DATA MODELS FOR VIRTUAL REALITY

Eng. Gabriel Felician Muresan – Continental Teves AG&Co OHG,
e-mail: gabriel.muresan@contiteves.com
Dr. Eng. Liana Stanescu – University of Craiova
e-mail: stanescu@nt.comp-craiova.ro
Dr. Eng. Dumitru Dan Burdescu – University of Craiova
e-mail: burdescu@topedge.com

Abstract:The scenes represented through Virtual reality are becoming bigger and bigger, with a considerable increase of the details level. The high quality images require long computation times and memory-consuming scene description. Parallel architectures with distributed memories are increasingly being used for rendering and provide more memory and CPU power .Ideally speaking, each object composing a scene can be independently processed, so a natural parallelization method is distributing the scene's objects between the machine's nodes. On the other hand, a more important part is the method one chooses to represent the data. Of course, this depends also on the 3D data to be represented and it varies from one case to the other.

Keywords: VRML, virtual reality, data models.

INTRODUCTION

There are several possibilities in transforming a 3D world into a database, but all of them are depending on the way the database is represented.
For the 2 examples chosen in this article we chose the most popular of them – VRML (Virtual Reality Modelling Language).
VRML is a file format for the interactive description of the 3D worlds and objects. VRML is designed to be used on the internet, intranet and locally. It is supposed to be an universal interchange format for multimedia and 3D graphics. An advantage is represented by the fact that the VRML standard (ISO/IEC 14772) is not defining the physical devices or any other implementation dependent concepts (e.g. screen resolution and input devices).
The whole generality of descriptive language and its capability of running on any type of machine that has a web browser are making it a powerful candidate in winning the battle for the home-use virtual reality applications. Of course, depending on the application, the VRML files can have a more or less general format.
In the first part of this article will be presented an application used for encoding the static 3D data and in the second part an application that is used for storing the animation in a VRML world.

PART I

The application is done for a real-estate agency. It contains 2 modules and it's used to describe the apartments in a region.
There is a central database located on the server and written in SQL.
The application is written in Visual Basic.
Due to the fact that the application is working with 3D landscapes in which there are represented only apartments, these can be decomposed in the same type of primitives – parallelepipeds.
In VRML these are represented through the following structure:

```
Shape {

  appearance      Appearance {
    material        Material {
     ambientIntensity  ambient_val
     diffuseColor difcol_val1 difcol_val2
            difcol_val3
     specularColor speccol_val1 speccol_val2
            speccol_val3
     emissiveColor  emiscol_val1 emiscol_val2
            emiscol_val3
     shininess       shin_val
     transparency  transp_val
    }
  }
 geometry          Box {
                size  val1 val2 val3
  }
}
```

This structure can be used for all the primitives used by the program.
On this shape can be applied 3 transforms – rotation, translation and scaling.
An example of an apartment designed for this application is given in Fig.1:

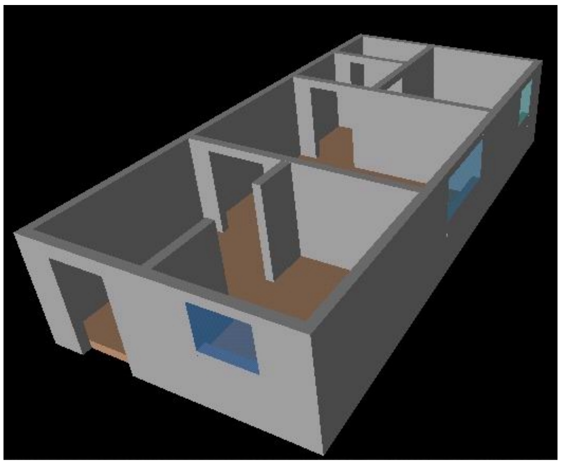


Fig. 1

Taking into account that this application is done for 3D we took into account also some predefined viewpoints. The general structure of a viewpoint is the following:

```
DEF vp_name Viewpoint {
    position        poz1 poz2 poz3
   orientation      orient1 orient2 orient3 orient4
   fieldOfView      fov
  description       "viewpoint description"
}
```

Taking into account the above described primitives and the elements describing them we can build the database structure:

Apartments

id_apartament	id_material	sizex	sizey	sizez	trx	try	trz

Viewpoint

id_ap	id_vp	positionx	positiony	positionz	orientationx	orientationy	orientationz	orientationt	fieldOfView	description

Material

id_mat	aInt	dColR	dColG	dColB	sColR	sColG	sColB	eColR	eColG	eColB	shin	transparency

As it can be seen the structure is quite simple (ordinary we might add), but it's efficiency is high (as it will be shown later on).

