AmbientVR: Blending IoT Interaction Capabilities with Web-based Virtual Reality

Shivam Verma
Felicitous Computing Institute, National University of Singapore
shivam@comp.nus.edu.sg

Darren Carlson
Department of Electrical Engineering, University of Hawaii at Manoa
darren.carlson@hawaii.edu

David S. Rosenblum
Felicitous Computing Institute, National University of Singapore
david@comp.nus.edu.sg

ABSTRACT
Emerging frameworks for Web-based Virtual Reality (WebVR) enable developers to easily create VR applications that can be experienced using a smart-phone, mobile web browser and a low-cost headset. Although WebVR has great promise, the limited input and output capabilities afforded by modern web browsers prevents user interaction with external VR controllers, precision hand tracking sensors and room-based user positioning devices. In this paper, we present AmbientVR, an open framework for blending a broad range of sensing, interaction and control capabilities with WebVR applications executing in unmodified mobile web browsers. The approach is able to discover compatible input/output devices within the user’s environment and integrate their capabilities into the browser on-demand, allowing users an added degree of control within the VR world. In addition to sensors, AmbientVR can also discover and integrate a broad range of Internet of Things controllers and actuators, making it possible to created hybrid IoT/VR applications that can be deployed without needing to download and install an application. In this paper, we present a detailed overview of the AmbientVR framework, introduce a fully functional WebVR prototype, and present a performance evaluation demonstrating that our approach imposes low processing and memory overhead, making it suitable for deployment on many commodity mobile devices.

ACM Classification Keywords
H.5.1. Multimedia Information Systems: Artificial, augmented, and virtual realities; D.2.12 Software: Interoperability—Distributed Objects

Author Keywords
Virtual reality, WebVR, interaction, input devices, middleware

INTRODUCTION
Virtual reality (VR) refers to a range of computer technologies that immerse users in a three-dimensional simulation of real or imaginary environments, which are experienced through a Head-mounted Display (HMD) and controlled through movements of the user’s body. State-of-the-art VR systems support highly-detailed 360-degree graphics and a variety of techniques for high-precision head, hand and body positioning tracking. Sensed tracking data is fed back into the VR system to provide the user with an interactive physical representation with the virtual world. State-of-the-art VR systems typically require powerful computer hardware, a dedicated headset and specialized software. Factoring in their (often prohibitive) cost, these powerful VR systems are not yet widely deployed.

The increasingly powerful sensing, communication and processing capabilities of modern smartphones provide an alternative hardware and software platform for developing low cost, mass-market VR experiences. An example of this approach is Google Cardboard (shown in Figure 1), which provides a simple way for users to head-mount a standard smartphone for use as a low-cost VR viewer. The software accompanying Cardboard splits the display from compatible applications into two slightly distinct images, resulting in a stereoscopic three-dimensional (3D) image that interactively updates to follow the motion of the user’s head position. Head position is tracked using the phone’s inbuilt accelerometer and gyroscope sensors.

There are several ways to build smartphone-based VR experiences. Developers can use the Google VR SDK for Android to create native applications in Java with OpenGL. Alternatively, the Google VR SDK for Unity can be used to create native applications with the Unity 3D engine. However, in addition to a steep learning curve, these approaches require

Figure 1. Google Cardboard Prototype
users to download and install dedicated applications (apps), which must be managed and updated. Given the increasing capabilities of current web browsers, an alternative development approach is to build VR applications using HTML5, WebGL and JavaScript, and then deploy these virtual reality experiences on the Web. Web-based VR (WebVR) provides a low friction way for users to discover and try VR applications using nothing but a mobile browser, such as Google Chrome. Moreover, application updates can be pushed instantly without having to go through the app stores.

Virtual reality on the web has great promise. The ability to access rich VR experiences without a download or install and for a developer to be able to create VR content for the masses using simple, affordable tools could be truly revolutionary. However, amidst the potential lies one serious obstacle: the lack of intuitive, user-friendly interaction techniques for our in-browser VR world. Additionally, owing to the limited interaction capabilities of our web browsers, designing new techniques for WebVR can be extremely challenging.

