University of the Study of Naples “PARTHENOPE"
Facoltà di Scienze e Tecnologie
Corso di Laurea Magistrale in Informatica Applicata

Laboratorio di Sistemi Multimediali

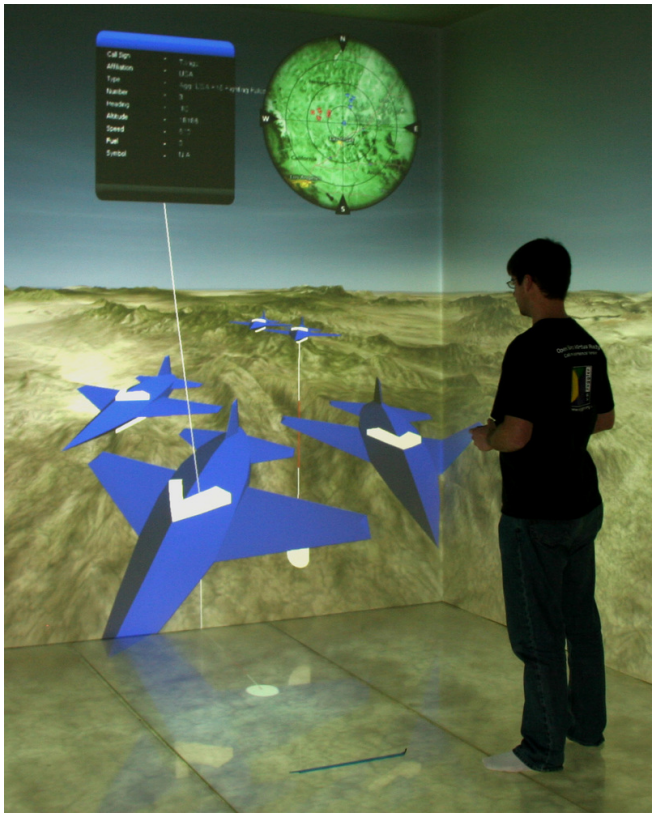
From X3D to X3DOM with Sensors implementation

Prof.
Francesco Camastra
Angelo Ciaramella

Marco Saviano
Matricola: 0120000114

Virtual Reality

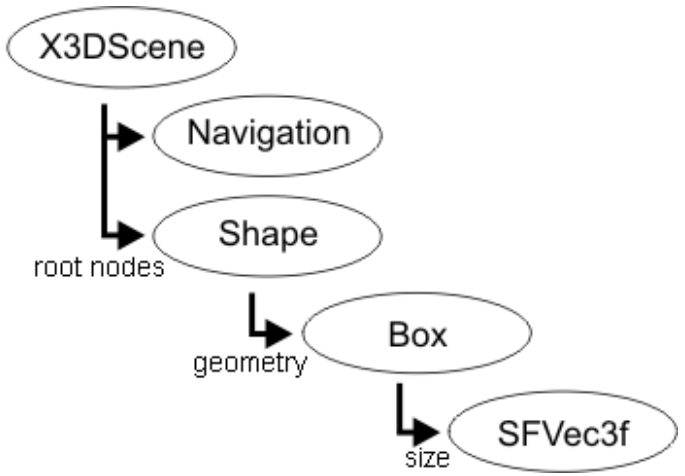
- A computer-simulated environment that can simulate physical presence in places in the real world or imagined worlds using computer



