Universita' degli Studi di Napoli "Parthenope"
Facolta' di Scienze e Tecnologie
Corso di laurea in Informatica Applicata

Progetto Laboratorio di Sistemi Multimediali

Da X3D a X3DOM con implementazione dei Sensor

Docente
Francesco Camastra
Angelo Ciaramella

Candidato
Marco Saviano
012000114

Anno Accademico 2013-2014

Indice

1 Realtà Virtuale **2**
1.1 X3D 2
1.2 X3DOM 3

2 Da X3D a X3DOM **6**
2.0.1 Sensor in X3D 6

3 Sensori da X3D a X3DOM **9**
3.1 TouchSensor 9
3.2 PlaneSensor 11
3.2.1 PlaneScript.js 12
3.3 CylinderSensor 16
3.3.1 CylinderScript.js 17
3.4 StringSensor 21
3.4.1 StringSensorScript.js 22

4 Software Implementato **23**
4.1 GUI.java 24
4.2 ParserX3D.java 24
4.2.1 parse(String input, String output, GUI gui) 24
4.2.2 writeX3DOMSchema() 24
4.2.3 parseX3DOM() 25
4.2.3.1 replaceTouchSensor(String mouseString) 26
4.2.3.2 replacePlaneCylinder(String mouseString) 28
4.2.3.3 replaceStringSensor() 30
4.2.3.4 checkFile() 31

5 Esempio di esecuzione **34**

Introduzione

Il progetto realizzato consiste nell'implementazione di un software in grado di trasformare una scena X3D in X3DOM.
E' stato aggiunto il supporto a vari sensori inclusi in X3D e non implementati in X3DOM riguardanti l'interazione con la scena utilizzando mouse e tastiera: tali sensori saranno implementati con eventi HTML5.

Nel primo capitolo sarà fatta un'introduzione alla realtà virtuale, a X3D e a X3DOM. Nel secondo si analizza l'integrazione di X3D in X3DOM e come sono gestiti i sensori in X3D. Nel capitolo tre si vede come sono implementati i Sensor di X3D utilizzando gli eventi HTML5. Capitolo 4 spiega nel dettaglio il software implementato e nell'ultimo capitolo c'è un esempio di esecuzione del software.

Capitolo 1

Realtà Virtuale

Il termine realtà virtuale è applicato solitamente a qualsiasi tipo di simulazione virtuale creata attraverso l'uso del computer, dai videogiochi che vengono visualizzati su un normale schermo, alle applicazioni che richiedono l'uso degli appositi guanti muniti di sensori (wired gloves) e infine al World Wide Web.
La realtà virtuale, per sua stessa definizione, simula la realtà effettiva. L'avanzamento delle tecnologie informatiche permette di navigare in ambientazioni fotorealistiche in tempo reale, interagendo con gli oggetti presenti in esse.
Uno dei linguaggi più diffusi per l'implementazione della realtà virtuale è VRML[1] e una sua evoluzione è X3D[2].
Nel seguito presentiamo brevemente il linguaggio X3D e la sua estensione per il funzionamento con HTML5[7]: X3DOM[6]

1.1 X3D

X3D è un linguaggio per la descrizione di ambienti virtuali interattivi. È stato sviluppato dal Web 3D Consortium come evoluzione del VRML, è basato su XML, è un formato non proprietario ed è stato standardizzato dall'ISO nel 2004. Fa parte dello standard MPEG-4 parte 11[3].
Con X3D è possibile rappresentare fedelmente una scena reale: infatti possiamo implementare oggetti sia 2D che 3D, animazioni, audio spazializzato, umanoidi, rendere la scena dinamica tramite script, navigare all'interno e interagire con la scena stessa grazie a mouse e tastiera.
La scena creata all'interno del file X3D è un DAG (*direct acyclic graph*). Come struttura dati viene utilizzato un file .xml che ben si presta nello strutturare i

nodi di un grafo.
Nell'immagine seguente troviamo sulla sinistra una parte del file xml contente la scena x3d e sulla destra il relativo DAG

