within minutes.

2) iOS App Store: The iOS App Store is Apple's app marketplace for iOS devices, including the iPhone, iPod Touch, and iPad. The App Store follows a similar approach to that of the Android Market, but in contrast is very strictly moderated, and costs developers $99 per year to develop, submit, and keep apps available at the Store. The App store has been notorious for rejecting apps for nebulous reasons [57].

3) Chrome Web Store: The Chrome Web Store is an app marketplace from Google, aimed at Chrome users [58]. The concept of a "Chrome App" is that of a Web app with additional metadata stored by Chrome, with an icon to launch the app from the Chrome home page. Many of these apps are free, but collecting payment is possible when the Chrome Web Store Payments system is used. The cost for becoming an app author on the Chrome Web Store is $5, and there are no recurring fees for authoring each app.

Several other app stores have emerged, including Amazon's App Store, Zeewe, a marketplace for Mobile Web Apps, TapJS, a game hosting service, and Playtomic. Based on the market share statistics presented above, developers can reach the largest majority of their potential audience by focusing on

Apple's App Store (the only marketplace for iOS apps) and Google Play for Android apps.

IX. Cloud Computing

Cloud Computing has become so prevalent in the computing scenery that it requires little, if any discussion. Cloud Computing is an umbrella term for various levels of storage and processing services deployed on outsourced servers (on "the cloud"), providing computing and storage infrastructure on an "as a service" basis. The three types of Cloud Computing are:

- SaaS: Software as a Service — application software running on a service-provider's network of servers instead of software packages installed and running on a user's local computer (e.g., Web-based email, Google Docs);
- PaaS: Platform as a Service — a computing platform, including an operating system, execution environments for various programming languages, databases, and Web servers (e.g., AWS—Amazon Web Services, including Amazon's EC2—Elastic Compute Cloud and S3—Simple Storage Service; Heroku; Google App Engine; Windows Azure);
- IaaS: Infrastructure as a Service — service providers offer computers, most often as virtual machines, raw storage, load balancing, and network support. Users have the responsibility to install operating systems and application software (e.g., Amazon CloudFormation, Rackspace, and Google Compute Engine).

One of the major tenets of the as-a-service model, in either of these types, is the *elasticity* concept: users rent (and pay for) only as much capacity as they need. They can increase or decrease their resource utilization as demand grows or wanes. This is a very attractive scalable model, in which a budding enterprise can obtain minimal amounts of resources, but can scale up seamlessly as demand and need grow. The business model behind Cloud Computing is similar to a utility: just like your electric, gas, or phone company bills you for metered usage, so does a cloud provider.

X. Cloud Computing and Mobile HTML5 Apps

With the proliferation of cloud computing and ever more capable mobile devices a new compute model is rapidly gaining foot: Mobile+Cloud Computing. We anticipate it to become the prevailing compute model for most applications. It is compelling because, while the increase in mobile devices' capabilities make them more and more "computers in our pockets" (or in our palms), they still exhibit processing limitations, even compared to desktop or laptop computers. However, linked to a cloud-based infrastructure, they can be the front end to a hybrid computation model, in which some computing gets executed on the mobile device, but the cloud is being used for more demanding compute tasks, larger storage requirements, as well as the ability to provide a fully synchronized experience among a user's increasing number of devices, including a smart phone, a tablet, laptop and desktop computers, and more.

Due to the recent growth in cloud computing discussed above, it has become possible to develop and deploy mobile apps using only a Web browser. For example, a budding app developer can get started at no cost with the following steps: create a repository in GitHub, deploy it as a Web site using GitHub Pages, use Cloud9 IDE to develop an interactive Canvas-based mobile Web app, then deploy it to the GitHub Pages site. When server-side resources are needed (such as a database or real-time communication service), the developer can then use Cloud9 IDE to build server-side software using Node.js and deploy it to the Heroku cloud in a scalable manner. At this stage, a service called PhoneGap Build can automatically build native apps from your source code in the cloud, giving you apps ready for deployment in app marketplaces. In this way, modern services such as GitHub, Cloud9 IDE, Node.js, Heroku, and PhoneGap Build can form a complete browser-based app development and deployment toolchain.

XI. Interactive Visualization on Mobile+Cloud

Visual computing in general, and interactive visualization in particular, stand to gain from the Mobile+Cloud model. This is because, on the one hand, they require substantial computing resources, and on the other hand, they need visual, interactive displays.

We briefly survey early demonstrations of these capabilities now.

