X3D Overview

Michalis Kamburelis `<michalis@castle-engine.io>`

Table of Contents

1. Introduction

1.1. What is X3D?

It’s a great format for 3D (and 2D) models.

It’s **open**, which means:

- The precise specification is public.
- The specification is really **not** tied to any particular implementation. There are numerous implementations of X3D, open-source and proprietary, and no implementation has any special priority for X3D specification.

- Authors check that **nothing** in the spec requires technology covered by non-free patents.

And it allows to define an incredible number of features. E.g.

- Meshes with materials and texture (obviously).
- Animation of **anything** (not only transformations, also e.g. material parameters or texture coords, also animation by skinning, mesh deformation etc.).
- Custom shaders.
- Interactive fetures (you can define sensors, e.g. `VisiblitySensor` or `ProximitySensor` or `TouchSensor` and express how world reacts to them e.g. *"the door opens when you press a handle"*).
- As of X3D 4.0 (we are working on it right now) the materials and lighting have all the modern features you expect, so we support 3 lighting models (Phong, Unlit, Physical). Physical (PBR) lighting model is completely and deliberately compatible with glTF 2.0, so you can express glTF content in terms of X3D nodes without losing anything.
- Many optional components e.g. for sound, NURBS, particles etc.

1.2. Why this book?

There are two practical problems with X3D, and I want to fix them:

1. Exporters from 3D modeling software are not maintained actively enough. For example:
    - Blender 2.8 exporter lacks some crucial features, like textures and animations.
    - There is no actively developed exporter for 3ds Max, as far as I know. You have to export to VRML 97, which is like an older brother of X3D (more than 20 years old now).
    - There is no actively developed exporter for Maya, as far as I know. There once was, but for an older Maya version, before 2008.
2. The X3D specicification is **very large**. The main spec is *Extensible 3D (X3D) — Part 1: Architecture and base components*, on https://www.web3d.org/documents/specifications/19775-1/V3.3/Part01/X3D.html . It describes all the X3D nodes. It's already huge… and it's not everything: you also need to know some X3D "encoding" (which is "how to write X3D nodes to file"), like an XML encoding on https://www.web3d.org/documents/specifications/19776-1/V3.3/index.html .

1.3. Who is this book for?

- If you want to read or write X3D, this book is for you.

  Maybe you want to read / write X3D in your own software. Maybe you want to read / write X3D by hand (to a large extent, you can easily modify X3D files in a text editor, in XML and classic encoding).

- If you think that X3D features and ecosystem are great, but the X3D specification looks too complex to understand, this book is for you.

1.4. Who am I?

This book is written by Michalis Kamburelis (and contributors are most welcome — go to https://github.com/michaliskambi/x3d-overview).

I'm the author of *Castle Game Engine* (https://castle-engine.io/), a cross-platform 3D and 2D game engine, that uses X3D for our "scene graph" (which means that almost everything you render is expressed using X3D nodes). We support a number of 3D model formats - not only X3D, also glTF, Collada, Spine JSON, and many others. (They are all converted into X3D nodes at import.)

I worked with X3D for many years now, and helped shape the current X3D 4.0 specification.

I'm also the author of *view3dscene* (https://castle-engine.io/view3dscene.php), which is a model viewer using *Castle Game Engine*, thus supporting X3D, glTF and all the other model formats. You can use this viewer to test example models presented in this book.

While I encourage you to try *Castle Game Engine* and *view3dscene*, they are not the only way to play with X3D, and you're welcome to try others. E.g. X_ITE and X3DOM implement X3D in a browser (so you can embed X3D content inside a webpage) — they are cool, go check them out! FreeWRL is another popular open-source browser for X3D. There are more — see TODO-web3dresources-link.

2. X3D files

2.1. How to view the example files

Save files in XML encoding (most of the examples in this book) as `xxx.x3d` file on your disk.

Save files in classic (VRML) encoding as `xxx.x3dv` on your disk.

You can edit these files in any text editor. For XML encoding, I encourage using some text editor capable of highlighting XML syntax (elements, attributes).

Then open them:

- On desktop, you can use a number of X3D browsers, like view3dscene (https://castle-engine.io/view3dscene.php) or FreeWRL (TODO).
- On mobile, view3dscene-mobile is coming soon. Alpha release on TODO.
- On the web, you can use X3DOM or X_ITE. Examples are on TODO.
- Many more software to render X3D is available and listed on TODO-web3dresources.

2.2. XML Encoding

simple example with header of X3D 4.0

2.3. Classic (VRML) Encoding

simple example with header of X3D 4.0

3. X3D Features

3.1. Meshes

IndexedFaceSet, primitives

3.2. Materials

Material, UnlitMaterial, PhysicalMaterial

3.3. Lights

XxxLight nodes

3.4. Textures

ImageTexture

BTW, data URI to embed data

CubeMapTexture

3.5. Transformations

Transform

3.6. Animations

TimeSensor and interpolators (see CGE explanation)

3.7. Shaders

You can write custom shader code, for example using *OpenGL Shading Language*.

Note: X3D specification standardizes how to use various shading languages, and pass parameters ("uniforms" in shading language terminology) to them. But when writing shaders you will encounter various browser-specific considerations into account. Various browsers supply uniform values under various names, e.g. in *Castle Game Engine* they are named `castle_xxx`. Differences go much deeper than naming, though: similar data is sometimes expressed using a different approach, packed differently and precalculated on CPU differently. And various browsers allow various versions of the shading language (due to using OpenGL, OpenGLES or other rendering libraries like Direct3D, Vulkan etc.). You can supply multiple alternatives for a shader code, to account for various browsers.

example of ComposedShader

example of ComposedShader with a texture

In *Castle Game Engine* and FreeWRL you can also use a different approach to writing shaders, where you supply only a part of the shader code that declares a function called `PLUG_xxx`. This shader code will be merged with the browser shader code, and the `PLUG_xxx` called when necessary. This allows to write shaders that easily integrate with browser rendering, e.g. you can enhance the look of some texture, while keeping the rest of the operations (lighting, shadows) working as by default.

example of Effect

4. About this document

Copyright Michalis Kamburelis.

The source code of this document is in AsciiDoc on https://github.com/michaliskambi/x3d-overview. Suggestions for corrections and additions, and patches and pull requests, are always very welcome:) You can reach me through GitHub or email michalis@castle-engine.io[1]. My homepage is https://michalis.ii.uni.wroc.pl/~michalis/. This document is linked under the *Documentation* section of the *Castle Game Engine* website https://castle-engine.io/.

Thank you for reading!

[1] mailto:michalis@castle-engine.io