The Bespoke 3DUI XNA Framework: A Low-Cost Platform for Prototyping 3D Spatial Interfaces in Video Games

Paul D. Varcholik*
Media Convergence Laboratory
Institute for Simulation & Training
University of Central Florida

Joseph J. LaViola Jr.†
School of Electrical Engineering
and Computer Science
University of Central Florida

Charles Hughes‡
School of Electrical Engineering
and Computer Science
University of Central Florida

Abstract

This paper presents the Bespoke 3DUI XNA Framework, an open-source software platform for research in 3D user interaction. The Bespoke 3DUI XNA Framework distinguishes itself from other platforms, in that it provides 3D user interface machinery in a game development framework. This combination leverages low-cost, widely available game technologies, enabling researchers to investigate 3DUI techniques, and providing game developers a foundation for prototyping 3DUIs in commercial video games.

The paper explores the functionality and utility of the software library and describes how researchers and game makers can leverage the platform to investigate 3D user interfaces in the context of prototypical interactive experiences.

Keywords: 3DUI, user interface, framework, game development

1 Introduction

3D user interfaces [Bowman 2004] give users the ability to spatially interact with 3D virtual worlds because they provide natural mappings from human movement to interface controls. These interfaces, common in virtual and augmented reality applications, give users, rich, immersive, and interactive experiences that can mimic the real world or provide magical, larger than life interaction metaphors [Katzourin 2006].

With the latest generation of video game hardware, 3D user interfaces are emerging as the gaming interaction paradigm of the future. The popularity of devices such as the Sony Eye-Toy, the Nintendo Wii, and 3D DLP HDTV are making it possible for gamers to interface with video games using 3D spatial input, and for developers to leverage concepts from virtual and augmented reality such as head tracking and stereoscopic vision. However, developing software for these interfaces can be a daunting process for newcomers due to hardware inaccessibility and a lack of 3DUI software tools. Thus, investigators must overcome a number of technical challenges to develop a 3DUI platform before they can begin research in this area.

Through an extensive literature review, and collaboration with 3DUI experts, we have compiled a set of requirements necessary for a 3DUI research platform. These requirements can be categorized into three areas: 1) high-level non-functional requirements; 2) primary components essential for basic 3DUI research; and 3) secondary elements necessary for longer-term research efforts. These categories include:

* pvarchol@ist.ucf.edu
† jjl@cs.ucf.edu
‡ceh@cs.ucf.edu

High-Level Non-Functional Requirements

- Commercial-off-the-shelf (COTS) hardware
- Open-source software
- Modern programming language, accessible to novice software developers
- Broad application
- Extensibility

Primary Components

- 2D/3D graphics rendering
- 6DOF head tracking
- 3D motion controller (ala Nintendo Wiimote)
- Stereoscopic rendering

Secondary Components

- Scene management
- Content pipeline with support for common scene elements (e.g. 3D models, animation, terrain)
- 3D audio
- Network support (e.g. multiplayer services)
- 2D/3D Gesture recognition
- Physics
- Motion tracking (of multiple, non-head points (e.g. hands, body or object)
- Chroma-key extraction
- Head-mounted displays
- 2D UI widgets (e.g. menus, buttons)
- Recording user experience
- Cross-platform communication

Notably, the requirements call for a generic platform for creating virtual environment but with a specific set of technologies for interacting with those worlds. In short, these requirements describe a game engine geared toward the input devices and displays involved in 3DUI development.

This paper presents the Bespoke 3DUI XNA Framework, an extensible, open-source software library developed to meet many of these requirements, thereby enabling the rapid creation of sophisticated 3DUIs. A detailed discussion is provided on design decisions, compatible hardware, software architecture and the employment of the framework.

The work presented has been adopted by a number of university colleagues, members of the open-source community, and as the development platform for a graduate course in 3DUIs at the University of Central Florida. This paper discusses feedback obtained from users of the framework as insight into the usefulness of the system.

2 Related Work

The idea of providing an open-source framework for supporting 3D spatial interaction research and development is not new. There have been many different software frameworks and toolkits, such as the SVE toolkit [Kessler et al. 2000], VR Juggler

[Bierbaum et al. 2001], DIVERSE [Kelso et al. 2003], Studierstube [Schmalstieg et al. 2002], ARToolKit [Kato 1999], and the VARU Framework [Irawati et al. 2008], that aid in developing spatial 3D interfaces. However, these frameworks and toolkits were designed with more conventional VR and AR applications in mind. Thus, they lack the video game specific development tools that the Bespoke 3DUI XNA Framework provides.

