Newton's Cradle

Computer Graphics Final Project

Aldearaban Group
Cadenas Quijano, Patricia
Cotrina Fernández, Manuel
García Tomillo, Javier
Hijarrubia Bernal, Luis

Universidad de Valladolid

Contents

1. Introduction

This document aims to present the final project developed by the Aldearaban group for Computer Graphics.

The project carries out a simulation of the famous Newton's Cradle (shown in the picture from the cover), adding different usage modes which allowed us to take advantage of most OpenGL characteristics.

This documentation is structured in three main parts. First, we include the planning with a Gantt diagram along with its description. Then, we will introduce the physics related to the simulation, as long as the equations used to get a realistic approach. In the third part we get into the main element of the project, and we thoroughly describe the different modes and the technologies used in the development.

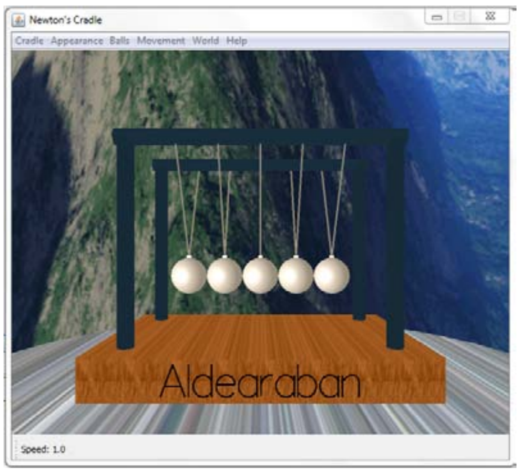


Figure 1. Main window

2. Planning

The planning has been made using the Unified Process, and can be shown in the Gantt diagram from the appendix.

In the first phase, Inception, the group meets to make a previous analysis of the project, establishing the initial use cases, the requirements and the possible risks which can occur during the project development. Besides, risk management is taken into account and a preliminary architecture is designed. In this phase the group meets several times within five days trying to reach an agreement.

In the second phase, Elaboration, for three days, the group meets to set a final structure which guides us along the process. It also includes the approval of the teacher about the use cases established in a second version. The architecture is completed.

In the third phase, Construction, the group makes fewer meetings, but we keep in touch almost daily by email, and using the online repository to store the advances made in the code safely. For almost a month, the group is devoted to code development, and in the last half of the month the documentation is started and developed thoroughly.

In the fourth phase, constant meetings are set, to put in common everything that has been developed, check it and value it, not forgetting to complete the documentation and prepare the final presentation.

3. Theoretical Base

Theoretically, Newton’s Cradle proves two physic laws: the law of conservation of energy, and the law of momentum.

The first one represents the first principle of thermodynamics and states that the total amount of energy in any isolated system (without interaction with any other system) remains constant in time, though that energy can be transformed in another kind of energy.

The second law states that the total amount of movement of every closed system (one which is not affected by external forces, and whose inner forces are not dissipating) cannot be changed and remains constant in time.

Obviously, the resolution of our problem with these physical laws was not the main goal, but they have been taken into account when simulating.

In order to simulate these two laws, we first need to care about the number of balls which will be moved, and the angle to which they have been dragged. Those are the data which will be carried to the other side of the cradle: the number of balls moving will be the same as the number of balls thrown, and they will move until the angle from which they were released. The time when the movement of the “opposite” balls (to the ones thrown) starts will be conditioned by the angle acting as counter being 0, that is to say, the ball thrown in the most centered position gets to its initial location.

This way we achieve what we have named after “**linear movement**”. This equation achieves the main goal but its appearance is unreal. In order to get this desired appearance we have created the “**quadratic movement**”. In it, while the ball is going down its speed increases and while it is going up, its speed decreases progressively until it stops and starts to fall again. Executing the application will show the more realistic appearance of this method.

Considering the laws previously mentioned, we will have achieved the simulation of the cradle if energy was only conserved attending to the movement, but in the real world that does not happen. In a real Newton’s Cradle, the friction of the materials, the air, or even the sound emitted by the balls make the movement finite. This is the reason why an option of applying a friction coefficient (1.5 in this case) is implemented. This coefficient will be subtracted to the maximum angle so that each time the balls go up a smaller distance, until they end up stopping.

