

UNIVERSITY OF ABERTAY DUNDEE

Programming Games for the PC and the Xbox

Assignment report

Pedro E Melendez – 0800123

5/15/2009

DESCRIPTION

This report describes the process of the development of a limited soccer game. The game consists in a character controllable by the player and an AI controlled Goalkeeper who chases the ball in an attempt to stop a score from the player. The figure bellow shows a screenshot of the game.



Figure 1 Screenshot of the game

The player controls are:

W	Move the player up.
S	Move the player down.
A	Move the player toward left.
D	Move the player toward right.
Mouse – Left button	Kick the ball.

VERTEX SHADER

A vertex shader is used to define a cartoon shade effect. That effect was chosen to give a humour look to the game. The vertex shader works alike a diffuse light which takes in consideration the cosine of the angle between the light and each vertex. A difference of a diffuse light a gradient is avoided. The goal is to have an abrupt change between dark and bright regions. To avoid the gradient the cosine of the angle between the vertex and the light is mapped to a texture. This texture only contains three colours which determine how bright a zone is. The texture used to map the bright regions is shown below.



Figure 2. Texture used by the vertex shader

This effect was applied to the player, the goalkeeper and the ball. An example about how looks this effect is shown below.

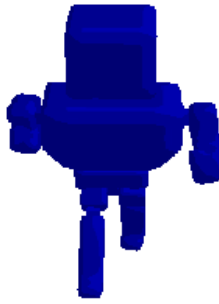


Figure 3 Toon shader applied to a character

GENERAL STRUCTURE

The program is divided in 16 header files and 16 code files.

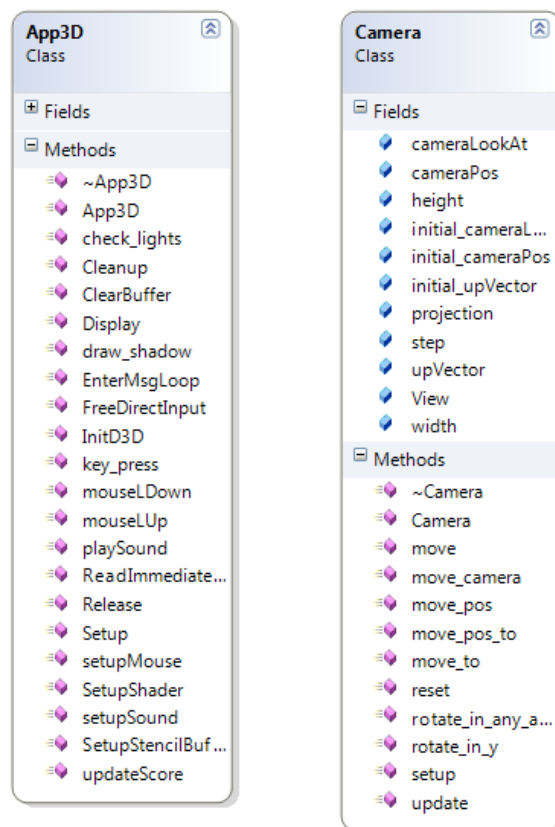
Source File	Header File	Description
App3D.cpp	App3D.h	General code of the program
Camera.cpp	Camera.h	Class that handles the camera.
D3dUtility.cpp	D3dUtility.h	Frank Luna's utility code.
Main.cpp	Stdafx.h	Main entry.
Mesh.cpp	Mesh.h	Class that handles the mesh.
Player.cpp	Player.h	Class that manage the playable character.
Net.cpp	Net.h	Class that handles the net logic.
SoccerBall.cpp	SoccerBall.h	Class that handles the ball logic.
GoalKeeper.cpp	GoalKeeper.h	Class that handles the Goal keeper logic.
CollisionMesh.cpp	CollisionMesh.h	Class that contains information regarding collisions.
CollisionSphere.cpp	CollisionSphere.h	Implements a collision mesh with a sphere shape.
CollisionWall.cpp	CollisionWall.h	Implements a collision mesh with a wall shape.
Vector3D.cpp	Vector3D.h	Generic implementation of a 3D vector.
SDKsound.cpp	SDKsound.h	Class that implements an audio player.
SDKwavefile.cpp	SDKwavefile.h	Class that open .wav files.
Hierarchy.cpp	Hierarchy.h	Class that support hierarchy mesh implementation.