WebVR users increasingly find themselves enveloped in an emerging world-wide network of interconnected physical objects, called the Internet of Things (IoT). The IoT includes a rapidly expanding variety of sensors, actuators and control devices (e.g., wearables, environmental sensors and connected devices) that are uniquely addressable and are available through networking technologies such as WiFi, Bluetooth and others. Many of these IoT devices are laden with sensors and actuators that could provide novel interaction capabilities for VR worlds in unforeseen ways; however, most of these IoT devices cannot be discovered and accessed by current mobile browsers, resulting in a “cyber-physical gap” [3] between WebVR applications and the IoT.

To tackle the aforementioned issues, in this paper we present AmbientVR, an open framework for blending sensing, actuation and control devices from the physical environment with VR applications experienced in traditional web browsers. Our approach leverages the Ambient Dynamix Framework [4] to provide a lightweight middleware platform for plug-and-play context sensing. AmbientVR is able to discover available input/output devices within the user’s physical environment and integrate their capabilities into the browser on-demand, allowing users an added degree of control within the VR world.

RELATED WORK

The concepts implemented in AmbientVR were drawn from several areas. MR toolkit [10] and more recently VRPN [12] implement device abstraction for traditional VR peripherals. A web interface to VRPN is also introduced in [9]. With the assumption that the types of input devices for VR are relatively fixed, VRPN provides a set of traditional device classes such as a Tracker, Button, Dial, etc. Similarly, device abstraction is also an important goal for us. However, a static interface of device classes hinders the easy inclusion of new devices and modalities that do have the potential but are not typically meant to be used in a VR environment. So instead of relying on a predefined set of device types, AmbientVR introduces a data-types based approach. As a result, AmbientVR-enabled apps can support a variety of devices for simpler interaction requirements or combine these data-types to support more traditional VR peripherals or devices with novel interaction capabilities.

OpenTracker [8] introduces a “write once, track anywhere” approach to VR tracking, and Ubitrack [7] allows mobile users to introduce their own equipment at runtime for VR tracking. Even though we are not limited to tracking, these architectures inspire the design of AmbientVR. Our support for discovery of compatible devices at runtime enables a “write once, interact anywhere” approach. A fully functional WebVR app can be shared simply through a URL. Additionally, the support for dynamic deployment of device drivers ensures that new devices can be supported on existing WebVR apps as and when they are available to the user. AmbientVR can also be easily extended to support each of these existing systems. An integration with VRPN is in the works.

From an IoT perspective, Middleware frameworks [1] [2] have explored novel techniques for adapting a mobile device’s protocol support and sensing capabilities. However, to the best of our knowledge, there has been no concerted effort to bridge the gap between the Internet of Things and WebVR.

BACKGROUND

As the IoT fragments, everyday networked environments are becoming saturated with heterogeneous (and often incompatible) IoT services. Problematically, IoT services adopting one set of protocols may be effectively invisible to mobile applications supporting another set of protocols, even if their integration would be highly advantageous to the user.

Projects like Ambient Dynamix [4] have explored middleware techniques for adapting a mobile device’s protocol support and sensing capabilities to fit a given physical environment at runtime. Dynamix runs as a lightweight background service on the user’s mobile device, providing adaptation support to applications (native and web) through plug-ins that can be installed on-demand and over-the-air. Dynamix provides several features relevant to our work on AmbientVR. First, Dynamix plug-ins provide access to low-level sensors, such as accelerometers and gyroscopes, in addition to higher-order data, such as gesture input, voice recognition, and network-based services operating in a room or larger smart-space. These plug-ins are integrated into an embedded OSGI container at runtime from a configurable set of public or private repositories. Within a Dynamix instance, plug-ins are also able to consume the services offered by other plug-ins (e.g., register for context sensing or device control services). This enables a Dynamix device to serve as a smart gateway between mutually incompatible smart devices situated in the user’s environment. Dynamix also exposes its plug-and-play IoT sensing and control capabilities to Web apps running in native mobile browsers, such as Google Chrome and many others. We refer to Web applications that blend physical device services (e.g., sensing, actuation) with virtual web content as Ambient Web pages [3].

