Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

COMMON SCENE DEFINITION FRAMEWORK FOR CONSTRUCTING VIRTUAL WORLDS

Lee A. Belfore II
Prabhu. V. Krishnan
Emre Baydogan

Department of Electrical and Computer Engineering
Old Dominion University
Norfolk, VA 23529, U.S.A.

ABSTRACT

Developing VR applications is a challenging and rewarding endeavor, complicated by the variety and complexity of the available VR platforms. Furthermore, efficiencies realized in a specific platform may be lost if the application is migrated to a different platform. In this paper, we introduce and investigate the Common Scene Definition Framework (CSDF), a modeling representation consisting of a superset of capabilities taken from a collection of existing VR platforms. The purpose of CSDF is to serve a quick prototype framework for synthesizing an interactive virtual environment for a particular platform while attempting to optimize the translation to leverage strengths of the target platform. In an implementation independent fashion, the CSDF is envisioned to extensibly represent all geometry, appearance, interaction, and behavior for a VR application. Finally, an example is provided demonstration these basic ideas among the VRML 1.0, VRML97 and Java3D platforms.

1 INTRODUCTION

Virtual reality (VR) applications are used in a wide range of disciplines to provide new and different perspectives and also improve upon existing processes of the respective domains (Vince 1995). Advances in computer systems and processing power (Belfore 2001), improved network bandwidth, advances in the gaming industry have enabled easier and more widespread deployment of virtual worlds. Development of VR applications has always been challenging often requiring application developers to have expertise in the different VR technologies. Indeed, new VR technologies impose significant learning curves for even the most experienced VR developer. VR applications bring together a diverse range of hardware platforms, software tools and algorithms. Making these disparate components work together in one system is a required aspect of VR and making VR applications challenging to develop (Bierbaum 2001). Furthermore, one may need to choose a target VR platform based on limited knowledge of that platform. Automating some aspects of the creation of prototype applications can enhance the ability to develop high quality VR applications.

In this paper, we introduce the Common Scene Definition Framework (CSDF) that represents in an implementation independent fashion the capabilities (solid modeling, appearance, interaction, and behaviors) of a number of VR platforms and provides the ability to synthesize to a desired target platforms. The goal of the CSDF is to make possible the evaluation of function and performance of an application developed on a familiar platform on an unfamiliar platform while minimizing the time necessary to perform that evaluation and optimizing to limit the performance impact.

Several other approaches can provide a similar capability. First, powerful commercial applications such as 3D Studio Max, support a wide range of capabilities and are intended as both an authoring and rendering platforms having all of the expected supporting functionality. In the event design requirements force the selection of a different rendering platform, models developed from 3D Studio Max may need to be converted or reauthored to support the new platform. Second, VR Juggler (Bierbaum 2001) defines a virtual platform upon which the VR application is built. The virtual platform hides the complexities of low level details allowing the developer to construct an application from a collection of objects. In addition, the virtual platform enables the description of the application in a fashion that is independent of the implementation platform, enabling a VR Juggler application to run on a variety of platforms. Third, DIVERSE (Device Independent Virtual Environments-Reconfigurable, Scalable, Extensible) is an API that provides the ability to operate on a span of platforms from laptops through high end CAVE platforms without modification (Arsenault 2001). Furthermore, DIVERSE includes the ability to abstract the integration of several devices and provides the ability for an application to is an open source framework developed to facilitate the creation of distributed interactive simulations.

The proposed CSDF offers several benefits over these approaches. First, the CSDF offers application language and format independence. Second, a path exists whereby an existing world on a different platform can be imported into CSDF preserving the institutional investment incurred in its development. Third, in the synthesis step, the developer can receive early feedback on whether the capabilities of the target platform matches the source platform. Fourth, and finally, the synthesis step offers an optimization step whereby the module interfaces can be optimized.

This paper is organized into five sections including an introduction, an introduction to the CSDF, the prototyping in the CSDF, an example implementation, and a section with the summary and future work.

2 THE COMMON SCENE DEFINITION FRAMEWORK

In addition to the capabilities previously described, two key CSDF capabilities are described in more detail. CSDF is ultimately envisioned to support the following:

1. Platform dependent optimizations across module interfaces,
2. Synthesis integration of modules from heterogeneous platforms.

