Web 3D Formats

**Table of Contents**

1 Introduction 2

2 Exporting VRML from 3ds Max with VRML Helpers 4

2.1 Animate the Coke can 5

2.2 Add VRML Helpers to the Coke Scene 13

2.2.1 VRML Helpers — Touchsensor and NavInfo 14

2.3 Export to VRML97 20

3 Rendering with VRML and X3D 23

3.1 Web 3D Format Workflow 24

4 Convert VRML97 based 3D model to X3D 31

4.1 Test your VRML first 31

4.1.1 Convert with the Instantreality Online Transcoder 32

4.1.2 Using AOPT to Create Optimized X3DOM Content 37

4.2 Convert VRML97 direct to X3DOM for rendering in HTML5 38

4.3 Convert X3D to X3DOM for rendering in HTML5 42

5 Integrating VRML, X3D and X3DOM models into a 3D App 46

5.1 Write X3D into the HTML5 DOM 46

5.2 Embedding a VRML/X3D plugin into the web page 47

# Introduction

Lab 3 has been designed to introduce you to the process of converting your 3D models from 3ds Max into a Web 3D format, e.g. X3D, that will allow you to display your 3D model and interact with it on the Internet, i.e. in a responsive and mobile first Web 3D application (your 3D App). To do this you will need to convert your 3ds Max model into a compatible 3D format that will allow you to integrate your 3D models into your 3D App.

There are several ways to allow 3D to be rendered in your 3D App, but we will largely focus on exploiting a technology referred to as [X3DOM](http://www.x3dom.org/) — it is well worth browsing through the online catalogue of examples that you can access from the carousel! We can think of X3DOM as an open source JavaScript framework used to create 3D scenes in web pages. Or, quoting directly from the X3DOM website, X3DOM allows us to:

“Integrate 3D content seamlessly into your webpage — the scene is directly written into the HTML markup. No Plugins needed. Simply include a JavaScript file”.

By scene, we mean your 3ds Max models, etc. This is quite powerful stuff, IMHO, because as implied, if the 3D model is written directly into the web page, it must have some form of a tag (or node) structure, and if so these tags or nodes can be directly manipulated with JavaScript. Indeed, as the quote says, X3DOM has a JavaScript file developed to manage this embedding of, what is actually, an X3D model in the HTML5 DOM.

This is really good, but we will also utilize some popular web-based VRML/X3D players (some of which can also be used as web browser plugins) for testing your 3D models first, and after that we will exploit X3DOM as the Web 3D format of choice. We will initially focus on how to use VRML (Virtual Reality Modelling Language) and its successor X3D (championed by the [Web3D Consortium](http://www.web3d.org/x3d/)) because generally speaking 3ds Max does not export direct to X3DOM — well, as you will learn, you can get a plugin for 3ds Max to export, but it only works with very early versions of 3ds Max.

So, we will use VRML first, to test if our 3ds Max models can be used in a web browser, simply because 3ds Max only has a VRML97 Exporter by default. But, also because 3ds Max has some interesting VRML Helpers, which may be useful for in scene interaction with your 3D model.

It is worth noting that some other 3D authoring packages do export to X3D, and you can in fact install [InstantExport](https://www.x3dom.org/documentation/tutorials/3ds-max-export/) into older versions of 3ds Max up to 2014. This X3D exporter is provided by [instant**reality**](http://www.instantreality.org/).However, working in 3ds Max 2017, and I do not know if the 2014 version will work 3ds Max 2017.

So, in this tutorial you can try 3 methods to arrive at an X3DOM format.

1. Use 3ds Max VRML Export, then convert to X3D or X3DOM using an instant**labs** online tool
   * This is the way I usually do it, for convenience, but sometimes the Instantreality.org server is down.
2. Use 3ds Max VRML Export, then convert to X3D or X3DOM using an instant**labs** command line (aopt) tool.
   * The command line tool should be installed in the labs as part of the instant**reality** framework, which also includes the instant**player**. You can also download this framework and install at home: <http://www.instantreality.org/downloads/>. It is a good idea to install this on your home PC or laptop.
3. Try to install the Instant**Export** plugin, to convert to X3D or X3DOM, as mentioned above.
   * I did use this method up to 3ds Max 2013, but since upgrading to 2017 I have not tried it. So, currently I don’t use this method because the first two methods work ok. Feel free to give it a go, the download site is: <http://www.instantreality.org/downloads/dailybuild/?dir=/InstantExport> you will see that the last build was done 25 Sep 2014. It might well be that this works for 3ds Max 2017, unfortunately I haven’t tested it — if someone wants to give it a go and it works, spread the news. If it does work, then it’s ok to use it, but be aware that is not installed in 3ds Max 2017 in the labs.

There are also other possibilities for inserting 3D into a responsive mobile first web page, way too many to consider. For example, the [Unity](http://unity3d.com/webplayer/) gaming engine has a web player (no longer supported though), and you could also code at the WebGL level: <https://get.webgl.org/>. However, these methods are beyond the scope of this module. Instead, this module will exploit X3DOM, which is effectively an abstraction on top of WebGL anyway.

In the context of this laboratory we will focus on integrating 3D into your 3D App using [X3DOM](http://www.x3dom.org/) with its associated CSS3 and JavaScript libraries for embedding or writing X3D code directly into the HTML5 DOM (Document Object Model) — you will appreciate that this method is very effective, and will ultimately be the key method you will use for your assignment.

Primarily, for this Lab 3, we will focus on exporting your 3 models created in Lab 2 (i.e. your Coke can, Sprite bottle and Dr Pepper cup) as VRML97 objects from 3ds Max. So, we will export to VRML97 and then test this VRML model in a VRML player such as the Cortona3D Viewer, instant**player** or BSContact player — I don’t think the BSContact player is installed on the lab image. However, you should be able to download it and install in your local drive, my guess is ITS will have shut the door on that one now. Nevertheless, you can install it on your own machine.

Here is another X3D viewer you can also use called Xj3D, which is built into an X3D Editor, called X3D Edit. X3D Edit should also be installed in the lab image, but I tend not to use it — it would probably be very good if you were developing X3D models from scratch via code, rather than developing them in 3ds Max and converting to X3D. We will only need to do some simple modifications to the X3D code, and 9 times out of 10 we can probably avoid even this, so a lightweight text editor will do, particularly if it recognizes XML tags.

We will then, in this lab, investigate converting your models to X3D and again test these in the Instant**reality** player or BSContact player (note Cortona3D Viewer does not support X3D). Along the way, we will investigate adding VRML helpers to provide some interaction with the 3D model.

A point to note is that X3DOM does not, at the time of writing, support directly some of the VRML Helpers that are available in 3ds Max, in particular the Touch Sensor, so we need to use JavaScript (e.g. with an HTML onclick button) to trigger any touch based interactions, such as animations that would have been triggered by a VRML TouchSensor node. However, that is cool, that is what we want to do anyway, i.e. we want to manipulate X3D nodes with JavaScript to get useful interactions in your 3D App.

Finally, we will convert either your VRML into X3DOM (and inline X3D) to demonstrate how a plugin approach, such as the Cortona3D Viewer and BSContact players can be eliminated, as being inferior in many ways, from the 3D App — nevertheless, they are useful for testing the 3ds Max exported 3D model (VRML).

As you get towards the end of this Lab 3 you should be able to appreciate the advantages of integrating an inline X3D model into the HTML5 DOM against that of embedding a VRML97 model into the HTML5 web page using a plugin — the difference is striking! Then, we will finish off by implementing a very simple Bootstrap based template to integrate your three X3D models into the HTML5 DOM — the makings of a 3D App, no less!

Let’s start by looking at how to export your 3D models from 3ds Max with the VRML97 Exporter.

# Exporting VRML from 3ds Max with VRML Helpers

In this tutorial, we explain how to export your 3ds Max model into VRML using the VRML97 Exporter in 3ds Max. First, we’ll use your Coke can model, you can then apply the same principles for your other models, e.g. the Sprite bottle, Dr Pepper cup, and any models you create for your assignment. We will also cover the basic principles behind interacting with a 3D model through the use of VRML helpers. VRML helpers are used in linking actor objects or actions to object model animations, for example. There are several useful VRML helpers you may want to use to add some kind of interaction to your 3D model while it is rendered in your 3D App (as opposed to web page interactions that manipulate your 3D model). Further, several of these VRML helpers convert to X3D well, while some do not — obviously don’t use those that do not convert well.

This tutorial also covers the setup of an actor camera that will be used in a 3D viewing application (such as the Cortona3D Viewer, BS Contact or X3DOM) to navigate the virtual environment and better visualize your object in the context of a 3D scene (rather than just a single object). For example, you could imagine walking around several objects triggering animations by proximity or touch. This tutorial only covers a few basic VRML components and requires a basic understanding of 3ds Max, which you already gained in Lab 2.

To explore VRML Helpers before we export to VRML, you need to have a 3ds Max model that you can animate in some simple way, e.g. an animation being something interesting that can be triggered through, say a touch sensor, so make sure you have finished at least the Coke can, and ideally the Sprite bottle and Dr Pepper cup models as well. We will animate the Coke can in a simple way using a 3ds Max key frame animation. We will then attach a VRML touch sensor to the scene to allow us to trigger the animation.

Tip! Don’t get wrapped up in animation, this is not a 3D Animation module, however being able to trigger, from a web page button, some simple animations such as rotating your 3D model in the assignment is a good interaction feature. Any more detailed animations will depend on your 3D model and may or may not be worth your effort.

## Animate the Coke can

Remember, this lab tutorial is not meant to focus on key frame animation as such, we are simply using animation to illustrate the use of some of the VRML helpers!

So, let’s Figure out how to create the simple animation first.

1. Open up your Coke can model you created in 3ds Max in Week 2. At this stage, you should have a textured Coke can, if you haven’t finished it yet, then you are getting behind and will need to catch up quickly! Conversely, feel free to adapt this tutorial and use your Sprite bottle or Dr Pepper cup if you have finished these. I’ll use my coke\_final.max model in this tutorial first.
2. Set up your Coke can in the viewports so you can see your Coke can ok, see Figure 1, and then save it as a new file for your animation. This ensures you still have a copy of your final lab 2 Coke can 3D model. You will also need to ensure you have cameras set up that can see the animation.

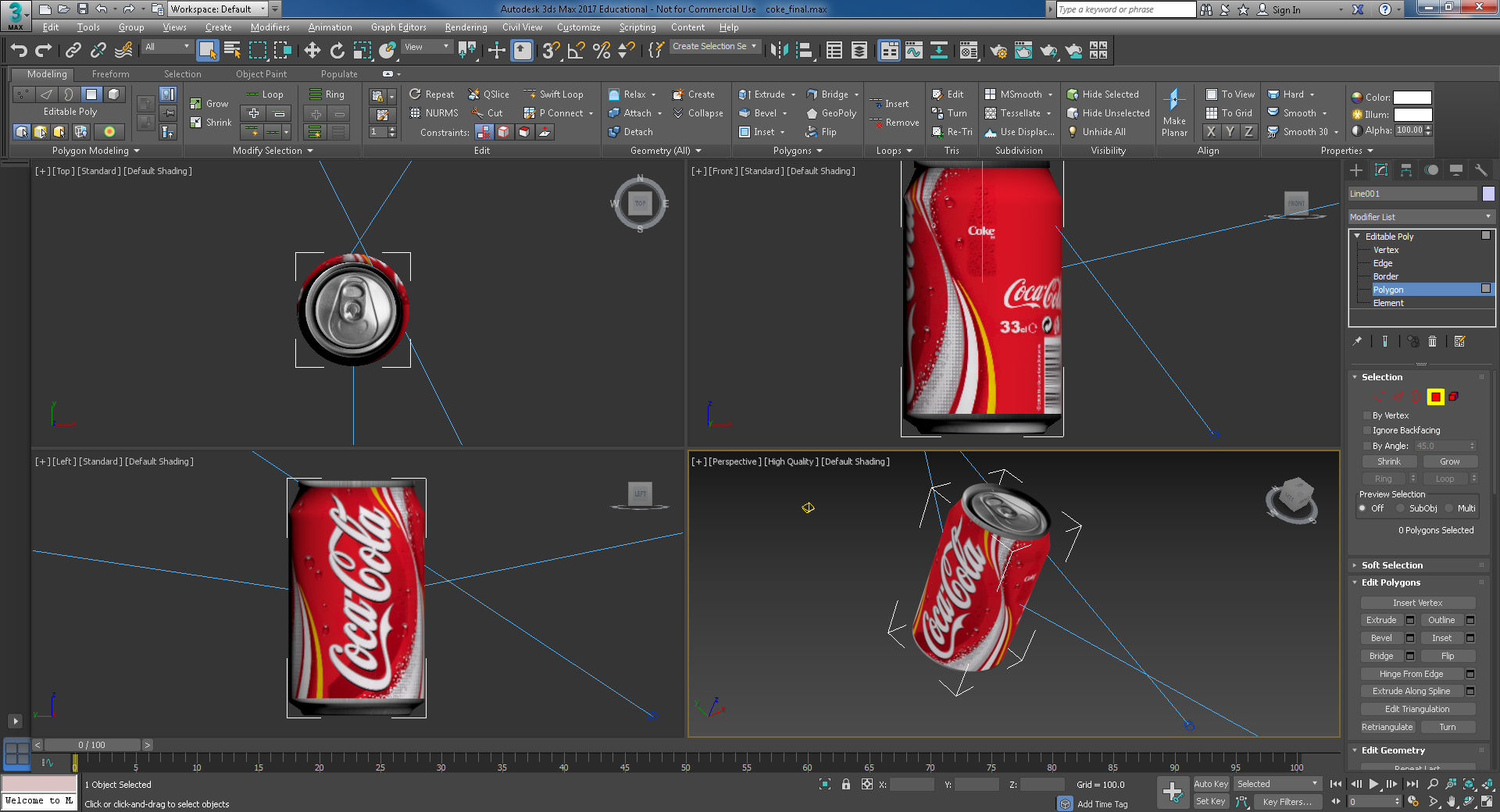


Figure 1: Coke can animation to be triggered by the VRML Helper: Touch Sensor

1. Using an appropriate viewport so that you can see the animation move from left back to 0,0,0, move the Coke can to the left a short distance. We will animate the Coke can so that it returns to its original position X,Y,Z = 0,0,0. So for example, if you are looking at the front viewport (in my case), move the can to the left as shown in Figure 3. It doesn’t matter how far, in my case I moved it -3000 units in the X direction. You can move the Coke can to the left in several ways, e.g. use the Select and Move after clicking on the can, and observe the X value increase to -3000 or simply set X to -3000, see Figure 2. Zoom to get a reasonable viewport. Your values may be different depending on the Units you set up when you modelled your 3D model.



Figure 2: X set to -3000

Also, looking ahead, I have changed some of my camera parameters. My scene happens to have 3 target cameras, all roughly looking at the Coke can when it is located at 0,0,0. So, I have set two of the target camera targets to 0,0,0. I have then set the third camera target about midway between the new position of the Coke can and 0,0,0 and changed it lens to get a wider view (35 mm lens). This camera will see the whole animation, hopefully, see Figure 3.

