CONTROL OF LARGE GRAPHICS DATA SET VISUALIZATION USING SCRIPT LANGUAGE

Branislav SOBOTA
Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic, tel.: +421 55 602 2563, e-mail: branislav.sobota@tuke.sk

ABSTRACT

This paper describes certain area of scripting languages application in visualization process. The focus is on 3D geometric transformations. Algorithm of 3D object rotation is present in more detail and resulting real time implementation in Ruby is demonstrated. Integration into existing VR system for immersive visualization of complex datasets is explained. The object movement from start to destination position is implemented as sequence of sub-movements with particular increments of path. Visualization engine combines script and object to final result and the output is a scene with rotating object in real-time.

Keywords: *computer graphics, geometric transformations, script language, visualization*

1. INTRODUCTION

During recent years scripting languages gained importance in areas of computer graphics and other IT systems. [1][2][3][4]. Scripting languages usually belong into group of interpreted languages. Main difference to compiled type of languages such as C++, C# or others is that script is mostly executed from source code. Sometimes scripting languages are executed from pre-compiled code. This compiled code is called byte-code. Script is compiled to byte-code by interpreter. From history point of view script language was developed in 1960s. Since that year script continued to improve and expand into many programs and forms of scripting languages. The most widespread script languages found application within computer graphics. As of today there are many kinds of scripting languages as for example *RUBY* [5], *PYTHON* [6] or *LUA* [7]. It is possible to program or control computer graphics or virtual-reality applications on higher level using scripting language. Following chapters of this paper focus on *RUBY* scripting language and implementation of real-time geometric transformations using this language.

2. GEOMETRIC TRANSFORMATIONS

The standard set of linear geometric transformations consists of translation, scaling and rotation [8]. General geometric transformation of object *O* to destination object *O'* in a space (virtual world) is depicted in Fig. 1.

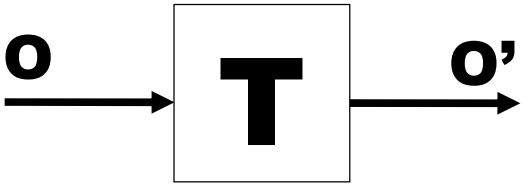

Fig. 1 General geometric transformation

Scaling (by application of matrix **S** (2)) and translation (via matrix **T** (1)) represent simple transformations. A rotation is more complex transformation. A rotation is a movement of object in a circular motion. In 2D space object rotates around a center of rotation. Situation in 3D space is a bit more complex (see Fig. 2). 3D object rotates around a line (e.g. *p*). Simpler case is when line is one of coordinate system's axis. Then it is possible to use one of transformation matrices directly ($\mathbf{R_X}$, $\mathbf{R_Y}$ or $\mathbf{R_Z}$ depend on which one is aligned with rotation axis, (1)). However general 3D rotation requires multiple steps to be performed. First is to move (translate) the virtual world (scene, space) to align line *p* through the origin. Next step aligns line p with one of coordinate system axis (using Euler's angles). Then rotation around appropriate axis is applied. Last step represents all inverse alignment transformations applied to get rotated object back into its original position.

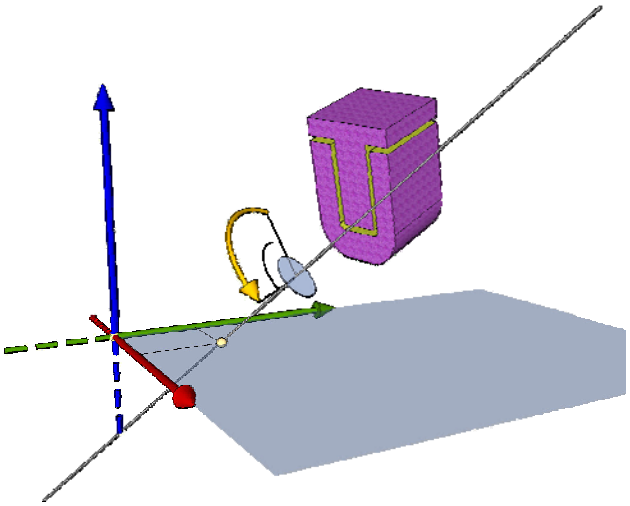

Fig. 2 Object rotation in 3D space

$$\mathbf{R_X}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\alpha & -\sin\alpha & 0\\ 0 & \sin\alpha & \cos\alpha & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\quad \mathbf{R_Y}=\begin{bmatrix}\cos\beta & 0 & \sin\beta & 0\\ 0 & 1 & 0 & 0\\ -\sin\beta & 0 & \cos\beta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\quad (1)$$

$$\mathbf{R_Z}=\begin{bmatrix}\cos\gamma & -\sin\gamma & 0 & 0\\ \sin\gamma & \cos\gamma & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\quad \mathbf{T}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ p_X & p_Y & p_Z & 1\end{bmatrix}$$