Supporting Item 1 results in a synthesized application that achieves higher performances than what might be expected from a direct functional transformation. More importantly, target platform specific optimizations are transparently handled by the synthesis. Supporting Item 2 provides the ability to pull content from disparate platforms by representing the content in CSDF as an intermediate form. Furthermore, by employing synthesis in the integration process, some inefficiencies lost by purely integration methods may be mitigated.

The CSDF further enables the support of short design cycles by making development possible on a familiar platform and then synthesizing a compatible version on a desired target platform. Since technology and standards are constantly evolving, developers are constantly challenged to keep up.

Technologies have different and potentially incompatible functionality. The virtual world developer may wish to evaluate among these different technologies to choose a virtual world technology to best satisfy requirements. These requirements serve as the thrust for applying CSDF for creating prototypes virtual worlds. Such an approach can enable a virtual world developer to develop the virtual world application in the VR technology of his expertise and have the framework to automate the migration of the application to the new virtual world platform/technology and thus significantly reduce the time to develop a working prototype.

Employing the CSDF as described enhances the development process in several ways. First, such a methodology would alleviate situations where a developer may need to use a VR technology that he or she is not familiar for various reasons such as project constraints specific application specific needs, and etc. As employed in several domains such as the manufacturing sector, the developer may need a prototype to serve as a proof of concept demonstration using an unfamiliar technology. By using synthesis tools to automate the process of creating an application on the unfamiliar platform, a prototype may be quickly developed on that VR platform. The synthesized prototype is likely less efficient than an hand crafted version, but may assist the developer in making high level design decisions that may reduce the overall design time for the final product. This insight gained from the prototyping phase can enable the creation of a more robust and efficient final virtual world and also enable verification of requirements and specification of the virtual world implementation.

3 THE CSDF PROTOTYPING FRAMEWORK

The ultimate goal of this research is to develop a framework that provides the capability to quickly generate an efficient virtual world prototype either from an existing virtual world or from a collection of components imported from several platforms.. To build such a virtual world prototyping system, a component-based approach is proposed. The following section provides a conceptual discussion of this proposed virtual world prototyping system. A representation of the proposed solution is shown in Figure 1. The prototyping framework requires the development of several capabilities that are described in the following subsections.

3.1 Prototyping Methodology

The prototyping methodology applied to the design of virtual worlds is elaborated. Figure 2 shows the top level view of any virtual world.

Creating virtual worlds requires composition and implementing interactions. Scene composition functions might include integrating geometric primitives, sensors, textures, and etc. specific to the world being modeled. Interaction might involve specifying behavior for entities composed in the virtual world scene. The scene composition and interactions will dictate the capabilities of the virtual world.

Virtual world creation and deployment on an unfamiliar platform may require an extended learning period for the developer. For a specific deployment, the developer may have to choose among several virtual world technologies and find the platform that best meets the application requirements. To reduce the development cycle time, the

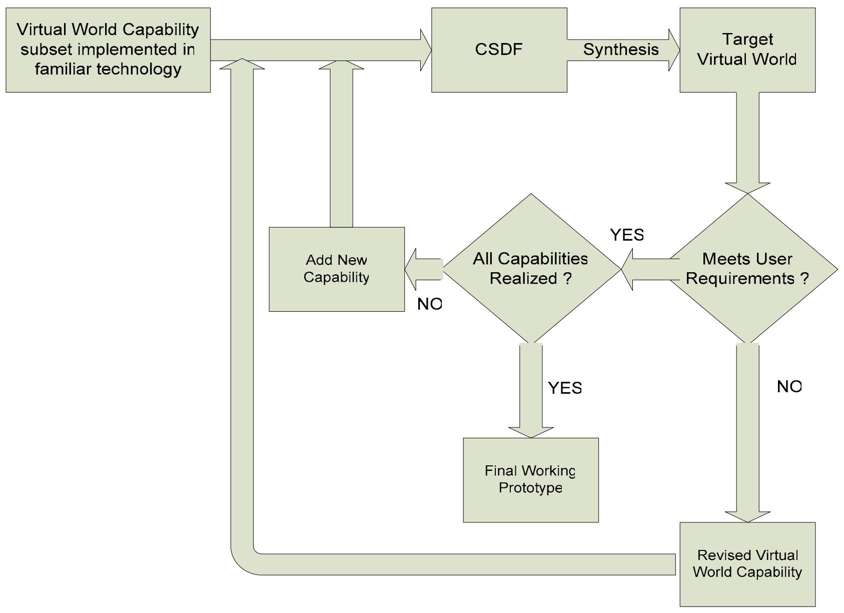

Virtual World Capability subset implemented in familiar technology
CSDF
Synthesis
Target Virtual World
Meets User Requirements ?
YES
All Capabilities Realized ?
NO
Add New Capability
YES
Final Working Prototype
NO
Revised Virtual World Capability

