Building a Framework to Support Platform Independent Visualization

Tracy A. Lenuik
Georgia Tech Research Institute
Quantico, VA
Tracy.lenuik@gtri.gatech.edu

Samuel R. Murley
Design Mill, Inc.
Dubuque, IA
Sam_murley@designmillinc.com

ABSTRACT

For complex systems of interest to numerous stakeholders, spanning a broad range of disciplines and perspectives, the necessity and priority of visualized information is personal. Customization of large data sets typically includes elements of searchability, retrieveability, and configuration management; but the visualization of that data is often limited to a designated platform, such as a tablet or mobile phone. Georgia Tech Research Institute and Design Mill, Inc. are partnering to make Platform-Independent Visualization (PIV) a reality for the U.S. Marine Corps. This approach of stakeholder-driven design, rather than platform-driven design, pushes visualization into a new category of customizable information. Authors Lenuik and Murley walk through the development and deployment of a scalable PIV framework to support visualization across any device (desktop, web, mobile, wearable-computing), in both connected and disconnected states.

ABOUT THE AUTHORS

Tracy A. Lenuik is a Research Scientist at the Georgia Tech Research Institute (GTRI). Working out of GTRI's Quantico, VA Field Office, her primary area of focus is supporting the U.S. Marine Corps in Systems Engineering. Tracy serves as GTRI's Visualization Lead for the U.S. Marine Corp's Framework for Assessing Cost and Technology (FACT). She earned a B.S. in Engineering Physics from the Colorado School of Mines and a Professional Masters in Applied Systems Engineering from Georgia Tech.

Samuel R. Murley is the Research and Development Manager at Design Mill, Inc. His specialty is developing software that connects the physical and digital worlds through mobile platforms and 3D technology. Sam is primarily focused on identifying and integrating innovative technology into client applications and creating strategic plans to keep them ahead of the competition. He earned a B.S. in Computer Science from Clarke University and has over 10 years of experience in his field.

Building a Framework to Support Platform Independent Visualization

Tracy A. Lenuik
Georgia Tech Research Institute
Quantico, VA
Tracy.lenuik@gtri.gatech.edu

Samuel R. Murley
Design Mill, Inc.
Dubuque, IA
Sam_murley@designmillinc.com

INTRODUCTION

This document showcases the approach and implementation of a Platform Independent Visualization (PIV) Framework including challenges and efficiencies discovered during design, development and implementation. Visualization has been used for decades in Modeling, Simulation and Training environments but an aggregate framework and visualization platform to connect existing and new data through custom visual representation is missing to support a broad range of use. Georgia Tech Research Institute (GTRI) and Design Mill, Inc. (DMI) are working towards a solution to bring scalable and streamlined PIV to any user on any device including desktop, mobile, web and wearable platforms.

ARCHITECTURE APPROACH

Infrastructure

The architecting of platform independent or cross-platform software development requires taking a generic or object-oriented approach to infrastructure and design. This ensures acceptance across various use-cases and client-side requirements. Figure 1 illustrates a platform independent infrastructure.

