jReality — Interactive Audiovisual Applications Across Virtual Environments

Peter Brinkmann* The City College of New York
Charles Gunn† Technische Universität Berlin
Steffen Weißmann‡ Technische Universität Berlin

ABSTRACT

jReality is a Java scene graph library for creating real-time interactive applications with 3D computer graphics and spatialized audio. Applications written for jReality will run unchanged on software and hardware platforms ranging from desktop machines with a single screen and stereo speakers to immersive virtual environments with motion tracking, multiple screens with 3D stereo projection, and multi-channel audio setups. In addition to euclidean geometry, jReality supports hyperbolic and elliptic geometry.

jReality comes with a number of graphics rendering backends, ranging from pure software to hardware-accelerated to photorealistic. A distributed backend is available for cluster-based virtual environments. Audio backends range from a basic stereo renderer to a high-performance Ambisonics renderer for arbitrary 3D speaker configurations.

jReality achieves device-independent user interaction through a layer of abstract input devices that are matched at runtime with available physical devices, so that a jReality application will work with keyboard and mouse in a desktop environment as well as with motion tracking in a virtual environment.

Keywords: Virtual Reality, Desktop Virtual Reality, Immersive Virtual Environments, Java

Index Terms: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Audio input/output; H.5.5 [Information Interfaces and Presentation]: Sound and Music Computing—Signal analysis, synthesis, and processing;

1 INTRODUCTION

Immersive virtual environments are becoming increasingly common in areas as diverse as mathematics, sciences, engineering, architecture, and arts. This raises the question of how to develop and deploy software and content for use in virtual environments. One common approach is to create 3D content on a desktop computer and then to export it as a file that can be viewed in a virtual environment, leaving the interactive capabilities of the virtual environment (e.g., motion tracking) largely unused. Another common approach is to write software specifically for a virtual environment, using specialized libraries. On a desktop system, software written in this fashion may run poorly (e.g., requiring input simulators to work) or not at all.

We present a third approach, taken by *jReality,* a visualization and sonification library for virtual reality applications. Based on the notion of a *scene graph*, it facilitates the creation of interactive 3D audiovisual scenes that are portable across a wide range of hardware and software platforms. A jReality application will run unchanged on platforms ranging from desktop machines with a single screen and stereo speakers to virtual reality environments with motion tracking, multiple screens with 3D stereo projection, and multi-channel audio. jReality is written in Java and will run on any common operating system.

In addition to its portability to a wide range of hardware and software platforms, jReality is *metric-neutral:* It supports hyperbolic and elliptic geometry as well as euclidean geometry. In particular, jReality handles real-time shading and sound propagation in curved spaces, and it provides the first genuine non-euclidean tracked motion.

jReality is open source software, covered by a BSD license. The code and documentation are available at `www.jreality.de`.

jReality is the primary software platform of the PORTAL at the Technical University of Berlin and the VisorLab at the City College of New York. It was among the winners of the Open Source Software Competition of ACM Multimedia [9].

2 RELATED WORK

jReality differs from other scene graph libraries in terms of scope and flexibility. We briefly review related work in the context of portability, abstraction of input devices, metric-neutrality, and audio functionality.

Related packages include Geomview (`www.geomview.org`), an interactive 3D viewer for Unix; OpenSceneGraph (`www.openscenegraph.org`) and OpenSG (`www.opensg.org`), two flexible, OpenGL-based scene graph packages; Syzygy [7, 6] and VRJuggler (`http://vrjuggler.org/`), two development frameworks for VR applications; and Java3D™[8], a complex scene graph with fast graphics via OpenGL.

Except for GeomView, none of them is metric-neutral. Except for Java3D, they are written in C or C++, often with support for scripting languages such as Python, and are not easily deployed through the web. Syzygy and VRJuggler do not offer a scene graph API (Syzygy used to include a scene graph API, but it has been removed from recent releases). OpenSceneGraph and OpenSG have no built-in support for tracking devices, while Syzygy and VRJuggler expect tracking devices and require simulators for desktop use. Except for VRJuggler, they rely on additional libraries for audio; none of them supports spatialized audio beyond stereo or surround sound.

3 OVERALL DESIGN AND CAPABILITIES OF JREALITY

Key design features of jReality include

- are a clear separation between the scene graph frontend and rendering backends,
- a clear separation between the meaning of user interactions and the means of user interaction,
- thread safety, and
- metric neutrality.

We discuss each of these features. A separate project that integrates the jBullet physics engine (`jbullet.advel.cz`) into jReality is beyond the scope of this overview.

*e-mail: brinkman@sci.ccny.cuny.edu
†e-mail: gunn@math.tu-berlin.de
‡e-mail: weissman@math.tu-berlin.de

IEEE Symposium on 3D User Interfaces 2010
20-21 March, Waltham, Massachusetts, USA
978-1-4244-6847-8/10/$26.00 ©2010 IEEE

3.1 The scene graph

A jReality *scene graph* is a rooted, directed graph without cycles; it differs from a tree in that individual nodes as well as entire subgraphs may be attached in several places at the same time. The scene graph API was designed to provide all functionality with a small number of node types. There is only one type of node that defines the hierarchical structure of the graph; all other nodes (transformations, geometries, lights, cameras, appearances and audio sources) are leaves.

Required attributes such as colors or textures are stored in *appearances* as key-value pairs. Attributes will be inherited by subsequent scene graph nodes. There is no fixed list of attribute keys or types as different rendering backends may honor different sets of attributes.