Figure 1: Virtual World Creation Methodology

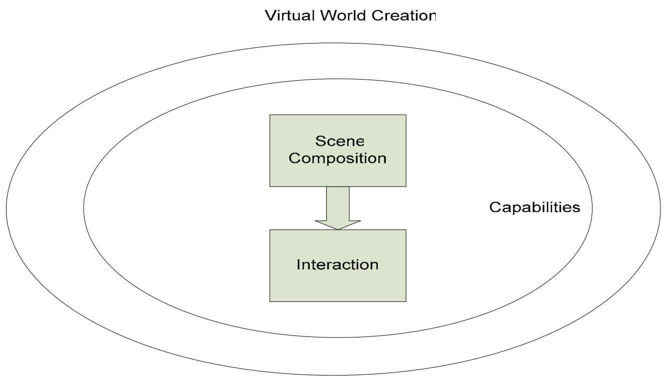


Figure 2: Virtual World Top Level View

developer can implement a subset of the virtual world capability in a familiar technology and follow an iterative, incremental prototyping methodology to implement the virtual world. This approach cannot be considered as migration to a target platform as it is incremental in nature. After all the required capabilities are realized in the synthesized prototype, it may result in the evolving prototype being accepted as the final end product. Even so, the approach to developing the virtual world system is incremental, iterative and requires an evolving prototype at each stage. This development process is shown in Figure 1.

3.2 CSDF Synthesis and Class Structure

The overall flow of the prototyping process as shown in Figure 3 is summarized as follows. The virtual world developed in a VR technology by the virtual world developer is imported into the framework as described in the next section. The framework synthesizes a working prototype based on the user requirement for the specific output synthesis platform. One of the important design objectives in the framework is maintain modularity by developing a consistent set of interfaces and functional modules to implement these interfaces. These interfaces also provide an easy mechanism to extend the framework.

To synthesize a virtual world meeting the design and implementation requirements of the end user, the model must be isolated from its implementation specification. Since the model of the virtual world is independent of the target synthesis platform, the framework may collectively have capabilities represented that do not exist in any of the individual output synthesis formats. As an example of the prototyping process, an end user may design a complex 3D model using a commercial product such as 3D Studio Max and export the model as a VRML file. This model may be part of a virtual world scene that may be synthesized in different output formats such as OPENGL, X3D, Java3D, and etc. In order to achieve this goal, the model needs to be imported into the CSDF.

In order to synthesize the scene in the target platform, the synthesis module is required to handle the capabilities

