Winter 2008

Rapid Prototyping for Virtual Environments

Emre Baydogan
Old Dominion University

RAPID PROTOTYPING FOR VIRTUAL ENVIRONMENTS

by

Emre Baydogan
B.S. June 1999, Marmara University, Turkey
M.S. June 2001, Marmara University, Turkey

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirement for the Degree of

DOCTOR OF PHILOSOPHY

ELECTRICAL AND COMPUTER ENGINEERING

OLD DOMINION UNIVERSITY
December 2008

Approved by:

Lee A. Belfore, II (Director)

K. Vijayan Asari

Jessica R. Crouch

Frederic D. McKenzie

ABSTRACT

RAPID PROTOTYPING FOR VIRTUAL ENVIRONMENTS

Emre Baydogan
Old Dominion University, 2008
Director: Dr. Lee A. Belfore, II

Development of Virtual Environment (VE) applications is challenging where application developers are required to have expertise in the target VE technologies along with the problem domain expertise. New VE technologies impose a significant learning curve to even the most experienced VE developer. The proposed solution relies on synthesis to automate the migration of a VE application to a new unfamiliar VE platform/technology. To solve the problem, the Common Scene Definition Framework (CSDF) is developed, that serves as a superset/model representation of the target virtual world. Input modules are developed to populate the framework with the capabilities of the virtual world imported from VRML 2.0 and X3D formats. The synthesis capability is built into the framework to synthesize the virtual world into a subset of VRML 2.0, VRML 1.0, X3D, Java3D, JavaFX, JavaME, and OpenGL technologies, which may reside on different platforms. Interfaces are designed to keep the framework extensible to different and new VE formats/technologies. The framework demonstrated the ability to quickly synthesize a working prototype of the input virtual environment in different VE formats.

To

My Family

ACKNOWLEDGMENTS

I would like to acknowledge and extend my heartfelt gratitude to my advisor, Dr. Lee A. Belfore II, for his support and encouragement, which has made the completion of this dissertation possible.

I would also like to thank Dr. K. Vijayan Asari, Dr. Jessica R. Crouch, and Dr. Frederic D. McKenzie for serving on my dissertation committee and for their constructive comments and feedback.

My sincere gratitude also goes to Dr. M. Borahan Tumer, who advised me during my Master's years at Marmara University, and who encouraged me to start my PhD education endeavor in the United States.

This project was started together with my colleague, Prabhu Krishnan. Many thanks go to him for helping me build a strong foundation on which to add my research ideas.

I would also like to thank my parents, my brothers, and my sister, who have always helped me ascend during difficult times, by supporting me morally and financially.

My appreciation also goes to my good friends, Catalina, Hector, Marija, Ulas, Keremcan, and Saurav, who kept me true to who I am and who offered their valuable time for moral support and discussions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Virtual environment (VE) applications are used in a wide variety of areas and can be deployed on many different platforms. A VE can be defined simply as "an environment that is simulated by a computer which includes aspects of reality." Since most of the VEs provide primarily visual experiences, much of the work is done in the improvement or construction of visual environments. Applications of VEs have been employed in many domains (e.g., medicine, military and training), and have greatly benefited from the continous improvements in computer systems, processing power, and network bandwidth. New directions and interesting projects are now available to be explored by researchers and workers in the area of VEs, and different types of VE development technologies and platforms have been created to meet the great demand for VE in many areas.

I.1 CURRENT FRAMEWORKS

There are currently several tools and environments that serve as development and prototyping platforms to VE technologies. VR Juggler, DIVERSE (Device Independent Virtual Environments Reconfigurable, Scalable, Extensible), InTml (Interaction Techniques Markup Language), Unicron CVE (Collaborative 3D Virtual Environments) and ALICE (Rapid Prototyping for Virtual Reality) are some examples of

The reference model for this work is *IEEE Transactions on Visualization and Computer Graphics.*

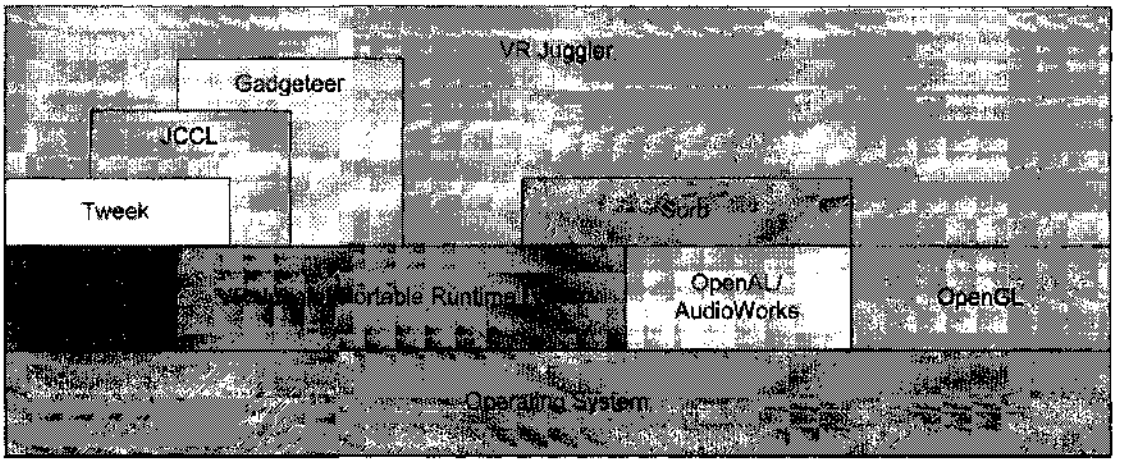


FIG. 1: Layered architecture of the Juggler Suite (Reproduced from [1]).

such tools and environments. A brief overview of each is given in the following sections.

I.1.1 VR Juggler

VR Juggler defines a system–independent virtual platform which provides the ability to deploy VE applications onto different platforms [1]. Figure 1 shows the layered architecture of the Juggler Suite and pieces of the foundation upon which it is built. VR Juggler hides the low level complexities of the target platforms, such as desktop Virtual Reality (VR), HMD (Head Mounted Display), Cave Automatic Virtual Environment (CAVE) like devices, and Powerwall–like devices [2]. A universal way of describing 3D environments, employed by VR Juggler, facilitates the platform independence.

```
#include <dpf.h>
int main(void) {
  dpf *app = new dpf;
  app->config();
  app->display()->world()->addChild(pfdLoadFile("model.pfb"));
  while(app->state & DTK_ISRUNNING)
    app->frame();
  delete app;
  pfExit();
  return 0;
}
```

FIG. 2: DIVERSE sample code.

I.1.2 DIVERSE

DIVERSE provides a platform for distributed, device–independent VE applications [3]. A program created by DIVERSE can be run on a CAVE, ImmersaDesk, HMD, desktop, and laptop systems without requiring modifications to the VE application. The modular design of DIVERSE allows the user to use the different modules separately. Thus, DIVERSE may be used in conjunction with many other APIs and toolkits, like OpenGL, OSG (Open Scene Graph), etc. There is also support for distributed computing, and distributed rendering via means of "remote shared memory," and "message passing." The code fragment in Figure 2 runs equally well in a CAVE, ImmersaDesk, HMD, and on desktop/laptop, without modification.

I.1.3 INTML

InTml is a language designed to describe VE applications in a platform independent and toolkit-independent manner [4]. In addition, InTml can be used to work with

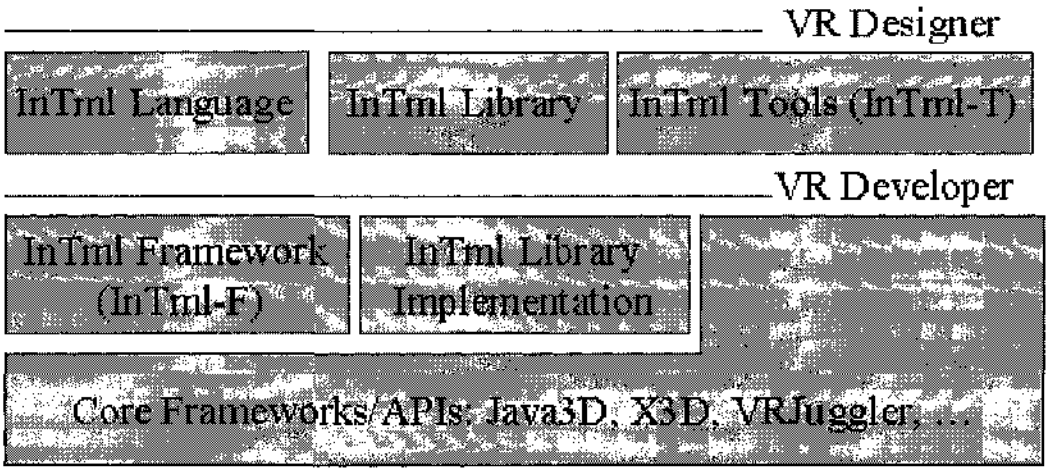


FIG. 3: InTml Architecture [5].

several VR toolkits for rapid prototyping of VE applications. Figure 3 describes the overall architecture of an implementation in InTml. It consists of a domain specific XML-based language to describe VR, and AR (Augmented Reality) applications. Hence, high level elements of an application are used: references to virtual objects, to devices, and to interaction techniques, and not to VR low-level components (I.e., geometry and texture, haptic capabilities).

I.1.4 Unicron CVE

Unicron CVE is a rapid development platform for VEs that focuses on developing 3D collaborative environments for education [6]. Unicron includes three APIs: a custom simple 3D language, a network, and the audio modules. The system includes environment builder tools, collaboration tools, and a class library for the infrastructure

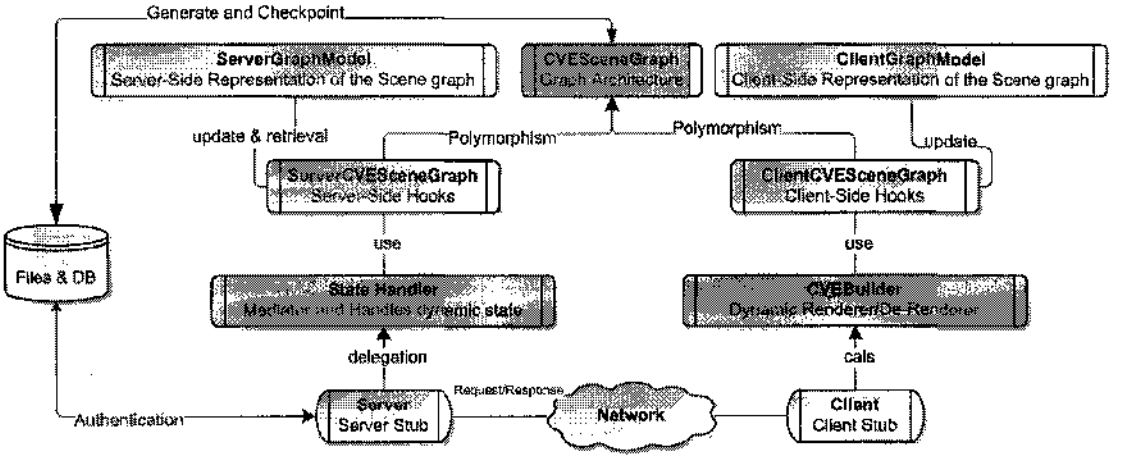


FIG. 4: Unicron CVE Architecture (Reproduced from [6]).

of the 3D environment (e.g., doors, whiteboards, avatars, etc.). For the low 3D requirements of the collaborative environment, a very high level simple text–based 3D language is developed. Figure 4 shows several aspects of the Unicron client–server architecture.

I.1.5 ALICE: Rapid Prototyping for Virtual Reality

ALICE is a prototyping system for VR, developed by User Interface Group at the University of Virginia [7]. Alice provides an authoring environment to create VEs by using object manipulation interactively, programmatically (Python), or via a graphical user interface. The objects present in the VE are stored in a scene graph tree, as can be noted in Figure 5. The sample hierarchy illustrated in Figure 5 produces a VE such as the one in Figure 6.

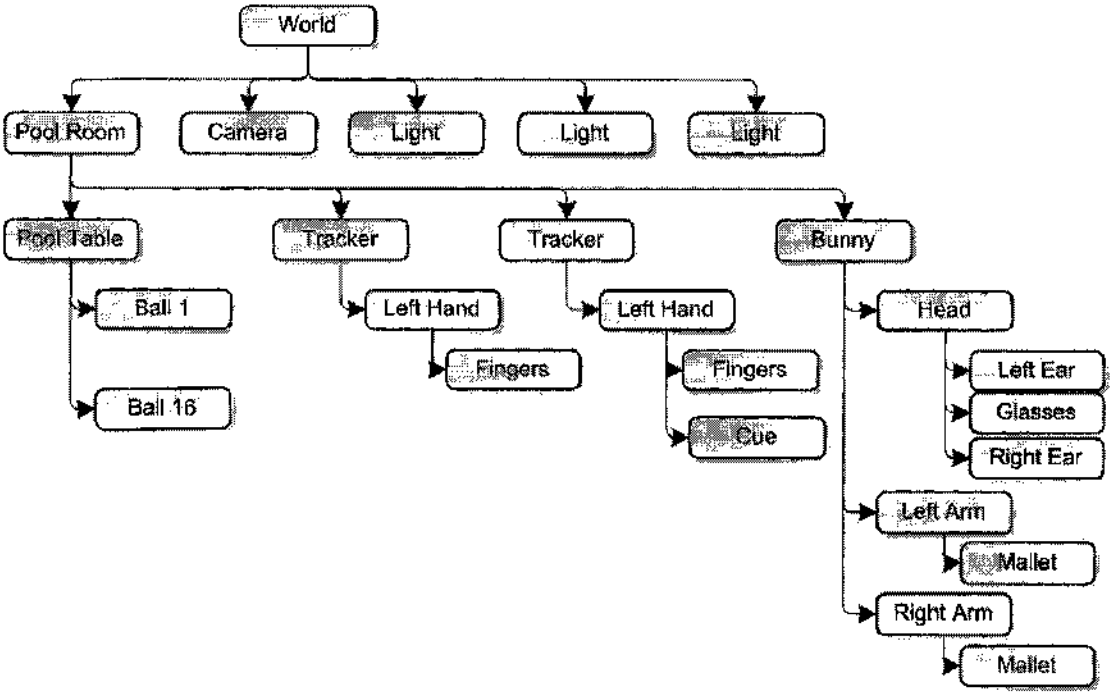


FIG. 5: ALICE hierarchy sample (Reproduced from [7]).

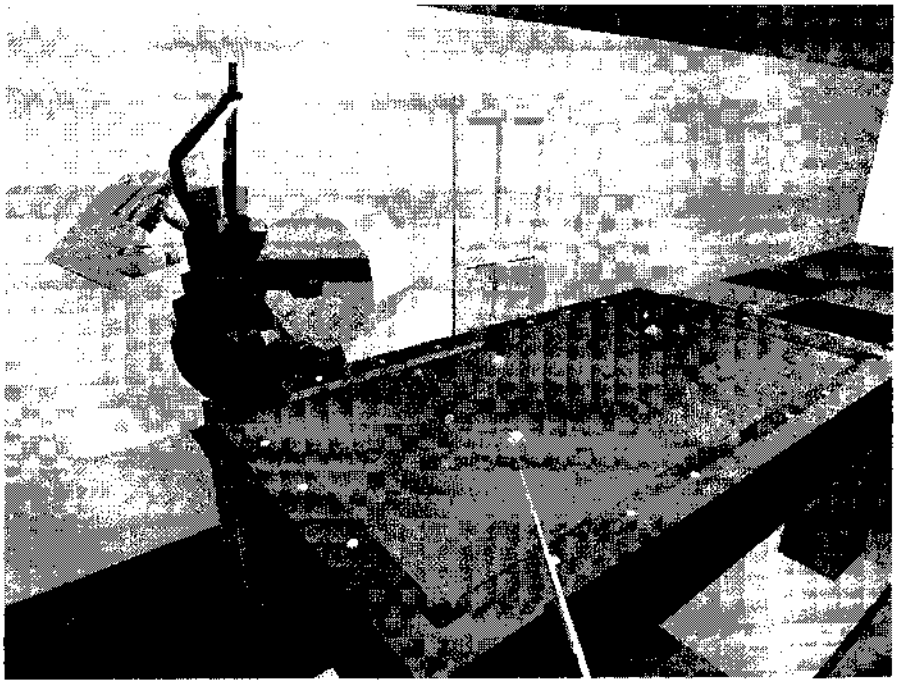


FIG. 6: ALICE sample [7].

I.2 METHODOLOGY: RAPID PROTOTYPING

Rapid prototyping is used often in non-physical systems, as in the case of software systems [8, 9]. In software prototyping, incomplete versions of software are created as prototypes. A software prototype typically implements a small subset of the final software/program. This implementation might be different from the eventual final software implementation, since the traditional programming languages might be costlier than prototyping languages [10].

There are two major types of prototyping: throwaway prototyping and evolutionary prototyping [8]. In the throwaway prototyping, the prototype is eventually discarded rather than becoming a part of the final product. The purpose of this type of prototyping is to allow the developers to evaluate possible proposed designs quickly and efficiently [11]. Such an evaluation is performed for proof of concept verification and to test key components and functionalities of the final product. On the other hand, in evolutionary prototyping, the main purpose is to create a very robust prototype so that it can be revised iteratively until it becomes the final product [12, 13]. Iterative prototyping helps in defining user requirements at the design level and provides continuity from the initial concept to the finished application.

Development of VEs is very similar to software development. A VE is a software system that represents and simulates a physical system via use of computer graphics. Using rapid prototyping methodology to quickly develop a VE is an attractive strategy for the developer. Such a methodology would allow using such technologies

to a developer, who must use a VE technology with which he or she is not familiar for various reasons such as project constraints, and specific application needs. As applied in several domains such as the manufacturing sector, the developer may need a throwaway–prototype for proof of concept demonstrations in a new unfamiliar technology. Moreover, using synthesis tools that automate the porting of the existing technology, a prototype may be rapidly created on an alternate VE platform. The synthesized prototype may be potentially less efficient than a product that is the outcome of a monolithic development cycle, but may enable the developer to be able to make high level design decisions that may reduce the overall design time for the desired final product. This knowledge gained from the prototyping phase can enable the creation of a more robust and efficient final VE and also enable verification of requirements and specification of the VE implementation. Rapid prototyping is discussed in further detail in §II.2 and §III.1.

I.3 PROBLEM STATEMENT

Some of the problems deploying VE content include the following:

- Application developers are usually expected to have expertise in many different technologies.
- VE applications bring together a diverse range of hardware and software tools required by VE application developers. Integrating these tools and their products to serve as a single entity is a crucial step which proves to be challenging.
- Significant expertise is required to use these tools and the platforms in which

they reside. Attaining expert proficiency increases the time necessary for developing the VE solution.

- Collaborative work and communication among developers working with different software tools have always been problematical.

One of the solutions to the problems listed above is the creation of a rapid prototyping environment to aid the developers in building VE applications on many platforms. Some of the benefits of such an environment are as follows.

- A single development environment eliminates the need to learn a variety of software/hardware platforms, by defining the capabilities for each platform in a common representation that incorporates their definitions.

- The design time is reduced, by minimizing or eliminating the training required for new hardware/software platforms, as well as by providing optimizations for a specific platform through the stored knowledge of the target platform.

- The very same rapid development environment might also help communication and collaboration of the developers who have expertise in different tool sets and platforms.

A developer who is not an expert in a specific technology might have to use the technology for various reasons such as project constraints, specific application need, etc. Moreover, the developer may be required to build a prototype using an unfamiliar technology just for the concept demonstration purposes.

As a solution, a rapid prototyping framework, Common Scene Definition Framework (CSDF), is developed, that serves as a superset/model representation of the target virtual world. Input modules are developed to populate the framework with the capabilities of the virtual world imported from VRML 2.0 and X3D formats. The synthesis capability is built into the framework to synthesize the virtual world into a subset of VRML 2.0, VRML 1.0, X3D, Java3D, JavaFX, JavaME, and OpenGL technologies, which may reside on different platforms. If needed, VEs can be authored in this superset of all technologies to ensure that maximum number of capabilities are supported on target VEs. Another important feature of the prototyping system is its knowledge of the capability limitations for a particular synthesis platform.

I.4 PROS AND CONS OF PRESENT FRAMEWORKS AND CSDF

In contrast to current rapid prototyping and development environments, the proposed framework and its encapsulating modules provide VE applications with language and format–independence, as well as platform and hardware independence. Theoretically, through the use of the corresponding import module, any existing virtual world, 3D environment, or model can be imported into the proposed framework (CSDF) to be modified or synthesized for output of different platforms/languages. This is one of the most significant advantages over other existing prototyping/development environments (i.e., VR Juggler, DIVERSE, InTml, Unicron CVE and ALICE).

Table I shows features and differences of current frameworks and CSDF.

TABLE I: Features and Differences among Frameworks.

Feature	VRJuggler	DIVERSE	INTML	UnicronCVE	ALICE	CSDF
The framework allows arbitrary technology for development.			✓			✓
The framework is device independent.	✓	✓	✓			✓
The framework is extensible.	✓	✓				✓
The framework supports low level VE elements.	✓	✓				✓
The framework allows develepment inside the framework representation.				✓	✓	✓

CHAPTER II

BACKGROUND

In this chapter, discussions and descriptions on thesis domain, technologies, methodologies, and platforms are presented. Brief explanations of VEs and Prototyping are given. Furthermore, technologies that can be used to create VEs are described, as well as the platforms in which these technologies are used. Even though a few of these technologies and platforms are not directly used, they are in close relation to the thesis domain and describe the proof of concept.

II.1 3D VES

A VE is a computer generated environment that allows human-to-computer interaction. Most VEs are displayed either on computer monitors or projected surfaces. Some VEs provide multi modal interaction including, for example, audio and haptic interaction capabilities. Audio interaction is provided by speakers or headphones, whereas haptic interaction can be provided by special hardware that can simulate tactile feedback. Indeed, users can interact with VEs through input devices (i.e. keyboard, mouse, and enhanced human-computer-interface devices). VEs can be similar to real world, representing real world scenarios, or they can be very different from real world, as in fictional games that uses VEs.