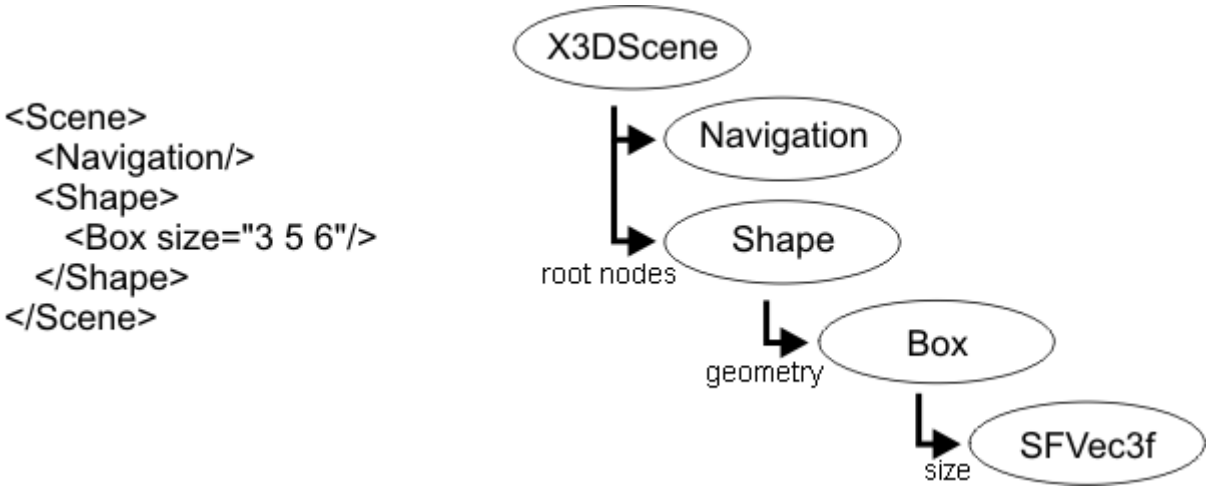

Figura 1.1: Grafo della scena X3D

Gli oggetti in X3D sono raggruppati in *componenti* (components). Il dominio dei componenti che possiamo utilizzare in una scena X3D è definito dal *profilo*(profile) utilizzato: sono definiti i profili interchange, interactive, immersive e full. Full comprenderà tutte le componenti di X3D (es. nurbs) ma renderà il caricamento della scena più lento proprio perché deve caricare tutti i componenti: il programmatore definisce il profilo giusto in base ai componenti richiesti dalla scena.
Per poter visualizzare una scena X3D contenuta in un file xml abbiamo bisogno di un player. Esistono molti player disponibili in rete sia free che non e uno di questo è InstantReality[4].

1.2 X3DOM

X3DOM (pronunciato X-Freedom) è un framework open-source in continua evoluzione per l'integrazione di HTML5 con X3D: esso permette di inserire una scena X3D in una pagina html (o xhtml) includendola nell'albero DOM di HTML5.

L'obiettivo è quello di avere una scena di realtà virtuale in una pagina web(web application) la quale può essere manipolata semplicemente inserendo ed eliminan-

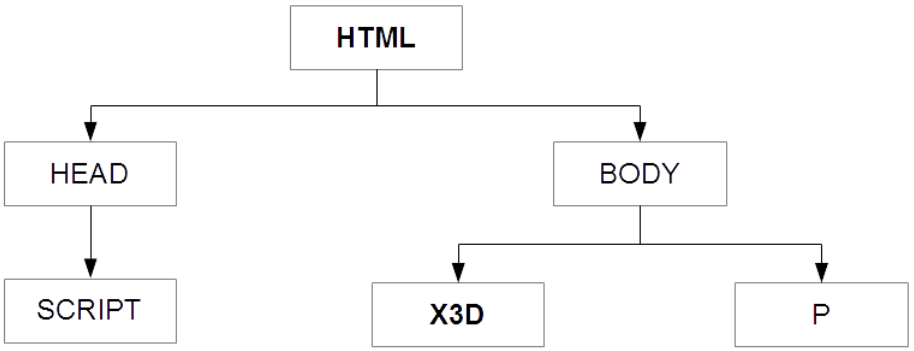

Figura 1.2: Rappresentazione ad albero del DOM di un documento X3DOM

do elementi dell'albero DOM (html). Non sono necessari plugin aggiuntivi ed è possibile utilizzare gli eventi DOM sugli oggetti grafici (e.x. onclick).
Il framework X3DOM è in continua evoluzione e il fine ultimo è quello di una perfetta integrazione tra HTML5 e X3D proprio come lo è ora SVG. Esistono alternative ad X3DOM come WebGL[8]: la natura gerarchica di X3D e la facilità di unire più oggetti nella stessa scena fa si che quest'ultimo possa essere preferito ad un linguaggio procedurale come WebGL. Inoltre rispetto a WebGL è possibile utilizzare i metadati per operazioni di indicizzazione e ricerca.
La visualizzazione di scene X3D tramite X3DOM è pienamente supportata anche da dispositivi mobile Android.

E' possibile integrare una scena X3D in un file html semplicemente aggiungendo degli script .js e .css disponibili in rete. Fatto ciò, aggiungiamo la scena x3d all'interno del body dell'html.
X3DOM utilizza un particolare profilo chiamato 'HTML', estensione del profilo Interchange. Esso non include alcun supporto a Script e Prototypes quindi tutte le operazioni di scripting sugli elementi dell'albero DOM devono essere effettuati dal lato HTML tramite JavaScript o jQuery. Nell'immagine seguente vediamo la relazione tra i profili X3D e il profilo proposto per X3DOM

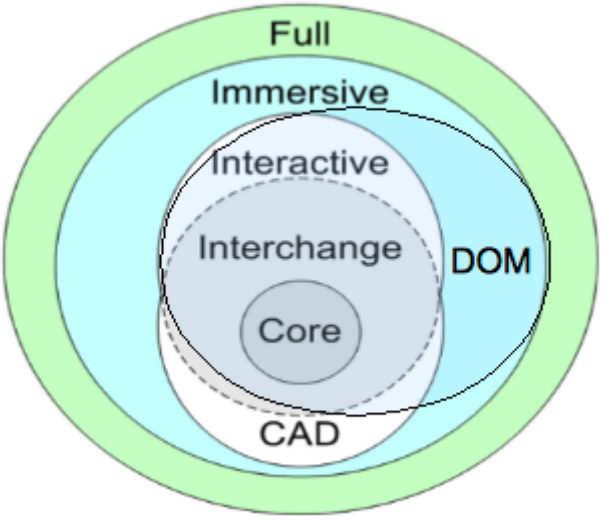

Figura 1.3: Profilo HTML

Capitolo 2

Da X3D a X3DOM

Come detto nel capitolo precedente, per creare una scena X3DOM bisogna inserire una scena X3D all'interno di un documento HTML. Le operazioni da fare sono le seguenti:

- Inserire i collegamenti agli script X3DOM disponibili in rete
- Inserire la scena X3D nel body del documento HTML