A. Case studies

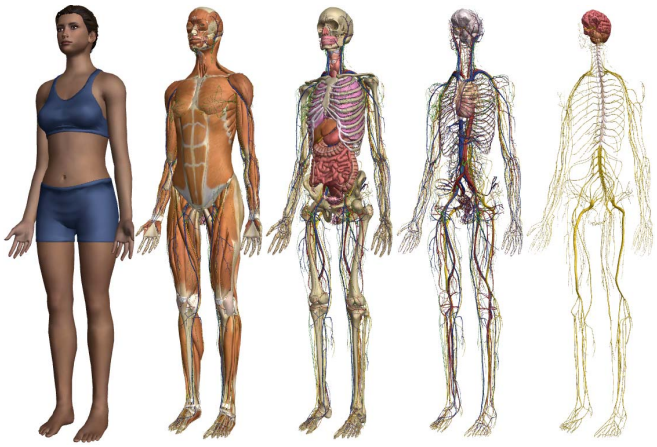

Fig. 4. Layers of Zygote Body (formerly Google Body Browser) [59], an exemplary application of WebGL.

1) Google Body Browser: Google Body Browser, now known as Zygote Body [59], is a virtual human body. Fashioned after the National Library of Medicine's Visible Human Project, it was originally launched in the end of 2010 to demonstrate the Chorme Web browser's capabilities, utilizing HTML5, its Canvas element, and WebGL.

Users can peel away layers, such as skin, muscles, bones, the vascular system, and the nervous system. Users can zoom

and rotate the 3D model. Users can click on each individual body part to highlight it and get its name. A search box allows for finding body parts by name. This is an exemplary example application of WebGL. Body Browser has been ported to a native Android app [60], taking advantage of native app features such as data bundling and accelerated graphics. Other similar projects have appeared, such as *BioDigital Human* [61] and the *OpenWorm Browser*. Figure 4 shows layers of Zygote Body.

2) Google Maps: Google Maps, a Web-based mapping service with a rich feature set, is a perfect example of a visual computing application in which the graphics "heavy lifting" of rendering the map images takes place on the server side. Google Maps provides services for navigating maps with several views (satellite imagery, street maps, bicycle maps, terrain, weather, photos, traffic, and more), find directions from place to place, share map configurations and more. Google Maps exists as a Web app for desktop browsers, a Web app for mobile devices, and an optimized native app for many mobile platforms. The native apps take advantage of the device's geolocation, and provide additional features, such as "show me where I am now" and live driving and walking directions. Google also provides an API that allows developers to effortlessly embed maps within their apps [62]. See Figures 5 and 6 for examples of some Google Maps' capabilities.

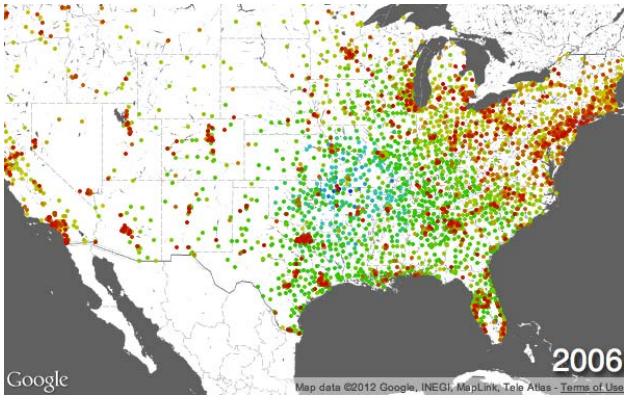

Fig. 5. An animated visualization of Wal Mart stores popping up across the US created using the Google Maps API (Google Maps blog post).

3) TileMill: TileMill is an open source project managed by the company MapBox for rendering map tiles on the server side, styled using their own CSS-like map styling language called CartoCSS [63]. TileMill is based on open source projects, including Mapnik and Node.js. TileMill is a powerful tool for developing interactive map applications that can, for example, overlay a data visualization on a map, or re-style existing map data, such as OpenStreetMap [64]. See Figures 7–10 for examples.