In practice, it is currently very difficult to create a high-fidelity VR experience,

due largely to technical limitations on processing power, image resolution and communication bandwidth. However, those limitations are expected to eventually be overcome as processor, imaging and data communication technologies become more powerful and cost-effective over time.

II.1.1 Domains

3D VEs are used in a variety of application domains, such as national defense, team training, academic learning, medicine, manufacturing, and gaming.

National Defense

VE can improve readiness to military organizations by bringing training opportunities that can not be made available in the real world. These systems can reduce training costs by substituting actual equipments and weapons, and even old expensive simulators that are not using virtual technologies. Modern military equipment is very expensive to operate. In addition, some weapons and equipment are rarely used that they are not available for training. VEs can used for several purposes in military field: training, mission rehearsal, evaluation of new concepts and equipment, performance measurement, and knowledge elicitation [14, 15, 16].

Team Training

VEs have potential for individual and team training [17]. VE technology may immerse an individual or a team in computer generated worlds. Then, users are allowed to interact with autonomous agents and/or human actors/team members. Combinations

of multiple trainees and trainers, human or virtual, may exist in in such environments. Thus, effective teamwork training can be achieved for complex tasks.

Academic Learning and Education

It is known that well-designed simulations can provide learning experiences that are not available via normal means. Virtual field trips consume no fuel; simulated laboratories do not explode; virtual dissection kills no animals [18]. VEs provide infinite possibilities of experimentation for their users. Specific skills may be practiced and observed from different viewpoints [19].

Medicine

In the medical domain, the core technology is the use of interactive three–dimensional visualization toward training personnel; and improving the quality of patient care in emergency situations, hospitals, and battlefields. The experience of VEs can be immersive or augmented; with a head mounted display (HMD), on a 3-D video monitor, or in a room size CAVE; stand alone, distributed, or Internet-based. Thus, it is possible to customize the required VE experience for the respective health care provider or patient. The most important benefit is that VEs provide a risk–free experience to practice new procedures that have not been tried on human patients, to train medical professionals, who traditionally practiced on real patients, and to measure performance metrics for specific procedures. VEs are used in diagnosis, therapy, education, and training [20].

Manufacturing Industry

Every type of good available to society, from consumer goods to electronics, computers, and automobiles results from manufacturing. In order to manufacture items, a manufacturing system, which is a collection of machines, equipment, and labor, is required. VEs are beneficial to manufacturing systems as a means of visualization, simulation, information provision (information display via VEs), and telerobotics (remote control of robotics in VEs representing remote real environment) [21].

Gaming and Entertainment

Use of VEs in the gaming and entertainment industries has the longest history among all the domains, dating back to start of the twentieth century [22]. Today, VEs are non-replaceable components of games and other entertainment platforms. Computer games may run on many different, platforms including arcade machines, personal computers, gaming consoles, and palmtop devices. Even though early games were two-dimensional (2D), with the improvements on three-dimensional (3D) computer graphics technology, the domain moved into using truly interactive 3D gaming. Along with the gaming industry, today, many branches of the entertainment industry, such as the film industry, adopted VEs. The push from the gaming industry has resulted in the leading companies making large investments in game engine design, terrain management, clustering, data segmentation, etc. In addition, gaming and computer graphics hardware companies made large improvements in a wide variety of hardware/software combinations to satisfy the hunger of the industry and to compete

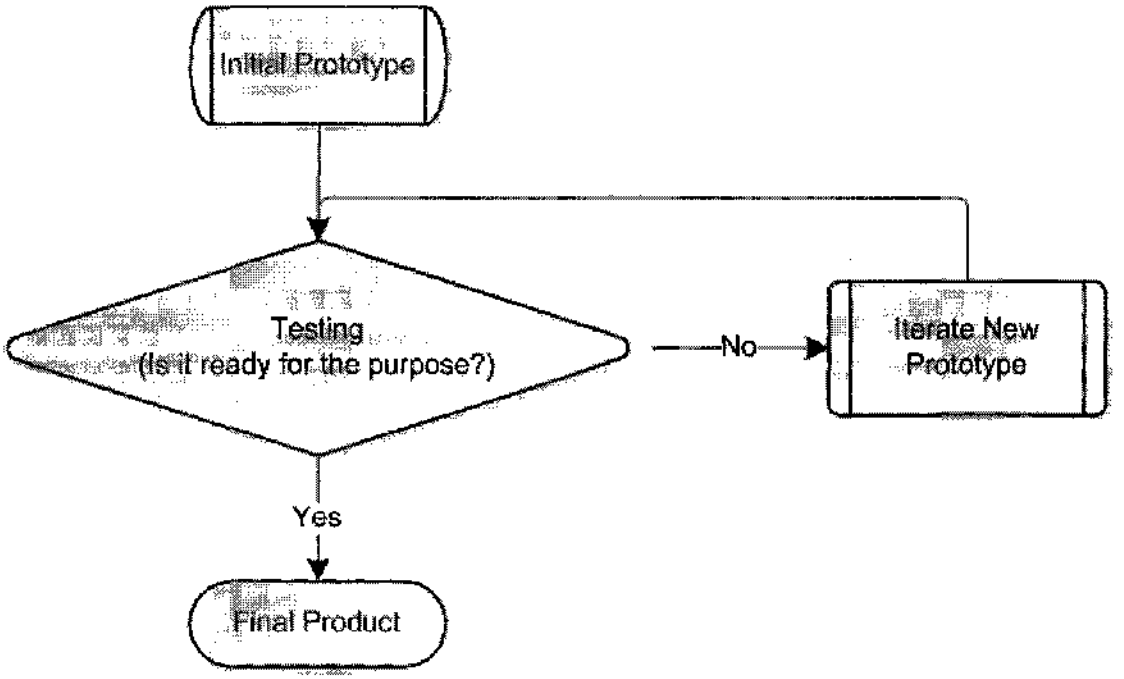


FIG. 7: Prototyping Process.

with rival companies.

II.2 PROTOTYPING

Prototyping is the process of quickly creating a working model in order to test a design, demonstrate ideas and/or features with potential users. In a development cycle one or more prototypes are generated iteratively (Figure 7), where each subsequent prototype is affected by the assessment of the previous one [9, 23, 24, 25]. Hence, the deficiencies of the earlier prototypes are eliminated. The target product is ready when the prototype meets the initial requirements in terms of functionality and robustness.

The advantages of using prototyping in system design are [8, 26, 27]:

- May provide the proof of concept necessary to attract funding

- Early visibility of the prototype gives users an idea of what the final system looks like
- Encourages active participation among users and producer
- Enables a higher output for user
- Cost effective (Development costs reduced)
- Increases system development speed
- Assists to identify any problems with the efficacy of earlier design, requirements analysis and coding activities
- Helps to refine the potential risks associated with the delivery of the system being developed
- Various aspects can be tested and quicker feedback can be gotten from the user
- Helps to deliver easily the product in quality
- High end-user involvement
- Early detection of design issues
- Early deployment of end product

Moreover, the disadvantages are [8, 26, 27]:

- User expectations for the prototype may be beyond its performance
- Possibility of causing systems to be left unfinished

- Possibility of implementing systems before they are ready
- Producer might produce a system inadequate for overall organization needs
- User can get too involved, whereas the program can not reach a high standard
- Structure of system can be damaged since many changes could be made
- Producer might get too attached to it
- Not suitable for large applications

II.3 TECHNOLOGIES

There are several alternative technologies to construct VEs. Each of these competing technologies has its inherent strengths and weaknesses compared with the other technologies. The choice of which technology or delivery medium to use is dictated by several factors including:

- Capabilities required by the specific application or domain
- Nature of the required application (stand alone, Web capable)
- Ease of distribution (Web enabled vs. distribution on CDROMs etc)
- Proprietary software interfaces such as plug-ins required to interact
- Capabilities of end user system (high end graphics workstation vs. PDA etc.)

A discussion of some of the important 3D formats, APIs, and development environments are presented in the following subsections.

TABLE II: Node Types and Nodes in VRML 1.0 [30].

Node Type	**List of nodes**
Shape nodes represent the shape nodes that specify the geometry	AsciiText, Cone, Cube, Cylinder, IndexedFaceSet, IndexedLineSet, PointSet, Sphere
Property nodes represent the properties of the geometry and its appearance, and matrix or transform properties	Coordinate3, FontStyle, Info, Material, MaterialBinding, Normal, NormalBinding, Texture2, Texture2Transform, TextureCoordinate2, ShapeHints, MatrixTransform, Rotation, Scale, Transform, Translation
Group Nodes used to create aggregate objects using the single or other group nodes	Separator, Switch, WWWAnchor, LOD
Light Source nodes used to add different light sources within the VRML 1.0 scene	DirectionalLight, PointLight, SpotLight
Camera Nodes are used to specify the type of projection system for the view.	OrthographicCamera, PerspectiveCamera

II.3.1 VRML

Virtual Reality Modeling Language (VRML) [28], originally Virtual Reality Markup Language, is a text–based file format for representing 3D interactive vector graphics. Initial design of VRML was made for the World Wide Web [29] and based on the Open Inventor file format developed by Silicon Graphics Inc. The initial version, namely VRML 1.0, was completed in May 1995 and provided basic support for shapes, lighting, surface color, UV mapped textures, shininess, transparency. Table II lists node types and nodes in VRML 1.0. VRML 1.0 could be used to develop passive VR systems where, once developed, the virtual world scene presents a read-only view to the audience. The user was not able to author or interact with the scene, unless the user generated an entirely new content file.

```
#VRML V2.0 utf8

# Red cone

Shape {
  appearance Appearance {
    material Material {
      diffuseColor 1 0 0
    }
  }
  geometry Cone {
    bottomRadius 0.75
    height 1.6
  }
}
```

FIG. 8: VRML sample code

In 1997, VRML 2.0 was created to extend the capabilities of VRML 1.0. VRML 1.0 was among the first open formats for 3D objects and provided only very primitive capabilities for user interaction. The draft of the VRML 2.0 specification also known as VRML97 was accepted as an ISO standard.

VRML files are commonly called "worlds" and have the **.wrl* extension. In addition, VRML worlds use a text format, so they may often be compressed using GZIP so that they transfer over the Internet more quickly . Figure 9 depicts the representation of the VRML 2.0 code listed in Figure 8, using ParallelGraphics' Cortona VRML client for Microsoft Internet Explorer.

VRML [31, 32, 33] employs tree based structure, namely scene graph structure, to describe 3D objects (vertices and edges for a 3D polygon) along with surface color, UV mapped textures, shininess, and transparency within the virtual world. VRML

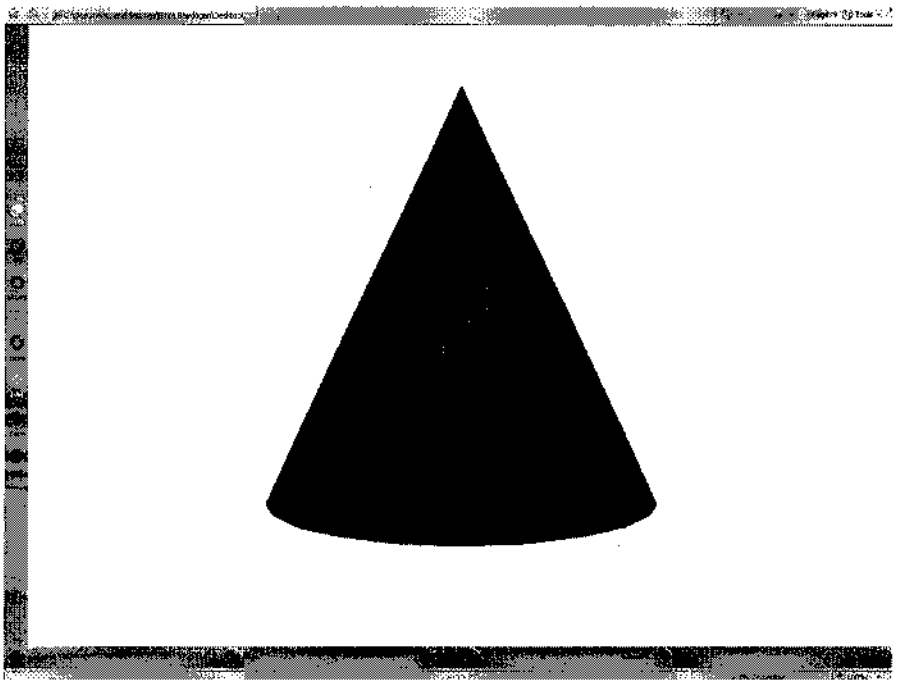


FIG. 9: VRML Sample View in Cortona Player.

specification allows VRML nodes, which are essentially the entities in the scene graph structure, to communicate with each other along with multiple referencing of the existing node. Multiple referencing is provided by use of *DEF* and *USE* keywords. Any node labeled with `[DEF <name>] <NodeType> {<body>}` may be referenced later via use of *USE* keyword. Thus, a pointer link is created to previously declared *DEF* name in place of *USE* node. The communication among scene graph nodes is provided via an event mechanism called Route statements. Hence, source and destination of the events in the environment are connected.

Summarized below are some key enhancements altered by VRML97 compared with VRML 1.0.

- Static type
  - *Sound*, *MovieTexture* and *AudioClip* nodes allow addition of multimedia experience.
  - *Extrusion* node describes a rectangular array of varying height and allows for the modeling of terrain.
  - *Background* node is to describe the background view of the environment.
  - New *Shape* node allows encapsulation of other geometry type nodes.
- Interaction type
  - *TimeSensor* node provides a timer to trigger other events.
  - *PlaneSensor* node maps a pointing device motion into a two-dimensional translation.
  - *ProximitySensor* node allows testing of proximity of the user view to the scene object in question.
  - *Collision* node can detect collision of the geometries or a geometry with an avatar.
- Behavior type
  - *Interpolator* nodes are used to perform calculations.

- *Script* node can contain functions written in programming languages such as Java or Javascript.

- Prototyping type

  - *Proto* and *Externproto* nodes allows developer to extend the VRML specification by creating new re–usuable objects that consist of multiple existing VRML nodes

II.3.2 X3D

X3D is an open–source standards file format and run–time architecture, which builds on VRML97. X3D represents and communicates 3D scenes and objects using Extensible Markup Language (XML) [34], a general-purpose specification for creating custom markup languages. X3D is under active development and ISO ratification [35], latest specification dating April 2008, through the Web3D consortium [36]. X3D system specifies storage, retrieval and playback of real time graphics content embedded in applications within variety of domains.

The use of XML encoding provides easy integration to Web services, distribution on networks, cross-platform operability and easy file and data transfer. Rich sets of components within X3D can be authored for custom use in domains such as engineering, scientific visualization, medical visualization and multimedia. Moreover, the system is extensible for added functionality for other domains. The component–based architecture of X3D system allows creation of different *profiles*, which can be individually supported, as well as addition of new *levels* and components. Hence,

improvements can be made quickly on a specific area without modifying the overall specification of X3D. The modular blocks of the baseline profiles for X3D architecture are shown in Figure 10.

The four baseline profiles are:

- **Interchange**: profile for communicating between applications (geometry, texturing, basic lighting, and animation).
- **Interactive**: profile for basic interaction with a 3D environment by adding various sensor nodes for navigation, interaction, enhanced timing, and additional lighting (PlaneSensor, TouchSensor, Spotlight, PointLight).
- **Immersive**: profile for full 3D graphics and interaction, including audio support, collision, fog, and scripting.
- **Full**: profile for all defined nodes including NURBS, H-Anim and GeoSpatial components.

Additionally, since the legacy specification for VRML97 is supported, the VRML97 content may be updated or preserved. Table III shows the features supported by X3D system [36]. Figure 12 depicts the representation of the X3D code listed in Figure 11, using Xj3D Browser [38].

II.3.3 Java3D

Java3D is a scene graph based 3D API for the Java programming language available as a separate API since Java 2 Java Development Kit (JDK)[39]. With the Java 3D API,