Although Dynamix provides a foundation for building Ambient Web applications, specialized application domains, such as WebVR, require comprehensive application support to be
layered on top of Dynamix. The next section describes our high-level approach for making WebVR context-aware using the low-level features of Dynamix.

**AMBIENTVR**

In this section, we introduce AmbientVR: an open source framework for blending rich sensing, interaction and control devices with WebVR apps in the browser. The AmbientVR architecture consists of a drop-in JavaScript (JS) library and an open repository of VR-related data models that provide service descriptions for various IoT devices. The JS library is designed to serve as a conduit between a WebVR app and the gateways providing input/output capabilities in the user’s environment. For our prototype implementation, we use the Ambient Dynamix Framework as our IoT gateway on the user’s mobile device. The approach however, can easily be extended to support a variety of other gateways.

**Meta Tags and Data Models**

The first requirement in building a context-aware web app is identifying the Ambient-Resources that are required for the web app to function properly. We define the term “Ambient Resource” as any physical or virtual context provider (e.g., sensor), actuation provider (e.g., controller), or network-based service (e.g., Web information or service) that is relevant to the user’s interaction with an application. On the Web, we need an open and flexible way to express the Ambient-Resources required for the page to operate correctly in tandem with the IoT gateway.

HTML meta tags have long been used to describe complex web page attributes, such as presentation or behavior, or to convey “out of band” information such as descriptions, keywords or language, which can assist search engines. In this work, we introduce a meta tag set that allows Ambient Web developers to describe required sensing or actuation support needed by the page for proper operation. We call these meta tags ambient-web meta tags.

While designing ambient-web meta tags, we noted two principal mechanisms by which Ambient Resources communicate events: broadcast and request/response. Broadcasts use a publish/subscribe messaging model, where an application subscribes to one or more event publishers, which publish event data to registered subscribers as needed. Gesture recognition and collision detection are both suitable examples of broadcast-based communication. Request/response, on the other hand, is based on the call-to-action messaging model. In this case, the application sends a request (possibly with some configuration data) in order to perform an action. The Ambient Resource attempts to perform the requested action, and may send a response back to the application. Ambient-web meta tags mirror this model of communication and as such the data-ambient-broadcast and data-ambient-request attributes can be used to describe their Broadcast and Request support requirements. The structure of an ambient-web meta tag is as follows:

```html
<meta
  data-ambient-type = 'ambient-web',
  data-ambient-broadcast = '{
    "response": "data-type",
    "input": "data-type"
  }',
  data-ambient-request = '{
    "output": "data-type",
    "input": "data-type",
    "response": "data-type"
  }',
/>
```

Listing 1. Ambient-web meta tag structure

We have defined three custom data attributes that application developers can include in an ambient meta tag. The attribute values are defined as JSON compatible strings for easy parsing using JS. Additionally, as suggested in the HTML standard, we use the ambient keyword in every attribute to reduce the risk of namespace clashes with other JS libraries.

- **data-ambient-type** (mandatory): A keyword specifying that the web-app is able to provide enhanced interaction capabilities through Ambient Resources situated in the user’s physical environment. This attribute also informs Ambient Resource providers about the application being ambient-web compatible.

- **data-ambient-broadcast** (optional): This attribute indicates requested support for broadcast events, and is defined as a JSON object string. The response property of the JSON object specifies the event data-type needed from a device. The input property is used to specify if the device should support any configuration options when initializing support for the given response type. For example, an IoT device that broadcasts orientation sensor data could support sampling frequency as a configuration option.