4) Tableau Public: Tableau Public is an adaptation of the Tableau visualization software to the social Web. The Web-based version of Tableau enables interactive visual analysis

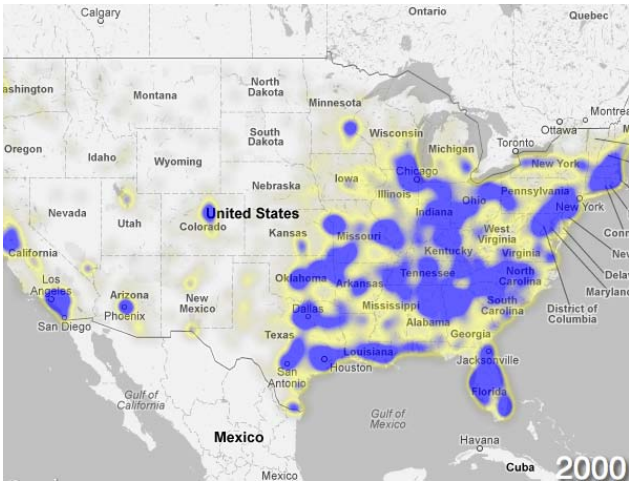

Fig. 6. A demonstration of Heat Map capabilities in the Google Maps API. The Heat Map is rendered on the client side as a layer on top of map tiles rendered on the server side (Google Maps blog post).

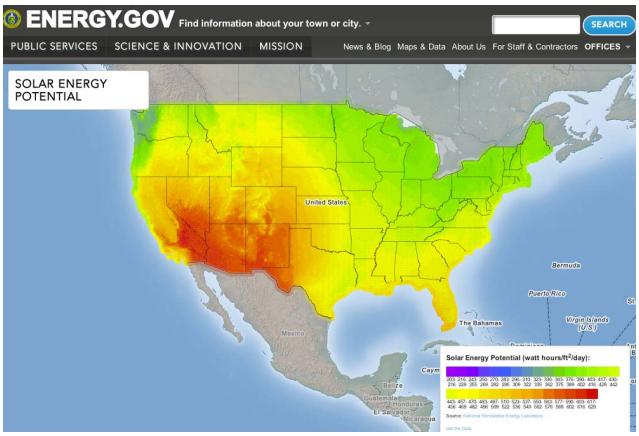

Fig. 7. A HeatMap visualization of solar energy potential across the US (data from the National Renewable Energy Laboratory). The visualization is rendered on the server side using TileMill (Energy.gov).

by relaying user interactions to server-side instances of the Tableau software, rendering the visualization on the server, then sending the rendered image to the client. Tableau Public has a gallery feature, where users can post their visualizations, comment on them, share them across the Web, and embed them into their own pages [65].

B. Web Based Development Showcases

1) OpenProcessing: "OpenProcessing is an online community platform devoted to sharing and discussing Processing sketches in a collaborative, open-source environment." [42], [66]

2) Google Chrome Experiments: Google Chrome Experiments is an initiative to collect example applications of HTML5 technology [67]. This initiative has showcased projects including:

Fig. 8. 3D Buildings, server-side rendered: An example of server-side 3D building rendering using TileMill, showing a close up of Manhattan (building data from Sanborn)

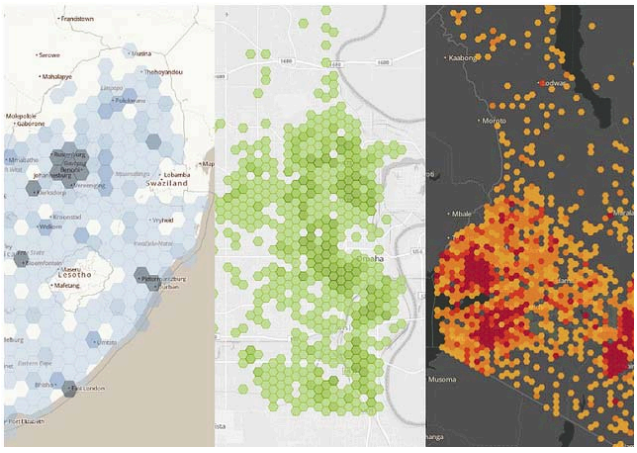

Fig. 9. Binning aggregation to show density: Examples of hexagonal binning to show density data on maps using TileMill.

- The Wilderness Downtown — an interactive music video that has you enter your address when it starts, then builds dynamic scenes based on images from that address taken from Google Maps and Google Street view.
- 3D Water Waves — a demonstration of fluid simulation and advanced 3D graphics techniques using WebGL.
- Progressive Julia Fractal — a lightning fast implementation of Julia set rendering using shader-based image distortion by Felix Woitzel. This demonstrates the potential for general purpose GPU computing using WebGL [68], see Figure 11.
- WebGL Experiments — demonstrations of several other WebGL-based implementation [69].

