

Creating 3D Educational games

With Director, Flash, and Lightwave

Overview

This document summarizes and extends upon the session *Behind the Scenes on "Science Pirates"* presented at the ACE/NETC 2007 Conference by C.C. Chamberlin.

Inside, you will find a description of the software we use to create 3D educational games, the technical workflow involved, and resources to help you get started on your own 3D educational game projects.

If you have further questions about this document, need designers, programmers, or artists to consult or assist with you on a project, or simply want to show us the fantastic 3D educational games you've developed, please contact C.C. at cc@nmsu.edu.

Software Overview

Here is a list of the software we primarily use for development, along with a brief description of what each one does and why we chose it:



NewTek Lightwave

3D modeling and animation package. Used to create the 3D worlds and the objects and characters that inhabit them (with the exception of items that are “billboards” - see below). Textures are applied to the 3D models, and then it is exported to Director.



Adobe Flash

2D drawing and animation package. Used to create the textures for all 3D objects and interface elements. Because we wanted a “cartoony” feel to the graphics for the game, we chose Flash, since it truly lends itself to this look and its files are natively supported by Director.



Adobe Director

Multimedia scripting package. Used to present and interact with the 3D world. This is what the game is “programmed” in. It supports both 2D and 3D animation and interaction formats, and has a rapid development timeline compared to other packages such as Torque game engine, but pays for it in some flexibility.

All told, the three packages above will run you under \$1000 (as of mid-2007, prices from www.creation-engine.com, taking advantage of educational institution pricing; corporate and personal will run you more).

Choosing Alternate Software

The above trio of solutions work well together, but they are by no means the only available solution for creating 3D educational games. You should consider your team’s size, strengths, and weaknesses before settling on a final solution.

3D Software

For 3D software, there are a lot of solutions available. Some big contenders are 3D Studio Max, Maya, and Cinema4D. While 3D Studio Max is probably the market leader, it was price prohibitive for us, and also suffered from not being cross-platform; having a large amount of Macs in our production pipeline, we wanted a solution that worked on Mac. Maya and Cinema4D are both cross-platform and should work fine. If your target development environment is Director, there are differences in the way they export the .w3d files (the 3d model information format used by Director), which may have to be worked around. For instance, Maya includes a lot of cruft in their export that needs to be stripped out programmatically for best performance (although it shouldn’t affect small projects) and Cinema4D has their z-scales flipped because that software uses a different “handedness” to the coordinate system. All of these can be worked around and there is ample information on the web to deal with them.

Texturing Software

For texturing models, as mentioned above, Flash was our package of choice to give a cartoony look. Cartoony characters and textures are a simple matter to achieve with this package, thanks to its elegant stroke smoothing and stroke manipulation tools. However, if you prefer something more photorealistic - such as 3D visualizations of real objects for teaching - then you may want to instead use something like Photoshop or Gimp which can deal elegantly with photographic images. (And even then, Flash is still excellent for making the interface elements, so you may want to invest in both.)

The other benefit of using Flash is that it doubles as a 2D animation package; it's becoming something of a standard everywhere but the high-end animation studios. In *Science Pirates*, we have a series of music videos that explain certain important topics, such as what Agar plates are and how they work. Since Director natively plays back Flash content, we were able to easily integrate these music videos with the gameplay, and have the added benefit of being able to play them back on the web if desired, or even export the animations to video.

Scripting Software

For programming, there are many options. We chose Director for its ability to easily and quickly integrate a wide variety of media types (2D, 3D, audio, text, etc.) and script them in response to player input. It's a rapid application development (RAD) environment that - once you know what you're doing - allows you to very quickly make prototype games and deliver the final product. Director naturally delivers a cross-platform product (Windows and MacOS), and can be written to play back both as a standalone application or within a web page.

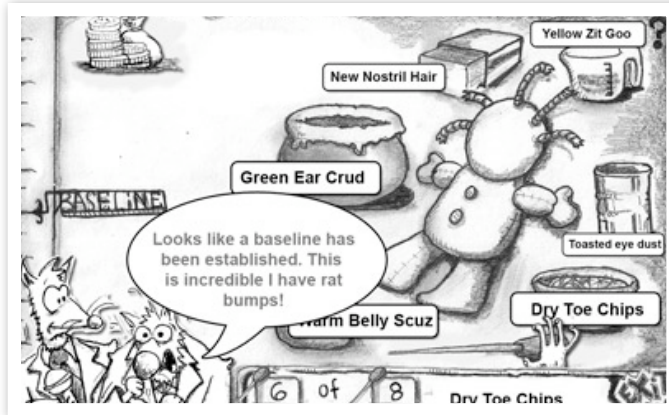
However, Director pays for this speed by being less flexible and fast compared to something written from scratch in something like C++. However, the cost and time to develop full-blown C++ applications (especially to make them cross-platform!) is quite high, and is typically not feasible for small development shops unless they are tightly focused on application development. There are ways to mitigate that cost, such as by using third-party libraries such as Torque or Ogre3D, but we have found that if you can fit your vision into a RAD tool, your required budget gets a lot smaller.

(As a side note, Director does allow the addition of custom C++ libraries, so you can leverage Director's strengths and only code the parts you need in C++ if Director's out-of-the-box capabilities are not enough.)

There are some other RAD tools out there:

- **Unity** (unity3d.com) is an aggressively developed 3D-only environment (it doesn't do the 2D things Director can do) which produces cross-platform and web-deliverable content, but its development environment is (currently) Mac-only. (A Windows IDE is in the works.)
- **Blender3D** (blender.org) is an end-to-end open source 3D development framework, which is truly cross platform... and free. It allows you to model, texture, and script all in one package.

