Rapid Development of Virtual Environments

A systematic approach for interactive design of 3D graphics

Xuelei Qian
University of Derby
Kedleston Road
DE22 1GB, Derby, Derbyshire
x.qian@ieee.org

Zhengxu Zhao
University of Derby
Kedleston Road
DE22 1GB, Derby, Derbyshire
z.zhao@derby.ac.uk

Richard Thorn
University of Derby
Kedleston Road
DE22 1GB, Derby, Derbyshire
r.thorn@derby.ac.uk

ABSTRACT

It has long been a bottleneck for VE popularity that the development of VE normally acquires heavy time, labour and monetary investment. Although so-called high-level, abstracted graphical libraries which have been delivered by third parties based on industrial standard like OpenGL speed up the VE development to certain extent, the involved engineering process which largely relies on the system computing approach is by all means not developer-oriented but application-specific, thus it remains technically difficult and expensive to create VE application from scratch. This research attempts to propose an ultimate solution for VE rapid development by exploring the boundary between system programming, interpretative computing, interfaces wrapping, abstracted scene-graph libraries, grouping and database technology. The convergence of ideas from these technological fields has formed a systematic approach by which developers are encouraged to design and implement 3D interactive graphics via making necessary reconfiguration to both graphical content and rendering context take place at system runtime. The whole development cycle of VE application can be further accelerated by using similar existing drawings from the database as reconfigurable VE templates. In this way, the developers can avoid creating graphical application completely from scratch by making runtime changes to retrieved VE template in terms of its rendered graphics, user interfaces and related functional modules.

Keywords
Virtual Environment, Rapid Development, 3D Interactive Graphics, Runtime Reconfiguration

1. INTRODUCTION

Although virtual reality (VR) technology has become prevalent in modern 3D design, training, education and media applications, construction of virtual reality worlds or virtual environments (VEs) remains a technically difficult and time consuming process [Oli03a] [Ran95a] [Zha98a]. Therefore rapid modeling of VEs is recently a much-researched area [Gri96a] [Win95a]. Since Randy (1995) first proposed to use scripting or interpreter-based programming as a *de facto* design paradigm for rapid prototyping large-scaled and complex VEs, little similar research has been reported in the literature. In this paper, we present the idea of "runtime evolution" for construction of VEs. The approach behind this idea is to find out similar drawings from VE database. The expected VE is to be created by revising different parts and reserving the uniform regions in VE template. In traditional approaches, such reconfiguration to VE is only possible with a so-called "system reengineering process", which means both graphics and their control can only be changed in off-line mode even though only a minor change to VE is required. Intending to shift developers away from such a source code reengineering process imposed by compiled language driven environment, a scripting, namely, Tcl/Tk [Ous98a] based infrastructure was proposed with provision of "VE rapid prototyping system", which is dedicated to simplify the VE design and implementation process by realizing "interactive design". Interactive design kits free VE engineers from struggling with edit-compilation-linking process via a straightforward way of implementing runtime code interpretation. A comprehensive library of graphical rendering components bundled with the Tcl interpreter has been adopted to develop variant approaches for reconfiguring VE at system runtime. Further more, one of the most attractive facts is that a well-developed VE database management system provides the top-down administration for VE templates which are clustered by VE coding and classification module. Newly developed VE can be featured with predefined dependencies and assigned with unique identification code for VE query.

The proposed infrastructure is intended to integrate the off-the-shelf computing and graphical rendering technologies into a single platform, within which the VE design, implementation and reconfiguration can be done in an effortless way. Its software architecture includes VRS [Döl02a] enabled 3D data visualization unit, Tcl/Tk enabled VE reengineering process component and a unified scripting interpreter. In this paper, development of a demonstration system is also reported to provide concrete details of proposed infrastructure. The benefits of applying the scripting based VE design approach is investigated with a walkthrough use of the demonstration system.

2. AN SCRPTING BASED SOLUTION

The scripting based VE modeling system is in an infrastructure which is able to accelerate the procedures for constructing interactive 3D graphics by adopting the modern graphical rendering systems and interpretative computing technology. The script binding of graphic libraries is an important part of this infrastructure. The architecture of a scripting oriented VE modeling system is shown in Fig.1.