XII. Data on the Web

As we have argued above, we are convinced that this new Mobile+Cloud platform we have been surveying will

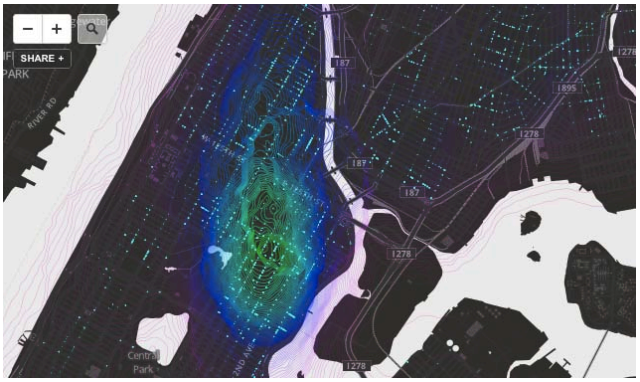

Fig. 10. Police Stop and Frisk Data: An example of using contour plots for non-topographic visualization.

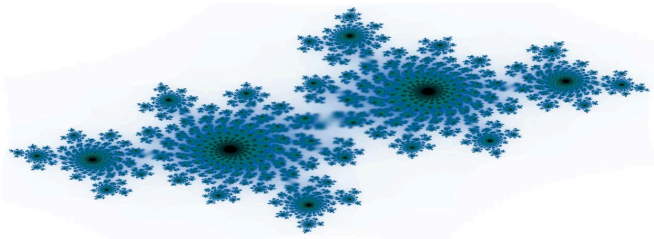

Fig. 11. An image-based progressive Julia Set fractal, computed on the GPU using shaders defined with WebGL.

democratize data analysis, we discuss now some technologies and initiatives that have already been launched, which already demonstrate this evolution.

A. Semantic Web and the democratization of data

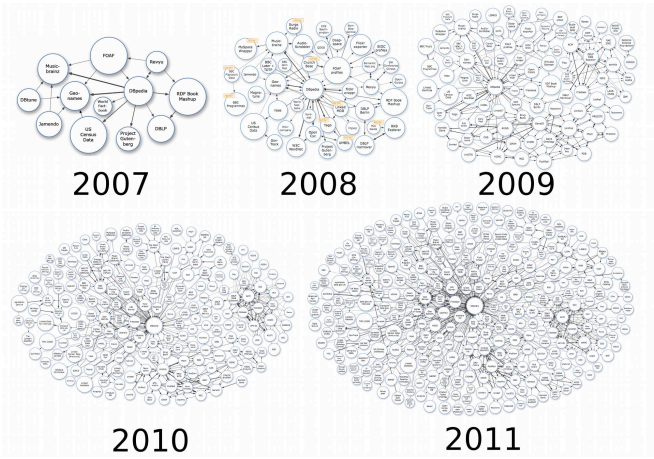

Fig. 12. The growth of the Linked Open Data Cloud [70], from 12 datasets in 2007 to 295 in 2011.

The Semantic Web has been termed the "Web of Data," the "Giant Global Graph," the "Data Web," the "Linked Data Web," and the "Enterprise Information Web." These are all terms for the same thing. The key vision of the Semantic Web

is to link explicit data published on the Web in a machine-readable fashion in order to enable applications, including targeted search, data browsing, and intelligent agents [71].

Resource Description Framework (RDF) [72] is the foundational data representation framework for the Semantic Web. RDF represents data as triples of $(subject, predicate, object)$. With this model, RDF is capable of representing any data stored in relational databases. Higher levels of the Semantic Web framework include the Web Ontology Language (OWL) for representing domain ontologies and inference rules [73]. Vocabularies provide data publishers common means of expressing domain concepts. More general vocabularies, such as the RDF Data Cube Vocabulary [74] aim to enable integration of statistical data from many data providers.

Public data sets are increasingly being published within the Semantic Web. Visual data analysis tools, when built to consume data of this form, will be able to access data as it becomes available. The combination of the Semantic Web and Web-based Visual Computing may lead to a world in which a digital mirror world is readily available for access by anyone, and rich Web-based visualization and analysis tools that can be applied to such information are commonplace. Figure 12 shows the growth of the Linked Open Data Cloud [70], from 12 datasets in 2007 to 295 in 2011.

B. Public data initiatives

We survey now a few initiatives providing public data.

1) UN, World Bank, US Federal Data, more: The United Nations has published a huge amount of publicly-available data, consisting of 34 databases and 60 million records in total [75]. The data sets consist primarily of indicators over years at the level of countries. Topics covered include economics, energy, human development, crime, health, tourism, telecommunications and more.

`Data.gov` provides access to many data sets.

