X3DOM Documentation

Release 1.3dev

Fraunhofer IGD/VCST

January 24, 2012

CONTENTS

Welcome to X3DOM's documentation. This documentation is divided into different parts. We recommend that you read the *Getting started* section first. Besides the getting started there is also a more detailed *First steps with X3DOM* tutorial that shows how to create complete (albeit small) application. If you'd rather dive into the internals of X3DOM, check out the *API* documentation.

CHAPTER

ONE

GUIDE

This part of the documentation, which is mostly prose, begins with some basic information about X3DOM, then focuses on step-by-step instructions for building applications with X3DOM.

1.1 Getting started

1.1.1 Downloading X3DOM

The recommended way of using X3DOM in your application is downloading a released version to your local machine or server. The code of a released version should be usable in all modern browsers.

X3DOM itself depends on no external libraries. All you need in order to make it work are the following files:

`x3dom-v1.2.js`	The minified X3DOM library in a given version
`x3dom-v1.2.css`	Stylesheets for X3DOM, you need to include this file in your webpage in order for X3DOM to display. However you can also take it as template to adapt your own stylesheet.
`x3dom-1.2.swf`	The Flash 11 integration, for browsers not supporting native X3DOM or WebGL.

You can download the files from the X3DOM server and put them on your harddisk or your webserver. The file naming follows the pattern: `x3dom-vMAIOR.MINOR.js`.

1.1.2 Development builds

If you are adventurous and want to work with the latest development build of X3DOM download the latest builds from the X3DOM server. Just use the files ending in no version number:

- x3dom.js
- x3dom.css
- x3dom.swf

WARNING: Do **NOT** use the development builds in a production system. It is not thoroughly tested. It is not stable and will probably break things. If in doubt, use the current release.

1.1.3 Build from revision control

Note: The following is for advanced developers. If you wish to use a rather recent version of X3DOM and do not want to tinker with Python, just use the development build.

All source code to X3DOM is kept under the Git revision control and you can browse the repository online. There is a download link available for any file or directory, if you only nee a portion of the X3DOM code.

If you have access to Git, you can get a copy of the repository here:

```
git clone https://github.com/x3dom/x3dom.git
```

You can also check out a specific release from GitHub:

```
git clone https://github.com/x3dom/x3dom.git
git checkout <version>
e.g. git checkout 1.2.0
```

If you want to build your own copy of X3DOM from the Git repository, you need to build it from the sources.

Build requirements

X3DOM currently requires the following components to be installed on your computer:

- Python: The Python programming language is available for all major platforms
- Sphinx: A documentation tool, you will need this in order to build the documentation. Sphinx is a Python package and can be installed by running `easy_install sphinx`.
- argparse: For Python 2.6 or earlier.

Once you have all prerequisites installed, you can build X3DOM:

```
python manage.py --build
```

The resulting build will be written to the `dist/` directory.

Fore more detailed information on working with the source, please see the developer wiki.

1.2 Tutorial

You want to develop an application with X3DOM? Here you have the chance to learn that by example. In these tutorials we will create simple applications of different kinds, but they still feature everything you need to get started.

Basic operation

1.2.1 First steps with X3DOM

This tutorial is intended to be a quick start for people who have no experience with 3D graphics so far but want to try out X3DOM. Those who want to learn more about it, should have a look at the book X3D: Extensible 3D Graphics for Web Authors about X3D, on which X3DOM is based, by Don Brutzman and Leonard Daly.

Authoring X3DOM content is very similar to authoring HTML. So just open an editor and start with the usual stuff as shown next. Please note the `<link>` tag, which includes the X3DOM stylesheet for having everything nicely formatted, and the `<script>` tag, which includes all JavaScript functions that are necessary to run your 3D scene:

```
<html>
  <head>
    <title>My first X3DOM page</title>
    <link rel="stylesheet" type="text/css"
          href="http://www.x3dom.org/download/x3dom.css">
    </link>
```

```
    <script type="text/javascript"
            src="http://www.x3dom.org/download/x3dom.js">
    </script>
  </head>
  <body>
    <h1>My X3DOM world</h1>
    <p>
      This is my first html page with some 3d objects.
    </p>
  </body>
</html>
```

Save your file and open it in an WebGL capable browser. As you can see, there is only some text. What's missing are the X3DOM-specific tags for specifying the 3D objects.

Hence, we'll now insert a red box into our page by inserting the following code after the closing <p> tag. Similar to a <p> or element, the <x3d> element defines a rectangular region that contains all its children elements (in this case the red box).

```
<x3d width="500px" height="400px">
  <scene>
    <shape>
      <appearance>
        <material diffuseColor='red'></material>
      </appearance>
      <box></box>
    </shape>
  </scene>
</x3d>
```

You might wonder, why the <box> tag isn't enough and what the other tags are good for. <scene> simply says, that you are going to define a 3D scene. And a <shape> defines the geometry (here a <box>) as well as the <appearance> of an object. In our example, the whole appearance only consists of a red <material>. If you want to learn more about these elements (or nodes as they are called in X3D), just follow this link and click on the node you are interested in.

Because simply looking at one side of the box is bit boring, you can navigate within your scene with the help of the mouse. If you move the mouse with pressed left mouse button inside the area surrounded by a black border, you'll rotate the point of view. With the middle mouse button you can pan around and with the right button you can zoom in and out. For more information see: *Camera Navigation*.

Ok, now you can move around, but admittedly this scene still is sort of boring. Thus, we'll add another object, a blue <sphere>, into our little scene. As is shown next, the <shape> is now surrounded by another element, the <transform> tag. This is necessary, because otherwise both objects would appear at the same position, namely the virtual origin of the 3D scene.

Thereto, the 'translation' attribute of the first <transform> element moves the box two units to the left, and the 'translation' attribute of the second <transform> element moves the sphere two units to the right. As can be seen in the example below, the value of the 'translation' attribute consists of three numbers. The first denotes the local x-axis (movement to the left/ right), the second defines the movement along the local y-axis (up/ down), and the third defines the movement along the local z-axis (back/ front).

```
<x3d width="500px" height="400px">
  <scene>
    <transform translation="-2 0 0">
      <shape>
        <appearance>
          <material diffuseColor='red'></material>
        </appearance>
```

```
        <box></box>
      </shape>
    </transform>
    <transform translation="2 0 0">
      <shape>
        <appearance>
          <material diffuseColor='blue'></material>
        </appearance>
        <sphere></sphere>
      </shape>
    </transform>
  </scene>
</x3d>
```

Now you know the basics of X3DOM. As you might have expected, there are certainly more nodes or elements you can try out like the `<cone>` or the `<cylinder>`. Also, there are other material properties like 'specularColor' and 'transparency'. By applying an `<imageTexture>` to your object, you can achieve fancy effects like a teapot that looks like earth as demonstrated in this example. When you have a look at the example's source code, you'll find a new tag called `<indexedFaceSet>`, which can be used for creating arbitrary kinds of geometry.

The example discussed here is also available online. Moreover, there are already lots of other X3DOM examples. Just try them out, and even more important, have a look at the source code to learn what's going on.

Please note, that there are slight differences between XHTML and HTML encoding: e.g. the latter does not yet work with self-closing tags but requires that tags are always closed in the form `</tagName>`.

If you ever have problems, please first check the *Troubleshooting* section of this guide, much helpful information is collected there.

1.2.2 Styling with CSS

This tutorial guides you through the process of using CSS with X3DOM. In order to demonstrate the functionality, we are going to create a HTML document with a X3DOM scene. That scene is then amended with a button that allows to resize the scene by setting CSS attributes using JavaScript.

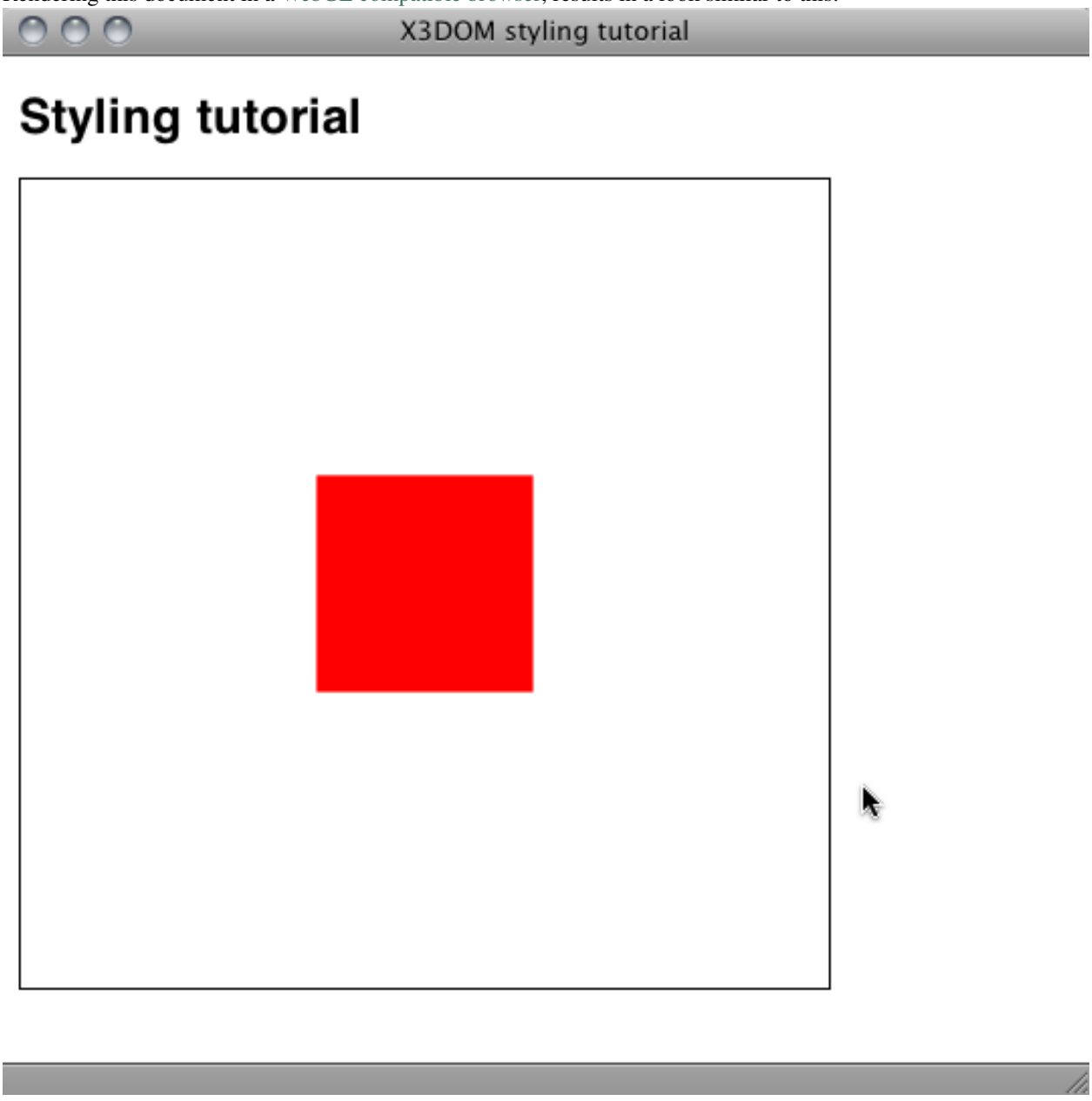
Basic scene document

We are going to use the box example scene established in the *First steps with X3DOM* tutorial:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>CSS Integration Test</title>
    <link rel="stylesheet" href="http://www.x3dom.org/download/x3dom.css">
    <script src="http://www.x3dom.org/download/x3dom.js"></script>
  </head>
  <body>
    <h1>Styling tutorial</h1>
    <x3d width="400px" height="300px">
      <scene>
        <shape>
          <appearance>
            <material diffuseColor='red'></material>
          </appearance>
          <box></box>
```