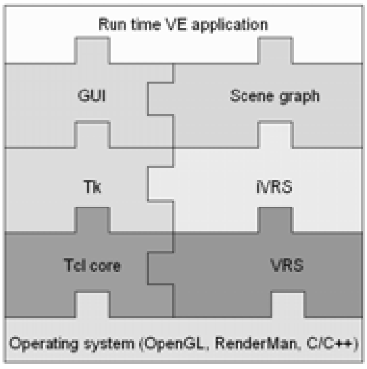


Figure 1. Underlying computing infrastructure

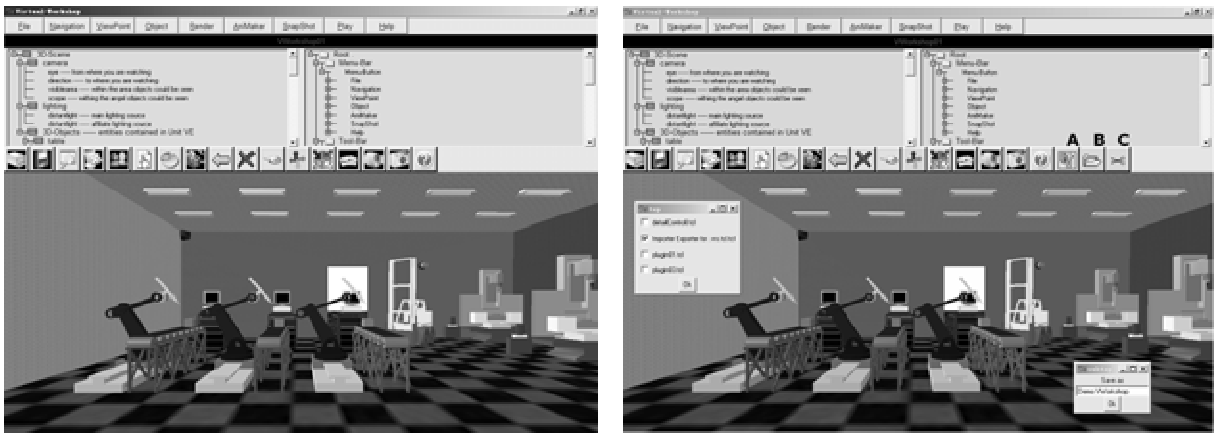
In this system, the graphical library VRS built on top of industrial specification OpenGL provides a collection of rendering components which takes care of real-time data visualization, while user interactivity is designed and implemented though Tcl/Tk. Although Tcl core and VRS are developed by third parties with compiled language, the user interface (UI) system and scene graphs involved in VE software are to be implemented at scripting level. This is because Tk is a scripting-oriented extension package of Tcl core, while iVRS provides VE developers with a full access to 3D features of the VRS, which means VE developers can initiate VRS objects and call their respective methods using Tcl or Tk commands even at application runtime.

Abstracted Graphic Rendering Engine

Although the functionality of low-level graphical application programming interfaces (APIs) covers 3D rendering, scene modeling, interaction handling, 2D imaging and animation, the development of 3D graphics including VEs using, for example, OpenGL, remains expensive since their implementation often requires thousands of lines of code and this fact counteracts the “rapid development of VEs”. The on demand generation of high-level graphical rendering libraries including VRS is based on the widely used and successful scene graph [Fol97a] metaphor, which can be seen as an object-oriented representation of low-level graphics need to be rendered and displayed. The design of high-level graphics rendering engines aims to abstract the complexity of the low-level APIs like OpenGL. This kind of API abstraction was achieved by the design and implementation of a large collection of reusable building blocks, each of which provides users with programmable interfaces for controlling low-level rendering details. Because the scene graph supported by abstracted graphical rendering systems has a full representation of the whole scene, VE developers can easily take advantage of built-in, ready-to-use algorithms and data structures, which a graphic driver can hardly do. As a main part of the proposed infrastructure, VRS is expected to provide VE developers high-level design options and isolate them from underlying graphical rendering details, by which means complex visual representation can be created with less engineering efforts.