Geometries allow storage of arbitrary user-defined attributes in addition to the standard attributes (like coordinates and connectivity information). In this way also exotic data (maybe a temperature or a tensor field) may be encoded and visualized.

AudioSources generate audio samples upon request, at the sample rate of their choosing. Audio sources are only expected to render mono samples; any multi-channel processing is the responsibility of the backend. The current list of audio sources includes sources that draw their samples from audio files and streams, from audio interfaces, and from software synthesizers such as Csound (`www.csounds.com`).

3.2 Graphics backends

Graphics backends convert a jReality scene graph into graphics. There are various backends for interactive and non-interactive rendering.

Interactive backends include a pure Java software renderer and a hardware accelerated OpenGL renderer. The former offers superior quality particularly for transparent scenes and can also be deployed remotely where native extensions (required by the OpenGL backend) are not allowed. In cluster-based virtual environments, a distributed backend will display scenes on multiple screens.

Non-interactive backends include a backend that exports scenes to Pixar's RenderMan for high-quality batch processing as well as a backend that uses Sunflow (`sunflow.sourceforge.net`), a sophisticated Java rendering system.

3.3 Audio backends

Audio backends render the spatialized audio of the scene. The audio rendering pipeline of jReality includes auxiliary sends and returns for inserting effects and distance cues like reverberation and distant-dependent attenuation, following [3]. It also implicitly models sound propagation, yielding physically accurate Doppler shifts [1, 2].

The Java Sound backends will render a stereo signal when running on a desktop system, or 5.1 surround in home theater setups, providing basic audio capabilities without requiring any setup. For high-quality, low-latency audio on a 3D multi-channel speaker rig, the JACK (`jackaudio.org`) backends of jReality will render spatialized audio using Ambisonics [4].

3.4 Portable Interaction

The tool system of jReality separates the meaning of user interaction from the hardware of the interaction. For instance, a scene may allow the user to rotate an object. The designer of the scene attaches a rotation tool to the object without considering how the user will effect a rotation. At runtime, the tool system determines what input devices are available and matches them with the tools in the scene. On a desktop computer, a rotation will typically be caused by a mouse dragging event. In a virtual environment, motion tracking or wand gestures may trigger a rotation. The application remains the same in both cases.

Currently drivers are available for keyboard, mouse, joystick, Trackd (supporting all popular motion tracking and immersive input devices such as Ascension MotionStar or A.R.T. DTrack systems), Wii Remote, Space Navigator, and GameTrak.

3.5 Non-euclidean Visualization

In addition to the familiar euclidean geometry, jReality supports elliptic and hyperbolic geometry. All three of these geometries can be modeled within projective geometry. Since modern rendering systems implement projective transformations, this naturally leads to the possibility of *metric-neutral* visualization systems [5]. By building its foundational classes on projective geometry, jReality offers seamless support of all three geometries.

The audio components of jReality are aware of non-euclidean geometries and will model sound propagation accordingly. A number of other features (e.g., distance-dependent attenuation) also depend on the current metric, and jReality comes with distance cues and effects that can be plugged into the audio rendering pipeline to achieve the correct behavior.

jReality offers non-euclidean tracked motion in a virtual environment providing for the first time the full experience of moving within a non-euclidean world.

4 Conclusion

While jReality was originally conceived as a tool for mathematical visualization, it has since grown to become a general platform for real-time interactive audio and video. Being portable across a wide range of hardware and software configurations, it erases the distinction between development for desktop machines and development for virtual environments. Its support for non-euclidean graphics and sound makes advanced mathematical concepts available to a wide audience, opening up new creative possibilities.

Acknowledgements

We wish to express our gratitude to all members, past and current, of the jReality community, especially Tim Hoffmann and Holger Pietsch. Stefan Sechelmann designed and implemented the plugin system.

References

[1] P. Brinkmann and M. Gogins. Doppler Effects Without Equations. In *Symposium on Virtual Reality Software and Technology*. ACM, 2009. Poster.

[2] P. Brinkmann and S. Weißmann. Real-time Interactive 3D Audio and Video with jReality. In *International Computer Music Conference*, 2009.

[3] R. W. Furse. Spatialisation - Stereo and Ambisonic. In *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing,and Programming*, 2000.

[4] M. Gerzon. Surround-sound psychoacoustics. *Wireless World*, 80:486, 1974.

[5] M. Phillips and C. Gunn. Visualizing Hyperbolic Space: Unusual Uses of 4x4 Matrices. In *1992 Symposium on Interactive 3D Graphics*, pages 209–214. ACM SIGGRAPH, ACM, 1992.

[6] B. Schaeffer, P. Brinkmann, G. Francis, C. Goudeseune, J. Crowell, and H. Kaczmarski. Myriad: Scalable VR Via Peer-to-peer Connectivity, PC Clustering, and Transient Inconsistency. In *Virtual Reality Software and Technology*. ACM, 2005.

[7] B. Schaeffer and C. Goudeseune. Syzygy: Native PC Cluster VR. In *Virtual Reality Conference*. IEEE, 2003.

[8] D. Selman. *Java 3D Programming*. Manning Publications, 2002. https://java3d.dev.java.net.

[9] S. Weißmann, C. Gunn, P. Brinkmann, T. Hoffmann, and U. Pinkall. jReality: A Java Library for Real-time Interactive 3D Graphics and Audio. In *ACM Multimedia*, pages 927–928, 2009.