Equations:

Linear movement: $angle = last\_angle \mp (angle\_increase * speed)$

Quadratic movement: $angle = \left(\frac{maxangle \mp last\_angle}{12} + 0.2\right) * speed$

Friction: $speed = last\_speed \; - 1.5 friction\_coef$

4. Practical Approach

The structure of classes that we have followed can be seen in the class diagram in the appendix. The main classes are Newton1 (includes the GLPanel) and GLRenderer, which is in charge of all OpenGL functions. Apart from these, there is a class for each item (textures, lightning…). Those items which have different modes have been implemented with internal classes so that they work as a trigger. This way, in the display there is a single call for each item and therefore, it works faster, since its amount of work is lighter.

There is an important class called Movimiento which knows the exact time when the events must happen, so is related to many other classes (just like the GLRenderer is).

As previously mentioned, the application is composed of several modes which can be enabled or disabled at anytime. This is the reason why this documentation is divided into common issues and modes.

4.1 Common issues

Some aspects of the application are common for every mode, such as lightning, camera position, scenario (skybox), load of some objects from an OBJ file and application of sounds and OpenGL text.

It is important to note that the user can increase or decrease animation speed, by using the controls explained in the Help provided by the application. This change in the animation speed will affect the balls movement and the lights (moving sun).

4.1.1 Lights and camera

The scene is rendered using two light sources: an ambient light and a point source.

The ambient light achieves a uniform light level in the scene, and is set to light gray, almost white. As for the point source light, it emits light equally in all directions and it is located behind the camera, on the left.

The camera is set at the center of projection, but translated along the z axis. Nevertheless, it is not fixed, there is a keyListener which allows the user to move the camera around. This movement is restricted so that the user is not able to leave the cube (skybox).

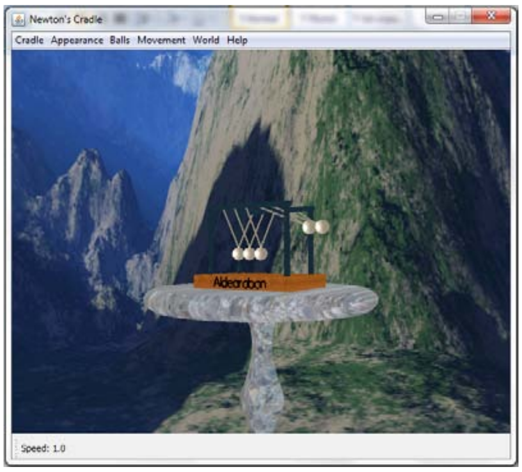


Figure 2. Camera movement and zoom

4.1.2 Materials and textures

Every object in the program has a material: the cradle, the skybox, the balls… But only the balls change their material from one mode to another. Material values are set in order to obtain the best appearance for each object. The wooden part of the cradle is solid and reacts softly to light but the table reacts harder.

The balls have special materials for transparent and firefly modes that have either transparent or emission factors.

The skybox uses eye linear mapping, while the table and cradle use object linear. Some visual modes use texture for the balls, some with sphere mapping, but cow mode needs to use eye linear as sphere mapping made the texture to always look at the camera. With eye linear and repeat mode we needed a cow tiled texture and with it everything worked as desired.

4.1.3 Skybox

In order to recreate the world where the cradle (along with the other objects) is located, we have taken advantage of the Skybox technique. In it, all the elements in the scene are put inside a box, preventing the user from moving the camera outside this box.

The cube has been basically rendered, and then applied a 170 scale, big enough to put the cradle and the table inside it and allowing the user to move around. The drawing mechanism is the same as for a ball: a new class is created with a draw function inside it, which will be called by the display method in the Renderer.

The skybox's texture can be chosen from the main menu, changing the environment instantly. Just like balls textures, a class has been written for them. This class is similar to the others, that is to say, it is able to listen to the events coming from the menu so that the texture can be changed to the corresponding option chosen.

When loading the texture, different configurations were tested until the desired effect was achieved. This was possible by applying "Eye Linear Mapping" and "GL_CLAMP" as a parameter, since this way it applies the texture to the whole cube without repeating it.

It is interesting to note that the first option which we tried to carry out was to include the scene inside a sphere, but the results were not satisfying.

