Cent. Eur. J. Comp. Sci. • 2(3) • 2012 • 261-271
DOI: 10.2478/s13537-012-0016-6

VERSITA

Central European Journal of **Computer Science**

On building an object-oriented parallel virtual reality system

Research Article

Branislav Sobota*, Štefan Korečko†, František Hrozek‡

Department of Computers and Informatics,
Faculty of Electrical Engineering and Informatics,
Technical University of Košice,
Letná 9, 041 20 Košice, Slovakia

Received 10 February 2012; accepted 17 August 2012

Abstract: The paper deals with an issue of a design, development and implementation of a fully immersive virtual reality (VR) system and corresponding virtual worlds, specified in an object–oriented fashion. A virtual world object structure, reflecting a division of VR system into subsystems with respect to affected senses, is introduced. It also discusses virtual worlds building process, utilizing the software development technique of stepwise refinement, and possibilities of parallel processing in VR systems. The final part describes a VR system that has been implemented at the home institution of the authors according to some of the ideas presented here.

Keywords: virtual reality • virtual object • stepwise refinement • parallel computing

1. Introduction

A Virtual reality (VR) system represents an interactive computer system that is able to create an illusion of physical presence in places in an imaginary world or in the real one. It can be also seen as a tool providing a perfect simulation within the environment of tightly coupled human–computer interaction [8]. VR systems are improving to provide more immersive experiences and they are becoming more interactive. On the other hand the complexity of their implementation and their hardware requirements are rising significantly. The VR system can be divided to subsystems with respect to senses affected [8]: Visualization subsystem, Acoustic subsystem, Kinematic (and statokinetic) subsystem, Subsystems of touch and feel and other senses (e.g. smell, taste, sensibility to pheromones, or pain). Of course, not all of these are included in a typical VR system. Some are omitted because the related senses have only limited impact on people's everyday experience. Others have their importance, but we can question the possibility of their implementation in virtual reality. Taste belongs to such senses, so we do not expect a virtual food to come into existence in a nearby future. From

* *E-mail: branislav.sobota@tuke.sk*
† *E-mail: stefan.korecko@tuke.sk (Corresponding author)*
‡ *E-mail: frantisek.hrozek@tuke.sk*

the point of view of VR systems design and implementation, it is necessary to think about some of the above mentioned subsystems, at least about the visualization, acoustic and kinematic and statokinetic.
VR systems provide virtual worlds that consist of a number of objects with defined properties, appearance, relations, behaviour and so on. To specify these worlds various description and scripting languages are used [2, 8, 14]. Description languages, such as *VRML* [23], *Open Inventor* [17, 18] or X3D [24], are similar in their nature to HTML or XML. Scripting languages are used to describe dynamic (behavioural) aspects of the worlds and usually belong to a group of interpreted languages. This means that a script is "executed" directly from a source code in most cases. To boost the performance the scripts are sometimes executed from a pre–compiled code and some VR systems even allow to use compiled languages such as C++ or C#. The interpreted scripts are usually referred to as "soft–coded" [1] to distinguish them from "hard–coded" compiled parts of the VR system itself. Many scripting languages are used for virtual reality nowadays, for example *RUBY* [20], *PYTHON* [19] or *LUA* [16]. Because of the high price of some specialised VR equipment the recently available VR systems, or graphics engines, usually limit themselves to classical input (keyboard, mouse, gamepad, joystick) and output (display) devices and are optimised to run on a single machine. Many of them are supplied with sophisticated world editors with built–in support for scripting (Unity 3 [22]) and some of them are built around existing editors (Blender [15]).
The next two sections of the paper present our ideas on the design, development and implementation of a fully immersive virtual reality system and corresponding virtual worlds. Section 2 focuses on an overall architecture of the system that is composed from subsystems mentioned above and on a process for building virtual worlds. Parallel computing possibilities for the system are discussed in Section 3. Section 4 describes the implementation of a VR system developed at Technical University of Košice (Košice, Slovakia) that obeys some of the principles described in this paper. The paper concludes with a few words about the actual and intended utilization of the implemented system and goals for future research and development.