Esiste un software free in grado di fare le operazioni citate: X3D-Edit[5]. Il software crea un file xhtml: pagina HTML scritta in conformità con lo standard XML contenente la scena X3DOM visualizzabile tramite un moderno browser web (Firefox, Chrome etc.). Visualizzando la scena, oltre a visualizzare gli oggetti grafici, X3D-Edit crea una tabella nella quale sono inseriti quali nodi presenti nella scena X3D sono supportati e non da X3DOM.
Spiccano tra i nodi non supportati da X3DOM i vari *Sensor* come TouchSensor, StringSensor, PlaneSensor e CylinderSensor. Questi sensori permettono l'interazione dell'utente con la scena attraverso l'uso di mouse e tastiera. Gli sviluppatori di X3DOM hanno dichiarato che la non implementazione di questi sensori è voluta: tali eventi devono essere gestiti tramite Javascript/HTML.
Il software che verrà presentato farà proprio questo: sostituisce i Sensor di X3D con gli eventi HTML5 scritti in Javascript.

2.0.1 Sensor in X3D

Vediamo ora il funzionamento dei Sensor relativi a mouse e tastiera all'interno di X3D. Di seguito vediamo un xml contenente una scena X3D che rappresenta

un cubo rosso: quando il cursore del mouse si trova sul cubo esso ruota. Ogni nodo nella scena X3D può essere identificato da un identificativo usando DEF ed è costituito da uno o più campi (field)

```
<?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'
                          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>
```

Codice 2.1: X3D: TouchSensor con field isOver

E' presente il sensore TouchSensor il quale gestisce le operazioni fatte tramite il mouse. Tale sensore è associato al nodo Transform il quale a livello gerarchico è più alto rispetto a Cube (il quale non sa niente riguardo la sua posizione nella scena). In X3D, per far "comunicare" due nodi utilizziamo il nodo *ROUTE*: come vediamo nel primo dei tre nodi ROUTE quando dal nodo Touch (touchsensor) abbiamo il campo *isOver* attivo (cursore sulla figura), il campo *set_enabled* del nodo Clock (TimeSensor) viene impostato a true. In questo caso quindi il cursore del mouse spostandosi sulla figura avvia il TimeSensor, il quale a sua volta attiva il nodo OrientationInterpolator che setta il camp *rotation* del nodo Cube(Transform).
In definitiva l'interazione con la scena utilizzando mouse e tastiera fa si che la modifica di un campo associato al nodo di tipo Sensor vada a modificare i campi di altri nodi presenti nella scena utilizzando il nodo ROUTE.
Come detto in precedenza in X3DOM il nodo TouchSensor non è stato implementato, quindi per gestire l'evento del mouse dobbiamo utilizzare l'evento HTML *onmouseover()*

Capitolo 3

Sensori da X3D a X3DOM

In questo capitolo vediamo come si possano implementare i Sensori di X3D in X3DOM utilizzando gli eventi di HTML5.
Negli esempi troveremo il codice relativo alla scena in X3DOM: la scena X3D è identica eliminando gli eventi HTML5.

3.1 TouchSensor

Il TouchSensor come visto nell'esempio del capitolo precedente gestisce l'interazione attraverso il mouse. Quest'ultimo attiverà un TimeSensor (sempre tramite ROUTE) il quale attiverà altri eventi in altri nodi della scena (e.x. Transform). Il nostro obiettivo è quindi quello di settare il campo di TimeSensor set_enabled=true utilizzando gli eventi HTML5.
Di seguito i campi relativi a TouchSensor:

- **isOver**: cursore del mouse sulla figura
- **touchTime**: click del mouse sulla figura
- **isActive**: tasto premuto sulla figura

Per poter attivare il TimeSensor, per prima cosa dobbiamo assegnargli un *id*: grazie ad esso possiamo richiamarlo all'interno degli eventi HTML5. Di seguito vediamo come viene implementato in X3DOM la scena X3D presentata in Code 2.1

```
...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  />
                  </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...

Codice 3.1: X3DOM: TouchSensor con field isOver

Anche se non riconosciuto, possiamo lasciare il nodo TouchSensor senza che esso dia errori. Vediamo che gli eventi HTML5 onmouseover e onmouseup sono stati assegnati al nodo Group: in questo modo sono associati a tutti i nodi inclusi in esso gli eventi del mouse (in questo caso a Box). Notiamo la piena compatibilità tra gli eventi DOM di HTML5 con i nodi X3D: facilmente richiamiamo il nodo TimeSensor con document.getElementById passando il suo id='timesens' e settiamo il campo a true/false.
Lo stesso procedimento viene effettuato per i campi touchTime e isActive a differenza dell'evento HTML5 associato alla specifica operazione del mouse. Di seguito vediamo quindi come sono associati i campi di TouchSensor e gli eventi HTML5

- **isOver**: onmouseover=document.getElementById('timesens').enabled='true'; onmouseout=document.getElementById('timesens').enabled='false';
- **touchTime**: onclick=document.getElementById('timesens').enabled='true';
- **isActive**: onmousedown=document.getElementById('timesens').enabled=true; onmouseup=document.getElementById('timesens').enabled=false;

3.2 PlaneSensor

PlaneSensor permette di trascinare un oggetto all'interno della scena: il campo di nostro interesse è translation_changed. X3D utilizza sempre il nodo ROUTE per associare il campo del PlaneSensor al campo translation di un nodo Transform. Di seguito vediamo il codice di una scena contenente il PlaneSensor direttamente in X3DOM. Non avendo cancellato nulla (nemmeno PlaneSensor) in X3D è uguale tralasciando gli eventi HTML5 relativi al mouse

