Building the Local Navigation System

Štěpán Kment
xkment@cslab.felk.cvut.cz

Computer Graphics Group
Czech Technical University
Prague / Czech republic

Abstract

This document discusses the problematic of design of the Local Navigation System **(LNS)**, de facto the interactive map useful for 3D objects like houses with multiple floors. It covers philosophy of the local navigation system, list of useful technologies and guidelines of design in HTML and VRML languages.

Keywords: local navigation system, interactive maps, VRML, HTML, JavaScript, 3D scene design

1. Introduction

Mankind has developed lots of navigation systems in last millenium. First, and very long time the only systems, were not interactive; the user simply had to search for information himself. We can join maps, for instance, to this group.
New technologies brought new ideas to this area, through use of satellites we can detect our position anywhere on the Earth's surface with resolution of less than tenths of meters, because it is good enough for navigation of the homing missile, it is also good for navigation of far slower car on the road - the system exists, it is called GPS - Global Positioning System - and the hardware, which detects current position, is quite inexpensive.
All that works fine, but it still works only on 2D surface of the Earth; we need to rebuild these systems completely, when we need to work in 3D environment, such as the skyscraper with hundreds of floors; also there is infinite number of details in even a small wood, but sometimes there are only gray walls in the modern building, there is a lack of navigation points there. There must be used completely different interfaces, different views (imagine 300 sheets of paper, a map of one floor of the skyscraper on each!). Also, maps or GPS systems cover areas of, say, from one to thousand or more square kilometers, but typical 3D structure built by mankind is much smaller from top view, we are discussing "Local Navigation Systems - LNS". Now you probably see, that there are enough differences and there is a reason to think about this different class of navigation systems a bit more.

2. Local navigation systems - philosophy

There are some problems with 3D nature of objects created by mankind. It could be surprising, but just a few people, who are working in some special jobs, are used to think in 3 dimensions, most people can not orient themselves in such environment except they are guided. If you ask anyone,

what is the farest object they can see, they probably look to the Earth's horizon and say, that it is the building 5 kilometers far. Mostly they do not think about top direction, where they can see the Sun in distance about 30.000.000 times farer! That is our nature, we walk on the Earth, we are not used to fly. That means that if we build complete 3D model of the building (with all interiors), it would be nearly useless as the map, we must cut the building into separate 2D floors. Or there is another way of presentation, which is available few last years, when computers became able to replay animations in random order without slow seeking - it is possible to record movie, which shows the movement from one point, say the main entrance, to the other, for instance director's room.
There are not many straight paths in normal building, if one describes the way to the other point, it mostly looks like

turn left, go ahead, there is an lift, use it - 2 floors up, then right, ahead, left, upstairs, ahead, there it is

You probably agree that this sentence is a bit hard to remember, especially when it is more than three times longer (this sometimes happens in one of the old buildings of our university). Man's memory much more remembers animation and some unique orientation points, especially when one can replay the animation many times simply. Also, orientation points can make one remember more information seen previously - the brain's memory works on principle of associations (and, of course, these unique objects are good input for this process).
Let us now derive some results form these ideas:

- The system must contain unique orientation objects - like flowers, noticeboards, lifts, stairs, doors, passages from one building to another etc.
- The system of navigation should contain at least descriptions of paths from these important points in the building to the place one is going to, if it is not capable of searching for the path from one to another point interactively.
- The system, should contain information about overall architecture of the building - for instance the user should learn quickly, that when he comes to the red lift, he is in the right corner of the building, and yellow lift is in the left.
- The information provided by the system must be very simply accesible, that means faster than asking the college. That is why most interactive solutions of this problem are useless - we are limited to interactive, computer-accelerated solutions.
- The system should also contain more than a map. It is useless to have complete list of rooms described by the numbers, when one wants to search for the concrete office.
- The system should be simply updatable, that means server-based solution, perhaps in only very special cases, where system operator is able to configure replication of files, we can recommend distributed solution.
- The creation process of the design should be fast and inexpensive, otherwise none would make it. On the other hand the system should be quite complex and full of useful information.
- Common networks in the building of common companies or offices are too slow for realtime video solutions, also office computers are not graphic workstations, the solution must be small and not too CPU-intensive - at least for some next years.
- The solution should be designed by using industry-standard technologies with continuous development and support of new releases of operating systems. Also, client licenses should be free of charge.