```
        </shape>
      </scene>
    </x3d>
  </body>
</html>
```

Rendering this document in a WebGL compatible browser, results in a look similar to this:



Basic styling

In the initial example above, we created the scene using the `<x3d>` tag initializing it with `'width'` and `'height'` attributes. In order to take advantage of CSS, we can use a CSS rule to set height and width in the visual layer.

The following CSS rules, added to the `<head>` of the HTML document element will resize the scene to 50% height and with of the parent element - in this case the `body` element. In order to make this work with IDs, we need to add

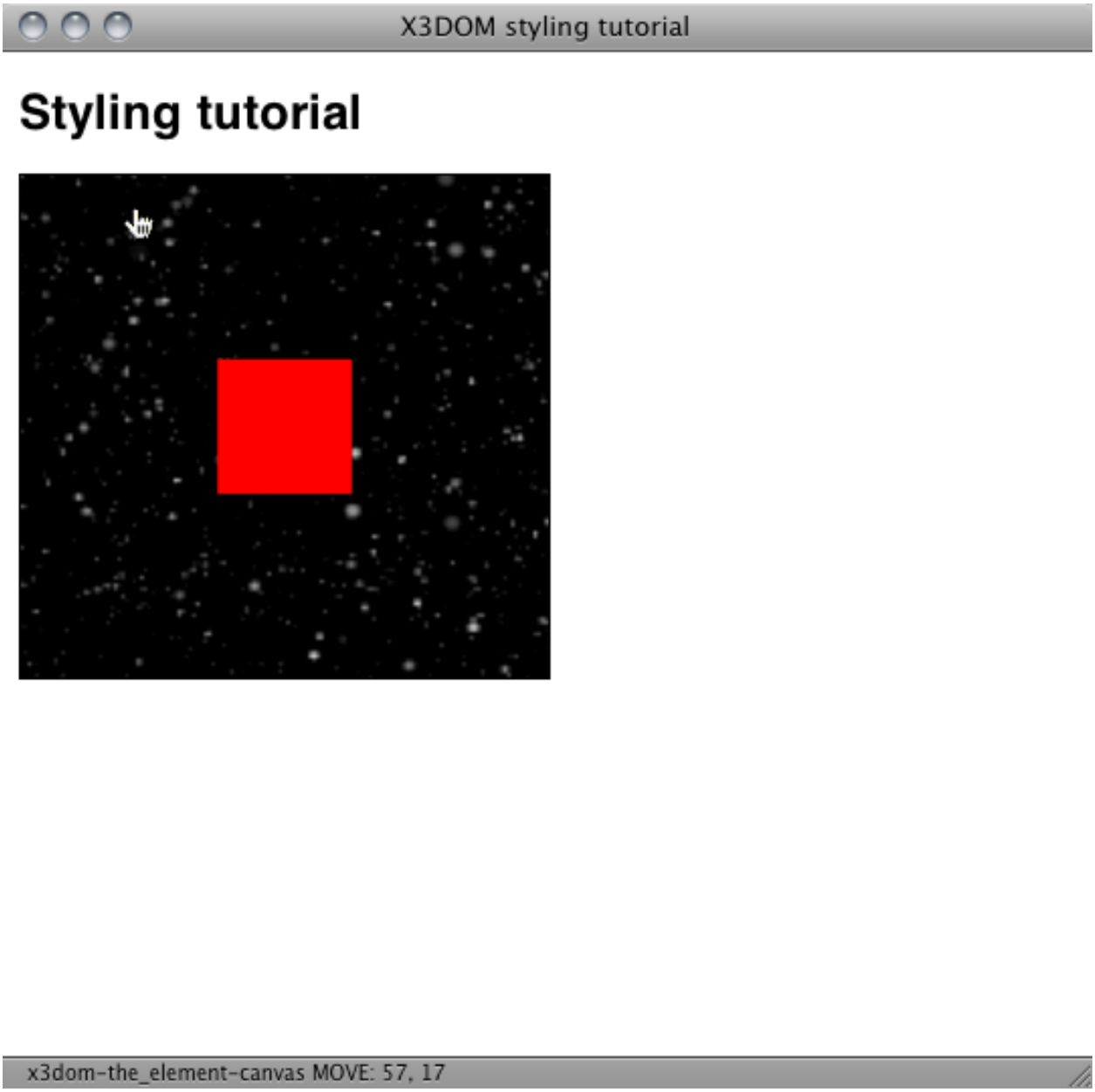
the `id` attribute to the <x3d> element:

```
<head>
  <style>
    #the_element {
      width: 50%;
      height: 50%;
    }
  </style>
</head>
...
<x3d id="the_one">
...
```

We need to remove the `width` and `height` attributes as well because they take precedence over the CSS rules. In order to change the background of the WebGL viewport we add the CSS background rule. To make this work, the scene must be transparent (default). If you change the background in the X3D definition, it will not be visible because it is disguised by the X3D background.

```
#the_element {
    width: 50%;
    height: 50%;
    background:#000 url(http://www.x3dom.org/x3dom/example/texture/solarSystem/starsbg.png);
 }
```

The result looks something like this:



Note that the dimensions are relative now and adapted when resizing the browser window.

- View demo video
- View live example

Adding interaction

The dynamic resizing showed in the last chapter is great for automatically adapting to browser resize and for positioning elements in your layout. In order to add more elaborate interaction with the scene, we can use JavaScript. For example, to change the dimensions of the X3D element so we can resize it to "fullscreen" like the ubiquitous video player.

In order to achieve this we are going to add a button to our example that allows to switch our X3D element to fullscreen – or more precisely to 100% of the parent HTML element.

First step is adding a piece of markup that we can use for styling as a button. Fortunately there already is a HTML element that is meant for exactly this purpose: `<button>`. And since we move in a HTML5 context, we put the button element within the `<x3d>` element:

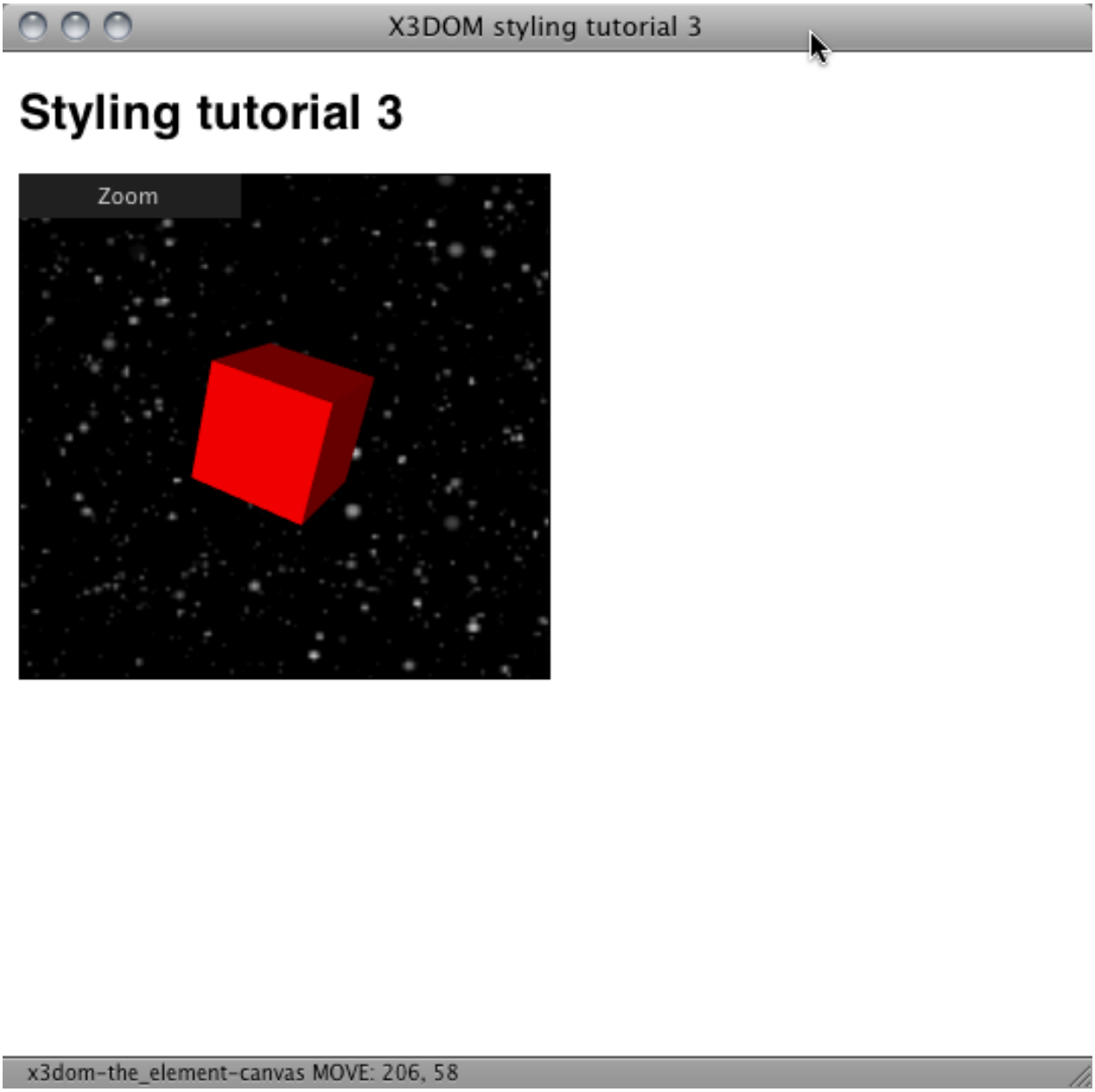
```
<x3d id="the_element">
    <button id="toggler">Zoom</button>
    <scene>
    ...
</x3d>
```

Semantically we are fine, but now we need to style the button so it floats over the scene. The following style rules will accomplish this. Please note the *position: realtive* property we are setting on the `x3d` element. We need this to allow absolute positioning of the button within the `x3d` element. Otherwise it would position absolute to the page. Additionally we need to remove the default 1 pixels border added by the X3DOM default stylesheet.

```
#the_element {
    ...
    border: none;  // remove the default 1px border
    position: relative;
 }

#toggler {
    position: absolute;
    float: left;
    z-index: 1;
    top: 0px;
    left: 0px;
    width: 10em;
    height: 2em;
    border: none;
    background-color: #202021;
    color: #ccc;
}
```

Looking at our example in a browser reveals that there is a “Zoom” button floating over the x3d element in the top left corner.



A button alone isn't interaction

Nice. But a button alone is quite useless, we need to be able to do something with it. But first things first. In order to give the user some feedback what is going on, we add a hover effect by simply changing the background color. This is basic usability and a simple style rule will do the job:

```
#toggler:hover {
    background-color:blue;
}
```

Next we add some JavasScript to the mix, because we want to actually change something when the user clicks on the button: Fullscreen. First we think of a method name to use, like `toggle()` and attach it to the `onclick` event of our button:

```
<button id="toggler" onclick="toggle(this);return false;">Zoom</button>
```

Note to the purists: Yes, there are several, more elegant ways of achieving this. For the sake of clarity of this tutorial, we are using the `onclick` attribute of the `button` element. In practice you probably want to use a DOM library like jQuery et al.

Next, we need to implement the `toggle` function. Within a `script` element. **After** the inclusion of `x3dom.js` we add the following code:

```
var zoomed = false;

function toggle(button) {

        var new_size;
        var x3d_element;

        x3d_element = document.getElementById('the_element');

        title = document.getElementById('title')
    body  = document.getElementById('body')

        if (zoomed) {
                new_size = "50%";
                button.innerHTML = "Zoom";
                title.style.display = "block"
                body.style.padding = '10px'
        } else {
                new_size = "100%";
                button.innerHTML = "Unzoom";
                title.style.display = "none"
                body.style.padding = '0'
        }

        zoomed = !zoomed;

        x3d_element.style.width = new_size;
        x3d_element.style.height = new_size;
}
```

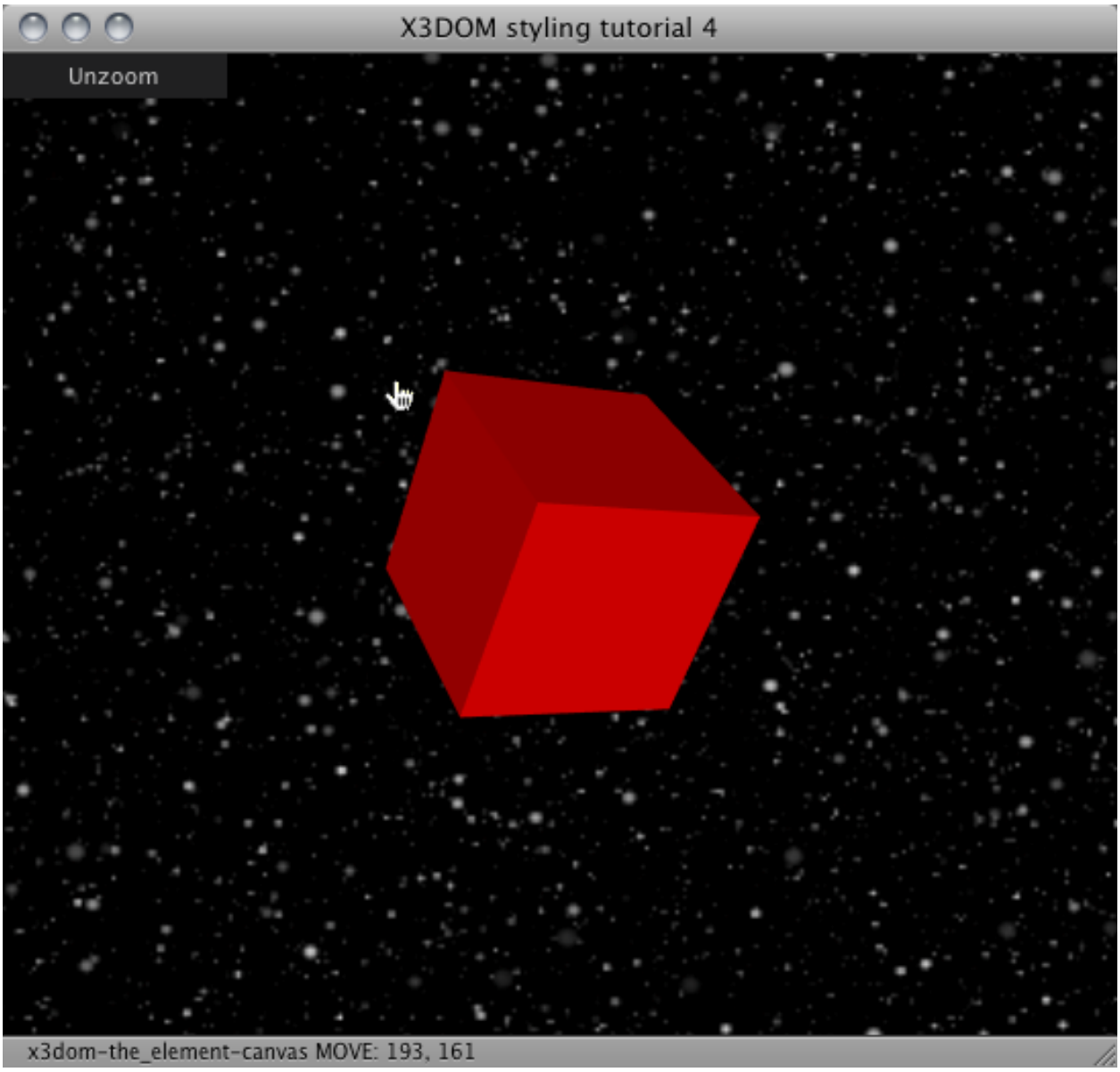
This code implements a simple toggle function. The boolean variable `zoomed` tracks the state of the resize. Depending wether the `x3d` element is sized to 100% or not, the new size is set and applied to the `x3d` element (`x3d_element`). Since we use a square viewport `width` and `height` have the same values. Additionally, the text of the button is changed to show the action performed when the user is clicking it.

The rest of the styling affects the surrounding elements like the title and the body. In order to achieve the fullscreen effect we obviously hide the *h1* title and remove the default padding of the `body` element. Likewise when zooming back, the values are restored.

The last bit in the puzzle is another style rule which resets margin and padding of the body element. Add this rule in front of all others:

```
<style>
    body {
        margin:0;
        padding:10px;
    }
    ...
</style>
```

Finally the fully zoomed result looks like this:



- See how it works (video)
- Try it for yourself (html)

You will also find another example which also styles other properties here.

1.2.3 Images, sound and movie formats

This tutorial shows what type of image, sound and movie formats can be used in X3DOM and what are the features and restrictions.

Images

You can use PNG, JPEG or GIF to encode your static Texture data. JPG has a low memory profile but has a lossy compression and it does not support alpha channels. PNG compression is lossless and can handle alpha. GIF is also

lossless and has alpha.

General: If you do not need an alpha channel and the content does not have hard edges (e.g. Text) use JPG. Otherwise use PNG. You should really not use GIF anymore. PNG is more flexible for future content (e.g. 16-bit channels).