X3D: Extensible 3D

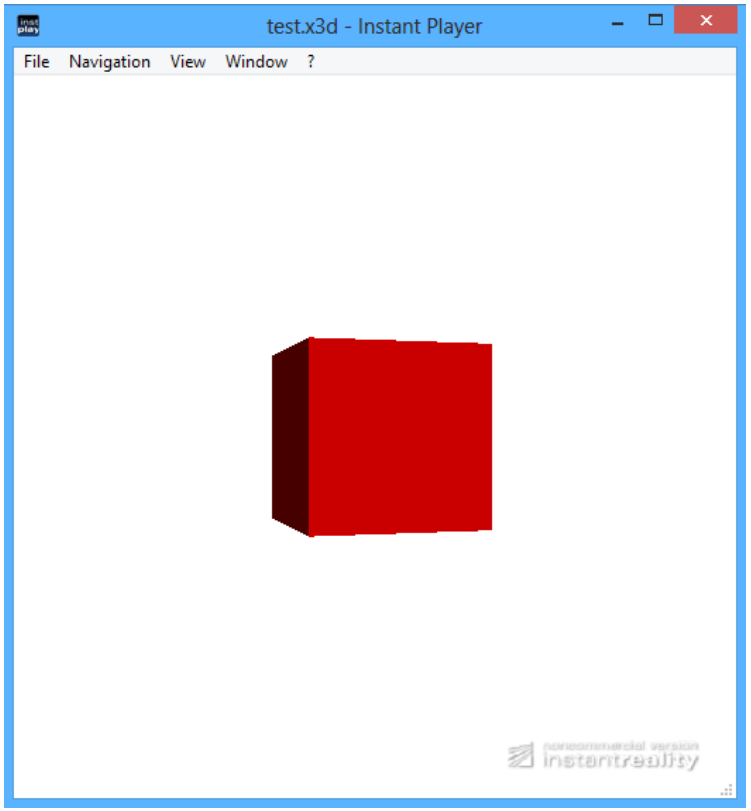
- Royalty-free ISO standard XML-based file format for representing 3D computer graphics, successor to VRML
- Domain components defined by *profiles* (interchange, interactive, immersive and full)
- Defined in MPEG-4 Part 11
- Scene defined by a DAG (*direct acyclic graph)*

```
<Scene>
  <Navigation/>
  <Shape>
    <Box size="3 5 6"/>
  </Shape>
</Scene>
```



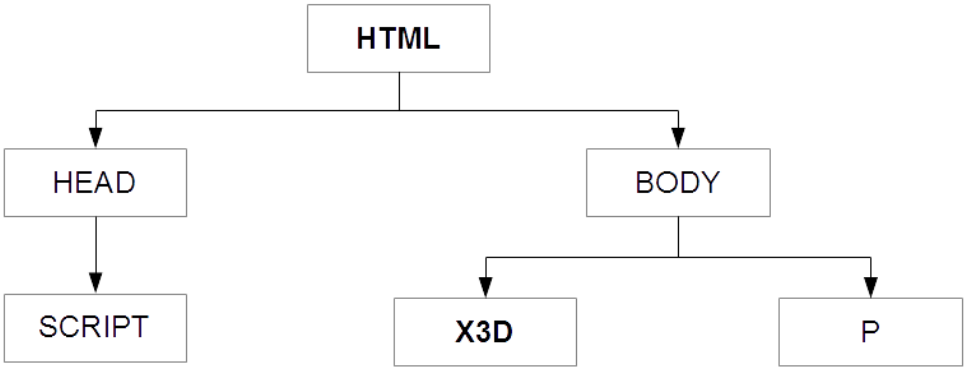
X3D: Extensible 3D
Example

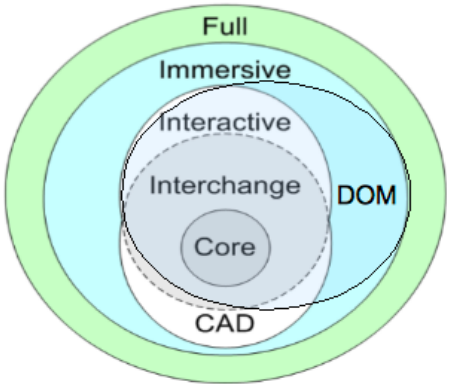
- It needs a player (e.g. Instant Player)



X3DOM

- Experimental open-source framework to integrate HTML5 with X3D
- It allows to insert a X3D scene in the body of a HTML page
- Virtual reality scene in web pages
- No plugins needed
- *HTML profile*