2. Virtual world building process

A VR system is usually designed to host various virtual worlds. While the appearance and purpose of these worlds may vary significantly, they all have to follow the same strictly defined structural rules, which are, at least partly, given by the used description and scripting languages. The reverse is also true: We can have two virtual worlds similar in appearance and purpose implemented in different VR systems. A definition of virtual world's architecture greatly affects possibilities of the VR system, e.g. how it can be modularised or optimised for parallel processing. In general, the process of virtual world building can be seen as consisting of the following tasks:

1. Definition of virtual world objects (VR objects) and data that will be received from an input or shared among the objects. This task includes determining how the input data will be stored in the objects, what properties are important for individual subsystems of the VR system and what other information the object should contain. We can also specify what operations will be available for the objects or how they will interact.

2. Implementation of the VR objects and operations on them by means of a stepwise refinement. The properties and methods of a VR object can be divided into two groups:

   - Properties (variables, constants) and methods related to sensory information received from the object, i.e. to its representation in the VR system (appearance, sounds, etc.). The process of their creation mostly consists of modelling in appropriate editors rather than programming.
   - Properties and methods defining object behaviour (often referred to as "logic" or "AI"). The methods define what can be done with the object, how it reacts to a given situation and how it interacts with other objects. They can call underlying representation methods.

   Usually it is good to keep these two groups separated, to allow more freedom in a VR system choice.

3. Adjustment for parallel processing.

The third task is not mandatory but when it is required, it should be accomplished together with the first two ones. An issue of interaction level between an observer and the virtual environment is of great importance. From our point of view the most interesting type of VR system is the Dynamic Environment Dynamic Observer (DEDO) as we intend to use a fully immersive VR model. For the sake of simplicity we consider just one observer.

2.1. Specification of data and virtual world structure

Because we deal with fully interactive systems, it is necessary to acquire information about the movement of a human (an observer or his avatar) in the virtual world. These input data are acquired from the kinematic and statokinetic subsystem that contain devices such as position trackers or data gloves. They include

- information concerning the position of all the rotational joints on the human body (position of head, hands, individual finger phalanxes position, position of body, etc.) and
- information about observer view direction.

Some VR objects properties need to be shared. Such properties are acquired from global data of the objects (Fig. 2) and provide information about

- the position or location of individual objects in the virtual world from the point of view of their interaction with other objects or observers, and
- material features of the objects that have to be taken into consideration when dealing with interaction between objects or between an object and the observer. This is primarily connected to the so called feedback, i.e. what sort of sound should be generated for the given object after the interaction with another one, how it should be deformed, etc.

The design of the virtual world representation is a non–trivial problem because it determines the implementation of the VR system as well. We need the best possible implementation providing fast modification, visualization and information retrieval [6, 12]. A possible structure is shown in Fig. 1. The Virtual World object is composed of and described by the regular objects that will appear in the world, the special sensor objects and characteristic frames. Other attributes are also included.

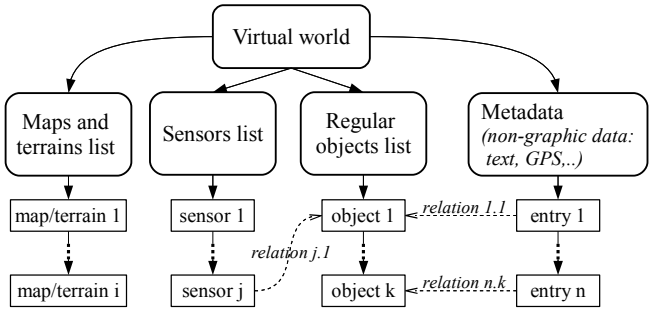

Figure 1. Virtual world elements.

While realisation of the virtual world is an implementation task, some related issues should be considered from the very beginning of its development. Three–dimensional area modelling used in VR systems is not yet a well–researched field [7]. The first question arising is whether to model the whole environment as one complex entity, including all the details, or to create a less concrete model and postpone the details processing to a phase of actualization [9]. Another important question is about the way models are design. There are several alternatives:

- the use of existing concept material,
- the creation of a model from new data and
- a combination of the previous two alternatives.