The Fedstats site at **`fedstats.gov`** publishes links to numerous public data sets hosted by various US government organizations.

2) Example visualizations of public data: The following are a few examples demonstrating how public data and Web-based visualization can be employed to help understand different problems, issues, and the world in general.

- *GapMinder World* is a Web-based visualization of important trends around the World, including "Wealth & Health of Nations (how long people live and how much money they make)," "CO_2 emissions since 1820," "Africa is not a country! (there are huge differences among the countries in Africa)," "Is child mortality falling?" and "Where is HIV decreasing? (changes in the number of people living with HIV)." In each one of these, an interactive visualization allows a visitor to move through time (from 1800 through today), and to interact with the various parameters to see how trends have changed over time, geographic location, and various other categories. Parameters include income per person, children per woman, child mortality, life expectancy, economy, society, education, energy, environment, health, infrastructure, population, and work. Each one of these offers additional choice granularity. Advanced users can select from additional advanced features to further enrich their interaction with, and exploration of the data [76]. See Figure 13 for an example.

Fig. 13. Gapminder World: A screenshot from Gapminder World, a Flash-based application for visualizing global socioeconomic indicators over time. This tool has been used extensively by Professor Hans Rosling in explaining the closing gap between rich and poor countries.

- *Italy budget Viz using D^3* is a Web-based visualization that provides access to public data about Italy's administrative expenses during the years 2002-2008. A map-based visualization shows the various regions of the country, providing a color-coded visualization of the levels of spending (between "min" and "max") of the various regions, during each year. Linked visualizations provide bar charts of the levels of 30 spending categories (such as water, agriculture, justice, energy, and more) per each region. Interaction is accomplished by hovering over a year to select it, clicking an industry sector to select it, or hovering over a region to select it. As the visitor makes her/his selections, a separate tooltip display, centered between Italy's map on the left and the bar chart on the right, shows the name of the region and the actual spending value for that region, as well as a comparison to the national average, using a red line.
  Thus, a visitor can effortlessly and effectively navigate across three dimensions of a data cube just by hovering over visual representations of various members (by "member" we mean a record along a dimension in the OLAP terminology).
  The page has been implemented using D^3, and recommends access via either Firefox, Safari, or Opera, but not Internet Explorer [77].
- CNN Economy Tracker — US Bureau of Labor Statistics (BLS) employment visualization. This is an interactive Choropleth map, colored in proportion to the economic variable being explored. Under the "stimulus funds"

tab, color coded variables are “funds awarded,” “funds received,” and “jobs created/saved” for each US state. Additional tabs provide similar visualizations for “foreclosures,” “unemployment,” and “jobs by industry.” One can probe the various states and obtain detailed values of the variables for the probed state, which is colored relative to these variables. The data spans across US states, industry, and time [78].

See Figure 14–17 for additional examples.

Fig. 14. Tableau Public Hurricane Visualization: A visualization of historical hurricane trajectories from Tableau Public. The user interface supports switching between years. The rendering is performed on the server side.

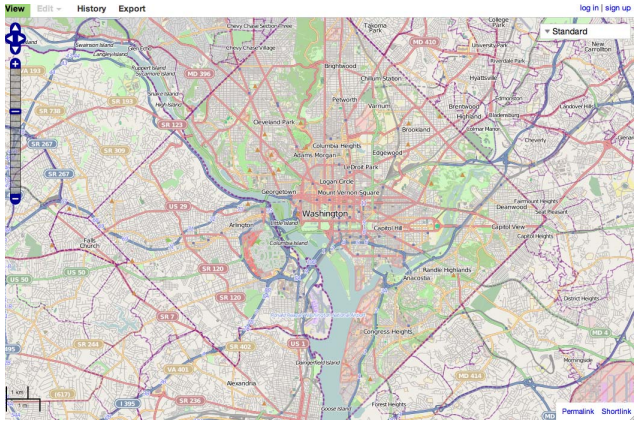

Fig. 15. OpenStreetMap: A screenshot from the OpenStreetMap project showing the detail of its crowdsourced street-level data for Washington DC.