The application works as follows: the client makes a request t the server for apartments fulfilling certain characteristics. The server solves the request and returns the positions on the map where the apartments can be found. The client chooses one of the apartments, thus requesting supplementary information to the server in order to visualize the apartment. The server questions the database and takes the element corresponding to the chosen apartment and it constructs the VRML file from them. From now on we'll call this part "decoding".

When a client wants to insert an apartment into the database, it sends the descriptive VRML file to the server. The server takes the file, it makes a linearization, decomposes it into primitives and inserts it into the database. We'll call this part "encoding".

Taking into account that this article's subject is not the client-server communication, we'll further discuss only the encoding/decoding.

In the encoding part, the VRML file should firstly be linearized. The object linearization means passing them from their reference system to the initial reference system. A virtual reality scene has the following structure:

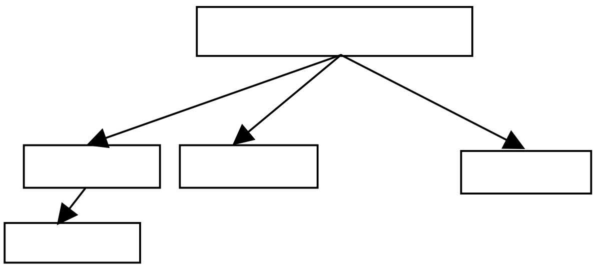


Fig.2. A VR scene structure

where n(k, i, j) represents an object and K is that object's level, I is the level index and j is it's parent index.

After the linearization process, the internal structure of the VRML file is the following:

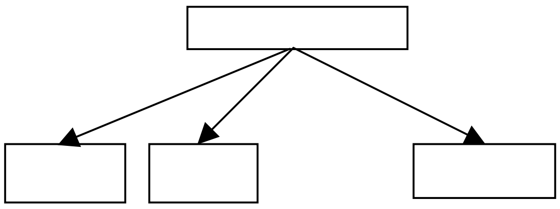


Fig. 3. The linearized VR scene structure

To describe the scene, after the linearization, the order in which the objects are drawn is not important anymore, taking into account that the objects are drawn in the initial reference system, not depending on the their parents (all the objects are on the same level of dependencies), so in the database is not necessary anymore a tuple to describe the father-child relationship. After the object linearization, the file is analysed and the data inserted into the database.

For this purpose it was created the formal language and the grammar for the virtual reality scene.

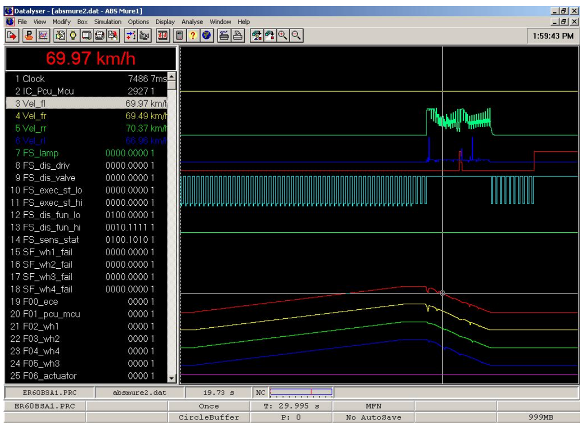
The decoding consists only in taking the parameters from the database and reconstructing the VRML file accordingly.

One must notice that the reconstructed VRML file is directly linearized, but, as we mentioned before, this is not bringing out any change in the content of the virtual reality scene.

