The Argon AR Web Browser and Standards-based AR Application Environment

Blair MacIntyre[1] Alex Hill[1] Hafez Rouzati[1] Maribeth Gandy[2] Brian Davidson[3]
[1]Augmented Environments Lab, [2]Interactive Media Technology Center, [3]Research Network Operations Center
Georgia Institute of Technology, Atlanta, GA 30332

Abstract

A common vision of Augmented Reality (AR) is that of a person immersed in a diverse collection of virtual information, superimposed on their view of the world around them. If such a vision is to become reality, an ecosystem for AR must be created that satisfies at least these properties: multiple sources (or channels of interactive information) must be able to be simultaneously displayed and interacted with, channels must be isolated from each other (for security and stability), channel authors must have the flexibility to design the content and interactivity of their channel, and the application must fluidly integrate with the ever-growing cloud of systems and services that define our digital lives.

In this paper, we present the design and implementation of the Argon AR Web Browser and describe our vision of an AR application environment that leverages the WWW ecosystem. We also describe KARML, our extension to KML (the spatial markup language for Google Earth and Maps), that supports the functionality required for mobile AR. We combine KARML with the full range of standard web technologies to create a standards-based web browser for mobile AR. KARML lets users develop 2D and 3D content using existing web technologies and facilitates easy deployment from standard web servers. We highlight a number of projects that have used Argon and point out the ways in which our web-based architecture has made previously impractical AR concepts possible.

Keywords: augmented reality, web-based architecture

Index Terms: H.5.1 [Information Interface and Presentation (e.g., HCI)]: Multimedia Information Systems— Artificial, augmented, and virtual realities; H.5.4 [Information Interface and Presentation (e.g., HCI)]: Hypertext/Hypermedia— Architectures.

1 Introduction

Since *augmented reality* (AR) was first demonstrated by Ivan Sutherland in 1965 [20] the idea has captured researchers imagination. Spurred on by science fiction authors, the term conjures dreams of people immersed in a hybrid physical/virtual world where synthetic content of all kinds is blended with the physical reality around them. AR research picked up in the late 1980s, with various researchers focused on the enabling technologies (e.g., tracking software and hardware, display technology), exploring different application domains (e.g., maintenance [5], medical [1], military [23]), understanding human factors (e.g., user perception of depth [12] or registration error [4]) and creating the authoring tools necessary to support this research and exploration (e.g., DART [13], Studierstube [16], GoblinXNA [14]).

Each of these components is necessary if the dream of immersive AR is to become a reality. However, success in each of these areas is not sufficient; the user experience implied by the visions of AR all share the idea that all AR content is presented in one unified *AR application environment*, regardless of the source of the content. Any AR experience, from the simple to to the complex (e.g., games, training applications, social media, search results, advertising, and playful toys), should always be available within one environment and should be able to be authored and made available by independent developers with no coordination or approval process.

The idea of a single AR environment, in which all AR content is presented, has been proposed multiple times over the past two decades (e.g., [10,15,16,17,19]), and is the (implicit) motivation behind many of the so-called "AR Browsers" appearing in the smartphone marketplace[1]. Unfortunately, none of the proposed (research or commercial) systems comes close to achieving the necessary functionality. Previous research systems have focused on specific research questions (e.g., interaction techniques, col-

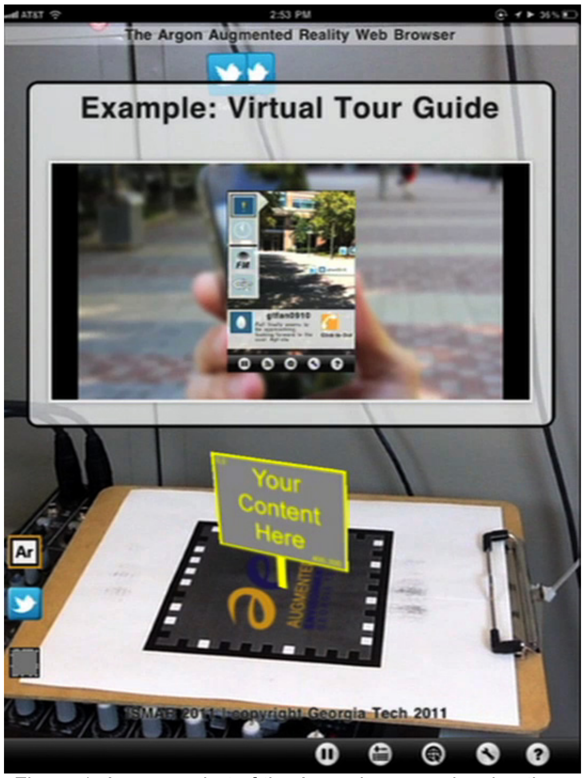
The Argon Augmented Reality Web Browser
Example: Virtual Tour Guide
Your Content Here
ISMAR 2011 - copyright Georgia Tech 2011

Figure 1: A screen shot of the Argon browser showing three simultaneous AR channels (presentation slides with embedded video, live twitter search and marker tracking).

e-mail: {blair, ahill, hafez, mg129, bdavidson}@gatech.edu

laboration, etc.) without worrying if the proposed architecture could be deployed in a practical way. The various "AR Browsers" focus on search and browsing of information snippets, but ignore AR applications that cannot be represented as a collection of "information nuggets" (consider the breadth of AR applications proposed and prototyped over the years; most could not be implemented in one of these "Browsers"). Furthermore, none of these systems addresses the practical issue that individual "AR application" authors may want a high degree of control over the look, feel and interaction of their content, even if it is displayed in parallel with other AR content. Finally, there are a range of practical issues, from "cross-application" security to e-commerce and offline data management to efficiency and scalability concerns that a real system must address.

These concerns are not unique to AR, even though the style of content presentation is unique; re-examining the history of our existing 2D interactive computing systems helps to frame the problem. When 2D and 3D graphical applications began to appear, each application was written to control the entire display. Various SDKs and tools appeared to support application authoring, and researchers and practitioners experimented with a wide range of interaction techniques and metaphors. Akin to the data-centric AR system ideas, pluggable data-centric architectures for 2D content were created and championed (e.g., OpenDoc[4]), driven by the appeal of composable "active objects" rather than monolithic applications. In the end, the application/document model and the desktop metaphor for 2D user interfaces emerged as the dominant approach to sharing graphical displays between multiple applications, and is the foundation on which all modern graphical interfaces are based. The key concept behind the desktop metaphor is the "virtual device" abstraction, where each application is authored as if it has access to the full capabilities of an abstract collection of input and output devices. Users decide which programs are running, how they are arranged and how they interact with them. While this model has its limitations, the reality is that it successfully balances the needs of the application developer, the user, and creators of the underlying systems: the model is simple, and can result in robust, secure and practical systems.

When viewed in this historical context, what is needed for an AR application environment is analogous to the 2D desktop and windowing system. We are not suggesting literally moving 2D windows into the world around us (as done in [6]), but rather the related idea of an ecosystem in which independently created "AR applications" co-exist without needing to know what other AR content is also displayed. The granularity of the content elements (e.g., the windows, menus, palettes, and dialog boxes of the 2D desktop) will evolve over time, and may be different for different applications. Just as early windowing systems, such as the X11 window system, provided core mechanisms but allowed different policies and metaphors to be explored (i.e., through different "Window Managers"), we need a flexible system based on a robust set of policy-agnostic mechanisms. Similarly, we must ensure that the AR content authoring is at a reasonable level of abstraction, such that authors have sufficient control, but are not needlessly tied to a specific platform or hardware.