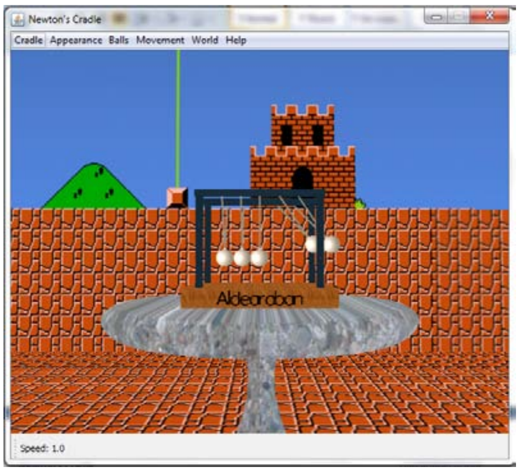


Figure 3. Videogame skybox

4.1.4 Load from external files: OBJ

As specified in the project wording, part of the scenario is loaded from an external file. For this matter, OBJ format is chosen.

This type of format is a geometry definition file format first developed by Wavefront Technologies for its Advanced Visualizer animation package. The file format is open and has been adopted by other 3D graphics application vendors. For the most part it is a universally accepted format.

The internal structure of an OBJ file is:

# List of Vertices, with (x,y,z[,w]) coordinates, w is optional.
# Texture coordinates, in (u,v[,w]) coordinates, w is optional.
# Normals in (x,y,z) form; normals might not be unit.
# Face Definitions

In this specific project, the OBJ files contain the description of the table, the base of the cradle and the cylinders which hold the threads of the balls.

In the application, the ModelLoaderOBJ class looks for the OBJ/MTL file in the selected path and reads it. Once read, the process is divided into two: on one hand the content of the OBJ file goes to the GLModel class and, on the other, the MTL file goes to the MTLLoader class.

In the GLModel class, the content of the OBJ file is parsed, this way joining in different sets object vertices, normal vectors, texture coordinates and object coordinates. Once this procedure is finished, the object is made by joining the different polygons obtained when parsing the OBJ file, then the material properties generated by the MTLLoader class are applied to it, and it is finally drawn.

In the MTLLoader class, the file is loaded in a similar way to the one used for the OBJ file, getting this way the different properties of the material for each polygon. When the parsing is done, these properties are sent to the GLModel class so that they are applied to the object.

4.1.5 Sound

As an extra, we have included the emission of sounds every time the balls collide.

In the real world, this sound is one of the causes of the movement stop, since the energy is transformed into acoustic signals, so it must be present in the scene.

Obviously, the sound depends on the material of the balls. This way, in the casper mode we will hear a bubble-like sound, whereas in the cow mode the sound will be a moo.

Two new clases have been developed to carry out sounds. The first one is called Sonido, and it is used to define all possible sounds in the application and is responsible for activating the right one. The second one (ReproducirWav) is in charge of playing the sound itself. The behaviour is simple: when the angle between the balls is 0, ReproducirWav is called and creates a new thread to reproduce the sound. The creation of the thread is required since otherwise, the movement will stop until the sound finishes, because they will share the same execution thread.

It is important to note that sound is only enabled when speed is set to 1x, since if it is increased sounds will probably be superimposed due to the faster collision of the balls and if it is decreased the sound should be a little slower.

The library used for playing sounds is javax.sampled, included in the Java Development Kit since its 1.3 version. This library provides interfaces and classes for capture, processing, and playback of sampled audio data.

4.1.6 OpenGL Text

In order to include text in our system we have used the following functions provided by the GLUT library:

- glutStrokeCharacter: Renders a character on screen, with the font chosen as a parameter.
- glutStrokeLengthf: Returns the length of the string to be shown. It is used for the calculation of the factor by which it should be scaled, so that the text is shown in an appropriate size.

A class fully dedicated to text printing has been developed. This way, in order to draw a text on screen you just need to declare an object of Texto type, and call the function Dibujar, with the parameters: text, color and x,y,z coordinates for its location. This position is relative to the central point of the set of balls in the cradle.

The front side of the cradle is the place chosen to show the text 'Aldearaban'.

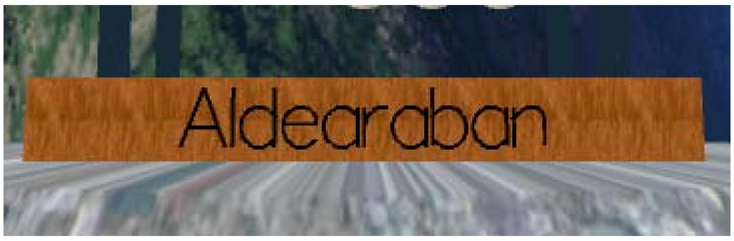


Figure 4. Text in the base of the cradle

4.2 Casper mode: transparency

In order to make the transparent mode, a new material was created. This material has an Alpha value.