The problem of the first alternative lies in the questionable actuality of the existing data, which are often outdated and therefore their quality might not meet today's standards. Errors can be included in the representation as well. The second alternative does not suffer from these problems, but it is much more expensive. There are technologies that allow

for the acquisition of a lot of actual data quite quickly, such as laser scanning of large areas, but their price is very high. Because it is also necessary to integrate non–graphical data into the VR system, the combination of the methods is the most effective and feasible option. Besides mentioned scan techniques, terrain can also be modelled using polygons. Surface textures are also applicable for modelling terrain. Another good approach is the use of geodetic surface maps (in the form of height maps where vertical coordinates are represented by different colours), which allows automatic surface model generation. Textures are applicable in this case as well.
Sensors are objects used with collision detection and they are useful in the case of interactive or automated presentations of the virtual reality (VR) scenarios. Events on sensors are of two kinds:

- Stop an actual movement if collision occurs
- Change the movement direction if collision occurs

The use cases of sensor objects are:

- Prevent a camera from moving through VR objects
- Display non–graphical information about the VR object when entering its checked area

The collision detection is executed on the scene objects (regular and sensor objects) and the camera, which is represented as a sphere object. Depending on the kind of the object in collision, the VR system stops the camera and/or displays additional information related to the selected regular or sensor object. In general, we do not have to assign a sensor to each regular object, and the system also might have more types of sensors implemented. The types of the sensors are as follows:

- *Vertical sensor polygon* – used for detection of manual object selection in the case that appropriate tools (e.g. mouse or data gloves) are used.
- *Horizontal sensor polygon* – for camera position detection. The camera status is set to active when collision between the camera and other object occurs.
- *Point-of-view sensor* – the sensor status is active if the object is inside the view frustum and it is the closest one to the camera (observer).

2.2. Virtual world as object–oriented system

The whole virtual world can be defined as a strictly object–oriented system. This means that everything that exists in the world is an object (maps, buildings, sensors etc.). Even the VR system itself can be defined as an object which contains other objects. In terms of such definition it is possible to use the features of inheritance, encapsulation and polymorphism. In Fig. 2 a structure of objects, reflecting the division of properties and methods into two groups and the VR system composition, is shown.
The higher position of the behaviour block is given by a development process, described in the next section. The representation block is divided to frames, according to the basic subsystems of VR system. Each of these frames contains internal information invisible for other frames. Some of the object properties are necessary for each of these frames and for the logic part, so they are shared as global data. A portion of the global data is also accessible by other objects.
Virtual object implementation naturally implies parallel problem solution by means of computer with parallel architecture. However, the problem concerning data structures creation is very extensive. Its optimization is of great importance. It is obvious when we consider that the whole system consists of great number of objects and each of these objects must be defined in terms of its shape and visual features, mechanical features, sound features and time characteristics.

2.3. Development of virtual worlds by means of stepwise refinement

To cope with the extensive problem of designing and implementing the data structures and operations (i.e. classes) for VR objects we propose to adapt the well–known software engineering technique of stepwise refinement, introduced by Niklaus Wirth [13]. The technique should be used in such a way that after preparing an initial specification of

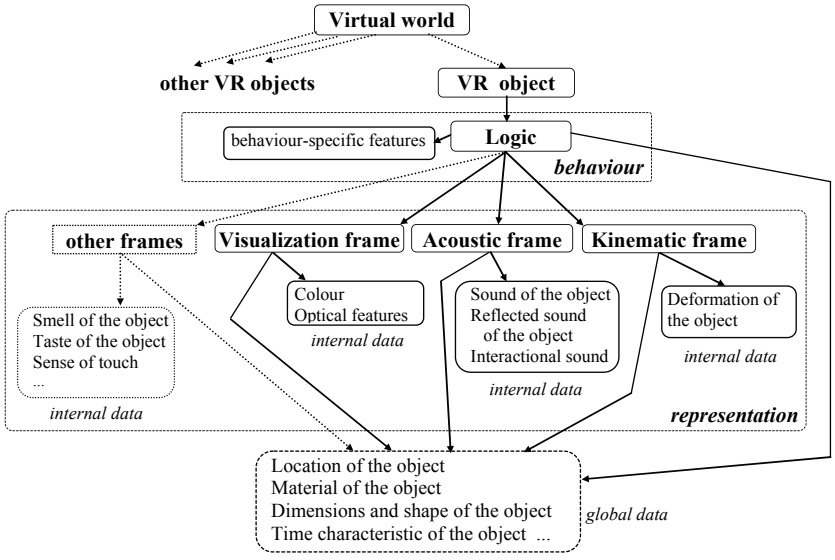

Figure 2. Virtual world object structure.