Over the years, as different ideas and designs for a single AR environment were put forth, mobile hardware technology was not mature enough to support such an environment, nor were there any sufficiently powerful and flexible mobile system architectures on which to base an implementation. As we will illustrate in this paper, the combination of powerful mobile devices and the full featured mobile web addresses these problems, and can serve as the foundation for an AR application environment that moves us one step closer to the dream of immersive AR. Over the past two years, we have designed and built such an environment, including a set of AR-specific web "application" abstractions, and an "AR web browser" supporting them. Argon, the AR web browser, has been freely available for iOS since February 14, 2011, and is starting to be used by researchers and developers around the world.

The overall architecture, called KHARMA (KML/HTML Augmented Reality Mobile Architecture), is based on standard web technologies, whenever possible [8]. We have extended the semantics of KML (the markup language used by Google Earth (GE) and Google Maps) to support the requirements of AR. This extension of KML is called KARML, and lets an author specify where AR content lives in the world. AR applications (called channels) live on standard web servers, and one or more of those channels can be viewed simultaneously in Argon, as shown in Figure 1. Each channel is independent, and can have its own user interface and interactive content.

In this paper, we discuss the motivations behind the design of the system, the specific research contributions of this work, some of the more relevant details of Argon and KARML, and the implementation of Argon on iOS. We present a variety of example channels created by us, our collaborators and other developers, and highlight how they leverage the unique attributes of our platform.

1.1 Background: Deciding to Build on The Mobile Web

This project started in the fall of 2009, when we observed that the development trajectory of modern smart phone hardware and mobile web software would soon make the combination a suitable foundation for a comprehensive AR application environment.

First, it was clear that mobile computing technology was maturing rapidly, and would soon support the necessary system technologies (both hardware and software) for mobile AR. Powerful mobile phones with GPS and orientation sensors had already made a limited form of AR, handheld sensor-based video-see-through augmented reality, practical for commercial developers and accessible to millions of people. While early AR applications for mobile devices still rely almost entirely on the built-in sensors (i.e., GPS, compass, accelerometers and gyroscopes), newer computer vision toolkits, such as Qualcomm's AR SDK[2], are enabling developers to create a more powerful collection of applications that accurately register graphics with the physical world.

Second, we believed that the modern mobile WWW architecture would soon be mature enough to serve as the basis for an AR application environment. What was once exemplified by impoverished WAP browsers[3] had been replaced by mobile browsers with features similar to their desktop counterparts. Mobile web renderers and the corresponding web standards included highly accelerated Javascript and HTML/CSS engines, and will soon include WebGL for arbitrary 3D content, the ability to safely run platform independent native code, and access to hardware such as the camera and the various sensors. Furthermore, a glance at a typical

[1] www.wikitude.org, www.layer.com, www.acrossair.com

[2] developer.qualcomm.com/dev/augmented-reality

[3] www.wapforum.org

[4] en.wikipedia.org/wiki/OpenDoc

computer display shows that even then, many of our tools lived in the web ecosystem, from stores like Amazon to services like Facebook to entire operating environments like Google's ChromeOS. As more of what we do lives in the cloud, a cloud-based ecosystem makes increasing sense.

Apple's implementation of the WebKit 3D extensions in Mobile Safari provided a key starting point for a web-based approach, by allowing any interactive 2D web content to be rendered efficiently in 3D. While 2D-billboards-in-3D is not the ideal solution for all AR applications, the trajectory of web technologies is pointing in the right direction (e.g., a combination of WebGL and native 3D rendering will, in the near future, allow mobile web-based applications like Argon to support full 3D content as well).

Beyond the specifics of software, we do realize that the smartphone (by itself) is not the ideal vehicle for all AR applications, because of its small screen and the need to hold it up to see "through" it. However, when paired with a head-worn display (which a number of display companies are working on), this limitation will cease to be a problem. And the greatest advantage of the mobile phone will continue to hold, it's ubiquity: the best device is the one everyone already has in their pocket.

1.2 Goals

We had three main goals driving our development of Argon. First and foremost, we wanted to create an AR application environment that supports the vision of an immersive AR system: a "window system" for AR. Our motivation to create such an environment is driven by our desire to push AR technology forward; we firmly believe that, unless AR technology is put in the hands of millions of designers, engineers, artists and entrepreneurs around the world, we will not fully understand where the "killer apps" might lie, and what the true requirements of the technology are.

Our main goal was tempered by a second goal: to build on existing mobile technology as much as possible. We did not want to just leverage web technologies (for example, integrating a JavaScript/HTML engine into an AR system); we wanted to integrate with the web ecosystem as tightly as we could. As AR researchers, we often forget that AR is just one technology among the many that are needed to solve real problems. Some non-trivial mobile AR applications will be complex, involving a spectrum of 2D and 3D content and interactions, and will need to be networked and distributed. The enormous benefits in terms of authoring, deployment, access to web services and existing content that are gained by integrating with the web outweigh the limitations, for many possible AR applications.

Our final goal was to create an ecosystem that supports easy and sophisticated authoring of applications; this again points to the web as an ideal platform. KARML is based upon KML, along with the full collection of contemporary Web 2.0 standards (HTML, CSS, JavaScript, etc). While KARML extends the KML language to better support handheld AR, we were careful to support traditional KML (most KML files will display in a predictable way in Argon). Conversely, even complex combinations of HTML and JavaScript can be used in Argon with minimal changes. Taken together, experienced web developers can use tools and techniques with which they are already very familiar (e.g. HTML5, CSS, PHP, JavaScript, Google Earth, DreamWeaver, Yahoo Pipes, etc.) to create their mobile AR applications, which allows for existing web content to be repurposed with ease. Furthermore, AR applications can be hosted on the same web servers (since Argon uses the standard HTTP protocol), and even share URLs with traditional web browsers (since Argon's browser ID string can be used by the server to identify requests from Argon and respond appropriately). Together, these dramatically simplify distribution and management of content.

2 Contributions

In this paper, we present the Argon AR web browser, the KARML markup language and their integration with the web. The main contributions of this paper and project are summarized here.

Demonstration that the web is a viable mobile AR platform. We do not claim Argon is, or will be, the ideal AR platform for all mobile AR applications. However, Argon clearly demonstrates that mobile web technologies are a viable basis for a wide range of mobile AR applications. Argon currently supports sensor-based AR and marker-based AR using 2D-billboards-in-3D content; it will soon support much more complex computer vision-based tracking and full 3D content.

The KARML specification. The variation of KML we have defined is a living example of a markup language for AR content. The specification is far more comprehensive than previous efforts.

The Argon multi-channel AR architecture. Argon supports multiple independently authored, but simultaneously displayed, channels of AR content. Each is fully scriptable, interactive and can define its own 2D/3D interface. By layering multiple transparent WebKit instances on top of each other, each channel is sandboxed in its own JavaScript context (for security and robustness). Argon provides channels with notification that their channel has gained or lost focus (so they can change appearance or behaviour when not in front), a shared location across channels (even when one channel "moves" the browser to a synthetic location), and access to *GeoSpots* (geo-located panoramic images that can be included in channels and used in place of live video and GPS location).