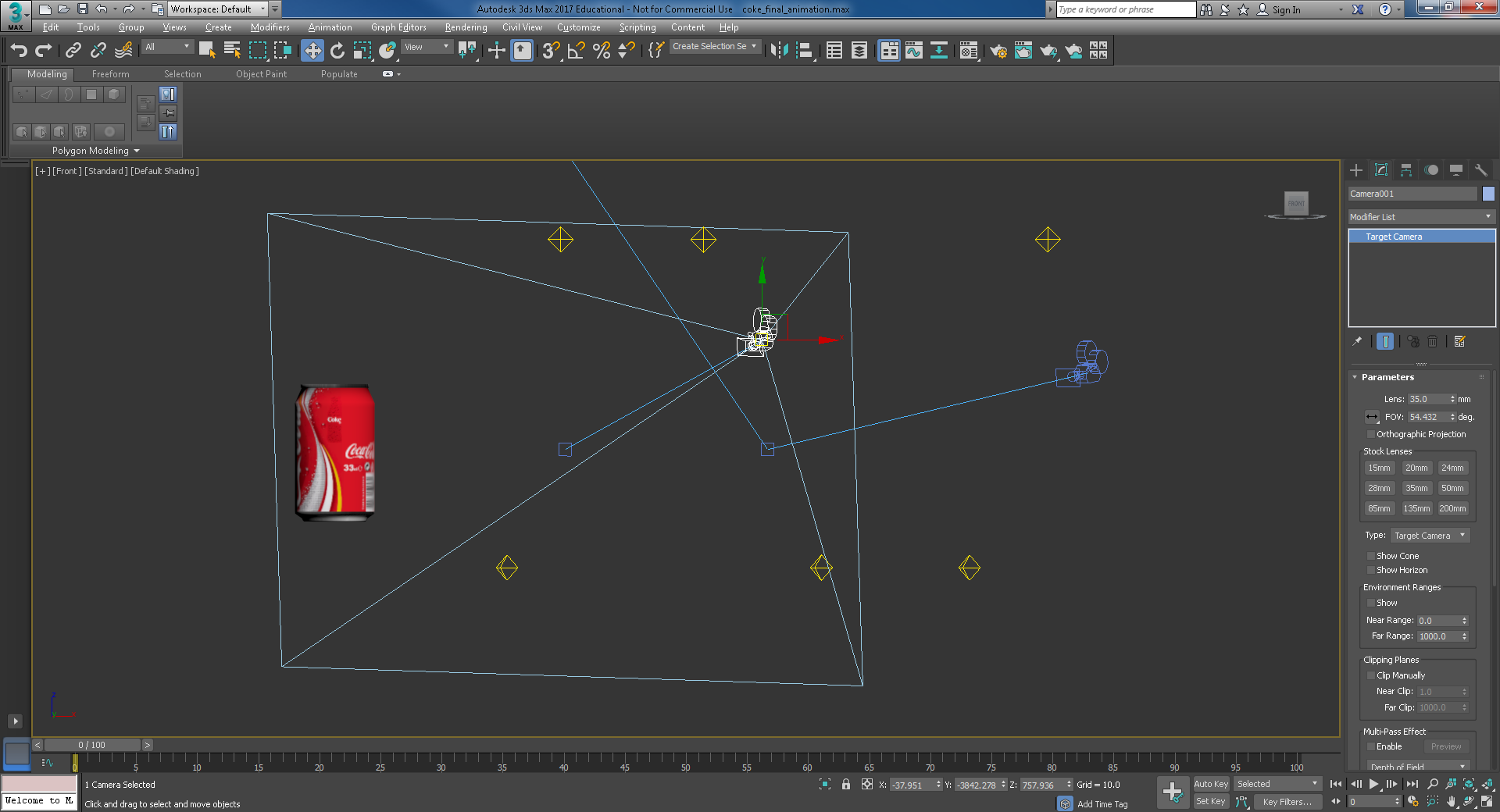


Figure 3: Coke can moved to the left as seen from the front viewport, with cameras adjusted

1. Now we will return it to 0,0,0 using a key frame animation. Select the auto-key, which will make the (front) viewport outside frame turn RED, and the key frame tool highlight in RED, see Figure 4.

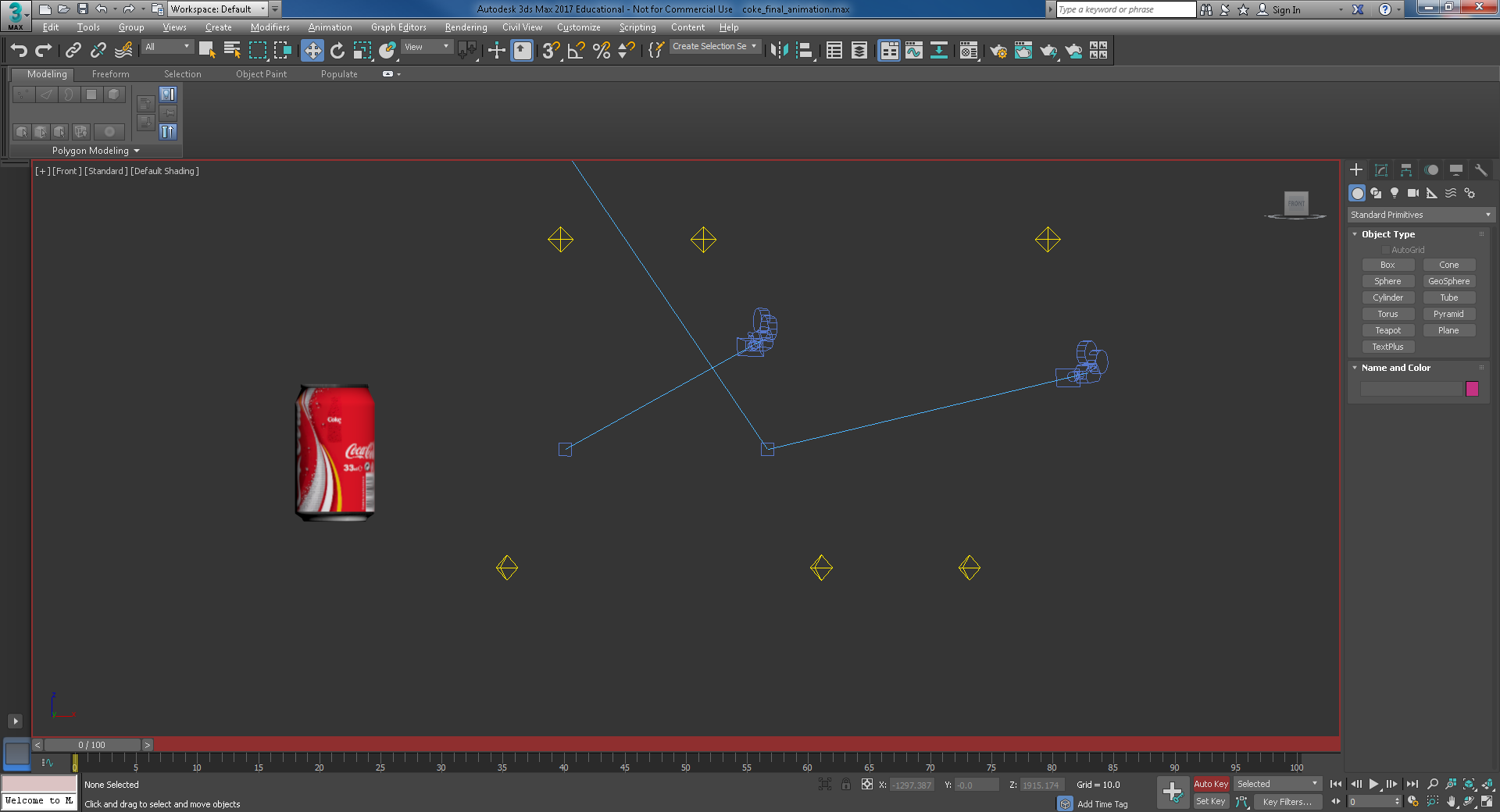
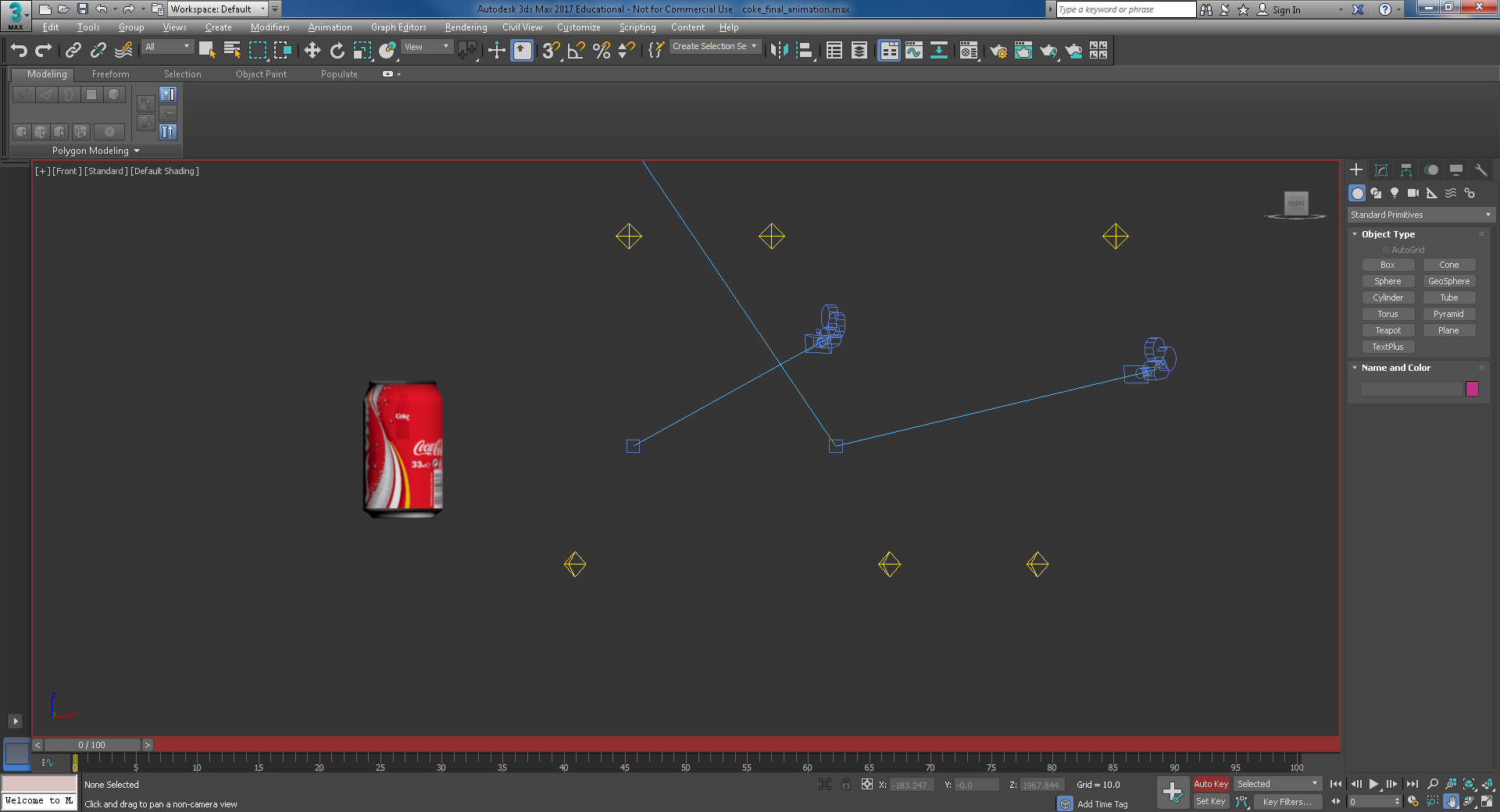


Figure 4: Selecting Auto-key

1. Press the Set Keys icon (just to the left of the Auto Key — it’s the Key symbol) to sample the Coke can’s initial position, see Figure 5, then move the time slider, with auto-key still on, to the time destination of your first animation movement, say 10 frames, see Figure 6. For your first animation movement, do something simple like raise the Coke can to do a hop to about halfway back to its original position of 0,0,0. You can do this with the Select and Move button — in general you might like to keep it simple with a combination of translate and rotate actions to affect your moving animation.



Initial animation position

Figure 5: Initial animation position sample with the Set Key

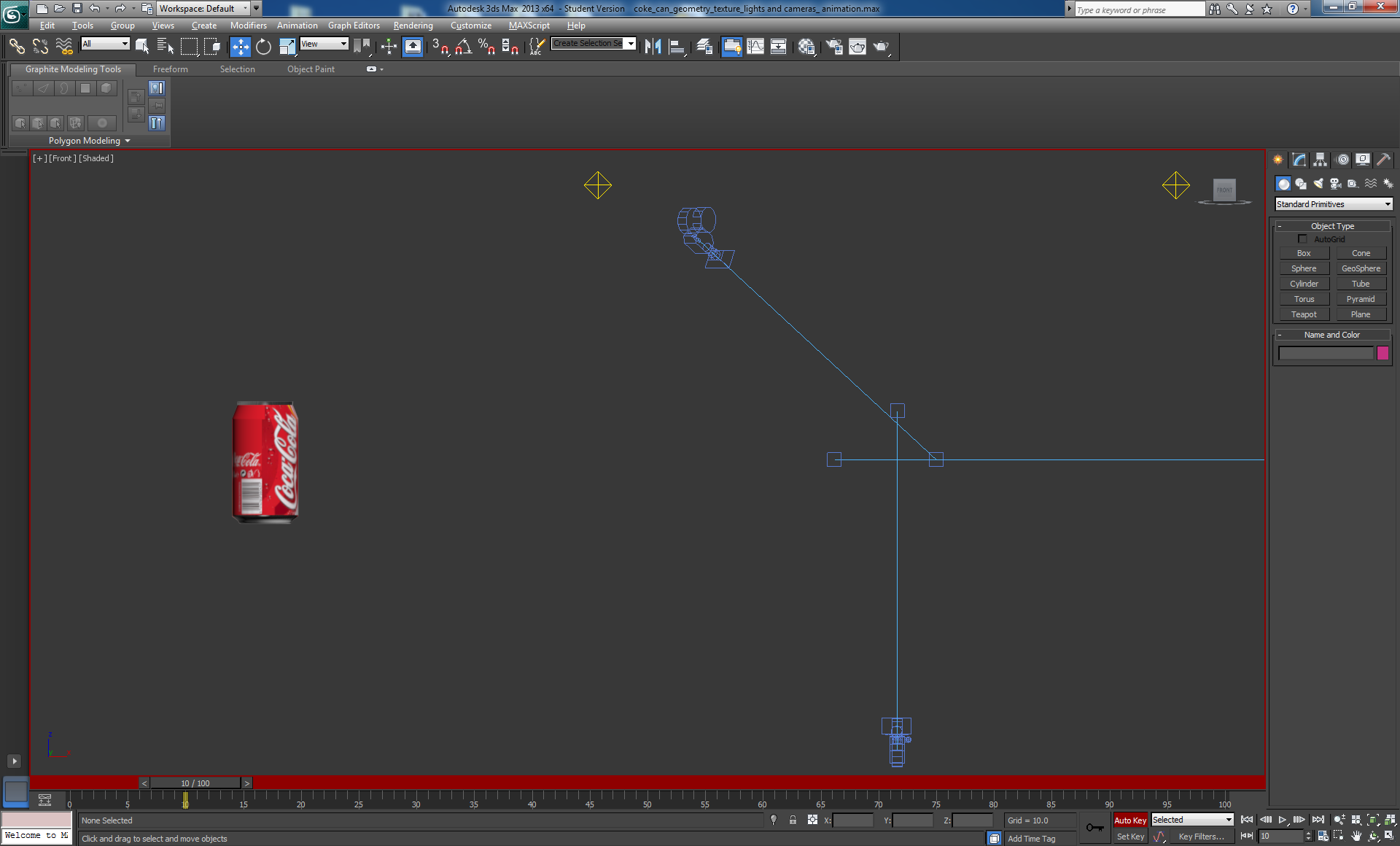


Figure 6: First animation up to 10 frames

1. Repeat the process every 5 or 10 frames, creating some animation until about frame 90.
2. Turn auto-key off (it is important to stop recording so you don’t mess up your animation), then play your animation — you can play the recorded animation by scrubbing the animation slider back and forth, see Figure 7 and Figure 8, which shows the can at frame 75/100 of the animation that I did (yours will be different depending on how you made the can animate). Or you can play continuously by selecting the Play Animation button.