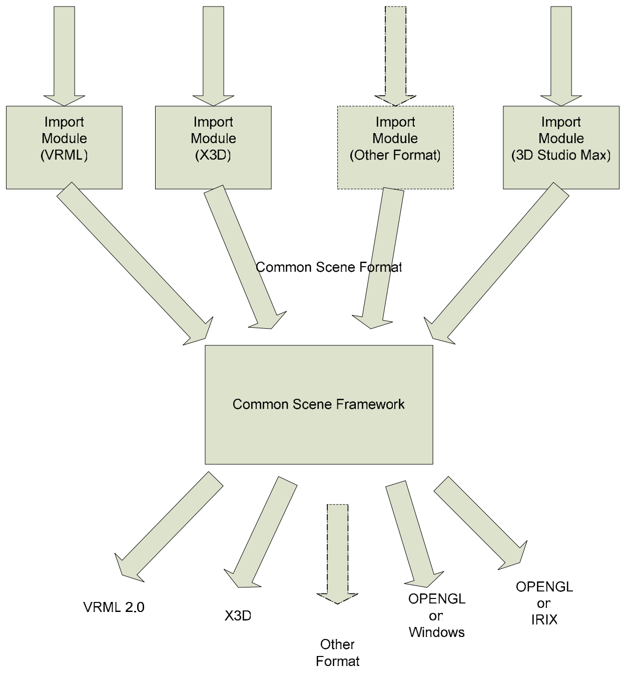


Figure 3: Conceptual Representation of the Target Virtual World Synthesis

and features of the target platform. The synthesis process is a mapping from the requirement specification represented by CSDF to the target hardware/software virtual world architecture. Some of these mappings may be infeasible for various reasons. Since the prototyping system has complete knowledge of the capabilities of the synthesis architecture, the prototyping system can give immediate feedback on whether the requirement specification can be satisfied for the selected output platform. Consider, for example, the requirement for keyboard input from the user. If VRML is the synthesis target, the prototyping system can report that standard VRML does not support keyboard data entry.

In contrast to the approach of programming the provided API in both the VR Juggler (Bierbaum 2001) and Diverse (Arsenault 2001) systems, the proposed prototyping system can leverage the expertise in one patform to another platform through CSDF and synthesis. Thus, the developer is able to reuse virtual world models in a platform independent fashion, saving the developer from the steep learning curve in the new environment/platform. The authoring process may occur on the platform from which models are imported or may continue on the target platform as expertise is gained. An important feature that can be implemented within CSDF is the ability to alert the developer of limitations in the target synthesis platform. This can help the developer to make correct choices in selecting target synthesis platforms to meet the requirement specification for the VR application. This approach fits with the current approach followed by most virtual world developers to design the geometry models and animations in commercial off the shelf (COTS) design tools such as 3D Studio Max and Maya and later import these models into their virtual worlds. A possible future component of the prototyping system may be a graphical authoring tool that helps a virtual world developer to link multiple models within the common scene framework. This component can potentially reduce several iterations from the development process.

Some VR platforms may provide the developer with primitives with complex capabilities. This may correspond to an assembly of simpler capabilities in the target platform. The gist is that the prototyping system must bridge the differences in inherent capabilities of the synthesis platform and that needed to represent the virtual world. The prototyping system has complete knowledge of the capabilities and limitations of the synthesis platform, the synthesis modules may have opportunities to approximate some capabilities in the synthesized world. For example, the system may approximate movie textures by a simple image to keep the virtual world lighter on a less powerful device such as a PDA. In addition, a higher resolution image requiring a higher network download time may be approximated by a lower resolution image. Furthermore, in the synthesis step, optimization opportunities may be available, i.e. to remove software layers and optimize interfaces, that an integration tool cannot provide.

The X3D specification has been the basis for the selection of the nodes and skeletal structure that constitutes the CSDF. Significantly, X3D is under active development and review by the Web3D consortium. Thus, basing the framework on X3D makes it easier to remain consistent with the latest developments proposed by the Web3D consortium (2003,2005). Furthermore, X3D includes a rich set of primitives for modeling 3D geometry, behaviors and interactions and thus, X3D is an ideal starting point with respect to synthesis to all output VR formats. Supporting an evolving CSDF, X3D is an extensible open standard text based XML-encoded scene graph. Finally, in web services, the programmatic interfaces for application to application communication on the internet use XML technologies to construct messages that can be exchanged using a variety of underlying protocols. This helps to make integration with web services easier as specified by the World Wide Web Consortium (2002).

Figure 4 is the class diagram showing the base class CSDFNode and several derived CSDF classes. CSDFNode is the node class that defines the tree structure for the framework. All classes extend this base class CSDFNode.

Serialization in Java is the process by which the JVM converts an object instance and all the references the object contains, into a linear stream of bytes, which can then be sent through a communication socket, stored to a file or database (Greanier 2000). Java also has the capability to read these bytes and restore the Java object represented by the bytes. The base class CSDFNode implements the Serializable interface. All the CSDF classes that extend the CSDFNode class thus automatically become serializable. This feature is used by the framework to save an input virtual world parsed into the framework for later restoration and use. The CSDFNode class has a collection class member and methods to manipulate the tree structure.

