ICP4133
X3D Practical Exercises

Nigel W John

Introduction

You can use any text editor to create X3D content. However, you may find the free X3D-Edit editor useful as it provides features such as syntax checking for X3D authoring:

https://savage.nps.edu/X3D-Edit/

For Windows, several X3D viewers exist. Try BS Contact Player, Octaga Player, Instant Reality, or Xj3D (written in Java):

http://www.bitmanagement.com/en/download
http://octagavs.com/softwarem/octaga-player
http://www.instantreality.org/downloads/
http://www.xj3d.org/

The X3D specification document is available online at:

http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/

You will find this a useful reference source.

NOTE: In this tutorial, and the assignment, you are required to use the XML encoding. All of your X3D files should be saved with the extension ".x3d".

Exercise 1: Hello World

First start up your text editor and enter the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D profile='Immersive'>
    <Scene>
    </Scene>
</X3D>
```

Save the file and call it *simple.x3d*. This is the simplest X3D world possible - an empty one. You can check that it is empty by opening it with the X3D browser that you are using and see if you get a black window. If the browser gives any warnings consult the browser documentation to ensure that it supports the X3D XML encoding - some browsers might only support the X3D Classic VRML encoding.

Let's have a look at the code:

```
<?xml version="1.0" encoding="UTF-8"?>
```

This is the XML file declaration and it's used for easy identification. The file declaration is followed by the X3D document root element, which specifies a profile:

```
<X3D profile='Immersive'>
```

A complete overview of the profiles concept can be found in the X3D specification. Simply put it tells the browser what kind of nodes the world uses so that the browser can check if it supports the profile (and the nodes associated to that profile). The `Immersive` profile used here is targeted at "implementing immersive virtual worlds with complete navigational and environmental sensor control".

Next comes the empty `Scene` element:

```
<Scene> </Scene>
```

What is missing in our world is the content. Let's add a sphere by inserting the following element in between the Scene tags:

```
<Shape> <Sphere radius="3.0"/> </Shape>
```

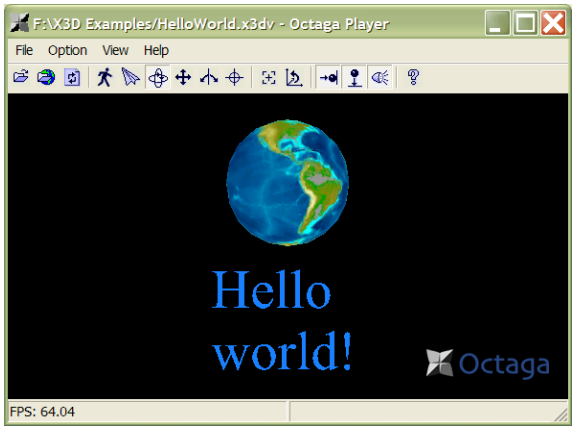
Try loading the new *simple.x3d* into your web browser.

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D profile='Immersive'>
    <Scene>
    <Shape>
          <Sphere radius="3.0"/>
    </Shape>
    </Scene>
</X3D>
```

Now edit the file to contain a Text node instead of a Sphere node – refer to lecture slides. Use a string field to say "Hello World". Try including an Appearance node to add colour and other effects. Split the text so that each word appears on a separate line. Save your work as *HelloWorld.x3d*.

Exercise 2: Adding the Earth

We will now extend *HelloWorld.x3d* to include a texture mapped sphere to look like the planet Earth:



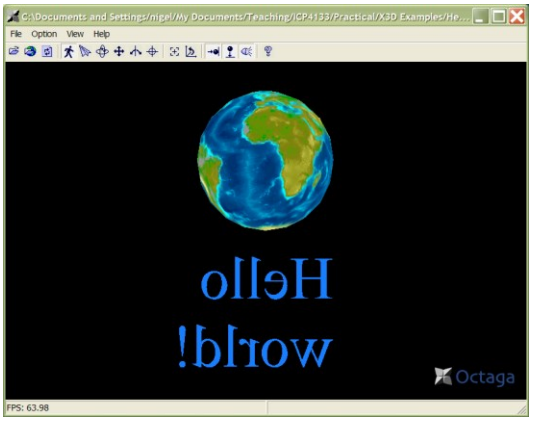
Required steps:

- Download "earth-topo.png" to use as a texture map from the Blackboard Labs Folder. Store in the same directory as your Hello World X3D file.
- Add a Shape node containing a Sphere of radius 1.0 unit to your Hello World X3D Scene. Enclose the Shape node in a Transform container node so that you can apply a translation of 1.0 unit along the y-axis to raise its height.
- In the Appearance node for the Sphere, apply the texture using (be careful with the quotes):
  `<ImageTexture url='"earth-topo.png"' />`
- You will also need to enclose the Shape node containing the Text in a Transform container node so as you can position the text accurately below the Earth.

Try and get your scene to look similar to the above snapshot. Save your work as *HelloWorld-1.x3d*

Option:

- Look up the Syntax of the Viewpoint node from www.web3d.org. This node allows you to set the position of the camera from which you view the scene.
- Add a Viewpoint Node to your X3D scene and position the camera so that when you load your X3D scene the default view is the Earth from the other side of the World (assume a right handed coordinate system). Note this will result in the Text appearing back to front! We will see a method for preventing this later.



Save your work as *HelloWorld-2.x3d*

Exercise 3: Creating a Prototype for Text Labels

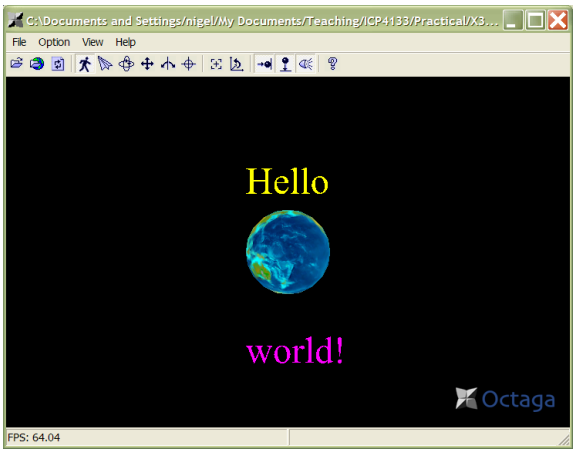
Add the following prototype declaration to *HelloWorld-1.x3d*. Save your work as *HelloWorld-3.x3d*:

```
<ProtoDeclare name="TextString">
  <ProtoInterface>
    <field accessType="inputOutput" name="offset" type="SFVec3f"
              value="0 0 0"/>
    <field accessType="inputOutput" name="string" type="MFString"
              value='"Some text"'/>
    <field accessType="initializeOnly" name="size" type="SFFloat"
              value="1.0"/>
 </ProtoInterface>
 <ProtoBody>
    <Transform>
      <IS>
        <connect nodeField="translation" protoField="offset"/>
      </IS>
      <Shape>
        <Text>
          <IS>
            <connect nodeField="string" protoField="string"/>
          </IS>
          <FontStyle>
            <IS>
              <connect nodeField="size" protoField="size"/>
            </IS>
          </FontStyle>
        </Text>
      </Shape>
    </Transform>
  </ProtoBody>
</ProtoDeclare>
```

Use your new TextString prototype to add the “Hello” and “world!” text to your scene using a call to ProtoInstance. Position “Hello” above the Earth, and “world!” below the Earth.

```
<ProtoInstance name="TextString" containerField="children">
  <fieldValue name="size" value="1"/>
  <fieldValue name="offset" value="-1 2 0"/>
  <fieldValue name="string" value="Hello"/>
</ProtoInstance>
```

Now extend the TextString protype to include an additional field, name=”color”. Then change the colour of your text strings in the call to ProtoInstance. Use different colours for each string:



Option:

- Create an external prototype. Save the TextString Prototype declaration into a separate X3D file and call it *TextStringPrototype.x3d*. Make sure it is in the same folder as the other files.
- Edit *HelloWorld-3.x3d* to reference TextString as an External Prototype:

```
<ExternProtoDeclare name="TextString"
                    url="TextStringPrototype.x3d#TextString">
  <field accessType="inputOutput" name="offset" type="SFVec3f"/>
  <field accessType="inputOutput" name="string" type="MFString"/>
  <field accessType="initializeOnly" name="size" type="SFFloat"/>
  <field accessType="inputOutput" name="color" type="SFColor"
                    value="1 1 1"/>
</ExternProtoDeclare>
```

- Store you work as *HelloWorld-4.x3d*.
- Note that some X3D browsers do not currently support external prototypes. Xj3D and Octaga should work fine.

Exercise 4: Using a PlaneSensor

Next we will introduce a PlaneSensor so that you can click on the Earth and drag it around the scene. Start with *HelloWorld-1.x3d*, and save your changes as *HelloWorld-5.x3d*.

- Use DEF to make sure that the Transform node containing the Earth model is named “MoveMe”
- Add a PlaneSensor node and use DEF to call it “Mover”. Use a Transform node (or other container node) to group the PlaneSensor together with the Transform node that contains the Earth.
- Add the appropriate ROUTE statement so that the “translation_changed” event from the PlaneSensor is mapped to the “set_translation” of the Transform node containing the Earth.
- Load your scene into a X3D player. You should be able to click on the Earth and drag it around the scene. Note how the cursor changes when you move it over the Earth. This indicates that a PlaneSensor is active.

Option

- Extend the PlaneSensor definition so that motion is clamped to the horizontal direction only.
- Experiment with using a SphereSensor and CylinderSensor instead of a PlaneSensor.

Exercise 5: Using a Head UP Display (HUD)

A Head Up Display (HUD) is set up as a Group Node whose children will follow the viewer, no matter what its position is in the scene. The viewers location is obtained from a ProximitySensor node. The output is then sent via a ROUTE statement to translate the HUD panel to follow the current view location. The net effect is that the HUD panel stays in the same screen location regardless of any user navigation. Note that it is important to set the relative z-coordinate to be negative, otherwise geometry will be behind the viewpoint and not visible.

Download HudPrototype.x3d from the Labs folder on Blackboard. This x3d file contains an example implementation of a HUD that contains a user-defined text string. Edit this file as follows:

- Add the X3D code to include the Earth model from the previous exercises in your X3D scene. Sphere radius should be 1.0, and position it at '0 1 0'. Note: the Earth model should not be contained in the HUD.
- Add a ProtoInstance definition with the string "Hello World!". The hudOffset value should be set to be '-2 -1 -8'
- Save your work as *HelloWorld-6.x3d*.

Load *HelloWorld-6.x3d* into a X3D player. Us the camera controls to move around the scene. Your text string should remain fixed, but you can freely roam around the Earth model. Look closely at the HUD definition. The ProximitySensor node generates position_changed and orientation_changed events when the viewer enters, exits, and moves within a region in space (defined by a box). Note how the ROUTE commands have been set up so that the ProximitySensor updates the position and orientation of the HudContainer. The Proximity sensor has been set up with a size of 1000 units in each direction. The HUD will not work if the camera moves outside of this volume.

Exercise 6: A Simple Script

Download the BouncingWorld-Template.x3d and milkyway.jpg files from the Labs folder on Blackboard. Load this into an X3D player. The Earth will be suspended above a plane, which has a texture map of the Milky Way placed on it. Using the Bouncer script example from the lecture notes, extend this X3D scene to include a script that will make the Earth bounce up and down on top of the milky way plane.

The supplied solution to this exercise (but don't download it yet!) also contains an example of how to use Browser.print() to write information from the script to the X3D console. Useful for debug purposes.

Note that this X3D file also contains example use of the following X3D nodes:

- NavigationInfo
- DirectionalLight
- Background
- IndexedFaceSet (use here to define the plane, so just 4 coordinates needed).

Last updated: September 2011