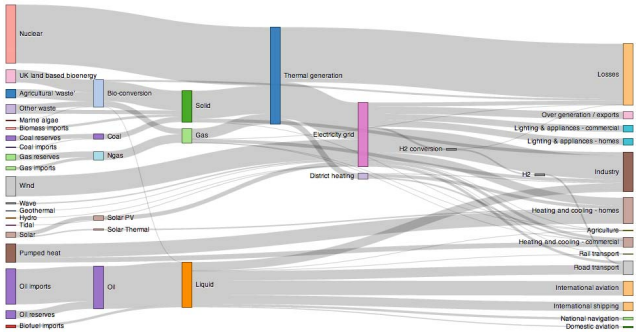

Fig. 16. D^3 Sankey Diagram Example: A Sankey Diagram made using D^3 showing the flow of energy from sources to uses. The visualization shows one possible scenario for the UK in 2050, and was created as part of an emissions reduction initiative.

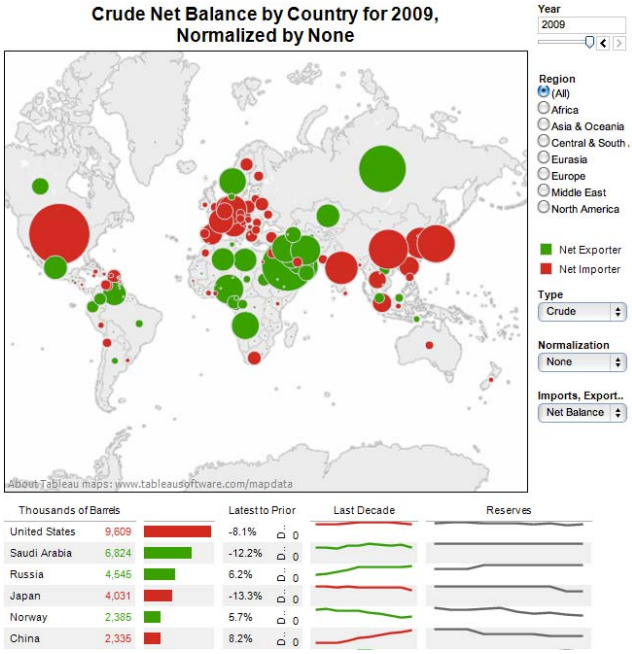

Fig. 17. Tableau Public Oil Visualization: A visualization of net crude oil production (green) and consumption (red) by country from Tableau Public.

XIII. Conclusion

We have posited that computing is going through a paradigm shift: the combined proliferation of cloud-based compute environments and services, and ever more capable mobile devices, such as smart phones and tablets, has led the way towards a new computing environment, in which mobile devices and cloud-based services cooperate to provide the best computing resources. We have predicted that a large majority of all computing will eventually migrate to this model, and we have observed that this migration is already taking place.

In anticipation of this paradigm shift, we have focused on visual computing (defined as the combination of computer graphics, image processing, vision, and visualization) as a

perfect beneficiary of the Mobile+Cloud computing paradigm. As such, we have surveyed fundamental technologies, such as HTML5, HTML5 Canvas, and WebGL, as well as additional technologies, such as 2D and 3D graphics libraries, all of which are catalyzers for this transition. We have provided the main characteristics, features, and capabilities, and have identified strengths, but also potential weaknesses of these technologies.

It is important to remember that this model, these technologies, are all very young, and will continue to evolve and change over the coming months and years. Some will disappear while other will become cornerstones of the future of computing in general and visual computing in particular. Readers will have unique opportunities to shape these trends.

REFERENCES

[1] J. D. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice*, 2nd ed. Reading, MA: Addison Wesley, 1990.

[2] W. Shoaff, "A short history of computer graphics," http://cs.fit.edu/~wds/classes/graphics/History/history/history.html#SECTION00020000000000000000, 30 August 2000.

[3] cfxweb.net, "The evolution of computer graphics," http://www.cfxweb.net/evolution.

[4] I. Sutherland, "Sketchpad: A man-machine graphical communication system," Ph.D. dissertation, Massachusetts Institute of Technology, 1963.

[5] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *j-IBM-SYS-J*, vol. 4, no. 1, pp. 25–30, 1965.

[6] J. Bresenham, "A linear algorithm for incremental digital display of circular arcs," *Commun. ACM*, vol. 20, no. 2, pp. 100–106, Feb. 1977. [Online]. Available: http://doi.acm.org/10.1145/359423.359432

[7] S. A. Coons, "Computer graphics and innovative engineering design—super-sculptor," *DATAMATION*, vol. 12, no. 5, p. 3234, 1966.

[8] ——, "Uses of computers in technology," *SCIENTIFIC AMERICAN*, vol. 215, no. 3, p. 177, 1966.

[9] D. Ahuja and S. A. Coons, "Geometry for construction and display," *IBM SYSTEMS JOURNAL*, no. 3-4, p. 188, 1968.