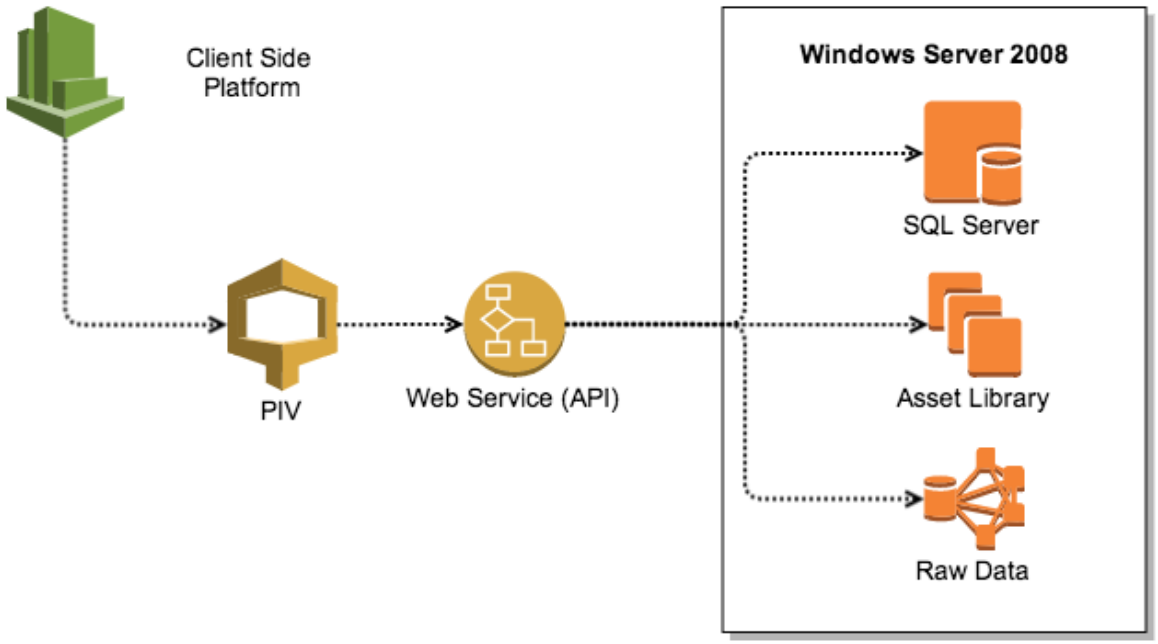

Figure 1. Platform Independent Infrastructure

The infrastructure developed for the platform independent visualization (PIV) framework mirrors a web stack. The system is housed on a hosted SQL Server database, Windows 2008 server with IIS 7, and the front end is built with basic web languages (HTML / JavaScript). The assets library is stored on the same Windows 2008 server for on-demand access from any platform.

A representational state transfer (REST) web service is used to manage PIV data and assets. REST defines a set of architectural principles for designing Web services, which focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. If measured by the number of Web services that use it, in the last few years REST has emerged as a predominant Web service design model. In fact, REST has had such a large impact on the Web that it has mostly displaced SOAP- and WSDL-based interface design because it's a considerably simpler style to use.

Visualization Properties

Visualization is a fairly broad term and it is important to understand the definition of visualization for PIV as well as its properties and components. We are defining visualization as a 3D interactive space where users interact with data-driven 3D models through customizable input controls. This type of visualization consists of the following features: editable 3D models, dynamic parametric data, interactive 3D view, macro-to-micro layering and component selection. In its basic form, the PIV framework loads any 3D model, allows the user to change the design or environment variables, and connects external data to the model, system, or sub-system to support visual changes on the 3D object.

To support current and future hardware platforms, the PIV framework accepts various hardware inputs but generalizes these inputs into consistent results in the 3D space.

Independent Pillars

The PIV framework relies on one or more data types from the independent pillars. Illustrated in Figure 2, the pillars of assets or types are truly independent if created following the PIV Standards & Commonality (discussed below in the *Standardization & Commonality* section). This means that the 3D models can be added without having to worry about where/how to bring the data into the 3D scene. The same can be said when finding and converting data for the PIV; the data is manipulated and brought into the PIV without having to touch the 3D component it is paired with, as long as the data is formatted as described below in the *Standardization & Commonality* section.