```
...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 />
                   </Appearance>
               </Shape>
            </Transform>
            <PlaneSensor DEF= 'Sensor' />
        </Group>
        <ROUTE fromNode="Sensor" fromField="
           translation_changed" toNode="Cube" toField="
           set_translation"/>
    </Scene>
            </X3D>
            ...BODY HTML...
```

Codice 3.2: X3DOM: PlaneSensor in X3DOM

Gli eventi HTML5 usati sono *onmousedown*, *onmouseup* e *onmouseremove*: ad ognuno di essi è associato una function in Javascript. Il codice delle function è contenuto nel file "PlaneSensor.js". A differenza del TouchSensor, dove associavamo gli eventi HTML5 al nodo Group che conteneva il TimeSensor, in questo caso associamo gli eventi HTML5 direttamente al nodo definito in ROUTE (Cube).

3.2.1 PlaneScript.js

Vediamo ora le function presenti in PlaneSensor.js

```
var cellSize = 1.0;
var lastMouseX = -1;
var lastMouseY = -1;
```

```
var draggedTransformNode = null;
//vectors in 3D world space, associated to mouse x/y
   movement on the screen
var draggingUpVec = null;
var draggingRightVec = null;
var unsnappedDragPos = null;
//-------------------------------------------------------------------

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;
};

//-------------------------------------------------------------------


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"));

    //compute the dragging vectors in world coordinates
    //(since navigation is disabled, those will not change
        until dragging has been finished)

    //get the viewer's 3D local frame
    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);
};

//-----------------------------------------------------------
```

```
var dragObject = function(dx, dy)
{
    //scale up vector and right vector accordingly
    var offsetUp = draggingUpVec.multiply(-dy);
    var offsetRight = draggingRightVec.multiply(dx);
    //document.getElementById("mouseOverEvent").innerHTML
       ="Mouse right:"+offsetRight;
    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"');
};
```

Codice 3.3: PlaneScript.js

Al metodo *startDragging* viene passato come parametro il nodo Transform ed inizializzate delle variabili, in *mouseMoved* viene calcolata la posizione del mouse e chiamata la function *dragObject* passando come parametro la distanza del punto in cui abbiamo effettuato il primo click del mouse e la posizione attuale del cursore. In dragObject viene settato il campo translation del nodo passato come input a startDragging (transform): in questo modo ad ogni movimento del mouse viene settata la translazione dell'oggetto dando la "sensazione" che si stesse muovendo. Infine quando rilasciamo il tasto del mouse viene chiamata *stopDragging* la quale setta null le variabili utilizzate e riattiva la navigazione all'interno della scena.

3.3 CylinderSensor

CylinderSensor permette di ruotare l'oggetto grafico sull'asse y tenendo premuto il tasto del mouse e muovendo il cursore. Il campo di nostro interesse è *rotation_changed*. Come prima, vediamo qui la scena X3DOM contenuta nel body del file xhtml.

```
...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>
          <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>
...BODY HTML...
```

Codice 3.4: X3DOM: CylinderSensor

Il funzionamento è identico a quello del PlaneSensor, cambiano solo le functions JavaScript richiamate (presenti in CylinderScript.js).

3.3.1 CylinderScript.js

Di seguito troviamo le functions utilizzate per l'implementazione del CylinderSensor tramite eventi HTML5

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


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;
};
//-----------------------------------------------------------------------------


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
                "));

    //compute the dragging vectors in world coordinates
    //(since navigation is disabled, those will not change
        until dragging has been finished)
    //get the viewer's 3D local frame
    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);
};

//-----------------------------------------------------------------------------


var rotateObjectx = function(dx, dy)
{
    //scale up vector and right vector accordingly
    var offsetUp = draggingUpVec.multiply(-dy);
    var offsetRight = draggingRightVec.multiply(dx);
    //document.getElementById("mouseOverEvent").innerHTML
       ="Mouse right:"+offsetRight;
    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;
    //re-enable navigation after dragging
    document.getElementById("navInfo").setAttribute("type
       ", '"EXAMINE" "ANY"');
};
```

Codice 3.5: CylinderScript.js

Le functions sono quasi identiche a quelle in PlaneSensor.js (sono stati modificati i nomi). Quello che troviamo di differente è in rotateObjectx: qui si somma alla rotazione attuale del nodo Transform rispetto l'asse y la rotazione calcolata in base al movimento del cursore sulla figura. Alla fine viene settato il campo *rotation* di Transform

3.4 StringSensor

Un altro modo di interazione tra utente e scena X3D è dato dall'utilizzo della tastiera per inserire dinamicamente delle stringhe di testo: questo viene implementato in X3D con StringSensor. Il campo utilizzato è *string*.

```
...BODY HTML...
<X3D id='x3dElement' keysEnabled='false' onkeypress="
    writeText(event);" showStat='true' showLog='true'
    profile='HTML' >
  <head>
  </head>
  <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 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>
      <!-- convert StringSensor SFString into Text node
         MFString by using explicit typecasting in
         ECMAScript -->
      <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'/>
  </Scene>
```

```
</X3D>
...BODY HTML...
```

Codice 3.6: X3DOM: StringSensor

In X3D la gestione dello StringSensor è fatta tramite gli Script (anch'essi assenti in X3DOM). In X3DOM è stato aggiunto l'evento HTML5 *onkeypress* al nodo X3D (all'intera scena) ed è stato assegnato l'id='textKeyboard' al nodo Text. La function chiamata dall'evento è presente in StringSensorScript.js. Per poter inserire una stringa di testo in runtime, bisogna prima dare il focus tramite mouse alla scena X3D presente nella pagina e poi iniziare a scrivere tramite tastiera.