The framework has a number of interfaces to support the nodes defined in the X3D specification. Figure 5 shows the class diagram of some example CSDF classes implementing the CSDFGeometry interface. During the parse phase of an input VRML or X3D scene, the CSDFGeometry interface is used to verify that only a valid geometry class is assigned as the geometry field of a Shape object. Similarly another interface defined in CSDF is the CSDFTexture interface to handle texture fields. The X3D specification allows geometries to be textured with different types of textures such as an ImageTexture, a MovieTexture or a PixelTexture. The CSDF nodes for the ImageTexture, MovieTexture and PixelTexture nodes implement the CSDFTexture interface. Figure 6 shows the class diagram of a few synthesis modules implementing the CSDFSynthesis interface. All synthesis modules in rapid prototyping system must implement the CSDFSynthesis interface.

4 IMPLEMENTATION

The example CSDF framework was implemented in Java using JavaCC, a Java compiler-compiler, and JDOM API. The translation phase is the phase in which the parser translates the input file into an abstract syntax independent of the syntax of the output platform. Within the context of the rapid prototyping system, this abstract syntax refers to the CSDF. The translation can be done after type checking or it can be done at the same time. For example, to import a scene in an alternate scene format that uses Cartesian coordinates, the parser for that format must translate the coordinate data in order to store in the common scene format

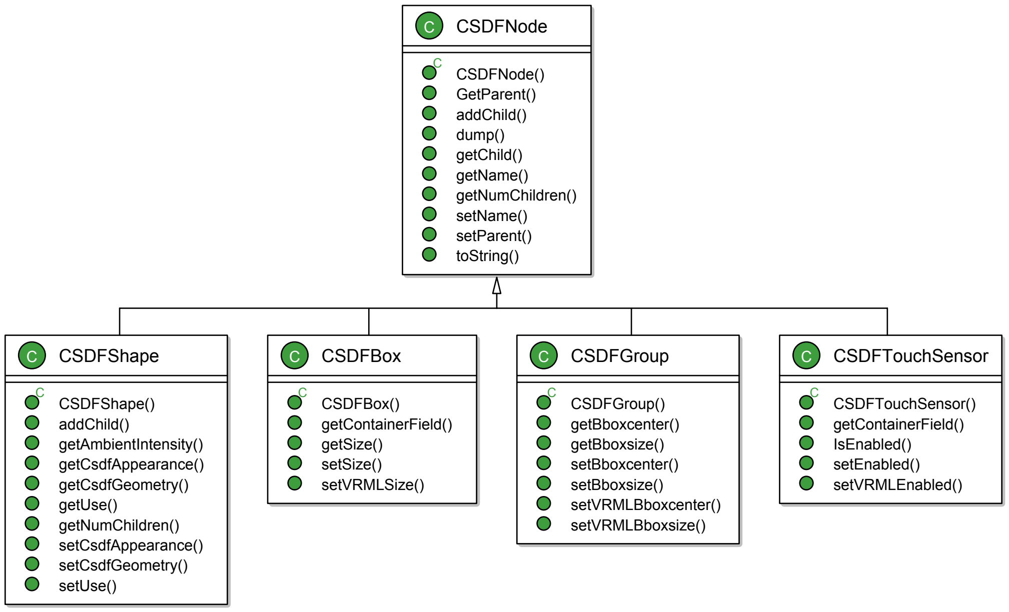


Figure 4: Class Diagram Representing Base Class CSDFNode and Some Derived Classes

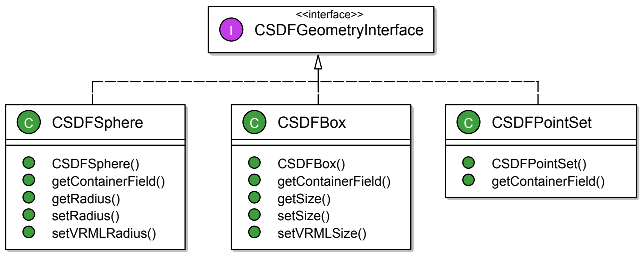


Figure 5: CSDFGeometry Interface Class Diagram with Example Implementations

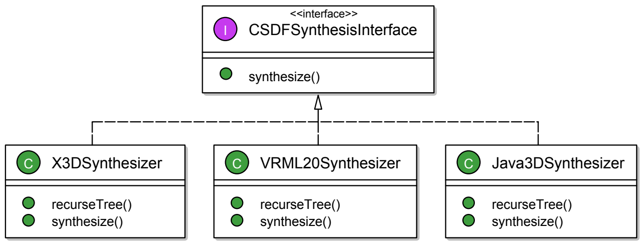


Figure 6: CSDF Synthesis Interface Class Diagram with Example Implementations

of the rapid prototyping system. In the VRML parser, part of the translation is performed as part of the type checking and the rest is done within the framework classes. Another set of translations of data occurs during the synthesis phase. The output of the translation phase is the CSDF representation of the scene represented in the input environment.