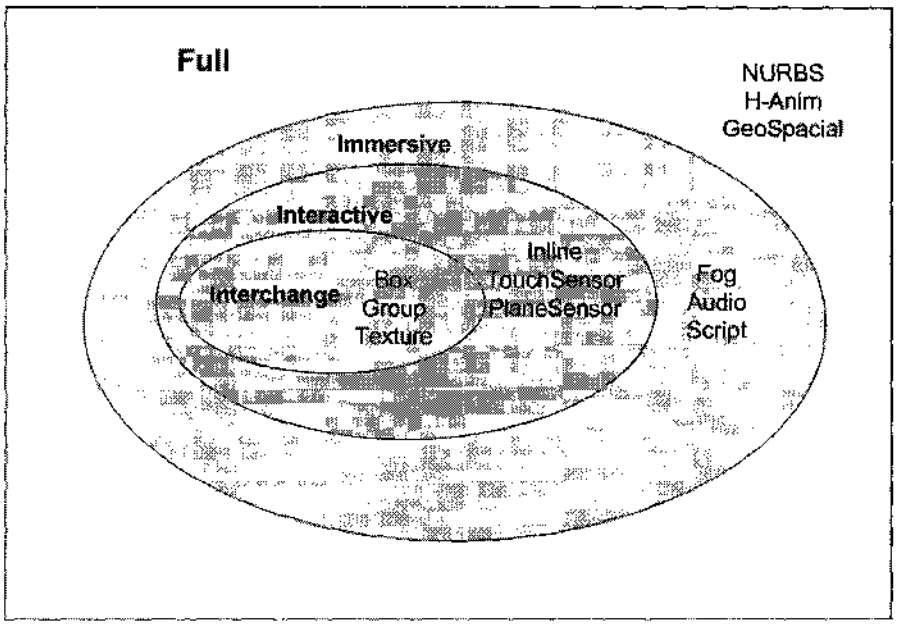


FIG. 10: Modular block of X3D baseline profiles (Reproduced from [37]).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "http://www.web3d.org/specifications/x3d-3.1.dtd"
  "file:///www.web3d.org/TaskGroups/x3d/translation/x3d-3.1.dtd">
<X3D profile="Immersive" version="3.1"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
  xsd:noNamespaceSchemaLocation=
  "http://www.web3d.org/specifications/x3d-3.1.xsd">
  <Scene>
    <Shape>
      <Appearance>
        <Material diffuseColor="0.3 0.5 0.8"/>
      </Appearance>
      <Sphere/>
    </Shape>
  </Scene>
</X3D>
```

FIG. 11: X3D sample code.

TABLE III: X3D Supported Features.

Feature	**Explanation**
3D graphics and programmable shaders	Polygonal geometry, parametric geometry, hierarchical transformations, lighting, materials, multi pass/multi–stage texture mapping, pixel and vertex shaders, hardware acceleration
2D graphics	Spatialized text; 2D vector graphics; 2D/3D compositing
CAD data	Translation of CAD data to an open format for publishing and interactive media
Animation	Timers and interpolators to drive continous animations; humanoid animation and morphing
Spatialized audio and video	Audio visual sources mapped onto geometry in the scene
User interaction	Mouse–based picking and dragging; keyboard input
Navigation	Cameras; user movement within the 3D scene; collision, proximity and visibility detection
User–defined objects	Ability to extend built in browser functionality by creating user–defined data types
Scripting	Ability to dynamically change the scene via programming and scripting languages
Networking	Ability to compose a single X3D scene out of assets located on a network; hyperlinking of objects to other scenes or assets located on the World Wide Web
Physical simulation and real time communication	Humanoid animation; geospatial datasets; integration with Distributed Interactive Simulation (DIS) protocols

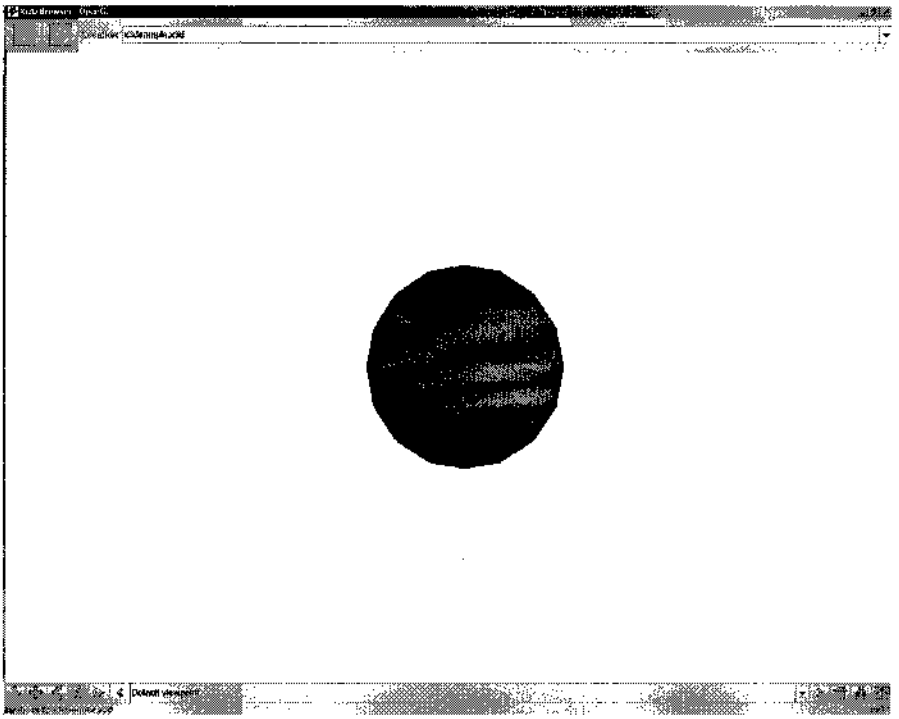


FIG. 12: X3D Sample View in Xj3D Browser.

high-quality, scalable, platform independent 3D graphics can be incorporated into Java applications and applets. Additionally, Java 3D offers extensive sound support. Java3D provides display and interaction with 3D graphics via means of a collection of high level constructs for creating and manipulating 3D geometry and structures [40, 29]. These geometries and structures reside in a virtual universe represented by a scene graph before they are rendered using native OpenGL or Direct3D libraries. To define a Java3D scene Java programming language is used. Then, in the run–time, the program creates instances of a Java3D object and places them in the scene graph tree structure.

Figure 13 shows a sample Java3D scene graph [41]. *VirtualUniverse* object is the root node of the scene graph, representing the largest unit of aggregate representation

of the scene. A *Locale* object is attached to a *VirtualUniverse* object and provides a coordinate point of reference for objects in a scene and thus serves as the origin for scene graph objects attached to it. The *BranchGroup* object is a grouping node that acts as the root node for other grouping nodes, or geometry nodes, which can be attached to it. *BranchGroup* node is the only grouping node in Java3D that can be attached to a *Locale* object.

The *TransformGroup* object is a grouping node that specifies a transformation that can change the position, orientation, or scale of all of its children nodes. Thus, geometric primitives must be children of a *TransformGroup* object so that the transformation can be applied. Interactive behavior can be built into the scene graph by incorporating sensors with *TransformGroup* objects. For example, to create a scene having a cube that can rotate in response to a mouse trigger, the event handler for the mouse event applies a transform to the *TransformGroup* object, thus changing the position, and orientation of the cube. Figures 14 and 15 depict a sample Java3D program that displays a rotated 3D Cube object.

Some of the Java3D API features [40, 42] are:

- Platform-independent, cross-platform
- Support for retained, compiled-retained, and immediate mode rendering
- Includes hardware-accelerated OpenGL and Direct3D renderers (depending on platform), as well as JOGL, a wrapper library that allows OpenGL in Java programming
- Sophisticated virtual-reality-based view model with support for stereoscopic

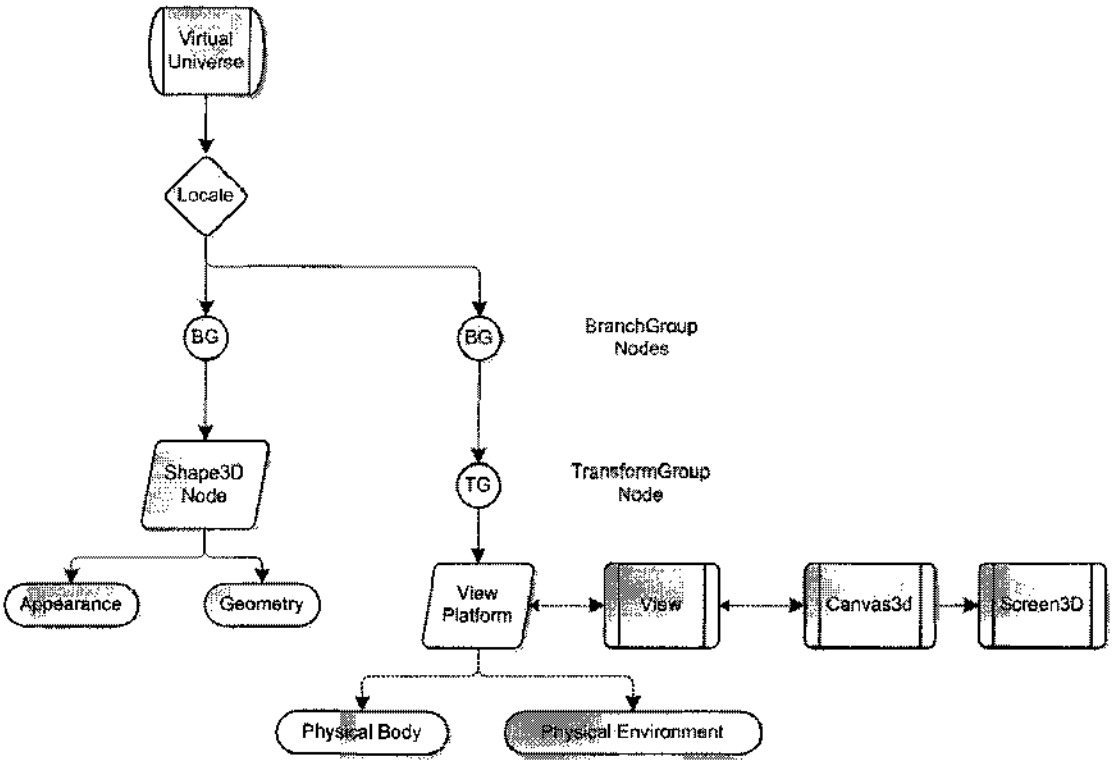


FIG. 13: Java3D Sample Scene Graph Diagram.

rendering and complex multi-display configurations

- Native support for head-mounted display
- CAVE support(multiple screen projectors)
- 3D sound support
- Support for shaders, set of software instructions, which is used by the graphic resources primarily to perform rendering effects

```
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.vecmath.AxisAngle4f;
import com.sun.j3d.utils.geometry.ColorCube;
import com.sun.j3d.utils.universe.SimpleUniverse;

public class sample3d {
  public sample3d() {
    SimpleUniverse universe = new SimpleUniverse();
    BranchGroup group = new BranchGroup();
    TransformGroup tg = new TransformGroup();
    Transform3D transform = new Transform3D();
    transform.setRotation(new AxisAngle4f(0.7f, .7f, 0.7f, .7f));
    tg.setTransform(transform);
    tg.addChild(new ColorCube(0.3));
    group.addChild(tg);
    universe.getViewingPlatform().setNominalViewingTransform();
    universe.addBranchGraph(group);
  }

  public static void main(String[] args) {
    new sample3d();
  }
}
```

FIG. 14: Java3D Sample Code.

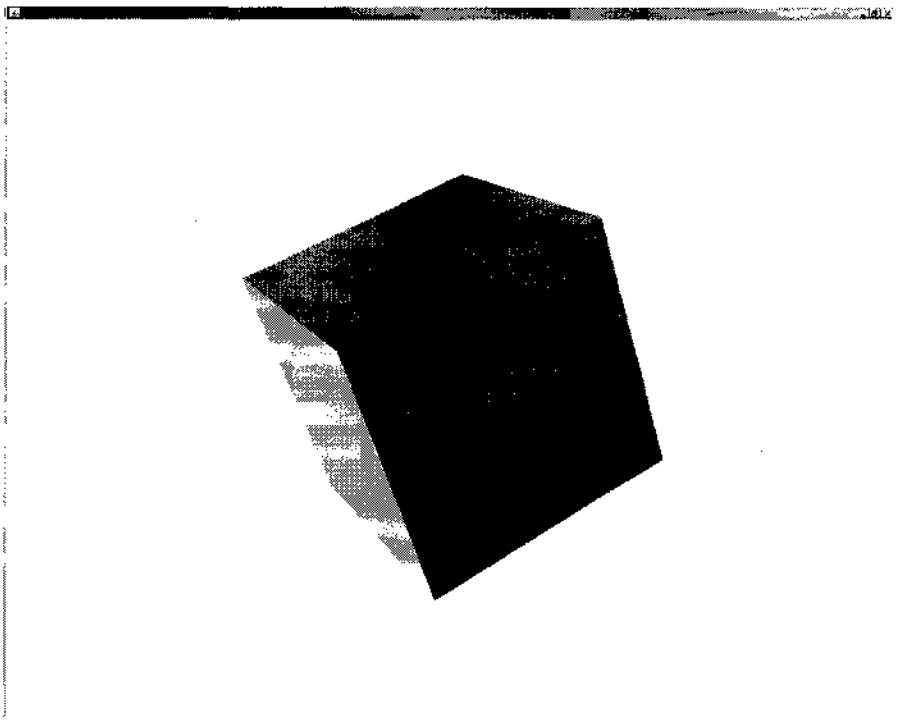


FIG. 15: Java3D Sample View.

II.3.4 JavaFX

JavaFX is a family of products and technologies from Sun Microsystems, announced in May 2007 [43]. Currently JavaFX consists of JavaFX Script, JavaFX Desktop, and JavaFX Mobile. JavaFX provides simplified and rapid content creation accross browser, desktop and mobile platforms. It is possible to create 3D VEs via tools provided JavaFX [44] in addition to advanced enterprise and Internet applications. JavaFX allows use of most components in Java3D API. Java3D code needs to be leveraged and encapsulated so that it can be used in platforms supporting JavaFX. Already, a number of new mobile and portable devices are adopting JavaFX functionality [43]. Figure 16 shows a sample JavaFX script displaying a Cube object.

```
package javafx3d;
import javafx.ui.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.lang.Math;
import java.awt.BorderLayout;
import javax.swing.JPanel;
class RenderPanel extends Panel {}
operation RenderPanel.createComponent() {
var config = SimpleUniverse.getPreferredConfiguration();
    var c = new Canvas3D(config);
    var univ = new SimpleUniverse(c);
    univ.getViewingPlatform().setNominalViewingTransform();
    univ.getViewer().getView().setMinimumFrameCycleTime(5);
    var objRoot = new BranchGroup();
    objRoot.addChild(new ColorCube(0.4));
    objRoot.compile();
    univ.addBranchGraph(objRoot);
    var panel=new JPanel();
    panel.setLayout(new BorderLayout());
    panel.add(c, BorderLayout.CENTER);
    return panel;
}
 var renderPanel=new RenderPanel();
 Frame {
     visible: true
     screenx: 50
     screeny: 100
     width: 600
     height: 400
     content: BorderPanel {
         border: EmptyBorder {left: 16 top: 16 right: 16 bottom: 16}
         center: BorderPanel {
             background: new Color(1,1,1,1)
         }
         center: renderPanel
     }
}
```

FIG. 16: JavaFX sample script.

II.3.5 M3G (Mobile 3D Graphics API)

The Mobile 3D Graphics API (M3G) is an API for creating 3D computer graphics on Java Platform, Micro Edition (JavaME) [45]. M3G extends the capabilities of JavaME so that 3D computer graphics can be produced on embedded devices such as mobile phones and PDAs. In contrast to Java3D, M3G is designed for low memory and processing power devices. Thus Java3D and M3G are incompatible and seperate. M3G API was designed by JSR expert group including Sun Microsystems, Sony Ericsson, Symbian, Motorola, ARM, Cingular Wireless, and specification lead Nokia [46]. The API has 30 classes (I.e., World, Object3D, Light, Camera, etc.) that can be used to draw complex animated three-dimensional scenes. M3G has two modes of rendering; immediate and retained. In immediate mode, graphics commands are issued directly into the graphics pipeline and the rendering engine executes them immediately. The developer has to control each key frame of the animation in immediate mode. On the other hand, retained mode uses a scene graph that links objects in the 3D world in a tree structure, and specifies world much like Java3D and X3D. Figure 17 lists a code piece from a M3G application in retained mode. The M3G standard also specifies a file format (".m3g" extension) for 3D model data, including animation data. This allows developers to create content on PCs that can be loaded by M3G on mobile devices.

```
public class MyCanvas extends Canvas
    Graphics3D g3d;
    World world;
    int currentTime = 0;

    public MyCanvas() {
        g3d = Graphics3D.create();
        Object root[] = Loader.load("world.m3g");
        world = root[0];
    }

    protected void paint(Graphics g) {
        g3d.bindTarget(g);
        world.animate(currentTime);
        currentTime += 50;
        g3d.render(world);
        g3d.releaseTarget();
    }
}
```

FIG. 17: M3G code piece (Retained Mode) [46].

II.3.6 JavaME (Java Platform, Micro Edition)

The Java Platform, Micro Edition (JavaME) is a specification of a subset of the Java platform, which provides a collection of Java APIs for the development of software for small, memory and computation constrained embedded devices such as mobile phones, PDAs and set–top boxes [47]. Sun Microsystems, the designer of Java and JavaME, provides a reference implementation of the specification, but does not provide free binary implementations of its JavaME runtime environment for mobile devices. The implementation is left to third parties or to device manufacturers, to provide their own.

JavaME is widely used for creating games for cell phones, since they might be

emulated on a PC during the development stage and easily uploaded to phones in contrast to development and testing for other gaming consoles such as those made by Nintendo, Sony and Microsoft, as expensive system-specific hardware and kits are required.

There are currently two types of profiles, subsets of configurations: the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC). CLDC is the strict subset of Java class libraries that requires minimal JVM operation. On the other hand, CDC contains almost all the libraries Java standard edition. Thus CDC is richer than CLDC.

Two important APIs for creating VEs on such micro devices are Mobile 3D Graphics API (M3G) (§II.3.5) and Java Binding for OpenGL ES. Java Binding for OpenGL ES implements OpenGL ES common profile on supported devices.

II.3.7 OpenGL (Open Graphics Library)

Open Graphics Library (OpenGL) is an API for developing applications that create 2D and 3D computer graphics. OpenGL was originally developed by Silicon Graphics Inc. (SGI) in 1992 [48, 49]. OpenGL is implemented over different languages for different platforms. OpenGL, under its specification, has a little over 250 function calls that can be used to construct 3D scenes [50]. Each hardware vendor has to implement the OpenGL specification on its hardware in order to open the hardware to OpenGL development. There are implementations for variety of platforms, such as Microsoft Windows, Linux and Mac OS. OpenGL hides the complexities of the different variety of underlying 3D accelerators by requiring that all implementations

support OpenGL feature set. OpenGL forces the implementation to emulate the feature via software where the feature does not exist on the hardware in question.

OpenGL uses a graphics pipeline, OpenGL state machine, that accepts primitives (I.e., points, lines, and polygons) and converts them into pixels. In OpenGL, the programmer has to dictate the exact steps required to render a scene in contrast to scene graph (retained mode in Java3D) APIs, where the programmer only describes the scene leaving the rendering to the API. Thus OpenGL programmer needs to have a good knowledge of the graphics pipeline and rendering algorithms. Figure 18 depicts a simplified version of the OpenGL graphics pipeline [51, 48]. In the pipeline, if necessary, polynomial functions are evaluated for certain inputs such as calculation of NURBS surfaces, approximation of curves and surface geometry. Then, vertex operations (e.g., transformation and lighting) are done as well as clipping non-visible parts. Third, previous information is turned into pixels by rasterization process. Fourth, per-fragment operations, like updating values depending on incoming and previously stored depth values or color combinations, among others, are done. Lastly, the fragments are inserted into frame buffer.

For handling of events such as key press, mouse movement, mouse button press and resize window, OpenGL relies on the underlying operating system. This adds the complexity of the development for the OpenGL developer. Several libraries are built on top of OpenGL to provide features not available in OpenGL (e.g., OpenGL Utility Toolkit (GLUT), OpenGL Utility Library (GLU), Simple DirectMedia Layer (SDL), OpenGL User Interface Library (GLUI) and Fast Light Toolkit (FLTK)). There are also other libraries to enable scene graph (retained mode) in OpenGL such

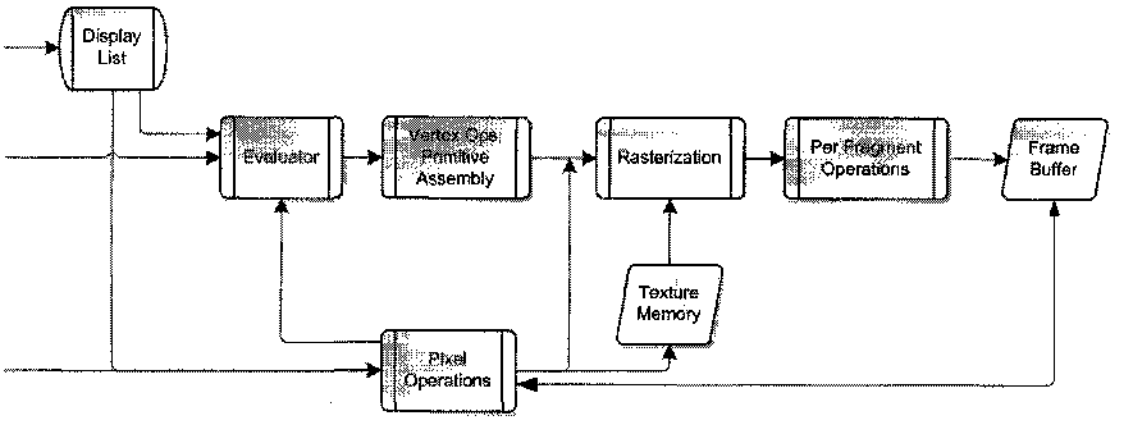


FIG. 18: OpenGL Graphics Pipeline Process.

as OpenSG, OpenSceneGraph and OpenGL Performer.

Figures 19 and 20 depict a sample OpenGL program that displays a 3D colored cube object.

II.3.8 DirectX – Direct3D

Microsoft DirectX is a series of APIs for Microsoft platforms to handle tasks related to multimedia. DirectX is widely used for game programming/development and video handling on Microsoft Windows, Microsoft Xbox and Microsoft Xbox 360, a gaming console. Table IV lists the components (APIs) of the Microsoft DirectX system [52].

The main purpose of Direct3D is to provide a communication between the graphics application and the graphics hardware drivers in Microsoft systems. In case there is a sub feature that is not implemented in a specific hardware driver, the Direct3D emulates using a software–based generic graphics card. Though it is too slow to be used for tasks that require high performance, such as video games.