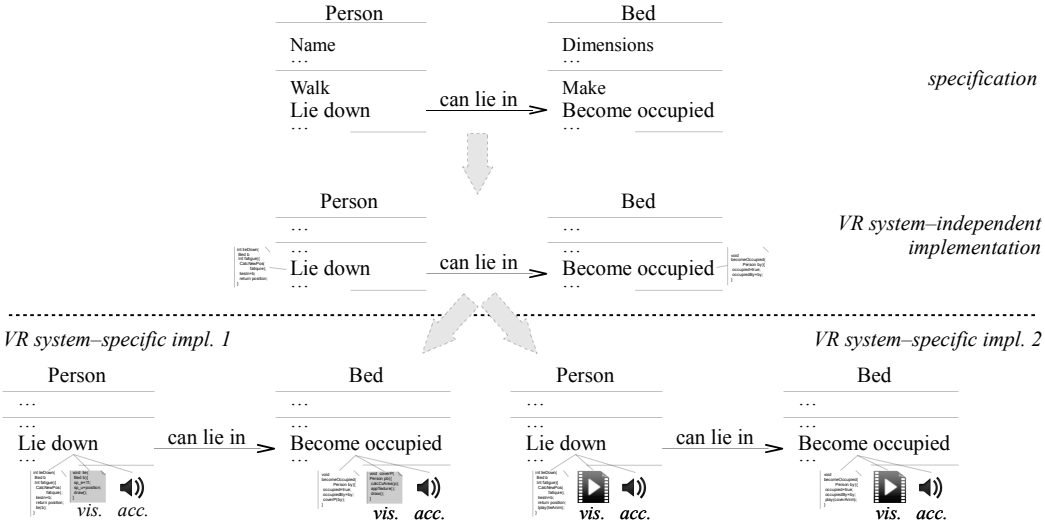

Figure 3. Example of stepwise refinement.

the world, the developers focus on representation–independent behavioural aspects of the objects. When the world's behaviour is implemented suitably, properties, methods and other resources for the individual frames of the VR system are added. Of course, the process is not always straightforward and sometimes facts revealed in the later phases require a re–implementation of parts already done.

The whole idea is illustrated in Fig. 3, which depicts a three–step refinement of objects Person and Bed. Both objects may have many properties and methods, but here we deal only with one of relations between them, the one described by an association "A person can lie down in bed". In the first step an abstract specification is created. The specification can have a diagrammatical form (e.g. UML diagrams) and should name all the important properties and methods. In our objects the relation "can lie in" materializes into methods Lie down and Become occupied. Some description of the properties and methods is also required. It can be informal, written in a natural language or formal, written

in a description language with fixed semantics. The second option is preferred as it allows some automation of the refinement process and verification of the specification against requirements. We can also define a set of operators for basic operations on the objects. For example, we can have an operator "⋈" for merging of objects and define what happens when a person lies down in bed by an expression "$person \bowtie bed = occupiedBed$". The specification is refined to a VR system–independent implementation in the second step. Here concrete data types are associated to the object variables and constants and methods contain code defining the corresponding behaviour. The code of Lie down method can, for example, define that the pulse of the person is decreased, his position changed to horizontal and there is a higher probability to fall asleep. The code is represented by white sheets of paper in Fig. 3. For debugging purposes the objects can also have some visual representation.
Finally, in the third step a specific implementation for a given VR system is developed. As it was already mentioned, this includes many "non–coding" tasks such as creation of 3D models of the objects or recording sounds for them. The code and resources created here will be used as a representation of the object and its behaviour by the subsystems of the VR system. They should be kept separated from the VR system–independent parts to allow an easy change of the system itself. The independent parts are also extended in this step – at least it is needed to define how they will communicate with the newly added ones. We can have several specific implementations for a single independent one. They can differ in the VR system used or in the way how some processes are implemented. This can be also seen in Fig. 3, where we have two implementations. Both of them use recorded sounds to represent a process of lying down in the acoustic subsystem (*acc.*). But they differ in the implementation for the visualisation subsystem (*vis.*).

- The first implementation computes an exact move of the human body for the Person object according to its properties and corresponding equations. Similarly a deformation and how a blanket will cover the lying person is computed for the Bed object. The code of these computations is represented by gray sheets of paper in Fig. 3.
- The second implementation uses an animation prepared before to represent the process visually.

The first implementation will provide more convincing visualisation of the process and the second will be less resource–demanding. We can also combine these approaches, for example in the form of selecting from several prepared animations on the basis of a simple computation. Implementations can also differ in complexity of computations or used resources.