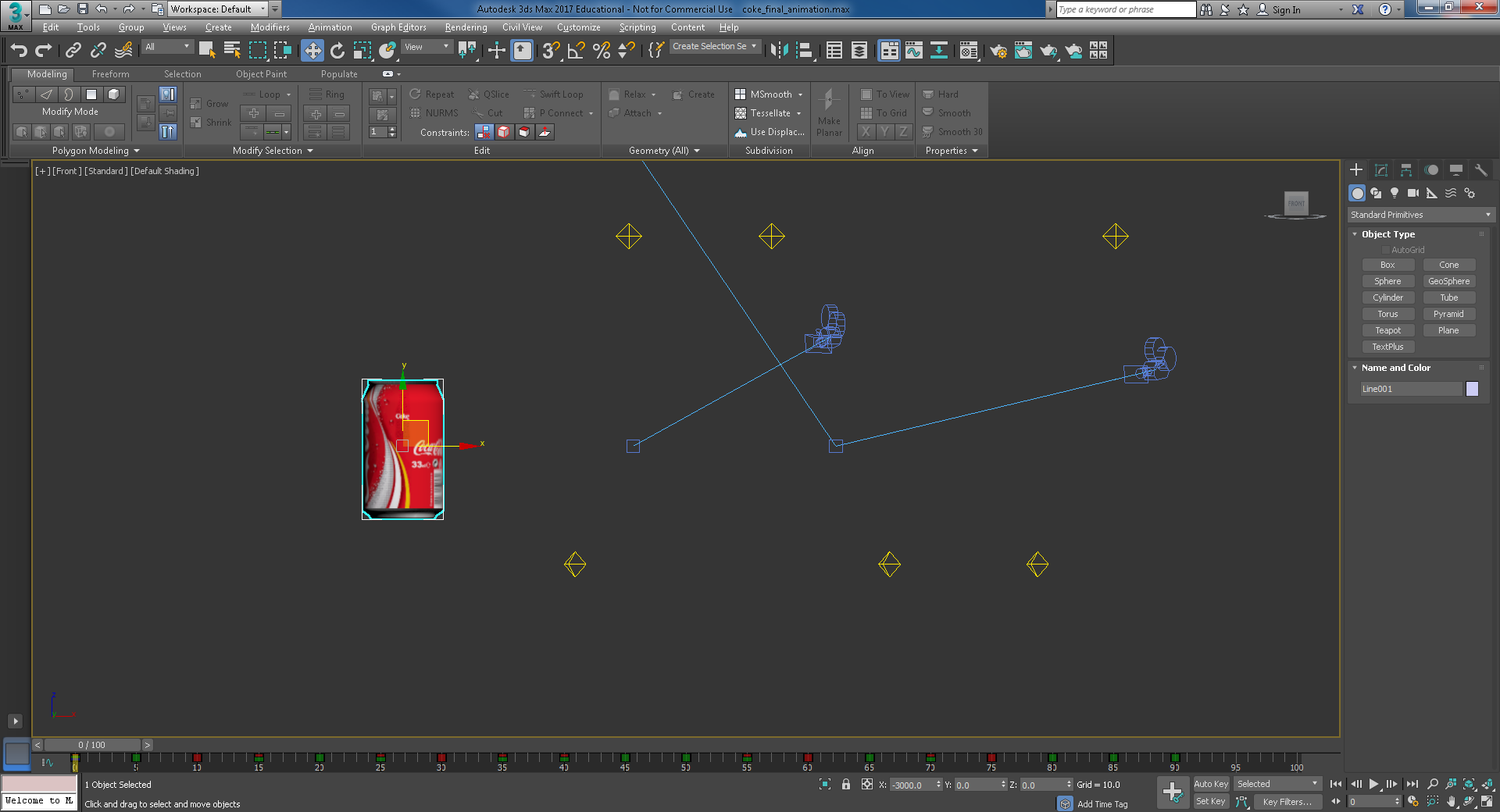


Figure 7: Play the recorded animation

1. You can play around with the animation. For example, I recorded the animation for 90 frames, and did an animation movement every 5 frames. On the animation tool, you can see the little red, green and blue striped squares representing key frames in the animation where you recorded the animation as you made it. You can re-time the animation by moving these key frames along the timeline. Have a go at stretching the animation to say 100 frames, or compressing it to say 35 frames, delete a key frame or two to see what happens. I just stretched out the animation at the start and end, see Figure 8.

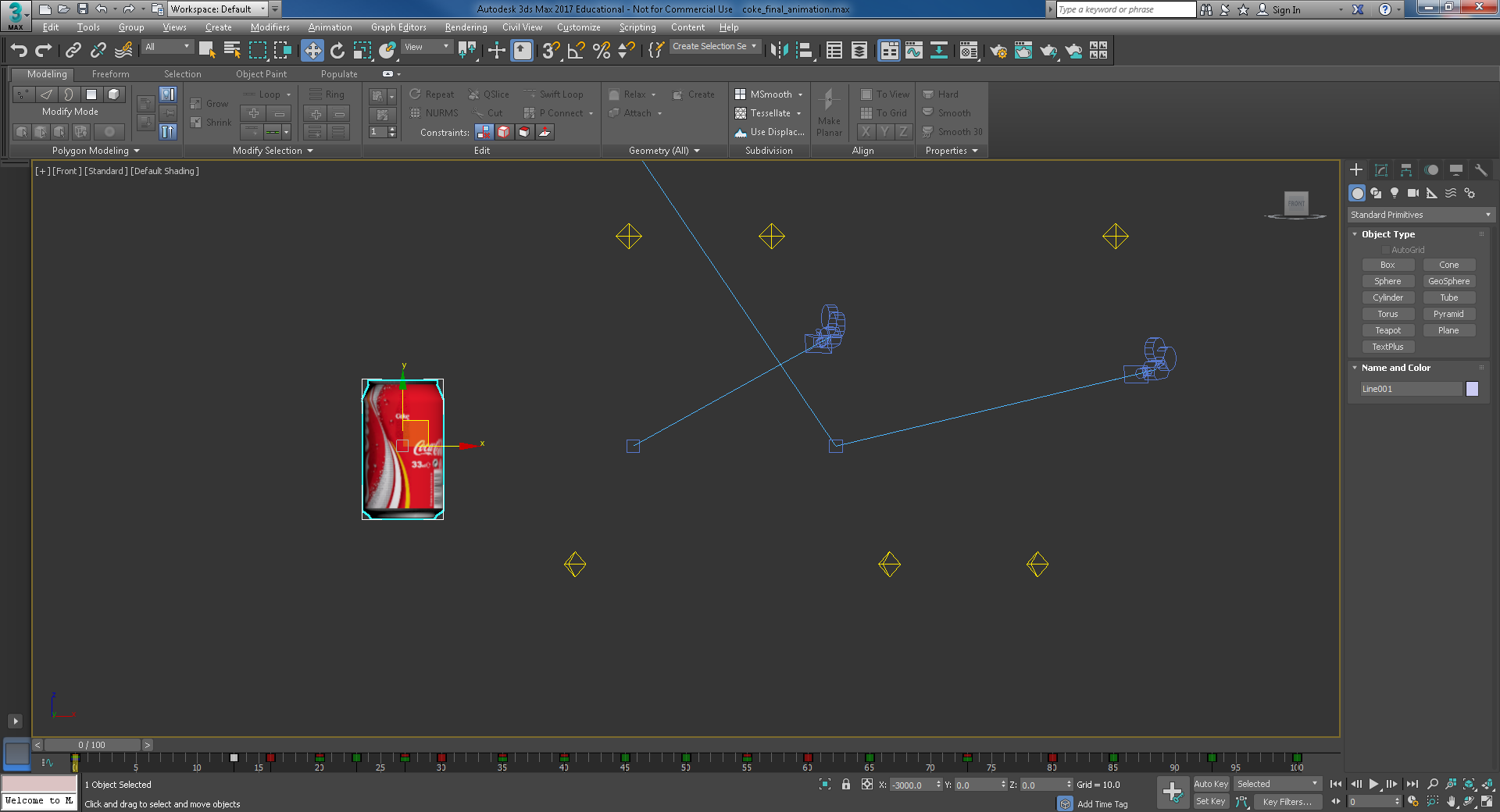


Figure 8: Modified animation

1. You can now have a further play with different animations (it’s worth repeating this exercise to get a feel for setting up amusing animations of the Coke can); you’ll need to delete the old one by selecting the key frames and deleting to start again. Or, save this one and start a new one.   
     
   Tip! When you build your 3D App for the assignment you could consider setting up some simple animations to use as media objects along with photorealistic renderings of your models. You would build these media objects into your overall 3D App architecture.
2. Also, you can play with the animation curves using the Edit Curve editor, to adjust the animation, see Figure 9. Select the Curve Editor and have a play.

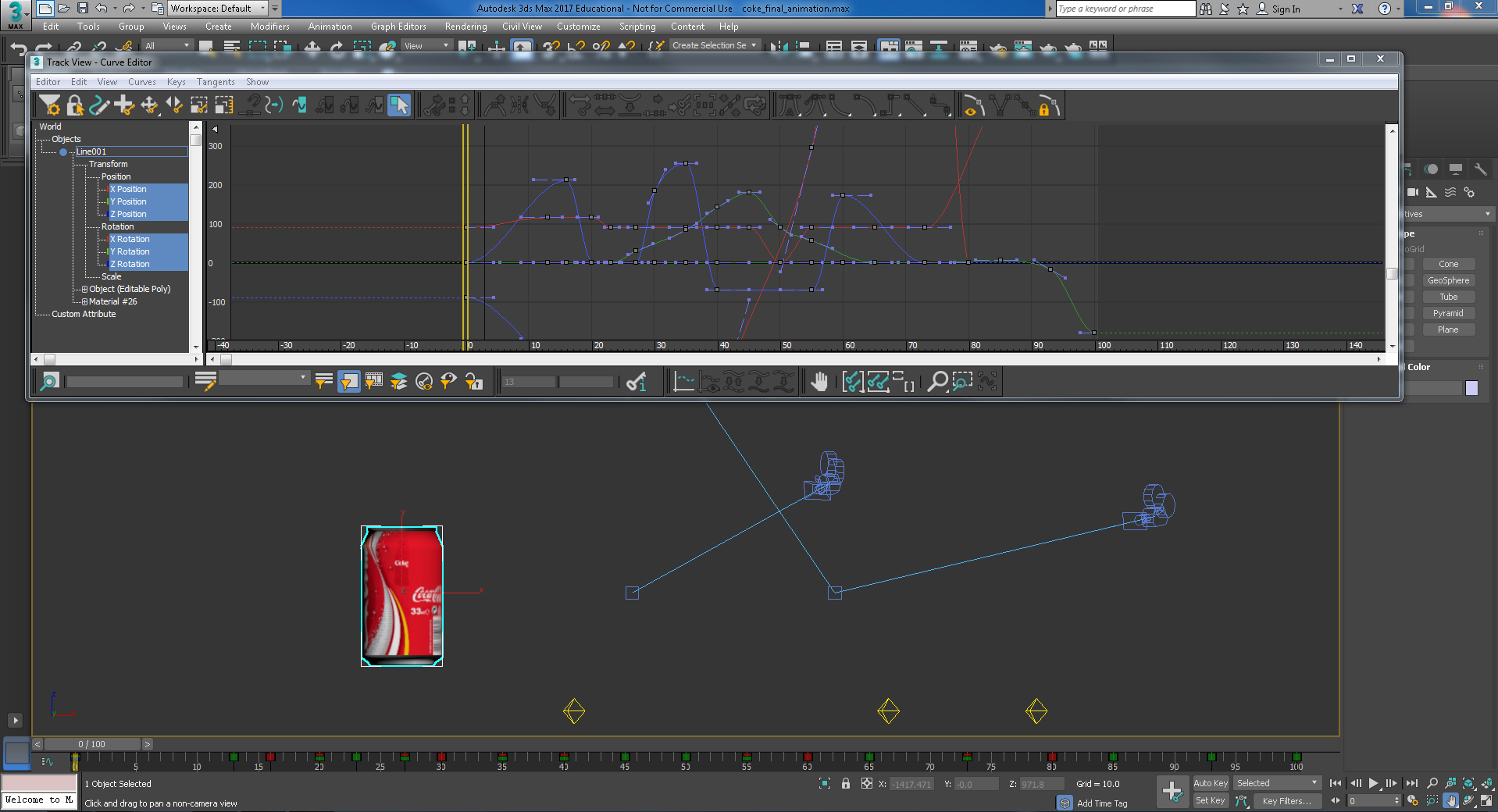


Figure 9: Using the Curve Editor to adjust a key frame animation

1. Looking at the Curve Editor, we can see the position of the object (the Coke can) at the start of the animation shows the cola can is -3000 along x axis, and zero on the z and y axis. Clearly the red curve is the x-axis, so adjusting this curve should modify the x-axis components of your animation. In this particular example, if you follow the red curve you will see at about 50 frames the cola can is back at zero on the x-axis and continues into positive x returning back to zero in x, which is exactly what I animated, see Figure 10. Study the other curves to better understand what is happening, explore to discover and learn more.

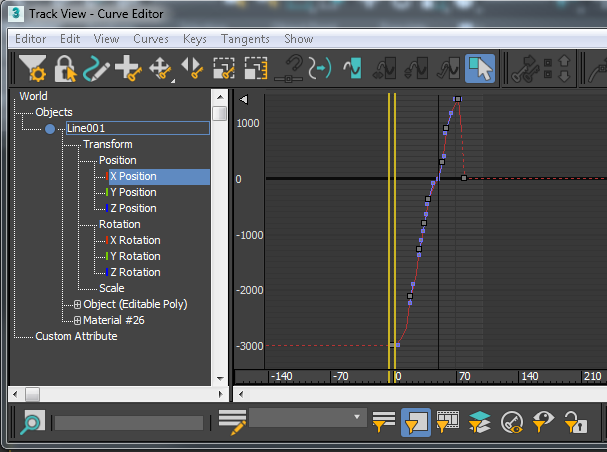


Figure 10: Isolating the X Position animation curve.

If you click on a particular point you can fine tune the animation parameters.

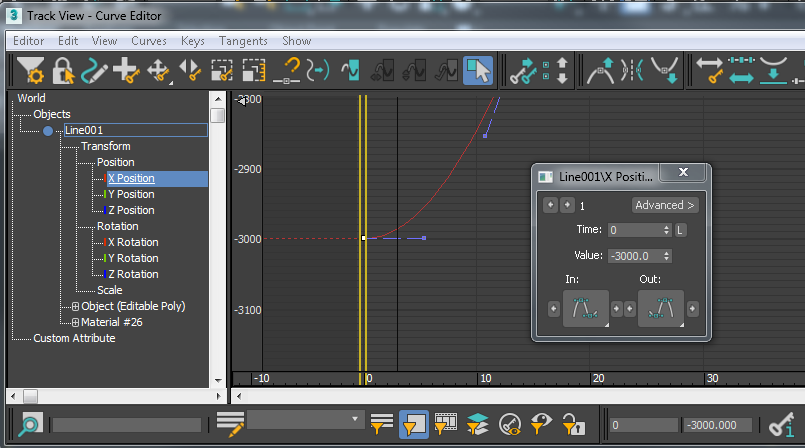


Figure 11: Adjust the animation parameters, such as time and position of the X components

1. Close the Curve Editor if you have been using it. Next, we will attach a touch sensor to trigger this animation.

## Add VRML Helpers to the Coke Scene

Now that you have an animation, you need to use the VRML97 Helpers to create a touch sensor to trigger that animation.