```
<ImageTexture url='foo.jpg' />
```

Sound

You can use WAV, MP3 and OGG for sound sources. All UA should support WAV. If you would like to use compressed formats (e.g. MP3 or OGG) provide alternative encodings in your AudioClip node.

```
<AudioClip url='"foo.wav","foo.ogg"` />
```

Movies

There is right now no single movie file supported by all user agents. Use the X3DOM formats exmaple to check your browser.

The best solution right now is to encode your content as MP4 and OGV movie and provide alternative sources in your MovieTexture node.

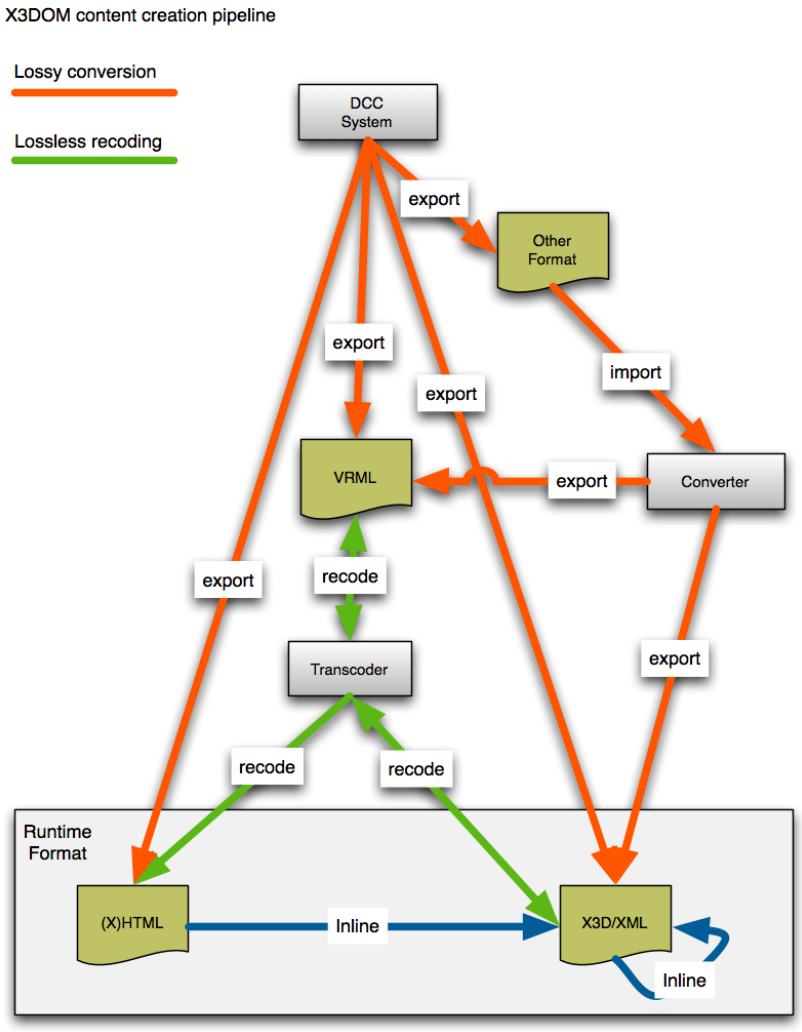
```
<MovieTexture url='"foo.mp4,"foo.ogv"` />
```

Content creation

1.2.4 Generic 3D data conversion

How to get your 3D asset in your application is one of the essential questions that every 3D runtime environment has to answer.

X3DOM uses X3D data embedded in (X)HTML pages and optional X3D-XML files referenced by the embedded part. The X3D-XML files can reference further X3D-XML files and therefore build a hierarchy of asset containers.



This tutorial shows how to get your data into the X(HTML) page and how to convert it to X3D-XML so it could be externally referenced.

DCC export

Usually people use some form of Digital Content Creation (DCC) tool to build the 3D models. This can be a modeling system like Maya or 3D Studio Max, and also a CAD-System or simulation package.

They all usually allow exporting the internal representation to some form of 3D data file. Most support X3D or VRML, some even both (e.g. blender) plus other formats. For X3DOM you should look for a X3D exporter. VRML is your second best choice. X3D is a VRML derivate and superset.

Converter

If your DCC-tool does not support X3D or VRML you are forced to utilize another tool which will introduce a extra level of conversion. Depending on your format there are usually different converters. Refer to X3D/web3d.org data conversion for more information.

However, you should really try to avoid this step and export directly to X3D or VRML.

Transcoding

If you have an X3D-XML or VRML file you can easily recode your data without any data loss. There are different options but the easiest right now is properly the Avalon-Optimizer (aopt) from the InstantReality packages. You can use it online or on your local machine to recode your data.

Offline Transcoding

Download and install the InstantPlayer system. The package includes a command line tool called aopt(.exe) which we will use for conversion. Setup your shell-environment to find and include the binary. The usually paths are:

- Windows: `C:\Program Files\Instant Player\bin\aopt.exe`
- Mac: `/Applications/Instant Player.app/Contents/MacOS/aopt`
- Linux: `/opt/instantReality/bin/aopt`

Then run `aopt -h` command to get a full list of options and usage instructions. For this tutorial the most important are:

```
aopt -i foo.wrl -x foo.x3d      # Convert VRML to X3D-XML
aopt -i foo.x3d -N foo.html     # Convert VRML or X3D-XML to HTML
aopt -i foo.x3d -M foo.xhtml    # Convert VMRL or X3D-XML to XHTML
aopt -i foo.x3d -u -N foo.html  # Optimization and build DEF/USE reuses
```

Building the File Hierarchy

A hierarchy of files can be built up with Inline nodes. The advantage here is that bigger objects/ meshes do not need to be directly part of a page's source code, but can be loaded in parallel in the background.

Important: If you use `<Inline url="foo.x3d" />` nodes in your content, you need a real server to run your application. This will not work locally from your disc.

1.2.5 Blender export

Converting Blender scenes into X3DOM webpages is pretty simple: Blender already supports direct X3D export even so there are some issues (Don Brutzman wrote about). Blender Version 2.4 seems to export some more nodes (e.g. lights), but in general it works. We will explore this more in the future, but an exported X3D file (such as the little horse shown below) may afterwards be easily be integrated into an HTML webpage using X3DOM.

Just finish your model in Blender and export to x3d file format (see next image).

There are two ways to get your X3D data into the HTML page, and both include no coding at all:

Two-file solution, link the X3D-file

Just use a very simple X3D scene in your HTML file, which, more or less, only includes an `<inline>` node. This nodes references and asynchronously downloads the X3D file. Therefore you need a real web server (e.g. Apache) running while using `<inline>` nodes in X3DOM.

The result: http://x3dom.org/x3dom/example/blenderExport/horse-inline.html

Figure 1.1: Horse model courtesy of http://etyekfilm.hu/

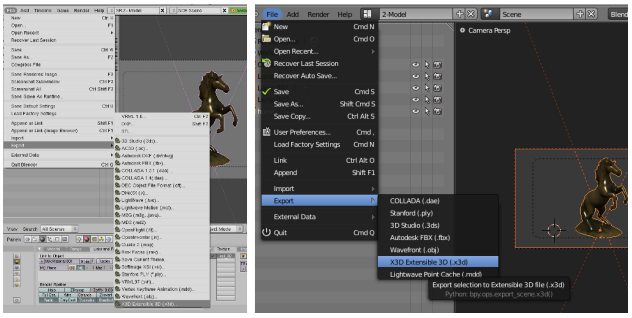


Figure 1.2: Export in Blender 2.4 (left) and 2.5 (right)

One-file solution, embed the X3D data inside of the HTML page

You can embed the X3D file by hand in a (X)HTML page, but this may include some hand-tweaking. Better use the aopt-converter described in *Generic 3D data conversion*. This can be done offline with a single command:

```
aopt -i horse.x3d -N horse.html
```

You also may use the converter online. Just open *horse.x3d* with your favorite text editor and paste it into the source text field. Choose *XML encoding (X3D)* as input type and *HTML5 encoded webpage* as output type and press the *Convert encoding* button.

The result: http://x3dom.org/x3dom/example/blenderExport/horse.html

The main difference between the two versions is the handling of Viewpoint nodes (as cameras are called in X3D). If you use the two-file solution, you get a spec-compliant standard camera, while the viewpoints in the included data are not available at the beginning. In the one-file solution you already have the Viewpoint nodes from Blender at the start time. Just copy one of the viewpoints into the main HTML page to correct this behavior if you want.

Here is a zip archive (272kb) with all files used in this tutorial including blender model, texture, and x3d model.

1.2.6 3ds Max Export

If you are using Autodesk 3ds Max for modeling (available only for Microsoft Windows), you can install our exporter plug-in InstantExport. If you do not yet have installed 3ds Max, there is also a 30-day trial version of the modeling software available. Nightly beta builds of InstantExport are available for download here.

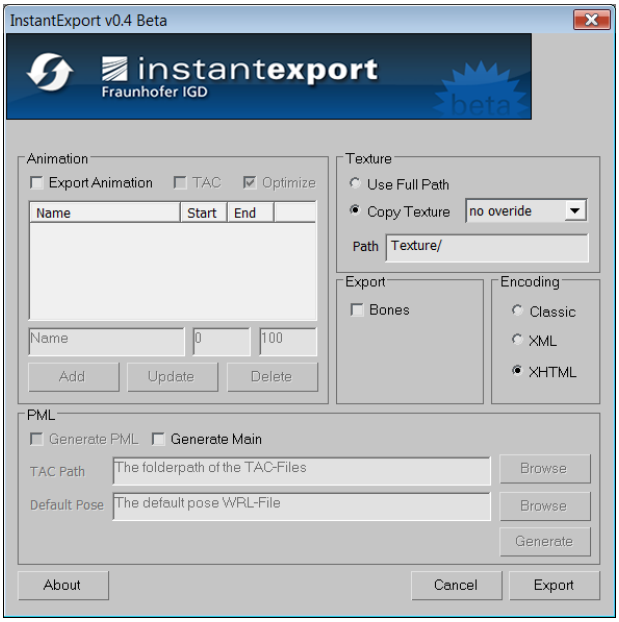
InstantExport is the InstantReality X3D exporter for 3ds Max and not only exports XML-based X3D as well as VRML, its classic encoding, but it can also now directly export to HTML/XHTML. But please note that – as the exporter plug-in is still under development – there are still lots of features in Max, which yet cannot be properly exported. So, if you find a bug, please report it in the InstantReality forum.

Installation

After having downloaded the exporter, unzip the zip file and choose the correct version for your system and Max version. After that, (assumed you are using the standard installation path and 3ds Max 2008) copy the file `InstantExport.dle` (the Max version of a DLL) into `C:\Program Files\Autodesk\3ds Max 2008\plugins`.

Export

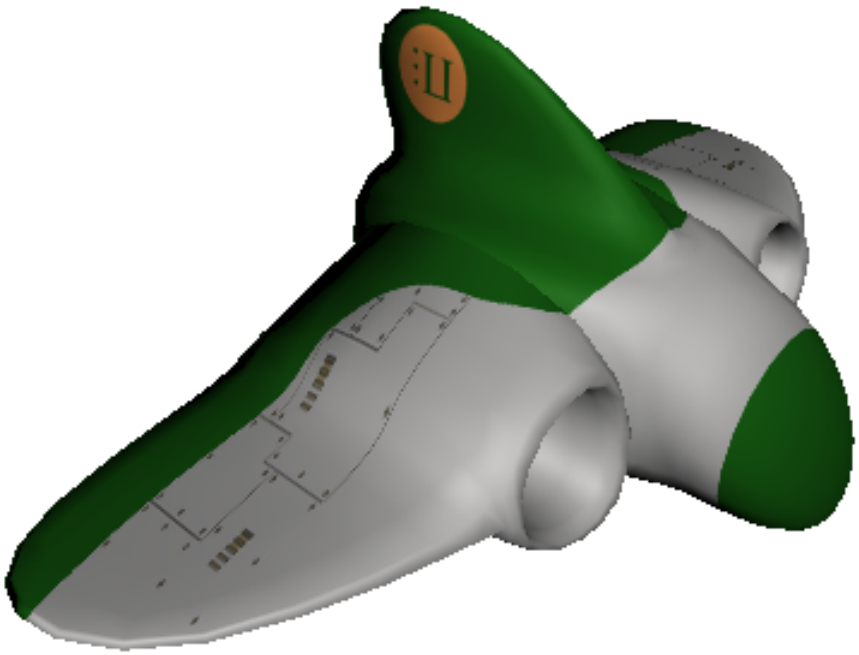
Then start 3ds Max, load the 3d model you want to export, choose *Export* in the File menu, type in a file name, e.g. `test.xhtml`, and select the file type – in this case *InstantExport (*.WRL,*.XHTML,*.X3D)* After that, the exporter GUI pops up. Here, under *Encoding* choose *XHTML*, as shown in the screenshot below. Finally, press the *Export* button. For more information, the zip file also includes a help file for the exporter.



1.2.7 Maya export

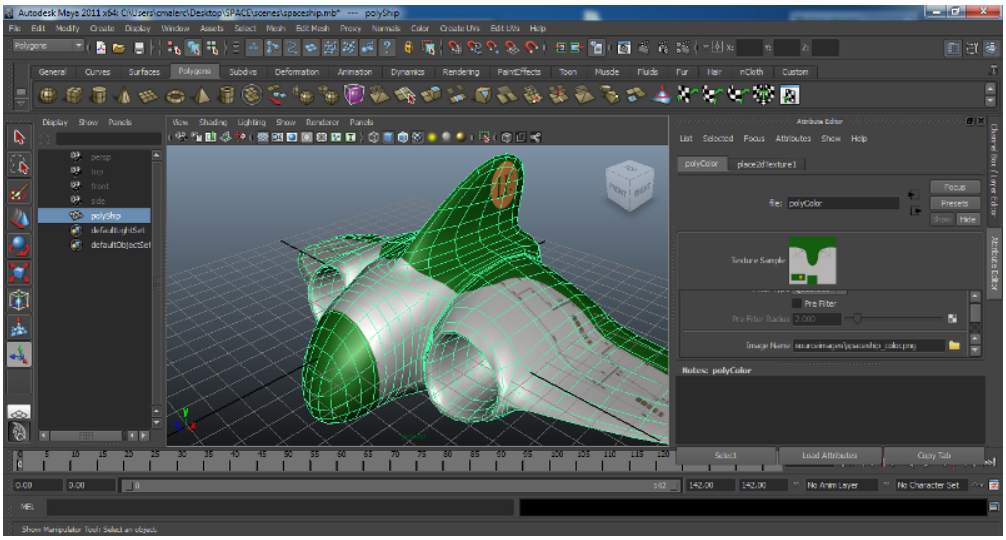
If you are working with Autodesk Maya for modeling, shading and animating your 3d scenes, use this tutorial to create an interactive X3DOM website out of your model. This tutorial is tested with Autodesk Maya 2011. Nevertheless, the procedure should work even for older Maya versions.

The basic idea is to export your scene to VRML and convert this to an X3DOM/HTMLsite using InstantReality's `aopt` binary (see *Generic 3D data conversion*).



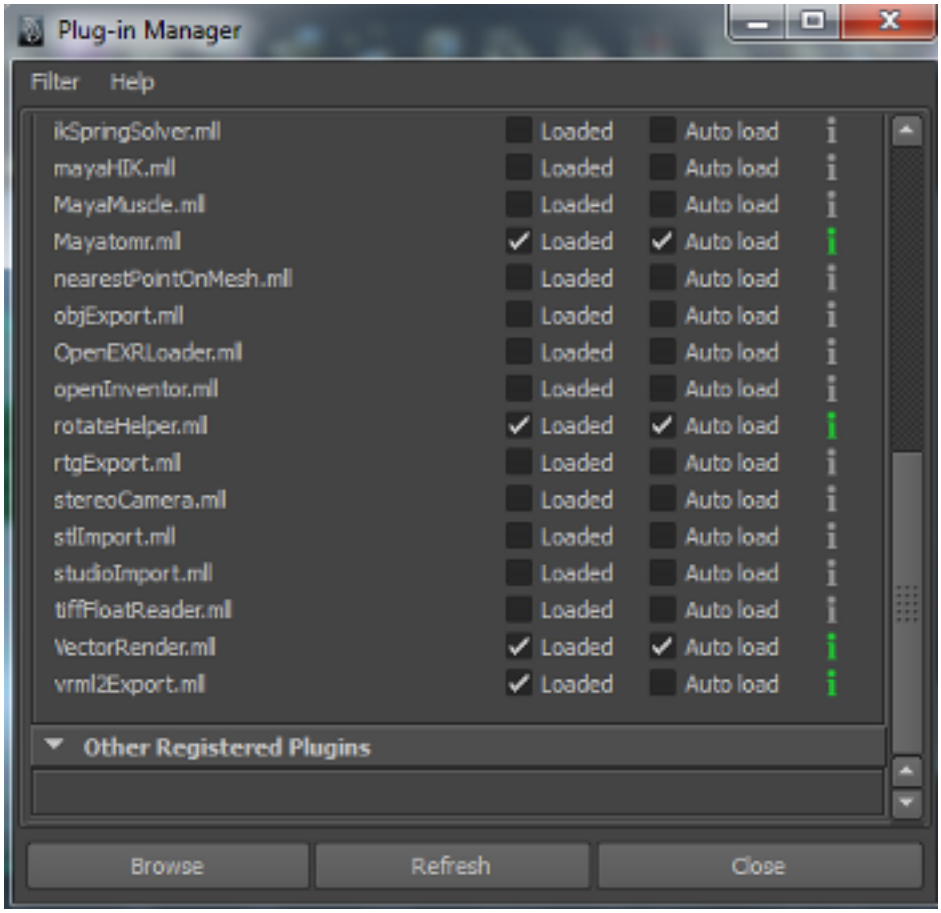
Step 1

Model and shade in Maya as usual. You should use PNG images for texturing.



Step 2

Open 'Window | Settings/Preferences | Plug-in manager' and check the 'loaded' or 'Auto load' option vrml2Export.

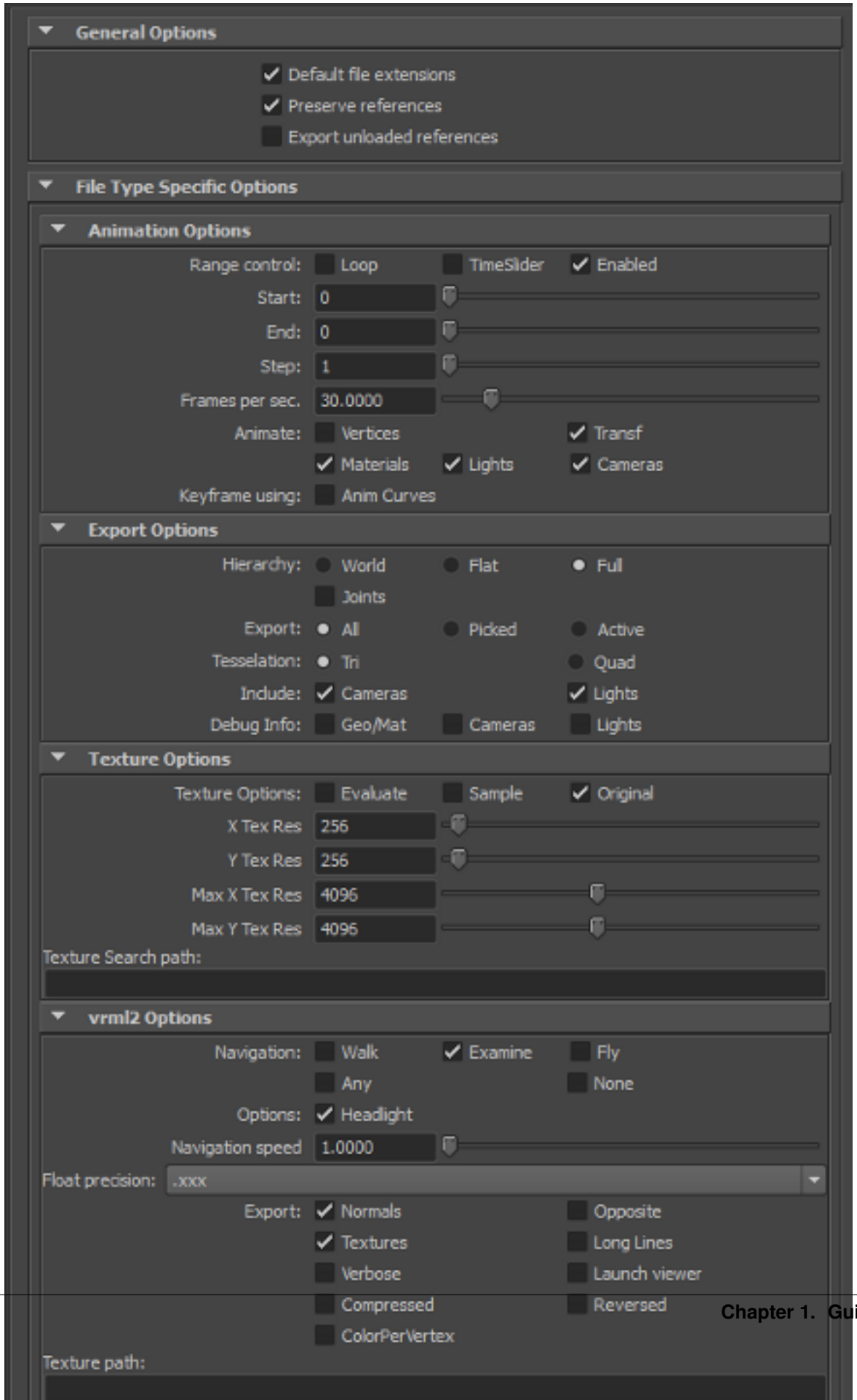


Step 3

Open the Export dialog under 'File | Export All..', Enter a filename (.wrl suffix) and switch to filetype 'vrml2. Don't forget to check the following export options:

Hierarchy: Full
Texture Options: Original
Export: Normals and Textures

Click the "Export All" button. This will create a vrml2 file in your scenes folder.


General Options
Default file extensions
Preserve references
Export unloaded references
File Type Specific Options
Animation Options
Range control:
Loop
TimeSlider
Enabled
Start:
0
End:
0
Step:
1
Frames per sec.
30.0000
Animate:
Vertices
Transf
Materials
Lights
Cameras
Keyframe using:
Anim Curves
Export Options
Hierarchy:
World
Flat
Full
Joints
Export:
All
Picked
Active
Tesselation:
Tri
Quad
Include:
Cameras
Lights
Debug Info:
Geo/Mat
Cameras
Lights
Texture Options
Texture Options:
Evaluate
Sample
Original
X Tex Res
256
Y Tex Res
256
Max X Tex Res
4096
Max Y Tex Res
4096
Texture Search path:
vrml2 Options
Navigation:
Walk
Examine
Fly
Any
None
Options:
Headlight
Navigation speed
1.0000
Float precision:
.xxx
Export:
Normals
Opposite
Textures
Long Lines
Verbose
Launch viewer
Compressed
Reversed
ColorPerVertex
Texture path:

Step 4

Open a terminal or command prompt, change to the folder containing your vrml2 model and your textures and run *aopt* (part of InstantReality, see *Generic 3D data conversion* for details) by typing the following command (assuming to be *spaceship.wrl* the name of your model):

```
| aopt -i spaceship.wrl -d Switch -f ImageTexture:repeatS:false
|      -f ImageTexture:repeatT:false -u -N spaceship.html
```

Note: `aopt` is automatically coming with your InstantReality player installation. You will find the executable within the `bin` folder of the Player. If you don't have Instant Reality installed yet, download and install from www.instantreality.org.

Step 5

Maya is using absolute path names. Therefore, open your html file with a standard text editor (vi, emacs, notepad++, etc.) and remove all paths from `ImageTexture`. For example, replace:

```
url='"c:\users\me\maya\project\sourceimages\spaceship_color.png"'
```

with:

```
url='"spaceship_color.png"'
```

Step 6

Copy HTML and textures into your web folder and open the website with your X3DOM capable browser.

If you want to try out this tutorial: Here is a zip archive (208 kb) containing all relevant files including Maya model and texture.

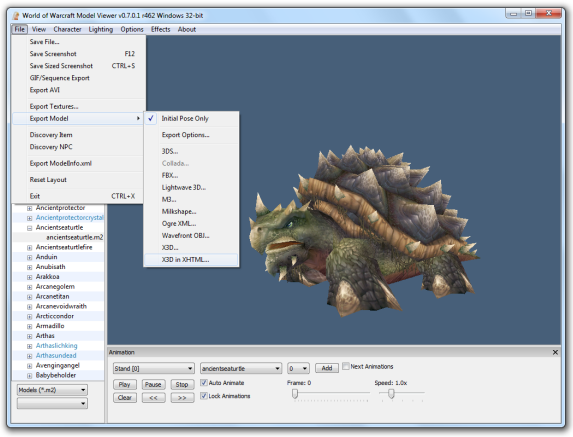
You want to see the result live in your browser? Here is the final webpage

1.2.8 World of Warcraft Models to X3DOM

WARNING: World of Warcraft Models are Blizzard property. You can not use them on your site without permission from Blizzard.

The WOWModelViewer project is an open source application to create machinima with characters and models form the World of Warcraft MMORPG. One of its features is the ability to export models into various formats, two of them being X3D and X3DOM. The X3DOM option directly outputs an XHTML file with the appropriate header code. In its current released version 7.0.1 r462, only static models can be exported, but an option to export animations as well is already available in the code.

To export a model you'll need a full World of Warcraft installation (for instance a 10 day trial version) and the WOWModelViewer. Open the application, select a model on the right hand side and click on *File->Export model->X3D* in XHTML.



The exported files can be viewed in a X3DOM capable browser.

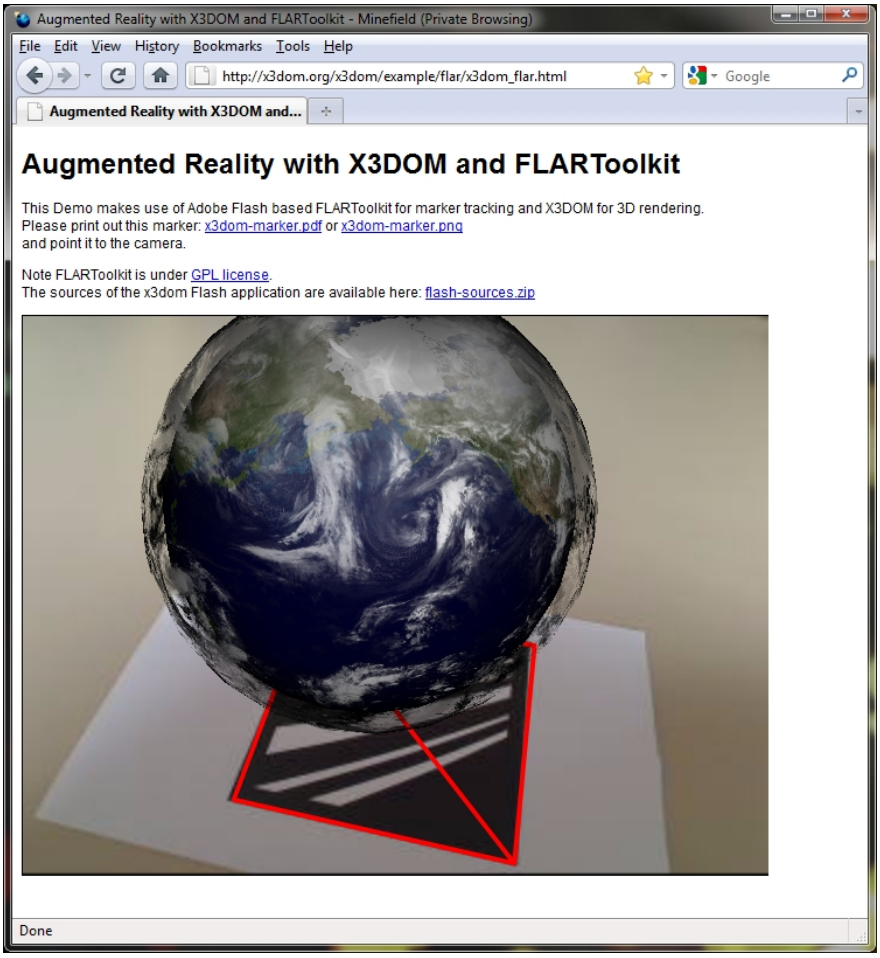
Some video results can be watched here:

- http://www.youtube.com/watch?v=16P6_e7VUmw
- http://www.youtube.com/watch?v=h4OnrXiA_Zc

Application prototypes

1.2.9 Flash + AR / X3DOM Mashup

This tutorial describes how to create a simple desktop augmented reality scene. We are using Adobe Flash based FLARToolkit for marker tracking and X3DOM for rendering of the 3D scene. By Jens Keil and Michael Zoellner.



View the AR Demo

At a glance

The tutorial shows the first online Augmented Reality application with Plugin-free hardware accelerated rendering on a GPU. Having the advantages of browser-supported WebGL technology, there is no need to download any kind of plug-in anymore to create Augmented Reality inside web browsers. Its a fast, simple and declarative way to integrate 3D content into HTML and bases on well known and documented standards, like HTML 5, CSS and JavaScript.

Although the tracking still uses Adobe Flash, its modular enough to change and switch tracking as soon as there are native standards for camera access available.

How does it work?

Our FLARToolkit marker tracker shows the webcam in the background and sends a model view matrix of the recognized marker to a Javascript function in the HTML file. From there the MatrixTransform around a bunch of 3D objects in the X3D scene is set with these values.

Setting up FLARToolkit marker tracker

Don't worry. You don't need the Flash IDE or any Actionscript knowledge for this tutorial. We are providing a compiled FLARToolkit marker tracker ready for including into an HTML page. It consists of the compiled SWF file (x3domflartoolkit.swf) and a Data folder with the camera parameters (camera_para.dat) and the marker pattern

(x3dom.pat). You can change the marker by creating a new one with the pattern generator, putting the results into the Data folder and renaming it to x3dom.pat. Please note that you should keep the generator's default values for marker resolution and segment size of 16×16 and 50% in order to work properly.

Including the FLARToolkit marker tracker

The compiled SWF is included via object and embed tags into the HTML page. It calls the Javascript function:

```
set_marker_transform(value)
```

as soon as it recognizes a marker in the video. The exchanging values include the marker's position and orientation. As mentioned, they are used to set the 3D object's position and rotation.

A simple 3D Scene

The demo scene shows a simple AR application: The earth globe, which hovers above the marker. A second layer shows the actual clouds surround the whole planet; live data loaded into the 3D scene.

Our demo is declared in HTML and structured in several divisions. Both, the 3D content and the compiled SWF, are grouped inside two several <Div /> nodes. The layer containing the 3d markup is styled with CSS and positioned on top of the compiled flash movie. Note that both have to have the same size and position in order to achieve a well augmentation effect.

Then, we set up a `<MatrixTransform />` node, which groups every 3D object we want to be positioned on the marker. Inside we declare a simple `<Sphere />` geometry and texture it with a png file of earth's appearance. Around the first one, we place a second `<Sphere />` object at the same position but with a larger scale and texture it with the transparent cloud data.

The basic structure

```
<x3d>
<scene>
    <viewpoint fieldOfView='0.60' position='0 0 0'></viewpoint>

     <matrixtransform id="root_transform">
            <transform  translation="0 0 20" scale="50 50 50"
            rotation="0 1 0 3.145">
                    <transform def="earth" rotation="1 0 0 -1.57">
                            <shape>
                                    <appearance>
                                            <imageTexture url="some_texture.jpg">
                                            </imageTexture>
                                    </appearance>
                                    <sphere></sphere>
                            </shape>
                    </transform>
                    <transform def="clouds" rotation="1 0 0 -1.57"
                    scale="1.1 1.1 1.1">
                            <shape>
                                    <appearance>
                                            <imageTexture url="some_texture2.jpg">
                                            </imageTexture>
                                    </appearance>
                                    <sphere></sphere>
                            </shape>
```

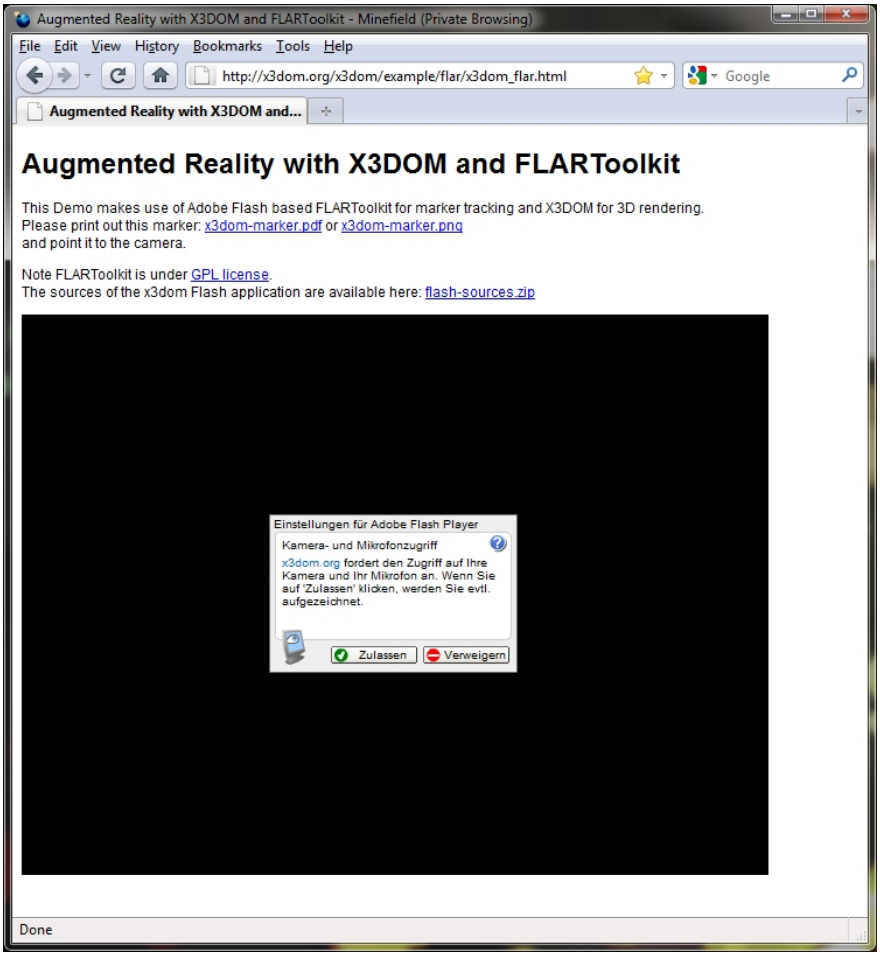
```
                    </transform>
                </transform>
            </matrixtransform>
</scene>
</x3d>
```

You don't need to calibrate your webcam. All of this is handled by the tracker's `camera_para.dat file`. Hence, our `<Viewpoint />`, i.e. our 3D camera, is fixed in its fieldOfview 0.6 and position of 0. The tracker's values only change and transform our 3D objects; not the camera.

The Javascript functionality

After declaring the 3D content, we add the Javascript code, that handles the data exchange between the Flash based marker tracking and our 3D scene.

First, we declare a function that hides the X3DOM canvas with the 3D content after the document has loaded. The user needs to allow the Flash tracker to access his camera by clicking a button. This is not possible, when x3dom is rendered on top at start up. As soon as the user confirmed and the marker is detected, we show the 3d content up again.



Our code:

```
var show_canvas = false;

 // Hide x3dom canavs on page load
```

```
 $(document).ready(function() {
    $('#topLayer').hide();
    show_canvas = false;
 });

// Show x3dom canvas again
// function is triggered inside set_marker_transform()
function show_x3dom_canvas(){
    $('#topLayer').show();
    show_canvas = true;
}
```

Lets take a closer look to the data exchange between X3DOM and the optical tracking:

We declare the `set_marker_transform(value)` function, which is expected by and triggered from inside the flash tracker. The function sets the new values for the MatrixTransform's position and rotation. Then we fetch the root `MatrixTransform` node

```
var root_transform = document.getElementById('root_transform');
```

and update the values with the `setAttribute(attribute, value)` function

```
root_transform.setAttribute('matrix', q.toString());
```

Since the tracking triggers new values for every (video) frame, the position is updated as long as the marker is detected. Note, that we also need to convert the received marker values, since X3DOM's and the tracking's coordinate system don't match.

Our code:

```
//This function is triggered by flash based tracking
function set_marker_transform(value) {
var q = value;
var root_transform = document.getElementById('root_transform');

// if not enabled, show x3dom canvas
if(!show_canvas)
    show_x3dom_canvas();

// Convert rotation form left to right handed coordinate system
// mirror z
q[2][3] = -q[2][3];
q[0][2] = -q[0][2];
q[1][2] = -q[1][2];
q[2][0] = -q[2][0];
q[2][1] = -q[2][1];

// update the grouped 3d object's matrixTranform
root_transform.setAttribute('matrix', q.toString());
}
```

The tracking also gives feedback when the marker is lost. If you want to work with this information, just declare and use this function inside your Javascript:

```
function on_marker_loss(value){
    //marker not detected anymore, do something
}
```

Trouble shooting

Sometimes the 3D content doesn't show up. This may have two reasons: Be sure you are using a browser who supports WebGL. Also texture loading may take a bit longer and hence may take X3DOM several seconds until the geometry shows up.

You can also control if the marker tracking is working: Check, whether there is a red outline around your marker. If not, ensure the marker is on a plane surface, not occupied and there is enough ambient light.

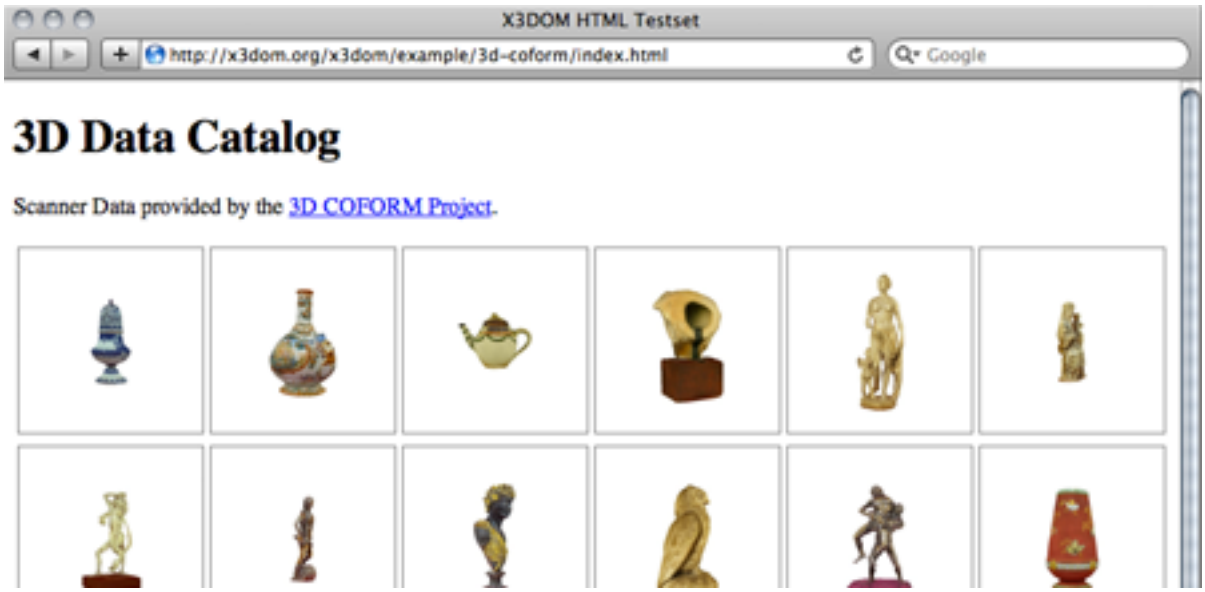
1.2.10 3D Lightbox Gallery of Historical Objects

This tutorial describes how to load arbitrary 3D geometry inside your web page with x3dom. We are going to develop an online catalog of 3D objects, that popup inside our page using the popular lightbox overlay principle (click here for the demo). In our case, the 3D objects are X3D files of 3D scanned historical objects. By Jens Keil.

Generating the grid

Our main page is only the overview of all objects. Hence, we are going to generate a grid with thumbnail images of our objects. We link these images to a second page with the X3DOM content. Since we have 36 objects our grid consists of 6 rows and 6 columns. Let's use a table for that.