3.4.1 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;
}
```

Codice 3.7: StringSensorScript.js

In aggiunta allo StringSensor di X3D, con questo evento è possibile cancellare il testo tramite Backspace ed è abilitato lo spazio.

Capitolo 4

Software Implementato

In questo capitolo vediamo come funziona il software implementato scritto nel linguaggio Java[9].
Può essere diviso in due fasi: preso in input un file .x3d, nella prima fase viene creato un file xhtml contenente lo scheletro per l'implementazione di X3DOM (inclusione di script .css e .js), nella seconda viene scritto il codice della scena X3D all'interno del body dell'xhtml. In questa fase viene effettuata un'operazione di parsing per sostituire i Sensor non implementati in X3D con gli eventi HTML5 come visto nel capitolo precedente.

Nel file X3dToX3dom.java è presente il main del progetto: da qui viene invocato il metodo *startGUI()* che crea l'interfaccia grafica per poter utilizzare facilmente il software. La classe ReadReverse contiene metodi che permettono la lettura in modalità inversa di un file. Nel seguito vengono descritti i file .java che compongono il progetto insieme a X3dToX3dom.java e ReadReverse.java: GUI.java, ParserX3D.java e ReadReverse.java.
Prima però vediamo quali regole dobbiamo rispettare nella scrittura del file .x3d di input affinché il software possa funzionare

- Inserire i nodi ROUTE dopo aver dichiarato i nodi interessati
- Scrivere il nodo ROUTE su una sola linea
- Dopo fromNode=, toNode= (nel nodo ROUTE) e DEF= NON devono essere presenti spazi

- Utilizzare il nodo Group per raggruppare gli oggetti
- Nella dichiarazione di un nodo, non lasciare spazio dopo < (es <Transform)

4.1 GUI.java

L'unico metodo presente è startGUI() che viene invocato dal main. In questo metodo viene creata una semplice interfaccia grafica utilizzando le librerie Java Swing. Inseriti nei box il file .x3d di input e il file .xhtml di output, alla pressione del button OK viene invocato il metodo *parse()* della classe ParserX3D passando come parametro i nomi dei file input/output e l'oggetto della classe GUI (per poter visualizzare messaggi di info o errori tramite il frame creato).

4.2 ParserX3D.java

Esso contiene il cuore del software cioè il parsing del file .x3d e la creazione del file .xhtml. Viene prima effettuata una prima fase di parsing e scritto nel file temp, dopo una seconda fase viene scritto il definitivo file di output col nome outputFile. Vediamo nel dettaglio i metodi presenti

4.2.1 parse(String input, String output, GUI gui)

Presi i parametri passati, inizializza i suoi attributi private inputFile, outputFile, gui. Elimina se presente il file col nome uguale a outputFile e il file temp. Come struttura dati per la lettura, scrittura e parsing del file viene utilizzato il RandomAccessFile in quanto esso predispone del metodo *seek(pointer)* per potersi muovere tra le righe del file. Vengono chiamati in sequenza i metodi della stessa classe *writeX3DOMSchema()*, *parseX3DOM()* e *checkFile()*

4.2.2 writeX3DOMSchema()

Viene creato lo scheletro X3DOM all'interno del file temp. E' creato l'head del file xhtml contenente il link agli script necessari per l'implementazione di X3DOM

presenti in rete. Viene definito inoltre lo stile css per visualizzare correttamente la scena tramite browser web (es. Firefox).

4.2.3 parseX3DOM()

Viene effettuato il parsing del file di input .x3d e scritto il file temp. Quello che facciamo è leggere ogni linea del file .x3d (tranne la prima) e controllare se ci siano delle parole chiave di nostro interesse. La linea corrente viene trasformata tutta al maiuscolo. Come abbiamo visto in precedenza, la comunicazione tra il Sensor e un altro nodo della scena viene effettuata tramite il nodo ROUTE: è proprio in questo nodo che possiamo trovare le stringhe di nostro interesse.

- “fromField=’isActive’ ” “toField=’enabled’ ”: possibile presenza di un TouchSensor che attiva un TimeSensor tramite il campo isActive. Viene chiamato il metodo *replaceTouchSensor* (spiegato in seguito)
- “fromField=’isOver’ “ “toField=’set_enabled’ “: anche qui possibile presenza di un TouchSensor che attiva un TimeSensor tramite il campo isOver. Chiamato anche qui il metodo replaceTouchSensor
- “fromField=’touchTime’ “ “toField=’set_enabled’ “: possibile TouchSensor che attiva un TimeSensor tramite il campo TouchTime. Chiamato il metodo replaceTouchSensor
- “fromField=’translation_changed’ “ “toField=’set_translation’ “: possibile PlaneSensor che modifica il campo translation di un nodo Transform. Chiamato il metodo *replacePlaneCylinder* (spiegato in seguito)
- “fromField=’rotation_changed’ “ “toField=’set_rotation’ “: possibile CylinderSensor che modifica il campo rotation di un nodo Transform. Chiamato il metodo replacePlaneCylinder()
- “toField=“string” “: possibile StringSensor. Chiamato il metodo *replaceStringSensor* (spiegato in seguito)

Inoltre quando troviamo un nodo TimeSensor, dobbiamo aggiungere l’id utile all’evento HTML5 tramite il metodo setTimeSensor().

4.2.3.1 replaceTouchSensor(String mouseString)

Viene passato come parametro al metodo la stringa contenente l'evento HTML5 relativo al campo del TouchSensor individuato. Vengono estratti gli identificativi del fromNode e del toNode dal nodo ROUTE nel seguente modo