There are few technologies we can use, but probably the simplest and completely inexpensive is to design common web site on the building intranet. Nearly everyone has access to WWW, that means he or she has installed browser. Downloading the plug-ins for extra capabilities is also simple, the plug-ins are mostly free.
There are following technologies available:

- Frames, that means more documents loaded in one window. The user has still access to main menus and the site is very simple to use.
- Java, JavaScript and dynamic HTML is very strong "weapon" in our hands. One can do unbelievable things with it, even more complex effects than in most special presentation programs. Currently it also means ability to render 3D graphics, move objects on the screen, sort database on the client's machine, play sound animation and using Java or ActiveX run complex business software. I have used Internet Explorer as a base for my friend painter Martin Zhouf's interactive gallery commercial presentation!
- We can use Apple QuickTime VR or other (for instance LivePicture) plug-ins to show spherical or cylindrical views of the scene.
- VRML2 is quite complex and powerful technology, it is our choice because of ability of scripting, fast development of the technology, very small data files and real-time (well, sometimes near-to-real-time) output rendered using client's CPU.

I decided to use combination of VRML2 and HTML extended by Java scripts in my systems. There exist very good authoring tools for each of these technologies and they are widely accepted as industry-standards. The rest of this paper is just a list of experiences I have learnt about these technologies and partially also a how-to guide.

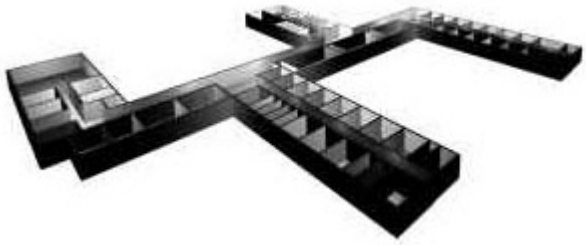
3. Main stages of the Local Navigation system's design

LNS is created in some steps, let us discuss stages of the production pipeline:

- First, one has to decide, if he or she wants to create LNS for whole building. Then think about main orientation points of the building. And finally, what is the power of hardware of client computers and network in the building (for internet presentation loaded through WAN data must be as small as possible, of course). More power means more pictures, texture maps, sound etc.
- Then one has to install authoring tools, that means editors of HTML, tools for VRML design (it is in fact impossible to author VRML by hand) and tools for 2D paint (for texture design). I personally work professionally with Microsoft FrontPage (HTML and web site management), 3D Studio Max from Kinetix (3D scene design and automatic export to VRML format) and Adobe Photoshop (2D painting, export of bitmaps in appropriate formats and even export of partially transparent bitmaps). I often mention technologies available in these programs, but other packages work nearly the same way (for instance Alias|Wavefront's Power Animator or Softimage|3D).
- One has to obtain plans of the building. It is in fact impossible to reconstruct plans by hand. From top view of each floor it is quite simple to reconstruct the scene in 3D.
- Then one has to model each floor in 3D modelling tool. It is quite time consuming process, especially we need to keep sizes in the same scale ratio to original - perhaps it is a task for CAD programs and not for tools for animation, but I do not have any knowledge whether these systems support VRML export. [described in detail later]
- Then one has to refine the scene and add the interactivity to it, that means camera movements, moving doors, elevators etc. and also bitmap textures mapped onto orientation-important objects. [described in detail later]
- Link VRML worlds together through anchor nodes.
- Add HTML pages and framesets. That means one has to add pictures of floors shot from top view and map image clicking maps in it, create scripts with control logic, add lots of additional text with description. [described in detail later]
- Publish the site on intra/internet.
- Keep information on the site actual (periodic updates).