Some modes had to be activated in the OpenGL engine, such as blending, alpha test, changing the alpha function and blending function (Alpha source and One minus source alpha were used).

As only the balls are transparent and all the other objects are opaque some visual errors appeared. One of these errors was that the transparent objects which were behind the opaque ones, appeared to be above if the depth test was disabled. If this test was enabled, then the balls could be drawn correctly or not depending on the camera angle. Finally it was fixed using a different order to draw the balls depending on the angle of the camera, so they are always drawn back to front.

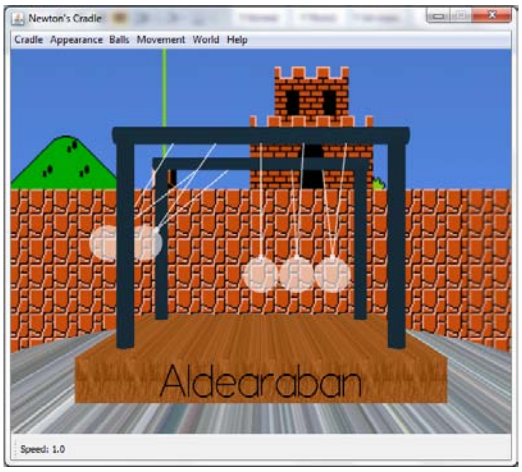


Figure 5. Casper mode

4.3 Firefly mode: emissive material

This mode is based in modifying the main scenario so that default lights are turned off (ambient light and point light source emulating the sun) and making each ball in the cradle look like a light bulb. Therefore, the balls will be the only light sources in the scene.

Every sphere (ball) now has a point light source of low intensity in its center. Furthermore, the emission property is enabled for the material of the balls. This way, the balls emulate being round little light bulbs emitting low intensity light. This light is most appreciable when the movement is enabled and consequently, the effects of the lights on the other objects in the scene can be shown.

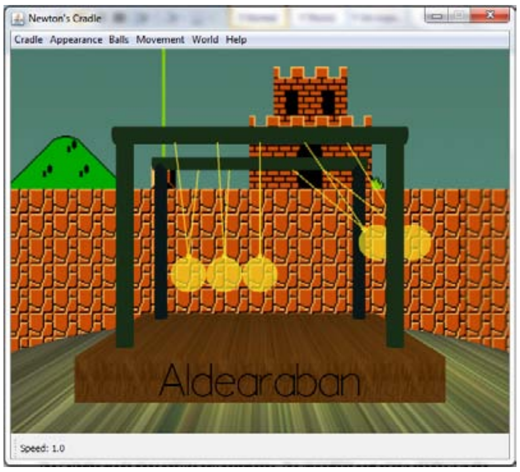


Figure 6. Firefly mode

4.4 Shaders

Shaders are low level programs executed in certain moments along the OpenGL pipeline. Two kinds of shaders have been used: Vertex shaders and Fragment shaders. The first ones affect vertices before they are rasterized, while the second ones manipulate fragments which come out of the rasterizer.

Some tests had to be done to check if the graphics card can handle shaders. If any of these tests fail, the program will end notifying the appropriate error to the user.

The Java code needs the shader program to be read from a string. Then the shader program can be activated. Both vertex and fragment shaders are considered as the same program in the OpenGL code and are attached together.

Two different modes are developed using shaders. Both used them to perform some effects on the balls and affect the whole ball, not every single vertex. In both of them the modified Phong model from Angel's presentations is used.

In the **vibration** model, a random number is generated in the Java code and then attached to the shader as a parameter. This number is later used to modify the Y coordinate of the vertex, making the ball randomly move up and down, making the visual sensation of vibration.

The **Colorize** mode does not use any parameter. The important part of this shader is in the Fragment one. Fragment coordinates are used to modify the fragment color. The result is balls changing their color while moving up and down, being this color different whether they are on the right or on the left of the screen.

Note that the shaders files must be in the project's root folder.