It is interesting at this point to consider the other VRML Helpers, there are 12 you can use, some may be useful, but it depends on your final 3D App functionality:

* Anchor
  + Can click an object and jump to other areas in your scene (other camera viewpoints) or other HTML pages, or VRML worlds
* Background
  + Set Sky Color, Ground Color, and Images rollouts.

This is very useful to match the background colour of your VRML or X3D plug-in with your web site background colour! Alternatively, you can write the VRML code to do this — Google it!

* Fog
  + Can specify the colour and range of fog in your VRML world
* ProxSensor
  + Triggers an animation when in the region of the Proximity Sensor
* TimeSensor
  + Adds time-based animation controls, such as the start and end frames for a particular object's animation, and looping
* AudioClip
  + Specify the name and characteristics of an audio file that can be used by the Sound helper
* Billboard
  + Creates geometry that is camera-aligned in the VRML97 browser, the objects always align to the viewpoint in the VRML browser
* NavInfo
  + Set up how to navigate around the VRML97 world, e.g. walk, fly, and also used to specify whether headlight should be used
* Sound
  + Lets you place 3D (spatial) or ambient sounds in a scene, the sound may be located at a point & emit sound in a spherical or ellipsoid pattern
* Touch Sensor
  + An animation is triggered on touch of an object

Some useful interaction scenarios could be developed using a combination of some of these sensors, for example, a proximity sensor (ProxSensor) could be used to trigger a sound, which uses an AudioClip, along with animation, e.g. spinning the 3D model round. The Anchor could be used to jump to another viewpoint (e.g. between cameras) in the scene or even to an alternative web page outside the VRML scene, for example jump from the Coke 3D model to the Dr Pepper web page.

### VRML Helpers — Touchsensor and NavInfo

Let’s add some VRML Helpers to your Coke can scene:

1. In the right-hand side menu, click on the measuring angle icon labelled “helpers” and from the dropdown box select ‘VRML97’, see Figure 12. A list of primitives appears from which we will be using only ‘NavInfo’ and ‘TouchSensor’.

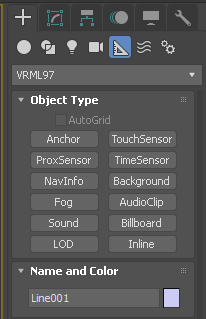


Figure 12: Selecting the VRML Helper: Touch Sensor to trigger the Coke can animation

1. Create a touch sensor and position it next to the Coke can. It is not important where you position it as it only serves as a link between a trigger object (that triggers an animation) and the animated object. It makes it easier to see which sensor is attached to what object if you put them next to that object, see Figure 13.

The touch sensor has two parameters:

* **Trigger Object:** the object that you click on to trigger an animation — this could be the Coke can itself, or another object in the scene.
* **Target Object:** the object that performs its animation when the trigger object is clicked.

To activate the animation, both trigger and target object can also be the same. However, you could model a small object and place it next to the Coke can (for example create a simple sign, ‘Touch to Animate,’ using for example one of the primitive objects) and use that as your trigger object trigger object so that if you click on this object it will trigger the animation. In this case, let’s just use the Coke can itself to trigger its own animation.

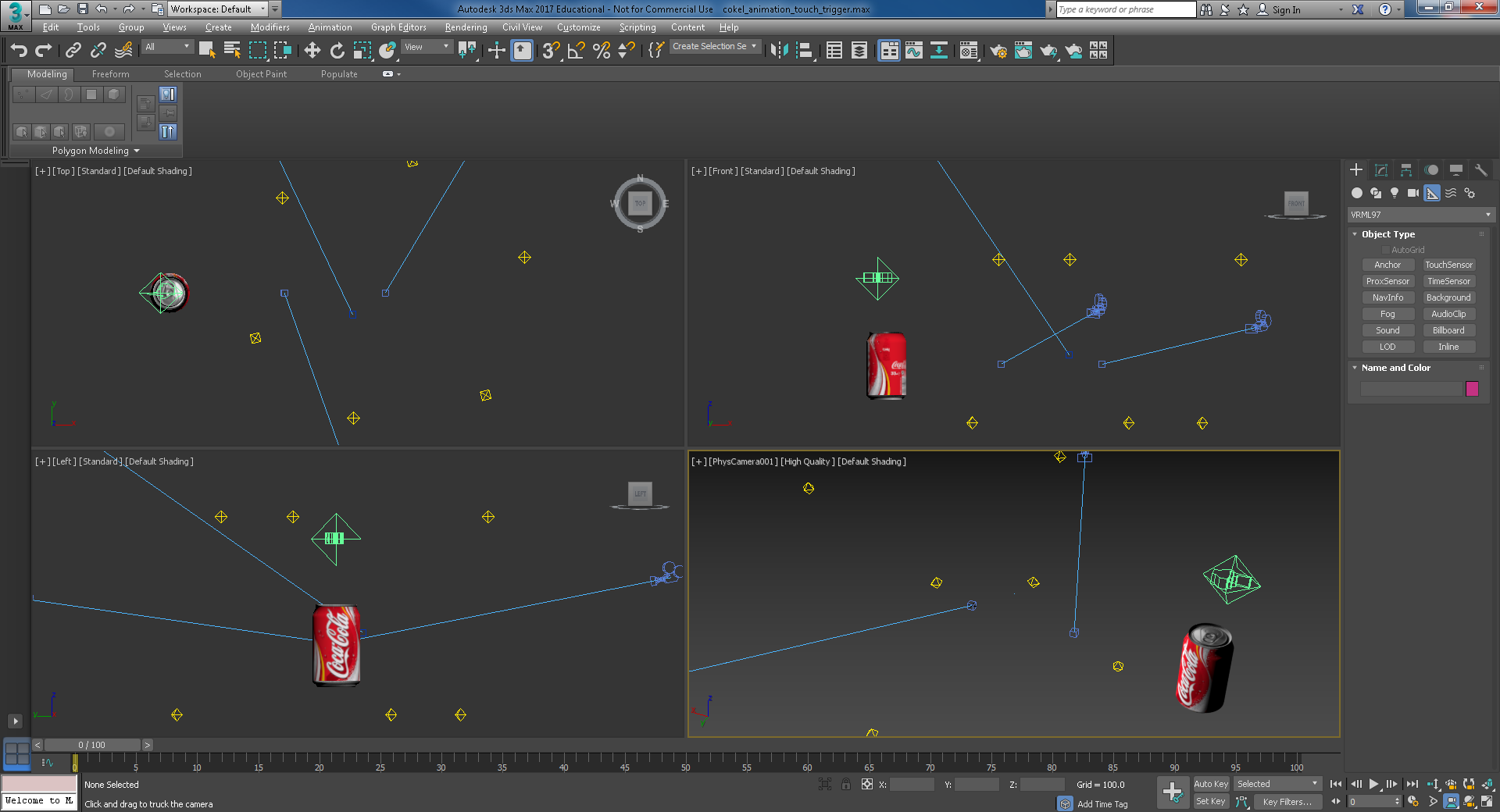


Figure 13: Placing the TouchSensor next to the Coke can, which itself will be used as the trigger object

1. Select the Touch Sensor and go back to the VRML Helper menu, and select TouchSensor. Notice the TouchSensor is called TouchSensor001, you could rename it, say CokeTouchSensor if you want. You now need to select the Pick Trigger Object on the Touch Sensor and then pick the object you are going to use to trigger the animation, i.e. the Coke can itself. Now, you will see the Trigger Object is set to Line001 (or whatever object you selected). Remember, when you created the Coke can, you started with a Line primitive, i.e. the spline, which was named Line001. You might want to rename Line001 to CokeCan, or something similar.
2. Next you need to select the Pick Action Objects on the Touch Sensor, in this case the Coke can again, see Figure 14.

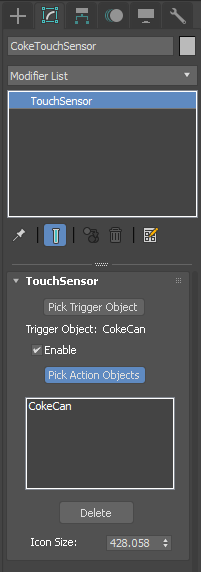


Figure 14: Setting up the Touch Sensor

1. A basic VRML scene also requires a camera and lighting. The camera usually acts as an actor that will project what the person sees when moving through the virtual environment. If you haven’t already added a target camera to your Coke can model, add a simple target camera. In my case, I already have a constellation of Omni lights and 3 cameras (target cameras), which you may need to adjust to see the animation — I already adjusted mine, but I anticipate needing to adjust again when I check the Exported VRML97 scene.
2. In general, a VRML scene also needs a flat plane underneath the VRML model for the actor to walk on, **but do this only if this is what you are designing into your scene**.

For example, if you decide you want to walk around the object(s) like in a virtual gallery, then a floor is needed. **If you are simply viewing a single object it is not needed**.

In the assignment, we will be examining single museum objects, **so we won’t need a floor!**

In week 2 you should already have set up Standard Lights and Cameras in the Coke can model to make the object visible in VRML. Your scene setup should look something like that shown in Figure 15.

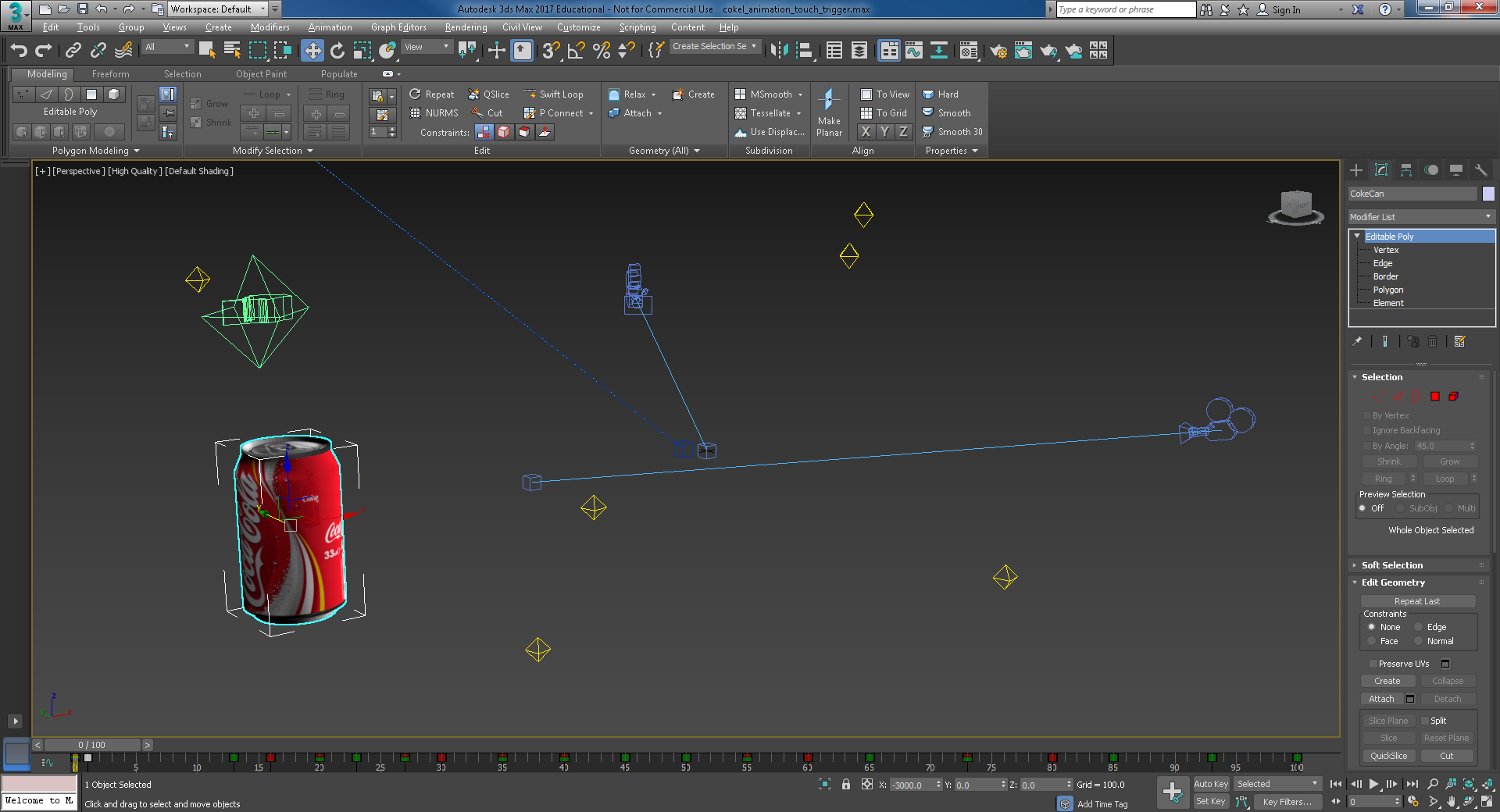
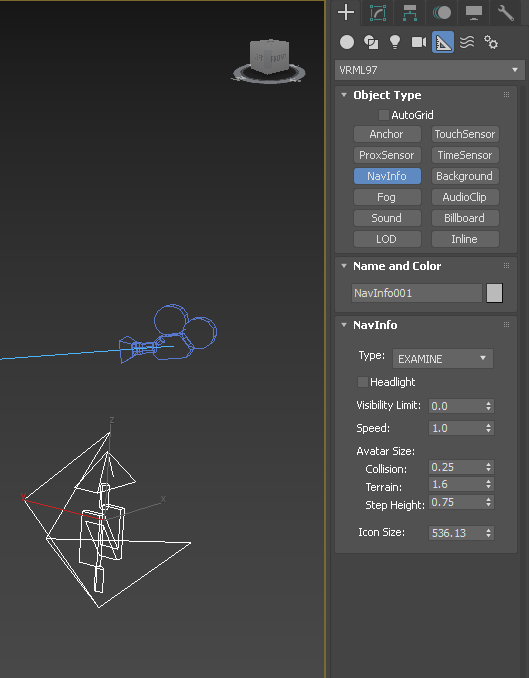


Figure 15: Setting up lights and a target camera

Here you can see that I have the trigger object (the Coke can itself) and the touch sensor located close to the starting position of the Coke can (-3000, 0, 0, where the Coke can will start and finish at the end of my key frame animation, 0,0,0).

1. Finally, you should create a NavInfo VRML object somewhere in your scene. It sets the parameters of your actor as shown in Figure 16.



Here is the NavInfo object, click in the view port and drag to adjust its size. When you examine the exported VRML code, you will see how this object is described. Set the parameters as indicated on the right panel and Type to EXAMINE

Figure 16: Setting up the NavInfo Helper

* **Speed:** sets up how fast you can walk.
* **Collision:** sets up how close you can get to a model before bumping into it.
* **Terrain:** sets up how far the camera is above a surface.
* **Step Height:** determines how tall an object can be for the actor camera to climb on top of it.
  + This is typically used in creating virtual environments for modelling buildings.

## Export to VRML97

Now, it is time to export and test the scene. But, first don’t forget to save your work.