There are basically four steps in the workflow of creating educational 3D game content: design, modeling, texturing, and scripting.



Designing your 3D game

It is important to develop your game design before plunging in and making all of your game assets. You should, before making a game, tinker with your tools to learn what they are capable of, and then design your game from top to bottom before beginning production in earnest. Design should be at least half of your production budget, and the more stakeholders you have to please with your design, the longer you should spend!

Part of the design process is producing a design document that contains the complete description of the game and its content. This makes the following steps much easier, faster, and cheaper.

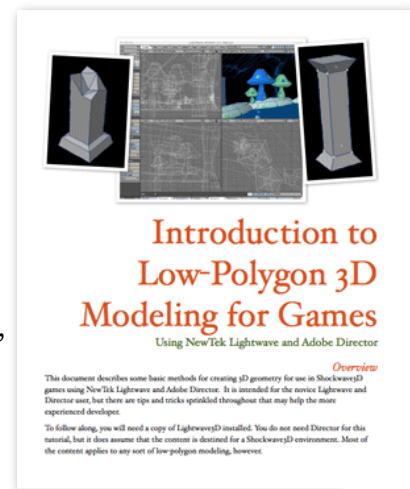
Modeling 3D Content

The 3D content for a game begins its life inside a 3D modeling package like Lightwave. A 3D artist creates geometry for the required assets as individual objects. Anything that needs to move independently of other things in the 3D world should be a separate 3D object.

Because the images need to be rendered in real-time, many times a second, the geometry has to be simple. The more complex a model, the longer it takes to compute its image, so making simpler models will give you the freedom to do more with the 3D engine. This is where having a design document comes in handy – if the modeler knows, for instance, that the back of the building will never be seen, he doesn't have to spend time modeling it, and it doesn't need to be in the model, which makes the 3D engine have to deal with it every frame.

Low-polygon 3D modeling is an art form, and takes practice to get right, but making smart design decisions about what your visual style is going to look like will help. Because smooth, curved surfaces require many more polygons than boxy, straight surfaces, you will want to eliminate them wherever possible, only using them when they can add visual impact or are required for the purposes of the game.

I've created an in-depth tutorial on modeling 3D content as a companion to this session. See the end of this document for information on downloading the tutorial.



Texturing 3D Content

Once the content is modeled, it needs to be textured. Often, the same person will be doing both tasks, but they are two separate skill sets, and you may find that it's better to separate the roles between two people with particular aptitudes in each.

One consideration for real-time 3D is the fact that there is a limited amount of memory on the 3D card for storing textures, so efficient use of textures is important. Reuse textures where possible, and choose appropriate texture sizes for the manner in which the objects will be displayed onscreen. Again, having a design document assists with this, because a texture artist cannot choose the appropriate size for a texture until he or she knows what size the object will be viewed at onscreen. If a painting on a wall is only seen from far away, for instance, the texture can be small, but if the player can zoom in and look at it closely, the texture must be large.

There is also an in-depth tutorial on Texturing 3D content as a companion to this session. See the end of this document for information on downloading the tutorial.

Scripting 3D Content

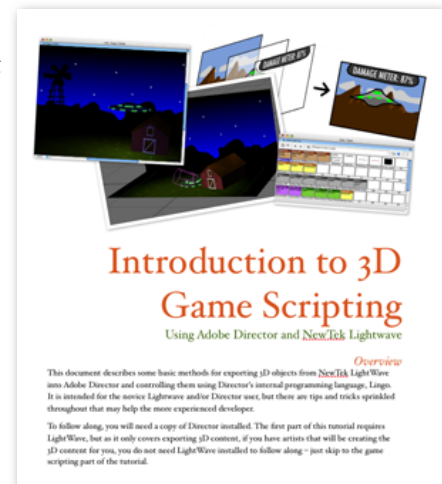
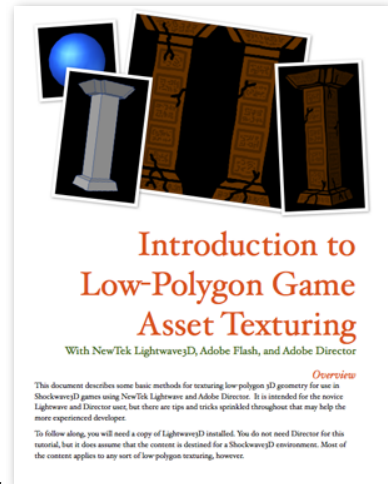
Once the models are modeled and textured, they may be brought into Director and scripted.

Essentially, this is just the process of telling the 3D models and the virtual camera to move around, rotate, and turn on and off based on user input.

Having a design document here is just as important as for the other areas, since the architecture built for the game needs to match the project's requirements. It's easy to build a project which ends up not meeting the requirements added at the last minute, so the more the programmer knows about the final requirements for the game, the better.

Of the three steps, game scripting is probably the part that requires the steepest learning curve to get started. However, anyone with any programming experience at all (especially in Javascript) should be able to make headway quickly, because Director's programming language is relatively straightforward.

As with the other two steps, there is also an in-depth tutorial on Game Scripting using Director available as a companion to this session. This tutorial includes some downloadable assets used to follow along.



Downloading the Tutorials

This document, and the companion documents mentioned above, are available for download at the Shockwave 3D Developer's Guide at:

<<http://homepage.mac.com/nephilim/sw3ddev/>>

In addition to this session's companion documents, there are several other articles there designed to assist in the process of creating 3D games in Shockwave3D, and a framework for helping build larger Shockwave3D projects.