There are several open source game toolkits available that make it easier to build video games by providing important infrastructure components such as 3D rendering, asset management, sound, event handling, scene graph support, and physics simulation. Examples of these types of toolkits include the Microsoft XNA Game Studio [Microsoft 2009], Panda3D [Goslin 2004], and Delta3D [Delta3D 2009]. Although these game development environments provide sophisticated tools and support for developing video games, they generally do not focus on 3DUI and virtual reality-based games.

One development framework that is closest in spirit to the Bespoke 3DUI XNA Framework is Goblin XNA [Oda 2007]. Goblin XNA is a framework for research on 3D user interfaces, including mobile augmented reality and virtual reality, with an emphasis on games. It is written in C# and based on the Microsoft XNA platform [Microsoft 2009]. Goblin XNA has many similarities to the Bespoke 3DUI XNA Framework in terms of using XNA as its underlying platform and supporting head tracking and 3D spatial interfaces. However, Goblin XNA is primarily focused on augmented reality games while the Bespoke 3DUI XNA Framework targets virtual reality-based games using 3D TVs and monitors and 3D motion controllers. To the best of our knowledge, the Bespoke 3DUI XNA Framework is one of the first development environments to provide comprehensive support for both 3D spatial interfaces and video game creation.

3 The Bespoke 3DUI XNA Framework

The Bespoke 3DUI XNA Framework is organized as a collection of .NET assemblies, application samples, and documentation; and is packaged within a Windows installer for easy distribution and inclusion of third-party dependencies. The Framework is written in the C# programming language and targets the .NET 3.0 Framework and .NET 2.0 runtime. As the name implies, the Bespoke 3DUI XNA Framework is built on top of the Microsoft XNA platform. XNA is a set of software libraries, tools, and community resources "focused on enabling game developers to be successful on Microsoft gaming platforms" [Microsoft 2009]. We chose C# and XNA to match our requirement of a system that utilized a modern programming language, and one accessible to novice software developers. We have anecdotal evidence, from many years of instruction on software development, that C# does indeed meet the criteria of a modern language that is approachable by novice programmers; and yet the environment is powerful enough to support real-time interactive simulation and games. Moreover, since XNA's initial release in December 2006, thousands of games have been created with the platform; including commercial games that have been released on Microsoft's Xbox LIVE Arcade service [Fristrom 2008]. We can confidently state that the XNA platform delivers a solid development environment for students and hobbyists as well as for professional game makers.

The Bespoke 3DUI XNA Framework is an open-source product, distributed with a complete set of source code and a significant number of application samples.

For non-3DUI-specific game development, the Bespoke 3DUI XNA Framework has no hardware requirements beyond those of XNA (which are primarily concerned with a graphics card that supports 3D acceleration and pixel shaders). When developing 3DUI applications, the Framework supports particular hardware for stereoscopic rendering, 6DOF head tracking, and 3D motion control. Figure 1 pictures a development workstation and typical hardware employed for creating games and 3DUIs with the Bespoke 3DUI XNA Framework. The specific hardware components and their corresponding Framework elements are described in the sections below.

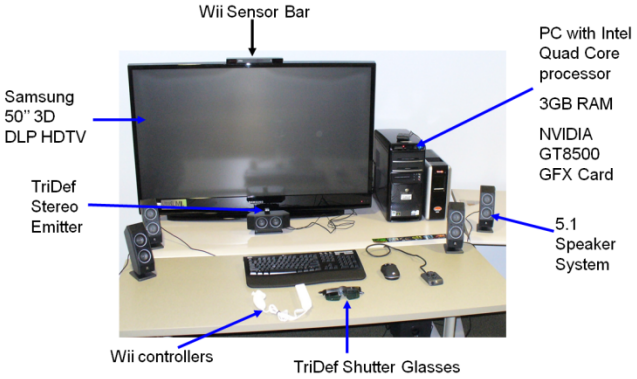

Figure 1: Development workstation

3.1 Stereoscopic Rendering

For stereoscopic rendering, we targeted the 3D DLP HDTV technology from Texas Instruments [Texas Instruments 2009]. This technology, incorporated into television sets by Mitsubishi and Samsung, generates independent views for the left and right eyes. The television is optically synchronized with a pair of shutter glasses worn by the viewer, and supplies images at 60Hz per eye (equivalent to 120Hz). To present the viewer with a 3D image, the video source must supply the television with an image stream following Texas Instrument's 3D Image Format [Hutchison 2008]. This format calls for the left and right eye images to be masked with alternate checkerboard patterns and then combined into a single image before transmission to the television. The 3D DLP technology correspondingly samples the interleaved image, reconstructing the left and right eye images for subsequent display.