Main.cpp just creates the window and an instance of App3D who is the class that handles all the code regarding with DirectX.

CODE ORGANIZATION

The main classes in the project are Camera, App3D, Mesh, Player, Net, SoccerBall, GoalKeeper and CollisionMesh. There are some supporting classes such as SDKsound, SDKwavefile Hierarchy that help with certain functions as Audio or Animation. Other kinds of classes manage specialized cases such as CollisionSphere and CollisionWall which offer collision shapes of sphere and wall respectively.

APPLICATION CONTROL AND CAMERA



Those classes define the only camera used in the application and the application control. Due to the nature of the game, a fixed camera that shows half of the soccer field is defined. The decision regarding why the game has just one fixed camera was taken considering that a soccer game with only one controllable character need to be able to show the whole field at any

moment. The camera class contains fields that define the direction of the camera (cameraLookAt), the position of the camera (cameraPos) and the normal vector (upVector).

The fields initial_camera_pos, initial_cameraLookAt and initial_upVector are the vectors for the initial state of the camera.

MESH

This class describes a mesh which is the wrapper for every operation that modifies any mesh. However, depending on its function some specialized classes which extend from the mesh class were defined. In the scene there are 5 meshes: the player, the goalkeeper, the ball, the net and the field.

This class was created in order to not repeat code, so it implements the loadFromX method which loads a mesh from a file. Also, there are some methods who transform the mesh like:

```
void move_to_origin(void);
void set_initial_position(float x, float y, float z);
void move_to_initial_position(void);
void scale(float factor_x, float factor_y, float factor_z);
void rotate_in_y(float angle);
void rotate_in_x(float angle);
void rotate_in_z(float angle);
```

Those methods modify the world matrix of the mesh, being called any time that the show method is executed. The following figure shows the classes that manage the mesh information.