```
#include <windows.h>
#include <gl/glut.h>
#include <iostream>
using namespace std;
void disp(void);
void keyb(unsigned char key, int x, int y);
static int win;
int main(int argc, char **argv){
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(100,100);
  win = glutCreateWindow("GLUT Sample");
  glutDisplayFunc(disp);
  glutKeyboardFunc(keyb);
  glClearColor(0.0,0.0,0.0,0.0);
  glutMainLoop();
  return 0;
}
void disp(void){
glClear(GL_COLOR_BUFFER_BIT);
  glPushMatrix() ;
glPushMatrix() ;
glColor3f(1,0,0);
glutSolidCube(0.5);
glPopMatrix() ;
glPopMatrix() ;
glutSwapBuffers() ;
}
void keyb(unsigned char key, int x, int y){
  cout << "Pressed key " << key ;
  cout << endl;
  if(key == 'q'){
    cout << "Quitting " << endl;
    glutDestroyWindow(win);
    exit(0);
  }
}
```

FIG. 19: OpenGL sample code displaying a solid cube, using GLUT.

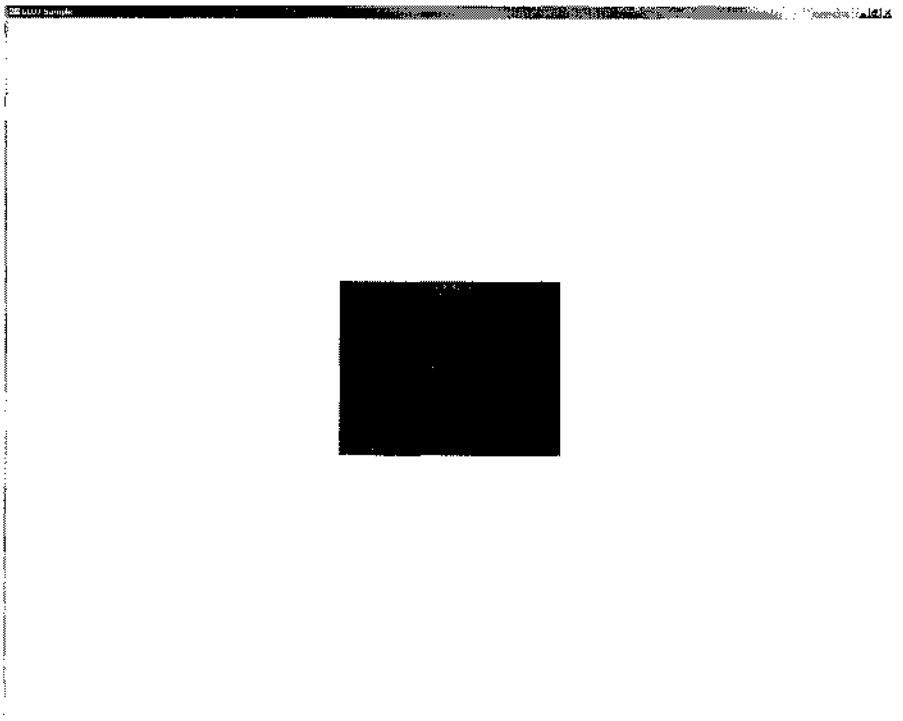


FIG. 20: OpenGL (GLUT) sample: Color Cube.

TABLE IV: DirectX Components.

Component	**Explanation**
DirectDraw	drawing 2D Graphics
Direct3D (D3D)	drawing 3D graphics
DXGI	multiple adapters and displays management
DirectSound	playback and recording of waveform sounds
DirectSound3D (DS3D)	playback of 3D sounds
DirectMusic	playback of soundtracks
DirectAnimation	2D web animation
DirectX Transform	Web interactivity
Direct3D Retained Mode	scene graph mode 3D graphics
DirectX Media Objects	streaming objects (encoders, decoders and effects)
DirectSetup	installation of DirectX components

Direct3D versus OpenGL

Direct3D, similar to OpenGL, is a immediate mode graphics API. The programmer has to specify every step in the rendering operation. Hence, developing Direct3D applications require advanced programming, again similar to OpenGL. Direct3d is a proprietary API by Microsoft for Microsoft systems, where as OpenGL is an open standard API that works on most modern operating systems (e.g., Microsoft Windows, Mac OS X, Linux, PlayStation 3 and Nintendo Wii).

II.3.9 Virtools

Virtools is a development platform has been widely used in the video game market (prototyping and rapid development), as well as for other highly interactive 3D experiences, in Web marketing, and virtual product maintenance. Virtools 3D Life Player is used to enable Web viewing experience for content that is created by Virtools. Virtools is developed and supported by Dassault Systemes Technology [53]. Figure 21 shows development of Virtual Chainsaw in Virtools environment [54].

II.3.10 TouchDesigner

TouchDesigner is a 3D animation development tool designed for realtime performance [55]. The content created can be viewed and interacted by TouchPlayer or TouchMixer. TouchDesigner is developed by Derivative Inc., based in Canada. Figure 22 shows the development of Wound Debridement Simulator in TouchDesigner environment [56, 57, 58].

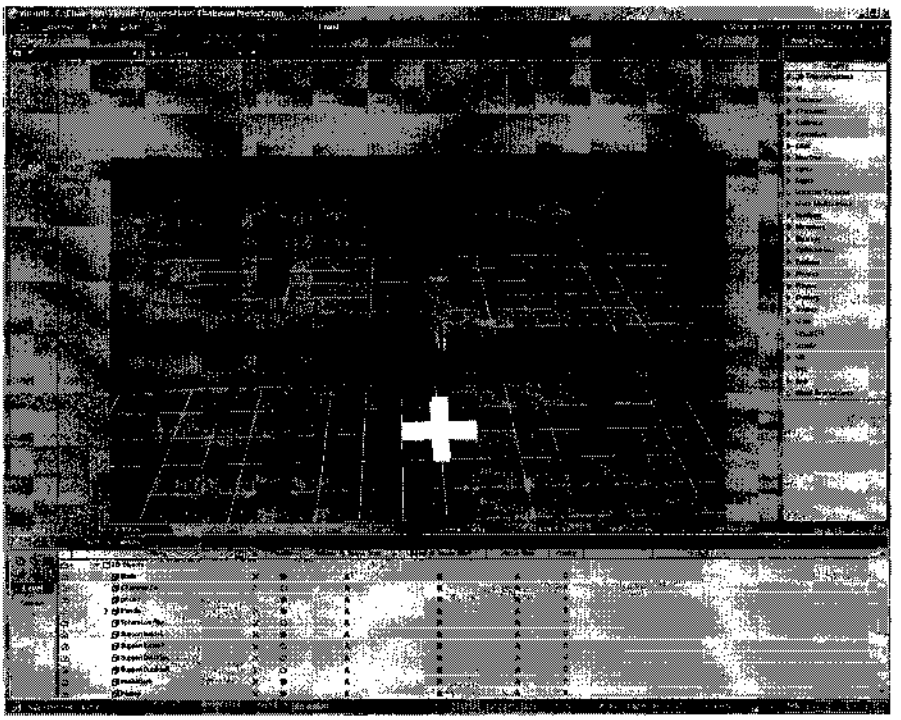


FIG. 21: Virtools development environment (Virtual Chainsaw).

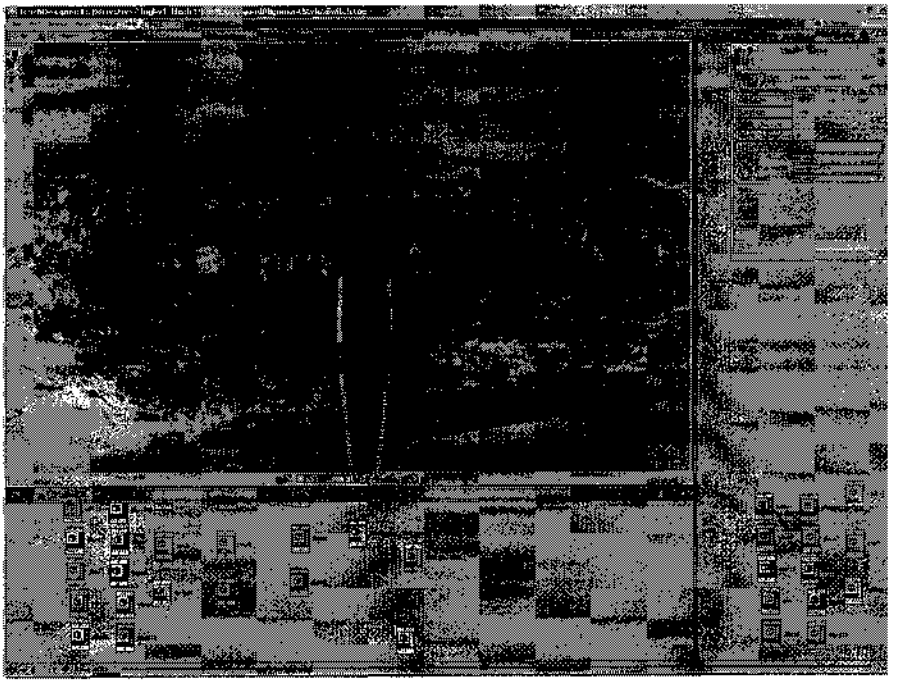


FIG. 22: TouchDesigner development environment (Wound Debridement Simulator).

II.4 PLATFORMS

Platform is often described as a set of hardware components that make up the system itself, that the software is written to run. Most commonly, operating systems are written/installed to allow other software to be run. Thus, the operating system and hardware couple become a platform themselves.

In the context of this thesis, a platform describes a hardware architecture or an underlying software that allows the technologies described in § II.3 to be run or realized. Operating systems such as the Microsoft Windows family system or Unix–like (e.g., Linux and Mac OS running on a personal computer) are the underlying software for some of the technologies described. Similarly, hardware devices, such as personal computers, portable and immersive devices, are platforms for the technologies along with their operating software or working specifications (e.g., a mobile phone with Symbian software or Head Mounted Displays (HMDs) with connection ports/cables).

II.4.1 Microsoft Windows PC

Microsoft Windows is a series of operating systems from Microsoft Corp. [59]. Although the most recent version of Windows is Windows Vista, Windows XP (32bit) is considered the standard Windows version for the Windows family of operating systems for the context. It is also assumed that technology support in the future versions of Microsoft Windows is backward compatible with Windows XP. Out of the technologies, only DirectX is embedded with the initial installation of the operations

system. Other technologies either have tools, APIs, plug–ins, components to install or have to use drivers that are loaded by the Windows operating system (e.g, OpenGL).

II.4.2 Unix–like PC(Linux & Mac OS)

Unix–like operating system is defined as an operating system that behaves in a similar manner to a UNIX system, which was developed originally in 1969 at Bell Labs [60, 61]. Linux is a name given to a Unix–like operating system that uses Linux (open source) kernel, which is the central component of the operating system. On the other hand, Mac OS is another Unix–like operating system that is developed by Apple Inc. Even though there are similarities among Unix–like platforms, from the perspective of VE development, there are some tweaks that have to be made when switching operating systems. Thus, each operating system has to be seen as a separate platform for development of VEs.

II.4.3 Portable Devices

There are three types of portable devices that are used for developing and realizing VEs; portable computers (i.e., Laptops, Palmtops, Personal Digital Assistants PDAs), portable communication devices (i.e., Mobile Phones and PDAs) and hand–held gaming consoles (e.g., Playstation Portable [62] and Nintendo DS [63]). Most of these devices require a different approach than personal computers, when developing or rendering VEs with them. Even though laptops are personal computers in basis, they have much less computational power and less storage space along with their lower capability graphics accelerators. Thus, laptop computers can be seen as a different

platform in some cases. For other devices, mobile phones, PDAs etc.,the difference is not only the computation/rendering power but also their operating system, modified support (clipped versions of languages and APIs) for different technologies, and low level of detail requirement (e.g., higher resolution rendering is meaningless for small displays).

II.4.4 Immersive Devices

There are several different immersive devices out in the market, such as CAVE [64], ImmersaDesk [65], VisionDome, [66] and Head Mounted Display (HMD) [67]. These devices are non–traditional when compared to regular displays (i.e., LCD, CRT, projection screens), since they require modification of rendering algorithms. Most of the time, special drivers or APIs are required in order to engage such devices. Hence, immersive devices along with their required software should be considered platforms.

II.5 GRAPHICS PROCESSING (RENDERING)

There are different varieties of capabilities available for VE developer from different technologies and platforms. The developer has to match the requirements of the VE to the capabilities of the technology and the platform. For example, a developer seeking high fidelity graphics rendering with soft body deformations should not select VRML as development technology, since it is trivially hard to define dynamic object behavior in VRML as well as relatively poor rendering quality than other technologies.

Modern graphics hardware support point transformations and lighting calculations without use of system resources [68]. Graphics processing units (GPUs) are improved to include support for special operations specific to graphics rendering, such as fast division, fast inverse square roots, transforms and dot product operations. Different virtual systems have different models for representing the 3D virtual world space. In some hardware renderers, the 3D VE has two meta spaces, the world space and the view space dictated by a camera model. A renderer first transforms 3D data from the world space to the view space. Next, the 3D data is projected onto a 2D plane that represents the viewing screen in a desired fashion (i.e., scanline rendering, rasterization, ray casting, radiosity and ray tracing [69]). Since the entire process is highly computationally intensive, at each step, optimizations are necessary to meet the requirements of real–time rendering. The renderer eliminates portions of data that are not visible to the viewer by two processes known as culling and clipping. Culling is the process where the graphics processor determines whether an object is completely outside the current view. By not processing objects that are outside the view volume, processor cycles are saved. Clipping is the process of splitting an object into smaller pieces and determining the visible from the invisible pieces [68]. Thus after culling and clipping, the renderer discards the invisible pieces and starts processing only on the visible pieces. Then, transformed 2D data is drawn to the to the screen. This process is called rasterization. Rasterization is the process of taking a geometric object in screen space and converting it into a raster image (pixels or dots) for output to be drawn. The majority of time in the rendering process is spent on rasterization. Rendering may be done in the general purpose CPU, in part by a

hardware–accelerated graphics card or totally on special purpose graphics hardware. A proper choice of the view space can reduce the rendering load on the graphics processor.

Application programming interfaces (APIs) for graphics accelerators such as Direct3D, OpenGL or Glide [70] provide an interface between the scene graph management and the actual rendering system or a direct interface between developer and the rendering system. Applications written using APIs such as Direct3D or OpenGL are portable across multiple hardware cards that support these standards. Writing applications using Glide which is a low level rasterizing API for 3dfx cards makes the application non–portable to other cards. But if the intended target platform is known to use a 3dfx card, then the choice of writing to the glide API may be a good design choice.

Besides the rendering done in lower level (e.g, rasterization) for the VE developer, there are two modes of rendering in the technologies explained, immediate mode and retained mode.

II.5.1 Immediate Mode Rendering

Immediate mode rendering is a style for application programming interfaces of graphics libraries, in which client calls directly cause rendering of graphics objects to the display. Every frame, the developer has to specify the steps of rendering and application redraws everything regardless of actual changes. This method provides the maximum amount of control and flexibility to the application program. OpenGL and Direct3D developer has access to immediate mode rendering.

II.5.2 Retained Mode Rendering

In contrast to immediate mode, in retained mode the technology retains a complete model of the objects to be rendered in the VE. The developer controls the environment by changing the internal list of the objects. The library (API) (e.g., VRML, X3D and Java3D) controls and applies the required rendering so that the objects in the retained list are drawn. This retained list is called scene graph.

Scene Graph

The scene graph is a data structure that represents grouped objects and their hierarchical organization [68]. It is used to store, organize and render 3D scene information such as geometries, lights, materials, and other parts of a scene. The simplest form of scene graph uses an array or linked list data structure, and displaying its shapes is simply a matter of linearly iterating the nodes one by one. Other common operations, such as checking to see which shape intersects the mouse pointer in a GUI-based application are also done by use of linear searches. Larger scene graphs cause linear operations to become noticeably slow, and thus more complex underlying data structures are used, the most popular being a tree. This is the most common form of scene graph. In these scene graphs hierarchical representation of group–nodes and leaf–nodes is created. *Group Nodes* can have any number of child nodes attached to it. Group nodes include transformations and switch nodes. *Leaf Nodes* are nodes that are actually rendered objects or effects in the scene. These include objects, sprites, sounds, lights, and anything that could be rendered. The scene graph hierarchy is logical and often spatial representation of a graphical scene.

Every object in a 3D environment needs to be to be tested for culling and clipping to identify whether that object is outside the current view. If a 3D world has a large number of objects, it is computationally inefficient to test every object for culling and clipping. The need to test every single object in the scene can be eliminated by organizing the objects into groups of objects according to their spatial location. Objects which are within the current view of interest will occur within the same spatial region. By transversing the tree of the scene graph hierarchically, large portions of the virtual world can be eliminated from processing by the renderer. This hierarchical organization of grouped objects also provides the information on how the objects in the real world are linked and operated upon.

VE systems may be broadly classified into active and passive systems. Passive VR systems present a read–only view to the viewer. The user can navigate in the scene, and the navigation changes characteristics of the view, such as perceived distance and angle of view, but the user does not change the structure of the scene structure. Active systems, in contrast, can dynamically create scenes based on user input and user interaction and thus present a higher level of interaction capabilities to the user. Using Script nodes, VEs built on VRML 2.0 can be designed to have capabilities to insert, update, and delete objects as a result of user interaction with the scene. In an active scene, as the state of the scene is modified, the scene graph tree must also be updated to represent the new state. When a VE is changed as the result of an external agent, which could be the result of a user interaction or the result of a simulation feed update, only the affected sub–trees of the scene graph need to be updated. Different systems have different implementations to update scene graphs.

Based on the structure of a specific scene graph, the result of an interaction may need to update leaf nodes at different locations of the scene graph. The order of updating the scene graph can improve the efficiency of the update operation.

CHAPTER III

THEORY

In this chapter, the rapid prototyping methodology for VEs and the Common Scene Definition Framework (CSDF) [71] [30] will be described. The capability of rapid prototyping for VEs is provided by the proposed framework, CSDF, and by the surrounding system.

III.1 RAPID PROTOTYPING

The VE designer has a general conceptual understanding that transcends his/her favorite technology. The developers are usually familiar with a specific technology within which they develop, but are required to learn a new one in which they lack the expertise. Similarly, the VE developers must learn the platform on which the content will be rendered. Therefore, having an automatic synthesis capability to target output on a desired platform would reduce the time required to learn different outputs and platforms either providing additional time to be devoted towards developing content or speeding up delivery times.

Moreover, the automatic synthesis can also be used to assess suitability of a particular platform for the desired VE. A prototype can be realized quickly via the synthesis, so that the developer may determine whether further investment is warranted or not.

More than usual, when a new platform is chosen by the VE developer to replicate a previously implemented VE in the old platform, neither the old VE nor its components

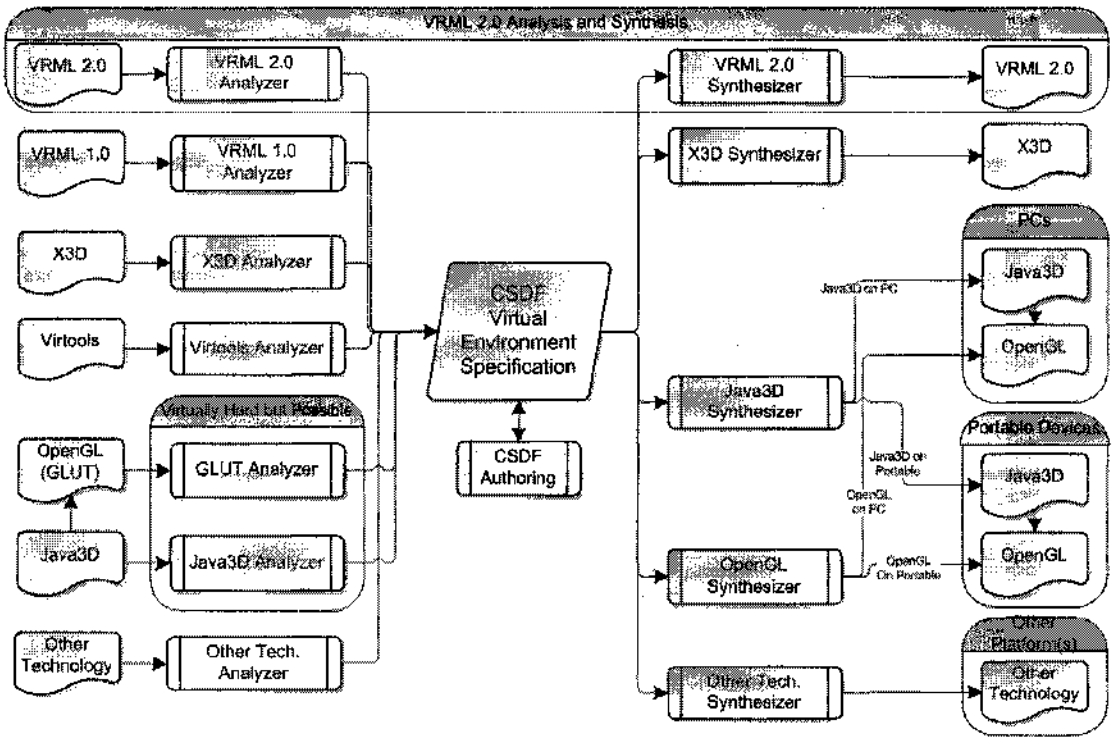


FIG. 23: CSDF Concept Diagram.

are reused. Hence, virtually, the development starts from scratch. The automatic synthesis allows reuse of existing content as an intermediate prototype for the targeted platform.

Figure 23 shows the conceptual diagram of the CSDF. A virtual world defined in a VR application, such as VRML, can be imported to CSDF, where the world would be translated according to CSDF specifications. From this representation in CSDF, it is possible to synthesize to any other technology on a specific platform. Note that for the sake of diagram simplicity, only a few technologies and platforms are shown in Figure 23. More detailed explanations are given in Chapter IV.

III.2 CLASSIFICATION OF VE CAPABILITIES

It is important to classify the capabilities of VE technologies, since the classification reveals what type of approach should be taken to handle and represent VEs. It also reveals what is possible and establishes the limits of what is trivial, and what is non trivial. When the representation difficulty classifies the capabilities, they might be separated into three sets. The following subsections visit these sets.

III.2.1 Geometry/Appearance

Geometries and their corresponding appearance attributes can be statically described. Neither dynamic behavior is acted upon them nor is dynamic behavior represented. This set of capabilities is easiest to handle. This set can be represented in a fairly straight forward way and usually can be broken into simpler capabilities of the same class. For example, a complex but static object can be represented by an indexed face set. Basic appearances can include attributes such as diffuse/emissive colors/shininess/alpha channel.

III.2.2 Behaviors

Behaviors are the dynamics that are defined (pre–made/standard) by the VE technology. These include basic sensors (e.g., mouse, keyboard, location etc.) and other scene elements like lighting/shading. Even though these behaviors affect the geometry/appearance in VEs, they are representable fairly universally across technologies. Therefore, for many behaviors, there are mappings providing equality of these behaviors.

III.2.3 Scripting/Custom Coding

The most complex capabilities are custom made by the VE developer via use of either scripting or coding in a computer language. These are, more often than not, quite difficult to represent in a format other than in their original format. In most cases, there are script or code pieces embedded in VE specification. In other cases, some technologies represent the VEs in pure scripting or custom coding (e.g., OpenGL). In OpenGL, very low level capabilities can not be easily recognized and represented in terms of higher level capabilities. These would require sophisticated parsing/semantics representations. Thus, OpenGL analysis is very challenging, and currently not supported by CSDF.

III.3 CSDF

In the methodology, the VE is described by the Common Scene Definition Framework (CSDF), which includes a superset of existing technologies. The superset consists of all capabilities of existing technologies, so that any VE defined in any technology can be represented in CSDF. This superset representation capability is necessary for analysis and synthesis of VE technologies.