3. VR objects parallel processing

To fulfil the aforementioned requirement for the fast implementation a parallel processing of virtual worlds has to be utilised. The levels of parallelism can be as follows [8]:

- parallelism on the level of virtual realities – parallel processing of more virtual worlds,
- parallelism on the level of subsystems – parallel processing of individual VR system subsystems such as visualization, kinematic and acoustic,
- parallelism on the objects level – parallel processing of individual VR world objects,
- frames level parallelism – parallel processing of individual object frames,
- algorithms level parallelism – e.g. parallel computing in individual frames.

Thus, according to these levels we can graduate granularity of parallelism, from coarse–grained (VR realities level) to fine–grained (algorithms level).
Parallelization on the virtual realities level and subsystems level is trivial. The objects level parallelism is achievable thanks to the fact that objects in VR system are relatively independent. So each object can be processed in an independent branch. The graph in Fig. 4 shows one of possible parallel solutions. In this highly parallel system it is possible to make the initialisation of VR world with N objects and processing of the individual VR objects parallel. Parallelization can be also made on the basis of other "objects" that occur in the VR system, for example collisions. This is illustrated in Fig. 5 where each of M collisions is processed in parallel.
For parallelism on the frames level it is obvious that the possibility of its parallel implementation relies on properties of individual frames. Nevertheless, some level of parallelism is available for each of them. An example of using parallel processing for the kinematic frame is:

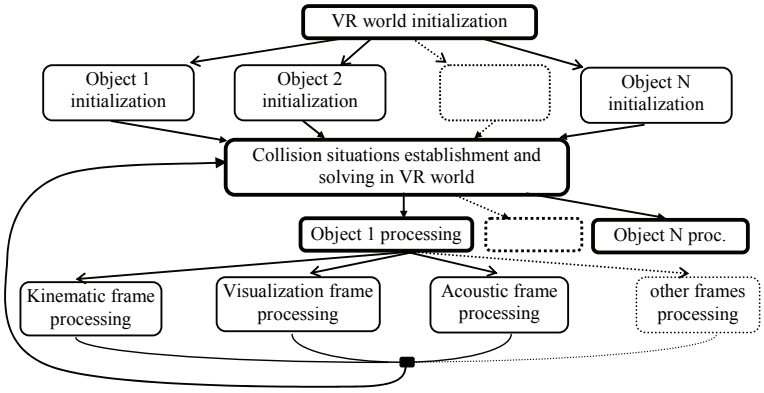

Figure 4. Parallel implementation of virtual objects and their frames.

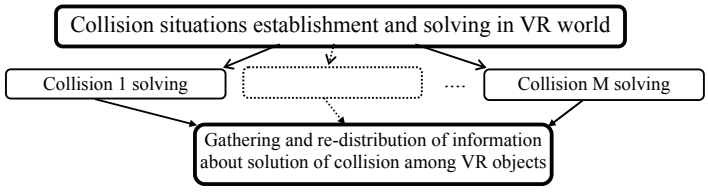

Figure 5. Parallel computing of "Collision situations establishment and solving in VR world" block.

- object information input (position, time, etc.),
- parallel computing of new object position (parallel matrix computing) and
- writing of new parameters into internal or global variables of the object.

The whole virtual world processing in a VR system that utilizes object and frames level parallelism is shown in Fig. 6. It is difficult to deal with the algorithms level in general as individual algorithms differ significantly.

4. Parallel VR system implementation

The virtual world structure and parallel processing described in the previous sections have been partially implemented in the VR system developed at the home institution of the authors. The system uses up-to-date VR hardware, such as a large–screen stereoscopic rear projection system, 42" Philips WOWvx autostereoscopic 3D display, Intersense IS900 ultrasonic tracking system, nVisor ST data helmet with see–through displays and DG5 VHand 2.0 data gloves. The software of the system runs on a computer cluster where the subsystem level parallelism and a parallelism within subsystems is utilized: one node serves as a master that controls the whole computation. It also serves as the acoustic subsystem. Two nodes (computers) are used for the visualization subsystem; each computes one part of the stereoscopic image. The fourth node runs the kinematic subsystem, which processes data from the tracking system. The kinematic subsystem is able to process data from 7 independent tracking inputs in parallel.