Demonstration that the web-centric approach is powerful. Beyond the web being viable for AR, by embracing the web we enable previously impractical or impossible AR applications to be created and deployed. Simple applications can be deployed rapidly (in hours, not weeks or months). Complex applications, involving cloud services, asynchronous agents, content filtering and so on, are tractable. Beyond this, by leveraging the web we don't have to reinvent the wheel with respect to content creation: content elements can be authored in tools such as Google Earth or Dreamweaver, and assembled as appropriate.

3 Related Work

Since Vannevar Bush first described his hypothetical "memex" device researchers have been seeking new ways to browse and create connections between all types of information [3]. From the beginning of AR research, systems were created that took data with spatial meaning and attached it to the real-world objects and locations. From merging ultrasound imagery with the patient [1] to providing operating instructions for a printer visually registered with the physical components [5], early AR systems demonstrated the power of linking information to relevant spatial contexts. Early outdoor AR systems expanded the range of scenarios to include geospatial scale content; the Touring Machine [7] and MARS [9] supported linking from 3D icons to the 2D web, and TINMITH [15] explored the potential of in-situ AR editing.

Many of these early systems could be recreated on modern smartphones, and informed the requirements for our work.

Many authoring tools, of different forms, have been created. Tools such as Studierstube combined software abstraction layers for AR infrastructure and technologies into a framework usable via code or GUI front-ends [16], with similar motivations to our work but before the technical ecosystem was sufficiently evolved. In contrast, DART added AR concepts to an existing high level media authoring tool, Adobe Director [13]. A variety of projects, like Goblin [14], focused on adding AR technology to game engines. Other researchers focused on creating simple authoring environments for a specific application domain (e.g., Amire, for assembly tasks [24]). We expect that systems like Amire could be implemented with Argon.

The ARToolkit [2], and the more recent FLARToolkit, provide marker tracking in C++ and Adobe Flash, respectively. The appeal of FLARToolkit is that, despite the limitations of being locked inside the Flash engine, it make it trivial for developers to author and distribute their applications, something that previously has been a major hurdle. Argon takes the next step beyond systems such as FLARToolkit, by supporting a wider variety of sophisticated web applications. Others have attempted to create a language for AR content and applications (e.g., Augmented Presentation and Interaction Language (APRIL) [11]), but without integrating with the web, have had little success.

"Windows on the World" incorporated an existing 2D window system within a 3D virtual world [6]. This system took X11 windows from the desktop and placed them into the physical world, but did not address authoring or real use. More relevant to this project are the WorldBoard and RWWW projects. WorldBoard envisioned a planetary augmented reality system that would provide innovative ways of associating information with places, with ideas for scalability, global access and so forth [19]. The Real World Wide Web (RWWW) project was our first attempt at creating a system like Argon [10], but the web was not mature enough at the time to serve as a solid foundation for the work. More recently, Schmalstieg et al have discussed leveraging the web ecosystem for AR [17] and they have presented some similar arguments (in terms of availability, scalability, etc.) in support of this general approach. They do not go as far as we do in proposing a system that not only interoperates with the web, but uses web technologies to actually realize rendering and interactivity elements. Nor do they build a complete prototype to test the idea.

In the last three years, a crop of commercially available "AR browsers[1]" have appeared, aimed at outdoor information browsing and retrieval. Each of these provides different degrees of openness to end-user content, but nothing on the scale or capability of even the early web. Junaio 2.0 introduces "indoor GPS" through the concept of LLA (longitude, latitude, altitude) markers. Like our GeoSpots, they provide precise location when GPS is inadequate. However, by encoding the location in physical form, rather than using indirect references, they have limited flexibility.

KARML is not the first attempt to extend KML for AR. ARML[4] extended KML with AR specific structures. These extensions were more modest, and focused on adding markup extensions to support specific browser features, such as "wikitude:thumbnail" and "ar:provider". The KARML extension is more comprehensive, and focuses on extending existing KML features and semantics while avoiding application specific additions wherever possible.

[4] www.openarml.org

[5] www.web3D.org

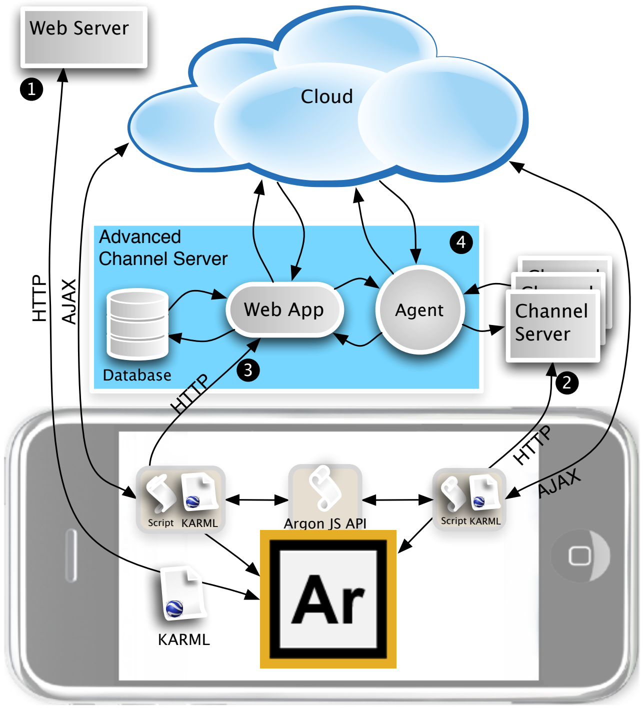

Figure 2: Argon leverages multiple web architecture models.

A tradition of abstraction and open tools define many technology advances in the field of 3D, AR, and the web. It is clear that technologies must be made accessible to be adopted. Components that are typically hard to work with or understand must be made easy. The Web3D standards of Virtual Reality Modeling Language VRML 97[5] were an attempt to make 3D content ubiquitous on the web. While later the Virtual-Reality Peripheral Network (VRPN) provides a device-independent and network-transparent interface to virtual-reality peripherals [21].

One feature of Argon is the ability to use panoramic images in the background instead of live video. Commercial systems such as Google StreetView as well as Microsoft's Photosynth and Bing Maps support the creation and navigation of 3D panoramic scenes augmented with geospatial data [18]. We have integrated this concept into KARML and Argon to support the authoring of mixed reality experiences that leverage the live channel data in various ways both at the physical site and for remote viewing. We have developed a web service that allows users to submit panoramas to the system that can be utilized by channel authors via an open API. Our plan is to eventually leverage the panorama service for both display and tracking. Wagner et al developed a method for the real-time creation and tracking of panoramic maps on mobile phones and authoring of experiences that use them. They note that this method can also be used in the creation of panoramic images for offline browsing, for visual enhancements through environment mapping, as well as standard tracking [22].

4 ARCHITECTURE

In this section, we discuss three main architectural components: how Argon integrates with the web, the internal architecture of Argon, and the KARML markup language. The intent of this discussion is to provide the essential details of what we did, both the unique features and the key engineering decisions.