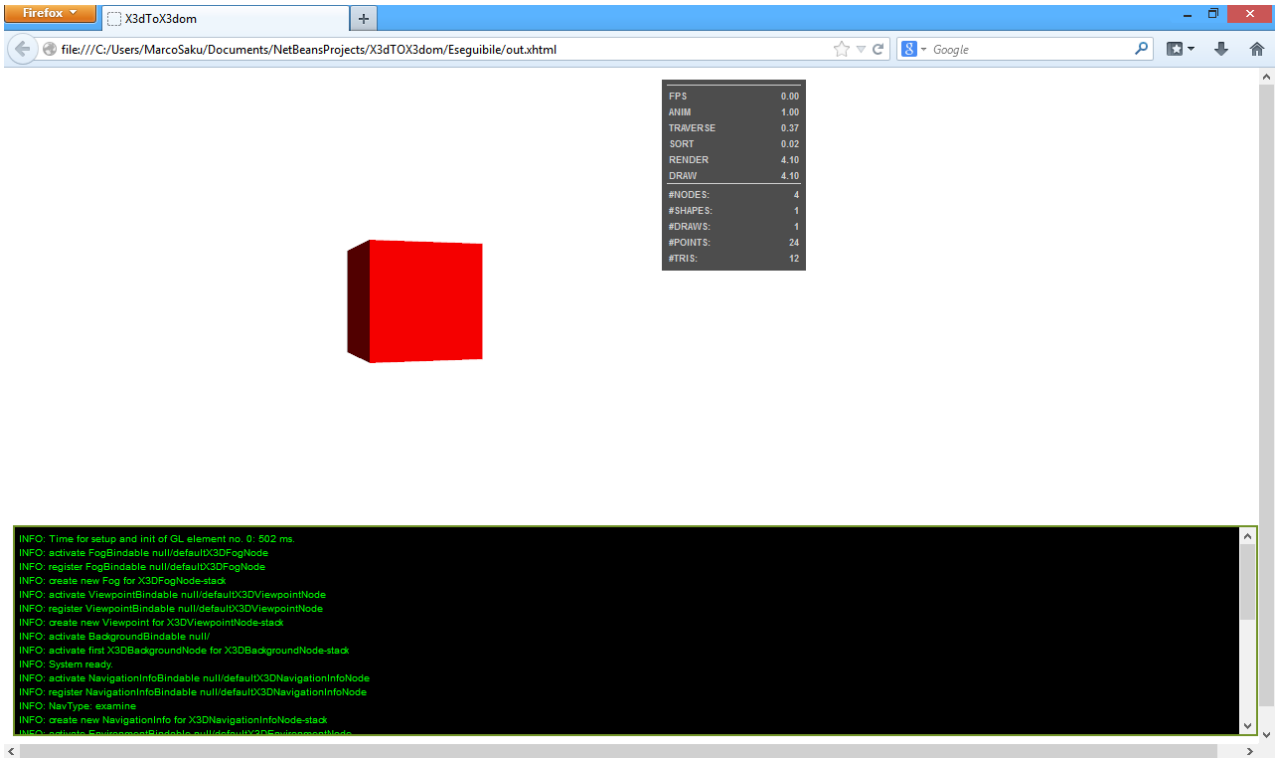




HTML Profile

X3DOM
Example

- Web Browser shows the scene (e.g. Firefox, Google Chrome etc...)
- Android compatible



X3DOM
Comparision with WebGL

- Hierarchical structure that makes simpler manage components
- Metadata for searching and indexing

From X3D to X3DOM
Steps

1. Create a xhtml page and insert the link to X3DOM scripts available on the web

2. Insert X3D scene in the HTML body

X3D-Edit (http://www.web3d.org/x3d/content/README.X3D-Edit.html) does this!