The Bespoke 3DUI XNA Framework supports the 3D DLP HDTV technology through:

1. A stereoscopic camera component
2. Independent render targets
3. Shader-based image masking

The stereoscopic camera component defines the concept of a left and right eye to correctly present the virtual camera's view and projection matrices from the corresponding perspective. The eyes are separated by a user-configurable interpupillary distance. Rendering a complete frame in 3D requires that the scene be drawn twice: first from one eye's perspective and then the other. Each perspective is rendered, not directly to the screen, but to independent render targets whose outputs are bitmaps that are fed to the image masking process. The image masking is offloaded to the graphics processing unit (GPU) with a simple pixel shader written in the High-Level Shader Language (HLSL) and using the shader model 3.0 specification. The masking shader interleaves the left and right eye images and presents the final image to the display.

By implementing render targets and shader-based image masking, the Bespoke 3DUI XNA Framework is able to transform any scene into 3D. The masking process runs at 800 to 200 fps for resolutions of 800 by 600 to 1920 by 1080, respectively, thereby having only negligible influence on overall frame rates.

There are a number of other considerations when rendering to this technology. First, the PC must connect to the television using an HDMI or DVI video cable. This is required, because the resolution must be 1920 by 1080 (1080p) to render in 3D. Additionally, video drivers must be configured so that they do not scale the display, nor should an application scale the image (e.g. by embedding the game in a resizable window). Any scaling will misalign the masking during the television's sampling and image reconstitution. Therefore, applications must run in full-screen mode.

At the Interactive Systems and User Experiences Lab (IS&UE) at the University of Central Florida, we have employed ten Samsung 50" 3D DLP HDTVs with Shutter Glasses from TriDef (pictured in Figure 1). Each television costs approximately $1,500 with an additional $200 for a 2-pack of 3D shutter glasses, software, and IR sync cable/emitter.

3.2 Head Tracking

Head tracking refers to the ability to track the position and orientation of the user's head [Bowman 2004]. This information can be used as input, commonly as a means to manipulate the virtual camera. We have adopted a commercial-off-the-shelf, 6DOF optical head tracking solution from Natural Point called the TrackIR [Natural Point 2008]. Pictured in Figure 2, the TrackIR mounts atop the display and detects a triangular array of infrared points within a 46° field of view. The IR points can be actively transmitted to the TrackIR, or the TrackIR can emit IR for use with a retro-reflective clip. The device operates at 120fps.

Figure 2: NaturalPoint TrackIR and retro-reflective clip [Natural Point 2008]

The TrackIR is programmatically accessible through a Component Object Model (COM) assembly provided free of charge from Natural Point. COM components can be accessed through the .NET Interop service and treated as native code elements by the Bespoke 3DUI XNA Framework. To make the device accessible to XNA, a small wrapper class encapsulates the functionality and makes the TrackIR behave similarly to the native XNA keyboard, mouse, and gamepad input devices. Additionally, the Bespoke 3DUI XNA Framework provides a virtual camera that is bound to the TrackIR input; giving game developers an instant option for a first-person, head-tracked camera. The framework also exposes a user-configurable sensitivity value for non-isomorphic mapping.

The TrackIR is sold for ~$130, making it a fairly low-cost solution for 6DOF head tracking.

3.3 3D Motion Controller

With the recent popularity of the Nintendo Wii, 3D motion control has garnered considerable attention from simulation and game developers. Game makers now have strong commercial motivation to explore spatial interaction, and can do so cost effectively with commercial-off-the-shelf input devices such as the Nintendo Wiimote.

Though initially designed for use on the Nintendo Wii, the Wiimote has been adapted for use on the PC. The Wiimote connects to the PC using the Bluetooth communication protocol and transmits data at 100fps. The device includes accelerometers for detecting forces applied to three axes (illustrated in Figure 3).

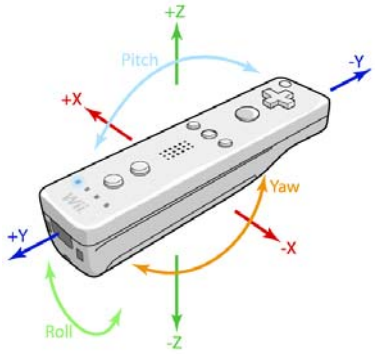

Figure 3: Nintendo Wiimote [Troillard 2008]

The open-source community has embraced the Wiimote, having reverse engineered the messaging protocol, and has provided a number of software libraries for PC-to-Wiimote communication. We have adopted one of these libraries, Brian Peek's Managed Wiimote Library [Peek 2009], for use with the Bespoke 3DUI XNA Framework. Brian Peek's library is a general-purpose package that allows a Wiimote to be accessed by any .NET assembly, not simply XNA. The Bespoke 3DUI Framework wraps the Wiimote functionality and makes it available in a manner consistent with other XNA input devices.