4.1 Argon and The Web Architecture

As we discussed in Section 1, a major design goal of this project is to take advantage of web technology by integrating as tightly as we can into the web. Figure 2 illustrates a spectrum of web programming models that we have leveraged with Argon through this integration. While these same models are commonly used for mobile web development, recognizing their value for the creation of AR applications represents a non-trivial shift in AR application design to a methodology that fully leverages existing distributed computing paradigms. Both the high-level architecture and Argon were designed with the goal of enabling the entire spectrum of web architectures. The example projects presented in later sections embrace one or more of the models depicted in Figure 2.

1) Static KARML/XML. This approach represents the simplest model for serving content to users. Static files are hosted on a web server and requested by a specific URI. All the content elements are contained within the returned document and referenced resources are resolved by the browser without requiring explicit management by the content author, just as with traditional HTML content.

2) KARML + AJAX + Client Side Processing. In this approach, the returned document will include a portion of the content or user interface elements used by the channel and a collection of scripts that use AJAX techniques to make requests to 3rd party data sources. Using the Argon JavaScript API, content elements are instantiated with the returned JSON or XML data. The client side scripts may also contain custom layout and user interaction code provided by a channel author.

3) Web Application With Dynamically Generated KARML. Web applications dynamically generate content similar to that discussed in 1 & 2. The web application keeps track of user sessions and sends updates either through the standard KML NetworkLink mechanism or by responding to AJAX requests.

4) Server Side Aggregation & Processing. An "advanced channel server" communicates with 3rd party data sources and/or other channel servers on the client's behalf, in a effort to provide a maximum level of server side processing. This architecture can support aggregation based on the user's preferences, even while the user may not be running the browser application or have the client channel loaded. This configuration acts as an intelligent agent, and we envision that this offers an environment where additional processing may take place such as image/content analysis or computation of complex layout/filtering/clustering algorithms. As mentioned in the introduction, the ability to perform these computationally intensive tasks in the cloud allows authors to create experiences that would otherwise require too much computation on the client device, too much data communication, or would require the user to run the browser more than they otherwise would want to.

4.2 Application Architecture

The current implementation of Argon has been built on the iOS platform and is deployed on the iPhone4 and iPad2 devices running iOS version 4.2 and above. The application features a hybrid architecture wherein portions of the application are implemented in native code (in particular, Objective-C and C on iOS) and exposed to content developers through custom bindings to the embedded JavaScript interpreter.

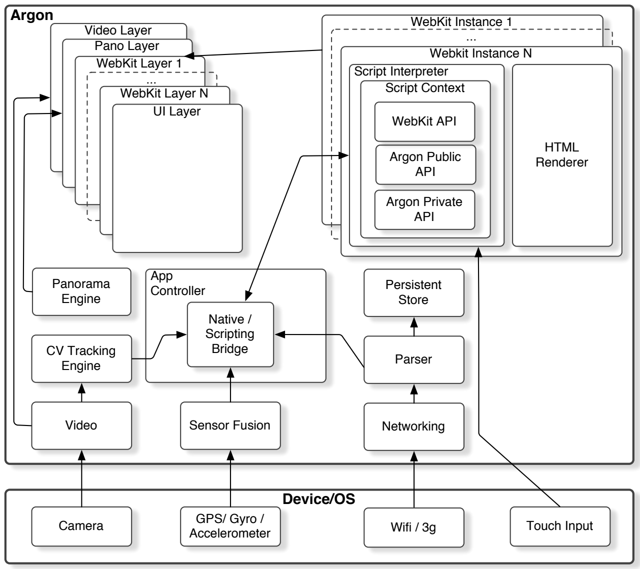

Figure 3: The internal architecture of Argon consists of multiple WebKit instances that communicate via an interpreted bridge.

Figure 3 illustrates the data flow between the components of the application as well as illustrating the layering of the user interface, WebKit view-layers, video layer, and panorama layer.

4.2.1 WebKit

At the time the project was started, iOS was the only mobile platform with an efficient implementation of WebKit that featured hardware-accelerated graphics support and support for CSS3 3D transforms. At the time of writing, iOS remains the only mobile platform that supports CSS3 3D transforms and this is a requirement for creating the HTML scene graph into which content elements can be placed and pushed out into the world.

Excluding panoramic content, all content for a given channel is rendered in a single WebKit instance which consists of a view, HTML renderer, and scripting context. Scripts associated with a channel are sandboxed in a manner that mirrors tabs in a desktop browser. Multi-channel functionality is realized by layering multiple overlapping transparent WebKit views/instances on top of the video and panorama views and behind the application user interface.

4.2.2 Private & Public JavaScript APIs

On the scripting side, functionality is divided between Private and Public APIs. As the name suggests, the Private API is not meant to be used by content developers.

The Private and Public JavaScript APIs act as connection points between native code and interpreted code. The Public API also provides a KML DOM whereby content authors may access and modify KML nodes that are implemented as JavaScript object prototypes. Authors use the KML DOM to dynamically instantiate KML elements and add them to the scene.

The main function of the Private API is constructing and maintaining the HTML DOM data structures for KML objects and responding to messages from native code. An example of this messaging interplay is the transmission and use of device orientation data. The application uses the iOS device APIs to obtain and fuse the raw sensor data. The combined data is sent over the

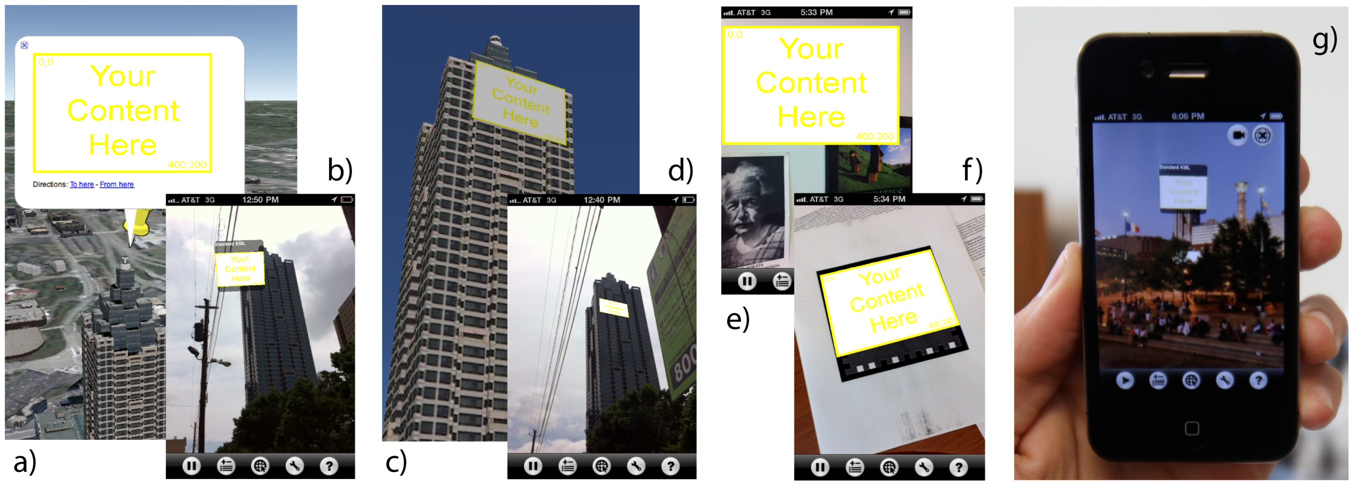

Figure 4: Examples of placing HTML code in standard feature balloons in a) GE and b) the Argon browser. The Balloon element is modeled after the c) GE Model element and d) allows accurate position, orientation and scale in Argon. The e) Overlay element positions content relative to the display and the f) Tracker relative to typical AR markers. Users can override the global position of the browser g) through GeoSpots.

bridge using pre-determined message structures. In the case of device orientation, the message consists of a transformation matrix. The Private API uses this information to transform the scene graph appropriately.