```
<table id="demo_table" class="gallery clearfix >
  <tr>
    <td><a href="external_html_page"><img /></a></td>
  </tr>
</table>
```



As mentioned, our 3D content is displayed inside a lightbox popup. This is a JavaScript based script that is normally used to overlays images inside the current web page. In our case, we are going to overlay a external page with the 3D object in it. We have used the prettyPhoto lightbox version of Stephane Caron, since it features the iframes which we need to load a second HTML file into our main page.

In order to tell the script that our linked content should be opened inside the overlay, we add some query parameters at the end of the URL. For example:

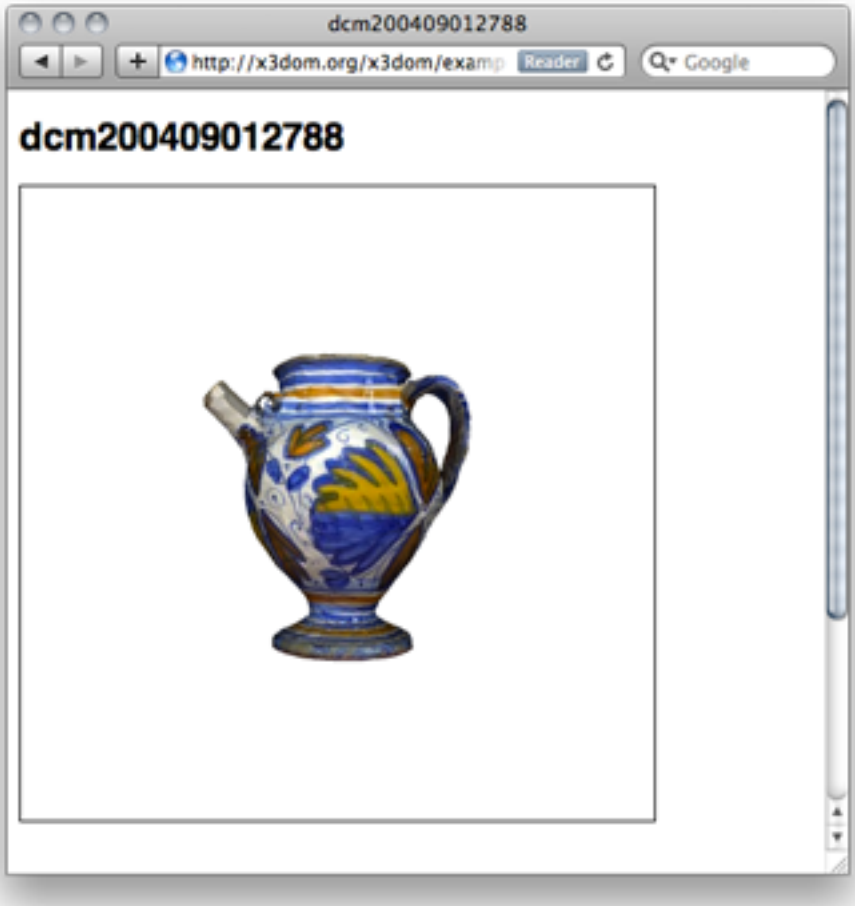
```
<a href="dcm200310301737.html?iframe=true&width=500&height=600"
   rel="prettyPhoto[iframe]" />
```

Having finished to set up the grid, we initialize the lightbox script after the table definition:

```
<script type="text/javascript" charset="utf-8">
    $(document).ready(function(){
        $(".gallery a[rel^='prettyPhoto']").prettyPhoto(
            {theme:'light_rounded'});
        });
</script>
```

Setting up the 3D object's HTML file

Now, let's take a look on the inlined page. We have such a page for every 3D object inside our grid. First, we export the scanned data into the X3D file format. Then we convert the X3D file into a X3DOM/HTML file (see *Generic 3D data conversion*).



Our X3D decoded 3D content is inside the generated HTML now. We may add a headline or some textual explanation here; indeed, even any other media we'd like to be displayed inside our lightbox overlay. Note, that adding the script node with a link to `x3dom.js` at the end is doing all the magic: from declarative X3D/HTML5 to visual 3D content inside your web page.

```
<html>
  <head></head>
  <body>
    <h1>dcm200409012807</h1>
    <x3d id='someUniqueId' showStat='false' showLog='false' x='0px' y='0px' width='400px' height='40
      <scene DEF='scene'>
        <worldInfo title='dcm200409012807'></worldInfo>
        <navigationInfo headlight='true' type='"EXAMINE"'></navigationInfo>
        <directionalLight on='false' ambientIntensity='1' intensity='0'></directionalLight>
        <transform DEF='ORITGT' rotation='1 1 1 -2.094'>
          <shape>
            <appearance>
              <imageTexture url='"dcm200409012807_texture.0.jpg"'></imageTexture>
            </appearance>
            <indexedFaceSet texCoordIndex=' ... ' />
              <coordinate DEF="COORD" point=' ... ' /></textureCoordinate>
            </indexedFaceSet>
          </shape>
        </transform>
```

```
        <background skyColor='1 1 1'></background>
        <viewpoint position='0 0 4'></viewpoint>
      </scene>
    </x3d>
    <script type='text/javascript' src='../x3dom.js'></script>
  </body>
</html>
```

Summary

This tutorial explained how to generate a grid of 3D object inside a web page. Clicking on a thumbnail image opens the 3D object inside a lightbox popup within the current page. Rendering as well as basic navigation is handled by the X3DOM Javascript back end.

Elsewhere

If you can read german, there is also some german content available on our website:

- Beispiele aus dem iX-Tutorial
- Überblick und einführende Beispiele

1.3 Camera Navigation

The current WebGL/JS implementation of X3DOM provides some generic interaction and navigation methods. Interactive objects will be handled by HTML-like events. Navigation can be user-defined or controlled by specific predefined modes.

Currently X3DOM supports the following interactive navigation modes:

- Examine
- Walk
- Fly
- Look-at
- Game

Non-Interactive movement encompasses the functionality of:

- Resetting a view
- Showing all
- Upright view

1.3.1 Interactive camera movement

Examine

Activate this mode by pressing the "e" key.

Function	Mouse Button
Rotate	Left / Left + Shift
Pan	Mid / Left + Ctrl
Zoom	Right / Wheel / Left + Alt
Set center of rotation	Double-click left

Walk

Activate this mode by pressing the "w" key.

Function	Mouse Button
Move forward	Left
Move backward	Right

Fly

Activate this mode by pressing the "f" key.

Function	Mouse Button
Move forward	Left
Move backward	Right

Look at

Activate this mode by pressing the "l" key.

Function	Mouse Button
Move in	Left
Move out	Right

Game

Activate this mode by pressing the "g" key.

To look around (rotate view) move the mouse.

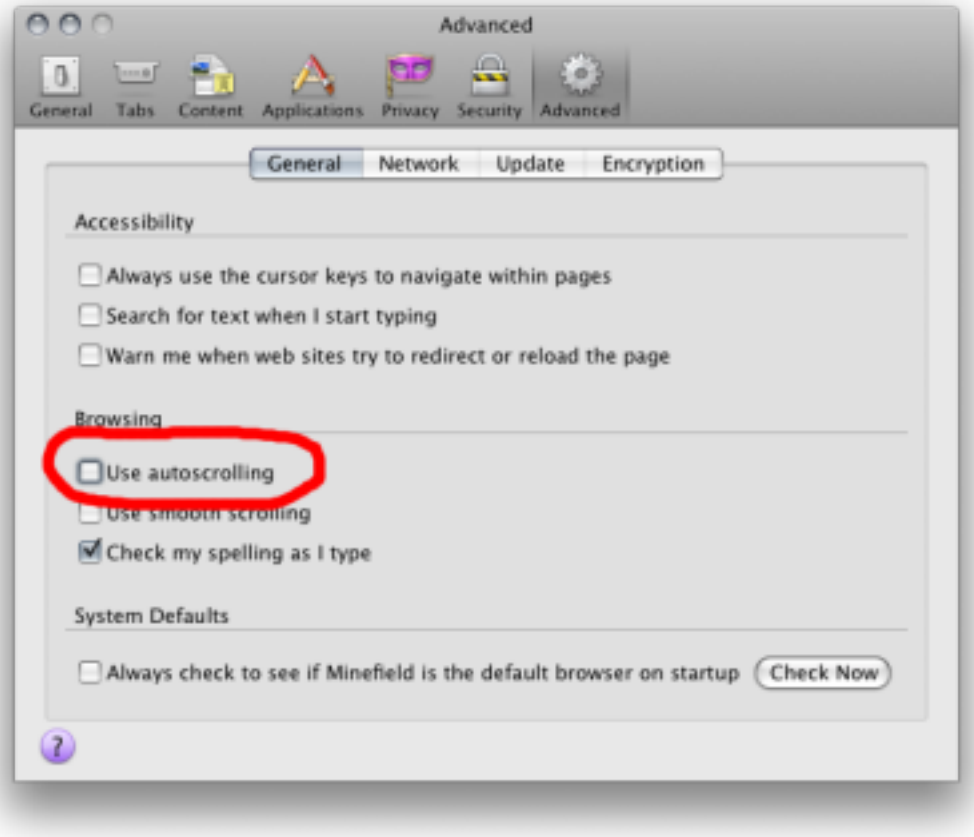
Function	Key
Move forward	Cursor up
Move backward	Cursor down
Strafe Left	Cursor left
Strafe Right	Cursor right

1.3.2 Non-interactive camera movement

Function	Key
Reset view	r
Show all	a
Upright	u

1.3.3 Mid-Button troubleshooting

If the web page has scroll bars and autoscrolling is enabled, Mid-Button currently does not work yet. As a workaround, you can disable autoscrolling by unchecking the **Use autoscrolling** checkbox in the Firefox browser options, as is shown in the screenshot below (for the Firefox case).



1.4 Configuration

The X3D element supports attributes and a param tag which allows to set configuration for the runtime.

1.4.1 Usage

The param element behaves just like any other HTML element. It must be nested below the X3D element. For XHTML you can use the self-closing syntax, for HTML a closing tag is mandatory:

```
<x3d>
    <param name="showLog" value="true" ></param>
    <scene>
        ...
    </scene>
</x3d>
```

Note: The param tag used to live as child of the scene element. This behavior has been changed with version 1.3 of X3DOM. You will get a deprecation warning and support will be removed in 1.4.

1.4.2 Options

The following table lists the parameters currently supported.

Parameter	Values	Default	Description
showLog	true, false	false	Hide or display the logging console
showStat	true, false	false	Hide or display the statistics overlay
showProgress	true, false, bar	true	Hide or show the loading indicator. The default indicator is a spinner. The value 'bar' will use a progress bar.
PrimitiveQuality	High, Medium, Low, float	High/1.0	Render quality (tesselation level) for Box, Cone, Cylinder, Sphere.
component	String (i.e. Geometry3D)	none	Name of the component to load
loadpath	String (i.e. nodes/)	none	The path or URI where to find the components
disableDoubleClick	true,false	false	Disables the default double click action on viewarea
disableRightDrag	true,false	false	Disable mouse right button drag and mouse wheel

1.5 Troubleshooting

As with most software systems, something can go wrong. But fear not. There are a couple of things you can check and try. You'll find some hints and tips here.

1.5.1 Common problems

I am not seeing anything

Assuming we are not dealing with an electrical or vision problem:

- Check if your HTML and X3D code is correct
- If you are using the HTML5 doctype make sure all X3D tags are properly closed. You can not use "self-closing" syntax. Instead you need to close the tag explicitly: `<color ...></color>` NOT `<color ... />`
- For the HTML part, swing by the W3C validator to check for syntax errors

There are weird chars or some gobbledygook on my web page

This is most likely an encoding problem or errors generated by using unsuited editors or HTML export tools (looking at you Word). Gobbledygook may be caused by improperly closed tags (see above).

- Check if you file encoding is OK. UTF-8 is recommended unless otherwise required.
- Use a HTML `meta` to denote the file encoding and make sure your file endcoding matches your meta tag.
- In case you are serving your files from a web server: make sure the server sets proper HTTP headers (especially mimetype and encoding) and middleware does not alter the file encoding (PHP et al. are sources for messing up multi-byte encodings). Again, maintaining UTF-8 throughout is a sensible choice (also read this).

1.5.2 How to ask Questions

In order to analyze and debug your problem, please be more specific about the nature of your problem. Before you sit down and write a mail or forum post, it is helpful to ask yourself these questions and include this info in your question/report:

- What exactly is not working?
- Is it really an error or are you just unhappy with aesthetics?
- What errors do you get?
- Can you reproduce this behavior in an isolated testcase?
- What did you try to solve your problem?

The following info is genuinely helpful in analyzing your problem:

- OS, Version, Architecture
- GPU type, driver versions
- Output of about:gpu in Chrome
- Browsers and versions you tried and the results
- Log output of X3DOM (javascript console)
- You can generate some of the information here http://doesmybrowsersupportwebgl.com/

Also note that hot-linking x3dom.css/js should only be used for testing and development. Once you deploy your site, it is best to copy those files over to your server or a CDN. We can not guarantee that those URLs are stable and our network bandwidth is rather limited.

This his also a helpful page and a good read as well: http://catb.org/~esr/faqs/smart-questions.html

CHAPTER

TWO

REFERENCE

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

2.1 API

The X3DOM runtime API provides proxy object to programmatically read and modify runtime parameters. The runtime proxy is attached to each X3D element and can be used in the following manner:

```
var e = document.getElementById('the_x3delement');
e.runtime.showAll();
e.runtime.resetView();
...
```

Some methods, like the `x3dom.ready()` function need to be called before the proxy object can be initialized. You can still override these functions globally. In order to provide you with the means to scope your actions to a specific X3D element, the methods receive the X3D element they are working on as first parameter:

```
x3dom.ready = function(element) {
    if (element == target_element) {
        // do something
    }
};
```

2.1.1 Runtime

ready()
: This method is called once the system initialized and is ready to render the first time. It is therefore possible to execute custom action by overriding this method in your code:

  ```
  x3dom.runtime.ready = function() {
      alert("About to render something the first time");
  };
  ```

  It is important to create this override before the document onLoad event has fired. Therefore putting it directly under the inclusion of `x3dom.js` is the preferred way to ensure overloading of this function.

enterFrame()
: This method is called just before the next frame is rendered. It is therefore possible to execute custom actions by overriding this method in your code:

```
var element = document.getElementById('my_element');
element.runtime.enterFrame = function() {
    alert('hello custom enter frame');
};
```

During initialization, just after `ready()` executed and before the very first frame is rendered, only the global override of this method works.

If you need to execute code before the first frame renders, it is therefore best to use the `ready()` function instead.

getActiveBindable(*typeName*)

Arguments

- **typeName** (*string*) – A valid Bindable node (e.g. Viewpoint, Background,

Returns Active dom element

This method returns the currently active bindable DOM element of the given type.

For example:

```
var element, bindable;
element = doucment.getElementById('the_x3delement');
bindable = element.runtime.getActiveBindable('background');
bindable.setAttribute('set_bind', 'false');
```

nextView()
Navigates to the next viewpoint.

prevView()
Navigates to the previous viewpoint.

resetView()
Navigates to the initial viewpoint.

uprightView()
Navigates to upright view.

showAll()
Zooms so that all objects are visible.

debug(*show*)

Arguments

- **show** (*boolean*) – true/false to show or hide the debug window

Returns The current visibility status of the debug window (true/false)

Displays or hides the debug window. If the paramter is omitted, the current visibility satus is returned.

navigationType()

Returns A string representing the active navigation type.

A readout of the currently active navigation type.

examine()
Switches to examine mode.

lookAt()
Switches to lookAt mode.

lookAround()
Switches to lookAround mode.

walk()
Switches to walk mode.

speed(*newSpeed*)

Arguments

- **newSpeed** (*float*) – The new speed value (optional)

Returns The current speed value

Get the current speed value. If parameter is given the new speed value is set accordingly.

statistics(*mode*)

Arguments

- **mode** (*boolean*) – true/false to enable or disable the stats info

Returns The current visibility of the stats info (true/false)

Get or set the visibility of the statistics information. If parameter is omitted, this method returns the visibility status as boolean.

2.2 Nodes

Please visit the Node Type Tree document and click on a Node to view the related X3D specification.

Warning: Not all nodes and fully implemented right now.

2.3 Components

X3DOM features a component system which allows you to load parts of the system at runtime. Components are a set of X3D nodes logically grouped together and put into a file. For example, the Geometry2D component consists of nodes named Arc2D, Circle2D, etc and is stored in a file named `Geometry2D.js`. Those components are then further grouped into profiles which combine them for specific application domains. For example there is a core profile, an immersive profile, Interchange profile, and so on. Except for the full profile, profiles are an abstract concept and not reflected in file layout.

While logical grouping like this falls mostly into the category of code organization, it can be utilized to load only the parts you need for your application. With X3DOM there are two versions of the library in the distribution package:

- the standard full profile file: `x3dom-full.js`
- the core containing only basic nodes: `x3dom.js`

You will find these files in release directory of X3DOM. Note that this is currently the development version of X3DOM. http://x3dom.org/download/dev/.

The full profile contains all the nodes of the official X3D specification, as far as they are implemented in X3DOM, merged into one file.

When using `x3dom.js` (core) you may need to either include or dynamically load additional nodes you need to render your model. This can be achieved by including the required node implementations files in your HTML using the `<script>` tag, or by instructing X3DOM to load the components at runtime.

By default X3DOM comes with the following additional nodes:

- Geometry2D
- VolumeRendering
- Geospatial

If you are using `x3dom.js` and you need to load the nodes above, you can use one of the methods described below.

Note: It is recommended that you use the full X3DOM version (`x3dom-full.js`) in production environments - unless there is a very specific reason not to. The full version is compacted, optimized and in almost all cases the right way of including X3DOM. Should you opt for using the methods described here, you are trading negligible saving in initial download size for a much slower loading system, additional requests, way more complicated setup and maintenance, inability to use the browsers cache, problems with firewalls, proxy servers, CORS issues, CDNs, and not being able to run run locally without a web server.

2.3.1 Static component loading

This technique works by manually including the X3DOM core library plus the components you need for rendering your model. Your resulting HTML could look like this.

```
<script src="x3dom.js"></script>
<script src="Primitives2D.js"></script>
<script src="Other.js"></script>
```

Benefits of this approach:

- static loading (no ajax requests)
- works locally without a web server instance

Drawbacks of this approach:

- more requests are required
- more files to manage in complex setups (could be somewhat mitigated using something like Sprockets)

This is essentially how we build the full profile library, except that we deliver everything in one optimized file. When you write your own components, you can use this method - it also works with the full profile X3DOM file.

When to use this method:

- When you write your own components
- During development and testing

2.3.2 Dynamic component loading

X3DOM features a mechanism to load files at runtime. With this approach it is possible to load anything from anywhere and inject that code into your application. Be aware of this possible exploit when using the technique described here.

> **Warning:** In order to allow dynamic loading of components, you need to tell X3DOM to turn off its security precautions *before* including the X3DOM library. These precaution prevents the library from executing code that is known to be insecure. Only use this feature if there is absolutely no other option for you.

In order to disable security measures in X3DOM, put the following statement in your document `<head>` section and before the inclusion of X3DOM:

```
<head>
  <script>
    X3DOM_SECURITY_OFF = true;
  </script>
  <script src="x3dom.js"></script>
  ...
</head>
```

Now, dynamic loading components at runtime is enabled and can be used by putting the following parameters in you X3D scene.

```
<x3d>
    <param name="component" value="Primitives2D,Other"></param>
    <param name="loadpath" value="http://yourserver/path/"></param>
    ...
</x3d>
```

If *loadpath* is not set X3DOM tries to load the component from the documents parent URL.

Keep in mind that the dynamic loading of X3DOM components performs an **synchronous** Ajax request. As such all the limitations of Ajax requests apply, plus the library is blocking your browser until it gets a response.

Drawbacks of this approach:

- load order is important and has to be maintained by developer
- needs a web server running (ajax)
- blocks the browser during loading of files
- code injection possibility high
- needs much more requests
- ajax request caching not really supported

When to use this method:

- never (unless there's no other feasible way)

2.3.3 Extending X3DOM

In this chapter you will learn how to extend X3DOM with your own nodes which you can load using the methods outlined above. We recommend to use the static loading approach in combination with the core profile `x3dom.js`. This results in the inclusion of `x3dom.js` and `YourComponent.js` which will contain your custom code.

To follow this chapter you need at least basic understanding of the following concepts, principles, or technologies:

- object orientation
- class based object model
- programming in general
- Javascript programming
- the Javascript object model
- XML and HTML5

Object system

In order to register a new node within the X3DOM system, you need to create the equivalent of a *class* that inherits properties from a superclass. Javascript itself does not implement a class based object model, it provides a prototype model. A prototype based object model can be seen as a superset of a traditional class based model. With a prototype based object system, one can implement a more limited class based system. That is exactly what X3DOM does.

For each node you want to implement in X3DOM you need to call the function:

```
x3dom.registerNodeType("YourNodeName", "GroupName", definitionObj);
```

This registers a node within the X3DOM system and provides a hook to the implementation of this class. The first parameter also is the name of the XML tag you are writing code for. The third parameter to registerNodeType is the return value of a call to the X3DOM function:

```
defineClass(superclassObj, constructorObj, implementationObj);
```

This function is roughly equivalent to creating a class definition in a language with an traditional class based object system.

Note: The `defineClass` function resides in the global Javascript namespace whereas the `registerNodeType` function is nested within the `x3dom` namespace. This is intentionally so and not a typo.

Hello World

Let's say we want to implement a custom node which echos a "Hello World" to the console, we first need to decided how the XML should look like. In this case, we simply want another XML tag that looks like this:

```
<x3d>
  <scene>
    <hello></hello>    <-- this is new
  </scene>
</x3d>
```

Since there is no *Hello* node in the X3DOM system nothing happens when we run this X3D in the browser. The `<hello>` tag is not recognized and therefore ignored by X3DOM. In order to make X3DOM aware of the `<hello>` tag we need to register a new node with the system and provide an implementation for that node. In order to do so we are using the two function calls described above:

```
x3dom.registerNodeType(
    "Hello",
    "Core",
    defineClass(x3dom.nodeTypes.X3DNode,
        function (ctx) {
            x3dom.nodeTypes.Hello.superClass.call(this, ctx);
        }, {
              nodeChanged: function() {
                  x3dom.debug.logInfo('Hello World from the console');
              }
        }
    )
);
```

First, the hello node is registered with X3DOM, the hello node belongs to the core nodes. We then create an implementation object of the type `x3dom.nodeTypes.X3DNode`, the superclass. We also define a constructor for our node in form of a function object that we pass to the `defineClass()` function (second positional parameter). The last parameter consists of an object literal containing function definitions for the node API. In this example we implement a function called `nodeChanged` which will be called by X3DOM anytime there is a change to the node element in

the DOM. It is also called when the node is encountered the first time. This is the place where print a message to the console using the X3DOM debug facilities.

The `nodeChanged` function is not the only function you can pass your implementation. For example, there is a `fieldChanged` method which is called whenever a attribute in the DOM changes, and you can implement your own methods here.

More

For more examples of nodes, please refer to the source code of the X3DOM nodes. It's the best way to learn how to deal with the X3DOM node system.

CHAPTER

THREE

NOTES & HOW-TOS

Design notes, legal information and changelog are here for the interested.

3.1 Notes

Different topics of interest. This section should be seen as incubation space for collaboration and ideas. Ultimately things move up to the main documentation.

3.1.1 Loading resources from external servers

Sometimes it is desirable to load resources, like textures, from other locations than your web server. The most common use case being serving textures from a CDN. While technically not a problem, there are security mechanisms in place to prevent injection of malicious code into your application. Browser vendors started to block loading of resources originating from domains unless these resources marked safe by the foreign web server.

The corresponding W3C specification is called Cross-Origin Resource Sharing [CORS2010] and adhered to by most browsers. And in essence, you need to configure the foreign web server to add a HTTP header that marks the resource safe for your domain. Say, your application is served from `http://yoursite.org` and needs to load resources from `http://othersite.org`, the webserver of `othersite.org` needs to set a HTTP header that marks `yoursite.org` safe for cross site requests. For example:

```
Access-Control-Allow-Origin: http://yoursite.org
```

An alternative to adhering to the CORS protocol, is to setting up a proxy server forwarding requests to the foreign server in the background. If you can do away with the benefits CDN provides this technique may be a viable alternative.

More information on CORS and setting HTTP headers:

- Cross-Origin Resource Sharing
- Apache mod_headers
- Lighttpd mod_setenv
- NGINX headers module

More information on proxy configuration:

- Apache mod_proxy
- Lighttpd ModProxy
- NGINX proxy module

Developing locally

While the HTTP headers method presented above is the best practice in production environments, it is unpractical for local development of your application. Fortunately there are a couple of workarounds making you develop with pleasure.

- Use a real web server (e.g. Apache) to deliver your site locally
- Use a web server with proxy module to fetch external resources form a live website
- Use browser flags to disable security measures

The latter one being the most flaky. It is not guaranteed that the browser will support disabling security in the long run. Also strange behaviour in case of magically enabled security after updates in combination with browser caches.

Using a web server

Installing a web server locally and serving your files under the localhost domain is the best way of developing web applications. We also recommend you use this technique when developing with X3DOM. Using a full web stack locally ensures that your browser behaves the same way it would when loading a website over the internet. Requests are sent and received by the browser just like they would in a production environment. It is also the only way to properly test Ajax functionality and HTTP features, like expiry headers.

There are various ways to install a web server on your machine. In case of Mac OS X, Apache is already installed and you can just put your documents in your *Site* folder.

On Linux there are various ways to install Apache depending on your distribution. Most likely two or three commands should suffice.

Windows users are best served with a package called XAMPP, which also caters various Unix based systems.

Using a web server with proxy pass

What about external resources in local development, I want to develop locally and load textures from `textureheaven.net`. You could install as system wide proxy server, which processes the request and response to `textureheaven.net` and adds the correspoding CORS header to the response. Another straight forward way is to leverage the power of what you already have: your local web server.

The setup is more elaborate and out of the scope of this document. Here are some pointer of how to get started.

First you need to confiture your web server to answer requests to `textureheaven.net` instead of sending those requests to the real `textureheaven.net` web server. To do so you need to make an entry in your `/etc/hosts` file so the address does not resolve to the real site but to `localhost`. Within your web server configuration you now create a virtual host that answers requests to `textureheaven.net` and proxies them request to the real `textureheaven.net` site. In order to make this all work, you finally need to add a CORS header to the response (e.g. `Access-Control-Allow-Origin: http://localhost`)

Sounds too complicated? There's a shortcut way. But as with all shortcuts, use it with caution.

Disable browser security

If you have all resources locally, there is a shortcut for circumventing the CORS mechanisms. Please use with care.

Chrome The Chrome browser allows you to disable a security precaution that prevents you loading resources from disk. Use the following startup parameters:

```
--allow-file-access-from-files
--allow-file-access
```

Firefox Enter the following in your location bar:

```
about:config
```

Acknowledge the secuirty warning and enter the following in the filter bar:

```
fileuri
```

Look for the option called **security.fileuri.strict_origin_policy** and set it to **false**. +++ *Draft* +++

3.1.2 Complex models

While X3DOM is very well suited for small models up to a few kilobytes in size, big models can become a major problem. This section takes a look at the different aspects involved and tries to find partial solutions to the problem. The problems can be broken down to the following areas:

- Loading a HTML/X3D file containing a lot of data
- Parsing data into the DOM
- Storing data in memory

And server side:

- I/O when sending big files (sendfile)
- Server stalling because user presses "reload" endlessly when not informed that an operation is in progress and consequently exhausting free server slots.

While most of these problems are inherent to the domain we are moving in, and not X3DOM specific, measures can be taken to optimize loading of large chunks of data, especially cutting down transmission time over the network.

Another, more complex problem, is presented by the way JavaScript and DOM work. There is no manual memory management possible and freeing up memory is left to the garbage collector that runs at intervals out of our control. We can not even manually start garbage collection. And if we could, there is quite some overhead involved in garbage collection.

The only alternative to cope with the memory specific problem is circumventing the DOM entirely parsing. While this somewhat defies the purpose of X3DOM, it may present a viable alternative for performance critical applications. Various techniques and ideas are explored further in the following sections.

It is paramount to keep in mind, no matter how much we optimize, the use of complex models is limited by the following boundaries:

- Memory of client (storing more data)
- Processing power of client machine (parsing more faster)

In the following sections we are presenting various tools and techniques to optimize various aspects of loading big models.

Delivering deflated content (HTTP/1.1)

The most obvious idea is to compress the HTML/XML files. Luckily this is the easiest to implement and will improve performance of loading time significantly. Most web browsers support the deflate algorithms (ZIP) and can handle compressed files on the fly. The web server is configured to compress HTML files before delivering them to the client. By means of setting the HTTP (Accpet-Encoding), header, denoting a compressed file arrives, the client can act on this information and decompress the deflated file on the fly. Any browser that supports the HTTP/1.1 should be able to handle deflated input.

In order to enable your webserver to compress the files during transport, the configuration needs to be alterd. How to achieve this can be found in you web server documentation. For exmaple:

- Apache
- NGINX
- Lighttpd

If you are using a different web server, refer to its documentation.

Benefits

It is considered good practice for web development to enable in-memory compression for text resources like HTML, CSS, JS. Tests showed that file size can be reduced to about 50%. Transmission time should be typically cut in half. For exmaple, the vary large model of the walking soldier () is about 13MB in size. Using GZIP compression, this model is only 5.2MB big.

Drawbacks

This method does not present us with any significant drawbacks.

A slight overhead on server- and client-side processing power is inherent with on-the-fly compression. Caching techniques of web servers and browser mitigate the small performance hit of on-the-fly compression.

An actual benchmark of decompressing the soldier model has not yet been conducted. However the ventured guess is that the savings of network time outperform the decompression algorithm which runs naively.

For very large files this technique may not be beneficial since the server may block too long during compression or the client, especially with slow clients, may take to long to decompress. This however needs to be verified and tested.

We recommend to enable compression on your web server and only out out if there are performance hits than can be attributed to compression.

Makeing use of browser side caching

- Etags
- Expire headers
- HTML5 offline stuff?
- etc.

Using asynchronous loading (aka Ajax)

X3D inline Ajax

Benefits

Drawbacks

The most significant drawback of the current XMLHttpRequest object implementations is the complete ignorance of the HTTP Accept-Encoding header. While lazy loading geometry data is possible using either the X3D inline element or custom code to load a model and modify the DOM, the lack of compression makes this process rather slow.

Using geometry images

- http://research.microsoft.com/en-us/um/people/hoppe/proj/gim/
- http://graphicrants.blogspot.com/2009/01/virtual-geometry-images.html
- http://www.cs.jhu.edu/%7Ebpurnomo/
- http://www.uni-koblenz.de/~cg/Studienarbeiten/gpu_mesh_painting_ritschel.pdf
- http://wiki.secondlife.com/wiki/Sculpted_prim

Benefits

- Uses canvas to decode (native speed)
- Circumvents the DOM for better performance

Drawbacks

- Circumvents the DOM
- Use cases limited to simpler geometry?

Web server optimizations

Optimization of a web server is not exactly a core topic of X3DOM. To give you a starting point, we collected some resources that should get you going:

- Apache
- YSlow
- yada, et cetera et al.

3.1.3 Platform Notes

System requirements and browser notes

In order to be able to work with X3DOM, you need a WebGL enabled web browser. We also support different fallback models for browsers not supporting WebGL natively. The best support of features however is only ensured with a browser sporting a WebGL implementation. You can check the status of supported browser here.

Chrome

Recent releases of Chrome require you to enable WebGL. Please use the following command parameters when launching chrome:

```
--enable-webgl
--use-gl=desktop
--log-level=0
--allow-file-access-from-files
--allow-file-access
```

The last two options enable the browser to load textures from disk. You will need this if you are developing your site locally.

Platform notes (OS/GPU/Drivers)

While WebGL is supported in most current browsers, there are various little differences in operating system and graphics driver support. We can not discuss any possible OS/GPU/Driver combination here, but you might find some valuable hints if you can not get WebGL up and running on your system. With all systems be sure to use latest drivers for your graphics card and be sure those drivers support the OpenGL standard.

Blacklists

Some GPUs and Driver combinations are blacklisted by browser vendors. In case of strange errors, please check the following lists for:

- Mozilla
- Chrome

Windows

Chrome Frame

Currently, the only way to use WebGL with Internet Explorer is by using the Google Chrome Frame plugin. In order to make X3DOM use the WebGL renderer with Internet Explorer, you need to install Chrome Frame and enable it in your HTML or web browser configuration. The most simple way to enable ChromeFrame is to put this line in your HTML head section:

```
<meta http-equiv="X-UA-Compatible" content="chrome=1" />
```

Download and furhter reading:

- Chrome Frame
- Getting started

Mac OS X

Safari 5.1+ is supporting WebGL, however you need to enable it in the Developer menu. This menu is invisible by default. Go to “Preferences” (Cmd-,) and select the “Advaned” tab. Enable the option “Show Develop menu in menu bar”.

Rubber band scrolling in Mac OS X 10.7 Lion

On Mac OS Lion, with Apple input devices scrolling behaves differently. When reaching the end of a page, a rubber band effect kicks in. This behavior is also present on iOS devices.

If you don’t like the effect, you can turn if off using a CSS rule:

```
body { overflow: hidden }
```

Keep in mind that this rule changes the default behavior of your browser and scrollbars might disappear entirely. It is only a workaround and the preferred fix is to wait for Apple to provide a switch to turn this functionality off. Also note that the rubber band scrolling might not be visible at all with non Apple pointing devices.

Ubuntu Linux

In order to enable WebGL for Firefox 4 and above you need to:

1. Install the libosmesa6 package. You can do so by issuing the the following command in a terminal window or one of the consoles:

   ```
   sudo apt-get install libosmesa6
   ```

2. Open the Firefox application and enter *about:config* in the location bar and *webgl* in the filter box.
3. Set the option *webgl.force-enable* and *webgl.prefer-native-gl* to *true*
4. Set *webgl.osmesalib* to the the path of the library you installed in step 1, usually thi should be: `/usr/lib/libOSMesa.so.6`
5. Restart Firefox

3.1.4 Compatibility Matrix

This section should contain a collection of known working/not working combinations of browsers, GFX cards, operating systems, etc.

There's much to do, so we'd appreciate any feedback from you.

	Mozilla 5/Win	Mozilla 5/Mac	Safari/Mac	Safari/Win	IE9
NVIDIA GeForce 9400M 256 MB		WebGL	WebGL	No	Flash 11

3.1.5 FAQ

Q: Why declarative 3D in HTML? Can I not just use WebGL, O3D, etc. WebGL, O3D or [your favorite 3D-JS lib] are made by gfx-coders for gfx-coders who know how to deal with a 4×4 transformation matrix and speak GLSL as their first language. Most web-page developer have a different profile. They just would like to build their 3D-UI, Visual-Analytics or Web-Shop application and want to utilize some 3D technology.

If you build the next high-end browser-based online game or visualization demo then use WebGL or O3D. But if you simply need some 3D elements in your Web-App then try X3DOM.

Q: Why X3D? Can I not just use Collada, VRML, OBJ, etc. 3D (X)HTML-ized graphics requires a royalty-free, open and standardized XML-encoded format. Therefore Collada or X3D are the best candidates (however you can easily convert VRML to X3D-XML).

Collada is really designed as interchange and intermediate format to transport and manage your 3D data. The Collada specification does not include, unlike X3D, a runtime or event model, which is needed for per-frame updates on the 3D-side (e.g. animations). Therefore this project uses a well defined subset of X3D (called Profile in the X3D-World) for X3DOM. For more background information about how Collada and X3D relate and why **"X3D is ideal for the Web"** please read the Whitepaper by Rémi Arnaud and Tony Parisi.

Q: Why JS? Can you not write the system/plugin in C++, Java, ...

Well, the developer of this project worked on different native commercial and open-source X3D-runtimes before. The limitations of the current plugin interface (e.g. how to monitor DOM-changes) would make a implementation hard or even imposible.

In addition, we just wanted an open source test environment, which is small, works without any plugin and can be easily modified to inspire and follow the integration-model easily. And first tests showed that the increasing performance of JavaScript and WebGL would lead to impressive results.

Q: Why any standardization and native implementation if there is already the X3DOM-JS runtime?

Short answer: Feature and Speed. The current JS/WebGL layer does not allow to implement e.g. spatial Sound or specific image loader and although WebGL is an impressive step forward we still have to do all the scene management and updates in JS.

3.2 Release notes

3.2.1 Version 1.3

- Flash
-

3.2.2 Version 1.2

After many months of work, we're proud to announce the release today of X3DOM 1.2. There's plenty of cool stuff we have worked on since the last release. You can also swing by the downloads page to grab a copy of the release package:

http://x3dom.org/x3dom/release/

Major Feature: Flash11/MoleHill Render backend

The new version supports an additional render-backend. The system now looks for a Flash11 plugin if the browser does not provide a SAI-Plugin or WebGL interface. This enables the user to view X3DOM pages, with some restrictions, using the Internet Explorer 9 browser. It utilizes the Adobe Flash MoleHill 3D-API for GPU-Accelerated rendering. To use Adobe Flash-based X3DOM you need the latest Adobe Flash Player Incubator build (11.01.3d).

Our `x3dom.swf` must either be placed in the same folder as your xhtml/html files, or you can specify a path with the swfpath-Parameter of the X3D-Node. With the X3D-Nodes backend-Parameter you can set Flash as default backend.

At the moment the Flash-backend is still in an early beta state with most of base nodes working. Special features like multiple lights, shadows, CommonSurfaceShader, etc. are being worked on

Additional Changes

- Internal reorganization of sources. Node type have now their own submodule (src/nodes) and are split into groups.
- Text node is partially working now. You can use any font available to the browser. This includes WebFonts of CSS3.
- Added system to pass parameters to runtime. The use attributes with the X3D element will be deprecated in 1.3. Currently supported parameters
- Partially working MT support with on Firefox beta. Experimental.
- It is now possible to directly apply CSS styles to the X3D element.
- Fixed display problem with textures which have dimensions not to the power of two.

3.2.3 Verison 1.1

Second stable release after almost 7 month.

http://x3dom.org/download/x3dom-v1.1.js

Most features from the 1.1. milestone are implemented:

- Unified HTML/XHTML encoding
- HTML5 <canvas>, <img> and <video> as texture element supported
- CSS 3D Transforms
- Shader composition framework
- multiple lights and support for Spot-, Point- and DirectionalLight
- Fog
- LOD
- Support for large meshes
- Improved normal generation
- Follower component
- WebGL compatibility
- The proposed HTML profile is almost implemented
- The fallback-model has changed a bit. We partially support X3D-SAI plugins now and removed O3D as technology.

Recently there have been several changes in the WebGL-API (e.g. interface changes in texImage2D() method, replacement of old WebGL array types with the new TypedArray specification, enforcement of precision modifiers in shader code, and changed behavior of NPOT textures). This leads to incompatibilities with previous versions, why the 1.0 X3DOM release does no longer work together with recent browser versions.

3.2.4 Version 1.0

First stable release of the framework.

http://x3dom.org/download/x3dom-v1.0.js

All initially planned features are implemented:

- HTML / XHTML Support
- Monitoring of DOM element add/remove and attribute change
- JS-Scenegraph synchronizer
- ROUTEs
- DEF/USE support
- External Subtree (via X3D Inline)
- Image (Texture), Movie (Texture) and Sound (Emitter) support
- Navigation: Examine
- WebGL Render backend
- The proposed HTML profile is partially implemented.

Full release note

3.3 License

X3DOM is dual licensed under the MIT and GPL licenses:

```
==[MIT]===============================================================
Copyright (c) 2009 X3DOM

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.


==[GPL]===============================================================

X3DOM - Declarative 3D for HTML

Copyright (C) 2009 X3DOM

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

BIBLIOGRAPHY

[CORS2010] Cross-Origin Resource Sharing, W3C, 2010. Available online at http://www.w3.org/TR/cors/