2018

Data Visualization Using Augmented Reality

Derek VanOverloop

Grand Valley State University

Data Visualization Using Augmented Reality

By
Derek VanOverloop
May 2018

Data Visualization Using Augmented Reality

By
Derek M. VanOverloop

A project submitted in partial fulfillment of the requirements for the degree of
Master of Science in
Computer Information Systems

at
Grand Valley State University

May, 2018

Jonathan Engelsma Ph.D. **Date**

Table of Contents

Abstract

Data visualization in computer science has been limited to computer screens and paper printouts. Often information can be displayed to the user, but it lacks context. Augmented reality technologies enable us to add context to the information. The goal was to experiment with embedding data representations into the physical world using augmented reality. The Unity™ game development platform was chosen to develop the models. C# was used to develop software to retrieve and direct data to user facing objects. It is also used to manipulate graph objects to display new graph data. Unity™ also offers plugins that allow for developers to use the recently released Apple ARKit® APIs for augmented reality development. Several user interactions were developed to allow the user to select and place graphs.

To date most of the experiences with augmented reality have been associated with the gaming industry. However, the technology has the possibility to impact a much broader audience. Augmented reality provides a unique way to synthesize data and the physical world in a way rarely seen in computer science. It can provide context to data representing physical objects by visualizing the data in the physical world.

The project demonstrates that data can be visualized using augmented reality. One of the most exciting observations is that this method can help contextualize data. Limitations in the augmented reality make some interactions less natural. Placing graphs require a horizontal surface, or extreme model drift is observed. However, the APIs are relatively new, and updates are available that might facilitate better user interactions.

Introduction

This project is about visualizing data. The goal was to specifically try to see acceleration data in a new way. I work at Vibration Research Corporation for my day job. Most of my time is spent developing software that charts different forms of data. These charts live in a 2-dimensional world on a computer monitor. However, the data is coming from the real world. The data is streamed from sensors placed on various products. My experience is that there is a disconnect between the device under test, or DUT, and the data coming in. One example of a type of disconnect faced by users is, which data stream comes from which sensor. Another disconnect might be what is happening to the DUT at the time a data recording is taken.

Recent advancements in augmented reality provided a possible mitigation of the abstraction between data and the DUT. The primary goal of the project was to use augmented reality to overlay live data on top of the DUT. There were several secondary goals. The first was to come up with several data presentation formats that were not a traditional X-Y graph. A second goal was to develop a positive user experience for placing graphs. The user experience of placing a graph in the real world needed to be intuitive. The final goal was personal. I have never worked with game engines, and I wanted to learn how to use those tool sets.

Background and Related Work

Augmented reality seems to be gaining traction. This is due to the popularity of games such as Pokémon Go. Another contributing factor is the recent release of augmented reality application programming interfaces, or APIs from major software companies like Apple and Google. Current applications like Pokémon Go provide a good foundation for the positive user experiences. Models of Pokémon's are placed on appropriate surfaces. The most popular augmented reality application currently on the Apple App Store® (Apple Inc., 2018), CSR Racing 2, allow users to place a model of an automobile in front of them. The application then lets them view the vehicle from any angle (NaturalMotion, 2018). Another similar experience is from the Ikea augmented reality application (Inter IKEA Systems B.V., 2018). The weakness of these applications is that they are presenting a model and what that model looks like. The popular applications are not presenting data to a user.

Virtual reality has several applications that do present data to the user. Applications like the Nasdaq history visualization by the Wall Street Journal are good examples of what presenting data can look like. The application lets the user ride the history of the Nasdaq stock market for the last 21 years like they are on a roller coaster (Kenney & Becker, 2015). Important dates are presented as the user passes them in time (. It does a wonderful job at presenting data traditionally confined to an X-Y plot into 3 dimensions. Another example of unique data presentation using VR is one published by Simon Rogers. The application provides an interactable globe with information about Brexit. The user moves the globe and selects different countries. When selected, information is displayed in a heads-up display format. The user experience provides a unique way to look at geographic data (Rogers, 2016). These applications, and most VR data applications, fall short of the goals for this project. The physical world isn't brought into these scenarios, so users can't easily correlate sensor data with the device being tested.