4.2.3 Native/Interpreted Bridge

Communication between native code and interpreted code is implemented through a Native/Interpreted Bridge. The WebKit Event Source API (part of the HTML5 specification) is used for sending high frequency updates (such as orientation and marker tracking) from native code to interpreted code. The Event Source is a string/message based API designed for high frequency uni-directional in-browser updates. Complementary methods and functions exist in the Argon Private JavaScript API that respond to the messages received from native code appropriately and/or notify content authors that various events have occurred.

Method calls from interpreted to native code are achieved by utilizing a URL interception scheme that leverages the fact that the iOS URL loading system lets an application inspect a given URL load request and decide whether to proceed with the load or react to specially encoded URLs in some other fashion. In this scenario, URLs of the form

kharma://Class.Method/arguments

describe a call to the *Method* of *Class* with the provided *arguments*. When the application encounters a URL of this type it sends a message to the appropriate class to call the desired method with the specified arguments. Content authors do not call the native classes directly. Instead, they call regular JavaScript methods exposed through the Argon Public JavaScript API.

4.3 KARML: KML AR Markup Language

The primary purpose of the KARML markup is to act as a binding between the presentation content and locations in the physical world. Using KML was attractive to us for a number of reasons. First, it is broadly used not only by Google Maps and Google Earth but also as an import and export format by numerous Geographic Information Systems (GIS). Secondly, the Google Earth application is already in some ways a demonstration of a truly global Virtual Reality (VR) system. Finally, standard KML already supports the binding of HTML 2D and COLLADA 3D presentation content to physical locations. Motivated by these traits, we developed the KARML extension in an attempt to re-conceive the existing markup in the context of augmented reality use.

Standard KML already supports the inclusion of arbitrary HTML content into the description element of geolocated features called Placemarks. In what has become a widely used technique, Placemarks generate geolocated labels and icons which, when selected, reveal descriptive information balloons. The following KML example markup demonstrates placing a single image in a balloon, the result of which can be seen Figure 4a:

```
1  <Placemark id="myPlacemark">
2      <name>Standard KML</name>
3      <gx:balloonVisibility>1</gx:balloonVisibility>
4      <description><![CDATA[<img style="width:300;height:200"
5          src="http://argonapps.gatech.edu/your_content_here.png"/>]]>
6      </description>
7      <Point>
8          <altitudeMode>absolute</altitudeMode>
9          <coordinates>-84.3866,33.7627,570</coordinates>
10     </Point>
11 </Placemark>
```

Figure 4b shows how this same markup is rendered in Argon, where we attempt to render standard KML faithfully. Neither the GE application nor KML provide a means to remove the framed balloon decoration. The KARML extension adds a *displayMode* enumerator that indicates balloon HTML content should be rendered undecorated. Replacing line 1 in the above markup with the following markup leverages this feature:

```
<Style id="undecorated_style">
    <BalloonStyle>
        <karml:displayMode>undecorated</karml:displayMode>
    </BalloonStyle>
</Style>
<Placemark id="myPlacemark">
    <styleUrl>#undecorated_style</styleUrl>
```

By default, feature balloons are oriented towards the viewer and scaled relatively in depth. A limitation of standard KML is that placemarks can only be given a geospatial translation using the Point element. The KARML extension adds a new Balloon element modeled after the existing KML Model element to add control for the location, orientation and scaling of balloon content. The KARML *orientationMode* and *scaleMode* elements let the user toggle billboarding and relative scaling modes respectively. Adding the following markup in place of or in addition to the

Point element positions the same HTML content at a fixed location, orientation and scale:

```
<karml:Balloon>
    <altitudeMode>absolute</altitudeMode>
    <Location>
        <longitude>-84.386935</longitude>
        <latitude>33.762888</latitude>
        <altitude>514</altitude>
    </Location>
    <Orientation>
        <heading>-90.0</heading>
    </Orientation>
    <Scale>
        <x>9.186</x>
        <y>9.186</y>
        <z>9.186</z>
    </Scale>
</karml:Balloon>
```

In the Argon browser, each pixel of HTML content equates to 1 centimeter in the real world. In Google Earth, we use a template COLLADA model (Figure 4c) to help position content in the real world. Figure 4d illustrates how the above markup appears in the Argon browser. Another limitation of standard KML is that all latitudes and longitudes are absolute references to degree coordinates. Any practical AR application is likely to benefit from having both hierarchical frames of reference and alternate units of measurement. The KARML extension adds a *locationMode* which enumerates "fixed" and "relative" modes. Replacing the Balloon element in the markup above with the following markup positions the HTML content relative to another KML feature:

```
<karml:Balloon>
    <karml:locationMode
        targetId="#otherPlacemark">relative</karml:locationMode>
    <Location>
        <karml:longitude units="meters">6.0</karml:longitude>
        <latitude>0.0</latitude>
        <altitude>0.0</altitude>
    </Location>
</karml:Balloon>
```

In the above markup, the Location element positions the balloon 6.0 meters north of a KML feature in the same document named "otherPlacemark". This fragment reference could instead point to content in another KML file currently loaded by the Argon browser. Argon supports several built in references including "#user" (the default) and "#display". Positioning content relative to the display is functionally equivalent to the following markup which uses the KML ScreenOverlay element to position arbitrary HTML content on the display screen (Figure 4e):

```
<ScreenOverlay>
    <name>Overlay HTML</name>
    <description><![CDATA[<img style="width:300;height:200"
        src="http://argonapps.gatech.edu/your_content_here.png"/>]]>
    </description>
    <screenXY x="0.0" y="1.0" xunits="fraction" yunits="fraction"/>
    <styleUrl>#undecorated_style</styleUrl>
</ScreenOverlay>
```

It is also our goal to include other sources of position information such as fiducial markers, Natural Feature Tracking (NFT) and peripheral devices through libraries such as VRPN. An upcoming release of Argon will allow using the following markup in place of the Balloon element to position the same HTML content on typical AR markers (Figure 4f):

```
<karml:Tracker>
    <karml:trackerDevice>stbTracker</karml:trackerDevice>
    <karml:trackerDescription>
        <stbtracker:type>simpleid</stbtracker:type>
        <stbtracker:id>31</stbtracker:id>
    </karml:trackerDescription>
</karml:Tracker>
```

Because Argon adds HTML content to each WebView dynamically, the normal document initialization often does not work as expected. The following markup demonstrates how to assign an initialization function by adding JavaScript into the description content:

```
<description><![CDATA[
<script language="text/javascript">
    function myPlacemark_init() {
        $(document).bind('focusChanged',function(event) {
            var placemark = KHARMA.getKMLElementById('myPlacemark');
            placemark.visbility = KHARMA.getFocus();
            placemark.render; }
    }
    </script>
    <img style="width:300;height:200"
        src="http://argonapps.gatech.edu/your_content_here.png"/>]]>
</description>
```

The above markup binds to a *focusChanged* event in order to call into the Argon Public API, find its associated KML object and set its visibility to the channel focus state. This has the effect of making the feature invisible when the channel is out of focus. In contrast to how it is implemented in the Google Earth application, all KARML content contained within a single Argon channel shares the same HTML DOM and CSS/JavaScript context.