1. Then, go to File > Export and select as a file type VRML97(\*.WRL). Give your VRML exported file a name, e.g. coke\_animation. For now, I am going to save the coke\_animation.WRL file in the 3ds Max export folder because I use 3ds Max project setup. But you can save anywhere so long as you know where to find it, see Figure 17.

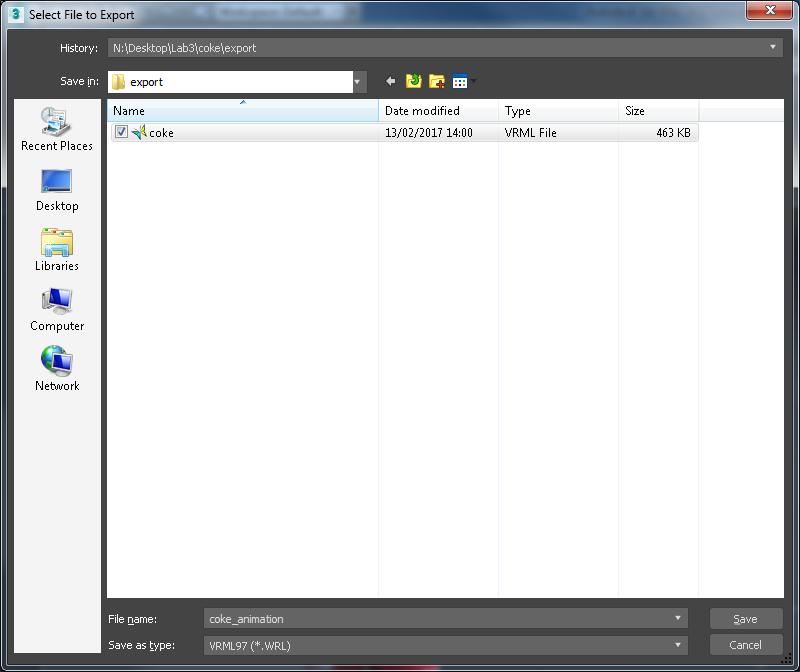


Figure 17: Exporting to VRML

1. Wherever, you save to, the VRML97 Exporter panel comes up next, see Figure 18. Make sure you have the correct objects selected:

* **Polygon Type:** Select Triangles.
* **Initial:** Camera001, note this was selected automatically (I have two cameras setup).
* Initial Navigation: Navinfo001, **which was set to examine**.
* Lastly, make sure you have the correct file path to your texture images. Because I have saved my coke\_animation.WRL file in the export directory I will need to set the path for the Coke can’s texture, which is in sceneassets/images. If you save your coke\_animation in the same directory as your .max file and your textures are also there you would not use this Bitmap URL Prefix, so untick it. See Figure 18.

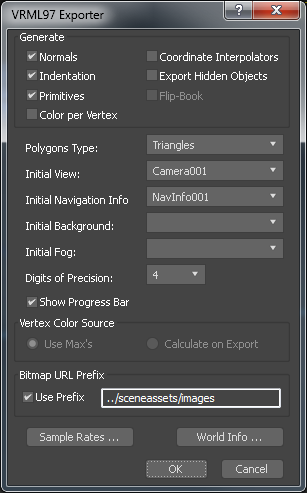


Figure 18: VRML Exporter menu

1. Figure 19 shows the resulting VRML file rendered in the Cortona3D Viewer (embedded in Internet Explorer— you should be able to double click the coke\_animation.WRL file or drag it onto Internet Explorer). Note, this is about the only time I expect you to use Internet Explorer (or Edge). If you are doing this tutorial from home, chances are you haven’t got a VRML viewer installed. In this case you will need to download the Cortona3D Viewer, and the instantreality player too.   
     
   **Note, it only works on Windows!** You may need to enable your browser to run the plugin in too. You should note that Cortona3D Viewer is not supported in Chrome anymore because ‘Google Chrome have ended support for NPAPI plugins’, see <http://support.cortona3d.com/kb/articles/google-chrome-ends-support-npapi-plugins> for more details. **Still, not to worry, use Firefox or Edge for now.**
2. You may need to move the current viewpoint to see your animation (chances are you might not have set up an ideal initial viewpoint with your cameras). What you will find is that you have to play around with the position of your initial camera (Camera001) to be able to see the animation clearly.
3. In the Cortona3D Viewer (drag and drop the VRML file into Edge or Firefox — one of these will support Cortona3D Viewer, if not try the instant**player**), if you select the Coke can it will now trigger the Coke can animation. Check out your animation from all your cameras. In the Cortona3D Viewer you should select Viewports => Cameras to choose a camera. From your cameras, you should see the animation depending on how well you have set everything up.

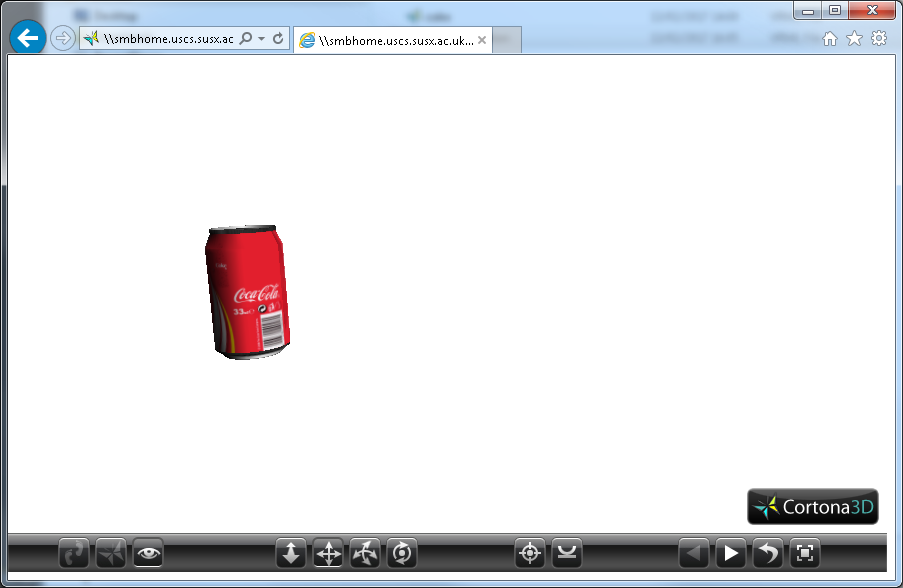


Figure 19: VRML rendered in Cortona3D Viewer in Internet Explorer from camera001, and headlight switched off — you can turn the headlight on in Cortona3D Viewer.

In my case I can only see the full animation from camera001, I can’t see the start point from camera002 unless I increase the size of the Cortona3D Viewer, so I need to go back into 3ds Max and set up cameras better to see the animation from all the cameras correctly. As part of the workflow, you will find that you have to iterate back and forth between testing your VRML scene and 3ds Max to perfect your results before you move onto converting the VRML to more useful X3D and X3DOM formats.

You will, of course, have different animations to mine, different camera angles, etc. because your set up will be different to mine.

**Tip!** You should explore the NavInfo sensor some more. This sensor writes some NavInfo parameters into the VRML file, which overrides the Cortona3D Viewer controls. For example, if the NavInfo sensor was set to WALK so that the other navigation modes are disabled, e.g. EXAMINE, etc. You can easily modify this in the actual VRML code, rather than going back into 3ds Max.

In fact, for your assignment we will only be rendering objects in your 3D App that will require EXAMINE mode. If you set it to Walk, for example, it will stop me from ‘examining’ your 3D objects in scene, making it difficult for me to mark that component!

# Rendering with VRML and X3D

You now have an exported VRML model of your Coke can with some VRML Helpers attached, i.e. a NavInfo, and a Touch Sensor to trigger your animation. Appendix 6.1 discusses some other options for rendering your VRML and X3D models using plug-in renderers, but these are becoming more and more limited. Cortona 3D Viewer, I have found very useful in the past for a quick and easy solution to test your exported VRML97 file in Microsoft Edge.

However, instant**player** is installed in the labs, so I would stick with that.

In Lab 3 we will look more closely at X3D as a Web 3D format or method that is more suitable for viewing your 3D model in a web page, i.e. 3D App. Specifically, we will exploit [X3DOM](http://www.x3dom.org/), CSS3 and JavaScript libraries for embedding X3D code directly into the HTML5 DOM (Document Object Model)

## Web 3D Format Workflow

Now, let’s take a closer look at a workflow for converting your 3D models created in 3ds Max into VRML, then X3D, and then into X3DOM for embedding into an HTML5 document (web page). There are several steps to this workflow illustrated in Figure 20. You will notice that in this workflow I have limited the number of tools used to those that display VRML and X3D. So, for example, instant**player** renders both VRML and X3D, but Cortona3D Viewer only renders VRML.

In this laboratory, you are expected to follow the workflow indicated in Figure 20, which is relatively straightforward, but we describe in detail below.

If you haven’t finished your three models, i.e. the Coke can, Sprite bottle and Dr Pepper cup yet; then you can substitute them for other simple objects from the primitive objects menu in 3ds Max.

Then, when you have the hang of it, you can do the same with your three models created in Labs 2.

Note in Figure 20 we still show the Cortina 3D Viewer as a possible way to test your VRM97 export just in case you do manage to install it — it may also still be installed in the lab machines.



Figure 20: Creating an X3D model for rendering in X3DOM and HTML5

Step 1: Create 3D models with appropriate lights and cameras

* Create your 3D model in 3ds max
  + To test this design workflow, we suggest you create 2 or 3 simple objects, see Figure 21, if your Lab 2 models are not ready yet. Convert them to editable poly.

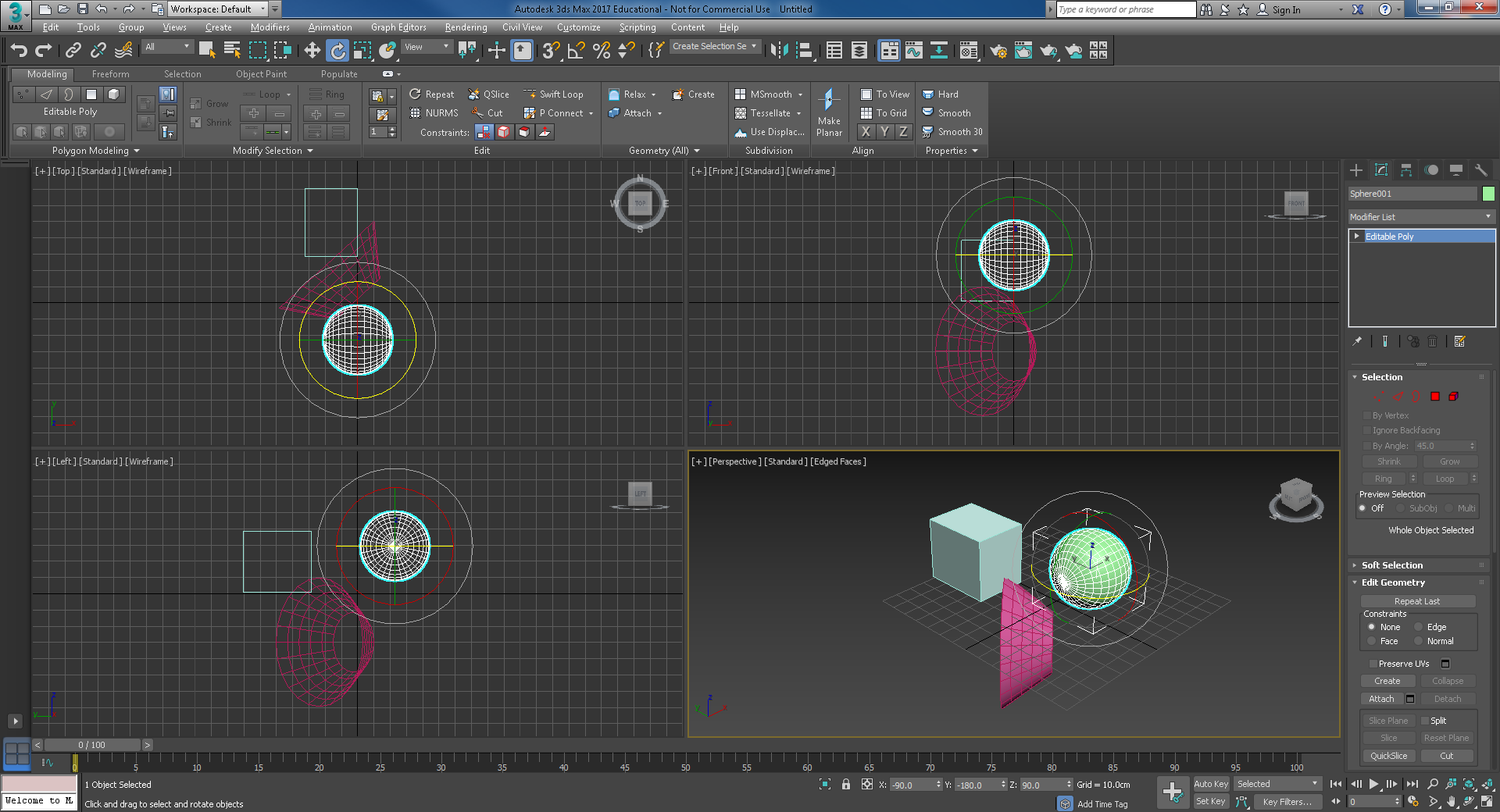


Figure 21: Creating 2 or 3 simple objects

* Add a couple of cameras (try a target and a free) so you can see the basic objects from two different angles
* Then add 2 or 3 omni lights and a spotlight so you can see the objects without the ‘headlight’. After some tweaking of the 2 cameras and 2 omi lights you will have a reasonable test scene, see Figure 22. Note, VRML, hence X3D only supports directional, point (omni) and spotlights.