Unauthenticated | 194.138.39.60
Download Date | 11/14/13 3:11 PM
VERSITA EMERGING SCIENCE PUBLISHERS

Generic geometric transformation is then performed as follows. First multiplication between base translation matrix **T** and rotation matrix **R** is performed (*X*-axes matrix $\mathbf{R_X}$ is multiplied by *Y*-axes $\mathbf{R_Y}$ and by *Z*–axes $\mathbf{R_Z}$).

Once we have all four matrices multiplied we get transformation matrix as result. If we now multiply vector that defines point in space by such transformation matrix we get new transformed point coordinates as result. In principle we get new position. When performing matrix multiplications it is important to keep order from left to right. Also it is important to first perform translation and only after that rotation of point. Scaling is needed to have complete set of geometric transformations. Scaling is the process to enlarge or reduce object. Transformation matrix that we got in previous steps (that allows us to perform translation and rotation) needs to be then multiplied by scaling matrix

$$S = \begin{bmatrix} S_X & 0 & 0 & 0 \\ 0 & S_Y & 0 & 0 \\ 0 & 0 & S_Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where S_X, S_Y, S_Z are factors of scaling.

These matrixes can be also transformed to some formulas. Full transformation is called sometimes transformation of world (**W** transformation matrix). In simple formula it is defined as:

$$\mathbf{W}(world) = \mathbf{T} \times \mathbf{R} \times \mathbf{S} \tag{3}$$

After substitutions

$$\mathbf{W} = \begin{bmatrix} s_X \cdot \cos\beta \cdot \cos\gamma & -s_Y \cdot \cos\beta \cdot \sin\gamma & s_Z \cdot \sin\beta & 0 \\ s_X \cdot \sin\alpha \cdot \sin\beta \cdot \cos\gamma + s_X \cdot \cos\alpha \cdot \sin\gamma & s_Y \cdot \cos\alpha \cdot \cos\gamma - s_Y \cdot \sin\alpha \cdot \sin\beta \cdot \sin\gamma & -s_Z \cdot \sin\alpha \cdot \cos\beta & 0 \\ s_X \cdot \sin\alpha \cdot \sin\gamma - s_X \cdot \cos\alpha \cdot \sin\beta \cdot \cos\gamma & s_Y \cdot \cos\alpha \cdot \sin\beta \cdot \sin\gamma + s_Y \cdot \sin\alpha \cdot \cos\gamma & s_Z \cdot \cos\alpha \cdot \cos\beta & 0 \\ p_X & p_Y & p_Z & 1 \end{bmatrix}$$

This matrix is used for one-step basic transformation of each object in virtual world.

3. VISUALIZATION ENGINE AND RUBY

As we had already mentioned *RUBY* belongs into family of script languages [5]. Its name is derived from ruby crystal. *RUBY* is programming language with focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write. History of RUBY is dated to year 1993. In that year Yukihiro Matsumoto present this simple scripting language. Interpreter of *RUBY* is written in C language.

RUBY script described in this paper is implemented as part of visualization engine (transformation module) developed at author's department [9][10][14]. This visualization engine represents execution environment used especially for visualization scripts programmed in *RUBY* language. Visualization engine uses objects exported from modelling programs e.g. *Sketchup* [11] (*Sketchup* supported *RUBY* scripting language too). Objects exported from *Sketchup* are exported in *.obj* file extension and must be converted into *.bin* file extensions by convertor (Visualization engine doesn't support *.obj* file extension, only *.bin* file extension yet). Visualization engine then renders these objects and RUBY script is used to perform task. Script provides transformations of objects in scene including camera. Visualization engine consists of two parts. First part of engine consists of libraries, batch files and control center. Second part contains renderer window where scene is rendered. The objects are here and they move as in movie such as people walk and cars are move on the road. Here are two ways to move objects in scene. First simple one moves camera around objects and second more difficult one uses script to script objects and their transformations.