In the web page created by X3d-Edit there is a table that shows supported and unsupported components in X3DOM

From X3D to X3DOM
Unsupported components

- Nurbs
- Humanoid
- Script
- **All sensors to interact with scene through mouse and keyboard**
  TouchSensor, PlaneSensor, SphereSensor, CylinderSensor and StringSensor

Non-implementation of Sensors desired by developers:

use HTML5 events using Javascript!

Sensors in X3D
TouchSensor

It is used to start a TimeSensor by the interaction of the mouse with a shape.

TouchSensor fields:

- **isOver**: mouse cursor on the shape
- **isActive**: left button of mouse pressed on the shape
- **touchTime**: click with left button of mouse on the shape

Sensors in X3D
TouchSensor

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D profile ='Immersive' version='3.1'>
<head>
</head>
<Scene>
<Background skyColor='1 1 1'/>
<Group>
     <Transform DEF= 'Cube'>
     <Shape>
          <Box/>
          <Appearance>
               <Material diffuseColor='1 0 0'/>
          </Appearance>
     </Shape>
     </Transform>
     <TouchSensor DEF='Touch'/>
     <TimeSensor DEF='Clock 'cycle Interval='4 'startTime = '1 'enabled = 'false' loop ='true' />
     <OrientationInterpolator DEF='CubePath' key='0.0 0.50 1.0' keyValue='0.0 1.0 0.0 0.0 ,0.0 1.0
0.0 3.14,0.0 1.0 0.0 6.28'/>
</Group>

<ROUTE fromNode="Touch" fromField="isOver " toNode="Clock " toField="set_enabled"/>
<ROUTE fromNode="Clock " fromField="fraction_changed "toNode="CubePath" toField="set_fraction"/>
<ROUTE fromNode="CubePath" fromField="value_changed " toNode="Cube" toField="set_rotation"/>
</Scene>
</X3D>
```

Sensors in X3D
TouchSensor

Components communicate using the ROUTE node:

1. When the field *isOver* of TouchSensor is true (mouse on the Box) the TimeSensor starts
2. TimeSensor starts the OrientationInterpolator
3. OrientationInterpolator sets the *rotation field* of *Transform* Node

In HTML5 use *onmouseover()* event

Sensors in X3DOM
TouchSensor

```
...BODY HTML...

<X3D id='x3dElement' showStat='true' showLog='true' profile='HTML'>
<head>
</head>
<Scene>
<Background skyColor='1 1 1'/>
<Group onmouseover="document.getElementById('timesens').enabled='true';
"onmouseout="document.getElementById('timesens').enabled='false';">
     <Transform DEF= 'Cube'>
     <Shape>
          <Box/>
          <Appearance>
               <Material diffuseColor='1 0 0'/>
          </Appearance>
     </Shape>
     </Transform>
     <TouchSensor DEF= 'Touch'/>
     <TimeSensor id='timesens' DEF='Clock ' cycleInterval='4'startTime='1' enabled='false'
loop='true'/>
     <OrientationInterpolator DEF= 'CubePath' key='0.0 0.50 1.0' keyValue='0.0 1.0 0.0 0.0,0.0 1.0
0.0 3.14, 0.0 1.0 0.0 6.28'/>
</Group>

<ROUTE fromNode="Touch" fromField="isOver" toNode="Clock" toField="set_enabled"/>
<ROUTE fromNode="Clock " fromField="fraction_changed" toNode="CubePath" toField="set_fraction"/>
<ROUTE fromNode="CubePath" fromField="value changed " toNode="Cube" toField="set_rotation"/>
</Scene>
</X3D>

...BODY HTML...
```

Sensors in X3DOM
TouchSensor

1. Assign HTML5 events *onmouseover* and *onmouseup* to Group component

2. Add *id='timesens'* to TimeSensor component

- Full compatibility between HTML5 event and X3D node: with *document.getElementByID()* we can change the value of field *enabled* of X3D Node Timesensor

Sensors in X3DOM
TouchSensor

For the fields **isActive** and **touchTime** is the same procedure, but the HTML5 events are different:

- **isOver**: `onmouseover=document.getElementById('timesens').enabled='true';`

  `onmouseout=document.getElementById('timesens').enabled='false';`

- **isActive**: `onmousedown=document.getElementById('timesens').enabled=true;`

  `onmouseup=document.getElementById('timesens').enabled=false;`

- **touchTime**: `onclick=document.getElementById('timesens').enabled='true';`

Sensors in X3DOM
PlaneSensor

- User moves an object in the scene with drag-and-drop
- Field: **translation_changed**

Sensors in X3DOM
PlaneSensor

```
...BODY HTML...