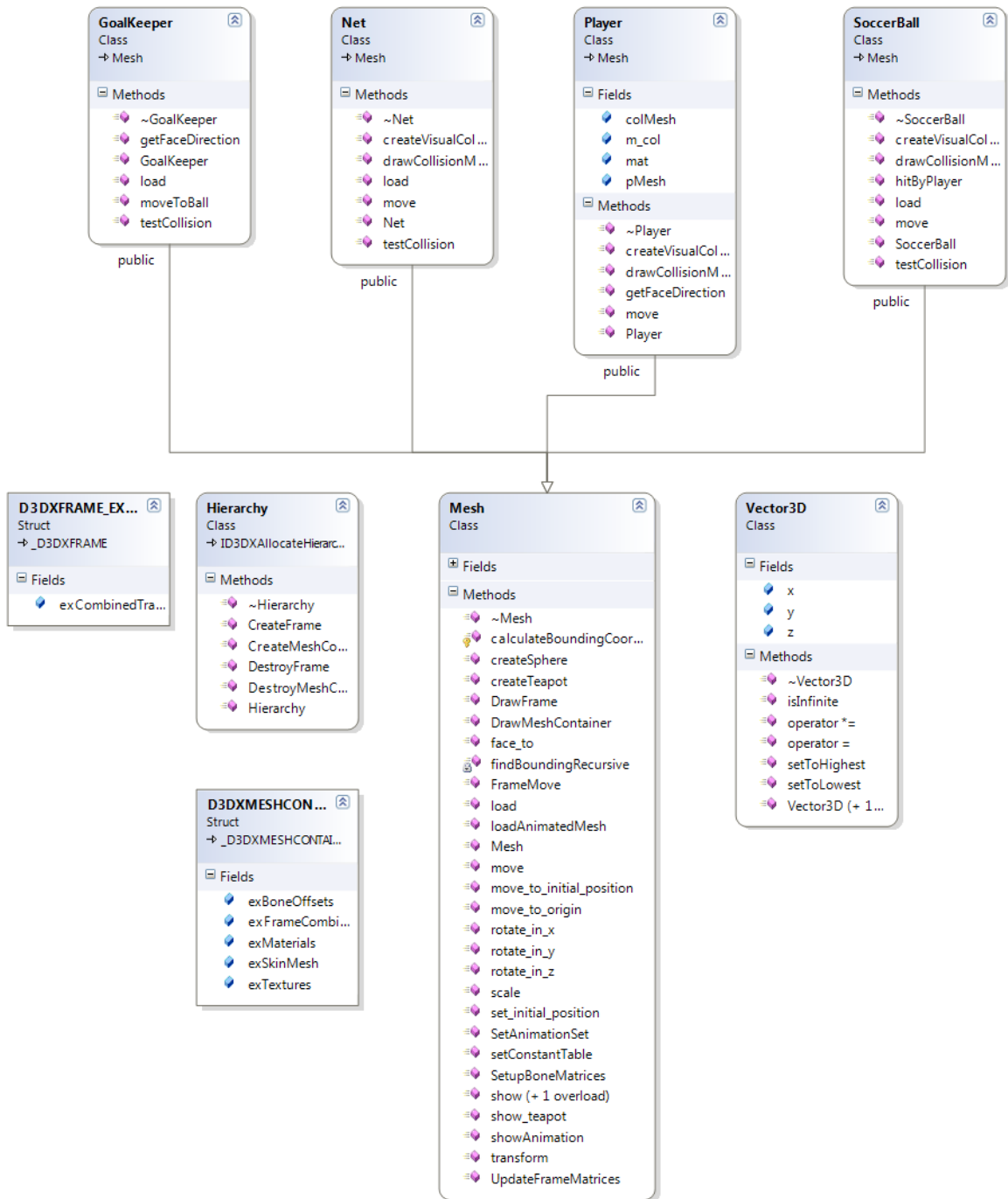


Figure 4 Structure of mesh classes

Since this game uses skinned meshes, some classes and methods were written to support this feature. The method `loadAnimatedMesh` is used to load the player and the goalkeeper mesh which are meshes with animation information.

In the resources available there was a lack of good skinned meshes that could fit in this project. So, a basic skinned mesh was produced in Maya animating the bone structure in a walk cycle. For each bone, a set of weights is produced to set up how much a bone transformation is going to modify a particular vertex. Once produced, the mesh was exported to an X file.

Since skinned meshes have a hierarchy organization of bones, a proper method which loads the hierarchy was required. The X file format was designed to include generic data in a hierarchy fashion. The drawback of having a generic way to include any kind of data is a more complex method to load that data. In the case of animation, though every element is properly linked they are not allocated in an organized way. So a recursive function was implemented to load a chunk of data, check if that data has siblings or children and repeat the process with any child or sibling.

There are some extensions to the Mesh class that manage the specific logic required in a particular case. The classes GoalKeeper, Player, Net and SoccerBall are intended to load static or animated meshes and to implement some game specific features such as collisions.

COLLISIONS

The collisions in the project were implemented using two shapes: Spheres and Boxes. In the case of spheres, a center point and a radius are defined. In the case of boxes, the data required are two points which represent the top left corner and the down right corner. Using both points as reference the proper values for width, height and depth were calculated.

The Figure 5 shows the structure of classes which manage collision events. The structure was designed to manage several shapes of collisions which offer flexibility when an extension is needed. However, only two shapes were implemented because they could manage all the requirements of the project.

The sphere shape is used to detect the collisions of the ball, the player and the goal keeper. The decision for the ball is obvious since both mesh and collision mesh share the same shape. However, the cases of the player and the goal keeper require some explanations. The first attempt for character collisions was to use a cylindrical shape. That shape did not work properly due to the morphological structure of the mesh which has a wide space between shoulders. That makes the radius too big and collisions with feet are not triggered properly.

To correct that problem a decision was made taking in consideration some features of the game. Since in this particular game the ball will not leave the ground, a sphere located in the character's feet makes sense and worked reasonable well in the experiments.