[10] S. A. Coons, "Modification of shape of piecewise curves," *COMPUTER-AIDED DESIGN*, vol. 9, no. 3, pp. 178–180, 1977.

[11] ——, "Constrained least-squares," *COMPUTERS & GRAPHICS*, vol. 3, no. 1, pp. 43–47, 1978.

[12] T. Sederberg, "BYU Bézier curves," http://www.tsplines.com/resources/class_notes/Bezier_curves.pdf.

[13] P. Bourke, "Bézier surfaces (in 3D)," http://local.wasp.uwa.edu.au/~pbourke/geometry/bezier/index.html, 1996.

[14] A. Appel, "The notion of quantitative invisibility and the machine rendering of solids," in *Proceedings of the 1967 22nd national conference*, ser. ACM '67. New York, NY, USA: ACM, 1967, pp. 387–393. [Online]. Available: http://doi.acm.org/10.1145/800196.806007

[15] ——, "Some techniques for shading machine renderings of solids," in *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, ser. AFIPS '68 (Spring). New York, NY, USA: ACM, 1968, pp. 37–45. [Online]. Available: http://doi.acm.org/10.1145/1468075.1468082

[16] F. C. Crow, "Shadow algorithms for computer graphics," in *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '77. New York, NY, USA: ACM, 1977, pp. 242–248. [Online]. Available: http://doi.acm.org/10.1145/563858.563901

[17] H. Gouraud, "Computer display of curved surfaces," Ph.D. dissertation, 1971.

[18] ——, "Continuous shading of curved surfaces," *IEEE Transactions on Computers*, vol. C-20, no. 6, pp. 623–629, Jun. 1971.

[19] B.-T. Phong, "Illumination for Computer Generated Pictures," *j-CACM*, vol. 18, no. 6, pp. 311–317, 1975.

[20] E. E. Catmull, "A subdivision algorithm for computer display of curved surfaces." Ph.D. dissertation, 1974.

[21] T. Whitted, "An improved illumination model for shaded display," *Commun. ACM*, vol. 23, no. 6, pp. 343–349, Jun. 1980. [Online]. Available: http://doi.acm.org/10.1145/358876.358882

[22] A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models," *Commun. ACM*, vol. 25, no. 6, pp. 371–384, Jun. 1982. [Online]. Available: http://doi.acm.org/10.1145/358523.358553

[23] Pixar, "The RenderMan interface specification," http://renderman.pixar.com/products/rispec/index.htm, November 2005, version 3.2.1.

[24] T. L. Lancaster, "Renderman books & other publications," http://www.renderman.org/RMR/Publications/index.html, July 2008.

[25] G. Anthes, "HTML5 leads a Web revolution," *Communications of the ACM*, 2012.

[26] B. Lawson and R. Sharp, *Introducing HTML5*, 2nd ed. Berkeley, CA: New Riders Press, 2011.

[27] World-Wide Web Consortium, "HTML5—a vocabulary and associated APIs for HTML and XHTML," http://dev.w3.org/html5/spec/Overview.html, May 8 2012, Editor's Draft.

[28] W3Schools, "HTML5 Tutorial," http://www.w3schools.com/html5/default.asp.

[29] Wikipedia, "Immediate mode," http://en.wikipedia.org/wiki/Immediate_mode, July 2011.

[30] D. Geary, *Core HTML5 Canvas: Graphics, Animation, and Game Development*. Prentice Hall, 2012.

[31] R. Miguel and collaborators, "Three.js," http://mrdoob.github.com/three.js/, 2012.

[32] Wikipedia, "Retained mode," http://en.wikipedia.org/wiki/Retained_mode, April 2012.

[33] A. Zakai, "box2d.js: Box2d on the Web is getting faster," http://mozakai.blogspot.com/2012/02/box2djs-box2d-on-web-is-getting-faster.html.

[34] W3Schools, "SVG Tutorial," http://www.w3schools.com/svg/default.asp.

[35] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," pp. 2301–2309, 2011.

[36] World-Wide Web Consortium, "HTML5 specification," http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html, 2012.

[37] "Web worker," http://en.wikipedia.org/wiki/Web_worker, June.

[38] E. Bidelman, "The basics of Web workers," http://www.html5rocks.com/en/tutorials/workers/basics/, May 2011.

[39] I. Hickson, "The WebSocket API," http://dev.w3.org/html5/websockets/, 2012.

[40] "The HTML5 test — how well does your browser support HTML5?" http://html5test.com/.

[41] J. Resig and collaborators, "Processing.js," http://processingjs.org/, 2012.