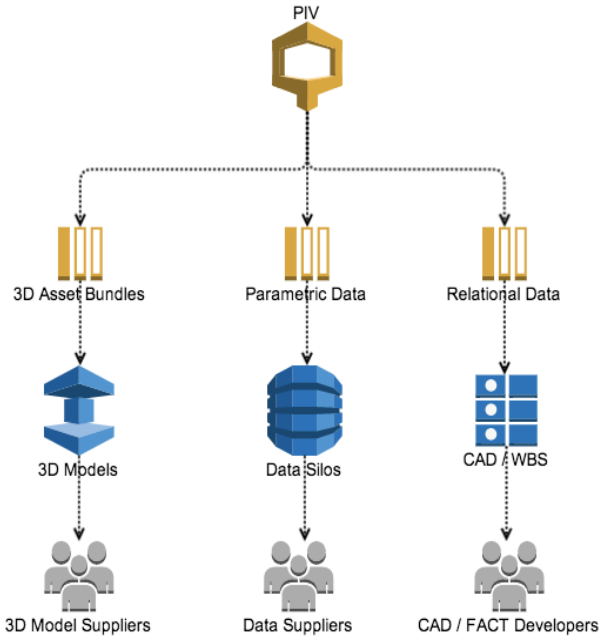

Figure 2. Independent Pillars

This architecture approach allows the PIV pillars to grow without affecting each other. They are then aggregated through the PIV framework where the 3D models are connected to the relational and parametric data. This is *extremely* important and the highlight of the PIV architecture. With this disconnected approach at the lowest level, the pillars scale and grow with minimal or no impact to the other components. The pillars contain existing data or assets and continue to grow through suppliers and developers. Again, if the Standardization & Commonality requirements are followed when developing or submitting to the pillars, the suppliers and developers don't necessarily need to know the intricacies of the other pillars; they can instead focus on their own area of expertise.

To further scale the pillars, the automation of manual processes is implemented to allow the pillars to grow directly from the raw data (CAD, external data, etc.).

The PIV contains these initial pillars to supply editable models to the viewer. However, more pillars can be added in the future or for a different use case, if required, to aggregate other information types to one or more of the existing pillars.

Platforms

Currently, the PIV can be used on web, desktop and mobile devices. The PIV uses a 3D simulation engine that is cross-platform to display the 3D interactive capability and it can be published to iOS, Android, Windows, and any web browser. The PIV framework is structured with a cross-platform approach from a device perspective, meaning it can run on any hardware device running any of the major platforms.

The only change to the system while running on new hardware would involve how the user interacts with the 3D objects as this change occurs across different hardware devices. Web and desktop environments support input from the keyboard and mouse, whereas mobile accepts touch and sensor input. Wearable computing further adds to the input options through head gestures, wrist actions, and a handful of different touch gestures. Our PIV approach to this challenge is further discussed in the *Input Methods* section.

Macro-to-Micro

Regarding the 3D interactive models themselves, a macro-to-micro architecture is created in the PIV framework to give the user the ability to create layers of transparent 3D geometry or remove these layers completely. The term "macro-to-micro" is used in Chemistry to describe the ability to look at an organism in its entirety or to smart-select by moving through its systems to view its smallest component at a molecular level; this concept is incorporated into the PIV framework.

Supporting a macro-to-micro view of visualization, the visualization capability allows the user to view a system, such as a vehicle, in its entirety, its subsystems, or its smallest physical component – the replaceable unit. This type of visualization supplies a form of visual discovery and research to the user without the user having to know specifics about what they are looking for; they just need to know where it is in relation to the rest of the system.

Connectivity and Synchronization

To support use cases where the PIV framework supports a disconnected state, data and asset 3D caching are required to provide the user with the full or partial capability, without connectivity to the back-end server. This type of issue is common, and a simple synchronization protocol is required to pull down location-based information or previously selected information when the user had connectivity. This is required with mobile and wearable devices.

STANDARDIZATION AND COMMONALITY

In order to ensure that the PIV pillars are truly independent along with the user experience / user interface (UX/UI), standardization and commonality must be developed and implemented for each pillar of the PIV framework (e.g., data, 3D models, and rich-media).

This ensures that the framework can be truly cross-platform and that development can be done within PIV pillars without affecting the other pillars or client-side implementation. The following sections outline the standardization and commonality put into place within the PIV framework.

Polygonal Parent-Child Relationship

A structured parent-child relationship is architected to support a linear, macro-to-micro drill-down feature or layering effect. This feature allows the user to select a parent component and visually see children or sub-system components for further discovery. The two-way API communication bridge supports the transmission of currently

focused system and sub-system properties. This ensures that the user can drill-down interactively through the 3D model or through web controls on the parent web page.

This parent-child relationship can be as granular as the work-breakdown-structure (WBS) for any system. In its simplest form, the parent-child rule is as follows: any object that requires independent focus in the 3D space must be a child object of a parent object or group. The only exception to this rule is the parent object (called a group) at the very top of the WBS. This is usually the system object: Vehicle_XYZ, Part_XYZ, etc. Since this is the entry point for the 3D model, this parent object does not need to and can't be a child object of another parent or group unless otherwise required.

A properly architected 3D space, API bridge, and parent web page interface rely heavily on this parent-child relationship to be properly implemented for this phase as well as future adaptations.