The most advanced and complex part of the VR system is its visualization subsystem, which we would like to present in more detail. The subsystem works in real–time and is based on the OpenSceneGraph [4, 5] 3D graphics toolkit. 3D scenes are represented using a scene graph data structure provided by the toolkit. Python and Ruby interpreters have been adapted for the system [11] to allow soft–coding (scripting). The system supports two stereoscopic visualization methods, the first one computes left and right image for passive stereoscopy on the rear projection system and the second one produces 3D images in the 2D-plus-depth format for Philips WOWvx display. To maintain a constant frame rate

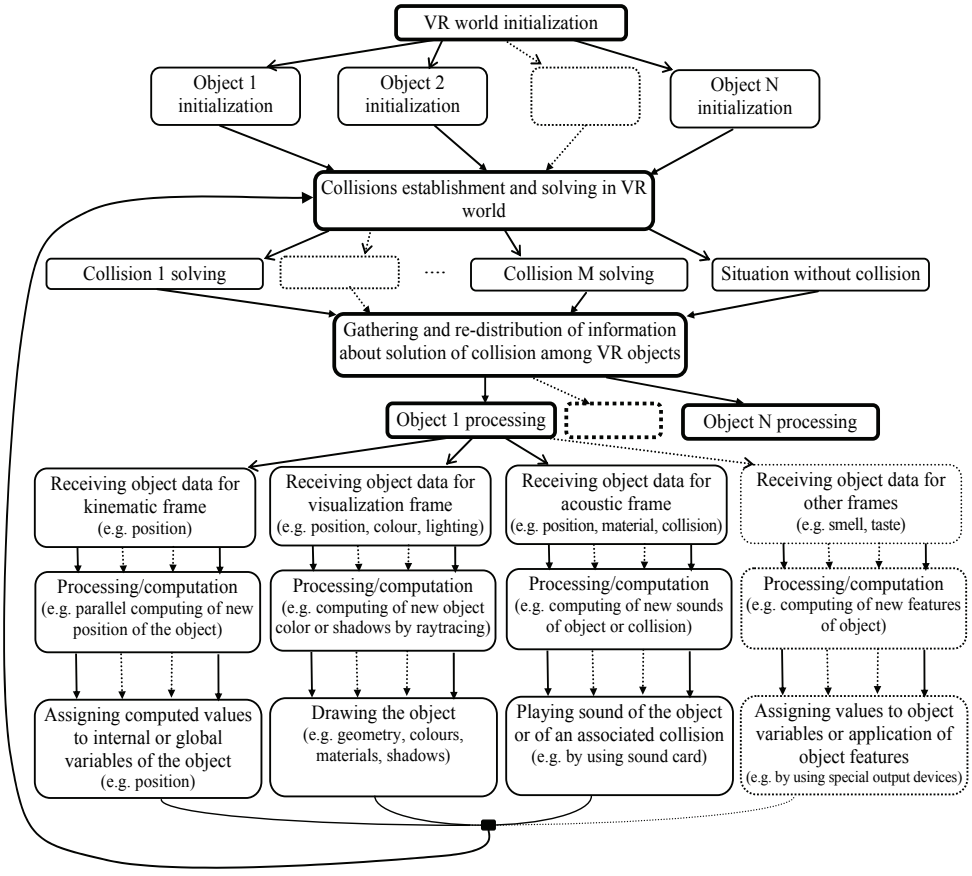

Figure 6. Global structure of parallel computing in VR system.

the subsystem also implements some optimization techniques, such as an omission of invisible objects from rendering. It can render 2 times 3 million triangles and complexity of models can be up to 10 million points in 3D space. For a better management of the virtual objects in the scene, references to the objects are stored in a dedicated array and unique identification numbers are used to recognize objects needed. All other subsystems can get the desired object just by calling the proper getter method of the visualizing subsystem with the object identification number as a parameter.

A scene saving and loading subsystem is also an essential part of the subsystem. Several methods can be used to save the virtual scene to disk. The first and the simplest one is saving the scene as one large three dimensional graphic model directly from some editing software such as Google SketchUp [21] or Blender [15]. The second method is to use an internal binary structure to hold the scene information data. Loading and saving the scene in its own binary format is probably the fastest method. The text formats, like XML can also be used. In our implementation a text format has been chosen to save the scene data. To be more precise, the scripting language of the VR system is used as the text format that holds the virtual scene data. With this method, the user can save and load the scene from the disk and also edit the saved scene and insert some dynamic parts into it [11]. These parts include animating object movement, animating camera movement, playing sounds, music, setting a skybox and so on.