IBM developed an augmented reality application that presents data to the user. The team was working with analyzing data using tensor flow and then presenting the analyzed data to data scientists using augmented reality. The program they came up with was well executed. The application plots data in 3 dimensions and allows a user to interact with the data by resizing and moving the chart (Resnick, 2017). The user experience was quite natural as well. Hand gestures were used to select and move the graphs. There were 3 weaknesses to the IBM project as relates to the goals of this project. The first is that the data being presented is relatively static. The program was analyzing historical data, while the goal of this project is to analyze live data. The second weakness is that they are using traditional points on a graph to visualize the data. They might have been able to take the visualization further. The third weakness is that they were not using common hardware. The hardware platform was Microsoft's HoloLens, an expensive and uncommon platform.

Program Requirements

The project had a small scope since the amount of learning was quite high. The application needed to be an augmented reality application that was able to present live data to a user. This broke down into three functional areas. The first area is data retrieval and processing. The application uses HTTP gets to retrieve live sensor data from a data recorder. The response must be parsed and then propagated to appropriate data processors. The second piece of the project is allowing the user to place graphics in the physical world, hopefully as accurately as possible. The intention is that the user can place a data representation on or near the sensor that is collecting the data. The last part of the project is to process the data and show it to a user in a meaningful way. The ideal presentation formats are ones that lend themselves to 3 dimensions. Finally, the application must be available on common platforms that users have access to. This means not using expensive or advanced platforms such as HoloLens or other dedicated augmented reality platforms.

Implementation

One of the first decisions that needed to be made was what toolset to use. There were 2 candidate APIs available that met the requirement that the project needed to be widely available. Those were ARKit by Apple and ARCore by Google. We chose the solution by Apple, because at the time it supported the most devices.

ARKit enables the augmented reality component to the project. The API controls all the device tracing, and model placement. It uses the build in accelerometers and cameras to orient the user's device in space, and then track the devices movement. In this project the API was implemented using version 1.0.14 of the Unity plugin Unity ARKit Plugin (2018). The API was configured to detect horizontal planes, and then detect when the camera was pointing at the horizontal plane. This was used to place the graph positioner model on a flat surface. Horizontal planes drift less than abstract points in space, so graph movement was minimized.

The main component in this project was implemented using Unity version 2017.3.1 (2018). This software is traditionally used for game development; although Unity is currently making a push to use the software in other domains such as movies. Unity fuses 2D, 3D, and user interface components, called game objects, with the software. It also manages the lifecycle of the components in a scene. The simplified version of the game object lifecycle is instantiation, set visible, frame rendering, set invisible, and the object destruction. Unity provides an API that allow developer to hook into these events. This allows users to initialize and clean up any values. Unity also handles the positioning, orientation, and scaling of the game objects. This was used to move and resize the graphs when data comes in. It was also used to move and rotate the graph positioner as the device is moving. Another Unity component that the project made use of is rays and raycasting. A ray is like a geometric ray. It has a start point and a vector. Where it differs is that a ray can have a max distance. Rays, collide with a collision layer set on any of the game objects. The Unity API can send the ray, called raycast, and then detect which objects were hit. It is used in this project for detecting if a device screen is pointing at a horizontal surface, or if the user pressed the graph positioner object. One last important concept in Unity is prefabs. They are very much like a class, but for game objects. A component is a predefined set of game objects. These prefabs are instantiated, much like you would instantiate a class in a programming language. The result of an instantiation is a copy of the prefab object. This allows for multiple copies of the same object to exist in the same scene, without having to define them manually. They were used in this project for each of the graph types, as well as the positioner. Any time the user adds a graph a positioner and graph are instantiated.

This software can be written in C#, JavaScript, and Boo. For this project I used C# because I was most familiar with the language. There was still a steep learning curve to understand and use the Unity libraries, since I had never used it before. In Unity a scene is the environment, and inside the scene are game objects. There are many kinds of game objects. In this project 3D game objects are the most common. All the graphs are composed of cubes. The traditional user interface portion of the application is composed of a canvas, buttons, text input, and plain text. C# scripts are added to game objects. They inherit the lifecycle events of the parent game object. This allows the script to perform a set of tasks when certain events happen. One commonly used trigger is the frame render function. This is used in conjunction with the ability of scripts to interact with the game objects. In the case of the graphs; they resize, reorient, and reposition the data representation in 3D space on every frame render.