The system presents more advantages: firstly it offers a way of inserting 3D objects into a database and it gives a direction for future refining and extensions.
On the other hand, although the VRML files are text files (so the space occupied by them is not too big), after inserting the data into the database, it's space is dramatically decreased.
According to the experimental data, after inserting one apartment into the database, the space allocated on the disc decreases with 65-68%.
After inserting the second apartment, the space decreases with 76-82%. Continuing the insertions, the limit will go up to 90% decrease of the space allocated for the data.

Part II

This second part of the article will present an application that deals with the animation in a VRML scene. The animation occupies a lot of space in a file.
The application describes the speeds of the wheels of a car during several manoeuvres.
We are supposing that the car is equipped with the latest types of brake systems (ABS, TCS, ESP).
Using the EZS200 simulation machine set on the RTS mode (real-time simulator) there were done a series of tests for the ABS, TCS, ESP and MASR manoeuvres on 2 real controllers – one for Audi TT and another one for BMW X5. For this purpose there are used protocol files in which there are described the signals that have to be measured during the simulation and the channel associated to each signal. To the simulator there was connected a real ECU (which is used in the car after the tests).
After there was written a script for describing the manoeuvres that are to be done by the car.
All the scripts were done in a language specific for the ZS2000 and were done for a 30 seconds time length.
After downloading the script into the machine was downloaded the vehicle emulation data.
To be noticed that the controller was used with all the software written inside.
During the simulation the controller and the simulator are communicating using messages through a Control Area Network (CAN) used by all the car manufacturers.
These messages contain the values of the signals written into the protocol files.
After finishing the tests, the result files were taken and analysed with a special application – Datalyzer.
In the figure below there is shown an ABS manoeuvre:


```
69.97 km/h
 1 Clock            7486 7ms
 2 IC_Pcu_Mcu       2927 1
 3 Vel_fl           69.97 km/h
 4 Vel_fr           69.49 km/h
 5 Vel_rr           70.37 km/h
 6 Vel_rl           66.96 km/h
 7 FS_lamp          0000.0000 1
 8 FS_dis_driv      0000.0000 1
 9 FS_dis_valve     0000.0000 1
10 FS_exec_st_lo    0000.0000 1
11 FS_exec_st_hi    0000.0000 1
12 FS_dis_fun_lo    0100.0000 1
13 FS_dis_fun_hi    0010.1111 1
14 FS_sens_stat     0100.1010 1
15 SF_wh1_fail      0000.0000 1
16 SF_wh2_fail      0000.0000 1
17 SF_wh3_fail      0000.0000 1
18 SF_wh4_fail      0000.0000 1
19 F00_ece          0000 1
20 F01_pcu_mcu      0000 1
21 F02_wh1          0000 1
22 F03_wh2          0000 1
23 F04_wh4          0000 1
24 F05_wh3          0000 1
25 F06_actuator     0000 1

ER60BSA1.PRC   absmure2.dat   19.73 s   NC
ER60BSA1.PRC                  Once           T: 29.995 s   MFN
                              CircleBuffer   P: 0          No AutoSave   999MB
```
Datalyser - [absmure2.dat - ABS Mure1]
File View Modify Box Simulation Options Display Analyse Window Help
1:59:43 PM
Fig.4. An ABS manoeuvre for an Audi TT controller

From this program the data were exported and after that inserted into a database. There are 2 sampling rates for the controllers used (and provided by Continental Teves) – 10 milliseconds and 7 milliseconds. So, for the 30 seconds, we can have for each test either 3000 records or 4285 records. For the 3D representation of the car movement there are important the wheel speeds, the forces on each of the wheels, the road model, the steering angle and the steering ratio. For the beginning the application is dealing only with the wheel speeds and there is taken into account the further development for inserting the other basic elements of a car movement.
The application was done in C++ Builder and for representing the 3D objects it was used a Cosmo ActiveX provided by Silicon Graphics. The script for the 3D objects is done in VRML.
In the following we'll present a summary of the application and the main points in it's implementation. In the figure below there is shown the ain form of the application:

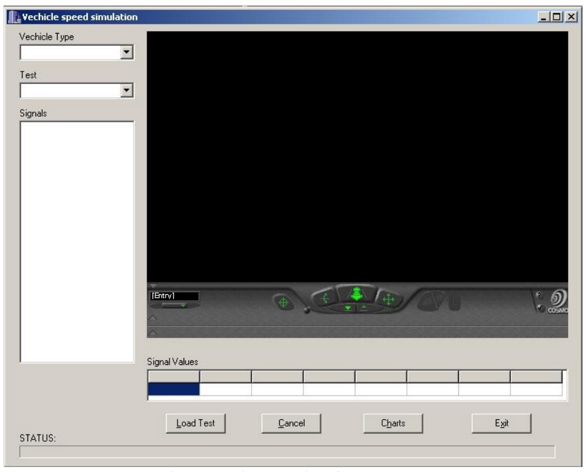
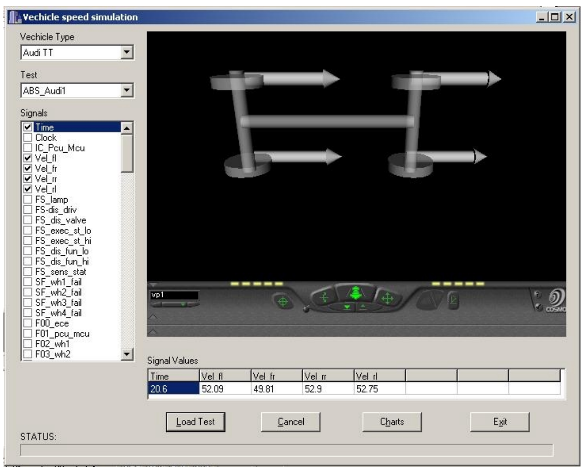


Fig. 5 The main form

As it can be seen, the user is able to choose the vehicle type. After that all the tests available for this type of vehicle are taken from the database and loaded into the appropriate combo. Once the test is also chosen, all the available signals for this test will be displayed:

```
void __fastcall TfrmSpeed::cmbTestChange(TObject
*Sender)
{
  clbSignals->Clear();
  String sSQL = "SELECT * FROM Signals WHERE Test=
\'"+cmbTest->Text+"\'";
  qrySimulation->SQL->Clear();
  qrySimulation->SQL->Add(sSQL);
  qrySimulation->Open();
  dtsSimulation->DataSet = qrySimulation;
  while (!dtsSimulation->DataSet->Eof){
    clbSignals->Items->Add(dtsSimulation->DataSet-
>FieldByName("Signals")->AsString);
    if (dtsSimulation->DataSet->FieldByName("Signals")-
>AsString == "Vel_fl")
       clbSignals->Checked[clbSignals->Items->Count - 1]=1;
    if (dtsSimulation->DataSet->FieldByName("Signals")-
>AsString == "Vel_fr")
       clbSignals->Checked[clbSignals->Items->Count - 1]=1;
    if (dtsSimulation->DataSet->FieldByName("Signals")-
>AsString == "Vel_rl")
       clbSignals->Checked[clbSignals->Items->Count - 1]=1;
    if (dtsSimulation->DataSet->FieldByName("Signals")-
>AsString == "Vel_rr")
       clbSignals->Checked[clbSignals->Items->Count - 1]=1;
    dtsSimulation->DataSet->Next();
  }
  Ucp1->SRC = "example.wrl";
  return;
}
```

After all this the test has to be started.
Taking into account the amount of data that has to be analysed and written into the VRML file, this process is taking up to 10 seconds, depending on the computer used. A part of the test looks like this:


Vechicle speed simulation

Vechicle Type: Audi TT

Test: ABS_Audi1

Signals:
- [✓] Time
- [ ] Clock
- [ ] IC_Pcu_Mcu
- [✓] Vel_fl
- [✓] Vel_fr
- [✓] Vel_rr
- [✓] Vel_rl
- [ ] FS_lamp
- [ ] FS-dis_driv
- [ ] FS_dis_valve
- [ ] FS_exec_st_lo
- [ ] FS_exec_st_hi
- [ ] FS_dis_fun_lo
- [ ] FS_dis_fun_hi
- [ ] FS_sens_stat
- [ ] SF_wh1_fail
- [ ] SF_wh2_fail
- [ ] SF_wh3_fail
- [ ] SF_wh4_fail
- [ ] F00_ece
- [ ] F01_pcu_mcu
- [ ] F02_wh1
- [ ] F03_wh2