Naming Conventions

A standard naming convention for 3D objects, assets, and data-points is authored and implemented to support two-way communication between the parent web page and the 3D space. All objects and materials have a unique name that coincides with the system WBS so that a user can interact with the 3D content and WBS tree seamlessly:

- When the user clicks a component in the WBS, the component in the 3D space is highlighted
- When the user focuses on a component in the 3D space, the web page highlights the component in the WBS

Here is an example of a properly applied naming convention for the GEP Optimizer 6500:

engine_johndoe_designmill_gepoptimizer6500_25

The naming convention includes the following describers separated by underscores:

Type of Object (i.e. Engine)
Author Name (i.e. 3D Modeler Name)
Company (i.e. Contractor name, company name)
Object Name (i.e. Component name)
Suffix (i.e. unique ID number)

ARCHITECTURE DESIGN

Input Methods

The PIV system accepts input methods in an object-oriented way. This ensures that the framework is compatible with current and future hardware devices with minimal development effort. Tables 1 and 2 describe the existing input controls for desktop and web platforms.

Table 1. Mouse Gestures

Name	User Action	Result
Rotate	Mouse Left Button Hold	Rotate vehicle (system) along Y or X axis
Zoom	Mouse Wheel (or equivalent)	+/- Camera field of view
Focus	Mouse Left Button Click	Focus on the clicked object
Isolate	Mouse Right Button Click	100% opacity on selected object and children

Table 2. Keyboard Gestures

Name	User Action	Result
Rotate	R + Arrow Keys	Rotate vehicle (system) along Y or X axis
Pan	LEFT SHIFT + Arrow Keys	+/- Camera Y or X position
Zoom	Plus (+) / Minus (-) Keys	+/- Camera field of view
Isolate	I Key (toggle)	100% opacity on selected object and children
Reduce Transparency	LEFT SHIFT + T Key	Decrease transparency on selected object by -5%
Increase Transparency	T Key	Increase transparency on selected object by +5%

Unity3D

The Unity3D engine is a physics-based simulation engine that utilizes gaming technology to deliver high-end visualization across web, mobile, gaming and desktop platforms. It is a cross-platform game engine with a built-in Integrated Development Environment (IDE) developed by Unity Technologies. It is used to develop simulation games for web plugins, desktop platforms, game consoles, haptic devices, and mobile platforms. It grew from an OS X supported game development tool in 2005, to a multi-platform game engine, within just a few years.

The GTRI/Design Mill team is leveraging the Unity3D capability to develop enterprise visualization. Unity3D calls this "Serious Games" and released the Unity Pro Modeling, Simulation and Training (MS&T) Bundle as an all-inclusive product package with access to Unity's cross-platform capabilities as well as GIS terrain importing packages and the Unity-SCORM (Sharable Content Object Reference Model) Integration Toolkit [1]. The Bundle is aimed specifically at the MS&T community to bring the power of the award-winning Unity platform to non-game markets, such as the Department of Defense (DoD).

Asset bundling is relied upon in the PIV framework and allows separate 3D geometries to be packaged and compiled as a .unity3D assembly from the Unity3D IDE. This assembly is then streamed down from a server location and loaded into the Unity3D web player. The asset bundle compiling process is a crucial step in the procedure to streamline the PIV architecture.

The Unity3D engine is chosen as the optimal simulation engine for the current needs of the United States Marine Corps and future support of high-fidelity 3D visualization, virtual worlds, and virtual environments:

- Unity3D supports the features required for custom configuration from a virtual and collaborative system design perspective
- Unity3D is a physics-based simulation engine
- The Unity Web Player delivers smooth and predictable performance with memory optimization and streaming asset capabilities
- Applications stream large modules, reducing user wait time
- Cross-platform development provides access to the 3D capability on mobile and console devices
- The run-time engine is very flexible, supporting program control of all 3D aspects
- The run-time engine interfaces with legacy and new database management systems (DBMS)
- The run-time engine scales to support streamable environments and virtual worlds
- Unity3D supports networking between viewers to support dynamic training scenarios and learner input
- Unity3D supports common asset file types and handles converted CAD geometry
- Unity3D supports the PIV scalable pillars of visualization that are independent when developed outside of the platform, but are used (or connected) by the PIV architecture as a whole to deliver high-end visualization

The usefulness of visualization is affected by the amount of detail shown. In the case where a system includes thousands of components, sufficient resolution is needed for the user to distinguish between all the components and physical attributes. Table 3 compares the Unity3D plugin with the popular Flash alternative.