- **data-ambient-request** (optional): This attribute indicates support for request-based events, and is defined as a JSON array string. Each object in the array describes a request that an Ambient Resource must support. The output property specifies the data-type of the output that the Ambient Resource must generate when a request is made. The response property specifies the data-type that the application expects in return when making this request. For example, an IoT device providing haptic feedback could support notification time period as an input data-type, vibration as the output and boolean as the response data-type. Each of the JSON properties defined above requires a data-type as the value. The data-type must be described as a well-known data-model, such as those provided by our project or by other projects such as the Open Connectivity Foundation (OCF).

The current lack of standardized IoT communication mechanisms creates a barrier to integrating proprietary or unknown IoT devices and services. To address this challenge, we are developing an open-source repository of VR-related data-models for AmbientVR. Our data models are based on JSON-Schema.
v4, an open-standard format used to describe data objects using key-value pairs. Our data-model describing orientation sensor data can be seen in Listing 2.

```json
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "Orientation Sensor Data",
  "description": "Pitch, Yaw, Roll in Radians",
  "type": "object",
  "properties": {
    "pitch": {
      "type": "number"
    },
    "yaw": {
      "type": "number"
    },
    "roll": {
      "type": "number"
    },
    "required": [
      "pitch",
      "yaw",
      "roll"
    ],
    "timestamp": {
      "type": "string",
      "format": "date-time"
    }
  }
}
```

Listing 2. Data-model for orientation as pitch, yaw and roll

With the use of JSON Schema, we are also trying to tackle the problem of data-heterogeneity. Wherever possible, our data-models are defined as a union of each representative form of the data. For example, orientation data of any object in the 3D space can be defined by the pitch, yaw and roll values or as a unit quaternion. To take into account both types, the orientation data-model is defined as a union of the data-models for pitch, yaw, roll and unit quaternion. We expect the ambient-web developer to use any of the underlying data-models as per requirements and knowledge. However, the onus of implementing support for all the underlying data-models lies on the domain expert (i.e., the Ambient Resource provider). This ensures that an ambient-web app is compatible with the maximum number of Ambient Resources.

The AmbientVR Framework

In this section, we describe the AmbientVR approach. For simplicity we use Dynamix as our IoT gateway, which is used as an Ambient Resource provider. We note that the VR developer never directly interacts with Dynamix. Rather, the intricacies of communicating with Dynamix are handled internally by AmbientVR. This lowers development complexity and ensures that AmbientVR can be extended to support additional Ambient Resource providers as required.

To create an ambient-web VR experience, a developer browses through our open repository of data-models and notes down what sensing or actuation capabilities the WebVR app requires. Using a set of ambient-web meta tags, the developer then describes the required Ambient Resources in the head of the HTML document. The description of requirements using declarative data-models ensures that any compatible Ambient Resources from the user’s environment can be used at runtime; hence, the approach is not limited to one device or a constrained set of devices. Finally, the developer starts creating the VR scene using the web technologies of his or her choice and the AmbientVR interaction model described below.

To add AmbientVR support, the VR developer includes the AmbientVR JS library in the HTML page. Including the library in the page creates a global JS object that acts as the entry-point for the developer into the IoT based interaction capabilities available in the user’s environment. To initialize communication with AmbientVR, the developer starts by calling the init() method, providing a listener as an argument. This listener receives updates about the availability of compatible Ambient Resources in the user’s environment.

Upon initialization, AmbientVR uses jQuery to parse the ambient-web meta tags on the page using the known attribute value $\text{meta[ambient-web]}$, creating a list of data-type requirements. It then proceeds to bind with Dynamix using helper methods provided by the Dynamix framework. On a successful bind, AmbientVR requests Dynamix for the data-types supported by its plug-ins. Each plug-in description provided by Dynamix includes the install status and the supported Broadcast and Request data-types. The plug-in metadata also contains a short description of the plug-in. Given that there is a higher probability of finding a device in the user’s environment if a plug-in is already installed, AmbientVR tries to match requested data-types starting with installed plug-ins, followed by the others. The comparison using data-types (rather than specific plug-in identifiers) ensures that new devices can be used with existing ambient-web apps as they are added to the Dynamix ecosystem.