The VE developer is capable of rapidly synthesizing the desired output at the target platform and technology. If a target technology lacks a feature in a source content and the target technology supports the implementation of the feature, CSDF will support filling in this gap. An alternate approach for the framework is to downgrade a complex VE to an environment that is a subset of the initial input, so that the

prototype of the environment can be realized on the target. For example, a Virtools scene can be analyzed into CSDF and then synthesized to a basic Java3D environment displayed on a mobile device. In synthesis, a number of capabilities and level of detail, which may not be supported by target platform and the technology, has to be omitted. Furthermore, knowledge of the lack of a capability can be of use to the developer in assessing the suitability of a platform for a particular application.

III.3.1 Framework

The fundamental representation for the CSDF is an extension of the X3D specification (§II.3.2). The reasons for this decision are as follows:

- X3D is sponsored by and actively supported 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 [36].
- X3D includes a rich set of primitives for modeling 3D geometry and behaviors. X3D can be naturally extended create a superset of capabilities, when compared to other candidate technologies.
- X3D is an extensible open file format standard and hence is suitable for supporting an evolving CSDF.
- Web services, the programmatic interfaces for application to application communication on the Internet use XML technologies to construct messages that can be exchanged over a variety of underlying protocols. This helps to make integration with Web services easier [71] [36].

As a result, CSDF has a scene graph structure similar to X3D. In CSDF, *CSDFNode* is the node class for the tree structure that was used to construct the scene graph. All the other entities within CSDF scene graph are extended from *CSDFNode*. All CSDF classes support Java Serialization via inheritance through *CSDFNode*. Java Serialization is defined as follows [72]:

> Object serialization is the process of saving an object's state as a sequence of bytes, as well as the process of reconstituting those bytes into a live object at some future time. The Java Serialization API provides a standard mechanism for developers to handle object serialization.

Java Serialization reduces the time necessary to develop code for saving and restoring object or application state. Moreover, Java Serialization makes it easier to send objects over a network connection. The CSDF can use Java Serialization to save the VE, represented within, to a database or to a file. The saved information can later be recalled and used.

Even though, theoretically, it is possible to synthesize/analyze to/from all the technologies described in §II.3, practical issues constrain some of such mappings to/from CSDF. The synthesis process may involve representation of a higher level capability in terms of lower level capabilities. For example, a primitive 3D model might be represented by vertices and face sets. On the other hand, the analysis process may be more complex in some cases, since setting low level capabilities up to a high level capability can be trivial. For example, making a primitive 3D object from

a collection of vertices and face sets is a very complex process. The CSDF will be discussed in more detail in Chapter IV.

III.3.2 Modules

The modularity of the CSDF is technology and platform independent. Thus, the new technologies, or changes to existing technologies, and platforms can be applied without changing the underlying architecture of the CSDF. In order to facilitate such additions, module interfaces are created (see §IV). Since all modules implement the common interfaces of CSDF, changing or adding a module will not affect the connection of this module to CSDF.

It is also possible to create custom 3D model template libraries within CSDF, which can be used by developers in their VEs. These libraries can be either stored within the CSDF repository or in one of the possible technology formats in which the analysis module exists.

III.3.3 Analysis (Parsing) from Technologies

VE developers use a variety of technologies and tools. A number of these technologies have been described in §II.3. The analysis of the VE consists of representing the VE in the CSDF by parsing/analyzing the respective formatted inputs of the technologies. The rapid prototyping system has the ability to import existing models (VEs) from popular technologies into the common scene definition framework. The CSDF also allows easy addition of import modules for other VEs because of its modular architecture and easy plugability of the analysis modules.

III.3.4 Synthesis to Technologies/Platforms

Different technologies and platforms have different requirements. This is why VE synthesis is necessary to transform from the requirements specification, defined in CSDF, to a technology/platform architecture, which has its own requirements specification acting as a confining factor. Along with the representation of the VE, CSDF also knows the capabilities and features of the target technology and platform. This knowledge helps the synthesis process, by mapping from the CSDF representation to the target technologies and platforms, to construct the customized output. In other words, the CSDF synthesis process may create a representation of the VE that is the best possible to replicate of the initial VE. In addition, CSDF can provide feedback on the capabilities that are not supported by either the target technology or the platform. For example, unmatching capabilities often occur when the platforms and technologies have various interaction and camera control methods.

If the representation in CSDF can not be directly synthesized, in case of a non–supported capability, the synthesis process handles the mapping in one of two ways. First, if the capability in question can be represented in terms of combinations of other capabilities supported by the target technology or platform, the mapping creates low level entities representing the high level entity in CSDF. For example, an entity in CSDF, like a 3D object primitive, might have to be represented by a series of surfaces or simpler primitives in OpenGL. Second, if the entity can not be supported by the target platform or technology, it is omitted and a warning is generated. An example for this situation might be the picking by mouse interaction capability in VRML,

which has to be omitted when synthesizing for VRML on a mobile device without advanced interaction capability (e.g., touchscreen).

Furthermore, the proposed framework might approximate the structure of some capabilities in the synthesized VE. This structural approximation can represent a higher level of detail in terms of more lower levels of detail, so that limitations on the target platform can be overcome. For example, the prototyping system may approximate movie textures by representing the texture with a simple image. Hence, the VE can be realized on a slower mobile device. This approximation can be applied in other ways as well, such as using low fidelity primitives instead of high level ones defined in CSDF. In some cases, an algorithm can be applied to simplify the 3D models by lowering the number of vertices or voxels used. Thus, the approximation optimizes the output for the target platform. Further optimizations can also be applied, depending on the needs of the target platform (e.g. removing layers of the representation or optimizing human to VE interfaces).

One other consideration for the synthesis process is the hardware requirements specification for platforms. Specific sets of hardware architectures require optimization. These hardware platforms may be single processor, multi–processor, or cluster type on a desktop computer, a laptop, or a mobile device. The framework has to synthesize an output according to the needs of the target hardware platform, and make necessary optimizations.

III.3.5 Authoring

The synthesized output by CSDF might also be used as a starting point for authoring the given VE on the target VE technology. The authoring can be either at the input/imported technology or at the target/synthesized technology and hardware platform. Thus, CSDF leverages the knowledge of the VE developer from any 3D technology to a new target 3D technology and platform, in contrast to VRJuggler [73] and Diverse [74] systems. The proposed framework can also warn the user about limitations of capabilities in the target 3D technology or the platform. This gives the VE user a quick idea about the selected synthesis technology/platform, and also lets the user know if the desired tuple of technology and platform is suitable for her needs. In addition, CSDF allows modification of the VE within the framework. Therefore, a VE developer can quickly analyze/import from a technology in which she is proficient, then author extra capabilities that are non existent in the imported technology.

CHAPTER IV

REFERENCE IMPLEMENTATION

The rapid prototyping methodology that is used in the development of VEs was given in Chapter III. In this chapter, the implementation of the rapid prototyping methodology to develop VEs and Common Scene Definition Framework are explained.

IV.1 SOFTWARE FOR IMPLEMENTATION AND TESTING

The rationale behind the choice of preferred development platform for the framework and other used tools that form part of the rapid prototyping system is provided in the following subsections.

IV.1.1 Java

Java [75] was chosen as the language to implement the framework, parsers, and other modules. Java is a strongly typed object oriented language originally developed by Sun Microsystems and released in 1995. The language has a very similar syntax to C and C++ but has a simpler object model and fewer low-level features [76]. When Java source code is compiled, the compiler creates a bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. Hence, Java is platform independent. This gives the freedom to switch platforms to the developer. Java Virtual Machine (JVM) ensures that programs cannot circumvent strict system restrictions. Also, Java has strong type checking enforced by the compiler

that minimizes data corruption due to precision conversions. Table V shows some of the differences between C++ and Java. The advantages of developing on the Java platform leads to shorter project implementation and debugging times.

Java also has a powerful Reflection API [79] that is used throughout the CSDF core and its modules. Reflection is the process in which a program can observe and modify its own structure and behavior. Java reflection API can do the following with/on any Java object [80, 81]:

- Determine the class of the object.
- Get information about the class's modifiers, fields, methods, constructors, and super–classes from the Java object.
- Determine the constants and method declarations that belong to an interface.
- Create an instance of a class whose name is not known until runtime.
- Get and set the value of an object's field, even if the field name is unknown to your program at runtime.
- Invoke a method on an object, even if the method is not known at runtime.
- Create a new array, whose size and component types are not known at runtime, and then modify the array's components.

The development language selected for the framework and other modules does not have any affect on synthesized VEs. It is only responsible for making necessary transformations via analysis, storage, and synthesis. These transformations could be

TABLE V: Differences between C++ and Java [76, 77, 78].

C++	**Java**
Some backward compatibility with C source code.	Not source-compatible with other languages.
Allows direct calls to native system libraries.	Called through the Java Native Interface (JNI).
Exposes low-level system facilities.	Runs in a protected virtual machine.
Optional automated bounds (Arrays etc.) checking.	Always performs bounds checking.
Explicit memory management, garbage collection optional via library.	Automatic garbage collection only. Automatically manages memory and instantiated objects by de allocating objects no longer referenced. This eliminates the need to explicitly free dynamic memory from the heap.
Allows explicitly overriding types.	Rigid type safety except for widening conversions.
C++ Standard Library has a more limited scope but includes: Language support, Diagnostics, General Utilities, Strings, Locales, Containers, Algorithms, Iterators, Numerics, Input/Output and Standard C Library. Platform-specific libraries differ for threads, Network I/O and GUI Programming and often require third-party libraries.	Extensive libraries including support for containers, locales, algorithms, iterators, GUI programming, graphics, multi-threading, networking and security. (easier to import than C++ due to highly portable jar library files)
Operator overloading.	Meaning of operators is immutable.
Full, multiple inheritance	Full single inheritance, multiple inheritance from interfaces only

done by other development languages (i.e., C++), and there would be no change in the synthesized VE.

As an integrated development environment (IDE), an opensource software, Eclipse [82] is selected and used. Eclipse provides a wide variety of extensible application frameworks, tools and runtimes for software development and management.

IV.1.2 CSDF and Proprietary Platforms

Analysis of a Virtools native file is not possible since the file format is proprietary and not available to us. In order to provide analysis capability to Virtools, a plug in for Virtools development environment is implemented. For this implementation, the Virtools software development kit is used. The API for the Virtools SDK can be accessed using Microsoft Visual Studio C++. The plug in creates an intermediate file that can be parsed by a CSDF analyzer created specifically for the intermediate file. This intermediate file contains scene information including objects and other elements of the scene in Virtools file. Similar to Virtools, TouchDesigner file format is proprietary. In this case, a TouchDesigner internal script is implemented to synthesize the scene by reading an intermediate file that is created by CSDF TouchDesigner synthesis module.

IV.2 MODULES

In this section, implementation details of CSDF core, analysis modules, synthesis modules and authoring process are briefly described.

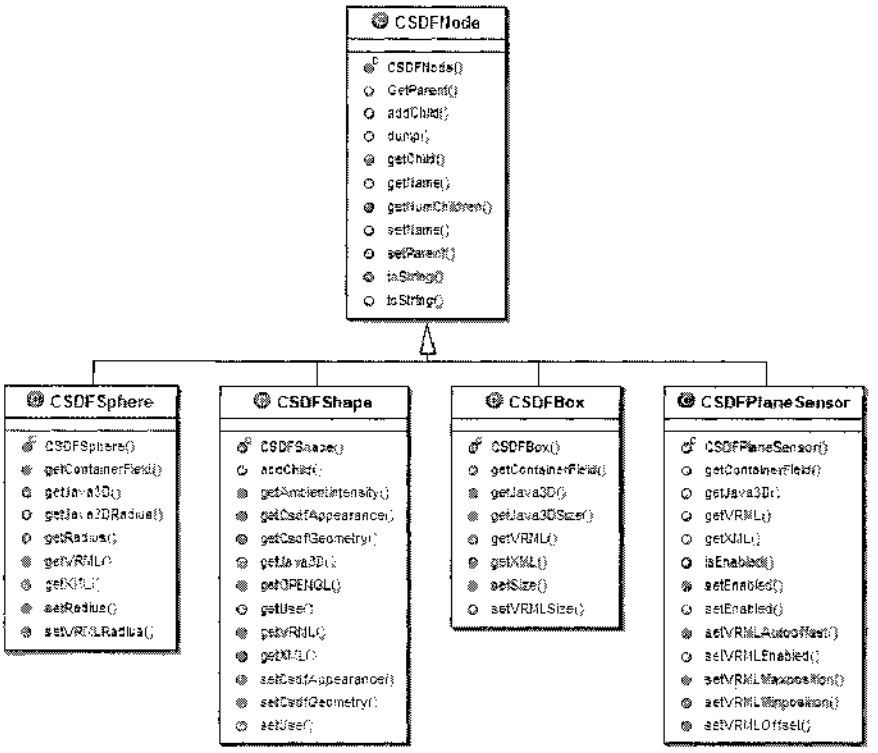


FIG. 24: CSDF Classes [71].

IV.2.1 CSDF Core

Figure 24 shows several entities from CSDF. Note that *CSDFNode* is the node class for the tree structure that constructs the scene graph. All the other entities within CSDF scene graph are extended from *CSDFNode*. As mentioned in §III.3.1, all the classes implement a Serializable interface so that VE residing in CSDF can be stored for later use or transmitted over networks.

CSDFShape node has two member fields; a Geometry field and an Appearance field. Geometry classes such as *CSDFBox*, *CSDFCylinder*, *CSDFSphere* implement the *CSDFGeometry* interface. Figure 25 shows the class diagram of the *CSDFGeometry* interface along with its implementors *CSDFSphere*, *CSDFBox* and *CSDFPointSet*.

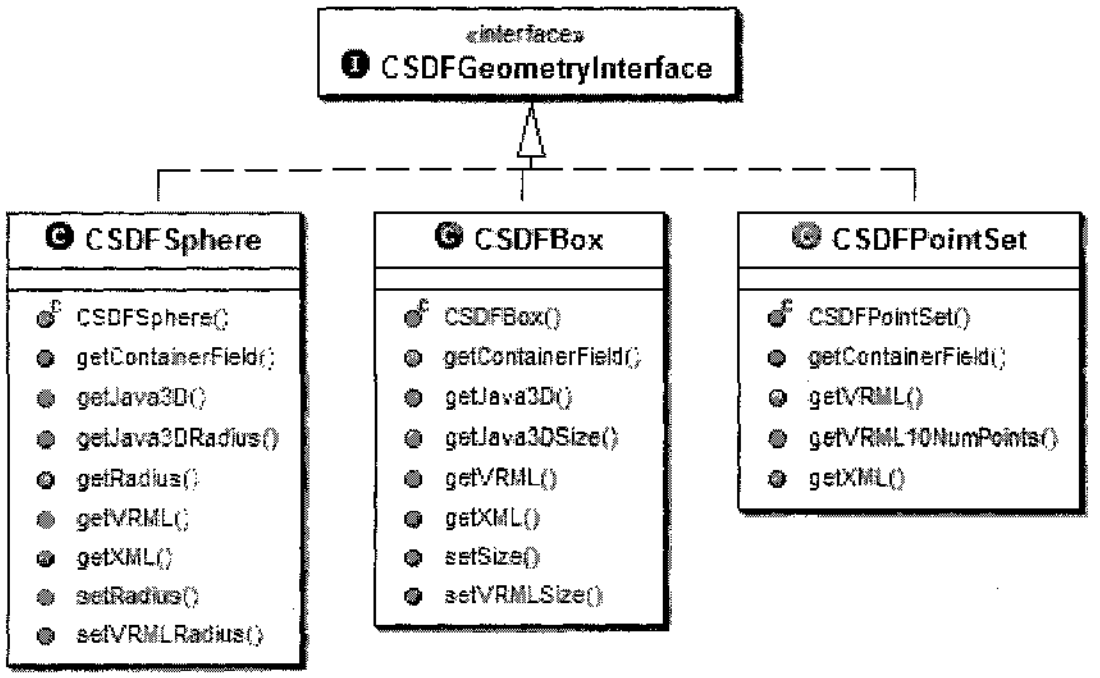


FIG. 25: *CSDFGeometry* interface and a few classes that implement the interface [71].

Similarly, CSDF nodes for ImageTexture, MovieTexture and PixelTexture implement *CSDFTexture* interface, where their shared functions are defined. Additionally, in order to distribute synthesis capability, all classes that are extended from CSDFNode implement *CSDFSynthesis* interface (Figure 26). The synthesis functions are defined in *CSDFSynthesis* interface. The output format transformations for the respective synthesis target platforms are implemented within each CSDF node.

IV.2.2 Analysis

In this subsection, analysis and parsing methodologies for several technologies are described. Figure 27 depicts the modules that analyze parse input technologies.

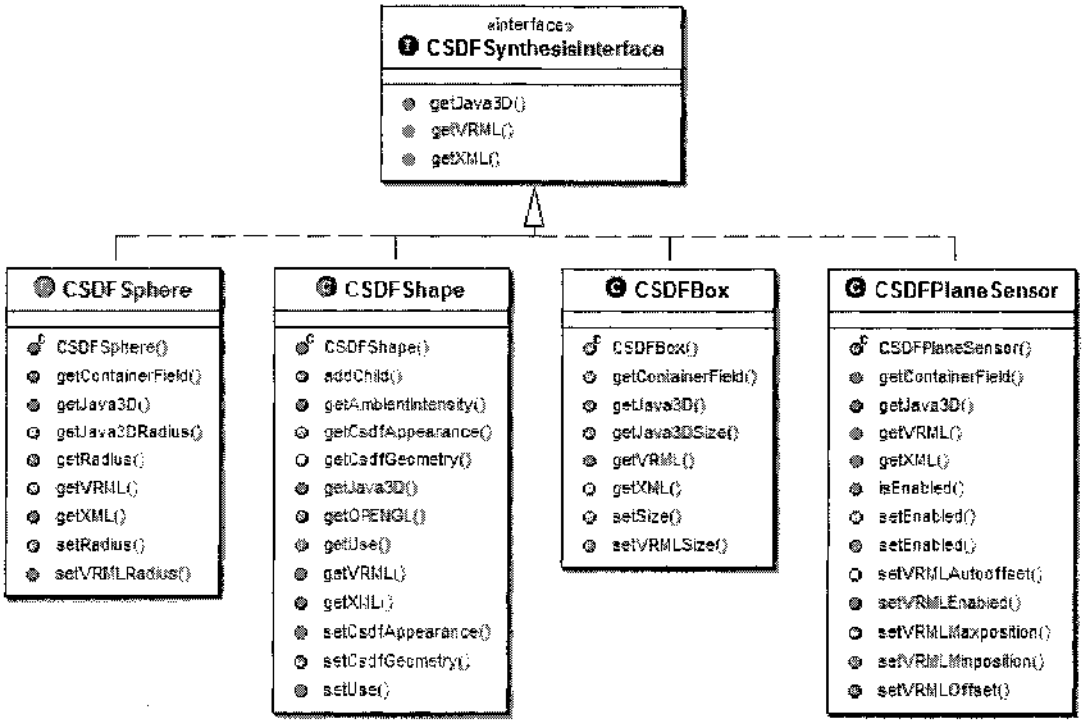


FIG. 26: *CSDFSynthesis* interface and a few classes that implement the interface [71].

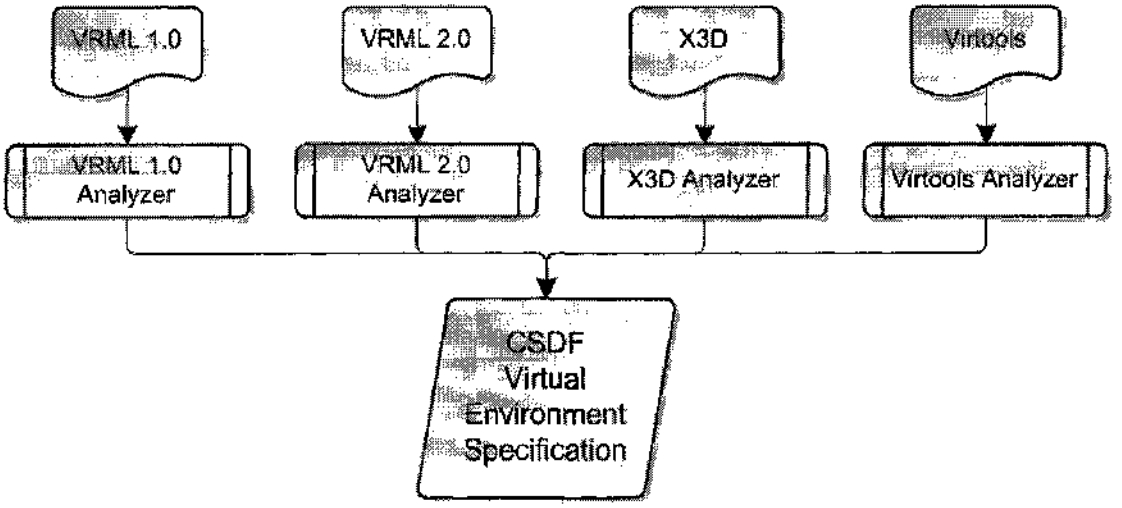


FIG. 27: Analysis components of CSDF.

FIG. 28: Phases of VRML Parser.

VRML

An important design requirement in implementing the analyzer (parser) is to ensure that adding capabilities to the common framework does not result in the redesign of the VRML parser (i.e., the parser is sufficiently scalable with respect to the addition of capabilities to the common framework). This requirement emphasizes the modularity required by the architecture of the proposed rapid prototyping solution.

The different steps in the parsing of a VRML input file are shown in Figure 28. The VRML parser has been implemented using JavaCC [83], a Java compiler–compiler. JavaCC is an open source parser generator comparable to "famous," yet another compiler–compiler (YACC) in Unix platform. JavaCC uses the Extended BackusNaur Form (EBNF), which is a syntax used to describe context–free grammars.

The function of the lexical analyzer is to convert the input VRML file into a sequence of tokens. These tokens consist of elements such as numbers, identifiers, begin-end blocks, statements, and program units. The tokens are matched by regular expressions used in the definition of the language [84]. For example, the definition made (in JavaCC for VRML) for regular expression to match a set of characters to be of numeric token type is `<NUMBER_LITERAL: ("-")? (".")? ["0"-"9"] (["0"-"9", ".","+","-"])*>`.

The expression defines a valid numeric token to have an optional '-' character followed by an optional '.' character followed by a single numeric digit followed by optional zero or more occurrences of a valid numeric character. Similarly, it is possible to define tokens for symbols and keywords (e.g., <LBRACE: '{'>, <RBRACE: '}'> and <EVENTIN: "eventIn"> for "{", "}" and "eventIn" respectively).

In syntactic analysis, sequence of tokens produced by lexical analyzer is analyzed to determine grammatical structure with respect to a given context free grammar. Figure 29 shows a portion of the context–free grammar (CFG) specification for VRML. In CFG of VRML (Figure 29), a *vrmlScene* consists of a group of *statements*, which may be composed of a single *statement*, a single *statement* followed by a group of *statements* or by no *statement* (empty). Similarly, a *statement* may be composed of a *nodeStatement* or a *protoStatement* or a *routeStatement*. A *nodeStatement* may be composed of a *node*, *node* definition (*DEF*), or a node use (*USE*).

For each grammar production, there exists a clause of a recursive function. The syntactic correctness of the input VRML file is enforced by the grammar definition on which the compiler is synthesized using JavaCC. When an input scene containing an incorrect construct is input to the parser, the construct will match any of the valid production rules specified by the grammar and as a result of this condition, a parse exception is thrown by the parser.

Semantic actions are actions performed when the parser matches a grammar production. In the parser, the semantic actions are the fragments of Java code that are attached to each grammar production. Semantic actions are distributed along with the control flow of the parsing. An example of a semantic action is the code