```
String fromNode = null; //String in "fromNode"
Pattern pattern = Pattern.compile("fromNode=\"(.*?)\"");
Matcher matcher = pattern.matcher(line);
while (matcher.find()) {
    fromNode = matcher.group(1);
}
```

Codice 4.1: Estrazione fromNode

La stessa operazione viene ripetuta considerando il caso che l'identificativo del nodo sia tra ' '. Nello stesso modo viene estratto l'identificativo di toNode. Cerchiamo all'interno del file, partendo dall'inizio, il nodo corrispondente all'identificativo fromNode. Se esso sarà un nodo di tipo TouchSensor, inseriamo nel nodo Group che lo contiene gli eventi HTML5 passati come parametro al metodo.

```
file.seek(0);
//read file from start
while ((line2 = file.readLine()) != null) {
    pattern = Pattern.compile("'(.*?)'");
    matcher = pattern.matcher(line2);
    while (matcher.find()) {
        strTemp = matcher.group(1);
        if (strTemp.equals(fromNode)) {
            break;
        }
    }

//if node with FromNode DEF is found
if (strTemp.equals(fromNode)) {
    //if we found TouchSensor node (nodeName=TouchSensor)
    if (line2.contains(nodeName)) {
        //read in reverse mode temp through readReverse
        while (((line = readRev.read(file, file.
            getFilePointer())).contains("<X3D id='
            x3dElement'") == false)) {
            if (line.toUpperCase().contains("<ROUTE") ==
                false) {
```

```
                //add mouse string into node <Group
                if (line.toUpperCase().contains("<GROUP"))
                    {
                                //message in GUI ("
                                   TouchSensor Found!")
                    gui.addLog(nodeName + " Found!\n");
                                        //create new line
                                           for <Group
                    StringBuilder str1 = new StringBuilder
                       (line);
                    str1.insert(line.toUpperCase().indexOf
                       ("<GROUP") + 6, mouseString);
                    str1.insert(str1.length(), "\n");
                    str1.insert(0, "\n");
                    buffPos = readRev.getBufPos();
                    file.seek(buffPos);
                    //replace old Node with blank ''
                    byte[] blankArray = new byte[line.
                       length() + 1];
                    Arrays.fill(blankArray, blankByte);
                    file.write(blankArray);
                    //moveText moves the text to create
                       the space to insert string into out
                       .xhtml
                    moveText(buffPos, str1.toString().
                       getBytes().length, (int) (file.
                       length() - file.getFilePointer()));
                    file.seek(buffPos);
                                        //write new Group
                                           node line
                    file.write(str1.toString().getBytes())
                       ;

                    exit = true;
                    break;

                }
            }
            if (exit == true) {
                break;
            }
```

```
            }
        }
}
```

Codice 4.2: Inserimento evento TouchSensor

4.2.3.2 replacePlaneCylinder(String mouseString)

Viene passato come parametro la stringa contenente gli eventi HTML5 da aggiungere alla scena. Viene chiamato sia per impostare il PlaneSensor sia per il CylinderSensor. Nello stesso modo dei metodi precedenti vengono estratti gli identificativi di fromNode e toNode. Si parte dall'inizio del file e si cerca il tipo di nodo con identificativo fromNode. Se esso è un CylinderSensor/PlaneSensor si inserisce all'interno del nodo identificato da toNode la stringa passata come input al metodo contenente gli eventi HTML5

```
while ((line2 = file.readLine()) != null) {
    //if node with FromNode DEF is found
    if (line2.indexOf("'" + fromNode + "'") != -1) {
        //if PlaneSensor or CylinderSensor node is found
        if (line2.toUpperCase().contains(nodeName.
            toUpperCase())) {
            //read in reverse mode out.xhtml through
                readReverse
            while (((line = readRev.read(file, file.
                getFilePointer())).contains("&lt;X3D id='
                x3dElement'") == false) && groupFound ==
                false) {
                if (line.contains("ROUTE") == false) {
                    toNodeFound = false;
                    pattern = null;
                    matcher = null;
                    pattern = Pattern.compile("\"(.*?)\"")
                        ;
                    matcher = pattern.matcher(line);
                    //find toNode between ' ' or " "
                    while (matcher.find()) {
                        strTemp = matcher.group(1);
                        if (strTemp.equals(toNode)) {
                            toNodeFound = true;
                        }
```

```
                    }
                    if (toNodeFound == false) {
                        pattern = Pattern.compile("'(.*?)'
                            ");
                        matcher = pattern.matcher(line);
                        while (matcher.find()) {
                            strTemp = matcher.group(1);
                            //System.out.println("temp "+
                                strTemp);
                            if (strTemp.equals(toNode)) {
                                toNodeFound = true;

                            }
                        }
                    }
                }
                //if DEF="ToNode" found. Usually it is a
                    Transform node
                if (toNodeFound) {
                    //Message from GUI ("PlaneSensor/
                        CylinderSensor is found!")
                    gui.addLog(nodeName + " Found!\n");
                    //add HTML5 event to node
                    StringBuilder str1 = new StringBuilder
                        (line);
                    str1.insert(line.indexOf(toNode) +
                        toNode.length() + 1, mouseString);
                    str1.insert(str1.length(), "\n");
                    str1.insert(0, "\n");
                    buffPos = readRev.getBufPos();
                    file.seek(buffPos);