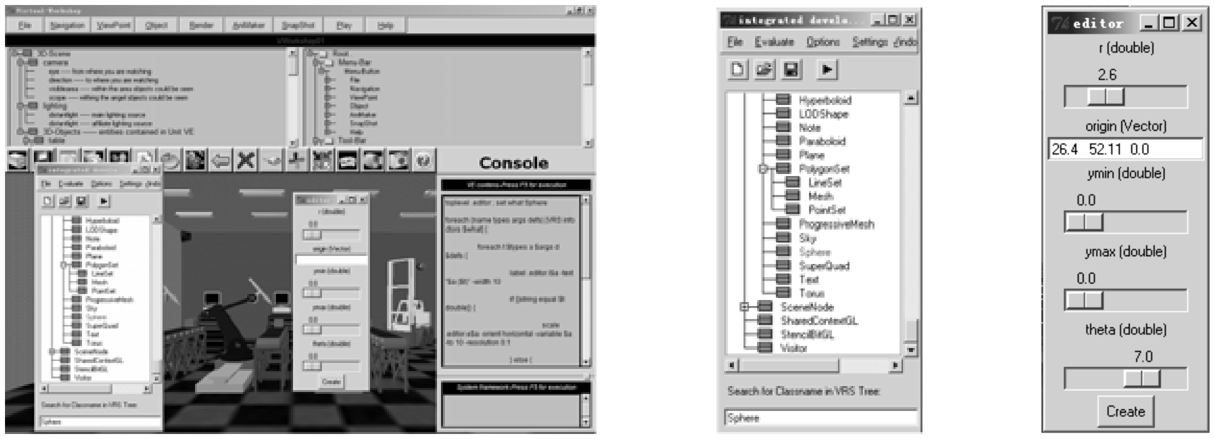
Applied Computing Tool

Our scripting based VE design approach is largely relied on the scripting language which is a typeless language. Unlike system programming languages including C/C++, all scripting variables or data blocks look and behaviour in the same way so that they are interchangeable. Since all scripting variables are changeable, the original code can produce new programme and execute it at runtime. For those VE engineers who can hardly foresee in the initial VE design stage what kind of functionality will be required in the application phase, this novel computing paradigm enable them to implement the VE system in a modular structure, that is, a micro kernel that can load at runtime the function modules of the system encapsulated as modular blocks whether abide the already existing user interface or not. For a typical VE which is driven by Tcl/Tk,



(a) Original system looks-like (b) File Exporter/Importer: a plug-in programme in use

Figure 3. VE system functionality and its enhancement



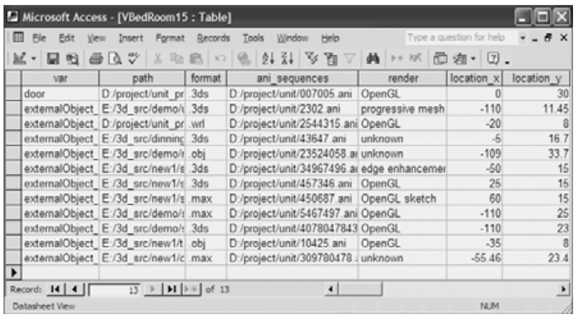
(a) Automatically generated construction for widget (b) API information inquiring system (c) Control widget

Figure 4. Runtime code generation and interpretation using console system

the initialization of default functional modules, for example, spatial navigation and scene management, can be done at the starting of VE, while the rejection, augmentation and override of modular blocks can be achieved at runtime to meet the requirements for specific application or simulation tasks.

VE Database and Its Connectivity

The design of VE database and its management system is intended to provide a top-down administration of VE templates and avoid building new VE completely from scratch. The VE database is set up with Microsoft® Access 2000 and utilizes its advantages including data access control, distributed access, different authoring types concurrency control, retrieval performance and data consistency. The developed VE database, see Fig.2, attempts to formulate a uniform way for abstracting 3D graphic contents and their dynamics by which each property is represented by an entry in the database, describing its format, path, spatial status, rendering details and animation sequences.



Microsoft Access - [VBedRoom15 : Table]

var	path	format	ani_sequences	render	location_x	location_y
door	D:/project/unit_pr	.3ds	D:/project/unit/007005.ani	OpenGL	0	30
externalObject_	E:/3d_src/demo/	.3ds	D:/project/unit/2302.ani	progressive mesh	-110	11.45
externalObject_	D:/project/unit_pr	.wrl	D:/project/unit/2544315.ani	OpenGL	-20	8
externalObject_	E:/3d_src/dinning	.3ds	D:/project/unit/43647.ani	unknown	-5	16.7
externalObject_	E:/3d_src/demo/	.obj	D:/project/unit/23524058.a	unknown	-109	33.7
externalObject_	E:/3d_src/new1/s	.3ds	D:/project/unit/34967496.a	edge enhancemer	-50	15
externalObject_	E:/3d_src/new1/s	.3ds	D:/project/unit/457346.ani	OpenGL	25	15
externalObject_	E:/3d_src/new1/s	.max	D:/project/unit/450687.ani	OpenGL sketch	60	15
externalObject_	E:/3d_src/demo/	.max	D:/project/unit/5467497.ani	OpenGL	-110	25
externalObject_	E:/3d_src/demo/	.3ds	D:/project/unit/4078047843	OpenGL	-110	23
externalObject_	E:/3d_src/new1/t	.obj	D:/project/unit/10425.ani	OpenGL	-35	8
externalObject_	E:/3d_src/new1/c	.max	D:/project/unit/309780478.	unknown	-55.46	23.4