Signal Values

Time	Vel_fl	Vel_fr	Vel_rr	Vel_rl			
20.6	52.09	49.81	52.9	52.75			

STATUS:

Fig. 6. A test example

There are also shown in real-time the values of the signals that were selected from the list. For this purpose we used a timer which is working on a separate thread so that these values can be displayed.
Taking into account the refresh rate which makes the human eye not to observe the changes we established the sampling rate to 200 milliseconds.
At the "OnTimer" event of the timer this makes the difference between the time elapsed from the last exit from this event and the actual time and depending on this it moves the positioning in the database to the corresponding values. Thus we are reducing as much as possible all the delays that can appear:

```
void __fastcall TfrmSpeed::Timer1Timer(TObject *Sender)
{
  Word h,m,s,j;
  TDateTime timp1 = Now();
  TDateTime ElapsedTime = timp1-timp0;
  DecodeTime(ElapsedTime,NULL,NULL,NULL,j);
  timp0=timp1;
 for (int i=0;i<j/10;i++){
   dtsSimulation->DataSet->Next();
   prbTest->Position++;
  }
 for (int i=0;i<nr_semnale-1;i++)
   sgrValues->Cells[i][1]=dtsSimulation->DataSet-
>FieldByName(semnale[i])->AsString;
  if (dtsSimulation->DataSet->Eof) {
    Timer1->Enabled=0;
    prbTest->Position=0;
  }
  return;
}
```

The most important part of the application is the creation of the VRML file.
It has 2 types of components – mobile and fixed.
The fixed part contains the central axe, the front and the rear axes and the 4 wheels.
The mobile part of the VRML environment is formed by the 4 wheel speeds, each of them being composed by a cylinder that represents the magnitude of the speed and cone for the pointer representing the peak.
The animation is realised using a scalar interpolator for the cylinder and a position interpolator for the cone, each of these being done for every wheel.
After writing the points for the interpolators for each wheel and the time points for them there have to be written the routes so that the animation can take place.
An example of the routes for one of the wheels is the following:

```
ROUTE
UnnamedTransformTranslationInterp_11.value_changed TO
_Cyl1.set_translation
ROUTE
UnnamedTransformTranslationInterp_12.value_changed TO
_Con1.set_translation
ROUTE UnnamedTransformScaleInterp_13.value_changed
TO _Cyl1.set_scale
ROUTE UnnamedAnimation0Time_4.fraction_changed TO
UnnamedTransformScaleInterp_7.set_fraction
ROUTE UnnamedAnimation0Time_4.fraction_changed TO
UnnamedTransformTranslationInterp_11.set_fraction
UnnamedTransformTranslationInterp_17.set_fraction
ROUTE UnnamedAnimation0Time_4.fraction_changed TO
```

The space occupied by the tests on the disc is approx 1.5MB for each test file and for each VRML file containing a test is approx 2MB.
On the other hand, when inserting a single test into the database, its dimension is 1.8MB, so between the space occupied by the tests values (which are meaningless without a visualization

application) and the space used by a single VRML file to describe the test. However, when inserting more data, the ratio is in the database favour, i.e., for 12 tests the dimension of the database is approx 13MB.

REFERENCES

1. SP Reifenwerke - Bericht über den Reifenlasten, 1999
2. Dirk Waldbauer - ABS Kurzbescreibung, 2000
3. Dr. J. Karner – Automatische Programverifikation, 2002
4. R. Gutwein, Wolfgang Kling – Software Dokumentation fur heckangetriebene Fahrzeuge – ABS Phasenerkennung, 1999
5. R Gutwein – Yaw torque control, 1997
6. R. Gutwein, W. Kling – Neue Arbitration, ABS-ESP, 1999
7. R. Gutwein – ABS Druckmodel, 2000
8. Ivo Bastic – Signalaufbereitung ABS, 1999
9. K. Roettger – TCS System, 2001
10. VRML specification