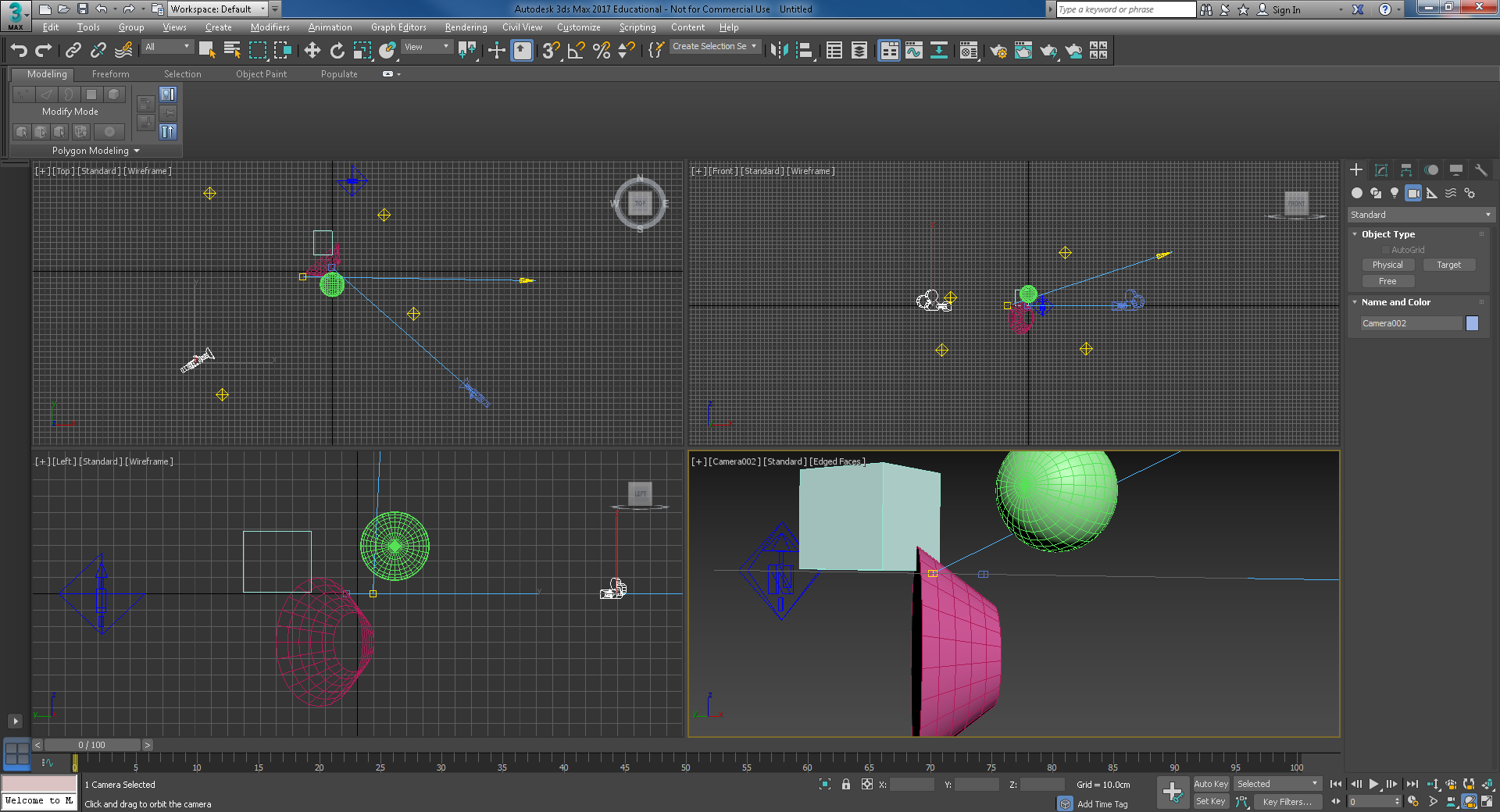


Figure 22: The scene with lights and cameras

* Next, we will export the scene to VRML and you should see something like that shown in Figure 23, you can use instant**player**, or view them directly in Edge or Firefox if Cortona 3D Viewer is installed. There are no texture maps so you can click off the Bitmap URL Prefix. Also, add a NavInfo helper set to headlight on and EXAMINE. You could do this now before we add more interactivity in step 2.

|  |  |
| --- | --- |
|  |  |

Figure 23: The scene with lights and cameras

Step 2: Add VRML helpers for Interactivity

* Add some interactivity to the box primitive
  + For example, in the same way we showed you in the section *Exporting VRML from 3ds Max with VRML Helpers,* where we added an animation to the Coke can that was triggered by a touch sensor, add similar interactivity to the three shapes, but this time use each shape as its own trigger for the Touch Sensor.

Step 3: Export your VRML

* Now export your test 3ds Max model as a VRML model as shown in the section *Exporting VRML from 3ds Max with VRML Helpers.* Test the animation in one of the VRML players, e.g. Cortona3D Viewer or the instant**player**.
* Open up the exported VRML file.
  + What do you see; can you understand it? Take the time to do some research on the VRML scene graph language — you are not expected to write this code, but sometimes it is quicker to modify it, rather than go back into 3ds Max, e.g. when you want to include a headlight that you forgot to set. Or set NavInfo to Examine if you forgot to set this in 3ds Max
  + In the VRML file, you should be able to see (without any knowledge of VRML) definitions for the cameras, light sources and NavInfo. If you don’t set up a NavInfo, it is exported by default, and a default NavInfo will set the Headlight to false. If you have a scene that looks black, chances are you didn’t:
    - a) add any lights or
    - b) didn’t set the headlight to true,
    - or c) both.
  + You can easily check this by setting headlight set to TRUE in the VRML code, try setting that to FALSE/TRUE and save the file, then reload it in the VRML player to see the effect. Also, we have available EXAMINE mode, so we can click in the scene and drag the mouse, which allows us to examine the scene (models) and we always have a good ‘headlight’.

Step 4: Examine your exported VRML code further

* Try modifying the code, save and re-render in the VRML player to see what happens, save a master version so you don’t lose your original, or simply export from 3ds max again.
  + If you are doing this at home, ensure you have an X3D/VRML viewer installed
  + Example bits of VRML code you can easily play around with includes:
    - Set the TouchSensor enable to FALSE — what happens?
    - Set the Headlight to TRUE/FALSE — what happens
    - Play with the camera settings, e.g. fieldofview — what happens?
    - Change the material properties …

Step 5: Convert the VRML to X3D

* While you have the VRML file open (assuming you haven’t corrupted it while experimenting with the code, if so, get a clean version open from the master you saved a few minutes ago).

Using a web browser go to this URL:

<http://doc.instantreality.org/tools/x3d_encoding_converter/>

* + Set choose input coding to classic encoding (VRML97)
  + Cut and paste the VRML code from your VRML file into the ‘paste input code’ box
  + Choose output encoding as ‘XML encoding (X3D) ‘
  + And press the Convert encoding button
  + You should then get the encoded output, cut and paste this into a new file, say test\_scene.x3d using Notepad++ or any other code editor.
  + Either double click on the test\_scene.x3d file, or drag it into an open Instant**player** and you will see it render just like the VRML file.

Step 7: Test your X3D code

* Check that your X3D code runs in an X3D viewer such as Instantplayer; Note, as mentioned above, Cortona3D Viewer does not render X3D code, see Figure 23.
* Just like you did with your VRML code, open up your X3D code and investigate
* Have a play with the X3D syntax, tweak a few things and see what happens, etc. just as you did with the VRML code. You should see that the X3D code works the same as the VRML code, see Figure 23.

|  |  |
| --- | --- |
|  |  |

Figure 23: The scene rendered with ‘headlight’ on as well as the omni and spotlights lights looking from the second camera before triggering — obviously a rubbish initial camera position, the sort of position that would lose you some marks!

Step 8: Convert your X3D or VRML code into X3DOM HTML code

* You can do this either manually by adding appropriate HTML5 to your X3D code file
* Or you can use the Instant**labs** online converter, as above, to generate the X3DOM HTML5 document
* Cut and paste the original VRML code (or the new X3D — this has been known to be a bit buggy in the past) code into the convertor tool and convert to ‘HTML5 encoded web page (x3dom html5)’ code
* Again, open up this document and examine the code so that you understand what is going on
* Experiment with the X3DOM HTML5 code to see how it works.

Step 9: Test X3D in an X3DOM/HTML5 web page environment

* Check that your X3D code runs in an X3DOM compatible browser, e.g. Chrome, Firefox and IE (or Edge), see Figure 24.

|  |  |
| --- | --- |
|  |  |
|  | The X3DOM conversion, i.e. VRML97 converted to an HTML5 encoded webpage (otherwiese know as X3DOM HTML5, or X3DOM for short) using the instantlabs online encoding converter, clearly works in the major web vbrowsers:   * Chrome * IE * Firefox   Note, here we are simply testing the file works, ideally we should be doing this on a web server. If the file had textures, for example the textures would not work. |

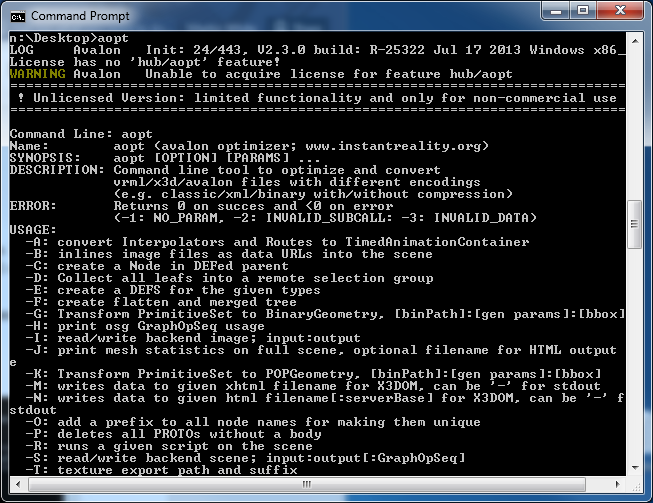
Figure 24: The scene rendered with ‘headlight’ on as well as the omni lights as an X3D model embedded in HTML5

Once you are familiar with this design workflow you can apply it to your 3D model, e.g. the Coke can, Sprite Bottle and Dr Pepper cup. If your 3D model is large and complex (not simple like the Coke can) you should design relatively small chunks of your model and push these through the design flow; incrementally increasing the complexity as you go. You should also check out the tutorials and examples on the [www.x3dom.org](http://www.x3dom.org) web site to see how each example works. This way you can apply this technology to getting your 3D model working in X3DOM and HTML5. Ok, you have now tried out the workflow with some test objects; let’s now convert your actual models.

# Convert VRML97 based 3D model to X3D

In this section, you will convert your Coke can model from VRML, which you exported from 3ds Max, to X3D, X3DOM and HTML5 code using tools provided by Instant**reality** labs. You will use either the:

* Online transcoding too, as used above while you were becoming familiar with the workflow:
  + <http://www.instantreality.org/tools/x3d_encoding_converter/>
* Or the command line tool (aopt — Avalon optimiser), see Figure 25:

1.   
     
    Figure 25: The aopt command line tool for converting VRML to X3D

You will also learn how to embed your X3D Coke can model into an HTML5 web page formatted with an X3DOM CSS and JavaScript libraries.

## Test your VRML first

Before we convert your VRML model to X3D, first check the VRML (.WRL) Coke can file (in my case, I saved this as coke\_animate.WRL, and you should note that 3ds Max uses capital WRL as the file extensions) is a valid VRML scene. In this case, see Figure 26, we have the VRML Coke can model loaded in the Cortona3D Viewer and instant**player**. You should only have to double click on the VRML file to get it to display if Edge or Firefox is your default lab browser, or drag the VRML file onto an open VRML viewer (e.g. Cortona3D Viewer or instant**player**).

|  |  |
| --- | --- |
|  |  |

Figure 26: VRML model of the Coke can rendered in the Cortona3D Viewer and the instant**playe**r

Here, the animation is triggered by clicking on the Coke can — I can’t show you the animation in the document, obviously, but you should check yours is working ok, mine is.

### Convert with the Instantreality Online Transcoder

We will use the online encoder first:

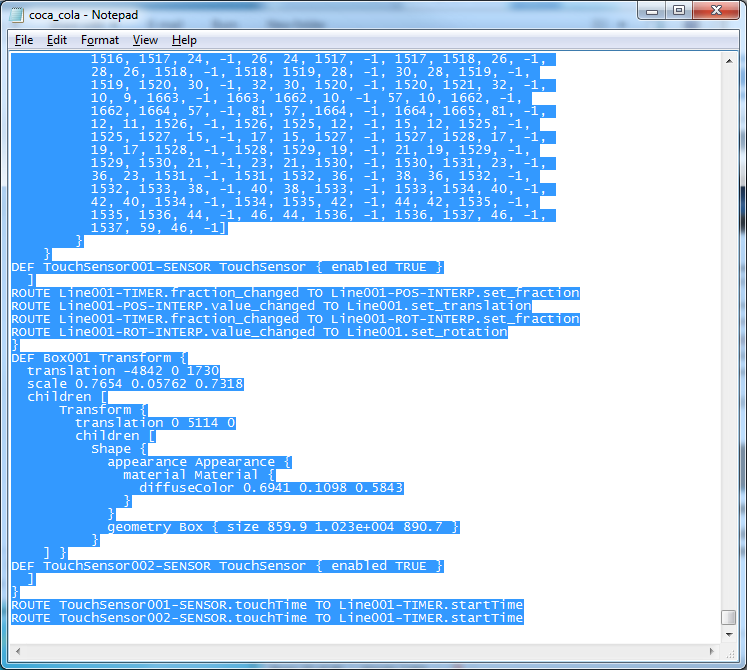
1. So, the next step is to convert the VRML to X3D, etc. We have had a practice above with simple primitives, now let’s do it again with your actual Coke can model. You can go to the Instant**labs** website to access the online tool to convert VRML to X3D, see Figure 27.

* <http://doc.instantreality.org/tools/x3d_encoding_converter/>



Figure 27.: Instantlabs X3d encoding converter

1. Make sure you have ‘Classic encoding (VRML97)’ option selected under ‘**1 | Input Encoding**’.
2. Open your VRML file (coke\_animation.WRL in my case) into a text editor such as: Notepad++, and copy the whole code and paste it into the text box ‘**2 | Paste Input Code**’ in the X3D encoding converter. See Figure 28. If the VRML code is very large, it may take a while to appear in the input box.



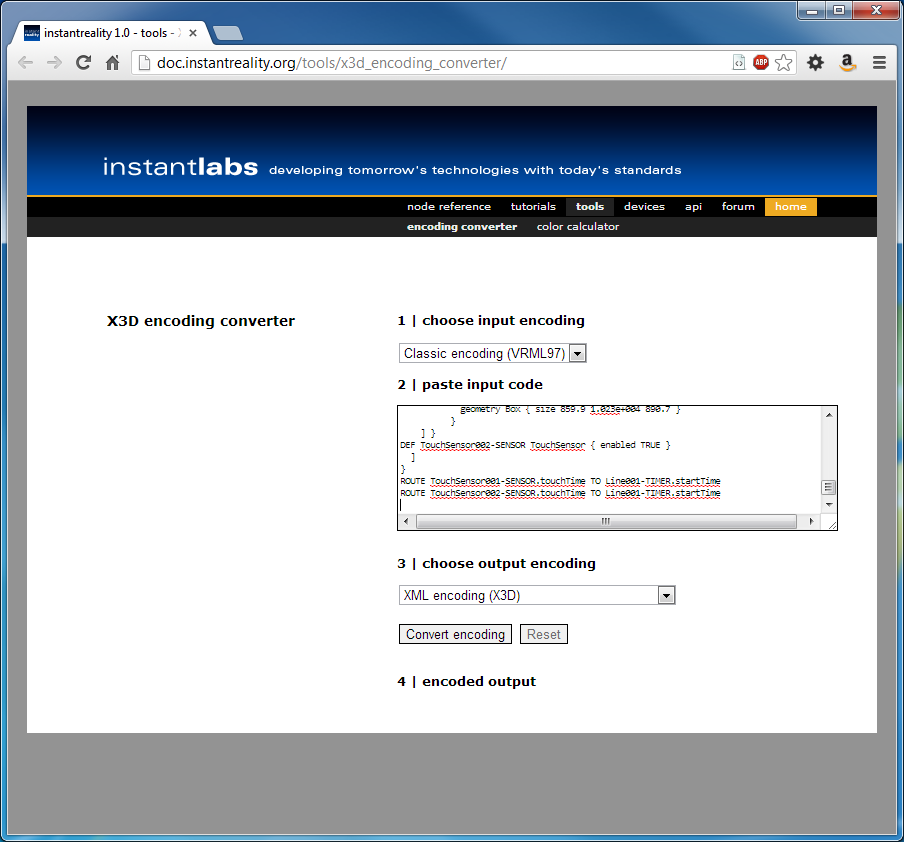


Figure 28: Pasting your VRML code into the Instantlab X3D encoding converter

1. Select ‘XML encoding (X3D)’ from drop down box under ‘**3 | choose output encoding**’.
2. Click the ‘**Convert encoding’** button
   * This should take few seconds to few minutes, depending on the size of VRML input file to convert into X3D code. See, the converted X3D code in Figure 29.

Tip! For small chunks of code this won’t take long to convert, but for large chunks it may take a while. For large models, you may want to organize your VRML as separate components in 3ds Max, exporting the separate models, with VRML helpers, and convert this to X3D and so on. Then you would connect all your X3D components together into one X3D file. However, you won’t, or shouldn’t need to do this for your assignment.