```
vrmlScene ::=
    statements ;
statements ::=
    statement |
    statement statements |
    empty ;
statement ::=
    nodeStatement |
    protoStatement |
    routeStatement ;
nodeStatement ::=
    node |
    DEF nodeNameId node |
    USE nodeNameId ;
```

FIG. 29: A small portion of the context–free grammar (CFG) for VRML language.

shown in Figure 30. The Java code between the braces executes every time the parser encounters a *USE* tag followed by a node definition.

In the semantic analysis phase, the compiler connects variable declarations to their use, checks each expression type for semantic correctness, and translates abstract syntax into a simpler representation. Compilers maintain symbol tables to match identifiers with their types and other important attributes. In the VRML language, variable identifiers are specified by the *DEF* keyword and variable use is specified by the *USE* keyword. The root node of the scene in CSDF has two symbol tables, one table to hold *DEF/USE* entries and the second table to hold Prototype references.

Event routing (behavior) between different entities in VRML is implemented via the *ROUTE* statements. Events can be routed for only those nodes that have defined names. Thus, references to nodes that have distinct and definitive names are kept in the symbol table. When the parser encounters a definition for a named node (declared

```
<USE> def_name = NodeNameId()
{
  System.err.println("Class Name = CSDF.CSDFUse");
  Class t = Class.forName("CSDF.CSDFUse");
  temp = (CSDF.CSDFNode)t.newInstance();
  if(CSDFScene.symbolTable.get(def_name) != null){
    System.err.println("DEF Reference for "+def_name+" found");
   ((CSDF.CSDFUse)temp).setDefname(def_name);
  }else{
    throw new ParseException("DEF Reference for "+ def_name
                          +" not found in the Symbol table");
  }
  parent.addChild(temp);
}
```

FIG. 30: Example of Semantic action in the VRML parser.

using *DEF*), a lookup is performed for the node name in the symbol table. Depending on the semantic rules of the target platform, if the lookup returns an existing node, in VRML, the new definition supersedes the previous definition, whereas in X3D, a duplicate node definition is implied and the parser throws a *ParseException.* If the lookup returns an empty or null reference, then a mapping between the node name and reference is added. When the parser encounters an instantiation (*USE* clause), a lookup is performed on the *SymbolTable* interface. If the lookup fails (i.e., an empty/null reference is returned), a *ParseException* is raised, indicating that an object is being used before declaration.

The parser must accept only valid node types and reject invalid node types while parsing a VRML file. This is performed by parser via type checking. For instance, the VRML parser must accept valid nodes such as *Box* or *Cylinder* while rejecting node declarations such as a *Cube* that is an invalid node with respect to VRML.