<X3D id='x3dElement' showStat='true' showLog='true' profile='HTML'>
<head>
</head>
<Scene>
<navigationInfo id="navInfo" type='"EXAMINE" "ANY"' typeParams="-0.4,60,0.5,1.55"></navigationInfo>
<Background skyColor='1 1 1'/>
<Group>
     <Transform DEF= 'Cube' onmousedown="startDragging(this);" onmouseup="stopDragging();"
     onmousemove="mouseMoved(event);">
     <Shape>
          <Box/>
          <Appearance>
               <Material diffuseColor='1 0 0'/>
          </Appearance>
     </Shape>
     </Transform>
<PlaneSensor DEF='Sensor'/>
</Group>
<ROUTE fromNode="Sensor" fromField="translation_changed" toNode="Cube" toField="set_translation"/>
</Scene>
</X3D>

...BODY HTML...
```

P.S. The Functions *startDragging(this), stopDragging(this)* and *mouseMoved(event)* are in a .js file

Sensors in X3DOM
PlaneSensor (PlaneScript.js) 1/3

```
var startDragging = function(transformNode)
{
//disable navigation during dragging
document.getElementById(”navInfo”).setAttribute(”type” , ’”NONE” ’ ) ;
draggedTransformNode = transformNode ;
unsnappedDragPos = new x3dom.fields.SFVec3f.parse(transformNode.getAttribute(”translation”));
var x3dElem = document.getElementById(”x3dElement”);
var vMatInv = x3dElem.runtime.viewMatrix().inverse();
var viewDir = vMatInv.multMatrixVec(new x3dom.fields.SFVec3f(0.0,0.0,-1.0) ) ;
// use the viewer’s up-vector and right-vector
draggingUpVec = vMatInv.multMatrixVec(new x3dom.fields.SFVec3f(0.0,1.0,0.0));
draggingRightVec = viewDir.cross(this.draggingUpVec);
//project a world unit to the screen to get its size in pixels
var x3dElem = document.getElementById(”x3dElement”);
var p1 = x3dElem.runtime.calcCanvasPos(unsnappedDragPos.x,unsnappedDragPos.y,unsnappedDragPos.z);
var p2 = x3dElem.runtime.calcCanvasPos(unsnappedDragPos.x + draggingRightVec.x,unsnappedDragPos.y+
draggingRightVec.y, unsnappedDragPos.z + draggingRightVec.z)

var magnificationFactor=1.0/Math.abs(p1[0]-p2[ 0 ]);
//scale up vector and right vector accordingly
draggingUpVec = draggingUpVec.multiply(magnificationFactor);
draggingRightVec = draggingRightVec.multiply(magnificationFactor);
} ;
```

Sensors in X3DOM
PlaneSensor (PlaneScript.js) 2/3

```
var cellSize=1.0;
var lastMouseX = -1;
var lastMouseY = -1;
var draggedTransformNode = n u l l ;
//vectors in 3D world space,associated to mouse x/y
movement on the screen
var draggingUpVec = n u l l ;
var draggingRightVec = n u l l ;
var unsnappedDragPos = n u l l ;

var mouseMoved = function(event)
{
     var x = event.hasOwnProperty('offsetX')?event.offsetX:event.layerX;
     var y = event.hasOwnProperty('offsetY')?event.offsetY:event.layerY;
     if(lastMouseX === -1)
     {
          lastMouseX = x ;
     }
     if(lastMouseY === -1)
     {
          lastMouseY = y ;
     }
     if(draggedTransformNode)
     {
          dragObject(x-this.lastMouseX , y-this.lastMouseY) ;
     }
     lastMouseX = x ;
     lastMouseY = y ;
};
```

Sensors in X3DOM
PlaneSensor (PlaneScript.js) 3/3