4.4 GeoSpots

The GPS sensors currently in use by mobile devices are heavily filtered and frequently only accurate to within 10 meters. This low accuracy means that objects depicted on the phone in front of the user can easily be actually behind them, effectively limiting the range within which those augmentations can be delivered. The Argon browser lets users manually override the reported tracking of the device by physically aligning themselves at pre-surveyed locations nearby called GeoSpots. The KML standard and Google Earth application use the Camera and LookAt elements to establish viewing locations in the virtual world. In KARML and the Argon browser, we re-appropriate the KML standard by denoting any features that have a Camera element as GeoSpots. In addition to improving tracking accuracy, GeoSpots allow Argon to report an improved accuracy range to the channel so that content authors can respond in kind. Beyond simply manipulating the range of objects within view, increased accuracy may also motivate changes in visual representation (i.e. from labels/icons to more detailed content).

When available, we also go one step further and let the user replace the video at GeoSpot locations with a panoramic image that changes with the orientation of the device (Figure 4g). Although the orientation sensor continues to determine the background viewed within, the relationship between that background and augmentations in the browser remains registered and stable. If the panoramas are an accurate representation of the GeoSpot location, this technique effectively eliminates any error in orientation accuracy. The use of panoramic backdrops not only increases the in situ options for viewing AR content but also greatly expands the potential audience for that content.

5 Illustrative Examples

A number of applications developed by ourselves, groups within Georgia Tech and outside groups have resulted in a rich set of examples that illustrate of the viability of Argon as an AR development platform. In this section we describe several of these projects and highlight how each leverages the unique attributes that Argon's web-centric models have to offer.

5.1 Server-less AR Mashups

This example demonstrates how the default Argon rendering of standard KML lets users create geospatial AR mashups from serv-

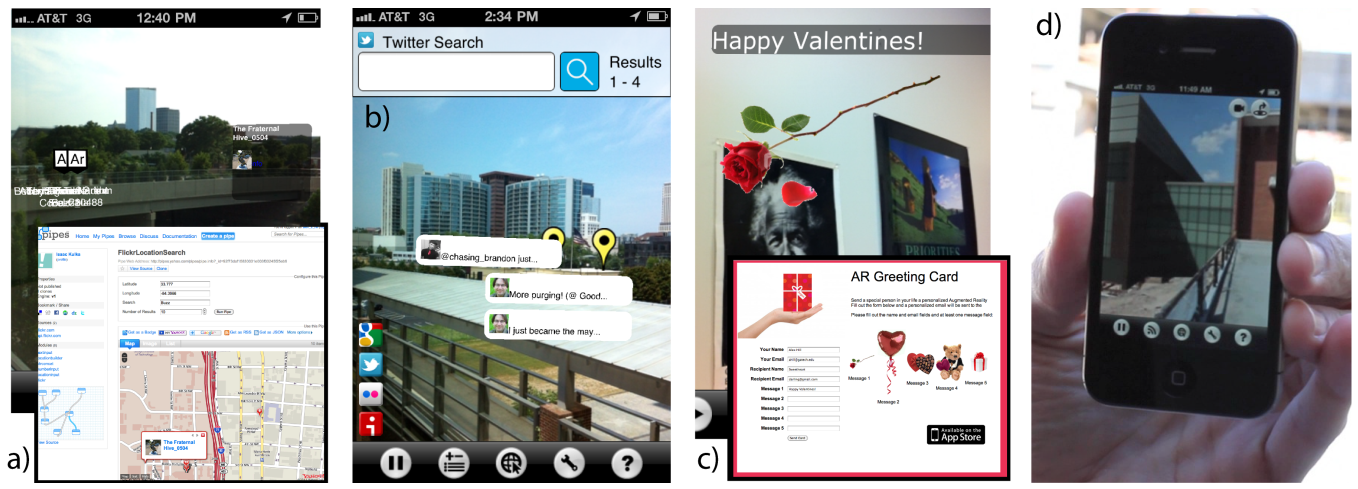

Figure 5. Examples illustrating a) server-less AR mashups using Yahoo Pipes, b) webservice-based dynamic creation of content with channel focus management, c) rapid development of server-based AR content and d) using regions and GeoSpots to manage tracking inaccuracy.

ices like Yahoo Pipes. Yahoo Pipes lets users create composite webservices in a drag-and-drop interface and retrieve those results as a map, JSON or KML. This allows anyone to create mashups of web content and deliver them to an Argon browser without hosting or writing any individual markup. Any Yahoo Pipe can be called by entering its URL into the Argon address bar along with parameters indicating it should return the results as KML (Figure 5a). The results returned by Yahoo create a new channel that renders placemarks as icons that can be expanded into balloons and brought to fill the HUD through a series of clicks.

5.2 Webservice-based Searches

Four example channels demonstrate how webservice-based AR searches can be implemented in Argon in as little as fifty lines of HTML and JavaScript code. Like the other three similar searches, the Twitter search channel places an input box in an overlay and uses AJAX techniques to call the Twitter webservice and return JSON data (Figure 5b). The resulting code is almost identical to similar code executed in desktop browsers except that the JavaScript uses the Argon Public API to dynamically create placemarks. These searches also illustrate how channels can register for application events and change their state when focus shifts from one channel to another. When running multiple active channels, a single channel remains in focus at any one time. Tapping on content in an out-of-focus channel changes focus (analogous to the desktop). The search examples register for *focusChanged* events in JavaScript in order to hide their respective search box and minimize placemarks to labels and icons when out of focus. Each search channel has an overlay image icon along the left of the screen to facilitate switching focus when no content is visible.

5.3 Rapid Server-based AR Development

The AR Greeting Card application (Figure 5c) was developed in about 16 man-hours over the weekend prior to the February 14th debut of Argon in the iTunes store. It consists of a single MySQL table and two PHP scripts. The webpage script presents a form, populates the table with a unique user ID plus desired greeting messages and sends an e-mail to the recipient with a link to a second script. This second PHP script, instead of returning HTML, sets the content type to KML and returns KARML specific to the passed in ID parameter. When Argon is installed on the iPhone or iPad, clicking on a link that uses the *kharma* scheme launches Argon and loads the KML generated by the URL. In this example, the recipient can click on images positioned relative to themselves to reveal a sequence of up to five messages.

5.4 Region Monitoring and GeoSpot Tracking Override

This example illustrates the use of KML region monitoring and GeoSpots to manage the presentational aspects of AR content. The Clough Undergraduate Learning Center is a new building under construction on the Georgia Tech campus. Regions attached to KML placemarks generate *regionChanged* events when a boundary-crossing event occurs. When inside a region surrounding the construction site, a billboarded placemark over the site instructs the user that they can view a pre-visualization of the new building from one of two nearby GeoSpots (Figure 5d).

