

Billiard game

This project is to show my technical fluency of several game programming techniques. This document will explain how I have designed the billiard game and what techniques I have used.

A video demonstration is available on my blog:

<http://wrice.blogspot.com/2009/09/billiard-game.html>

Used techniques

- a. PhysX
- b. Collada
- c. DirectX 9
- d. HLSL
- e. OpenAL
- f. MicroSoft C++/CLI Unit testing tool
- g. C++ std::tr1 library

Screen shots

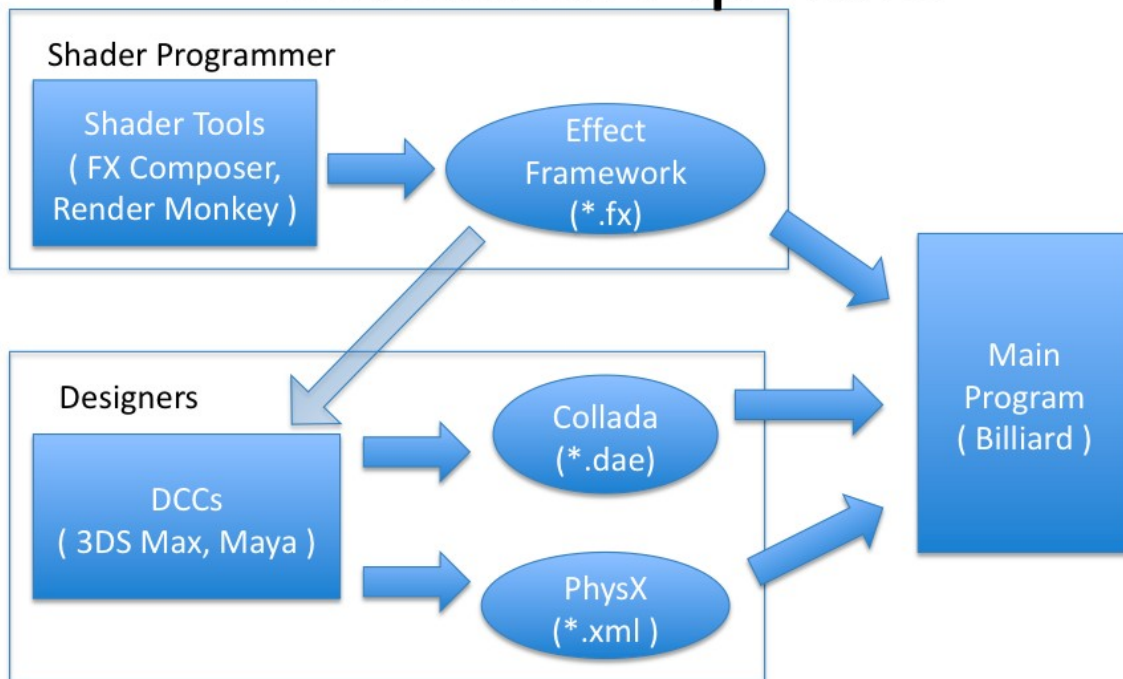


Contents Pipeline

Contents pipeline is pipeline that game contents pass through. Under a team development environment, several different teams need to cooperate. For example, one game model content will be generated by a designer then it will be passed to a programmer and the programmer will load the data to show visual result on game application and finally the game application with the game model content will be shown to game level designer. The process can be considered as a pipeline process.

Therefore, under a team development environment, it is important to build a game application to allow integrate team activity. Collada has emphasized on the contents pipeline, but I found that it doesn't fit to DirectX game development process, so that I have made a few modification. This project is to show how to integrate such team environment.

Contents Pipeline



As the diagram above shows, a game development team can play each different role. In the process, each role needs to have ways to communicate. I have adopted three standard ways: **Collada** for model data, **PhysX** for physics geometry, and **Effect Framework** for shader programs.

Model data

I have chosen to use Collada XML standard for model data. Collada is well documented game industry standard XML format. There are already well implemented plug-ins for certain DCCs such as 3DS Max and Maya.

Note that currently I haven't seen any Collada implementation for DirectX in C++. Most of them are implemented based on OpenGL or XNA.

Physics

I have chosen nVidia's PhysX for my implementation, because it has been broadly used for commercial games. PhysX offers 3DS Max and Maya plug-ins. It generates sufficient data for physic geometry information. Thus in a actual team environment programmers doesn't need to synchronize the gap between physics geometry and visual geometry.

Vertex/Pixel Shaders

The project heavily relies on Effect framework. One of the reasons is that there are shader programming tools for effect framework. One of them, NVidia's FX Composer, didn't work on my hardware so that I have used RenderMonkey and exported my shader programs as Effect file (*.fx).

The effect files are used for the game application. This effect file can also be imported onto 3DS Max.

General Structure

As many game engines supports multi-platforms, the design of my program also takes care of platform independency by separating interfaces from implementation. Although the concept of modern game engine has been enlarged a lot, my design approach can be considered as a small game engine. The interfaces are also categorized by three different abstract levels: Abstract engine level, OS dependent level, and Graphics API dependent level.



The architecture can hide specific platform or graphics APIs from game application developers as long as the game application uses only abstract engine interfaces. The actual game application will use only “Abstract engine” part, then it is supposed to be running on other platforms and other graphics API.

Since this project is to demonstrate my technical fluency, I wished to implement for every components. However, time was limited and I had to end up with implementing DirectX9 part only, so that I have finished “DirectX9”, “Windows”, “Abstract engine” parts and one billiard game application.

Shader Programming

This project is also intended to show my shader programming skill. Unfortunately my hardware had a certain limit to use complex shader programs, so that I have implemented only three shader effects.

- a. Phong textured shader.
- b. Shadow map shader.
- c. Depth culling shader.

Each balls in the billiard game has its own phong textured shader. They are projected on to shadow map from the perspective of a light. Then the shadow map is used to draw shadows on the pool table. In addition depth culling shader is used to cull geometries before actual colors are decided. It is used to draw every object from the camera perspective onto depth buffer and the depth buffer is reused to cull actual pixels.

Source code

The source code is on Google Code, so that anybody can access to it.

<http://code.google.com/p/mybilliard01/>

Requirements

These software must be installed to run the program correctly.

1. **PhysX**: http://developer.nvidia.com/object/physx_downloads.html
2. **OpenAL** : <http://connect.creativelabs.com/openal/Downloads/Forms/AllItems.aspx>
3. **DirectX 9c** : <http://www.microsoft.com/downloads/details.aspx?FamilyID=2DA43D38-DB71-4C1B-BC6A-9B6652CD92A3&displaylang=en>
4. Visual Studio 2008 Feature Pack for **TR1 library**.