Visualization kernel is implemented to perform real-time visualisation in visualization engine. This kernel is based on sequential approach to process information. On the input side is virtual world (space, objects) and on output side there is rendered image on the screen. Between these two stages there are multiple stages that process and transform input information to final image. The main design idea of graphical system implementation is to minimize the time used by each stage for real time system. In multiprocessor systems it is possible to execute certain stages in parallel, which is real speedup [10]. The rendering pipeline of visualization kernel is divided into three stages:

1. *virtual world model traversal*,
2. *polygon processing* and
3. *pixel processing* (rendering).

World model traversal is typically done on a host CPU and there is a problem if the traversal is unable to keep graphics pipeline full. Polygon processing including vertex transformations and lighting is sometimes done on host CPU, but sometimes by more specialized transformation engines. Polygon performance is often measured to a simple number of triangles per second. Pixel processing including depth buffer testing, antialiasing, transparency blending and texturing typically involves intensive memory access. Pixel performance is often measured in millions of pixels per second. The rendering optimization tasks can be divided into two categories, those which can be done as a *preprocessing stage* and those which must be done at *run-time* because of dynamic changes in world model. Real-time visualisation approach values time aspect above rendering quality. The block diagram of implemented visualization kernel is depicted in Fig. 3.

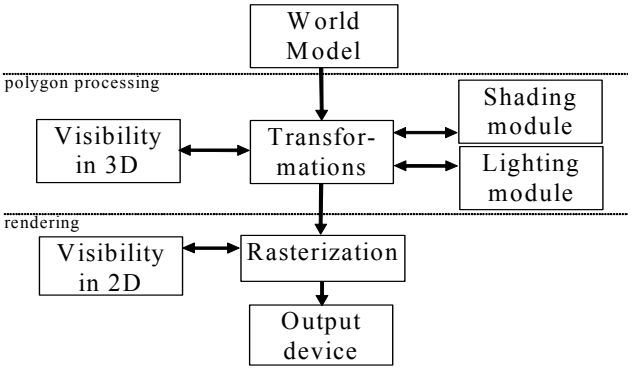

Fig. 3 Block diagram of visualization kernel

Unauthenticated | 194.138.39.60
Download Date | 11/3/14 3:11 PM
VERSITA EMERGING SCIENCE PUBLISHERS

4. TRANSFORMATION IMPLEMENTATION

Geometric transformations described in chapter 2 are used commonly in computer graphics systems. Their implementation is easy and lot of graphics engines/libraries provides these transformations as internal functions. When these functions are applied to virtual objects in graphics systems then the result is displayed in real time (using hardware acceleration). But only final result is displayed and not whole process of transformation. In first step user can to see initial state of virtual world and in next step (next visualization frame) he can to see destination state without dynamics (e.g. only frame 1 and 6 in Fig. 5).