```
var dragObject = function(dx,dy)
{
//scale up vector and right vector accordingly
var offsetUp = draggingUpVec.multiply(-dy );
var offsetRight = draggingRightVec.multiply(dx);

unsnappedDragPos = unsnappedDragPos.add(offsetUp).add(offsetRight);
var snappedDragPos;
draggedTransformNode.setAttribute("translation",unsnappedDragPos.toString());
}

var stopDragging = function()
{
DraggedTransformNode = null;
DraggingUpVec = null;
draggingRightVec = null;
unsnappedDragPos = null ;
//re-enable navigation after dragging
document.getElementById("navInfo").setAttribute("type",'"EXAMINE" "ANY"');
};
```

Sensors in X3DOM
CylinderSensor

- It converts select-and-drag pointer motion into a 3D rotation around the local y-axis
- Field: **rotation_changed**

Sensors in X3DOM
CylinderSensor

```
<X3D id='x3dElement' showStat='true' showLog='true' profile='HTML'  >
<Scene>
<navigationInfo id="navInfo" type='"EXAMINE" "ANY"' typeParams="-0.4, 60,0.5,1.55"></navigationInfo>
<Background skyColor='1 1 1'/>
<Group>
     <Group>
          <Transform DEF= 'Shape1' onmousedown="startRotating(this);" onmouseup="stopRotatingx();"
             onmousemove="mouseMoved(event);">
                <Shape>
                    <Appearance DEF= 'White'>
                        <Material diffuseColor='1 0 0'/>
                    </Appearance>
                    <Box/>
                </Shape>
            </Transform>
            <CylinderSensor DEF= 'Shape1Sensor' />
        </Group>
        <Group>
            <Transform DEF= 'Shape2' onmousedown="startRotating(this);" onmouseup="stopRotatingx();"
             onmousemove="mouseMoved(event);" translation= '2.5 0.0 0.0' >
                <Shape>
                    <Appearance USE= 'White'>
                    </Appearance>
                    <Cone/>
                </Shape>
            </Transform>
            <CylinderSensor DEF= 'Shape2Sensor' />
        </Group>
    </Group>
<ROUTE  fromNode="Shape1Sensor" fromField="rotation_changed" toNode="Shape1" toField="set_rotation"/>
<ROUTE  fromNode="Shape2Sensor" fromField="rotation_changed" toNode="Shape2" toField="set_rotation"/>
</Scene>
</X3D>
```

Sensors in X3DOM
CylinderSensor(CylinderScript.js) 1/3

```
var startRotating = function(transformNode)
{
    //disable navigation during dragging
    document.getElementById("navInfo").setAttribute("type", '"NONE"');

    draggedTransformNode = transformNode;
    unsnappedDragPos = new x3dom.fields.SFVec3f.parse(transformNode.getAttribute("translation"));


    var x3dElem = document.getElementById("x3dElement");
    var vMatInv = x3dElem.runtime.viewMatrix().inverse();
    var viewDir = vMatInv.multMatrixVec(new x3dom.fields.SFVec3f(0.0, 0.0, -1.0));
    //use the viewer's up-vector and right-vector
    draggingUpVec = vMatInv.multMatrixVec(new x3dom.fields.SFVec3f(0.0, 1.0, 0.0));
    draggingRightVec = viewDir.cross(this.draggingUpVec);
    //project a world unit to the screen to get its size in pixels
    var x3dElem = document.getElementById("x3dElement");
    var p1 = x3dElem.runtime.calcCanvasPos(unsnappedDragPos.x, unsnappedDragPos.y,
unsnappedDragPos.z);
    var p2 = x3dElem.runtime.calcCanvasPos(unsnappedDragPos.x + draggingRightVec.x,
            unsnappedDragPos.y + draggingRightVec.y,
            unsnappedDragPos.z + draggingRightVec.z)
    var magnificationFactor = 1.0 / Math.abs(p1[0] - p2[0]);