The ordered list of compatible plug-ins is then displayed to the user. The user then selects and confirms which Ambient Resources are currently available in his vicinity. While it is possible to automate the device discovery process using Dynamix’s capabilities, we made the decision to keep the user in the loop, giving the user an opportunity to select desired devices or make unavailable devices visible (e.g., by switching on the device).

Once the user selects Ambient Resources for integration, AmbientVR requests Dynamix to install the required plug-ins. This step is ignored if a plug-in is already installed. When the plug-ins are successfully installed, AmbientVR requests
Dynamix to add support for specific data-types. If the plug-in is able to successfully setup a connection with the Ambient Resource, AmbientVR informs the listener provided during init() call about the availability of a compatible Ambient Resource in the user’s environment. The developer at this point can start communicating with the Ambient Resource.

To start receiving Broadcast data, the developer can register a listener using the on() method provided by AmbientVR. On receiving this request, AmbientVR sends a request to Dynamix for the support and sets up a communication channel between the provided listener and the Ambient Resource. This data can now be easily hooked up with the app logic and the corresponding Ambient Resource can now be used to interact with the WebVR app.

Requests can be sent using the send() method provided by AmbientVR. If the developer specified an input type for the Request in the HTML Meta tags, a configuration object can be sent as an argument. If response data is expected for a Request, a callback method can also be specified in the send() method call. Internally, AmbientVR routes this request to the corresponding Dynamix plug-in with the configuration data. If a callback method is provided, it is invoked when a response is received from Dynamix.

For a more compelling user experience, the developer can utilize "progressive enhancement", whereby the web page is designed to progressively add functionality and associated presentation capabilities based on the ambient resources discovered in the user’s physical environment by the IoT gateway.

Prototype Implementation
To validate the AmbientVR framework described above, we created a fully functional WebVR game called Space Shooter, which is shown in Figure 2. WebGL was used for 3D rendering in a web browser. WebGL runs on many browsers and hardware platforms, and like anything web, there are many open source libraries and frameworks that ease WebGL development. Two of the most popular libraries with virtual reality support are Three.js, BabylonJS and the recent A-Frame, an open-source framework for creating virtual reality experiences with HTML markup alone. For developing the 3D environment for our prototype game, we chose the Three.js [13] library.

Setting up the scene
The prototype game consists of an airplane flying through space. When running in a desktop browser, the game renders as a 3D scene where the player can use the keyboard input as game controls. However, when running on a mobile browser, the game can render itself in a stereo VR mode. To enable VR support on a mobile browser, Three.js needs access to orientation data of the mobile device. Hence, we register a listener for the ‘deviceorientation’ event in the JS code. When this event is triggered on a mobile browser, the game pops up a small dialog box asking if the user wishes to play in immersive VR mode. If the user selects yes, this initializes AmbientVR support in the game and the display changes to a stereo view. The smartphone can now be placed into any Cardboard VR viewer, and then Space Shooter can be played as a VR-enabled game. In the background, AmbientVR uses Dynamix to discover and integrate nearby IoT devices and then sets up interaction and event support for available Ambient Resources, which include voice input, orientation data, haptic feedback and tap detection.

Interacting with the WebVR game
Controlling airplane orientation and speed – To provide example controllers for our prototype game, we have developed Dynamix plug-ins for the Sphero robotic ball [11] and the Myo armband [6]. The Sphero robotic ball communicates with an Android device over standard Bluetooth, providing orientation data in the pitch, yaw and roll format or as unit quaternions. The armband communicates via Bluetooth Low Energy (BLE), supporting streaming orientation data as pitch, yaw and roll values or unit quaternions. Both plug-ins provide orientation information in our defined orientation data-model; hence, they can be used with any ambient-web application requesting orientation data support. Once AmbientVR is able to setup connection with either of these devices, they can be used as a controller for the airplane in our VR game. The yaw and roll values of the selected Ambient Resource (the Sphero or the Myo in our case) are mapped to the yaw and roll values of the airplane in the VR world and can be used to control the direction of flight. The pitch angle is mapped to the speed of flight and can be used to speed up or slow down the airplane.