Record: 13 of 13

Datasheet View

Figure 2. A typical VE database

During the visualizing process, the VE database will make meta information available and thus the programme can deploy each constructive model correctively while assembling them into integration. TCLODBC [Tcl04a] which can provide the scripting-oriented database connectivity is employed for runtime data synchronization between VE and database. Once the meaningful graphical representation is created after the starting of VE, developers may apply available multiple reconfiguration utilities including 2D UI, plug-in programmes and console to change VE content, rendering context and control. Any event, for example, relocating selected graphic node with a drag-and-drop metaphor, will lead to consistent redisplay for the entire graphics in the view port. Before sending changed scene graph into the rendering pipeline for the drawing of the next frame, the copy of the meta information of new VE will be hold by certain Tcl variables. By creating new VE record with the information from those Tcl variables, the VE design can be saved as reusable VE templates.

Human Machine Interface

Tk which is the extension package of Tcl is used to implement human computer interactivity. By summarizing frequently referred interaction between human and desktop VE system, the practical human computer interaction scheme has been developed which cover multiple handlers, including 2D UI system, runtime plug-ins and console, for dealing with variant categories of developer requests on VE level tasks. Development of a VE with the script-enabled developing environment can reduce project costs.

VE engineers can perform navigating and selecting actions with the toolbar to communicate with already existing VE. In Fig.3.a, developers can wander through the 3D site in three different navigation modes. The involved graph nodes can be selected, relocated, rescaled, replaced or removed with default functionalities which is an organic part of the demonstrated UI system.

VE developers are enabled to extend current UI system by providing it the functionality enhancement. For example, if current VE needs to be saved in binary format using custom export toolkit which is not available, VE developers can package scripts with certain utility panel and temporarily install the scripts as buttons in the toolbar, as items in menus, or designate them to hotkeys. See Fig.3.b. The updated toolbar integrates custom import (button “B” and “C”) and export (button “A”) toolkits using ASCII and binary file input and output stream.

Given the fact that, for VE engineers, especially those who may have solid background in graphic computing, direct manipulation by editing and trying new command input at system runtime tends to be another advisable approach for both VE development and improvement. The UI system comes with a runtime console which is actually a fully interactive Tcl/Tk interpreter for graphical language and works similar to a DOS command prompt window. This working environment integrates a 2D text editor with which VE developers are able to retrieve and display source files and then comprehend implementation details of specific software components. It also provides a scripting command input console which is *de facto* a built in window listener embedded with an internal scripting interpreter. An inquiring system is already built up that can be used to send queries for and then returns crucial API information for VE developers.

In Fig.4, the command input console is used for automatically instantiating new control widget by taking advantages of collected API information. Fig. 4.b shows the tree view of the scene graph APIs. Developers can quickly explore the hierarchical structure of VRS classes when editing their own code fragments. In this case, required “Sphere” class is located from within the tree structure simply by entering “Sphere” in the entry panel. To display the API information, double-left-click the “Sphere” node in the tree view and constructors of class “Sphere” with their argument types and default values will be displayed. Using this utility, developers can thus design the widget within the console for sphere object defined by radius, cutting planes in *Y* direction and aperture angle. This control widget was automatically initialized and applied instantly for editing and adding experimental contents into existing graphs for rapid prototyping. See Fig.4.c.

3. A VE RAPID PROTOTYPING(VERP) DEMONSTRATION SYSTEM

In order to demonstrate the proposed VE developing infrastructure, basic functional modular libraries have been designed and implemented for a demonstration system. The demonstration system supports fast VE development with a novel systematic VE design approach supported by its underlying computing infrastructure, as is can be seen from Fig.5.