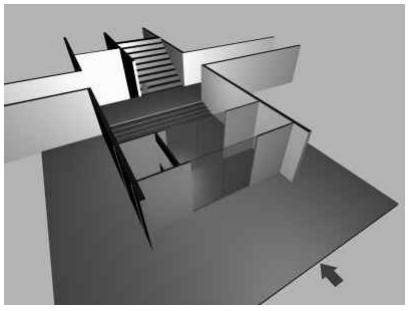
4. Building the scene in 3DS Max or other 3D package

When we have plans of the building, we can start building our 3D scene. In 3DS Max, I preferably use primitives - boxes, spheres, cylinders and others - VRML export plug-in exports them directly as sipmle geometry nodes. I recommend to build scene according to Max's units displayed in toolbar. Simply measure the sizes of object (for instance wall) and create the primitive in Max with appropriate sizes. I also used copying using movement limited to one axis often, in this case one has to set two or even all three sizes only once, then many rooms can be created by copying or using linear array tool. Also, if one sets the material and eventually mapping of the first box, all copies have these properties set authomatically.

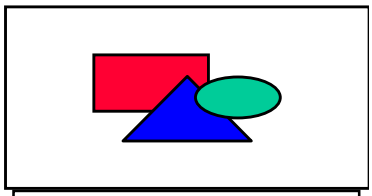


One VRML file should contain, depending on its complexity, one or more floors, but certainly not whole building. The model has to contain few objects, otherwise the rendering speed in VRML player would be too slow. Limit number of objects, lights and textures in the scene - all these nodes slow the speed down, especially lots of texture-mapped objects are problem for machines without hardware-accelerated graphics. Limit number of objects around the scene, for instance trees in a garden near the building are nice, but not necessary.
Export to VRML works, at least in 3DS Max 2.0, with beta0 version of the plug-in, without any problems and the scenes appear in VRML player nearly the same as in Max's interactive viewport with Gouraud shading set on.

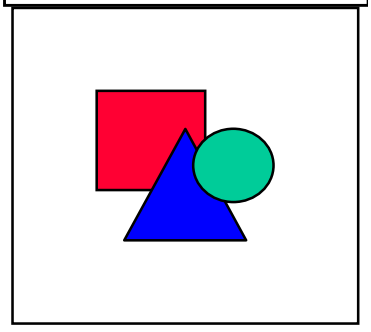
5. Experience with VRML - building interactive world



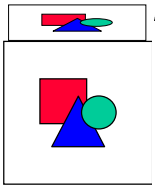
VRML nodes can be found in 3DS MAX in following location:



...and this is a list of available helpers:



5.1 How to setup VRML player's NavigationInfo node



```
NavigationInfo
{
type "NONE"
headlight FALSE
visibilityLimit 0 # infinite; if >0, then all farer than limit is not displayed
}
```

Rendering of the 3D scene is a very processor-intensive task and without appropriate hardware acceleration even very powerful machine (for instance Pentium II / 266 MHz with Matrox Millenium II 2D graphics adapter I have been working on) can't render larger scenes in more than 10fps in a small window (full-screen is absolutely unusable). That is why we need to optimise our VRML world.

- Turn off the gravitation force and player's controls - by switching the player mode from WALK to NONE we are allowed to fly, but as I've mentioned earlier, all our moves are pre-calculated and we don't need to worry about this. The result is that VRML player does not search for any object below us. Also all icons of the player are not visible and so the user is forced to use only predefined paths we have defined for our animated viewports.
- Turn off the headlight. We have defined other lights in 3DS Max and so there is no dark place in the scene. Another lighting is not useful.
- If the floor is really large, it is possible to limit the visibility - this can dramatically increase performance but be aware of it - if the limit is too small, all the walls on the screen may not be rendered (it is the same effects as if one goes through a fog and sees only near objects).

5.2 Turning collision detection off

```
Collision
    {
    collide FALSE
    children [...]
    }
```

Because we defined animation paths with care in 3DS Max, we don't need to worry about collisions with walls or other objects. So it is very useful to turn off the collision detection, we again save lots of CPU time. Just copy entire scene into the children field of this grouping node and set field collide to FALSE.