The Wiimote also includes seven input buttons, a digital directional pad, a speaker, vibration system, and an infrared camera, all of which are accessible through the Wiimote components incorporated into the Bespoke 3DUI XNA Framework. The Wiimote data is associated with a player enumeration for supporting up to four simultaneous Wiimotes; and the 3-axis accelerometer data is augmented with timestamp information for use with the gesture recognition system (described further in the next section) [Varcholik 2008].

While the Wiimote is a very capable and inexpensive spatial input device, it has a few limitations. First, the Wiimote is incapable of providing directionality with the accelerometers alone. For example, if one were to place the Wiimote upright on a table, in its natural position with the buttons facing the ceiling, the accelerometers alone could not detect the angle of rotation around the vertical axis. Likewise, the accelerometers cannot detect if the Wiimote is closer to or farther away from the user. In other words, the Wiimote's accelerometers cannot detect specific pointing angles or distance. This is where the Wiimote's infrared camera comes into play. With two neighboring IR source points, the Wiimote can determine distance from the IR source and relative orientation.

Another consideration is that the device must be in motion for the accelerometers to be used for gesture recognition. While it is possible to create one non-moving gesture per orientation, a gesture recognition system would not be able to disambiguate static poses.

In summary, the data collected from the Wiimote's accelerometers corresponds to forces applied to three axes: longitudinal roll (motion along the X-axis), pitch (motion along the Y-axis), and the upright/upside-down orientation of the Wiimote (vertical motion along the Z-axis). The Wiimote is a relative motion device, in that there is no innate sense of absolute position in space. To root the Wiimote, at least with respect to distance and directionality, the IR camera must be employed. However, the developer must consider the strong likelihood of camera occlusion, particularly when using the Wiimote as a full 3D motion input device.

3.4 Gesture Recognition

One of the key features of the Bespoke 3DUI XNA Framework is the gesture recognition system; a machine-learning platform that enables game makers to create an arbitrary set of 3D gestures and map them to gameplay elements.

The recognition system is separated into two primary components: the trainer and the classifier. The training component collects accelerometer data from the Nintendo Wiimote and associates the data with a user-defined label identifying the gesture. A gesture *sample* is the accelerometer data collected between the press and release of the Wiimote's *B* button (the trigger button). Any number of samples can be used to train a gesture, but our experimentation shows good accuracy (> 94%) with as few as ten samples per gesture [Varcholik 2008]. A set of 29 features is used to linearly separate the gestures in the set. These features are extensions of work by Rubine [1991] on 2D symbol recognition, and are further described in [Varcholik 2008]. Trained gestures can be serialized for reuse. Notably, no computed feature or weighting information is stored during serialization – only the accelerometer data. This allows for alternate machine learning algorithms without invalidating previously trained gestures.

The classification component records an unlabeled sample and attempts to match it against a trained gesture. Classification can be accomplished in 10ms or less, even on modest computing hardware (our development machine was a laptop with a 1.8Hhz Intel Core2 Duo and 2GB of RAM).

The gesture recognition system supports both one- and two-handed gestures. For two-handed gestures, the user can train the data simultaneously, utilizing the same *B* button start/stop mechanic for each hand.

The system uses a linear classifier to make its recognition decisions. We have also experimented with an AdaBoost implementation [Freund and Schapire 1997], and an artificial neural network (ANN) evolved using NeuroEvolution of Augmenting Topologies (NEAT) [Stanley and Miikkulainen 2002]. Our unpublished experiments show slightly better accuracy for AdaBoost over the linear classifier, but at the expense of higher training cost. The ANN implementation, with the topology and node weights evolved using NEAT, had the worst accuracy and highest training cost. The experiment was conducted using three gesture sets and with two sample sizes per set. We hypothesize that the ANN/NEAT implementation performed poorly because of the small amount of training data (ten and twenty samples per gesture). These results are preliminary and not rigorously analyzed so we cannot draw any formal conclusions. Informally, we note that the linear classifier, included with the Bespoke 3DUI XNA framework, can train in almost real-time with ten gestures in the set and ten samples per gesture; and can classify in real time with accuracy greater than 94%.

We have employed this system for computer mediated, human-to-human communication [Varcholik 2008] and for human-robot communication [Varcholik 2008]. The recognition system has also been extended to support 2D symbol recognition for multi-touch surface interaction [Varcholik 2009].