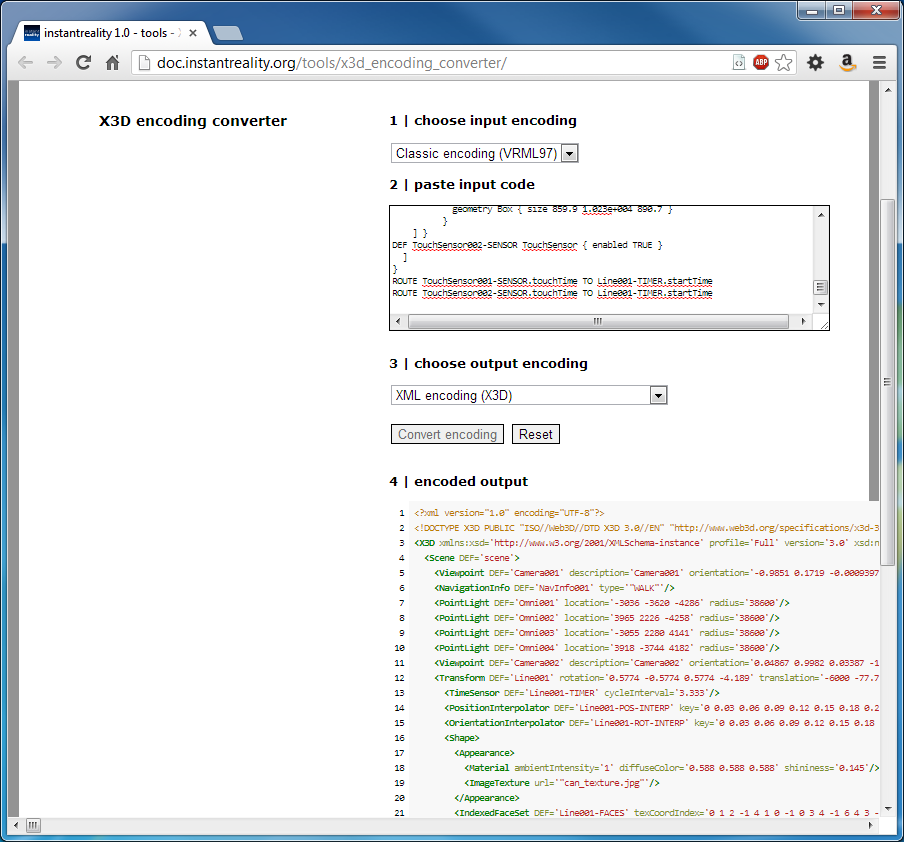


Figure 29: The resulting encoded output, in this case X3D

1. Using your mouse, select the newly created X3D code starting below the label ‘**4 | encoded output’**, right click and copy the code.
2. Open any ascii text editor such as Notepad++.
3. Right click and past the copied code inside the text editor, see Figure 30.

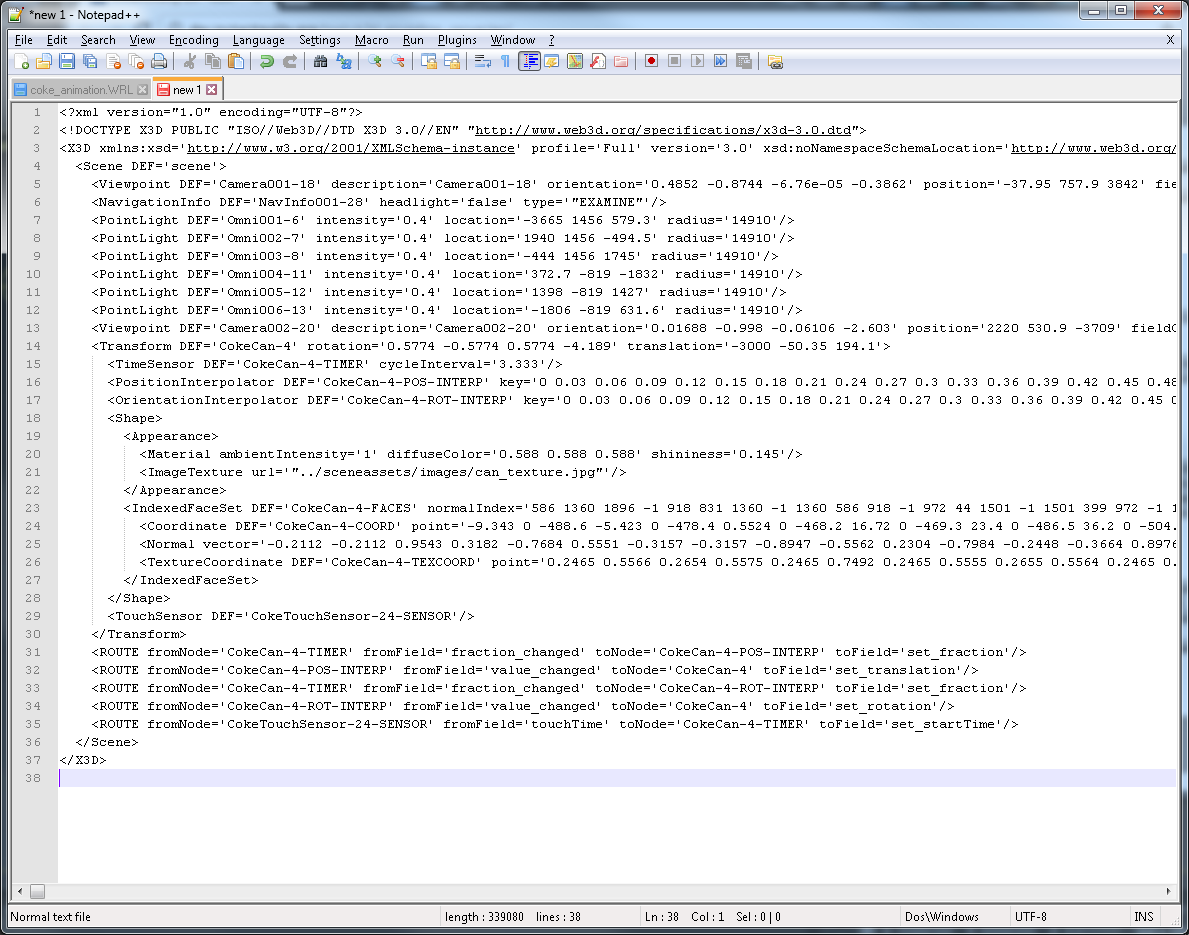


Figure 30: Your X3D code pasted into a text editor, in this case Notepad++

Tip! Note the large set of coordinates preceeded by the tag ‘<IndexedFaceSet coordIndex= …”; your VRML and X3D files will have a lot of these index face sets, don’t fiddle with these coordinates, these are generated by your modelling software (3ds Max). Unfortunately, if your model is very complex these make the file quite large, but still readable.

We will show you how to manage your X3D files and render them inline later, this will make your HTML with embeded X3D much more readable, **which is essential for your assignment**.

1. Save the file with a name of your choice and with extension ‘.x3d’, such as ‘coke\_animation.x3d’as in my case.
2. Open the newly created x3d file (in my case coke\_animation.x3d) into the Instant**player**, or any other X3D viewer you have installed — X3D-Edit installed in the labs has a built in viewer too, see Figure 30.
   * Note that instant**player**, can render X3D models. But, Cortona3D Viewer can not render X3D!
   * Also, you might want to edit the X3D file to set headlight = true  
     <NavigationInfo DEF='NavInfo001' headlight='true' type='"EXAMINE"'/>

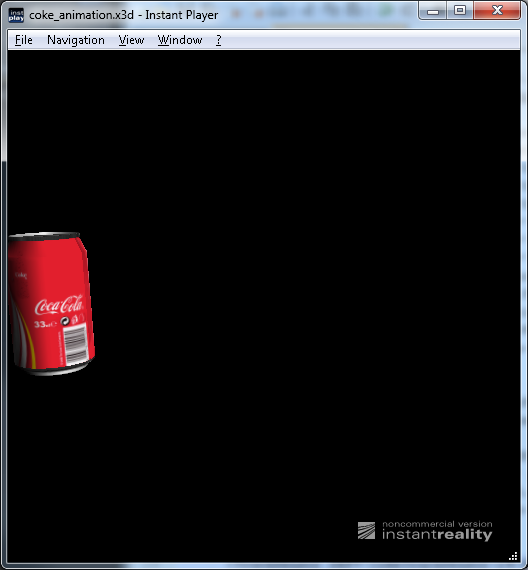


Figure 31: Your Coke X3D model rendered in instant**playe**r — note headlight is not turned on, but we have plenty of other lights

By now you should have your X3D model being rendered inside instant**player**. It looks identical to the VRML version, but this is X3D format. Take the time to check out the code in Figure 30.

Now let’s try the aopt command line tool.

### Using AOPT to Create Optimized X3DOM Content

Check out the method: <http://doc.x3dom.org/tutorials/models/aopt/index.html>

Note that the package should already be installed in the labs, but you might want to install on your home machine. You will need to navigate to the directory where you have your VRML file stored (or know the path to it). So, a very basic procedure, for example, would be to convert a file that InstantReality can open (e.g. in obj, ply or wrl format) into an X3D file. In my case, I am currently using my Desktop where I have a copy of Lab 3. So, an example aopt command to convert your VRML file to X3D would be:

* aopt –i coke\_animation.wrl –x coke\_animation2.x3d  
    
  See Figure 32 for an example

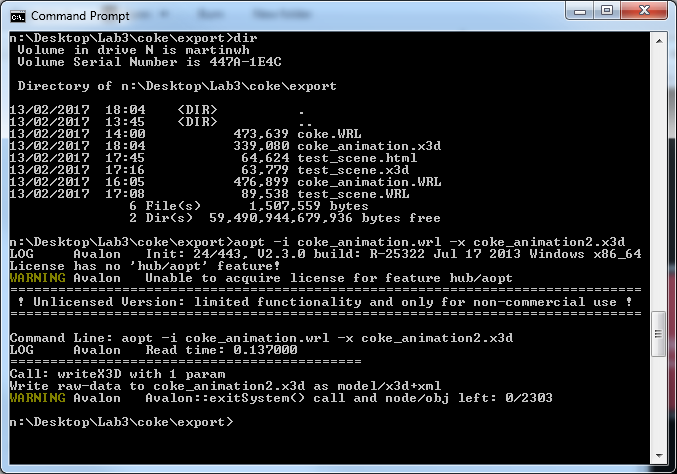


Figure 32: Using the Instantreality command line aopt tool

As you can see, I have used the CMD prompt to navigate to my export folder in my 3ds Max project. I will then run the aopt command from there — you will need to change from the C: drive default, maybe, on the CMD prompt to your your home drive is, e.g. N: drive, i.e. C: \Users\martinwh>n:

This will return the same result as shown in Figure 30 above.

You can also convert your VRML file into an X3DOM HTML page.

## Convert VRML97 direct to X3DOM for rendering in HTML5

1. Either Repeat steps 1 – 4 in the last section to copy your VRML file in ‘**2 | Paste Input Code**’
   * Your VRML code should already be pasted into the paste input code box, but you might lose it if you reset!
   * This time choose ‘HTML5 encoded webpage (x3dom html5)’ under ‘**3 | choose output encoding**’ option
   * Click ‘Convert encoding’
   * Copy paste the newly created code into a text editor as before
   * With a text editor save the file with the extension ‘.html’ (e.g. ‘coca\_cola \_animation.html’).
2. Or, use the aopt command line too, like you just did above:
   * aopt –i coke\_animation.wrl –N coke\_animation2.html

See Figure 33 for an example

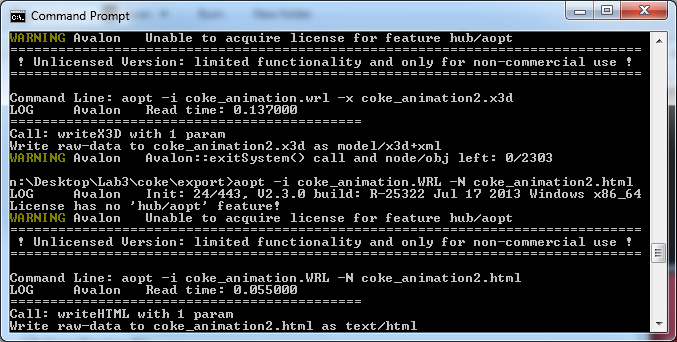


Figure 33: using aopt to create a web page with your 3D model

1. It is important now to use a WebGL compliant browser, see [www.x3dom.org](http://www.x3dom.org), to be able to view the X3D file embedded in the HTML5 code’s DOM (Document Object Model). While several browsers are now compatible, in this case, use Google Chrome or Firefox (I tend to try and stick with Chrome) to view your coke\_animation2.html’ file because these browsers support WebGL natively.

WebGL is the rendering engine used by X3DOM (X3D in the DOM) to render the 3D model, see Figure 34. Here you can see on the left that the texture is not displayed on the Coke can. This is because most web browsers have introduced security measures to prevent the injection of code from an external source, such as an external texture delivered from a content server not in the same domain, unless that external source is trusted. There are several ways to ensure your texture is displayed properly;

* + Setup a trusted web server environment so that the browser trusts the source (web server) from where the texture is served, etc. This is the ideal solution, but a bit over the top for development purposes
  + For development, you could just turn off security on the web browser, perhaps not a good idea!
  + Or better still, test your X3DOM and HTML5 in a local web development environment such as WampServer, XAMPP, MAMP or the built in Apache server on a Mac.
  + In your case, if you are saving your files to /public\_html/… or some such directory it should work ok. For now, I’ll just upload my Lab 3 development folder to my /public\_html/ directory to test it, as I am in the lab. If you are at home, you will need to use a localhost or an SFTP client to upload to your /public\_html/ space.

|  |
| --- |
|  |
|  |

Figure 34: Coke can exported as VRML, and embedded as an X3D object rendered with X3DOM and HTML5

Figure 34, bottom image shows the textured Coke can rendered ok with the HTML file (X3DOM version) being served from the University web server.

Previously, the X3DOM version failed to trigger the Coke can animation. While this still works in an X3D player such as the Instant**player,** it didn’t work in the X3DOM conversion.

This is because the TouchSensor (or more specifically the X3D pointing sensor component) was not supported by X3DOM. Instead, you needed to use JavaScript and an HTML onclick() method to trigger any animations that would have been triggered by the touchsensor, in your 3D App. We still need to use the <timesensor> node, which will have been set up by your 3ds max animation.

However, as of 2018, the x3dom.js library now seems to support the touchsensr, which is useful. Nevertheless, we still need to adopt the onclick() method for triggering animations. This is because, it is very useful to connect the start and stop of an animation to an HTML button in the web page.

We provide an example of how to trigger using onclick. Further, if we wanted to do a lot of manipulation of the DOM elements (including X3DOM) we would most likely use JQuery’s CSS class selector.

We will show you how to do this in Week 6 labs. However, if you want to look ahead you can use Chrome’s Inspect Element to see how we trigger an animation using JavaScript in the following examples:

[Live Feedback Site](http://users.sussex.ac.uk/~martinwh/3dapp/3dapp/labs/)

Then navigate to Lab 6 Results, Virtual Museum. Use the Chrome Dev Tools > Inspect to look for the animation triggering code.