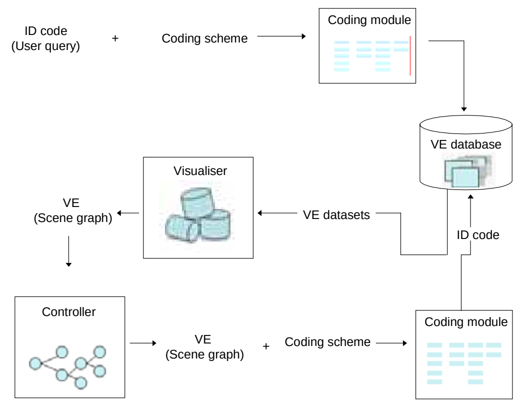


Figure 5. A typical VE database

The main idea behind the illustrated approach is to find out similar VE template from database according to user query. The query is sent to "Coding Module" which is actually a VE coder/decoder. The returned results will be displayed with a suitable presentation which provides a good overview of information relations without an overload of information for VE engineers. VE engineers can thus make decision for selecting and visualising particular VE inside the system according to the degree of relevance between VE templates and their query. During the visualising process, the datasets will be retrieved from VE database and then translated by "Visualiser" at system runtime for 3D scene reconstruction. The "Controller" will make incremental changes to current VE take place according to certain application specification. New VE is to be categorized before saving it in VE database.

VE Coding and Classification

The VE database has been designed to be a hierarchically structured container. The maintained VE library comprises several isolated VE families and sub-VE-families each of whom consists of numbers of VEs which differ from each other in terms of, for example, scene graph composition, physical layout of objects and their functions. Since large numbers of designed VEs will increase the complexity of the entire system and be not convenient for developers to collect specific environment, as well as the time that consumed on seeking for proper or similar visual representation among a pool of discrete candidates will counteract the "rapid" of rapid prototyping, both conceptual and visual coding and classification are employed in order to provide a broad, top-down control to VEs without plunging into the complexity of a fully inordinate system.

During the process of classification, the physical sceneries in the world have been roughly grouped into different categories according to following classification principles: (i) the environments should be typically divided into two categories, that is, outdoor sceneries and internal culture; (ii) the environments are to be classified by both visual appearance and function; (iii) the microcosm will be ignored and only those environments over a certain size, that is, human scale or larger, will be considered for visual ontology.

VE coding is used for establishing symbols according to the classification categories for meaningful communication. A hybrid structure is adopted in coding schemes, that is, the system employs monocode where they can, and apply polycode for other digits in such a way as to obtain a code structure that captures the essential information about a part shape.

Currently the demonstration system realizes a three-levels coding scheme in which the first two levels represent a hierarchical structure consisting of exclusive attributes while the third is a simple chain code composed of discrete, universal properties. Once the VE query is received by the "Coding Module", a similarity coefficient calculation will be automatically done, in which specific relevance value has been designated as the threshold for deciding the range of to be collected environments.

VE Database and Visualiser

Once VE engineers decide which VE template is to be visualised, the database connection is to be set up using TCLODBC. The pseudo code below shows how to open database connection.

```
set driver "Microsoft Access Driver (*.mdb)"
set dbFile "Absolute path of the database file"
set dsn VUNITPRO
database adddsn $driver ["DSN=$dsn" DBQ=$dbfile]
database db $dsn
set table [db select from database where flag=1]
```

Once after the database connection is established, the system will access proprietary conditions of to be visualised VE, such as the spatial information of involved models, their volume scale, rotation axis and angle, material and colouring attributes and so on in order for visualiser to decide the initial status of reconstructed VE. Following pseudo code presents how to relocate a VE model with the meta information delivered by database.

```
set locx [select locX from table where ID=givenID]
set locy [select locY from table where ID=givenID]
set locz [select locZ from table where ID=givenID]
$target locate [new Location $locx $locy $locz]
$canvas postAllForRedisplay
```

After the VE template is built up, VE engineers have to configure the VE from its original state and this reconfiguration process can be carried out repeatedly to get the system into the correct configuration.

VE Controller

UI system designed for VE reconfiguration makes it possible to control a graphical scenery with a Tcl programme and, conversely, to react in Tcl to input (the events due to user interaction) from the scenery. Any event, for example, relocating selected graphic node with a drag-and-drop metaphor will lead to consistent redisplay for the entire graphics in the view port. Before sending changed scene graph into the rendering pipeline for drawing of the next frame, the copy of the meta information of current VE will be updated. Take coordinates transformation for example, VE engineers can change the spatial location of involved VE models by dragging the right mouse button. Once the mouse button is released, the current state of the VE model will be maintained by temporary Tcl variables, see following pseudo code.

```
proc setDynamicDataset {target} {
        global tempX tempY
        set $target::locationX $tempX
        set $target::locationY $tempY
}
```

For each involved VE object, the system automatically generates a unique namespace at the starting of the VE. The name of the VE model is used to define homonymic namespace, there exist a set of global variables under each namespace to maintain the copy of meta information of VEs.