The synthesis phase of the prototyping methodology maps a conceptual model of the virtual world (CSDF) to a visual output virtual world as shown in Figure 7.

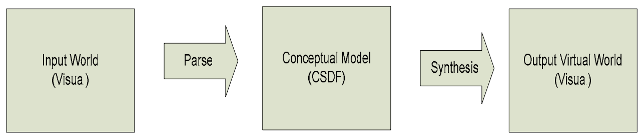
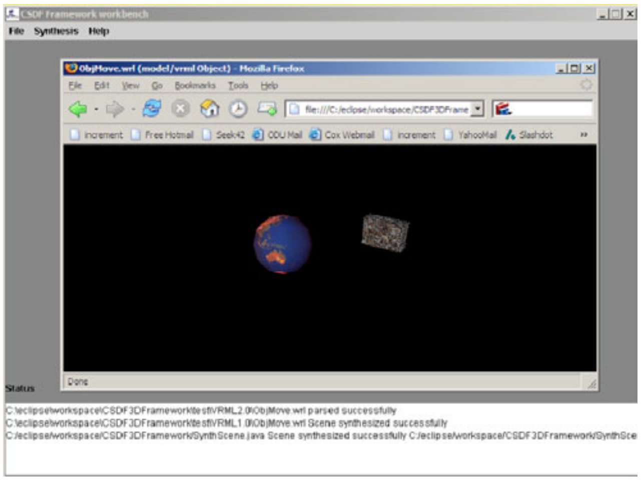


Figure 7: Simplified Synthesis Pipeline

Depending on the target VR platform, the synthesis module may filter information, aggregate lower level capabilities into higher level abstractions, or deaggregate higher level abstractions into lower level capabilities. These transformations are driven by the requirement specification of the application. For the synthesis of the virtual world into an output format, the scene graph hierarchy makes it logically simple to synthesize individual components within the model. The synthesis into an output format that does not have a scene graph organization of components is challenging. The virtual world as represented by the common scene format has all the information necessary to synthesize the world. Hence, starting from the root node, it is theoretically possible to extract this information stored at different levels of the hierarchy by traversing the scene graph up and down the scene graph and transform this scene to the required output format.

The synthesis modules are independent of the framework. The synthesis module for each VR platform takes the CSDFFramework object as input and synthesizes the output virtual world. The hierarchical organization makes the synthesis process simple in the following way. The synthesis module invokes the synthesis function on the root node of the scene. The root node in turn recursively invokes the synthesis function for each of the child nodes. When the control returns from the synthesis function of the root node, synthesis is complete. All synthesis modules must implement the CSDFSynthesisInterface. This interface shows the method specification for the synthesis function. Currently the prototyping system is able to synthesize a subset of the features of VRML1.0, VRML97, X3D, and Java3D.

Figure 8 shows the example virtual world that demonstrates the several important capabilities including the ability to create aggregate objects from geometry primitives, a demonstration of drag behavior from one platform synthesized in another, appropriately represent and then synthesize solid model appearances, and maintaining similar navigation functions.



CSDF Framework workbench
File Synthesis Help
ObjMove.wrl (model/vrml Object) - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///C:/eclipse/workspace/CSDF3DFrame
Increment Free Hotmail Seek42 ODU Mail Cox Webmail Increment YahooMail Slashdot
Done
Status

```
C:\eclipse\workspace\CSDF3DFramework\test\VRML2.0\ObjMove.wrl parsed successfully
C:\eclipse\workspace\CSDF3DFramework\test\VRML1.0\ObjMove.wrl Scene synthesized successfully
C:/eclipse/workspace/CSDF3DFramework/SynthScene.java Scene synthesized successfully C:/eclipse/workspace/CSDF3DFramework/SynthSce
```

Figure 8: Input VRML97 Scene Running in Mozilla Firefox Browser using Contact VRML Plug-In.