3.4 Game Engine Features

What is distinct about the Bespoke 3DUI XNA Framework is that it provides 3D user interface machinery in a game development framework. This combination enables academic researchers to investigate 3DUI techniques while leveraging game technologies, and provides game developers a foundation for prototyping 3DUIs in commercial video games. Thus far we've discussed some of the 3DUI specific functionality of the framework. In this section, we describe some components of the library that are commonly found in game engines.

As previously stated, the Microsoft XNA platform is a set of software libraries and tools designed to enable the creation of games on Microsoft gaming platforms. The software libraries support 2D/3D graphics rendering, 3D sound, multi-player networking, an extensible content pipeline, and include a comprehensive set of game-related helper classes. However, XNA should not be classified as an actual game engine. A game engine generally includes higher-level constructs such as scene management, object interaction, animation, graphical user interfaces, and artificial intelligence. Often, a game engine provides rendering through a graphics API such as Direct3D or OpenGL. A game engine is commonly considered *middleware* in that it sits between the rendering platform and the game code itself. XNA is built atop Microsoft DirectX, and in this respect acts as *middleware* between a game and the rendering system. However, XNA remains fairly low-level in the functionality that it provides, and is thus better compared to OpenGL than to games engines such as *id Software's* Quake III [id Software 2009] or Epic Games' Unreal [Epic Games 2008].

The Bespoke 3DUI XNA Framework extends the XNA platform to provide game engine features including those listed in Table 1.

Feature	Description
Virtual Cameras	An abstraction of the View and Projection matrices required for rendering, coupled with specialized attributes (e.g. chase camera, stereoscopic camera, orthographic camera).
Input Devices	Componentized abstractions of input devices including: mouse; keyboard; gamepad; Wiimote; and the TrackIR, which generally provide frame-to-frame history for relative tracking.
Actors	Scene object abstractions for dynamic and non-dynamic objects that generally have an associated 3D model.
Animation	Animated actor.
Menus & GUI	2D elements for game state transitions and in-game graphical user interfaces (e.g. buttons, fonts/strings, and menus).
Scene Loading	XML serialization of scene elements (e.g. actors, cameras, UI elements) for non-programmatic scene population. Allows for in-game scene reloading.
Scene Management	An abstraction of the active scene that provides an entry and access point for game objects. Encapsulates the primary render and update threads.
Particle System	3D particles for producing effects (e.g. explosions, smoke, projectiles)
Terrain	2D texture tiling over a height map
Skybox	5-sided texture mapped cube for scene background – floor removed.
Post Processing	Effects system applied after the scene has been rendered but before final display. Common effects: bloom, god rays, blur
Windows Forms	Allows an XNA application to be embedded within a Windows Form.
World Editor	Enables the dynamic compilation of XNA assets for WYSIWYG scene creation and asset previewing.
Cross-platform Communication	Generic network communication mechanism useful for serializing input device data to remote clients.

Table 1: Game engine features

A few of these components are particularly noteworthy: the scene management system, the world editor, and the cross-platform communication component. The scene management system provides an organizational element that is otherwise absent from XNA. Specifically, the scene manager acts as a collection point for all of the objects in the game and triggers the rendering of and interaction between those objects. Scene objects are actors, 2D UI elements, terrain, sky boxes, and sprites. These elements are added to the scene management system programmatically or via the XML scene loader. Scene objects can be disabled so that they are not considered during the update and render threads, or can be made invisible so that they are updated but not rendered.

The world editor, pictured in Figure 4, is a WYSIWYG scene creation tool built on top of the Bespoke 3DUI XNA Framework. As such, the editor uses the same rendering system as games authored using the framework. This is essential for accurately previewing the look of a 3D model as it will appear in-game. The scene author first populates a content library – a categorized list of source assets that can be compiled into game-ready content. Source assets come from a digital content creation (DCC) package in a format that generally contains more information about the asset than can be used in-game. *Compiling* the source asset into a game-ready format means that we process the asset through the XNA content pipeline, extracting only that information that is useful at runtime and writing it into a usable format. Ordinarily, XNA asset compilation is performed along with source code. Moving this process to the world editor allows the user to compile large sets of assets out-of-band. Content libraries can be serialized and loaded into the world editor for easy reuse.

Once an asset is included in the content library and compiled, it can be added to a scene. Scene objects can be selected through a mouse click, or through a categorized scene explorer window. Once selected, objects can be transformed through chorded keyboard-mouse movements in a manner similar to popular 3D modeling packages such as 3D Studio Max or Maya. An object's attributes can also be manipulated through a Visual Studio-like properties window. Indeed, the world editor borrows heavily from the dockable window scheme of Visual Studio. Scene's can be serialized to an XML document for in-game loading.