The program architecture has 3 components. The two main components are the data producers and the data consumers. There is 1 data producer in the scene. The main role of this component is to retrieve data and send it out to all the data consumers. The data is retrieved via a HTTP GET request issued to a REST server on the ObserVR1000. For most of the project a request for the latest sensor data for each channel was used. This turned out to be a limitation, because the GET commands couldn’t be issued fast enough to capture transient events. It meant that the spikes in acceleration due to shock events on an accelerometer were not captured by the software, and therefore not visible. The software uses a different request now. This request returns a history of the sensor data. The data processor component turns the response into a usable C# object with that history as a list. This list contains a few thousand data points. It would not be possible to display all those data points at 20 – 30 frames a second. To compensate for this limitation the highest value for each channel is recorded. This new value is passed on to the data processors.

Data processors are the second major component of the software architecture. Each data processor subscribes to the new data available event from the data provider. This wasn’t always the case. Initially the data providers and data consumers followed a custom observer pattern implemented using interfaces. The publish interface had a subscribe and unsubscribe function that the consumers could call. The consumer interface had a function that new data was published to. The data provider would store one subscriber for each channel of data. This placed an artificial limit that there could be only one graph per channel. This, along with the fact that it was essentially the event system in C# triggered a refactor of the software. The software now uses the C# event system, and each data consumer chooses which channel of data it listens to. On top of being less code, it is simpler, and concerns are separated. The data provider no longer needs to know what data consumers care about which channel’s data.

There are several different types of data processors, in fact graph has its own customized graph processor. The reason is that each graph has a customized graph processor is the application of the data is different per graph type. The moving graph needs to apply a location transformation to the graph game object, while the resizing box and bar graphs need to rescale the game objects. The application of each is different, since moving and resizing happens in different ways. During the development there was a lot of learning and experimentation with regards to these game object manipulations. Programming 3D objects was new for me, but with the help of online Unity documentation, and their YouTube tutorials I was able to figure it out. One challenge faced was to move a game object relative to an existing position, or to set its absolute position. The first required calculating a delta value for the sensor data and keeping track of where the graph needed to end up based on the delta value. The second option required more research because game movement is done incrementally from frame to frame. This meant the solution wasn't as available. The final product for most of the positioning proved to be relatively simple. A new 3D point was created using the latest sensor data, and then applied to the game objects position. Some of the graphs use linear interpolation for smoothing graph movement out. There is a Unity API that can handle the complex math for positioning an object from frame to frame. Using this library turned out to be a bit more problematic. This is because linear interpolation computes position as a function of time, so on top of the graph position time needed to be considered. The final product uses linear interpolation for the resizing graph. The size of the graph scales towards its goal, until a new sensor data point is received. The other two graph types use an absolute position without any smoothing. The data comes in fast enough that the movement is acceptably smooth for the overhead required for linear interpolation.

The last large component in this software solution is the graph positioning system. This component is not as major to the base functionality as the data provider and the data processor, but it proved to be just as complicated to implement. The positioner is instantiated from the user interface each time a user selects a type of graph to add. The prefab of the selected type is included as a parameter when the graph positioner is instantiated. This mechanism is how the selection is stored until the graph is placed. The positioner is state machine with 3 different states. The first state is where no horizontal plane is detected by ARKit. In this mode the orientation and location of the device passed into a ray, which is then cast and checked to see what was hit. If a horizontal plane was located, then the state machine transitions to the plane detected state. In the plane detected state the location of the positioner is updated based on where the device is pointing. Raycasts are used once again to determine where the device is pointing. If the device stops pointing at a horizontal surface, then the state switches back to plane not found. If the user touches the positioner game object, also determined by a raycast and device orientation, then the graph prefab is instantiated, and the positioner game object is deactivated. The graph positioner changes shape based on its state. The changing graphics are intended to convey to the user when they can and cannot place a graph. This strategy is recommended by Apple augmented reality user interface guidelines. In my experience it works quite well, once the user knows how this paradigm works.

Using the graph positioner to limit where users can place a graph was a deliberate decision. In the initial phases of the project a user could place a graph anywhere in 3-dimensional space. The 3D tracking by ARKit was less than optimal in this solution. There was substantial drift as the user moved the device around. Limiting graphs to the horizontal planes detected by ARKit proved to dramatically reduced graph drift over time.

Results, Evaluation, and Reflection

As a proof of concept this project was successful. The application can place augmented reality graphs and then view the graphs as data is updated live. It is also able to retrieve data from a data source, which was one of the major requirements for the application. The inherent nature of the application fuses the context and data. The beginnings of this fusion were evident during testing and demos.