The scene consists of a sphere and a box textured with two different images. In the scene, a plane sensor node is associated with the sphere. The effect of the plane sensor is routed to the box object. When the user drags the sphere, the box moves in magnitude and direction of the drag on the sphere. For example, such functionality may be part of a slider control in a user interface. In VRML97, the drag behavior is implemented using the PlaneSensor node. By adding a ROUTE statement to route events to the target node, the actions on the PlaneSensor node are translated to the target node. The above VRML97 scene is parsed by the VRML Parser and the corresponding CSDF tree was generated. Next, the scene is synthesized into VRML 1.0, VRML97, X3D, and Java3D formats. The input parser for VRML97 populated the CSDF framework classes with the attributes and behaviors of the corresponding VRML97 input scene and synthesized to X3D and Java3D format without any losses of information. Figure 9 and Figure 10 shows the screen shots of the synthesis to X3D and Java3D respectively. The synthesis of the X3D is straightforward following from the CSDF framework.

In a VRML97 and other VR environments, some features, such as lighting and navigation, are automatically provided by the environment. In a Java3D world, the developer must provide all these essential features to the scene. This expertise in the specific virtual world platform is incorporated into the synthesized world by the synthesis module of the framework. After comparison with VRML and Java3D environments, zoom and rotate characteristics have been enabled in all Java3D scenes. The zoom capability allows a user to view the virtual world from different distances. The rotate capability allows a user to rotate the viewpoint of the user. Also a light source is incorporated by default in the synthesized scene for illumination of objects within the scene.

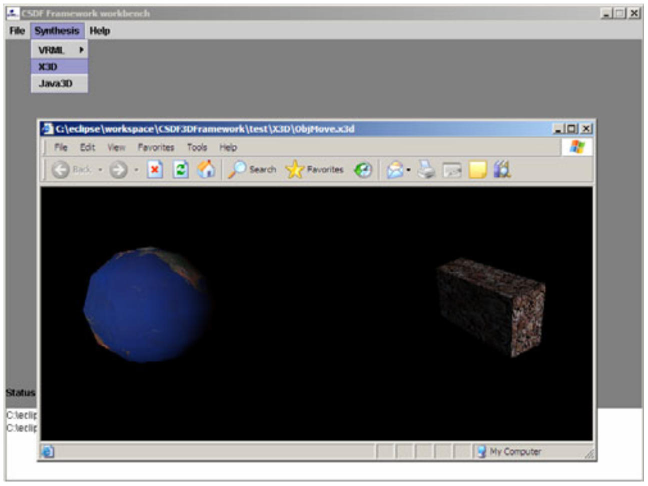


Figure 9: Synthesis to X3D Platform

In addition, the VRML drag capability has no direct counterpart in Java3D, requiring the implementation of a compatible drag functionality. This drag behavior was then inserted into the Java source and connected into the Java3D scene graph. The framework can be extended by implementing other common sensor capabilities as defined by X3D specification. This sample example thus highlights the usefulness of the proposed prototyping system.

VRML 1.0 has capabilities to represent geometric shapes and apply textures to surfaces, but does not have any capabilities for dynamic interactions. The input scene has features that cannot be represented on the target platform. Hence, this functionality cannot be represented in

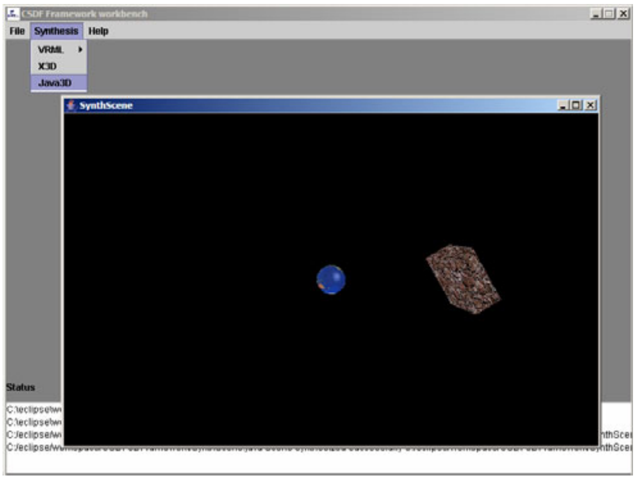


Figure 10: Synthesis to Java3D Platform

the output format. In this context, CSDF synthesizes the content that is implementable and reports on those capabilities that cannot be implemented.