5.3 Placing text in the scene

```
Text
    {
    string "TEXT"
    fontstyle

        {
        family "SANS" #face of the font
        size 20 #size of the font in space
        }

    }
```

I did not manage to export text object from 3DS Max dierctly. The problem is that one can, of course, create text in 3DS Max as a 2D shape extruded to 3D, but when one exports this object, he or she gets large list of vertices and faces. Max simply can't (at least this version of export plug-in) export Text node and replaces it with extremely large list - the text is then defined as any other complex object. I found simple, but still not perfect, workaround. First place the box or some other

primitive in the Max's scene where the text should be and give it name - for instance TEXTBOX. Export the scene to the VRML file and load that file in text editor. Now search for TEXTBOX word. You should see something like this:

```
DEF TEXTBOX Transform {
translation 0 0 0
rotation 0 0 0 0
scale 1 1 1
children [
   Transform {
      translation 0 0 0
      children [
         Shape {
            appearance Appearance {
               material Material {diffuseColor 0.6 0.8941 0.6}
            }
         geometry Box { size 173.9 60.87 85.02 }
        }
      ] }
   ]
}
```

now replace geometry field of the Shape node with the TEXT node:

```
Shape {
   appearance Appearance {
      material Material {diffuseColor 0.6 0.8941 0.6}
      }
  geometry Text {
      string "TEXT!!!"
      fontStyle FontStyle {family "SANS" size 30}
   }
}
```

Now there should be your text somewere around the previous place of the box. But it can be rotated or translated, also experiment with size field of FontStyle node. The only advantage is that the text object is somewhere around the place, where it should be. Of course, do NOT use export of font objects directly from 3DS Max, numbers of faces and vertices would increase catastrophically.

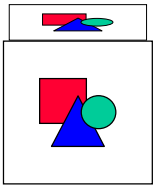
5.4 How to build "navigation web" in the scene

This is very important part of scene design. If navigation works badly or doesn't work at all, the system becomes unusable because the user interface is hidden. First we need to decide, where we can go in the scene and each place should be linked to the others at least in one direction. Connections are built in 3DS Max using standard free cameras.

- At first put the first camera in the scene in 3DS Max and experiment with its FOV - field of visibility (I would recommend to extend default 45 degrees to 60).
- Then move this camera to the first entry-point of the scene.
- Decide, where the paths could split, then you get some virtual plan of nodes and connections between them - I call it "navigation web".
- Now animate camera to the first node (that means turn on the Animate button, set different time and move and/or rotate the camera to its new position. If another path continues from this point, do not use this camera. Set the time to the key where the camera reaches the final position, now Shift-click the camera object and create a copy of it. Edit keys of this copy - delete all but the final position and rotation, then move these final keys to frame 0. Now we got another camera that starts its movement in the place, where has previous stopped. Repeat this task until all paths are created. (remark - it is highly recommended to name every object

- you'll probably script them and so *CAM-entrance* is probably better name than default *Camera01*.

- Create TouchSensors - buttons, which start the animation. At first place one button (I use blue arrows rotated to the direction of next movement) per one camera somewhere around its start position. Under the panel Create/Dummies/VRML2.0 you can find TouchSensor object - put these objects close to the buttons you have just created - one sensor per one button - and set their properties: First pick the trigger - the button object, then pick the action object - the camera. Keep the sensor enabled. Now is basic animation finished, if you set in the VRML player appropriate viewport and click the button, you should be navigated through the scene. But we still need to improve logic of the scene and by clicking the button we have to also set the viewport as current one.



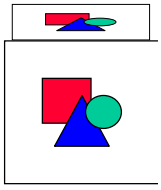
- Export the scene into the VRML file and load the file into the editor. We must add some ROUTE directives to the file. Each path between two nodes must consist of these nodes:

```
DEF Camera01 Viewpoint {...}
DEF Camera01-TIMER TimeSensor { loop FALSE ... }
DEF Camera01-POS-INTERP PositionInterpolator {...}
DEFButton-ROOT Group {
   children [
      Shape {...}
   DEF TouchSensor01-SENSOR TouchSensor { enabled TRUE }
   ]
}
ROUTE TouchSensor01-SENSOR.touchTime TO Camera01-TIMER.startTime
ROUTE Camera01-TIMER.fraction_changed TO Camera01-POS-INTERP.set_fraction
ROUTE Camera01-POS-INTERP.value_changed TO Camera01.set_position
```