[42] C. Reas and B. Fry, *Processing: a programming handbook for visual designers and artists*. The MIT Press, 2007.

[43] J. Lehni and J. Puckey, "Paper.js," http://paperjs.org/, 2012.

[44] M. Bostock and J. Heer, "Protovis: A graphical toolkit for visualization," pp. 1121–1128, 2009.

[45] "WebGL and OpenGL differences," http://www.khronos.org/webgl/wiki/WebGL_and_OpenGL_Differences, May 2012.

[46] G. Tavares, "WebGL fundamentals," http://www.html5rocks.com/en/tutorials/webgl/webgl_fundamentals/, February.

[47] "WebGL specification," http://www.khronos.org/registry/webgl/specs/latest/, July 2012.

[48] "WebGL — OpenGL ES 2.0 for the Web," http://www.khronos.org/webgl/, 2012.

[49] Wikipedia, "WebGL," http://en.wikipedia.org/wiki/WebGL, July 2012.

[50] "Physijs: Physics library for three.js," http://chandlerprall.github.com/Physijs/.

[51] "Ammo.js, a port of the Bullet physics engine from C++ to JavaScript," http://syntensity.com/static/ammo.html.

[52] Adobe Systems Inc. and contributers, "PhoneGap," http://phonegap.com/, 2012.

[53] Appcellerator Inc., "Appcellerator Titanium," http://www.appcellerator.com/, 2012.

[54] J. Seidelin, *HTML5 Games: Creating Fun with HTML5, CSS3, and WebGL*. Wiley, 2012.

[55] The Nielsen Company, "Two thirds of new mobile buyers now opting for smartphones," http://blog.nielsen.com/nielsenwire/?p=32494, 2012.

[56] The Android Open Source Project, "Android — the publishing process," http://developer.android.com/tools/publishing/publishing_overview.html, 2012.

[57] R. JR., "Rejected! 10 iPhone Apps That Didn't Make Apple's App Store," http://www.pcworld.com/article/159887/rejected_10_iphone_apps_that_didnt_make_apples_app_store.html, 2012.

[58] Google Inc., "Chrome Web store — overview," https://developers.google.com/chrome/web-store/docs/, 2012.
[59] Zygote Media Group Inc., "Zygote Body," http://www.zygotebody.com/, 2012.
[60] N. Weber, "OpenGL ES 2.0 graphics on Android: Lessons from Google Body," http://code.google.com/p/gdc2011-android-opengl/wiki/TalkTranscript, 2011.
[61] "BioDigital Human," biodigitalhuman.com.
[62] Google Inc., "Build maps for mobile apps," https://developers.google.com/maps/mobile-apps, 2012.
[63] MapBox, "TileMill," http://mapbox.com/tilemill/, 2012.
[64] R. Sharmes, "Designing an OSM Map Style," http://code.flickr.com/blog/2012/07/11/designing-an-osm-map-style/, 2012.
[65] "Tableau Public," http://www.tableausoftware.com/public/, 2012.
[66] "OpenProcessing," http://www.openprocessing.org/.
[67] "Chrome Experiments," http://www.chromeexperiments.com/.
[68] F. Woitzel, "Progressive Julia Fractal," http://www.chromeexperiments.com/detail/progressive-julia-fractal/?f=, August 2011.
[69] "WebGL Experiments," http://www.chromeexperiments.com/webgl/.
[70] R. Cyganiak and A. Jentzsch, "The linking open data cloud diagram," http://richard.cyganiak.de/2007/10/lod/, 2012.
[71] L. Feigenbaum, "The Semantic Web Landscape: A Practical Introduction," http://www.cambridgesemantics.com/semantic-university/the-semantic-web-landscape-a-practical-introduction, 2012.
[72] O. Lassila, R. Swick *et al.*, "Resource description framework (RDF) model and syntax specification," 1998.
[73] W3C, "OWL 2 Web Ontology Language — W3C Recommendation," 2009.
[74] R. Cyganiak and D. Reynolds, "RDF Data Cube Vocabulary specification," 2012.
[75] United Nations, "UN Data — A World of Information," http://data.un.org/, 2012.
[76] "GapMinder World," http://www.gapminder.org/.
[77] "Le spese amministrative in Italia dal 2002 al 2008 (administrative expenses in Italy from 2002 to 2008)," http://www.visup.it/misc/workshop/index.htm.
[78] CNN, "CNN Economy Tracker," http://www.cnn.com/SPECIALS/map.economy/index.html?mapIndex=3&hpt=C2.