Table 3. Unity3D vs. Flash

	Flash	Unity3D
Multiplatform	Web, iOS, Windows, Android, Mobile, Desktop, Linux	Web, iOS, Windows, Android, Console, Mobile, Desktop, Linux
Compiled Size	9223kb	7406kb
Code Execution	1965ms	1600ms
3D Performance	10FPS	49FPS
Development Environment	Web-based	3D / Engine-based
Browser Support	Multi-browser	Multi-browser
Navy Marine Corps Intranet (NMCI) Compatibility	Yes	Yes
Price	$699.00	Free, Pro: $75/month

Both plugins are similar in pricing, Navy Marine Corps Intranet (NMCI) support, and overall browser support. Unity3D on average is more lightweight, runs faster, and has a larger rendering pipeline. Unity3D and Flash are both multi-platform, but Unity3D interfaces with consoles such as haptic devices and high-end simulation hardware.

Due to Unity3D's faster rendering capability and rendering support for large scenes and virtual worlds, it is chosen as the optimal engine for high-end immersive visualization for the U.S. Marine Corps application.

FACT

The U.S. Marine Corp's Framework for Accessing Cost and Technology (FACT) is rapidly gaining acceptance as a decision support tool to help guide conceptual and design decisions for the Acquisition Community (term used broadly here to include the elements of acquisition, sustainment, and evolution). Since visualization is fundamental to how humans think, comprehend, and learn, it is a key consideration for any decision support system.

In response to U.S. Marine Corps requirements, the need for a multi-faceted visualization capability within FACT is met using the PIV to support both current and future 3D visualization technologies. Developing and integrating visualization in this fashion allows for ease of asset management and scalability within the FACT data management system and client user interface. This API protocol is the neck-of-the-hourglass for both storage and retrieval. Figure 3 depicts both sides of the visualization layer. The communication workflow from the visualization layer up to the user interface is shown in the blue triangle. Displaying the user-requested information requires appropriate architecture for retrieving data, assets, and third party software (bottom green triangle).

This protocol takes into account cross-platform and cross-browser support. Future visualization platforms and future technologies are added to this protocol with minimal effort.

The PIV framework that is integrated into FACT provides the following capabilities:

- Supports all types of media and visualization technologies
- Leverages reusable assets throughout all levels of visualization and interactivity

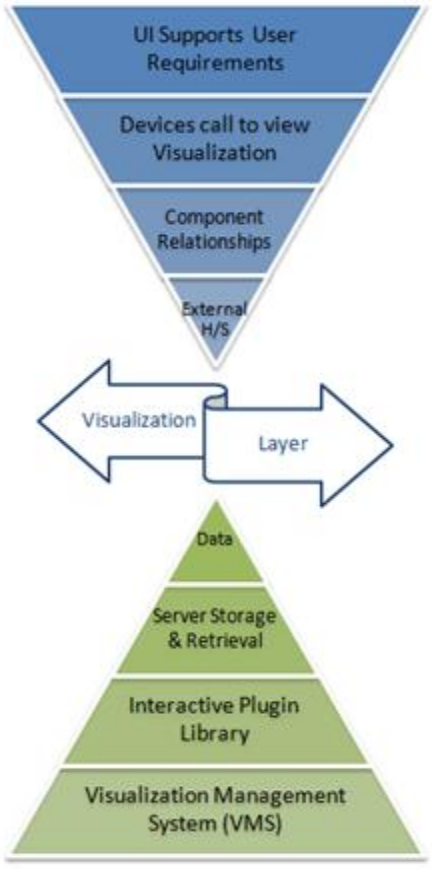

Figure 3. Visualization Layer

- Visualization is viewed from any device or browser
- The visualization layer does *not* take into account end-user functionality requirements and can therefore be reused and leveraged by other DoD software and systems outside of FACT
- Supports two-way communication between the user and visualization data and media
- Future technologies are integrated into the visualization layer with ease
- Data and assets are managed through a Visualization Management System (VMS), independent of FACT
- Integration with other systems and data is accepted through common web communication protocols (web services)

FUTURE OF PIV

With the PIV framework in place, the distribution pipeline can be expanded upon to work with new hardware platforms and technologies. The future of PIV involves discussion and integration with augmented reality, virtual worlds, simulation engines, immersive computing and wearable technology. Using the PIV base architecture as the primary interface, these newer technologies and hardware platforms can leverage and use what has already been implemented with small enhancements to the PIV framework.