The example above is a code generated for a camera named *Camera01*. There is set initial viewport posiotion, TimeSensor (generates timer events for the camera animation), Position and/or Rotation interpolator with large list of key values (is responsible for definition of position of the camera in space and time), button shape grouped with TouchSensor (responsible for sending the start event). Routes link all these elements together. First one sets the start time of timer and activates it, second passes periodic events from timer to interpolator, calculated new position of the camera is then passed to the camera object. We need to hang another route to the first event of the TouchSensor. When one comes to some node, there is still set old viewport. We need to set new viewport, otherwise the viewport would be animated but we would be looking into the scene through different one. Just add this line:

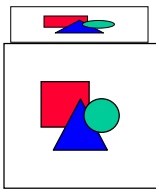
```
ROUTE TouchSensor01-SENSOR.isActive TO Camera01.set_bind
```

Now when the sensor is clicked, it sends one message of boolean type to the camera object and the camera is bound as the current viewport.
I still hve not explained why it was necessary to create so many cameras. If one doesn't want to split one animated path into more parts, 3DSMax simply exports data for one interpolator from the first to the last key of the object; if we used one camera for more than one path, it would be impossible to stop on the next node. We can also use different process - we can hang more TimeSensor helpers to one object, there we can set start and stop values and this way we split whole animation of the camera to more parts. Some extra work on WRL file may be necessary this way.



5.5 Refining the interaction

Now we have created basic navigation, we can refine it a bit. Note that the WRL file is now changed and any export from 3DS Max to this file would overwrite the changes!



- It is not necessary to have all the start buttons visible everytime. We can use LODs to hide them. In 3DS Max there is another VRML helper called LOD. Create it near the button and pick the button. Now click on the line with button's name and set the distance - you can see a blue sphere painted in the scene. If one comes inside the sphere, the button becomes to be visible. Next, pick another object (for instance some box) and set its distance bigger than button's (if only one object exists in LOD's list, the LOD node is not exported because it would be useless). Now export the scene and wiev the WRL file. There should appear LOD node in the text. In children field you should first find the button object grouped together with the TouchSensor, then another node you picked as the second object - delete this object and put *Group { }* instead. Now copy whole LOD node to the final WRL you got in previous step instead of previos button node. The result is very simple - the button is hidden until user navigates to the node button belong to (or passes by in the near distance).
- Buttons can change their colors when are clicked. To do this, we must add a script node:

```
DEF MyScript Script {
  field SFColor ActiveColor 1 0 1 #this is the color object gets after click
   eventIn SFBool RecolorIn
   eventOut SFColor RecolorOut
   url ["javascript:
      function RecolorIn()
         {RecolorOut=ActiveColor;}
      "]
}
```

and two new routes (we suppose TouchSensor for this button is DEF-named "TS" and material node (note: not the Shape node!) of the button is DEF-named "ButtonMaterial":

```
ROUTE TS.isActive TO MyScript.RecolorIn
ROUTE MyScript.RecolorOut TO ButtonMaterial.diffuseColor
```

When the sensor is clicked, it sends the message to the script and the script generates new event and recolors the button. The script is necessary because of the type conversion - we convert SFBool (boolean value) to SFColor (color RGB representation). One can catch boolean values incoming to RecolorIn function and can change the color of the button only in the moment when the button is actually pressed:

```
field SFColor PassiveColor 0 1 0 #add another field in the header of the node - passive color
function RecolorIn(switch) //message sent twice: TRUE-button down, FALSE-button up
   {
   if (sw==TRUE)
      RecolorOut=ActiveColor;
   else
      RecolorOut=PassiveColor;
   }
```

We can also catch event isOver sent by Sensor, the button would recolor, when the mouse was over it. For instance instead of ROUTE with isActive event just write

```
ROUTE TS.isOver TO MyScript.RecolorIn
```