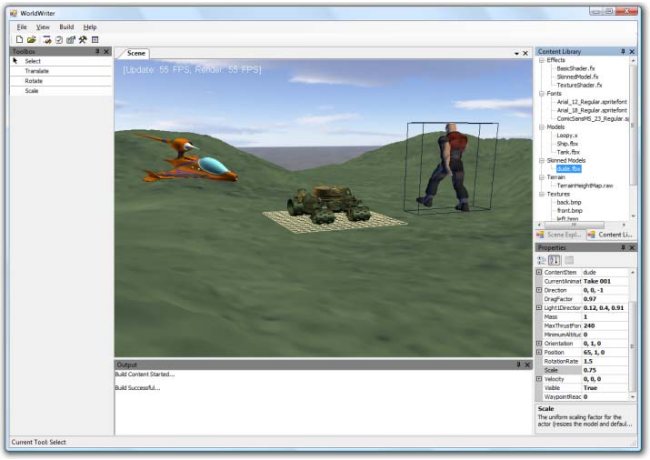

Figure 4: World Editor

The last game engine feature, discussed in more depth, is cross-platform communication. XNA supports the PC, Xbox 360, and the Microsoft Zune. The Bespoke 3DUI XNA Framework extends support to non-Microsoft environments such as MacOS or Linux through the Bespoke Open Sound Control (OSC) Library [Bespoke Software 2008]. OSC is an open, lightweight, message-based protocol that enables, for example, input device data to be transmitted over the network. The Bespoke OSC implementation uses UDP/IP as the transport protocol, and includes support for unicast, broadcast, and multicast. A remote application, therefore, need only support the OSC protocol and network communication via UDP; and need not be hosted on a Microsoft operating system or written in C#. Applications of the Bespoke 3DUI XNA Framework can use this mechanism for transmitting Wiimote and head-tracking data to remote clients.

As a final note on hardware and operating systems we note that, while XNA supports the Xbox 360 and Microsoft Zune, only subsets of the .NET Framework Class Library (FCL) are available on those platforms. Additionally, of the three XNA-supported platforms, only the PC is capable of accessing external hardware. As such, the Bespoke 3DUI XNA Framework must operate on a PC in order to support devices such as the Nintendo Wiimote and the NaturalPoint TrackIR.

4 Case Studies

The Bespoke 3DUI XNA Framework has been utilized as the development platform for a graduate course in 3DUIs at the University of Central Florida. In the context of that course, a number of students have published papers on games and techniques developed using the Framework. This section discusses two of these projects, pictured in Figure 5: RealDance [Charbonneau 2009] and an exploration of menu techniques using a 3D game input device (the Nintendo Wiimote) [Chertoff 2009]. These projects, and several others, were demonstrated at the 2008 Microsoft Faculty Summit [Microsoft 2008].

Figure 5: Projects created with the Framework: RealDance (left), Menu Techniques (right)

RealDance, is a rhythm and dance game in the spirit of games like Dance Revolution and Guitar Hero. The user is equipped with four Nintendo Wiimotes strapped to the arms and legs. A green icon signals a punch, and a purple icon a kick; with the icons displayed on the left or right side of the screen to indicate which arm/hand to use. A video of the song is played in the background and scoring is based on how accurately the user matches his motions with the song. In addition to the game itself, the two student developers designed a song creation system to associate motion icons with a song, and created adjustable straps to secure the Wiimotes to the arms and legs. They also employed a novel menu navigation system, using gesture recognition, whereby the user punches to the left and right to move through the menu, and claps their hands to make a selection. The game includes four popular songs with motion tracks that vary in difficulty.

The Menu Techniques project examined the efficacy of linear versus radial menus when using a 3D spatial input device such as the Nintendo Wiimote. To this end, the two-student development team created a simple game environment that asked the user to uncover a hidden artifact through the successive selection and application of digging "tools" accessed through a hierarchical

menu. The students ran a within-subjects experiment with twenty subjects to compare their menu techniques.

These case studies represent both the entertainment and the research capacity of the Bespoke 3DUI XNA Framework. Note that each project was developed in approximately four weeks (the amount of time each two-person group had for their final project in the 3DUI course).

5 Framework Evaluation

As of this writing, the Bespoke 3DUI XNA Framework has been publicly available for roughly sixteen months and is in its fourth major revision. These revisions have been created in direct response to feedback we have received on the Framework. Specifically, we have solicited feedback from the students of the two semesters of the 3DUI course and we have created an online survey for collecting feedback from external adopters. This feedback is presented here informally, but generally states that the Framework is useful, interesting, and easy to use. Figure 6 shows the aggregated feedback, of eighteen respondents, to a seven-level Likert scale concerning their overall reaction to the Framework.