Voice commands to Play and Pause – Every smartphone has a microphone and the ability to input speech. To capitalize on this ubiquitous resource, we developed a hot-word triggered speech recognition plug-in for Dynamix. Using CMU’s Pocketsphinx Android library [5], we developed a real-time, continuous hot-word detection plug-in that is able to recognize the "Ok Dynamix" keyword. A successful recognition of the hot-word initiates Android’s internal speech recognizer and is indicated using a beep, which is something many Android users are familiar with. The user can then say
a command, which is passed from Dynamix into the game logic running in the browser. In our prototype, we defined the “Play” and “Pause” voice commands to control the game state (play/pause respectively). Although the set of voice commands is small in our prototype, the number of possible voice commands is not restricted. Since the input speech is processed in real-time, and results are passed on to all subscribers, developers can define their own set of commands per application.

By abstracting sensor/actuator providers and providing on-demand integration of proximate providers through Dynamix, AmbientVR can dynamically link any ambient resource providing required sensing functionality (e.g., Sphero and/or Myo) to the game at runtime, based on which device is present in the user’s physical environment. This general approach can be extended to emerging sensors, actuators and other game controllers by the open source community.

**EVALUATION**

We evaluated the performance of our prototype by conducting a stress test scenario that investigated several critical performance metrics, including latency of messages received, total messages received per second, as well as the CPU and memory usage of the Dynamix process. We developed an evaluation web-app along with a set of Dynamix plug-ins that generated test data at controllable frequencies. To obtain maximum performance, the plug-ins only perform the task of streaming data without any hardware interaction. The event data consists of a timestamp of when the message was created along with the heap size at that moment. The plug-ins can be configured to stream data at a given rate using an AmbientVR request. For our evaluation, we used message frequencies between 5 and 100 Hz per plug-in, with test scenarios that consisted of up to 5 plug-ins streaming data simultaneously to the web-app. Each test scenario ran for a period of 5 minutes. The evaluation was performed on a Samsung Note 4 smartphone with the following specifications: CPU: Quad-core 2.7 GHz Krait 450 - Snapdragon 805; RAM: 3GB; Android version: 6.0.1 and Dynamix version: 2.1.0.

**Messaging Performance**

The results in Figure 4 show that the maximum achievable median transmission rate over the complete test run is 248 messages per second for 5 plug-ins when streaming data at the rate of 50Hz per plug-in. We see that increasing the number of plug-ins or increasing the data transmission rate does not affect the message throughput. However, it does affect the latency of the messages received. The measured message latency is stable in the range of 85-100ms for 2-4 plug-ins between the data transmission rates of 5 and 50Hz. For a single plug-in, the latency linearly increases from 70 to 99ms as the message frequency increases from 5 to 100Hz.

In most AmbientVR experiences, we can expect to have one main controller device, such as the Myo armband streaming data at a medium-high frequency (e.g., 50Hz) with two to three other devices streaming data at lower rates. The latency in such a scenario would range from 70 to 100ms. While such latency values may not provide optimal position tracking in 3D space, they can be used for lower-precision tracking scenarios. Furthermore, command interfaces, such as voice input or gesture input, do not have demanding performance requirements. For many WebVR use cases (e.g., gesture controlled menu navigation), AmbientVR provides good performance at relatively low latency.
While there certainly are Ambient Resources that stream data at very high frequencies, *AmbientVR* is not yet able to consume raw data from such devices without introducing excessive latency or loss of messages. This does not in any way mean that we cannot utilize the capability of such devices. It only puts a limit on the frequency of messages that can be sent to the WebVR app using *AmbientVR*.