- Adding periodically moving objects is another quite simple way how to refine the scene. These objects can be lifts, moving stairs and so on. Simply animate the object in 3DS Max and add the TimeSensor from VRML helpers. In sensor's menu check LOOP on check StartOnWorldLoad on, fill-in numbers of the first and the last key of the animation, pick animated object and export the scene. That is all you have to do.
- Synchronising start of animations with periodically moving objects is not so simple, for instance if we need to wait until moment it is possible to get on the lift, we need extra scripts. Every TimeSensor attached to moving object generates lots of events and it is possible to catch them. For instance imagine box moving upside as a floor of an lift and animation, when one gets on it. We need to synchronize start of animation with moment when the floor of the elevator is down. I developed one solution. First we catch an event from TouchSensor and using script bind new viewport and change global boolean field to TRUE. We still have not started any animation. We also catch another event cycleTime sent everytime new loop of TimeSensor, that animates the lift, starts. In this function we test, if user is not waiting (global variable set to true), and in that case we reset the variable and start the animation of the viewport. It means that in the first step there's no movement, perhaps only start button can change its color. In the second step, after sychronisation is reached, is also started animation.

5.6 Adding real views of the scene

VRML2 supports technology QuickTime VR (first introduced by Apple Inc.). It is based on a very simple idea: the user is put in the centre of big sphere, from where he can not move to the surface of the sphere. On the sphere there is painted surrounding area and because it is not displayed in stereovision, it is realistic vision of the scene.
The environment rendered by VRML browser consists of up to six unique pictures, which show surrounding scene from front, left, right, back, top and bottom (it is not necessary to define all six pictures, for instance we can define only four side pictures and leave top and bottom unset. These pictures can be generated using special 3D modelling tools or one can create them using special software and snapshot or videocamera.

- Very important is a resolution of the pictures - it should be perhaps at least 400*400 pixels each, otherwise there is not enough information for the rendering available.
- Also, the pictures must create perfect box and must continue on edges.
- The pictures can not be distorted, spherical transformations are done by browser!
- It may be necessary to rotate top and bottom images 90 degrees CW or CCW or even 180 degrees, because their orientation is undefined. The pictures are, of course, different, if camera looks down from "front" orientation or "left" orientation. The solution of the problem is very simple, every common 2D pictures editor, such as Adobe Photoshop, has the ability to rotate bitmaps.

An example of 3 maps of cubic environment

- It is possible to link these scenes to anchor nodes present at the correct location in the 3D scene - after clicking user can jump to different VRML file and see real picture of the building. We believe that this combination should help much, because 3D scene creates idea of the movement and real pictures create idea of "where I am".

5.7 Adding pictures and texture maps

Important objects should be mapped with textures or at least painted with the different color. Mapping of the surface is done in 3DS Max as normal definition of the object's surface, diffuse map is exported to VRML file. It is also possible to add special partially transparent pictures - it is the only reasonable way how to add flowers or even trees in the scene. For this task we have to have pictures exported in PNG or GIF format with defined transparency. For instance in Adobe Photoshop, transparent colors are set in export plug-in GIF89 export as selected colors of no more than 8-bit palette. In 3DS Max this information is not used for rendering, but in VRML players interpret it correctly. It is also possible to use tiling of the map on an object, so if client computers are powerful, we can snap wall and floor in the building, scan the pictures and map these pictures in the model; because of tiling, these maps can be quite small and then repeated on bigger objects.

6. HTML pages and connection to VRML

We discussed animation, interactivity, graphics and so on. But I still have not mentioned user interface, text information and binding the models together.
From Netscape Navigator 2, there exists possibility of splitting the browser's window into more areas called frames. Each frame can be named and using this name we can later adress it, for instance it is possible to change content of one frame by clicking on a link or button in a different one. That means that main menu or additional text information can be visible even when WRL world or some animation is displayed in the second frame.
Also, there is a programming environment, JavaScript, available. One can do lots of useful things with it (I've developed interactive game with it) and because many functions and properties of the browser are exposed to the script, one can simply do things like changing content in two frames per one mouse click, it is possible to put timers and even build simple multithreading (one task per one frame).
All the communication with the browser is done via special dynamic structure (please consult books about scripting for more information), I now mention only the script for reloading the frame:
First, create a frameset, name the frames and load the files there (shortened)

```
<frameset frameborder="0" framespacing="0" border="0" rows="240,*">
<frame src="fel.wrl" name="VRML" scrolling="no" noresize marginwidth="0" marginheight="0" border="0">
<frame src="text.htm" name="TEXT" marginwidth="10" marginheight="10" border="0">
</frameset>
```