Bringing up the Argon map displays nearby GeoSpots along with a detail that includes a textual description and an image of exactly where to stand. By "going to" a GeoSpot, the user overrides the GPS location for all active channels, automatically generating a new *locationChanged* event in each and modifying the associated location accuracy. When horizontal accuracy drops to within a threshold, the billboarded message is replaced with a rendering of the new building (created by members of the Georgia Tech Imagine Lab in the School of Architecture) from the GeoSpot location. Clicking on different parts of the building brings up detail renderings. The detail renderings and their associated interaction were developed in an HTML browser by re-appropriating existing online content and then pasted as a whole into separate KML placemark descriptions. Given the inaccuracy of magnetic compasses, there is often mis-registration between the rendering and surrounding buildings. When the user switches to a panoramic background at the GeoSpot, an *orientationChanged* event is generated, the heading accuracy drops below a threshold and the rendering of the building changes to one that is cropped to better represent its relationship to the surroundings.

5.5 Rapid AR Development Leveraging Existing Tools

Several projects illustrate how HTML, CSS, JavaScript and PHP skill sets can be leveraged to create AR content in Argon. The 22ndFloor Observation Deck demo was created by Engauge Interactive Atlanta (Figure 6a) by re-using existing material in a re-

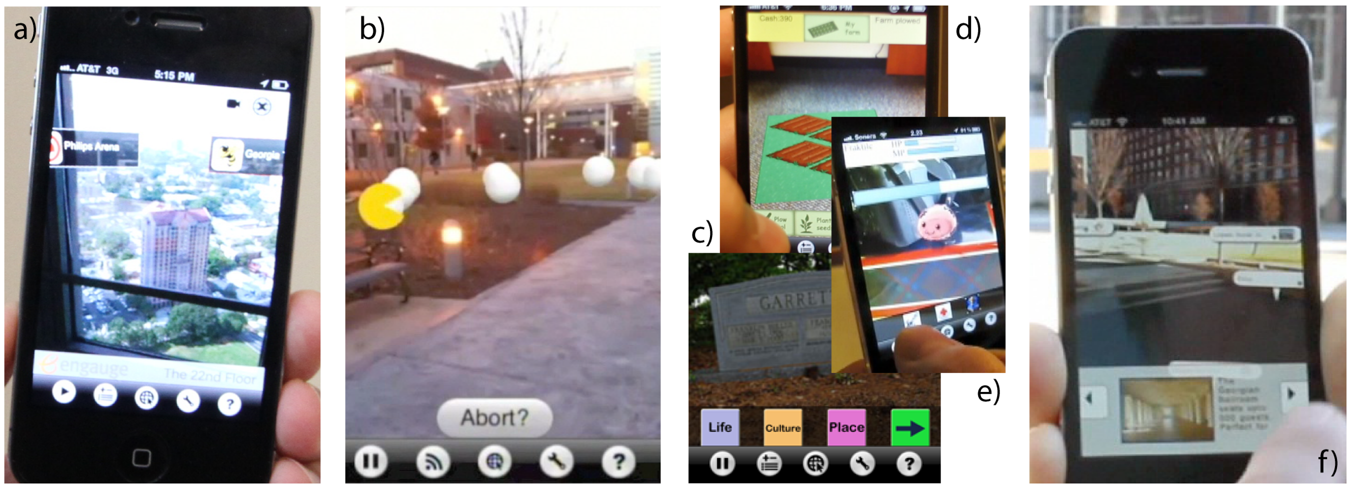

Figure 6: Examples including a) the re-appropriation of existing web content by Engauge Interactive, b) the Dotman's Revenge game by CS design students, c) the Oakland Experience at Oakland Cemetery, d) ARboretum and e) Poring AR by students from Alto University. The Virtual Tour Guide application combines server and client side content filtering with dynamic placement of balloon labels to avoid overlap.

ported man-hour investment of about 8 hours. The main application development, including reading of a JSON database, CSS styling and associated image galleries was done primarily in a desktop browser environment. Of over 400 lines of markup, only about forty lines are KML (a KML ScreenOverlay for application code plus image galleries and a KML PhotoOverlay for the GeoSpot) and 10 lines are specific calls to the Argon Public API to create placemarks and automatically move to the GeoSpot. The panorama was created by using the Photosynth application on the iPhone 4 and uploading it to a conversion utility on our website.

A four-student senior CS design project created an Argon-based game, Dotman's Revenge (Figure 6b), over the course of a semester that features multiple maps, leaderboards and fully realized game-play. The game characters consisted of 2D billboarded images of white pellets and a yellow protagonist. The application logic for the game and PHP-driven scoreboard was developed primarily in a desktop browser environment. Of over 1200 lines of PHP and HTML markup, less than 100 are KML and less than 200 lines of JavaScript were specific to the dynamic creation and deletion of placemark objects.

5.6 Blending of 2D Interfaces and AR Content

Several projects illustrate how projects based primarily on 2D content can incorporate AR aspects using Argon. The Oakland Experience (Figure 6c) is the continuation of an ongoing project based on the narratives of residents at the historic Oakland Cemetery in Atlanta. The application is primarily a linear tour of grave sites at which different audio voiceovers can be selected. The JavaScript and HTML used by the Digital Media (DM) students developing the project was not a significant departure from creating the application in a mobile browser (Argon did provide sound functionality not available at that time in Mobile Safari). The immediate benefit of using Argon was that the students could easily incorporate panoramic backgrounds they photographed and stitched at each grave location on the tour to provide an engrossing experience for offline users. Recently, the students have begun adding AR features to the tour including visual representations of ghosts at the various grave sites.

The multi-user ARboretum application lets users plow, plant and sell crops (Figure 6d). It was developed during a single semester by a DM student with no prior 3D, AR or server-based experience. The majority of the application consists of client-side application logic and HUD content. Logic that updates and queries user state happens after each UI action using a single table MySQL database and AJAX technique.

Another application, Poring AR, was developed during a 6-week class taught by one of our colleagues while at Alto University in Finland (Figure 6e). The application, about maintaining the health of virtual creatures called Poring, consists of rich HUD-based interactions and primarily uses AR concepts to manage the location of the Poring in the game-space. Our colleague remarked that this was likely the most fully realized AR application he had witnessed from students in such a short timespan.

5.7 Client-Server Content and Layout Management

Argon facilitates applications that use a combination of client and server-based interactivity and filtering of data. In collaboration with the Georgia Tech Research Network Operations Center (RNOC), the Virtual Tour Guide (VTG) aggregates social media content such as Tweets, Flickr images, YouTube videos through a proxy server into a personalized experience based on a priori social media affiliations (i.e. Facebook "likes") (Figure 6f). The VTG application uses a server-side geospatial database and current user position to fill 8 bins of orientation space around the user with prioritized content. The application uses regular server polling and the KML NetworkLink Updates scheme to dynamically create, delete and modify placemarks around the user. This polling scheme is combined with user-initiated keyword filtering through the client interface (i.e. "contacts", "sports").