In order for VE engineers to save the VE design, the VE controller is to create a new data table in VE database. Particular dependencies will be defined by which the current state of the VE can be held.

```
db "create table $renderingdata" (
        object          char (50)
        path            string
        location_x      double
        … …
        dynamics        string)
```

After populating a empty data table, the rendering data bits of each VE object are to be saved with this data table.

```
db "insert into $renderingdata" (
        object
        path
        location_x
        … …
        dynamics)
values (
        'object name'
        accessing path
        $objectName::locationX
        … …
        $objectName::dynamics)
```

The data table accepts only one VE model (its rendering data bits) at a time. To save a VE that has *N* objects, above process is to be repeated *N* times. After that, the VE code is to be generated automatically according to the VE coding scheme.

4. FUNCTIONALITIES OF THE DEMONSTRATION SYSTEM: A WALKTHROUGH USE

The demonstration system can be applied by, for instance, upholstery to design the layout of furniture, or construct the real scene with given information. The difference between these two sample tasks is, the later usually has a rigid frame of reference, while the previous allows the artists and VE developers to throw away violation and be free to utilize their creativity. To evaluate the demonstration system, certain real indoor scenery has been created with the reference to a 2D blueprint used as background image. See Fig.6.

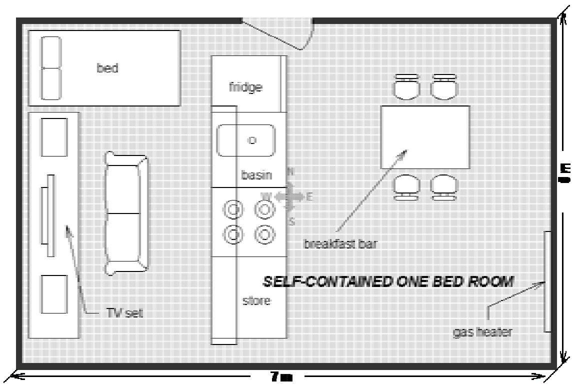
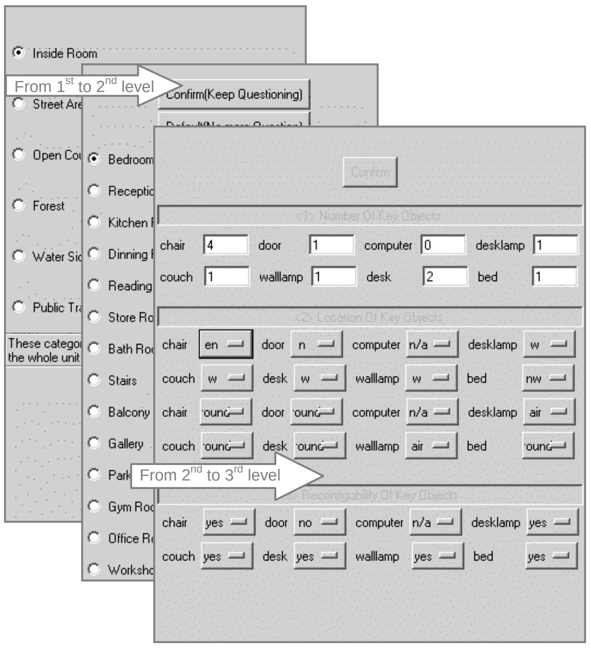


Figure 6. The 2D layout of the site

Based on the information encapsulated in above blueprint, user query for VE templates can be defined. The VE rapid prototyping system adopts 2D interaction metaphor to define the search for VEs, and browse the query results with panel utilities in order for VE engineers to get an impression of which VE template is suitable for reconstruction. Fig.7 shows how users started the search, and then initiated the query by shrinking the search range and inputting required values.