```
node_name = Id()
{
  PRINT("Class Name = CSDF.CSDF"+node_name+" Def name+defName);
  Class t = null;
  try{
    t = Class.forName("CSDF.CSDF"+node_name);
    node = (CSDF.CSDFNode)t.newInstance();
  }catch(ClassNotFoundException cnfe){
    THROW PARSE EXCEPTION ("ILLEGAL NODE FOUND")
  }
  if(defName != null) {
    //UPDATE SYMBOL TABLE - Hidden for simplicity
  }
  //ADD NODE TO TREE - Hidden for simplicity
}
<LBRACE>
(NodeBody(temp))?
<RBRACE>
```

FIG. 31: Semantic action for node name handling and type checking (VRML Parsing).

Figure 31 shows the semantic action code that executes when the parser encounters a node name. To achieve type checking abilities mentioned, framework classes in the common framework (CSDF) were constructed to have names mapped to nodes in X3D and VRML. By using the reflection feature of Java (see §IV.1.1), the parser creates the corresponding framework entity name at runtime and instantiates the object from existing CSDF class (Figure 31). For example, if the parser encounters a node name Box, the parser computes the framework class name as `CSDF.CSDFBox` and instantiates the class at runtime. Thus, the generic mechanism to instantiate the respective framework class keeps the implementation of the VRML parser independent of the addition of new capabilities to the common framework. The use of the Reflection API for dynamic runtime instantiation also performs the function of type checking

for node types. If the parser tries to instantiate a class that is not present in the framework, the Java runtime would raise a *ClassNotFoundException* and as a result of this condition, a *ParseException* is raised.

Similar to node type checking, type checking for valid node attributes is performed by the parser. Only valid attributes corresponding to a specific node should be accepted. For example, a valid attribute such as *radius* corresponding to a *Cylinder* node should be accepted and attributes such as *size* or *length* that are not valid attributes for a VRML *Cylinder* node should be rejected. Figure 32 shows the semantic action code that executes when the parser encounters a field name (attribute name) followed by a field value (attribute value). Again, the Java Reflection API is used to employ type checking, similar to node type checking. While parsing, if the parser encounters an attribute such as *size*, the parser first computes the method to be invoked as *setVRMLSize()* by appending the attribute to the string *setVRML*. A method object is created by the parser using the string *setVRMLSize*. This invokes the method (*setVRMLSize*) on the method object (*CSDFBox*), which gets executed with the field value ("0.5") as the parameter. Thus, for any attribute *Xyz*, the parser tries to invoke the respective *setVRMLXyz()* method.

The traditional way of handling type checking (i.e., checking for each specific node type or node attribute type) would make the parser code less manageable because every time a new attribute is added to a node the code of the parser has to be modified.

A CSDF class that is associated with a VRML node may have additional attributes that are not valid VRML attributes, since these additional attributes may

```
field_value = FieldValue(null,node)
{
  System.err.println("Field name "+field_name
                     +" Field value "+field_value);
  try {
    if(field_value != null){
      Class[] paramClassList = {Class.forName("java.lang.String")};
      Field_name = FormatUtils.capitalize(field_name);
      System.err.println("Method call = setVRML"+field_name);
      Method method = (node.getClass()).getMethod("setVRML"
                       +field_name,paramClassList);
      Object[] paramList = {(String)field_value};
      method.invoke(node,paramList);
    }
  }catch(Exception e) {
    //THROW PARSE EXCEPTION ("ILLEGAL ATTRIBUTE FOUND")
  }
}
```

FIG. 32: Semantic action for node attribute handling and type checking (VRML Parsing).

be associated with another technology such as X3D. A *ParseException* on encountering these attributes must be triggered when VRML parser encounters them. For every technology that can be analyzed by CSDF, each framework class must have a set of setter methods (i.e., functions that set value of fields of classes) specifying which valid attributes correspond to the input technology specification. If an attribute *Xyz* is a valid X3D attribute but not a valid VRML attribute for a particular node, then the CSDF requirements specify that the framework class must have a method *setX3DXyz()* and the class must not have the method *setVRMLXyz()*, since *Xyz* is not a valid attribute of the VRML node in question. The parser triggers a *ParseException* when the VRML node has an invalid attribute *Xyz*.

The translation phase refers to translation of the analyzed file into syntax of the output platform of the parsing (not output of the synthesis). The output platform of the translation phase is the CSDF. 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 during parsing process.

X3D Parser

X3D is an XML enabled 3D file format (§II.3.2). XML has a relatively simple syntax when compared to VRML. XML technology has a wide selection of general and powerful parser libraries. One of these libraries is open source Java–based document object model (JDOM) [85]. JDOM integrates with Document Object Model (DOM) and Simple API for XML (SAX) to facilitate reading, writing, and manipulating XML from within Java code. JDOM hides the complexities of XML manipulation.

Each node of X3D has an XML structure in the following form (see Figure 11 on page 25).

```
<element attribute='value'> content </element>
```

The Element tag is the name of the X3D node (e.g., *Material*). Similarly, attributes of the element represents the attributes of the node. With respect to XML, all X3D nodes have their children defined in their `content` section.

X3D analyzer calls JDOM methods to trigger lexical and syntactic analysis on the input X3D file. After these analyses, JDOM creates a tree structure representing the input file as each element becomes a node in the tree. Then, the X3D parser constructs the corresponding CSDF objects as the parser traverses the JDOM tree

```
FOR EACH CHILDREN
  CREATE OBJECT (ACCORDING TO ITS TYPE - USING REFLECTION)
  CALL CONSTRUCTOR RECURSIVELY(PASS JDOM ELEMENT AND PARENT CSDF OBJECT)
    Constructor<?> co = t.getConstructor(new Class[] { Element.class,
                        CSDFNode.class, CSDFFormatType.class });
    addChild((CSDF.CSDFNode) co.newInstance(new Object[] { component,
              this, CSDFFormatType.X3D }));
FOR EACH ATTRIBUTE
MODIFY SYMBOL TABLE IF NECESSARY
SET ATTRIBUTE VALUE (USING REFLECTION)
    Method method = (this.getClass()).getMethod("set" + fieldName,
                     paramClassList);
    Object[] paramList = { fieldValue };
    method.invoke(this, paramList);
```

FIG. 33: X3D parser embedded in constructor of CSDF nodes (recursive construction of CSDF classes).

recursively (Figure 33). Each node in CSDF has the capability to perform type checking on its own node structure and attributes fetched from the corresponding JDOM element. As the elements are fetched by CSDF objects recursively, the framework tree is constructed. At the end of the recursive operation performed on JDOM element objects, the CSDF classes necessary to represent the input X3D file are created and translated.

Virtools

As explained in §IV.1.2 and §II.3.9, Virtools uses a proprietary file format. Thus, an extra analysis layer is implemented in the Virtools development SDK as a plug–in to Virtools (see Figure 34). The Virtools plug–in creates an intermediate file by polling objects in the Virtools scene. This intermediate file is later parsed by Virtools

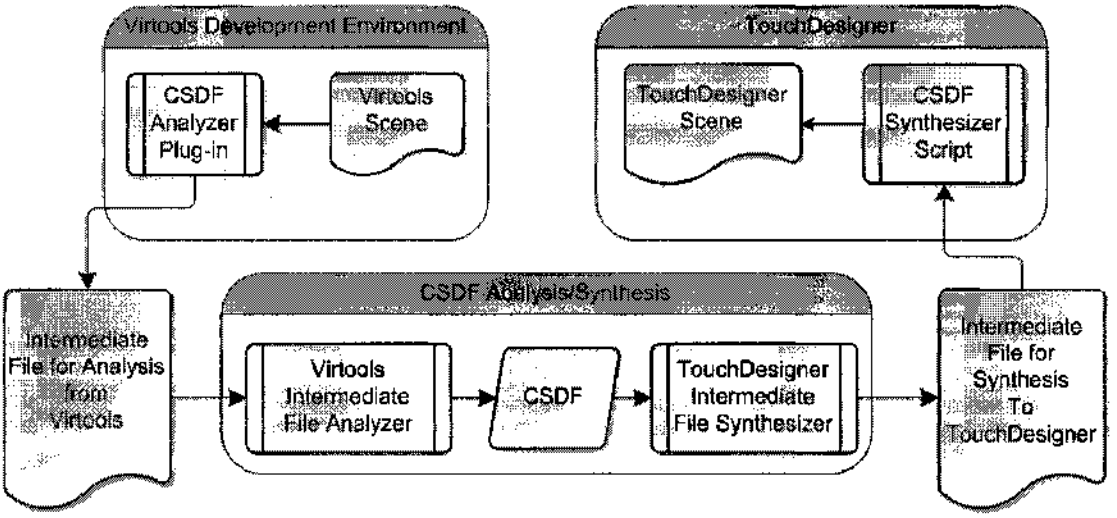


FIG. 34: Virtools to TouchDesigner Analysis/Synthesis.

analysis module that constructs corresponding CSDF objects.

IV.2.3 Synthesis

The synthesis phase of the prototyping methodology maps a conceptual model of the VE, which is CSDF, to a VE at a technology and platform tuple (Figure 35).

Depending on the target technology and platform, the synthesis module may do one or more of the following:

- filter information (e.g., ignoring unsupported capabilities)
- representing higher level abstractions in terms of lower level capabilities at the target
- optimizing by lowering level of fidelity

The requirements specification of the target technology and platform defines the transformations done in the synthesis framework. The scene graph hierarchy of CSDF

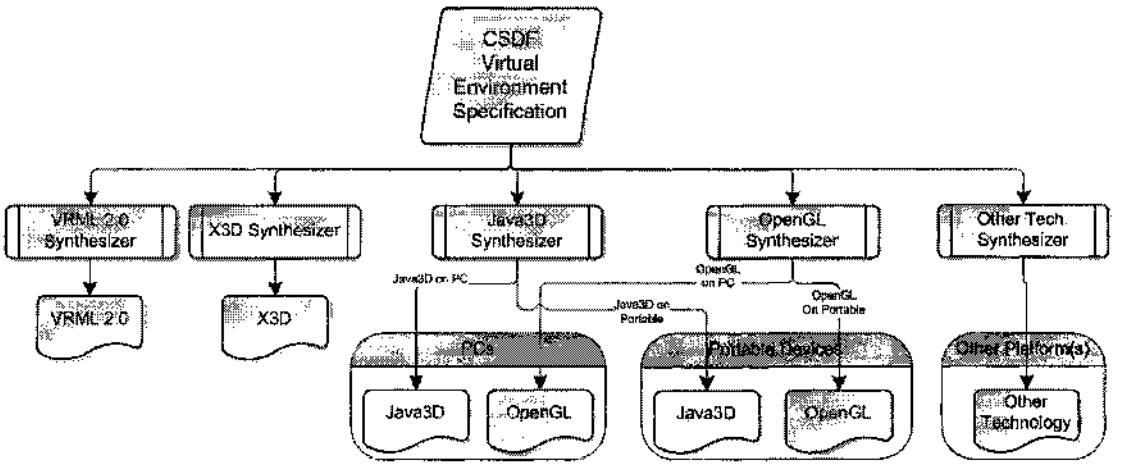


FIG. 35: Synthesis components of CSDF.

makes it logically simple to synthesize individual components within the VE defined in the framework. The synthesis into an output technology that does not have a scene graph organization of components (e.g., OpenGL Synthesis) is more challenging than a technology with scene graph organization (e.g., X3D and Java3D). The transformation to such technology with no scene graph organization (Immediate Mode) can be done by carefully describing the rendering steps necessary for target VE, at each node of CSDF. The VE as represented by the common scene format has all the information necessary to synthesize the world. Thus, starting from the root node of the CSDF, it is theoretically possible to collect all information stored at different levels of the hierarchy by transversing the scene graph tree and to transform scene in CSDF to the desired target format.

In the synthesis process, the scene graph organization provides a simple method to produce an output for the target technology. The scene graph structure in CSDF is traversed recursively starting with the root node of the tree. In this recursive

traversal, the corresponding synthesis function is invoked at each node. The synthesis function is defined by *CSDFSynthesisInterface* and implemented by every CSDF class. The synthesis function produces the necessary translation for the target technology. When the traversal is complete, synthesis is complete. Currently the prototyping system is able to synthesize a subset of the features of VRML 1.0, VRML 2.0, X3D, Java3D, JavaME, JavaFX, OpenGL and TouchDesigner, each with their own subset capabilities to show proof of concept. Chapter V gives the results from testing done via some series of analysis/synthesis processes.

IV.2.4 Authoring

Authoring is currently supported via application development using CSDF. A Java program using CSDF classes and its respective methods can author VEs residing in CSDF, which are either analyzed from other technologies or saved at an earlier point using Java Serialization. It is also possible to author multiple VEs using multiple sets of CSDF classes.

IV.2.5 Current Implementation status

Table VI depicts the supported analysis and synthesis capabilities for CSDF. A class or a group of classes represent a capability in virtual environment. For example, *CSDFIndexedFaceSet* represents the capability of creating a virtual object out of multiple geometric faces.

TABLE VI: Analysis and Synthesis Support for CSDF.

Class Name	Analysis				Synthesis							
	CSDF	VRML 2.0	X3D	Virtools	CSDF	VRML 1.0	VRML 2.0	X3D	Java3D	OpenGL	JavaFX	TouchDesigner
CSDFAppearance	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	
CSDFBackground	✓	✓	✓		✓		✓	✓				
CSDFBox	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	
CSDFColor	✓	✓	✓		✓		✓	✓	✓	✓	✓	
CSDFCone	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	
CSDFCoordinate	✓	✓	✓		✓		✓	✓	✓	✓	✓	
CSDFCylinder	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	
CSDFDirectionalLight	✓				✓							
CSDFExternProtoDeclare	✓		✓	✓	✓			✓				✓
CSDFField	✓				✓							
CSDFFieldObjRef	✓				✓							
CSDFFontStyle	✓	✓	✓		✓	✓	✓	✓				
CSDFGroup	✓	✓	✓		✓	✓	✓	✓	✓		✓	
CSDFImageTexture	✓	✓	✓		✓	✓	✓	✓	✓		✓	
CSDFIndexedFaceSet	✓	✓	✓		✓	✓	✓	✓		✓		

Continued on Next Page...

TABLE VI – Continued

Class Name	Analysis				Synthesis							
	CSDF	VRML 2.0	X3D	Virtools	CSDF	VRML 1.0	VRML 2.0	X3D	Java3D	OpenGL	JavaFX	TouchDesigner
CSDFIndexedLineSet	✓	✓	✓		✓	✓	✓	✓				
CSDFMaterial	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	
CSDFNavigationInfo	✓	✓	✓		✓		✓					
CSDFNode	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	
CSDFNormal	✓	✓	✓		✓		✓	✓				
CSDFOrientationInterpolator	✓	✓	✓		✓		✓	✓				
CSDFPixelTexture	✓	✓	✓		✓		✓					
CSDFPlaneSensor	✓	✓	✓		✓	✓	✓	✓	✓		✓	
CSDFPointLight	✓	✓	✓		✓		✓					
CSDFPointSet	✓	✓	✓		✓		✓	✓				
CSDFPositionInterpolator	✓	✓	✓		✓		✓	✓				
CSDFProtoInstance	✓		✓	✓	✓			✓				✓
CSDFProximitySensor	✓	✓	✓		✓		✓	✓				
CSDFRoute	✓	✓	✓		✓		✓	✓	✓		✓	
CSDFScene	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CSDFScript	✓	✓	✓		✓		✓	✓				

Continued on Next Page...

TABLE VI – Continued

Class Name	Analysis				Synthesis							
	CSDF	VRML 2.0	X3D	Virtools	CSDF	VRML 1.0	VRML 2.0	X3D	Java3D	OpenGL	JavaFX	TouchDesigner
CSDFShape	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	
CSDFSphere	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	
CSDFSwitch	✓	✓			✓		✓					
CSDFText	✓	✓	✓		✓		✓	✓				
CSDFTextureCoordinate	✓	✓	✓		✓		✓	✓				
CSDFTextureTransform	✓	✓	✓		✓		✓	✓				
CSDFTimeSensor	✓	✓	✓		✓		✓	✓				
CSDFTouchSensor	✓	✓	✓		✓		✓	✓				
CSDFTransform	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	
CSDFUse	✓	✓	✓		✓		✓	✓				
CSDFViewpoint	✓	✓	✓		✓		✓	✓				

CHAPTER V

APPLICATIONS

In this chapter, proof of concept testing results are given. Two VEs are described; a simple VE and the Virtual Operating Room. Then, the appropriate analysis and synthesis processes are given.

V.1 A SIMPLE VE

A simple VE is chosen to demonstrate some of the capabilities of the rapid prototyping framework. A VRML 2.0 implementation and a view (in Cortona Player [86]) of the selected VE are shown in Figure 36 and Figure 37, respectively.

The scene consists of a sphere and a box (Figure 36). In the scene, a plane sensor (*PlaneSensor*) node is associated with the sphere. *pSensor*, plane sensor, is routed to the box object so that any action on the plane sensor affects the box object. When a mouse drag is applied on the sphere, the box translates in the magnitude and direction of the mouse drag on the sphere. *Route* node provides the routing of the events to the target node (*Odu*), so that the actions on the *PlaneSensor* node are translated to the target node.

By using this simple VE developed in VRML 2.0, a demonstration of the framework is given for the following reasons.

- The framework should be able to create aggregate objects from geometry primitives in terms of the organization inside the framework (CSDF).

- If analysis and synthesis are performed from and to the same technology and platform, the framework should be able to preserve the original definition of the VE provided that all the capabilities are implemented per the technology in question.

- When synthesizing to limited capability platform, optimizations such as removing and lowering LOD (level of detail) should be performed by the synthesizer. Then, a warning should be produced to the developer of the VE.

- For capabilities implemented in a different manner for a different technology and platform, the framework should be able provide necessary translations for the capabilities.

The representation of behaviors, texture application processes and navigation methodologies in VEs are quite varied. The selected simple VE provides sufficient complexity to show some of the conceptual problems and solutions.

In the following subsections, various synthesis demonstrations for the simple VE residing in CSDF (through analysis from VRML 2.0) are given.

V.1.1 VRML Synthesis

The synthesis to VRML 2.0 of the simple VE is a one to one mapping from the input scene with no loss of capability. The input parser for VRML 2.0 is able to populate the CSDF framework classes with the attributes and behaviors of the corresponding VRML 2.0 input scene. Then, the framework classes are synthesized to VRML 2.0 format without any losses of information and capabilities.