**System Performance**

Along with a message throughput evaluation, we also performed CPU and heap utilization measurements for the Dynamix process during the execution of our test scenarios. To acquire the necessary data we developed a monitoring application that is capable of collecting continual CPU usage statistics for the Dynamix runtime process. The allocated heap value at each sampling time is included in every message generated by our test plug-ins. The monitoring application runs in its own process, which isolates the Dynamix process during measurement. The measurements were obtained on separate test runs from the message evaluation using the same setup with detailed logging disabled.

Results of the system performance evaluation can be seen in Figures 5 and 6, which show the median CPU and heap usage of the process over the test runs. It can be seen that at very high data transmission frequencies, the median heap allocation drops considerably because of garbage collection at a higher frequency. We can also see from Figure 6 that the number of plug-ins does not show any considerable impact on the allocated heap size. At very high message throughput (e.g., five plug-ins streaming data at 50Hz), the increased heap allocation leads to unstable behavior and application crashes. However, the heap size remained stable for all but the highest workloads.

The CPU utilization in Figure 5 shows that message frequency has a bigger impact on the system load than the number of plug-ins participating in a scenario. Even with 5 plug-ins streaming data at 20Hz the CPU utilization remains below 20%. For more practical use case scenarios, such as two plug-ins streaming data at 40-50Hz CPU usage varies from 19.4 to 23.2%. While CPU utilization does tend to increase at higher message frequencies, the load is comparable with resource-intensive applications such as games on the Android platform. Previous work [4] has shown that Dynamix CPU consumption can be reduced to 5% - 9% for lower sampling rates (e.g., a single plug-in sampling at less than 20Hz), meaning that battery drain can be reduced by adjusting sensing rates or by reducing features (e.g., reducing the number of connected ambient-devices) as needed.

**CONCLUSION AND FUTURE WORK**

The easy availability of low cost VR solutions, such as Google Cardboard, has opened doors to new opportunities for building WebVR experiences for the masses. However, the limited input and output capabilities of current browsers severely limits the user experience due to inadequate interactive capabilities. To address this challenge, we are developing *AmbientVR*, an open framework for blending rich sensing, interaction and control devices with web-based VR within the browser. Our approach connects conventional WebVR to novel interaction...
possibilities provided by emerging IoT devices. AmbientVR enables developers to add device and vendor independent interaction capabilities to build truly immersive VR experiences. The framework is easily extensible and can operate without any dedicated infrastructure or host.

This paper presented an overview of the AmbientVR framework, introduced a novel data-model based approach for VR interactions, and demonstrated a fully functional VR game that leverages IoT Resources to provide novel interaction capabilities for the web-based game. Our performance evaluation indicates that the AmbientVR prototype provides efficient messaging performance and imposes low processor and memory overhead, making it suitable for deployment on many commodity mobile devices. To the best of our knowledge, AmbientVR is the first open framework to approach the WebVR interaction problem from an Internet of Things perspective.

We are currently focusing on several areas of future work. First, we are designing additional data-models that focus specifically on VR interactions. We are also developing supporting Dynamix plug-ins for novel interaction techniques for WebVR. Examples of these plug-ins include step-detection using head mounted phones and advanced gesture recognition using commodity hardware devices. In addition, we are porting existing Cardboard VR input techniques such as Magnetic Switch and headset tapping [14] to the Dynamix ecosystem so that they can be used as easily with AmbientVR applications. We are also developing remote pairing capabilities for Dynamix and AmbientVR, which will enable these IoT interaction capabilities to be used on desktops. This will open up new interaction possibilities for deploying AmbientVR enabled applications on state-of-the-art head mounted displays such as the Oculus Rift and HTC’s Vive.

ACKNOWLEDGMENTS
This work was supported in part by the National University of Singapore and Singapore A*STAR - SERC PSF Grant 1521200084.

REFERENCES