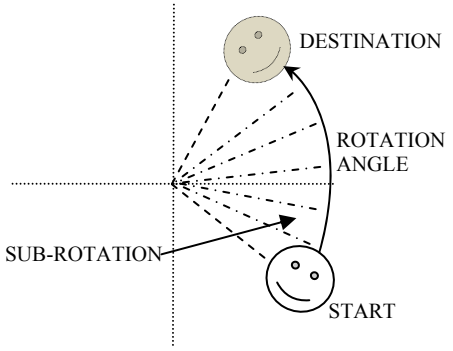

Fig. 4 Splitting of rotation to sub-rotation

The implementation of dynamics is nontrivial process. The minimal implementation supports uniform linear or rotating movement. The movement from start to destination position is implemented as sequence of sub-movements with particular increments (steps) of path (see Fig. 4). The increment size depends on desired movement speed and rendering performance (possible frame per second, respectively *FRAMETIME* in following code). The basic implementation of rotation transformation as a uniform rotating movement is depicted in following code.

```
Rotation(object,x,y,z,speed=1)
   pdx = (object.ox - x).abs;
   pdy = (object.oy - y).abs;
   pdz = (object.oz - z).abs;
   if(pdx<=0.001 && pdy<=0.001 && pdz<=0.001) then return true end

   dirx = 1.0; diry = 1.0; dirz = 1.0;
   dx = x - object.ox;  dy = y - object.oy;  dz = z - object.oz;

   if(dx<0.0) then dirx = -1.0 end
   if(dy<0.0) then diry = -1.0 end
   if(dz<0.0) then dirz = -1.0 end

  deltax = speed * dirx * $FRAMETIME;
  deltay = speed * diry * $FRAMETIME;
  deltaz = speed * dirz * $FRAMETIME;

  max = dx.abs;
  if(dy.abs > max) then max = dy.abs end
  if(dz.abs > max) then max = dz.abs end

  if(max == dx.abs) then
              deltay = deltay * (dy.abs/dx.abs);
              deltaz = deltaz * (dz.abs/dx.abs);
  elsif(max == dy.abs) then
              deltax = deltax * (dx.abs/dy.abs);
              deltaz = deltaz * (dz.abs/dy.abs);
   elsif(max == dz.abs) then
              deltax = deltax * (dx.abs/dz.abs);
              deltay = deltay * (dy.abs/dz.abs);
   end
   object.ox = object.ox + deltax;
   object.oy = object.oy + deltay;
   object.oz = object.oz + deltaz;

   if(dirx<0.0) then
        if(object.ox <= x) then object.ox = x end
   else
        if(object.ox >= x) then object.ox = x end
   end

   if(diry<0.0) then
        if(object.oy <= y) then object.oy = y end
   else
        if(object.oy >= y) then object.oy = y end
   end

   if(dirz<0.0) then
        if(object.oz <= z) then object.oz = z end
   else
        if(object.oz >= z) then object.oz = z end
   end

   pdx = (object.ox - x).abs;
   pdy = (object.oy - y).abs;
   pdz = (object.oz - z).abs;

   if(pdx<=0.001 && pdy<=0.001 && pdz<=0.001) then return true
     else return false end
end
```

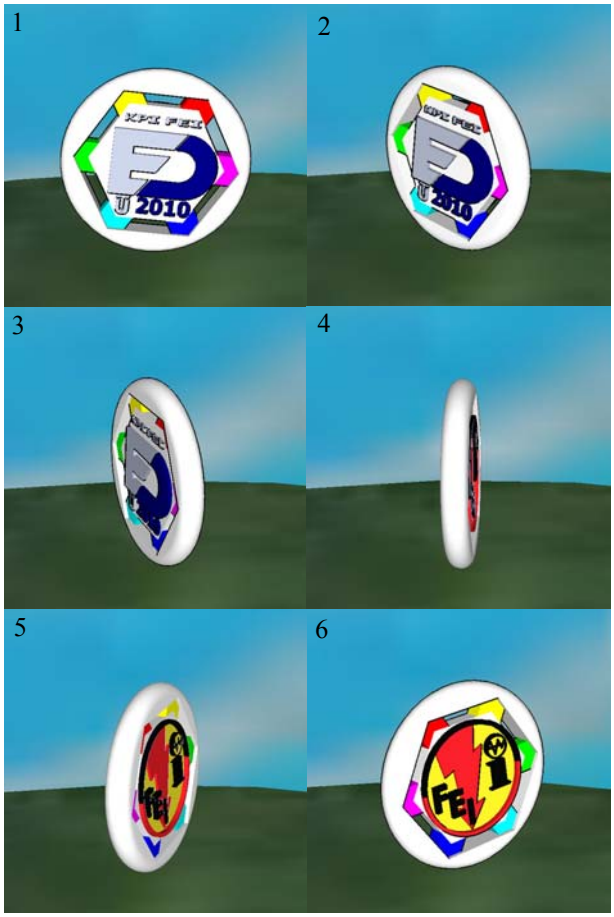

Fig. 5 Example: the phases of object rotation

In first step are computed angle increments in all axes. The methods *.ox*, *.oy*, *.oz* of object *object* are internal geometric rotation transformations. Next step represents the test of minimal increment size. In the next part is computed rotation direction and step size for every axis. Then whole sub-rotation is implemented and last part

Unauthenticated | 194.138.39.60
Download Date | 11/5/14 3:11 PM
VERSITA EMERGING SCIENCE PUBLISHERS

contains test of destination position. Visualization engine executes this script in loop. Examples of some frames of rotation are depicted in Fig. 5.

5. CONCLUSIONS

VR system for immersive visualization of complex datasets was developed at DCI FEI TU Košice [10][14]. This system is still under development (4.-th generation) and has these main features: hierarchical graph scene representation, support main 3D models formats, support script controlling (Ruby, Python), visualization engine with multi-screen and cluster support. Our VR system can be used like information or training immersive interactive system. This system presupposes use of GPGPU technology mainly for visualization.

In future we plan to use and develop more transformations in our VR system. Use of automatic generates input described in [12] can increase flexibility and range of usability of this system for next experimental work. We also intend to utilize formal methods suitable for design of time-critical systems, such as Time-basic Nets [13] during the development of algorithms and optimization of process distribution [15][16]. GPGPU technology will be used mainly for real-time visualization and for augmented reality applications.