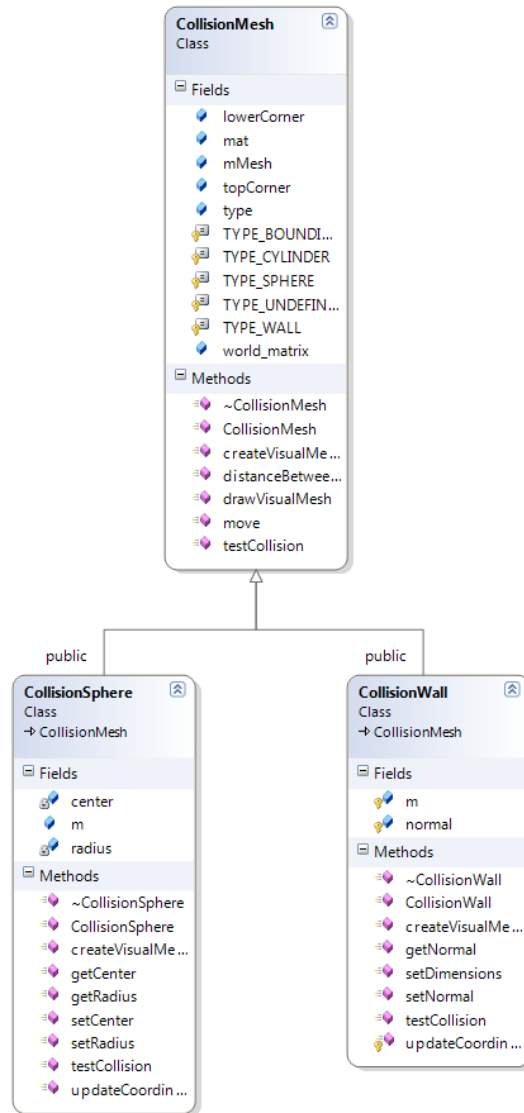


Figure 5 Collision meshes

For the net, a wall shape was used. The shape is called wall instead of box because only one face is tested and in the case of the net is the face that is facing the player. The Figure 6 shows how the collision meshes are located for each kind of mesh in the game.



Figure 6 Collision meshes in the scene

WORLD, VIEW AND PROJECTION TRANSFORMATIONS

Each object has its own world matrix, so if a mesh or even a collision mesh has to be moved, what is happening is that their world matrixes are being updated and they will be used the next time that a show method is called.

View and projection transformation matrices are initialized in the setup method of the camera class. Basically, a perspective is established with the following parameters:

Field of View = $\pi/4$

Aspect Ratio = $800/600 = 1.33$

Z-value of the near view plane = 1

Z-value of the far view plane = 1000

SOUND

DirectSound is used to playing sounds in the game. The Figure 7 shows the classes structures that are being used in the game. CSoundManager is in charge of manage the initialization process while the classes CSound and CWaveFile are in charge of playing sounds and load wav files respectively.