5 SUMMARY AND FUTURE WORK

The research summarized in this paper provided two outcomes. The Common Scene Definition Framework (CSDF) has been defined to serve as an implementation independent repository for VR platform functionality. The CSDF provides the ability to represent applications in a platform independent fashion. The CSDF provides a pathway for the synthesis of virtual worlds to different platforms. Synthesis implies that the developer need not know the target platform. Furthermore, synthesis implies that optimizations are performed to produce efficient prototype applications. An example application was created to demonstrate the CSDF concept using subsets of the VRML 1.0, VRML97, and X3D platforms. The example also demonstrated that the framework could provide early feedback on the limitations in the capabilities of a particular synthesis platform. This early feedback could help the virtual world developer to evaluate different choices of VR technologies to meet the user requirement.

A limited set of VR platforms and capabilities were selected to demonstrate the core concept. The focus initially was to start with geometric primitives and composite shapes and demonstrate the capability of the framework to transform from selected VR platforms to others. In addition, a few selected behaviors that are common to many virtual worlds were incorporated into the framework. Additional behavioral capabilities such as proximity sensors, timers and, etc. need to be implemented into the framework to enable the synthesis of more complex virtual worlds. The framework may also be extended to support import and synthesis to other VR platforms such as OpenGL, Macromedia Flash, and etc. The framework can be extended to facilitate the synthesis of distributed virtual world by incorporating, different interconnect mechanisms such as Sockets, HLA, DIS, etc. into the framework. An important thread of research must be conducted to assess the validity of the transformations and representations. For example, a process needs to be developed for the validation of sensitivities of the different sensors in different VR platforms. Also, default values of attributes need to be normalized across different VR platforms.

REFERENCES

Arsenault, L., J. Kelso, R. Kriz, F. D. Neves. 2001. DIVERSE: A software toolkit to integrate distributed simulations with heterogeneous virtual environments. White Paper.

Belfore II, L. A. 2001. An architecture for constructing large VRML worlds. *Transactions of the Society for Modeling and Simulation*, 18 (1): 24-40.

Bierbaum, A., C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. VR Juggler: A virtual platform for virtual reality application development. In *Proceedings of IEEE Virtual Reality 2001*. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 89-96.

Greanier, T. 2000. Flatten Your Objects: Discover the secrets of the Java serialization API. *JavaWorld*. <http://www.javaworld.com/javaworld/jw-07-2000/jw-0714-flatten.html> [Accessed on 15 July 2005].

Vince, J. 1995. *Virtual reality systems.* Reading, Massachusetts: Addison Wesley.

Web3D Consortium. 2005. Overview of X3D <http://www.web3d.org/x3d/overview.html> [Accessed on 15 July 2005].

Web3D Consortium. 2003. Extensible 3D (X3D) specification <http://www.web3d.org/x3d/specifications/ISO-IEC-19775-IS-X3DabstractSpecification> [Accessed on 15 July 2005].

World Wide Web Consortium (W3C). 2002. Web services activities page <http://www.w3.org/2002/ws/> [Accessed on 15 July 2005].

AUTHOR BIOGRAPHIES

LEE A. BELFORE, II is an associate professor in the Department of Electrical and Computer Engineering at Old Dominion University. He received his BS degree in electrical engineering from Virginia Tech, his MSE Degree in electrical engineering and computer science from Princeton University, and his Ph.D. degree from the University of Virginia in 1990 in electrical engineering. His research interests include virtual reality, medical modeling and simulation. Dr. Belfore is a Senior Member of the IEEE and members of ASEE, Sigma Xi, and AUVSI. His e-mail address is <lbelfore@odu.edu>

PRABHU V. KRISHNAN is a graduate student in the Department of Electrical and Computer Engineering at Old Dominion University. His research interests include virtual reality systems and enterprise business solutions. His e-mail address is <prabhu.Krishnan@gmail.com>.

EMRE BAYDOGAN is a doctoral student in the Department of Electrical and Computer Engineering at Old Dominion University. His research interests include virtual reality, 3D visualization, and simulation. He is a member of IEEE. His e-mail address is <ebayd001@odu.edu>.