ACKNOWLEDGMENTS

This work is the result of the project implementation: Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120020) supported by the Research & Development Operational Program funded by the ERDF.

REFERENCES

[1] ĎURATNÝ, M. – PEŤKA, R.: Photorealistic Visualisation in Cluster Environment. In: AEI '2008: International Conference on Applied Electrical Engineering and Informatics 2008: September 8–11, 2008, Košice: TU FEI, 2008, pp. 116–120, ISBN 978-80-553-0066-5.

[2] PEŤKA, R. – HAŠKO, M.: Visualization of Technological, Industrial and Architectural Buildings. In: AT&P Journal, Vol. 16, No. 6, 2009, pp. 63–65, ISSN 1336-233X (in Slovak).

[3] YANG, X. – PETRIU, D. C. – WHALEN, T. E. – PETRIU, E. M.: Script Language for Avatar Animation in 3D Virtual Environments. VECIMS 2003 - International Symposium on Virtual Environments, Human-Computer Interfaces, and Measurement Systems, Lugano, Switzerland, July 27–29, 2003, pp. 101-106.

[4] DIETRICH, A. – GOBBETTI, E. – YOON, S.-E.: Massive-Model Rendering Techniques: A Tutorial. IEEE Computer Graphics and Applications, Vol. 27, No. 6, pp. 20–34, 2007.

[5] Ruby official website, url: http://www.ruby-lang.org/en/

[6] Python Programming Language – official website, url: http://www.python.org/

[7] LUA – the programming language – official website, url: http://www.lua.org/

[8] SHIRLEY, P. – MARSCHNER, S.: Fundamentals of Computer Graphics. Publisher: A K Peters; 3rd revised edition, 2009, p. 804, ISBN 978-156-881-469-8.

[9] SOBOTA, B. – STRAKA, M. – PERHÁČ, J.: Some Problems of Virtual Object Modelling for Virtual Reality Applications. Journal of Information, Control and Management Systems, Vol. 6, No. 1, 2008, pp. 105–112, ISSN 1336-1716.

[10] SOBOTA, B. – PERHÁČ, J. – SZABÓ, Cs. – PETZ, I. – HROZEK, F.: Tasks Solution for Large Graphical Data Processing in the Environment of Parallel, Distributed and Network Computer Systems, Computer Science and Technology Research Survey, Košice, KPI FEI TU Košice, 2009, 4, pp. 45–53, ISBN 978-80-8086-131-5.

[11] SkethUp home page. url: www.sketchup.com

[12] PORUBÄN, J. – VÁCLAVÍK, P.: Separating User Interface and Domain Logic, Analele Universitatii din Oradea, Proc. 8th International Conference on Engineering of Modern Electric Systems, Oradea, May 24–26, University of Oradea, Romania, 2007, pp. 90–95, ISSN 1223-2106.

[13] HUDÁK, Š. – KOREČKO, Š. – ŠIMOŇÁK, S.: Reachability Analysis of Time-Critical Systems. Petri Nets: Applications, Vukovar, Croatia, In-Teh, 2010, pp. 253–280, ISBN 978-953-307-047-6.

[14] SOBOTA, B.: Some Problems Virtual Reality Systems Visualisation Frame Solution in Parallel Computing Environment; Habilitation thesis, FEEI TU Košice, 2008, p. 103.

[15] TOMÁŠEK, M.: Behavioral Scheme of Mobile Processes. Journal of Information Control and Management Systems, Vol. 5, No. 2, ISSN 1336-1716, pp. 371–382, Žilina, 2007.

[16] TOMÁŠEK, M.: Computational Environment of Software Agents. Acta Polytechnica Hungarica, Vol. 5, No. 2, ISSN 1785-8860, pp. 31–41, Budapest, 2008.

Received December 4, 2010, accepted April 8, 2011

BIOGRAPHY

Branislav Sobota was born on 22.05.1967. In 1990 he graduated (MSc.) with honours at the Department of Computers and Informatics of the FEEI at Technical University in Košice. He defended his PhD. in 1999 and habilitation thesis in 2008 in the field of virtual reality and computer graphics. He is working as an associate professor at the Department of Computers and Informatics. His scientific research is focusing on computer graphics, virtual reality, modelling and simulation and parallel computing.

Unauthenticated | 194.138.39.60
Download Date | 11/8/13 3:11 PM
VERSITA EMERGING SCIENCE PUBLISHERS