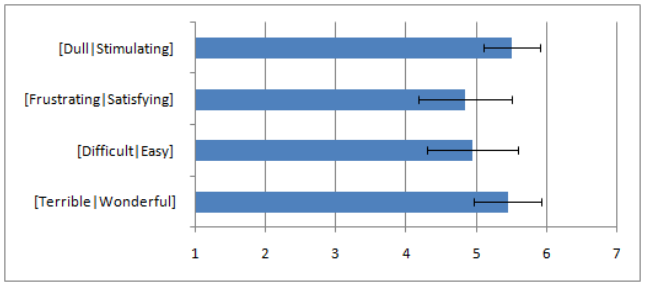

Figure 6: Feedback responses

Other questions allowed for free-form responses on the strengths and weaknesses of the Framework and what users would like to see added. In general, users of the Framework thought that the platform provided many of the common components that are required for game development; and that having the Framework let developers concentrate on their game/research instead of the underlying technology of a game engine. Indeed, comments on the strengths of the Framework overwhelmingly support the contention that this platform helps users to quickly get games up-and-running.

Feedback on the weaknesses of the system revealed two major themes: sparse documentation, and the lack of a physics system. Documentation is provided through a Microsoft help file (.chm) and a variety of code samples. The negative feedback indicates the need for more effort in documenting the system and providing more detailed tutorials. Indeed, many of the feedback respondents requested *cookbook-style* tutorials that would quickly demonstrate the use of a common feature.

A physics system is a game engine component that simulates Newtonian physics and interactions between game objects. XNA does not provide a native physics system, and this is absent from the Bespoke 3DUI XNA Framework as well. Simple bounding sphere collision detection is included in the Framework, but more sophisticated collision detection and dynamic simulation is left to the user to implement. This is a focus for future work on the system.

6 Conclusions and Future Work

With the latest generation of video game hardware, 3D user interfaces are emerging as the gaming interaction paradigm of the future. Researchers and game makers require tools to enable the development of interesting 3DUI techniques. Many commercial and open-source video game creation packages exist as do several 3DUI frameworks. However, there are few development platforms that combine 3D user interface machinery in a game development framework. The Bespoke 3DUI XNA Framework provides this combination and enables academic researchers to investigate 3DUI techniques while leveraging game technologies, and provides game developers a foundation for prototyping 3DUIs in commercial video games.

This paper has described the Framework: its organization, 3DUI functionality, and game engine features; and the requirements that drove the development of the system. We have discussed a graduate course in 3DUI that has adopted the Framework, and have detailed two student-created projects that have used the Framework for entertainment and 3DUI research. Furthermore, we have informally presented an evaluation of the Framework that has revealed strengths and weaknesses of the system, and has provided direction for future work.

Future work on the Framework will include improved documentation and tutorials, and will integrate a full-fledged physics system for collision detection and dynamic simulation. Additionally, the Framework does not satisfy all of the requirements that we identified for a 3DUI game development research platform. In particular, we intend to add support for augmented reality research through chroma-key extraction, head-mounted displays, and fiducial tracking. We will also add the ability to record the user's experience during a simulation, and include support for hand and body motion tracking. Finally, we will add support for the Wii Balance Board and the upcoming WiiMotion Plus – hardware peripherals that offer even more options for exploring user interfaces and gameplay mechanics.

The Bespoke 3DUI XNA Framework is available for download at http://www.bespokesoftware.org/3DUI.

7 Acknowledgements

This work is supported in part by IARPA, SAIC, and by the National Science Foundation under award number DRL0638977.

References

BESPOKE SOFTWARE 2008. The Bespoke Open Sound Control Library. Available from: http://www.bespokesoftware.org/osc/.

BIERBAUM, A, JUST, C., HARTLINK, P., MEINERT, K., BAKER, A. And CRUZ-NEIRA, C. 2001. VR Juggler: a virtual platform for virtual reality application development. In *Virtual Reality, 2001. Proceedings. IEEE*, 89-96.

BOWMAN, D. KRUIJFF, E., LAVIOLA, J. And POUPYREV, I. 2004. *3D User Interfaces: Theory and Practice*. Addison Wesley.

CHARBONNEAU, E. MILLER, A. WINGRAVE, C. AND LAVIOLA, J. 2009. RealDance: An Exploration of 3D Spatial Interfaces for Dancing Games. In *Proceedings of the IEEE Symposium on 3D User Interfaces*, 2009.

CHERTOFF, D. BYERS, R. AND LAVIOLA, J. 2009. An Exploration of Menu Techniques using a 3D Game Input Device. In *Proceedings of the Foundations of Digital Games*, 2009.