    //scale up vector and right vector accordingly
    draggingUpVec = draggingUpVec.multiply(magnificationFactor);
    draggingRightVec = draggingRightVec.multiply(magnificationFactor);
};
```

Sensors in X3DOM
CylinderSensor(CylinderScript.js) 2/3

```
var cellSize=1.0;
var lastMouseX = -1;
var lastMouseY = -1;
var draggedTransformNode = n u l l ;
//vectors in 3D world space,associated to mouse x/y
movement on the screen
var draggingUpVec = n u l l ;
var draggingRightVec = n u l l ;
var unsnappedDragPos = n u l l ;

var mouseMoved = function(event)
{
     var x = event.hasOwnProperty('offsetX')?event.offsetX:event.layerX;
     var y = event.hasOwnProperty('offsetY')?event.offsetY:event.layerY;
     if(lastMouseX === -1)
     {
          lastMouseX = x ;
     }
     if(lastMouseY === -1)
     {
          lastMouseY = y ;
     }
     if(draggedTransformNode)
     {
           rotateObjectx(x - this.lastMouseX, y - this.lastMouseY);
     }
     lastMouseX = x ;
     lastMouseY = y ;
};
```

Sensors in X3DOM
CylinderSensor(CylinderScript.js) 3/3

```
var rotateObjectx = function(dx, dy)
{
    //scale up vector and right vector accordingly
    var offsetUp = draggingUpVec.multiply(-dy);
    var offsetRight = draggingRightVec.multiply(dx);
    unsnappedDragPos = unsnappedDragPos.add(offsetUp).add(offsetRight);
    var snappedDragPos;
    var string = unsnappedDragPos.toString();
    //set rotation value
    var numbers = string.split(" ");
    var rotX = "0 1 0 " + numbers[0];
    //get old rotation value
    var oldRotation = draggedTransformNode.getAttribute("rotation");
    var numbersOld = oldRotation.split(" ");
    //if rotation=0
    if (isNaN(numbersOld[3])) {
        numbersOld[3] = 0;
    }
    //add the old rotation to the rotation calculated with mouse movement
    var num = parseFloat(numbers[0]) + parseFloat(numbersOld[3]);
    var rotX = "0 1 0 " + num;
    //set rotation attribute
    draggedTransformNode.setAttribute("rotation", rotX);
}

var stopRotatingx = function()
{
    draggedTransformNode = null;
    draggingUpVec = null;
    draggingRightVec = null;
    unsnappedDragPos = null;
    document.getElementById("navInfo").setAttribute("type", '"EXAMINE" "ANY"');
};
```

Sensors in X3DOM
StringSensor

- It allows to modify the string field of node *Text* through the keyboard
- Field: **string**

Sensors in X3DOM
StringSensor

```
<X3D id='x3dElement' keysEnabled='false' onkeypress="writeText(event);" showStat='true'
showLog='true' profile='HTML'  >
<Scene>
<Background skyColor='1 1 1'/>
<Viewpoint description='Book View' position='-0.02 0.01 6.85'/>
<StringSensor DEF='GenText' deletionAllowed='true' enabled='true'/>
<Transform>
     <Transform translation='0 0 -.1'>
          <Shape>
               <Appearance>
                    <Material diffuseColor='1 1 .6'/>
               </Appearance>
               <Box size='8 1.5 .01'/>
          </Shape>
      </Transform>
      <Transform translation='-3.8 0.2 0'>
        <Shape>
          <Appearance>
            <Material diffuseColor='0 0 1'/>
          </Appearance>
          <Text DEF='DisplayText' id='textKeyboard'>
            <FontStyle justify='"BEGIN" "MIDDLE"' size='0.75'/>
          </Text>
        </Shape>
      </Transform>
      <Script DEF='Converter' url='"converter.js" '>
        <field accessType='inputOnly' name='SFString_MFString' type='SFString'/>
        <field accessType='outputOnly' name='MFString_out' type='MFString'/>
      </Script>
<ROUTE fromField='enteredText' fromNode='GenText' toField='SFString_MFString' toNode='Converter'/>
<ROUTE fromField='MFString_out' fromNode='Converter' toField='string' toNode='DisplayText'/>
</Transform>
</Scene>
</X3D>
```

Sensors in X3DOM
StringSensor(StringSensorScript.js)