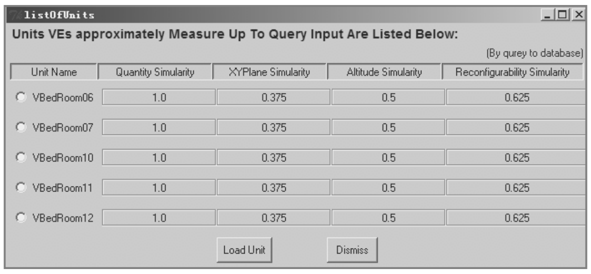
chair	4	door	1	computer	0	desklamp	1
couch	1	walllamp	1	desk	2	bed	1

chair	en	door	n	computer	n/a	desklamp	w
couch	w	desk	w	walllamp	w	bed	nw
chair	ound	door	ound	computer	n/a	desklamp	air
couch	ound	desk	ound	walllamp	air	bed	ound

chair	yes	door	no	computer	n/a	desklamp	yes
couch	yes	desk	yes	walllamp	yes	bed	yes

Figure 7. Define VE query

After calculating the input values by users, the system automatically evaluated the results and displayed the evaluation outcome with the utility panel shown in Fig.8.



Units VEs approximately Measure Up To Query Input Are Listed Below:

(By qurey to database)

Unit Name	Quantity Simularity	XYPlane Simularity	Altitude Simularity	Reconfigurability Simularity
VBedRoom06	1.0	0.375	0.5	0.625
VBedRoom07	1.0	0.375	0.5	0.625
VBedRoom10	1.0	0.375	0.5	0.625
VBedRoom11	1.0	0.375	0.5	0.625
VBedRoom12	1.0	0.375	0.5	0.625

Figure 8. The matching result

Above presentation of query outcome provided a good overview of information relations to help users to recognise which VE matches best regarding all properties; which property is fulfilled best; and whether a VE is determined by the system to be one of the candidates because it matches all properties well, or because it matches one property extremely well. At random, *VBedroom06* was selected to visualise. During the visualising process, the meta information was retrieved from VE database and used to deploy each model correctively while assembling them into integration. The visualisation outcome can be seen from Fig.9.

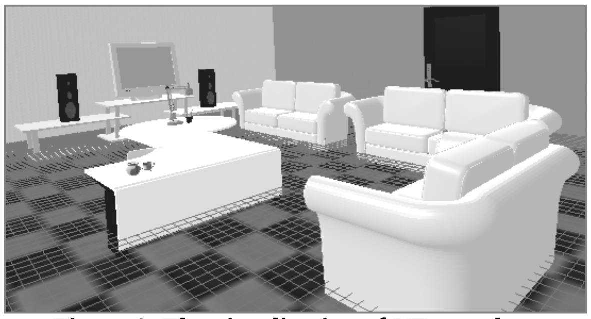
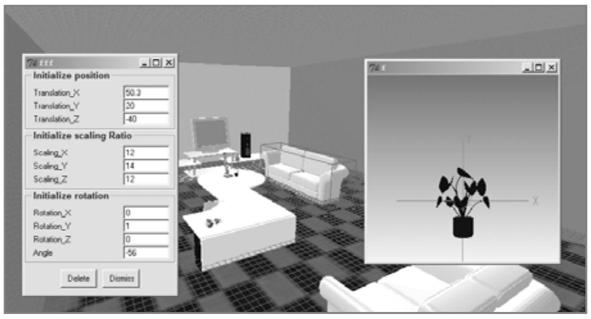


Figure 9. The visualization of VE template

Once the selected VE template is visualised by the system, the revision for progressively approximating to certain blueprint can be done accordingly. Fig.10 illustrates the introduction and relocation of new 3D objects.



Initialize position

Translation_X	50.3
Translation_Y	20
Translation_Z	-40

Initialize scaling Ratio

Scaling_X	12
Scaling_Y	14
Scaling_Z	12

Initialize rotation

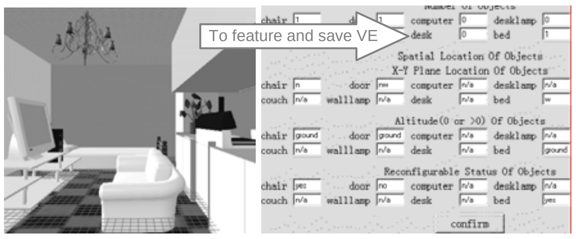
Rotation_X	0
Rotation_Y	1
Rotation_Z	0
Angle	-56