In this virtual museum example, the JavaScript code for triggering the animation that you are looking for is something like:

//Javascript function to rotate (animate) the cultural object's 3D media object function

rotateObject(){

if(document.getElementById('bowl\_\_GrpBowlTIMER').getAttribute('enabled')!= 'true')

document.getElementById('bowl\_\_GrpBowlTIMER').setAttribute('enabled', 'true');

else

document.getElementById('bowl\_\_GrpBowlTIMER').setAttribute('enabled', 'false');

}

This code simply sets the <timesensor> loop value to true if not true, and then enables the animation on click and stops on click by setting the enable value, see example:

<TimeSensor DEF="GrpBowlTimer" loop="true" cycleInterval="6" enabled="false"/>

So that we don’t get cluttered up with animations for now, before the next section, let’s get rid animation and the touch sensor. Either prepare a new 3ds max scene by deleting the TouchSensor, adjust the cameras and lights again to get nice views if needed, export to VRML, and convert to X3D, then use the new coke\_no\_animation.x3d file for the next section. Alternatively, you should already have you’re a Coke can 3ds max file saved somewhere without the animation, see Figure 35.

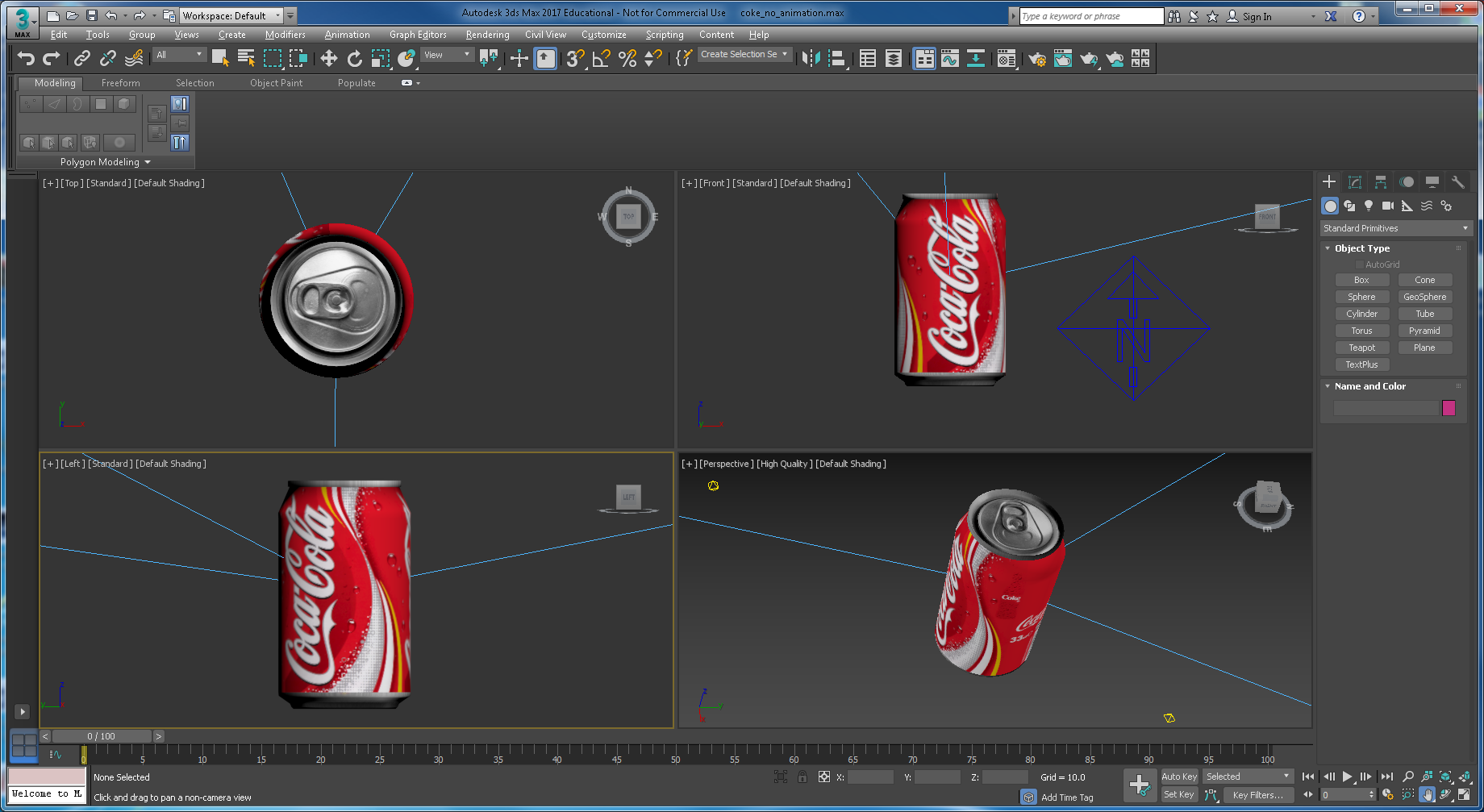


Figure 35: Coke can with cameras and omni lights and no animation

So, export this scene (without any animation) to VRML and create the X3D file as shown above, don’t forget to add a NavInfo with headlight on and EXAMINE set. I’ve called my VRML file coke\_no\_animation.WRL, call yours what you want.

If for some reason you forgot to save a file without animation, you can open your X3D file delete the animation code by deleting the key frames.

## Convert X3D to X3DOM for rendering in HTML5

We can of course convert the X3D code directly for rendering using the X3DOM libraries and HTML5.

Create the X3D file using either the online transcoder or the aopt tool the convertor and test it, my X3D file is called coke \_no\_animation.x3d, see Figure 36.

aopt –i coke\_no\_animation.wrl –x coke\_not\_animation.x3d

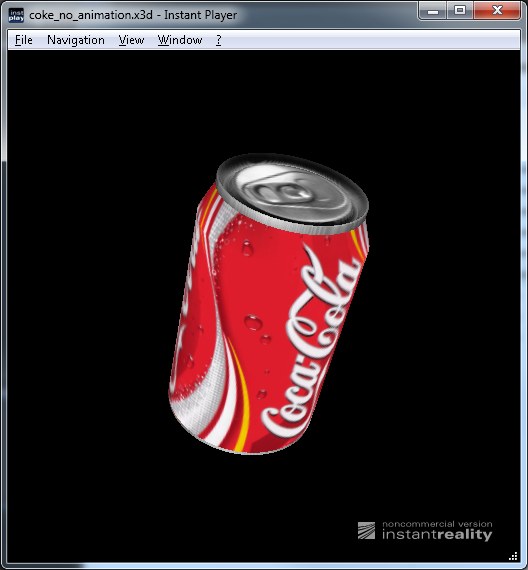


Figure 34: X3D file coke\_no\_animation.x3d with cameras, and omni lights

1. Again, use either, use aopt command line tool or the online transcoder to convert your X3D file into an HTML encoded web page:

Using the online encoder

* + Open your X3D file in a text editor such as: Notepad, copy the whole code and paste it into the text box under ‘**2 | Paste Input Code**’ in X3D converter. Don’t forget to set the **1 | choose input coding** to XML encoding (X3D)
  + Choose ‘HTML5 encoded webpage (X3dom html5)’ under ‘**3 | choose output encoding**’ option
  + Click ‘Convert encoding’
  + Copy paste the newly created HTML5 and X3D code into a text editor
  + With the text editor save the file but extension ‘.html’ (in my case ‘coke\_no\_animation.html’), and as before view in Chrome or Firefox, see Figure 35, and don’t forget to publish it in a localhost environment or your public\_html directory

Alternatively, use the aopt –i coke\_no\_animation.x3d –N coke\_no\_animation.html

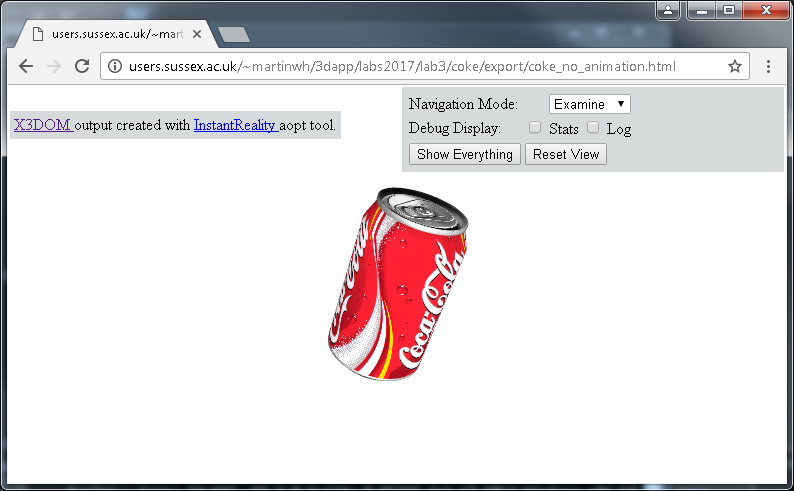


Figure 35: The Coke can (from X3D) embedded as an X3D object in X3DOM and HTML5

By now you should be able to see the same model in the browser again, but this time converted from X3D unlike last time where it was converted from VRML. So you now have the basis for inserting or enbedding 3D into a web page, i.e. the makings of a Web3D App.

Let’s summarize what we know so far:

1. The exported VRML works (e.g. we can render scenes in VRML, and the animation is triggered properly by a TouchSensor, etc.) in the instant**player** (and will work in Cortona3D, but only VRML)
2. We know that VRML converted to X3D works in the Instant**player**
3. And, we can embed X3D code directly into a web page (HTML5) and see the 3D render
4. Although we haven’t done it yet, we can also embed a VRML (Cortona3D Viewer) or X3D player (BSContact) directly in the web page, we’ll cover that below for interest’s sake, but we won’t use it for the assignments, because as you will see in your final assignment this is not an optimal solution.

We now need to look more closely at how we can embed our VRML or X3D models into the HTML5 web page more elegantly. That, we will do next week in lab 4.

But, some hints below.

# Integrating VRML, X3D and X3DOM models into a 3D App

We need to create a simple 3D App template to display our X3D models. This 3D App template doesn’t have to be anything complicated at the moment. You could simply adapt your Lab 1 template using your Bootstrap 4.0 code.

## Write X3D into the HTML5 DOM

To do this I suggest you take the following steps.

**Step 1**: Copy your Lab 1 site to your Lab 4 directory so that you have a good working copy of your Lab 1 site. Make sure this is live on /public\_html/

**Step 2**: Make a second copy of the Lab 1 template and rename it coke.html

**Step 3**: With your coke.html file, strip out the main text, and the three images and associated contents. This should leave you with a header, a main\_3d\_image, and a footer

**Step 4**: Make 2 more copies of the Coke.html file and call them sprite.html and pepper.html. You will now use the main\_3D\_image div tag to encapsulate your three X3D models.

**Step 5**: Link up your navigation menu. If you have used the Bootstrap version for Lab 1, you can ignore the About and Contact, but link the Home on each of the Coke.html, sprite.html and pepper.html to the index.html. Then, on the index.html link the dropdown menu for drinks to the appropriate Coke.htm, sprite.html and pepper.html, repeating this for each of the drinks html files so that the 4 pages are all linked up. If you have completed the Dreamweaver CS6 version of Lab 1, you will need to add a Home link in the navigation menu, and link up the navigation menu in a similar way. You should now have a simple 4-page site.

**Step 6**: On each of the Coke.html, sprite.html and pepper.html pages we now need to insert your inline X3D code. If you haven’t got inline X3D versions of your 3D models yet, go back to the online conversion tools and create these. Make sure you select the XML encoding (X3D) for the output encoding. You should have three X3D models, e.g. coke.x3d, sprite.x3d, pepper.x3d. The code you need to insert these three models is very simple:

<div id="main\_3d\_image">

<x3d id=”model3D”>

<scene>

<inline url="coke.x3d"> </inline>

</scene>

</x3d>

</div>

But, you will need to also include the X3DOM libraries, etc. in your HTML5 should download the x3dom.css flie and installed it in your css folder or use the linked release version of the x3dom.js file. However, sometimes, the X3DOM server goes down, so consider [downloading the complete X3DOM package](https://x3dom.org/download/1.3/docs/singlehtml/) and installing it locally into your project so you don’t get caught out later!

<link rel='stylesheet' type='text/css' href='css/x3dom.css'></link>

<script type='text/javascript' src='<http://www.x3dom.org/x3dom/release/x3dom.js>'></script>

**Step 7**: Test your X3D models in your very simple 3D App, in Lab 4 we will refine this 3D App some more. You should have a result looking something like that shown ion Figure 36 Dreamweaver version, and Figure 37 Bootstrap version.

## Embedding a VRML/X3D plugin into the web page

This section is largely historical!

Finally, let’s contrast this method of writing our X3D model directly into the HTML5 DOM with the method of embedding a plugin to view the X3D. You can do this by swapping out the X3D inline code for embedding an X3D player, e.g. BSContact or if you can’t download BSContact try embedding Cortona3D Viewer and use your coke.wrl file instead of the X3D.

As implied, this method is largely becoming obsolete, for example Chrome no longer support such plugins embedded in the web page. But, for interest’s sake here are two choices for embedding a plug such as Cortona3D Viewer, BSContact: 1) the Embed tag, or 2) the Object tag. The code is:

<div id="main\_3d\_image">

<div id="model3d">

<Embed src="Coke.wrl" />

</div>

</div>

But, to be clear, you will not use plugins for your assignment, instead you will focus on embedding X3D in HTML (so called X3DOM).

Figure 38 Lab 1 3D App home page connected to the three 3D models (Coke can VRML plugin, Sprite bottle X3D plugin and Dr Pepper cup as X3DOM — 2015 version

In Lab 4 and 5 we will take a closer look at responsive and mobile first design and X3DOM and bring together your results from weeks 1 to 5 into a more viable responsive simple 3D App, which will start to make the beginnings of your assignment 3D App framework.

# Appendix

## Rendering your 3D models in a Plug-in 3D Viewer

You now have an exported VRML model of your Coke can with some VRML Helpers attached, i.e. a NavInfo, and a Touch Sensor to trigger your animation. You can, of course, embed this VRML model (via a VRML player, such as Cortona3D Viewer or a BSContact Viewer) into your 3D App to render the scene, but you won’t use it in your assignment because it is too limiting. For more details on the Cortona3D Viewer check out:

* <http://www.cortona3d.com/cortona3dviewer>

You will, as already mentioned, notice from the link above that the Cortona3D Viewer works only on PCs (no longer supported on Macs), and it doesn’t work in Chrome anymore, not since [Chrome ended support for NPAPI (Netscape Plugin Application Programming Interface) plugins](http://support.cortona3d.com/kb/articles/google-chrome-ends-support-npapi-plugins). That sort of suggests the ‘games up’ for plugin-based 3D viewers, and frankly I am surprised Cortona3D Viewer is still around! But, it has been a good 3D viewer and is still supported in other browsers, etc. So, you can still visualize your exported VRML97 model in Firefox, and Edge on the Lab PCs. For what it is worth, Cortona3D Viewer is apparently available is for iOS:

* <https://www.youtube.com/watch?v=o5FIViNhPJ8>

But, these days the Cortona3D Viewer tends to have a niche market, with several limiting factors, not least that it doesn’t support X3D, you also need to program with its own SDK, and it is not open source. So, we will only use it for testing VRML exported from 3ds Max before we convert the VRML to a better more open source format.

Once you have exported your VRML model you can convert it into an X3D model and use one of the X3D players available embedded into your 3D App; BS Contact, for example, renders both VRML and X3D.

* <http://www.bitmanagement.com/products/interactive-3d-clients/bs-contact>

It is not clear if bs-contact can be easily downloaded these days, but you are welcome to try.

Alternatively, you could port your 3D model into a game engine’s web player, such as the Unity Web Player, but these methods still suffer from plugin issues — they are generally not open source and limited to using some kind of programming interface that is not well supported.