```
#VRML V2.0 utf8
DEF Odu Transform{
  children[
    Transform{
      translation -3.0 0.1 0.2
      rotation 0.0 0.70710677 0.70710677 0.9
      children Shape{
        appearance Appearance{
          material Material{
            diffuseColor 0.0 0.0 1.0
          }
          texture ImageTexture {
            url "oduncrown.gif"
            repeatS TRUE
            repeatT TRUE
          }
        }
        geometry Box{}
      }
    }
  ]
}
DEF slider Transform {
  children[
    Shape {
      appearance Appearance{
        material Material{
          ambientIntensity 1.0
          diffuseColor 1 0 0
        }
      }
      geometry Sphere{
        radius 1
      }
    }
    ,
    DEF pSensor PlaneSensor{enabled TRUE}
  ]
}
ROUTE pSensor.translation_changed TO Odu.set_translation
```

FIG. 36: VRML 2.0 implementation of the simple VE.

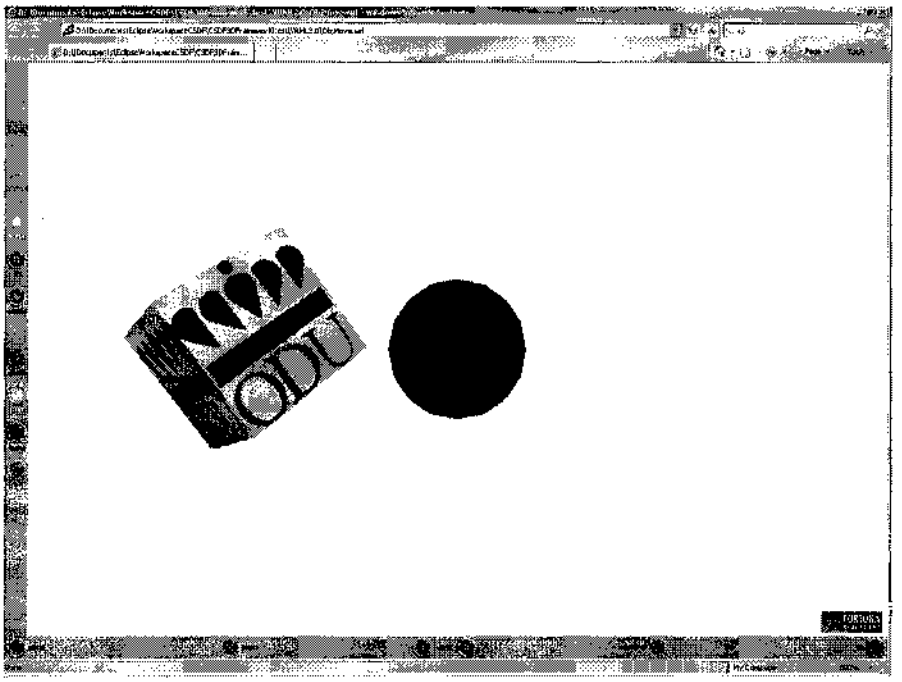


FIG. 37: VRML 2.0 implementation of the simple VE in Cortona plug–in for Internet Explorer.

VRML 1.0 has capabilities to represent geometric shapes and apply textures to surfaces, but does not have any capabilities for dynamic interactions (behaviors). The input scene has features that cannot be represented on the target platform. Hence this functionality cannot be represented in the output format. The CSDF framework synthesizes the geometry primitives and appearance accurately. The drag functionality is lost in the translation to a less capable VR platform.

V.1.2 X3D Synthesis

Since X3D is a more capable specification that includes all the capabilities of VRML 2.0 specification, the transformation from a VRML 2.0 to X3D format is also a loss-less transformation. The input parser for VRML 2.0 is able to populate the CSDF

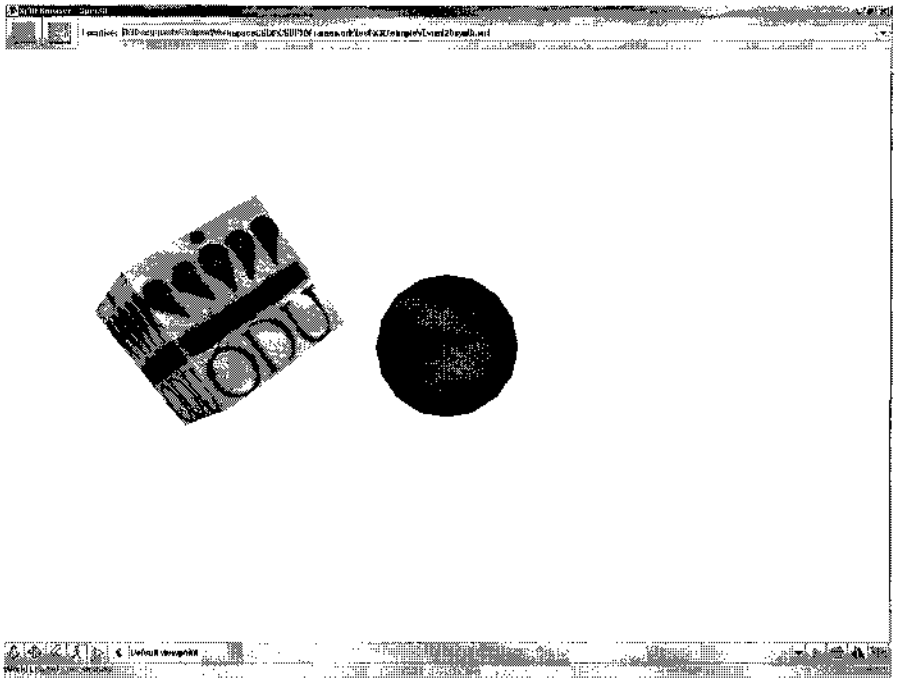


FIG. 38: X3D view of the synthesized VE in Xj3D Browser.

framework classes with the attributes and behaviors of the corresponding VRML 2.0 input scene and synthesize to X3D format without any loss of information. Figure 38 shows the screen shot of the output VE in X3D format in Xj3D Browser.

V.1.3 Java3D Synthesis

In a VRML and X3D, some features (e.g., default lighting, movement, zoom and pan) are automatically provided by the viewers/players of the technologies. In a Java3D scene, the developer must implement all these essential features to the scene. Navigation and views are more complicated in that the developer must use the API and program these capabilities. A content designer unfamiliar with Java3D or unaccustomed to programming will face challenges in producing compatible functionality.

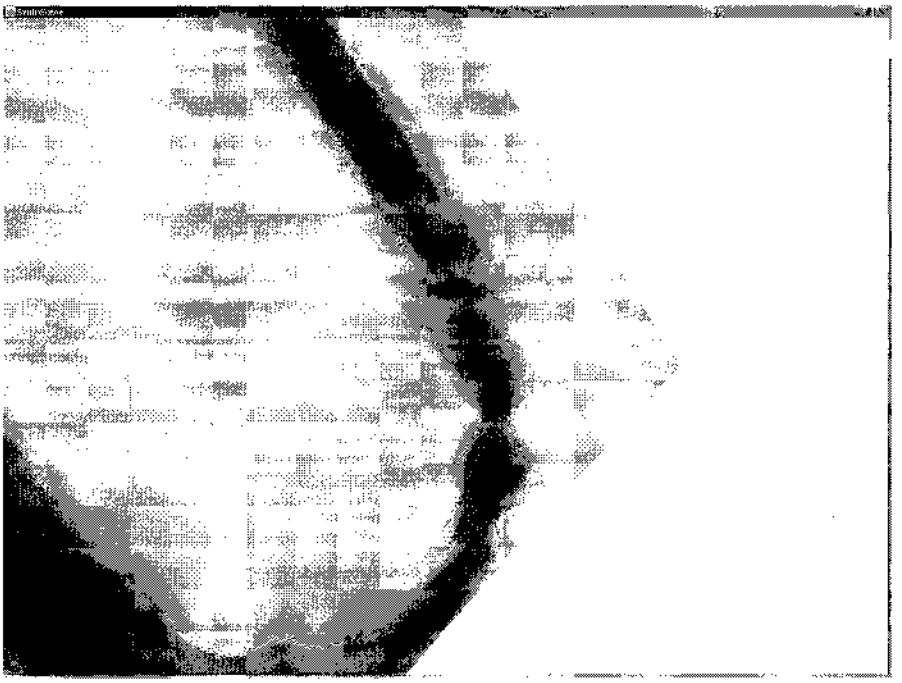
This expertise in the specific technology is incorporated into the synthesized VE by synthesis module of the framework. Figure 39 shows two screen shots of the output VE in Java application. The first one (Figure 39(a)) is taken before the user made panning and zooming to reveal the scene (Figure 39(b)). The default camera position and frustum orientation in Java3D is different than VRML and X3D. This results in a blocked initial view, one of the capability synthesis problems.

In order to provide a VE experience close to input technology, VRML 2.0, zoom and rotate capability along with a light source (*Headlight* in VRML) are integrated into the target VE with CSDF synthesis.

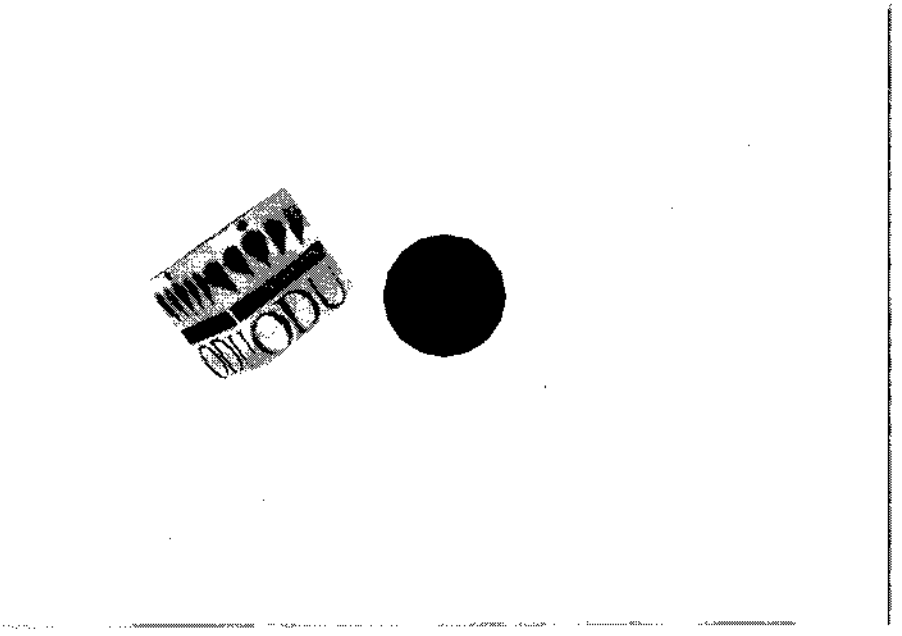
In Java3D, there is no class that directly handles the drag functionality, which exists in VRML 2.0 and X3D. Indeed, the drag functionality must be implemented using events and callback handlers. The Java3D *PickTranslateBehavior* class, which can be associated with any geometry primitive, has the capability to allow for picking of the geometry. A callback function is registered to handle *PickTranslateBehavior* events. In the callback function, the coordinates of the target geometry are transformed in magnitude as returned by the *PickTranslateBehavior* node. This is a proper example showing how CSDF can aggregate capabilities at the output technology to match the capability at the input technology.

V.1.4 OpenGL/GLUT Synthesis

OpenGL is a low–level API for rendering 3D VEs. It has virtually all capabilities of rendering systems, but it lacks high level functions such as behavior models. The developer of the OpenGL VE is responsible for implementing such behaviors and



(a) Java3D



(b) Java3D Corrected View

FIG. 39: Java3D view of the synthesized environment.

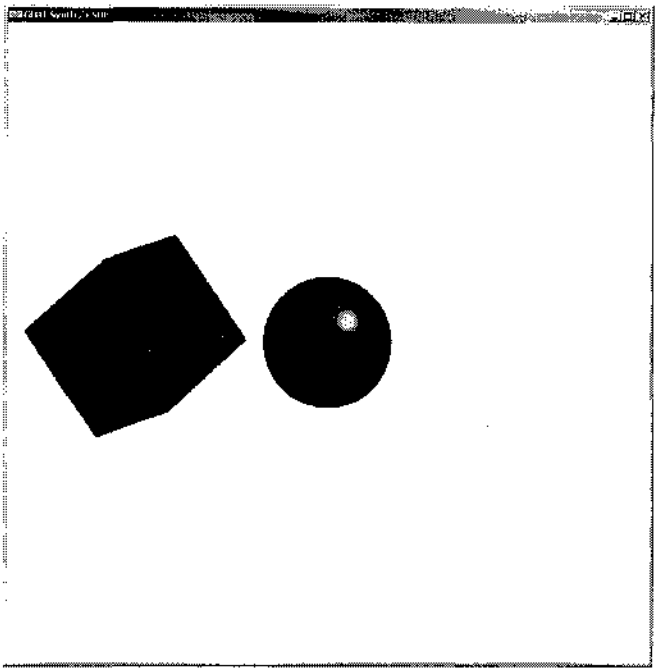


FIG. 40: OpenGL view of the synthesized VE.

other high–level functions. Libraries such as GLUT exist, but these libraries do not include high–level functional capabilities like Java3D and X3D. Thus, implementing some of these high–level functions in OpenGL is costly, and many different ways of implementation are possible. Thus, selecting the most effective method might be problematic.

To demonstrate the OpenGL synthesis, synthesis is implemented only for a handful of these higher level functions and is adequate to demonstrate the concept. In Figure 40, a screen shot of the synthesized OpenGL application is shown. The texture mapping and pick capability are missing from the original simple VE only because of practicality. The OpenGL synthesis virtually can realize any input capability.

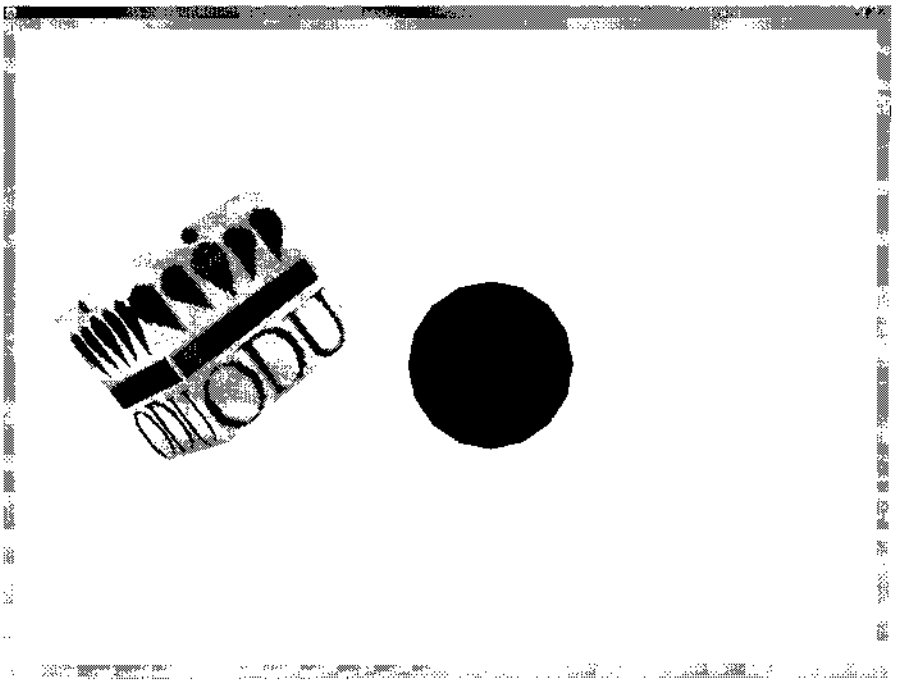


FIG. 41: JavaFX view of the synthesized VE.

V.1.5 JavaFX Synthesis for Portable Device

Although JavaFX uses Java3D classes for 3D rendering, JavaFX's script-like structure is incompatible with Jave3D code and thus does not permit the use of Java3D's synthesis module. A separate synthesis module is implemented, so that semantic differences in JavaFX are taken into account. The synthesis of the simple VE to JavaFX (Figure 41) can be done without any loss of capability. However, there are a few portable devices supporting JavaFX at the present.

V.2 VOR

Virtual Operating Room (VOR) [87, 88, 89] is a VE system that integrates procedural medical simulators into a context-relevant individual or team training facility.

Trainees interact with a surgical team comprised of real and/or virtual team members (e.g., attending surgeon, anesthesiologist, scrub technician, and circulating nurse). The virtual members (agents) of the surgical team are realized in a CAVE–like immersive projection platform (e.g., CAVE and Vision Dome). In order to render VOR, the Virtools VR Player is used. VR Player provides easy integration of VEs to the immersive projection system. Any VE developed on the Virtools environment can be modified to be rendered in CAVE or Vision Dome. Figure 42 shows rendered images on all four projection walls of a CAVE. Visual components of VOR that are subject to analysis by CSDF are agent models and parts of the operating room. The model is mostly designed via use of other tools such as 3DStudioMax, but later imported into Virtools to constitute a composition file. However, model positions and orientations are adjusted in Virtools Composition. The following sub–section describes the analysis process for Virtools composition files into CSDF.

V.2.1 Virtools Analysis

Virtools composition file format is a proprietary file format. Hence, it is not possible to analyze the file directly. An alternative approach is taken to analyze the Virtools compositions. The approach involves a plug–in implementation for Virtools development environment that parses the Virtools scene internally in the environment (Figure 43). The plug–in is implemented using Virtools SDK. The information related to the scene, agent models and the operating room components, are listed along with their position and orientation data in an intermediate file. Later, this intermediate file can be parsed by the Virtools analyzer module in CSDF. These components of

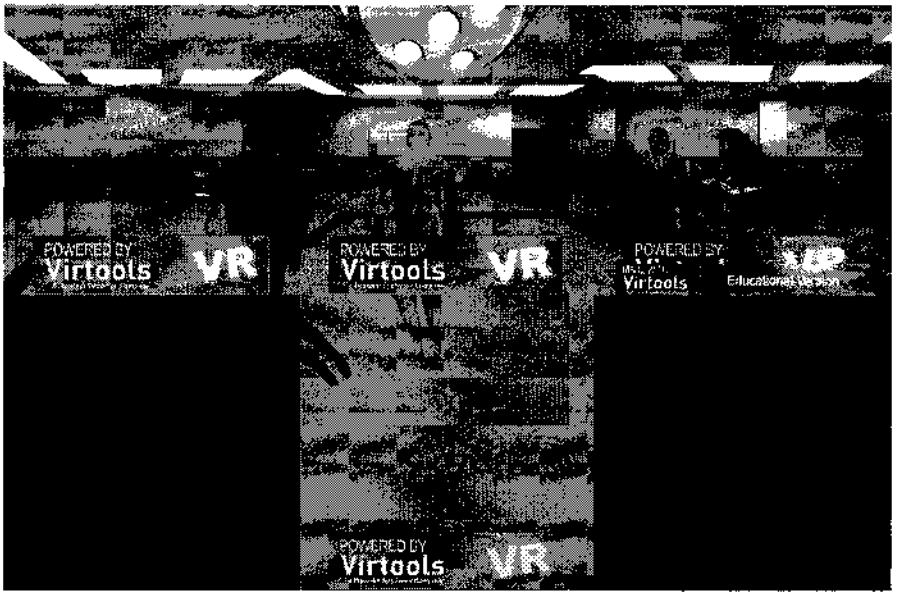


FIG. 42: Virtual Operating Room in Virtools VR Player (Four Projections).

the VE are stored in *CSDFProto* and *CSDFExternProto* nodes, since their geometry is stored in external files (mesh and texture info). A similar type of methodology can be used for many other VE technologies that have proprietary file format.

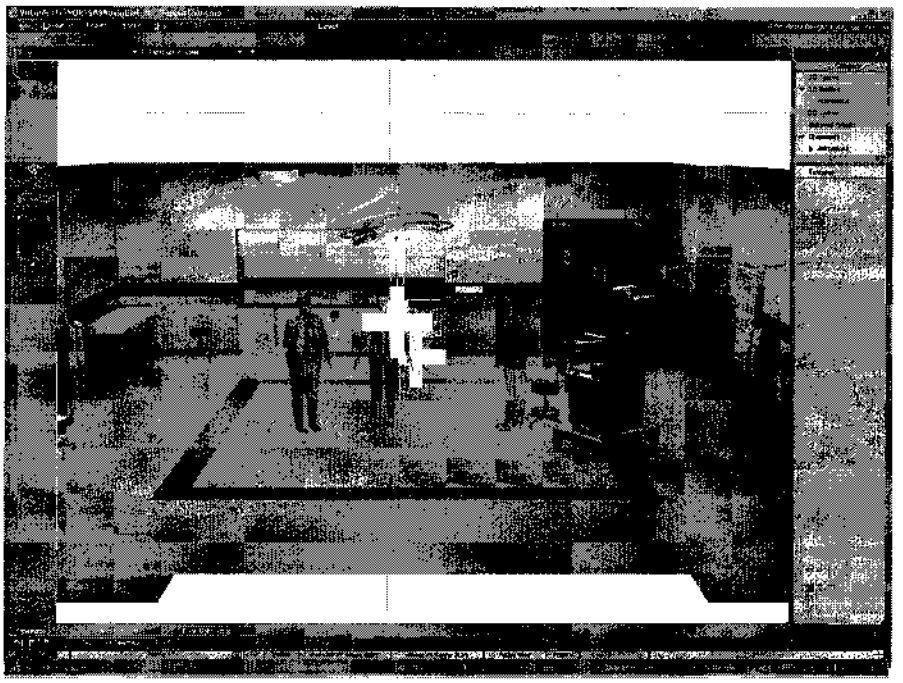


FIG. 43: Virtual Operating Room in Virtools Development Environment.

CHAPTER VI

CONCLUSIONS

The work described in this thesis is a conceptual process for rapidly prototyping VEs. In order to achieve rapid prototyping, CSDF was developed. CSDF serves as a superset/model representation of the VE technologies on which rapid prototyping is applied. The prototyping is provided by a synthesis to automate the migration of a VE to a new VE on target technology/platform that is unfamiliar to the developer. Analysis modules are presented to populate the CSDF with the capabilities of the input technology. Even though X3D, VRML, and Virtools are considered as input technology in implementation for the demonstration, it is possible to apply the same analysis approach to any other VE technologies. Hence, ultimately, CSDF can be the superset of all technologies that are considered in the implementation of CSDF. CSDF synthesis modules are built to achieve automatic synthesis to target technology and platforms. Again, for demonstration purposes, a limited number of technologies are targeted (i.e., VRML, X3D, Java3D, JavaFX, JavaME, OpenGL, and TouchDesigner), and only a subset of capabilities was chosen for each technology to experiment the research ideas to be used in the overall implementation. The framework is kept extensible so that either more capabilities of the existing technologies can be added or a completely new technology can be easily implemented.

In addition to Analysis and Synthesis capabilities, the rapid prototyping system provides extra capabilities such as authoring and storage of the VEs residing in CSDF.

Hence, if needed, VEs can be authored in this superset of all technologies to ensure that the maximum number of capabilities are supported on target VEs. Another important feature of the prototyping system is its knowledge of the capability limitations for a particular synthesis platform. Thus, an early feedback or warning can be produced for the user during the synthesis process, when such a limitation is detected.

The features of the rapid prototyping system, described in this thesis, offer a contribution to a solution for the problems (§I.3), whereas the other existing frameworks (§I.1) do not offer such contribution.

In the following subsection, a description of possible future work that enhances the dissertation is given.

VI.1 FUTURE WORK

For demonstration purposes, a number of VE technologies and platforms are considered in the implementation in the current version of the framework with a subset of their capabilities. Improving the supported capability subsets of the VE technologies and adding additional technologies are two possible future extensions. Indeed, these kinds of extensions are infinitely available, since newer technologies arise continuously and existing technologies/platforms evolve.

Some other possible improvements or future work are in the following.

- Analysis support for APIs or languages requiring semantic analysis on computer programming languages (e.g., OpenGL, Java3D) can be added. However, it is very hard to find the best representative scene graph structure for immediate

mode APIs or languages. Another dimension of the problem arises from the arbitrary use of custom behaviors and scripting in such technologies. Finding a solid solution to such behavior analysis and representation is worthwhile.

- Java3D (retained mode) technology analysis to CSDF may be achieved by creating an analyzer employing an interpreter to determine the scene graph that a Java3D program constructs. Such an approach might solve analysis problems for technologies similar to Java3D.

- Currently, authoring can be done by developing Java programs using CSDF. A user interface can be created to perform multiple authoring operations on VEs residing in CSDF.

- Collaborative authoring application, where multi users authoring VEs concurrently or iteratively, for CSDF is another possible endeavor. This would facilitate a development environment for multiple users.

- Immersive environments such as CAVE and Vision Dome are widely used VE applications. There are development environments to port VEs to such immersive platforms. Support for immersive platforms can be implemented as a part of technologies such as Java3D and OpenGL.

- Default values of attributes of CSDF classes need to be normalized across different VR platforms. Therefore, automatic synthesis is ensured to produce equivalent VEs for all technologies and platforms.

BIBLIOGRAPHY

[1] C. Cruz-neira, A. Bierbaum, P. Hartling, C. Just, and K. Meinert, "VR Juggler An Open Source Platform for Virtual Reality Applications," in *40th AIAA Aerospace Sciences Meeting and Exhibit 2002*, 2002. `http://www.aiaa.org/content.cfm?pageid=406`.

[2] `http://www.vrjuggler.org/`, 2006.

[3] J. Kelso and L. E. Arsenault, "DIVERSE: A framework for Building Extensible and Reconfigurable Device Independent Virtual Environments," in *IEEE Virtual Reality*, vol. 12, pp. 183 190, 2002.

[4] P. Figueroa, W. F. Bischof, H. J. Hoover, P. Boulanger, and R. Taylor, "InTml: A Dataflow Oriented Development System for Virtual Reality Applications," *Presence: Teleoperators & Virtual Environments*, vol. 17, no. 5, pp. 492–511, 2008.

[5] `http://www.cs.ualberta.ca/~pfiguero/InTml`, 2006.

[6] C. Jeffery, A. Dabholkar1, K. Tachtevrenidis, and Y. Kim, "A Framework for Prototyping Collaborative Virtual Environments ," *Lecture Notes in Computer Science*, vol. 3706, pp. 17 32, 2005.

[7] UVa User Interface Group, "ALICE: rapid prototyping for virtual reality," *IEEE Computer Graphics and Applications*, vol. 15, pp. 8 11, May 1995.

[8] R. Kelly and R. Neetz, "Rapid prototyping: the procedure for software," *Proceedings of the IEEE 1988 National Aerospace and Electronics Conference*, vol. 2, pp. 644 652, 23-27 May 1988.

[9] L. Luqi and R. Steigerwald, "Rapid software prototyping," *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, vol. 2, pp. 470–479, 7-10 Jan 1992.

[10] L. A. Belfore, II, "Rapid Prototyping and Synthesis of Virtual Worlds." Unpublished White Paper, 2003.

[11] J. Crinnion, *Evolutionary Systems Development, a practical guide to the use of prototyping within a structured systems methodology*, ch. 2, pp. 25 41. Plenum Press, 1991.

[12] A. Davis, "Operational prototyping: a new development approach," *IEEE Software*, vol. 9, pp. 70–78, Sep 1992.

[13] R. Acosta, C. Burns, W. Rzepka, and J. Sidoran, "A case study of applying rapid prototyping techniques in the Requirements Engineering Environment," *Proceedings of the First International Conference on Requirements Engineering*, pp. 66–73, Apr 1994.

[14] S. W. Knerr, R. Breaux, S. L. Goldberg, and R. A. Thurman, "National Defense," in *Handbook of Virtual Environments* (K. M. Stanney, ed.), pp. 857–872, Lawrence Erlbaum Associates, 2002.

[15] M. S. Atkin, D. L. Westbrook, and P. R. Cohen, "Capture the Flag: Military Simulation Meets Computer Games," in *Proceedings of AAAI Spring Symposium Series on AI and Computer Games*, pp. 1–5, AAAI Press, 1999.

[16] J. J. Wynn, J. D. Roberts, and M. O. Kinkead, "Visualizing the wargame-web-based applications for viewing a course of action (COA)," in *Proceedings of the 1999 International Conference of Web-Based Modeling & Simulation*, vol. 16, pp. 227–232, 1999.

[17] E. Salas, R. L. Oser, J. A. Cannon-Bowers, and E. Daskarolis-Kring, "Team Training in Virtual Environments: An Event-based Approach," in *Handbook of Virtual Environments* (K. M. Stanney, ed.), pp. 873 892, Lawrence Erlbaum Associates, 2002.

[18] H. G. Weller, "Assessing the impact of computer-based learning on science," *Journal of Research in Computing in Education*, vol. 28, no. 4, pp. 461 485, 1996.

[19] S. V. Cobb, H. R. Neale, J. K. Crosier, and J. R. Wilson, "Development and Evaluation of Virtual Environments for Education," in *Handbook of Virtual Environments* (K. M. Stanney, ed.), pp. 911 936, Lawrence Erlbaum Associates, 2002.

[20] R. M. Satava and S. B. Jones, "Medical Application of Virtual Reality," in *Handbook of Virtual Environments* (K. M. Stanney, ed.), pp. 937–958, Lawrence Erlbaum Associates, 2002.

[21] J. P. Shewchuk, K. H. Chung, and R. C. Williges, "Virtual Environments in Manufacturing," in *Handbook of Virtual Environments* (K. M. Stanney, ed.), pp. 1119 1141, Lawrence Erlbaum Associates, 2002.

[22] E. Badique, M. Cavazza, G. Klinker, G. Mair, T. Sweeney, and D. Thalmann, "Entertainment Applications of Virtual Environments," in *Handbook of Virtual Environments* (K. M. Stanney, ed.), pp. 1143–1166, Lawrence Erlbaum Associates, 2002.

[23] H. Lichter, M. Schneider-Hufschmidt, and H. Züllighoven, "Prototyping in industrial software projects—bridging the gap between theory and practice," in *ICSE '93: Proceedings of the 15th international conference on Software Engineering*, (Los Alamitos, CA, USA), pp. 221 229, IEEE Computer Society Press, 1993. `http://portal.acm.org/citation.cfm?id=257623`.

[24] Y.-R. Lee, S.-Y. Cho, and J.-B. Lee, "The design a virtual prototyping based on ARMulator," *Fourth Annual ACIS International Conference on Computer and Information Science*, pp. 387 390, 2005.

[25] O. Gutierrez, "Prototyping techniques for different problem contexts," in *CHI '89: Proceedings of the SIGCHI conference on Human factors in computing systems*, (New York, NY, USA), pp. 259–264, ACM, 1989.

[26] R. Budde and H. Zullighoven, "Prototyping revisited," *CompEuro '90. Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering*, pp. 418–427, 8-10 May 1990.

[27] S. Haag, M. Cummings, D. J. McCubbrey, A. Pinsonneault, and R. Donovan, *Management Information Systems for the Information Age*, pp. 311–315. McGraw-Hill Ryerson, 2006.

[28] `http://www.web3d.org/x3d/vrml/`, 2007.

[29] A. E. Walsh and M. Bourges-Sevenier, *Core Web3D*, ch. 2, pp. 30–61. Prentice Hall PTR, 2000.

[30] P. V. Krishnan, *Rapid prototyping for the design of virtual worlds*, ch. 3. Old Dominion University, 2005.

[31] J. Hartman and J. Wernecke, *The VRML 2.0 Handbook*, ch. 2–3, pp. 11–59. Addison-Wesley Publishing Company, 1996.

[32] B. Roehl and J. Couch, *Late Night VRML 2.0 with Java*, ch. 1–2. Hightstown, NJ, USA: Ziff-Davis Publishing Co., 1997.

[33] S. Matsuba and B. Roehl, *Using VRML, Special Edition*, ch. 7. QUE Corporation, 1996.

[34] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, "Extensible markup language," *World Wide Web J.*, vol. 2, no. 4, pp. 29 66, 1997.

[35] `http://www.web3d.org/x3d/specifications/`, 2007.

[36] `http://www.web3d.org/`, 2007.

[37] `http://www.web3d.org/about/overview/`, 2007.

[38] http://www.xj3d.org/, 2007.

[39] https://java3d.dev.java.net/, 2006.

[40] D. Selman, *Java 3D programming*, pp. 46 63. Greenwich, CT, USA: Manning Publications Co., 2002.

[41] A. E. Walsh and D. Gehringer, *Java 3D API Jump-Start*, ch. 2, pp. 35–61. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[42] K. Brown and D. Petersen, *Ready-to-Run Java 3D*, pp. 39 49. John Wiley & Sons, Inc., 1999.

[43] https://java.sun.com/javafx, 2007.

[44] R. P. Cook, "The design of a Java phone programming environment," in *PPPJ '07: Proceedings of the 5th international symposium on Principles and practice of programming in Java*, vol. 272, pp. 31–37, ACM, 2007.

[45] K. Pulli, J. Vaarala, V. Miettinen, T. Aarnio, and M. Callow, "Developing mobile 3D applications with OpenGL ES and M3G," in *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, (New York, NY, USA), pp. 1–127, ACM, 2005. http://doi.acm.org/10.1145/1198555.1198730.

[46] http://developers.sun.com/mobility/apis/articles/3dgraphics/, 2004.

[47] http://java.sun.com/javame/, 2007.

[48] http://www.opengl.org/, 2006.

[49] E. Angel, *Open GL Primer*, ch. 1. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[50] D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide: The Official Guide to Learning OpenGL*, pp. 7–19. Addison-Wesley Professional, 2005.

[51] E. Angel, *Interactive Computer Graphics: A Top-Down Approach With OPENGL primer package*, ch. 1-2, 13. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2001.

[52] `http://msdn.microsoft.com/en-us/directx/`, 2008.

[53] `http://www.virtools.com/`, 2008.

[54] M. W. Scerbo, H. M. Garcia, A. Nalu, and E. Baydogan, "Chain Saw Virtual Reality Safety Trainer." Unpublished White Paper, `http://www.lions.odu.edu/~ebayd001/misc/papers/chainsaw.pdf`, 2006.

[55] `http://www.derivativeinc.com/tools/touchdesigner.asp`, 2008.

[56] Y. Shen, J. Seevinck, and E. Baydogan, "Realistic Irrigation Visualization in a Surgical Wound Debridement Simulator," in *Proceedings of Medicine Meets Virtual Reality Conference*, vol. 119, pp. 512–514, 2006.

[57] J. A. Seevinck, M. W. Scerbo, L. A. Belfore, II, L. J. Weireter, J. J. R. Crouch, Y. Shen, F. D. McKenzie, H. M. Garcia, S. Girtelschmid, E. Baydogan, and E. A.

Schmidt, "A Simulation-Based Training System for Surgical Wound Debridement," in *Proceedings of Medicine Meets Virtual Reality Conference*, vol. 119, pp. 491–496, 2006.

[58] J. A. Seevinck, M. W. Scerbo, L. A. Belfore, II, L. J. Weireter, J. J. R. Crouch, Y. Shen, F. D. McKenzie, H. M. Garcia, S. Girtelschmid, E. Baydogan, E. A. Schmidt, and D. Spence, "Surgical Wound Debridement Simulation-Based Training System," in *Advanced Technology Applications for Combat Casualty Care (ATACCC) conference*, pp. 15 -17, 2005.

[59] `http://www.microsoft.com/windows/`, 2008.

[60] A. Siberschatz and P. B. Galvin, *Operating System Concepts*, pp. 737–741. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1993.

[61] A. Mason, "Criteria for UNIX-like systems," *SIGSMALL Newsletter*, vol. 8, no. 3, pp. 8 -13, 1982.

[62] `http://www.us.playstation.com/PSP/`, 2008.

[63] `http://www.nintendo.com/ds`, 2008.

[64] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart, "The CAVE: audio visual experience automatic virtual environment," *Communications ACM*, vol. 35, no. 6, pp. 64–72, 1992.

[65] M. Czernuszenko, D. Pape, D. Sandin, T. DeFanti, G. L. Dawe, and M. D. Brown, "The ImmersaDesk and Infinity Wall projection-based virtual reality displays," *SIGGRAPH Computer Graphics*, vol. 31, no. 2, pp. 46–49, 1997.

[66] `http://www.vrealities.com/vrdomes.html`, 2008.

[67] O. Cakmakci and J. Rolland, "Head-worn displays: a review," *Journal of Display Technology*, vol. 2, pp. 199–216, Sept. 2006.

[68] D. H. Eberly, *3D game engine design: a practical approach to real-time computer graphics*, ch. 1–2. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000.

[69] T. Strothotte and S. Schlechtweg, *Non-photorealistic computer graphics: modeling, rendering, and animation*, ch. 6. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002.

[70] `http://glide.sourceforge.net/`, 2008.

[71] L. A. Belfore, II, P. V. Krishnan, and E. Baydogan, "Common scene definition framework for constructing virtual worlds," in *WSC '05: Proceedings of the 37th conference on Winter simulation*, vol. 3, pp. 1985–1992, Winter Simulation Conference, 2005.

[72] T. Greanier, "Java Serialization." `http://java.sun.com/developer/technicalArticles/Programming/serialization/`, July 2000.

[73] A. Bierbaum, *VR Juggler: A Virtual Platform for Virtual Reality Application Development*, ch. 1. Iowa State University, 2000.

[74] L. Arsenault, J. Kelso, R. Kriz, and F. D. Neves, "Diverse: a software toolkit to integrate distributed simulations with heterogeneous virtual environments." `http://www.diverse.vt.edu`, 2001.

[75] `http://java.sun.com/`, 2007.

[76] K. Reinholtz, "Java will be faster than C++," *SIGPLAN Notes*, vol. 35, no. 2, pp. 25–28, 2000.

[77] S. S. Chandra and K. Chandra, "A comparison of Java and C#," *Journal of Computer Small Collection*, vol. 20, no. 3, pp. 238–254, 2005.

[78] S. Sangappa, K. Palaniappan, and R. Tollerton, "Benchmarking Java against C/C++for interactive scientific visualization," in *JGI '02: Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande*, (New York, NY, USA), pp. 236–236, ACM, 2002.

[79] `http://java.sun.com/docs/books/tutorial/reflect/`, 2007.

[80] M. Braux and J. Noyé, "Towards partially evaluating reflection in Java," in *PEPM '00: Proceedings of the 2000 ACM SIGPLAN workshop on Partial evaluation and semantics-based program manipulation*, vol. 34, pp. 2 11, ACM, 1999.

[81] I. R. Forman and N. Forman, *Java Reflection in Action (In Action series)*, ch. 1, pp. 3 26. Greenwich, CT, USA: Manning Publications Co., 2004.

[82] http://www.eclipse.org/, 2007.

[83] https://javacc.dev.java.net/, 2007.

[84] A. W. Appel and J. Palsberg, *Modern Compiler Implementation in Java*, ch. 3. New York, NY, USA: Cambridge University Press, 2003.

[85] http://www.jdom.org/, 2006.

[86] http://www.parallelgraphics.com/products/cortona/, 2006.

[87] E. Baydogan, S. Mazumdar, and L. A. Belfore, II, "Decoupled Agent Architecture for Virtual Operating Room Training Simulations," in *CGVR '08: Proceedings of The 2008 International Conference on Computer Graphics and Virtual Reality*, CGVR08, Las Vegas, 2008. http://www.lions.odu.edu/~ebayd001/misc/papers/davortsCgvr08.pdf.

[88] M. W. Scerbo, L. A. Belfore, II, H. M. Garcia, L. J. Weireter, M. W. Jackson, A. Nalu, and E. Baydogan, "The Virtual Operating Room," in *IITSEC '06: The Interservice/Industry Training, Simulation & Education Conference*, vol. 2006(Conference Theme: Training 21st Century Joint Force), 2006.

[89] M. W. Scerbo, L. A. Belfore, II, H. M. Garcia, J. Leonard J. Weireter, M. W. Jackson, A. Nalu, E. Baydogan, J. P. Bliss, and J. Seevinck, "A Virtual Operating Room for Context-Relevant Training," in *Proceedings of the Human Factors and Ergonomics Society 51st Annual Meeting–2007*, vol. 51, pp. 507 511, 2007.

VITA

Emre Baydogan
Department of Electrical and Computer Engineering
Old Dominion University
Norfolk, VA 23529

EDUCATION

PhD Candidate in Electrical and Computer Engineering — 2001 - 2008
Old Dominion University, Norfolk, VA
Advisor: Lee A. Belfore, II
Grade Point Average: 3.88/4.00

MS in Computer Science and Engineering — 1999 - 2001
Marmara University, Istanbul, Turkey
Advisor: M. Borahan Tumer
Thesis: Adaptive Data Compression in Networks: a Learning Automaton Approach
Grade Point Average: 85/100

BS in Computer Science and Engineering — 1994 - 1999
Marmara University, Istanbul, Turkey — High Honors
Grade Point Average: 3.58/4.00
Rank in major: 4 out of approx. 35
Rank in class: 5 out of approx. 120

Highschool in Pure and Applied Sciences — 1991 - 1994
Tuna High School, Istanbul, Turkey — High Honors
Grade Point Average: 4.21/5.00

WORK EXPERIENCE

Research Assistant — 9/03 - 5/08
VMASC, ODU, Norfolk, VA
Virtual Operating Room

designed system architecture for simulation
implemented the controller and the modules for the simulation, using Java, Visual C++, Visual Basic, Virtools, Dragon Naturally Speaking SDK, Microsoft Speech API
Wound Debriedement Simulator
implemented the modules for the simulation, using Visual C++, Touch Designer Haptics DSK, Haptek SDK
Virtual Chainsaw
implemented the modues for the simulation. using Visual C++, Virtools, Haptics SDK
VV&A for Enhanced Logistics Intratheater Support Tool
validated ELIST simulator
VRGait
implemented communication module for the simulator

Lab and Network Administrator 9/99 - 5/01
Marmara University, Istanbul, Turkey
administered laboratory containing 20 Intel based computers, one MS NT based server, and two Sun workstations

Intern, Accounting Department Summer/Winter 97 - 98
Krevitas Food Ind. Inc., Istanbul, Turkey
helped design product/raw product codes and helped customization of SAP/3

Intern, Accounting Department Summer/Winter 94 - 96
Global Securities Inc., Istanbul, Turkey
helped manage stock operations in Takasbank

TEACHING EXPERIENCE

Teaching Assistant 9/01 - 5/03
Old Dominion University, Norfolk, VA
responsible for grading labs/projects/assignments/exams
ECE 241/341 - Digital Design

Instructor 6/03 - 7/03

Old Dominion University, Norfolk, VA

responsible for teaching computer skills to K12 students (ECE Sponsorhip)

Teaching Assistant 9/99 - 5/01

Marmara University, Istanbul, Turkey

responsible for conducting lab sessions, grading labs/projects/assignments/exams

CSE 315 - Digital Design

CSE 337 - Computer Organization II

CSE 344 - Software Engineering

CSE 432 - Distributed Systems

CSE 381 - Modeling and System Simulation

CSE 482 - Introduction to Artificial Intelligence

PUBLICATIONS

E. Baydogan, S. Mazumdar, L. A. Belfore, "Decoupled Agent Architecture for Virtual Operating Room Training Simulations", CGVR '08: Proceedings of The 2008 International Conference on Computer Graphics and Virtual Reality, 2008.

E. Baydogan, S. Mazumdar, L. A. Belfore, "Simulation Architecture for Virtual Operating Room Training", 2008 VMASC Capstone Conference, 2008.

S. Mazumdar, **E. Baydogan**, L. A. Belfore, "OntoVOR: The Design of a Knowledge-base for a Virtual Operating Room", Proceedings of the 2007 Modsim World Conference, 2007.

M. Scerbo, L. A. Belfore II, H. M. Garcia, L. J. Weireter, Jr, M. W. Jackson, A. Nalu, **E. Baydogan**, J. P. Bliss, J. Seevinck, "A Virtual Operating Room for Context Relevant Training", Proceedings of Human Factors and Ergonomics Society 51st Annual Meeting, 2007

Y. Shen, J. Seevinck, and **E. Baydogan**, "Realistic Irrigation Visualization in a Surgical Wound Debridement Simulator", Medicine Meets Virtual Reality 14, pp 512-514, (Long Beach, CA), January 2006.

J. A. Seevinck, M. W. Scerbo, L. A. Belfore II, L. J. Weireter, Jr. J. R. Crouch, Y. Shen, F. D. McKenzie, H. M. Garcia, S. Girtelschmid, **E. Baydogan**, E. A. Schmidt, "A Simulation-Based Training System for Surgical Wound Debridement",

Proceedings of the 14th Medicine Meets Virtual Reality Conference, Long Beach, California, January 2006.

L Belfore, J Crouch, Y Shen, S Girtleschmid, **E Baydogan**. "A Software Framework for Surgical Simulation Virtual Environments". Medicine Meets Virtual Reality 14, IOS Press, pp. 46-48, 2006.

M. Scerbo, L. Belfore, H. Garcia, M. Jackson, A. Nalu, **E. Baydogan**, L. Weireter, "The Virtual Operating Room", Proceedings of the 2006 Interservice/Industry Training, Simulation & Education Conference, Dec. 4-7, 2006, Orlando, Florida.

L. A. Belfore, P.V. Krishnan, and **E. Baydogan**, "Common Scene Definition Framework For Constructing Virtual Worlds", Proceedings of the 2005 Winter Simulation Conference, 2005.

J. Seevinck, M. W. Scerbo, L. A. Belfore II, L. Weireter, Jr., J. Crouch, Y. Shen, F.D. McKenzie, H.M. Garcia, S. Girtelschmid, **E. Baydogan**, E. A. Schmidt, D. Spence, "Surgical Wound Debridement Simulation-Based Training System", Project description and prototype demonstration at Advanced Technology Applications for Combat Casualty Care (ATACCC) conference. ATACCC St Pete Beach, USA. 15-17 August 2005.

Typeset using LaTeX.