```
var charStr = ""
var writeText = function(evt) {
    evt = evt || window.event;
    var charCode = evt.keyCode || evt.which;
    charStr += String.fromCharCode(charCode);
    if (charCode == 8) {
        charStr = charStr.substring(0, charStr.length - 2);
    }
    document.getElementById('textKeyboard').string = charStr;
}
```

Unlike in X3D, with this script is allowed the space and the BackSpace to delete the last character

Sensors in X3DOM
SphereSensor

- Not implemented
- We don't know how the X3D player computes the values for the rotation moving the mouse cursor (e.g. 0.135 0.99 -0.002 0.46)

Software implemented

- It accepts in input a .x3d file and it creates in output a .xhtml file containing a X3DOM scene
- If it is necessary, it adds the HTML5 events
- Written in Java language (composed by 4 classes)
- GUI for simple use by users

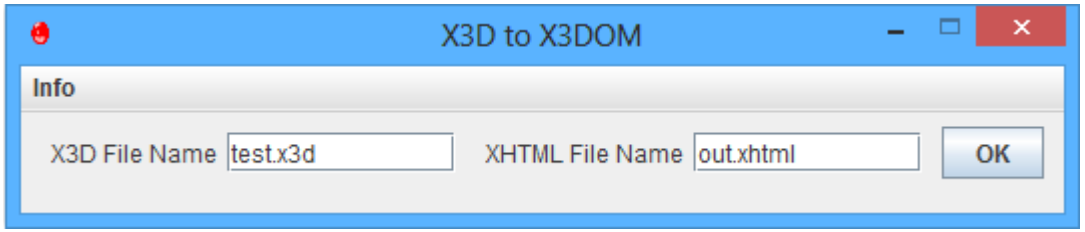
Software implemented
Rules

The software works if in the x3d file the following rules are respected

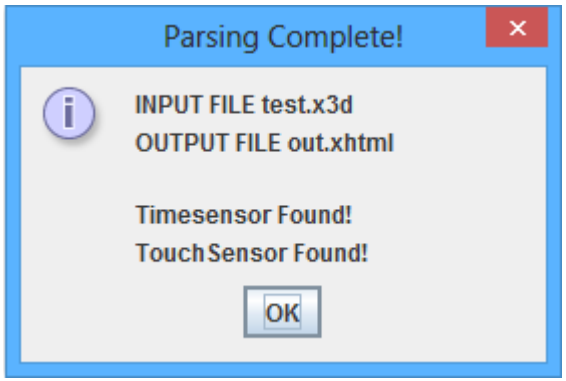
1. The nodes ROUTE are written after the declaration of interested nodes
2. ROUTE node written in a single line
3. No spaces after fromNode= , toNode= (in ROUTE node) and DEF=

Software implemented
Execution example

1. User inserts the input/output file names



2. The pop-up shows to the user the X3D node that it replaces with HTML5 events



References

Virtual Reality Modeling Language
http://www.w3.org/MarkUp/VRML/
Extensible 3D
http://www.web3d.org/realtime-3d/x3d/what-x3d
MPEG-4 parte 11
http://en.wikipedia.org/wiki/MPEG-4_Part_11
X3DOM
http://www.x3dom.org/
HTML5
http://en.wikipedia.org/wiki/HTML5
WebGL
http://www.khronos.org/webgl/wiki/Main_Page
Java
http://www.java.com/

THE END

Thanks for the attention!