When using the application, it is apparent that this is still a prototype. The user interaction to place graphs is relatively limiting. Only being able to place graphs on a horizontal surface detracts from the context fusion experience since it is difficult to place a graph next to its data source. The result is that physically further away from its data source, which reduces context.

While context is added to the data, the augmented reality graphs lose the precision of traditional graph-based solutions. As the graphs move and resize it is difficult to determine what a maximum value is at a point in time. Showing non-time domain data is also not intuitive. Augmented reality works on the principle of time. Objects and devices move over time, and they change their shape or orientation. This meant that the graph solutions that were developed as part of this application didn't convey information effectively about data whose X axis was not time.

Another weakness of the solution is where the graphs are located. Graphs don't show up in the location the user desires to put them. This is an issue with local and relative locations of nested game objects in Unity. I still have not figured out the correct solution to place the axis and the graph object directly on the location selected by the user. I attribute this problem to a lack of experience with the toolset. With enough time this problem will be solvable.

One of the more surprising aspects of the project was where the learning curve was. Most of the technologies used in this project were new to me. I hadn't use ARKit or Unity, however I was familiar with C#. I anticipated more time being required to learn the 3D object manipulation in Unity. The tool and C# API make that aspect relatively painless. Part of the reason is due to the prevalence of training material on Unity's website (Learn - Modules, 2018). Some of the biggest hurdles were with the user interface. Placing components is a non-trivial process. Dealing with device size was also not intuitive. A lot of time was spent figuring out adequate ways to propagate settings changes, like an IP address through the application. In the end I found a solution, but I feel too much contextual knowledge is required by the UI to make it maintainable, long term solution.

Conclusions and Future Work

The goal for this project is to turn it into an application that can be published. To do that there are a several issues that need to be fixed, and few new features need to be added. The first, and most major issue that needs to be fixed is the graph position after placement. As mentioned earlier the graph isn't located where the user selects. This should be fixed shortly. The second issue are the graphics. The images used are stock images from Unity example code. Spending time on polish and graphics will go a long way towards making the application more consumer friendly.

The second set of items in the future work are new features that will make this an application that is more feature rich for consumers. These are predominately features that were excluded because they weren't necessary for a proof of concept. The first feature set that is intended to be added is configuration. The user should be able to configure what channel each axis uses. The user should also be able to easily discover devices to connect to, not just type in an IP address. The second feature level change that is necessary is going to be updating the ARKit version. The new versions offer more tracking options that will make usability more intuitive. Timing prohibited the updates from being incorporated into this version of the application, but it will be in future versions. The last feature set that we want to be added to the project is cross platform support. Currently the application only runs on iOS devices that support ARKit. I would like to see this list grow to include Android devices that support ARCore as well.

Bibliography

Apple Inc. (2018). App Store [Mobile application software]. Retrieved from http://itunes.apple.com

ARKit. (2018). Retrieved January 14, 2018, from https://developer.apple.com/documentation/arkit\

ARKit support for iOS via Unity-ARKit-Plugin. (2017, August 24). Retrieved December 28, 2017, from https://forum.unity.com/threads/arkit-support-for-ios-via-unity-arkit-plugin.474385/

Inter IKEA Systems B.V. (2018). IKEA Place (2.0.1) [Mobile application software]. Retrieved from http://itunes.apple.com

Kenney, R., & Becker, A. A. (2015, April 23). Is the NASDAQ in Another Bubble? A virtual reality tour of the NASDAQ. Retrieved April 3, 2018, from http://graphics.wsj.com/3d-nasdaq/

Learn - Modules. (2018). Retrieved January 10, 2018, from https://unity3d.com/learn/tutorials

NaturalMotion (2018). CSR Racing 2 (1.18.0) [Mobile application software]. Retrieved from http://itunes.apple.com

Resnick, B. (2017, July 03). Visualizing High Dimensional Data In Augmented Reality. Retrieved April 3, 2018, from https://medium.com/inside-machine-learning/visualizing-high-dimensional-data-in-augmented-reality-2150a7e62d5b

Simon Rogers. (2016, June 21). How we made a VR data visualization. Retrieved April 3, 2018, from https://simonrogers.net/2016/06/20/how-we-made-a-vr-data-visualization/

Unity Technologies. (2018). Unity [Computer Software]. Retrieved from https://store.unity.com/

Unity Technologies. (2018). Unity ARKit Plugin [Computer Software]. Retrieved from https://assetstore.unity.com/packages/essentials/tutorial-projects/unity-arkit-plugin-92515