Here we have created two frames with names VRML and TEXT. Now we can load content to the VRML frame from TEXT frame using this script:

```
<script language="javascript">
function loadframe(data)
{parent.VRML.location.href=data;}
</script>
<a href="javascript:loadframe('asdf.htm')">Click to reload</a>
```

I believe this code is quite self-explanating with exception of line parent.VRML.location.href - this is a variable created by browser in its dynamic control structure and defines URL of frame called VRML (of course, there exists also structure for TEXT). We can simply use this technique for loading the VRML scene and contents of its paths in one step.

Bad news is that VRML players do not attach themselves to this structure, that is currently under development I was told. That means that you can do amazing things in frames with html contents, but you can not do so simple things such as starting any animation in VRML scene! The user still has to click once for reload of the frames and second time to start the animation in the VRML scene. Communication can be estabilished only in the opposite direction, that means from VRML to HTML, using this trick: create one frame in the frameset 0 pixels wide (that really works, the frame is then completely invisible; suppose it is called DUMMY). Create HTML documents with this content (shortened; create one document per one message, they probably should differ in the code of function sendmessage()). Let us call this HTML document as mess.htm.

```
<body onload="sendmessage()" language="javascript">
<script language="javascript">
function sendmessage()
{parent.TEXT.ping('HALLO');}
</script>
</body>
```

Add this line to the script in VRML, from where you want to send a message. The browser loads a specified document in a specified frame (this can be also used for loading next part of the VRML scene using sensors):

```
Browser.loadURL (new MFString(mess.htm'), new MFString('target=DUMMY'));
```

It is complicated, but it should work fine: loadURL makes the browser reload content of invisible frame and document newly loaded calls function "ping" with parameter HALLO somewhere in the script of document loaded in TEXT frame. Anyway, it is slow and lots of documents are needed for this communication.

7. Future plans

- Workgroups of VRML team are working on integration of VRML with dynamic HTML scripting model. When this technology is available, it should again decrease the complexity of design and increase interactivity.
- Better integration with spherical environments created from actual shots should be also tested.
- Abilities of VRML players on machines with hardware accelerated graphic should be tested.
- More browsers and VRML players shoud be tested (resulting perhaps in recommendations about scripting languages and commands available in different players).

8. Conclusions

It is possible to build working and useful Local Navigation system using VRML and HTML, although there is a lack of communication between these two systems (that means that actions in VRML can not be started by clicking on a button in different frame). Author believes, that in new versions of VRML this problem will be removed and optimal environment for such tasks will exist. Also, there still exist many differences among browsers and players and it is quite complex to create site compatible with at least MS Internet Explorer and Netscape Communicator. We can wait only, if next versions are more compatible.

9. References and related web sites

[1] Jed Hartmann, Josie Wernecke: The VRML2.0 handbook (Building virtual world on the web); SGI Inc. / Addison Wesley publishing, 1996
[2] VRML2.0 specification: http://www.vrml.org
[3] web links index: http://vrml.miningco.com
[4] Kinetix (3DS Max) home: http://www.ktx.com
[5] Adobe (Photoshop) home: http://www.adobe.com
[6] CosmoSoftware (CosmoPlayer): http://www.cosmosoftware.com
[7] Microsoft (Internet explorer): http://www.microsoft.com/ie
[8] Microsoft (VRML player): http://www.microsoft.com/vrml
[9] Microsoft (scripting documentation): http://www.microsoft.com/scripting
[10] Netscape (Netscape Communicator): http://www.netscape.com
[11] Apple (QuickTime): http://www.apple.com/quicktime/
[12] LivePicture (spherical environment tools): http://www.livepicture.com/

10. Acknowledgements