To increase the feel of being a part of the virtual scene, moving objects can be added. There can be cars moving around or airplanes flying above the terrain. In our system an object movement module is soft–coded in Python and provides two implementations that differ in the way how the object movement trajectory is computed. In the first method the trajectory (path) is defined by a set of fixed points, in the second it is computed by some known algorithm, such as Bezier. The path is then used to compute the object position for each picture frame. Code illustrating a use of the first method for a car movement visualisation is

Figure 7. Example: animated car on road.

```
timeForCar = 20; height = 8.0; s = 3.0;
car = gis.CreateModel( ”data/models/offroadcar.osg” );
gis.SetModelScale( car, s, s, s );
road = gis.CreateAnimPath();
## road (path) construction
  gis.AddAnimPathPoint(road,0.0*timeForCar,5067.496,height,-194.935,1.57,0.0, 0.0,s,s,s);
  gis.AddAnimPathPoint(road,0.4*timeForCar,4238.538,height,-107.304,1.57,0.0, 0.0,s,s,s);
  gis.AddAnimPathPoint(road,1.0*timeForCar,166.176,height,-139.565,1.57,0.0, 0.0,s,s,s);
 ## road (path) assign to the model
gis.SetModelAnimPath(car,road);
```

and a code for the second method is:

```
s = 3.0;
car = gis.CreateModel( ”data/models/offroadcar.osg” );
gis.SetModelScale( car, s, s, s );
## road (path) properties
numPaths = 100; duration = 30; radius = 2000;
cl = gis.CreateAnimPath();
## road (path) construction
for i in range(0, numPaths ):
fac = float(i)/float(numPaths);
    gis.AddAnimPathPoint(cl,fac*duration,(math.sin(fac*3.14159*2.0)*radius)
    +1988,400.0,(math.cos(fac*3.14159*2.0)*radius)-2431,(fac*3.14159*2.0)
    +1.57, 0.0, -1.57, s, s, s);
## road (path) assign to the model
gis.SetModelAnimPath( car, cl );
```

Visualisation results are shown in Fig. 7 and in Fig. 8 a part of our VR setting with the large–screen projection system and a user wearing the data gloves and nVisor ST helmet can be seen.

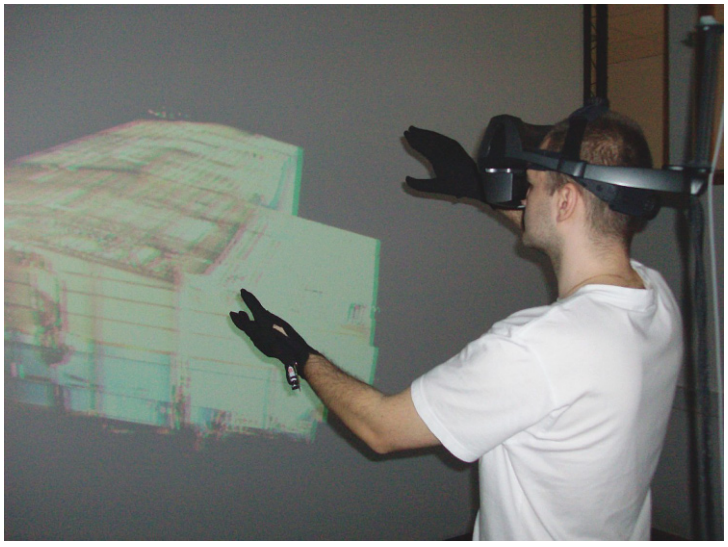

Figure 8. User working with VR system through interaction with virtual world (DCI FEEI TU Košice).

5. Conclusion

In this paper we introduced a virtual worlds building process that adapts stepwise refinement development, and a virtual world object structure that reflects a division of a VR system into subsystems. We also discussed parallel computing possibilities in the system and presented the VR system that partially implements the described structure and parallel processing strategies. This VR system is still under development (fourth generation) and primarily provides an immersive stereoscopic visualization of complex virtual scenes and worlds. Some of the already implemented worlds are a part of the Košice city and several historical buildings from other locations such as Spiš castle [10]. In addition the visualisation the system also provides some information about the objects represented, so it can be used as a fully immersive interactive information system working in real time. The system will also be used for interactive presentations of various products and projects, for example from the areas of engineering and construction. We also intend to use the system for simulation and visualisation of devices that are not available in their real form but are important for teaching purposes, especially in some specific subjects such as formal methods. Another rather exotic use of the system is as a training environment for robots. The system can be compared with other VR systems, most notably with VRECKO [3], that uses the same basic technology (OpenSceneGraph), and has some similar goals but utilizes different hardware and thus, implements different algorithms.