                    //replace old Node with blank ''
                    byte[] blankArray = new byte[line.
                        length() + 1];
                    Arrays.fill(blankArray, blankByte);
                    file.write(blankArray);
                    //moveText moves the text to create
                        the space to insert string into out
                        .xhtml
                    moveText(buffPos, str1.toString().
                        getBytes().length, (int) (file.
```

```
                        length() - file.getFilePointer()));
                    file.seek(buffPos);
                    //write new line Node with HTML5 event
                    file.write(str1.toString().getBytes())
                        ;

                    groupFound = true;
                    exit = true;
                    break;

                }
                if (exit == true) {
                    break;
                }

            }
        }
    }
    if (exit == true) {
        break;
    }
}
```

Codice 4.3: Inserimento evento PlaneSensor/CylinderSensor

4.2.3.3 replaceStringSensor()

Funziona sulla falsa riga dei due metodi precedenti. Individuato il campo "string" in fromField, controlla nel file se è presente un nodo Text con l'identificativo preso da fromNode. L'evento HTML5 è associato al nodo X3D (intera scena) e verrà scritto nella fase finale del parsing nel metodo *checkFile*. Nel nodo X3D disattiveremo le azioni sulla scena tramite tastiera (es. 'r' reset view) per poter scrivere liberamente con keysEnabled='false'

```
file.seek(0);
boolean stringSensorFound=false;
while ((line = file.readLine()) != null) {
    //when we find out the StringSensor node
    if (line.toUpperCase().contains("&lt;STRINGSENSOR")) {
        stringSensorFound=true;
    }
```

```
}
file.seek(0);
textStringSensor = "DEF='" + toNode + "'";
while ((line = file.readLine()) != null) {
    //when we find out the Text node
    if (line.contains(textStringSensor)) {
        System.out.println(line);
        //event written in the node &lt; X3D in checkFile
           ()
        writeStringSensor = true;
        //message from the GUI ("StringSensor found!")
        gui.addLog(nodeName + " Found!\n");
    }
}
```

Codice 4.4: Inserimento evento StringSensor

4.2.3.4 checkFile()

In questo metodo viene scritto il file .xhtml finale. Esso copia il testo dal file temp e se necessario aggiunge nuove righe, ad esempio i link ai file contenenti le functions JavaScript chiamate dagli eventi HTML5 aggiunti.

```
//it checks the temp file created and create the
   outputFile
public void checkFile() throws IOException {
    file.seek(0);
    // xhtml outputFile
    bw2 = new BufferedWriter(new FileWriter(outputFile));

    line = null;
    while ((line = file.readLine()) != null) {
        //it checks if we have to add the link for a JS
           script
        if (line.contains("<head")) {
            line += "\n";
            bw2.write(line);
            if (writeScriptPlaneSensor) {
                bw2.write(dragScriptString);
            }
            if (writeScriptCylinderSensor) {
                bw2.write(rotatexScriptString);
```

```
            }
            if (writeStringSensor) {
                bw2.write(stringSensorScript);
            }
            //it checks if we have to set the
               NavigationInfo node
            } else if (line.toUpperCase().contains("<
               NAVIGATIONINFO") && (writeScriptPlaneSensor
                || writeScriptCylinderSensor)) {
            //if we have to set NavigationInfo node and it
                is present in x3d file, delete it
            while (line.contains("/>") == false && line.
               toUpperCase().contains("</NAVIGATIONINFO>")
                == false) {
                file.readLine();
            }
            bw2.write(navInfoString);

        } //if it is necessary, write navigation info
           after &lt;Scene&gt; node
        else if ((writeScriptPlaneSensor ||
           writeScriptCylinderSensor) && line.toUpperCase
           ().contains("<SCENE")) {
            line += "\n";
            bw2.write(line);
            bw2.write(navInfoString);

        }//if we need to set a StringSensor we add mouse
           string into <X3D node
        else if (writeStringSensor && line.contains("<X3D
           id='x3dElement'")) {
            StringBuilder str1 = new StringBuilder(line);
            str1.insert(line.indexOf("'x3dElement'") + 12,
                " keysEnabled='false' onkeypress=\"
               writeText(event);\"");
            str1.insert(str1.length(), "\n");
            bw2.write(str1.toString());
        }//add the id for JS to StringSensor
        else if (writeStringSensor && line.contains(
           textStringSensor)) {
            StringBuilder str1 = new StringBuilder(line);
```

```
                str1.insert(line.indexOf(textStringSensor) +
                    textStringSensor.length(), " id='
                    textKeyboard'");
                str1.insert(str1.length(), "\n");
                bw2.write(str1.toString());