In an effort to create a single application experience, the VTG application hides the standard Argon user interface and manages activities such as moving to GeoSpots using Argon Public API calls. The application also does its own display management of placemark balloons. To avoid the overlapping of placemark labels, client-side CSS and JavaScript dynamically manage the apparent position of labels with leader-lines to their actual position. A priority assigned to each placemark delivered from the server is used in a heuristic that dynamically moves the four highest ranked labels towards the four corners of the screen.

6 Conclusions and Future Work

In this paper we have demonstrated that web technologies present a viable and powerful solution for creating mobile augmented reality applications using existing web standards. We described the software architecture of the Argon AR web browser and how our implementation leverages the existing WWW ecosystem to provide an application environment for AR that allows for multiple channels to be viewed simultaneously, bringing us one step closer to the vision of immersive AR. We described our extensions to KML in the form of KARML and provided details and examples of how we have re-appropriated KML for AR applications. We described a number of past and current projects and highlighted the salient aspects of each project with respect to both Argon and the vision of an AR application environment.

In the coming months, we plan to further develop Argon to add new features including full 3D model rendering, support for other markup languages (e.g., GML), natural feature tracking, protocols for inter-channel communication, space management and layout behaviors and abstractions, greater support for use of tracking data across independent channels without prior coordination, the ability to capture and upload images and video, manual control of view orientation and pinch-to-zoom capabilities, expanded client API, online authoring tools, and support for desktop & other mobile platforms.

Acknowledgements

We would like to acknowledge all the people who have contributed to this project, especially the many students at Georgia Tech who have used the browser during the year leading up to its release. We would like to thank Jay Bolter, Matt Sanders, Russ Clark, Jeff Evans, Elizabeth Mynatt and Mark Billinghurst for supporting this project in a myriad of ways, most especially putting their own classes and projects on the line by relying on this vision of a web-based AR environment. This project has been primarily supported by the Alcatel-Lucent University Innovations Program, but also by Motorola, Turner Broadcasting, and the Georgia Tech GVU Center and Institute for People and Technology.

References

[1] Michael Bajura, Henry Fuchs and Ryutarou Ohbuchi, Merging virtual objects with the real world: seeing ultrasound imagery within the patient, *ACM SIGGRAPH Computer Graphics*, v.26 n.2, p.203-210, July 1992.

[2] M. Billinghurst and A. Cockburn, Eds. ACM International Conference Proceeding Series, vol. 104. Australian Computer Society, Darlinghurst, Australia, 79-88.

[3] V. Bush. As We May Think. Atlantic Monthly, (July 1945). http://www.theatlantic.com/doc/19407/bush

[4] E. Coelho, B. MacIntyre and S. Julier, Supporting Interaction in Augmented Reality in the Presence of Uncertain Spatial Knowledge, *18th Annual ACM Symposium on User Interface Software and Technology*, October 23-26, 2005, Seattle, Washington.

[5] S. Feiner, B. Macintyre and D. Seligmann, Knowledge-based augmented reality, *Communications of the ACM*, v.36 n.7, p.53-62, July 1993.

[6] S. Feiner, B. MacIntyre, M. Haupt and E. Solomon, Windows on the world: 2D windows for 3D augmented reality, *Proc. UIST '93 (ACM Symp. on User Interface Software and Technology),* Atlanta, GA, November 3-5, 1993, p.145-155.

[7] S. Feiner, B. MacIntyre, T. Hollerer and A. Webster, A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment, *Proceedings of the First International Symposium on Wearable Computers (ISWC),* Cambridge, Massachusetts, USA (1997) p.74–81.

[8] A. Hill, B. MacIntyre, M. Gandy, B. Davidson and H. Rouzati, KHARMA: An Open KML/HTML Architecture for Mobile Augmented Reality Applications, *9th IEEE International Symposium on Mixed an Augmented Reality*, October 2010, p.233-234.

[9] T. Hollerer, S. Feiner, T. Terauchi, G. Rashid and D. Hallaway, Exploring mars: developing indoor and outdoor user interfaces to a mobile augmented reality system, *Computers & Graphics 23* (1999) p.779–785.

[10] R. Kooper and B. MacIntyre, Browsing the Real-World Wide Web: Maintaining Awareness of Virtual Information in an AR Information Space, *International Journal of Human-Computer Interaction*, Volume 16, Issue 3 December 2003 , p.425-446.

[11] F. Ledermann and D. Schmalstieg, APRIL: A high-level framework for creating augmented reality presentations. *In Proceedings of the 2005 IEEE Virtual Reality Conference*, Bonn, Germany, IEEE Computer Society (2005).

[12] M. A. Livingston, J. E. Swan II, J. L. Gabbard, T. H. Hoellerer, D. Hix, S. J. Julier, Y. Baillot, and D. Brown. 2003, Resolving Multiple Occluded Layers in Augmented Reality, *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR '03)*, IEEE Computer Society, Washington, DC, USA.

[13] B. MacIntyre , M. Gandy, J. Bolter, S. Dow and B. Hannigan, DART: The Designer's Augmented Reality Toolkit, *Proceedings of 2nd IEEE and ACM International Symposium on Mixed and Augmented Reality*, October 07-10, 2003.

[14] O. Oda, Ohan and S. Feiner. Rolling and shooting: two augmented reality games, *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*, 2010, Atlanta, Georgia, USA

[15] Piekarski, W, 3D Modelling with the Tinmith Mobile Outdoor Augmented Reality System, *IEEE Computer Graphics and Applications,* Vol. 26, No. 1, p.14-17, 2006.

[16] D. Schmalstieg, A. Fuhrmann, G. Hesina, G., Z. Szalav´ari, L.M. Encarna¸c˜ao, M. Gervautz and W. Purgathofer, The studierstube augmented reality project, *Presence: Teleoperators and Virtual Environments 11* (2002) p.33–54.

[17] D. Schmalstieg, T. Langlotz and M. Billnghurst, Augmented Reality 2.0, *Virtual Realities*, 2011, p.13-37.

[18] N. Snavely , S. M. Seitz and R. Szeliski, Photo tourism: exploring photo collections in 3D, *ACM Transactions on Graphics (TOG)*, v.25 n.3, July 2006.

[19] J. C. Spohrer, Information in places, Systems Journal, 38 (4), p.602-628, 1999.

[20] I. Sutherland, The ultimate display, *Proceedings of the IFIP Congress,* p.506-508. 1965.

[21] R. M. Taylor, II, T. C. Hudson, A. Seeger, H. Weber, J. Juliano and A. T. Helser, VRPN: a device-independent, network-transparent VR peripheral system, *Proceedings of the ACM symposium on Virtual reality software and technology*, November 15-17, 2001, Baniff, Alberta, Canada.

[22] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond and D. Schmalstieg, Real-Time Detection and Tracking for Augmented Reality on Mobile Phones, *IEEE Transactions on Visualization and Computer Graphics*, v.16, n.3, p.355-368 , 2010-May/June.

[23] S. J. Yohan, S. Julier, Y. Baillot, BARS: Battlefield Augmented Reality System, *Proceedings of the NATO Symposium on Information Processing Techniques for Military Systems*, p.9-11, 2000.

[24] J. Zauner, M. Haller, and A. Brandl, Authoring of a mixed reality assembly instructor for hierarchical structures, *Proceedings of ISMAR 2003*, IEEE, p.237–246, Tokyo, Japan, October 7–10 2003.