High hardware requirements of the stereoscopic visualisation were the primary motivation for the parallelisation. In the current version of the system the parallel processing is mostly implemented at the subsystem level. Further parallelisation, including an utilisation of the GPGPU technology, is one of the primary goals for the future development of the system. Another important goal is an implementation of a support for the stepwise refinement development process. This can lead to an extension of some of existing VR scene formats, such as VRML or X3D, that will include more dynamic properties. The process can be also helpful for the games design and development where it can simplify the development of multiplatform games.

Acknowledgment

This work has been supported by KEGA grant project No. 050TUKE-4/2012: "Application of Virtual Reality Technologies in Teaching Formal Methods".

References

[1] Bartle R.A., Designing Virtual Worlds, New Riders Pub., 768, 2003
[2] Dietrich A., Gobbetti E., Yoon S.-E., Massive-Model Rendering Techniques: A Tutorial, IEEE Comput. Graph., 27, 20-34, 2007
[3] Flasar J., Pokluda L., Ošlejšek R., Kolčárek P., Sochor J., VRECKO: Virtual Reality Framework, In: Theory and Practice of Computer Graphics 2005 (Canterbury, United Kingdom), Eurographics Association, 203-208, 2005
[4] Kuehne B., Martz P., OpenSceneGraph Reference Manual ver. 2.2, In: Skew Matrix Software and Blue Newt Software, 2007
[5] Martz P., OpenSceneGraph Quick Start Guide, In: Skew Matrix Software, 2007
[6] Sobota, B., Straka, M., Sobotová, D., 3D interface of an information system, In: Informatics and information technology (I&IT'04) (Banská Bystrica, Slovakia), 52-56, 2004 (in Slovak)
[7] Sobota B., Straka M., Hlinka F., Perháč J., Parallel processing of visualization of 3D virtual map project, In: Modelling and Simulation in Management, Informatics and Control (MOSMIC 2007) (Žilina, Slovakia), 9-14, 2007
[8] Sobota, B., Perháč, J., Straka, M., Szabó, Cs., Applications of parallel, distributed and network computer systems for solving of computational processes in an area of large graphical data volumes processing, elfa Košice, Slovakia, 2009 (in Slovak)
[9] Sobota, B., Perháč, J., Petz, I., Surface modeling in 3D city information system, J Comp. Sci. Contr. Syst., 2, 53-56, 2009
[10] Sobota B., Korečko Š., Perháč J., 3D Modeling and Visualization of Historic Buildings as Cultural Heritage Preservation, In: Tenth International Conference on Informatics, Informatics 2009 (Herľany, Slovakia), 94-98, 2009
[11] Sobota B., Control of Large Graphics Data Set Visualization Using Script Language, Acta. Electron. Inform., 11, 33-36, 2011
[12] Vokorokos L., Perháč J., Kleinová A., Parallel Computer System Utilization in Data Visualization, In: Informatics' 2007 (Bratislava, Slovakia), 89-92, 2007
[13] Wirth N., Program Development by Stepwise Refinement, Communications of the ACM, 14, 221-227, 1971
[14] Yangl X., Petriu D.C., Whalen T.E., Petriu E.M., Script Language for Avatar Animation in 3D Virtual Environments, In: VECIMS 2003 - International Symposium on Virtual Environments, Human Computer Interfaces, and Measurement Systems (Lugana, Switzerland), 101-106, 2003
[15] Blender - official website, url: http://www.blender.org/ [cited May 2012]
[16] LUA - the programming language - official website, url: http://www.lua.org/ [cited May 2012]
[17] OpenInventor VSG impl. - official website, url: http://www.vsg3d.com/open-inventor/sdk [cited May 2012]
[18] OpenInventor SGI impl. - official website, url: http://oss.sgi.com/projects/inventor/ [cited May 2012]
[19] Python Programming Language - official website, url: http://www.python.org/ [cited May 2012]
[20] Ruby official website, url: http://www.ruby-lang.org/en/ [cited May 2012]
[21] SketchUp home page. url: www.sketchup.com [cited May 2012]
[22] UNITY 3 - official website, url: http://unity3d.com/ [cited May 2012]
[23] VRML - Virtual Reality Modeling Language - http://www.w3.org/MarkUp/VRML/ [cited May 2012]
[24] X3D - official website, url: http://www.web3d.org/x3d/ [cited May 2012]