                //copy line without changes
                } else {
                line += "\n";
                bw2.write(line);
            }
        }
        bw2.close();
            groupFound = false;
        writeX3dNode = false;
        writeScriptPlaneSensor = false;
        navigationInfo = false;
        writeScriptCylinderSensor = false;
        writeStringSensor = false;
            }
```

Codice 4.5: Creazione file di output .xhtml

Capitolo 5

Esempio di esecuzione

Vediamo ora un esempio di esecuzione del software. Eseguiamo il file X3dToX3dom.jar contenuto nella directory eseguibile. Apparirà la seguente finestra

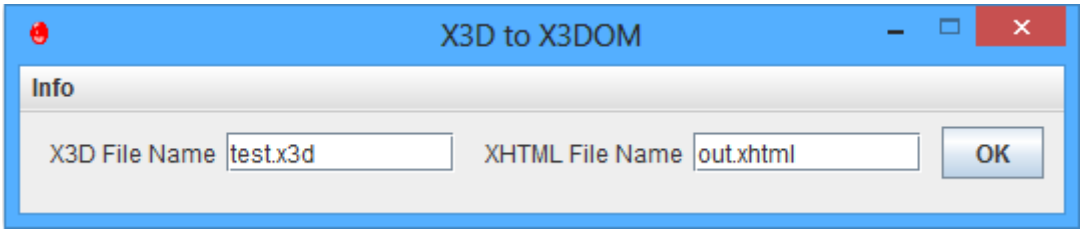

Figura 5.1: Finestra principale

Inseriamo il path al file di input .x3d e il file di output .xhtml e premiamo su OK. Se il file di input inserito è stato trovato (viceversa l'utente è avvisato tramite un pop-up) comparirà un pop up contenente il log e il file di output viene creato.

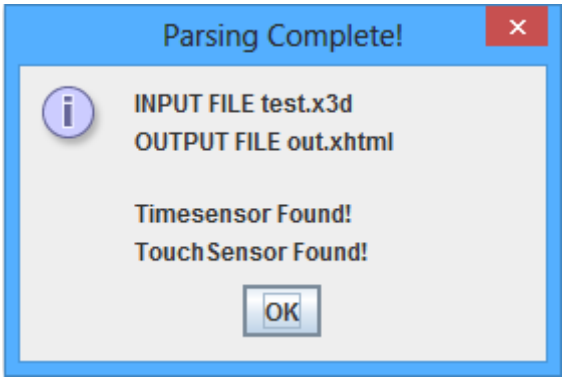

Figura 5.2: Parsing completato

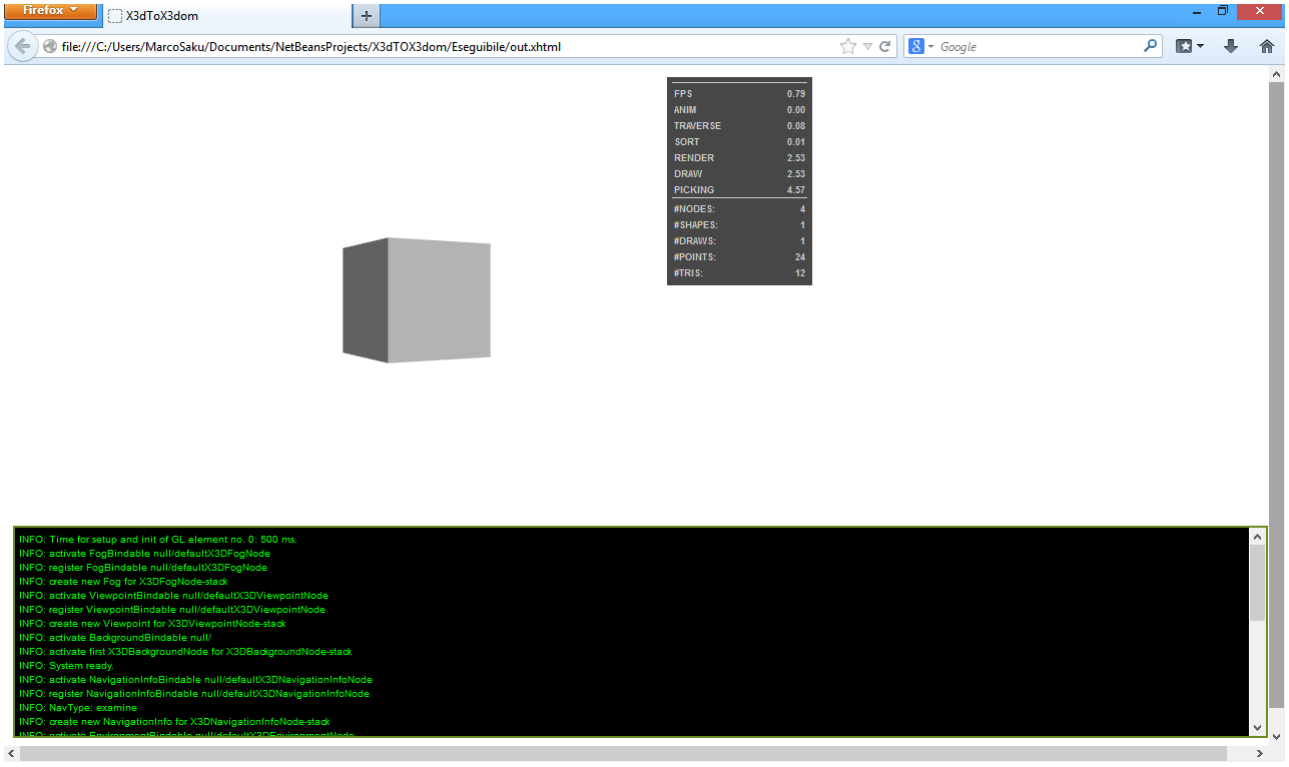

Figura 5.3: out.xhtml visualizzato tramite Mozilla Firefox

Bibliografia

[1] Virtual Reality Modeling Language
http://www.w3.org/MarkUp/VRML/

[2] Extensible 3D
http://www.web3d.org/realtime-3d/x3d/what-x3d

[3] MPEG-4 parte 11
http://en.wikipedia.org/wiki/MPEG-4_Part_11

[4] InstantReality
http://www.instantreality.org/

[5] X3D-Edit
http://www.web3d.org/x3d/content/README.X3D-Edit.html

[6] X3DOM
http://www.x3dom.org/

[7] HTML5
http://en.wikipedia.org/wiki/HTML5

[8] WebGL
http://www.khronos.org/webgl/wiki/Main_Page

[9] Java
http://www.java.com/