Figure 10. Merge new model into the environment

During the reconstruction process, the console was used to change the distance of the wall to adjust the length-to-width ratio of the room, and consequently the location of the door which should, according to the blueprint (Fig.6), stand at the center of the room but veering more to the left. This was done simply by typing following scripts into the console (Fig.4.a), and then pressed the key "F5" to evaluate the scripts.

```
#move the walls
$leftwall prepend [new Translation 1.0 0.0 0.0]
$rightwall prepend [new Translation -1.0 0.0 0.0]
#relocate the door
$mainscene remove $door
$facetas_frtwal   insertloop   0   $loop_4frtwal_hole
$normal
Set loop_4frtwal_hole [VectorItr
                    {-0.86 0.0 -4.5} {-0.86 0.0 1.5}
                    {-1.2 0.0 1.5}    {-1.2 0.0 -4.5}
$facetas_frtwal   insertloop   1   $loop_4frtwal_hole
$normal
$door prepend [new Translation -1.3 0.0 0.0]
$mainscene append $door
$canvas postAllForRedisplay
```

To save the reconstructed scene as a reusable VE template, users can either feature the VE by filling the entries in the utility panel (Fig.11) to save it as a database file, or activate "File Exporter/Importer", the runtime plug-in (Fig.3.b) to save it as a binary data file.



Number Of Objects

chair	1	d	1	computer	0	desklamp	0
				desk	0	bed	1

Spatial Location Of Objects

X-Y Plane Location Of Objects

chair	n	door	nw	computer	n/a	desklamp	n/a
couch	n/a	walllamp	n/a	desk	n/a	bed	w

Altitude(0 or >0) Of Objects

chair	ground	door	ground	computer	n/a	desklamp	n/a
couch	n/a	walllamp	n/a	desk	n/a	bed	ground

Reconfigurable Status Of Objects

chair	yes	door	no	computer	n/a	desklamp	n/a
couch	n/a	walllamp	n/a	desk	n/a	bed	yes

Figure 11. Describe the VE with given dependencies

Compared with other popular VE modeling applications, with the use of above demonstration system, both time and labour cost were largely saved due to the availability of user interface tools, which were designed for realizing multi-functional and multi-levels control over the whole VE during the system runtime. The VE can be created either from scratch or built up based on background image, while the application itself (the hosting shell) is capable of doing self-modification by, for example, introducing a console for realizing functional extension at runtime which is impossible in a compiled language driven development environment. With the built-in code interpreter, the VE control is enlarged to a limitless scope, that is, each memory piece can be manipulated for achieving each possible design task.

5. CONCLUSION

In our infrastructure, we apply interpretative computing tool and its binding of an abstracted graphic library to design, implement and redevelop VE applications. This infrastructure has been further developed into a VE rapid prototyping system, which benefits from scripting as a fundamental tool for runtime reconfiguration of VE graphics, rendering context and their control, as well as from the concept of "VE template" by which it is not necessary to build the whole VE completely from scratch.

6. REFERENCES

[Döl02a] Döllner, J., and Hinrichs, K. A generic 3D rendering system, IEEE Trans. on Visualization and Computer Graphics, Vol.8, No.2, pp.99-118, 2002.

[Fol97a] Foley, J.D., Andries, V.D., Feiner, S.K., and Hughes, J.F. Computer graphics: principles and practice, 2nd edition in C. Addison-Wesley, 1997.

[Gri96a] Grinstein, G.G., and Southard, D.A. Rapid modeling and design in virtual environments, Presence: Teleoperators and Virtual Environments, Vol.5, No.1, pp.146-158, 1996.

[Oli03a] Oliveria, M., and Crowcroft, J. An innovative design approach to build virtual environment systems, Computer Laboratory, Cambridge University, 2003.

[Ous98a] Ousterhout, J.K. Scripting: higher level programming for the 21st century, IEEE Computer, Vol.31, No.3, pp.23-30, 1998.

[Ran95a] Randy, P. A brief architectural overview of Alice: a rapid prototyping system for virtual reality, IEEE Computer Graphics and Application, pp.8-11, 1995.

[Tcl04a] Tcl open database connectivity, available at http://sourceforge.net/projects.tclodbc.

[Win95a] Wingfield, M.A. MITRE's virtual model shop, in SPIE'95 conf.proc., pp.147-154, 1995.

[Zha98a] Zhao, Z.X. Virtual Reality based robot mission control, Technical White Paper Document, Superscape Virtual Reality Software Ltd., Hampershire, UK/School of Engineering, University of Derby, Derbyshire, UK, 1997.