DELTA3D 2009. Delta3D: Open Source Gaming & Simulation Engine. Available from: http://www.delta3d.org/.

EPIC GAMES 2008. Unreal Technology. Available from: http://www.unrealtechnology.com/.

FRUEND, Y. AND SCHAPIRE, R.E. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences 55*, 119-139.

FRISTROM, J. 2008. Postmortem: Torpex Games' Schizoid. Available from: http://www.gamasutra.com/view/feature/3796/postmortem_torpex_games_schizoid.php.

GOSLIN, M., MINE, M. 2004. The Panda3D Graphics Engine. *Computer 37*, 112-114.

HUTCHISON, D. 2008. Introducing DLP 3-D TV. Available from: http://dlp.com/downloads/Introducing%20DLP%203D%20HDTV%20Whitepaper.pdf.

ID SOFTWARE 2009. id Technology Licensing. Available from: http://www.idsoftware.com/business/technology/.

IRAWATI, S. SANGCHUL, A., JINWOOK, K. AND HEEDONG, K. 2008. VARU Framework: Enabling Rapid Prototyping of VR, AR and Ubiquitous Applications. In *Virtual Reality Conference, 2008. VR '08. IEEE*, 201-208.

KATO, H., BILLINGHURST, M. 1999. Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System. In *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, San Francisco, CA, 1999, 85-94.

KATZOURIN, M., IGNATOFF, D., QUIRK, L,. LAVIOLA, J., JENKINS, O. 2006. Swordplay: Innovating Game Development through VR. In *Proceedings of the IEEE computer graphics and applications*, Nov./Dec. 2006, 15-19.

KELSO, J. SATTERFIELD, S.G., ARSENAULT, L.E., KETCHAN, P.M., AND FRIZ, R.D. 2003. DIVERSE: A Framework for Building Extensible and Reconfigurable Device-Independent Virtual Environments and Distributed Asynchronous Simulations. *Presence: Teleoperators & Virtual Environments 12*, 19-36.

KESSLER, G.D., BOWMAN, D.A., AND HODGES, L.F. 2000. The Simple Virtual Environment Library: An Extensible Framework for Building VE Applications. *Presence: Teleoperators & Virtual Environments 9*, 187-208.

MICROSOFT 2008. Microsoft Research Faculty Summit 2008: DemoFest. Available from: http://research.microsoft.com/en-us/um/redmond/events/fs2008/demofest.aspx.

MICROSOFT 2009. XNA Developer Center. Available from: http://msdn.microsoft.com/en-us/xna/default.aspx.

NATURAL POINT 2008. TrackIR: 6DOF Head Tracking. Available from: http://www.naturalpoint.com/trackir/.

ODA, O., LISTER, L., WHITE, S., FEINER, S. 2007. Developing an augmented reality racing game. In *Proceedings of the Proceedings of the 2nd international conference on INtelligent TEchnologies for interactive enterTAINment*, Cancun, Mexico, 2007 ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

PEEK, B. 2009. Managed Libary for Nintendo's Wiimote. Available from: http://www.codeplex.com/WiimoteLib.

RUBINE, D. 1991. Specifying gestures by example. In *International Conference on Computer Graphics and Interactive Techniques*, 329-337.

SCHMALSTIEG, D., FUHRMANN, A., HESINA, G., SZALAVARI, Z., ENCARNASALO, L.M., GERVAUTZ, M. AND PURGATHOFER, W. 2002. The Studierstube Augmented Reality Project. *Presence: Teleoperators & Virtual Environments 11*, 33-54.

STANLEY, K.O. AND MIIKKULAINEN, R. 2002. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation 10*, 99-127.

TEXAS INSTRUMENTS 2009. 3D TV - DLP HDTV. Available from: http://www.dlp.com/hdtv/3-d_dlp_hdtv.aspx.

TROILLARD, C. 2008. Wiimote Image. Available from: http://www.osculator.net/wiki/uploads/Main/pry-wiimote.gif.

VARCHOLIK, P., BARBER, D., NICHOLSON, D. 2008. Interactions and Training with Unmanned Systems and the Nintendo Wiimote. In *Proceedings of the 2008 Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC '08)*, 2008.

VARCHOLIK, P., LAVIOLA, J., NICHOLSON, D. 2009. TACTUS: A Hardware and Software Testbed for Research in Multi-Touch Interaction. In *Proceedings of the Human-Computer Interaction International (HCI International)*, San Diego, CA, 2009.

VARCHOLIK, P., MERLO, J. 2008. Gestural Communication with Accelerometer-based Input Devices and Tactile Displays. In *Proceedings of the 26th Army Science Conference*, Orlando, FL, 2008.