Third Party Integration

The PIV framework was built initially for visualization with Unity3D. However, its architecture is open and generic enough to work with other open source simulation and third party game engines. Since the architecture consists of industry and platform-agnostic technology, the front-end engine can be replaced with other game engines such as Unreal, Havok or CryEngine. For high-end simulation, the PIV framework could be expanded to support integration with open source simulation engines such as Simulation Open Framework Architecture (SOFA) and ParaView. Unity3D is the choice front-end engine for networked-visualization on multiple platforms as the IDE is easily extensible to mobile, web, console, HUD and PC. However, high-end simulation analysis would require a more data-intensive software add-on to the PIV.

SOFA and ParaView would add new capabilities to the PIV: decomposition and granular physics-based simulation, qualitative and quantitative analysis and large dataset analysis. For SOFA, integration is possible through the data-layer approaches that support both the PIV and SOFA. SOFA also uses data formatted in XML for simulation input and data submission. The ‘mapping’ concept for task optimization in SOFA can directly translate to the data or relational feature in the PIV, which is being used for visualization and collision detection. Unity3D can simulate fluid dynamics and physics-based visualization in 3D, but chained algorithms to create high-end simulation would be best ported from the PIV into SOFA. ParaView’s architecture leverages similar open standards for data visualization and data could be shared between the PIV and ParaView as well.

Both ParaView and SOFA could leverage Unity3D’s multi-platform approach to visualization. SOFA or ParaView would still take care of any simulation analysis. The lightweight and distributed visualization of the analysis would be accomplished through Unity3D on any platform. To support high-end simulation analysis, a future enhancement of the PIV would involve writing SOFA and ParaView plugins to support the sharing and leveraging of data, 3D assets and visualization to support a wide range of use cases.

Immersive Computing

The current framework relies heavily on input methods and interaction from the user in order to respond in the 3D space with a result. With immersive computing (Figure 4), the input methods can expand to accept sensory data such as voice commands, hand gestures, location-based information and Near Field Communication (NFC) recognition. Immersive Computing outlines the types of immersive computing inputs that can be integrated into the PIV to enable the architecture to work with camera recognition, voice and sensory data in an immersive environment.