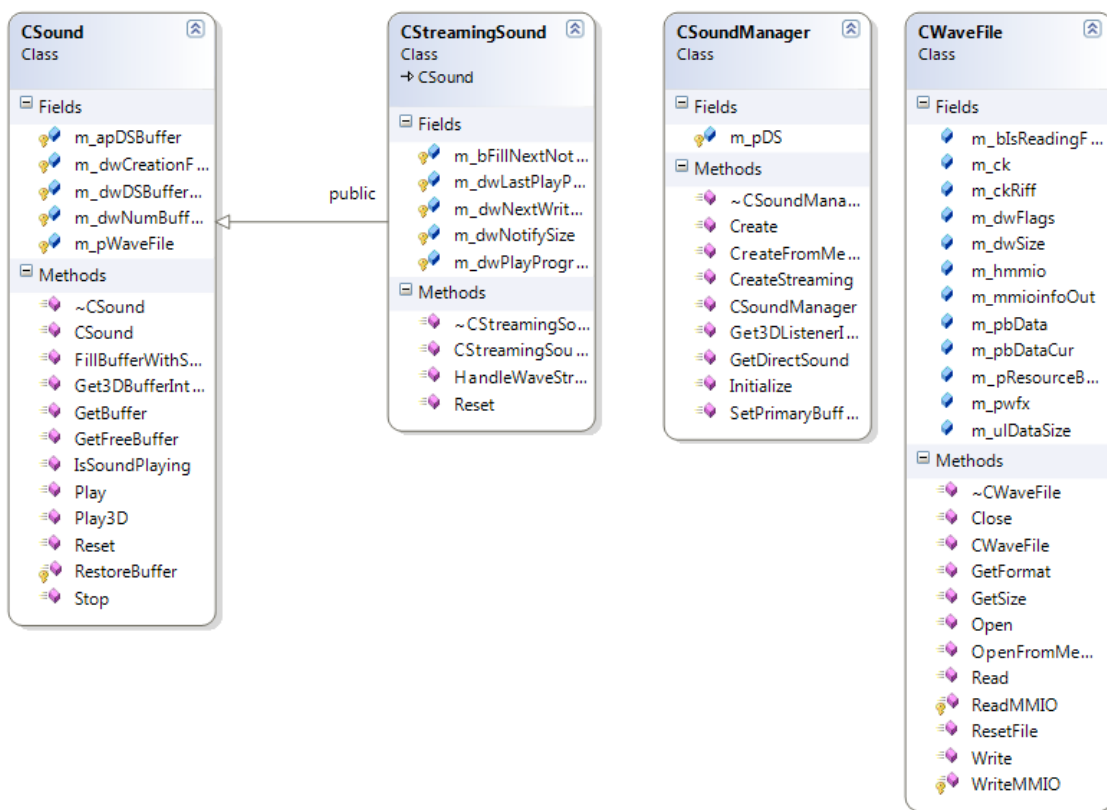


Figure 7 Sound classes

INPUT

DirectInput is used to get the information from the mouse. The whole process of initialization and updating is done in the App class. The mouse is used in order to kick the ball, the horizontal movement of the mouse before to press the left button will modify the intensity of the kick. In other words, a longer movement of the mouse along the X axis is going to produce a harder kick of the ball.

CRITICAL APPRAISAL

In the beginning of the project, was planted the idea of a mini soccer game with 2 teams of two people per team and an independent goal keeper. That idea was reduced mostly because a lot of time was spent in the animation process. One of the more time consuming tasks was generate the proper art asset in Maya. If a premade animation character was available the whole animation development time could be reduced. However, the animation process was not trivial and figure out how to load a hierarchy from an X file was not an easy task either. Even with the proper art the original idea could be big enough to be reduced.

Although the animation support was the most time consuming task, the result worth it. Also, once coded the inclusion of new animation meshes is trivial and the code is generic enough to be used in others DirectX projects.

Regarding the code organization, although is a good idea to have the code encapsulated into classes there is still space for improvement. Some areas look well like the collision mesh organization and the structure of meshes and children classes. Anyway there some areas that could be coded better. The display method of App class for instance, it is too big and could be reduced moving some code to the Mesh class. Unfortunately, there was no time to perform that refactoring.

One topic that was missing was rigid body physics. The ball is always on the ground mainly for simplified the process but the gameplay could be improved greatly with the use of a game physics engine such as PhysX or Havok.

Other issue that was missing is the independency of the Luna's framework. It could be rewritten in order to be more general. Anyway, because of the use of vertex shader there was no need to adjust the library files, unlike previous project where the lights parameters and materials had to be adjusted in the library files which is undesirable in a framework usage.

The decision of use a cartoon shader as a render technique allowed giving to the game a humorous look with combine with the kind of character used in the game. An outline that emulates an ink outline could be developed but once again the time was compromised and this feature was dropped.

In summary, there were some effects that unfortunately could be managed better in order to be included in this project. A better organization of code is still possible and necessary and in the future and for the sake of the game it would be mandatory use or develop a physics engine.

REFERENCES

F. Luna, 2003, Introduction To 3d Game Programming With DirectX 9, Wordware Publishing.

P, Walsh, 2003, Advanced 3D game programming using DirectX 9.0, Wordware Publishing.

J,Adams, 2003, Advanced Animation with DirectX, Thomson Course Technology

Microsoft, 2008, Visual Studio Help and MSDN website.

Thanks to my friends of the lab who help me every time that I had a problem, specifically Spencer Congdon, Erick Stock, Tommy Brett, Sunny Atwal, Jess Luger and Andy Lu.