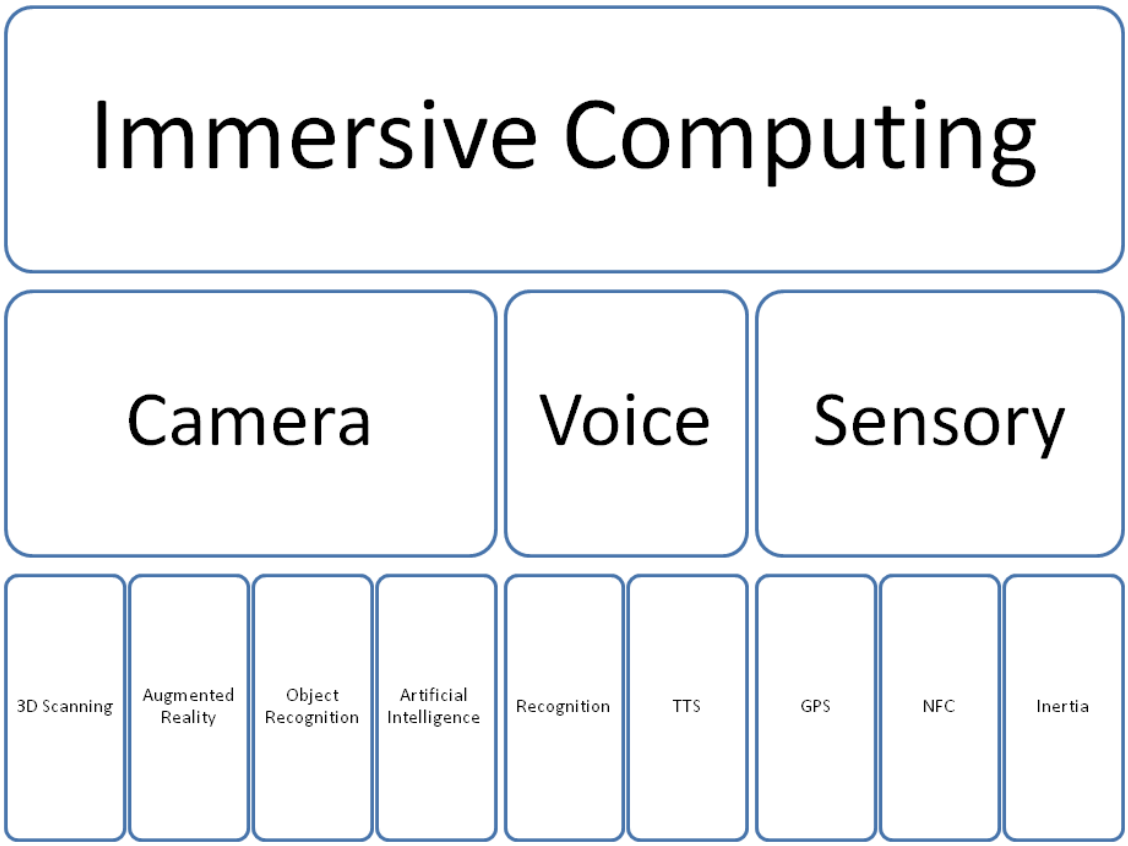

Figure 4. Immersive Computing

Augmented Reality (AR)

The PIV framework is developed with augmented reality (AR) and heads-up-display (HUD) devices specifically in mind. The 3D assets, data, and rich-media development standardization conform to AR capabilities. The Unity3D engine would also be leveraged to deliver streaming 3D data to the device as it is now on web and desktop platforms. AR is defined as layering digital data onto the real-world. This is done through facial, image, location-based and object-based recognition technology.

The only new PIV pillar that would have to be added is the Recognition Pillar. This pillar would be responsible for storing edge-based object detection data so that when a HUD or mobile device makes a match on a real world object, the match is confirmed and the correct data is supplied to the user's device based on the use-case. Submitting new object, location-based, and image data to this pillar would be available to non-technical users to grow the real-world recognition capability.

Augmented reality has seen a high adoption rate in the modeling, simulation and training industries over the past two years and is the natural next-step in the evolution of the PIV framework. With this progression, users can point their mobile device or HUD at an object and get real-time information based on the object or component they are viewing. Real-information could include how to replace a particular part with 3D step-by-step instruction, collaborative design ideas overlaid onto the physical system, and mission communication information. One of the biggest values of AR is the quick transmission of knowledge and the user's environment. Information is provided automatically based on what the device is recognizing and multiple manual steps to acquire data become obsolete.

AR will continue to grow and evolve in the coming years. The GTRI/DMI team recognized this and took this into account when developing the PIV framework to further prepare the framework to deliver AR data on mobile and wearable devices.

CONCLUSION

We believe our PIV framework is unique in that it's both a light-weight system as well as a very scalable framework in regard to emerging use cases and new technologies. The framework is currently functional on web platforms but with lithe rework can be pushed to almost any operating system (OS) or platform, including gaming consoles, mobile, heads-up displays (HUD), and wearable devices. During this process we also incorporated many efficiency and commonality standards to bring existing legacy data and assets into this new framework. Future efforts will focus on building out a management-based platform, allowing users to upload assets such as CAD, 3D models, rich-media and data without having to go through a manual conversion process. The assets are converted by the PIV automation protocols and added to the appropriate repository. Our goal for the initial development of this framework was to create a reusable architecture that was light-weight and easily extensible into other systems and use-cases, and we believe we have accomplished this short-term goal.

ACKNOWLEDGEMENTS

The authors wish to thank the Marine Corps Systems Command (MCSC) for funding the FACT Visualization effort. FACT is under the leadership of Mr. James Smerchansky, Director, Systems Engineering, Interoperability, Architectures & Technology (SIAT), and under the direction of Mr. Michael O'Neal, Modeling & Simulation (M&S) Lead, SIAT, and Mr. Luis Velazquez, Deputy, M&S Lead, SIAT.

REFERENCES

[1] Unity Technologies. (n.d.). Create serious games with Unity. Retrieved from http://unity3d.com/company/sim