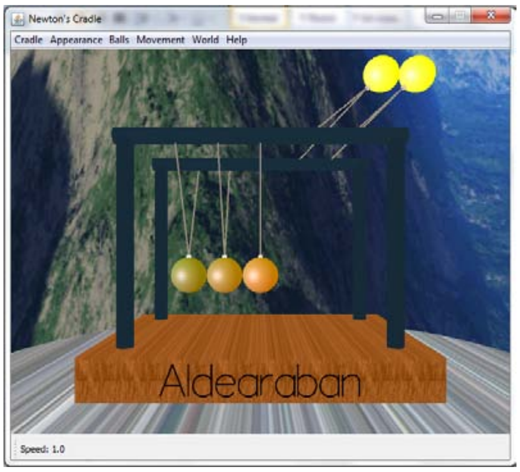


Figure 7. Colorize mode

4.5 Moving Sun

This is a feature that can be activated in any mode. When enabled, the point source light's location will rotate along Y and Z axis. This rotation is slow but can be clearly appreciated. The movement of the light source is affected by speed too: if we speed up the application, the light will move faster.

5. Conclusions

By developing this project we think that we have acquired the knowledge required in the subject. By building a complete project, including all possible options provided by OpenGL, we have understood its difficulty and we have solved the problems and complications which showed up (including a physic law simulation) in an adequate way.

The application takes advantage of the different capabilities usually used in an interactive multimedia system, from the answer to user actions (option choice from the menu, use of keyboard and mouse) to the implementation of lots of possibilities provided by OpenGL (use of textures, lightning, object importation from external files…)

We think that with this project we are close to the limit of JOGL capabilities, exploiting them at most. Since it is a barely used development platform and developed over Java, in the final version we have seen some slowness in the application execution and the lack of some functionality which was initially defined in OpenGL.

5.1 Future work

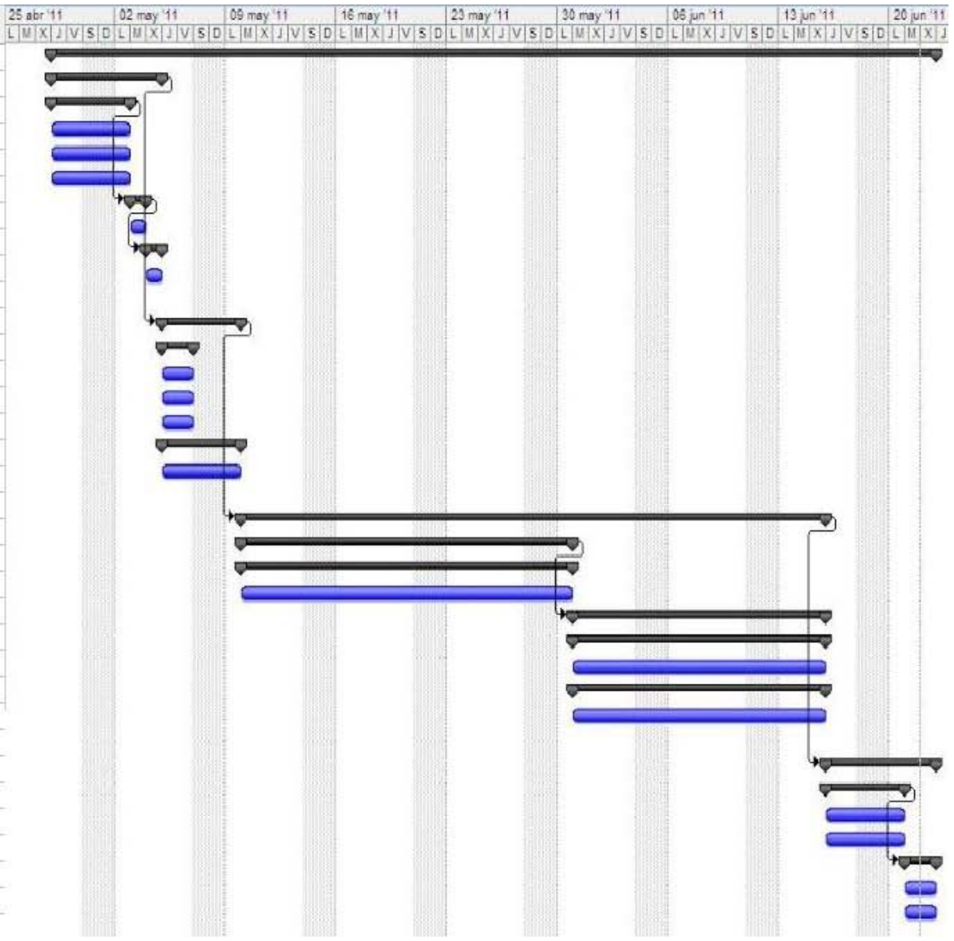
Nevertheless, there are lots of interesting things that could be added to this project. Some of them could have been implemented if we had had more time. Some of these possible additions are:

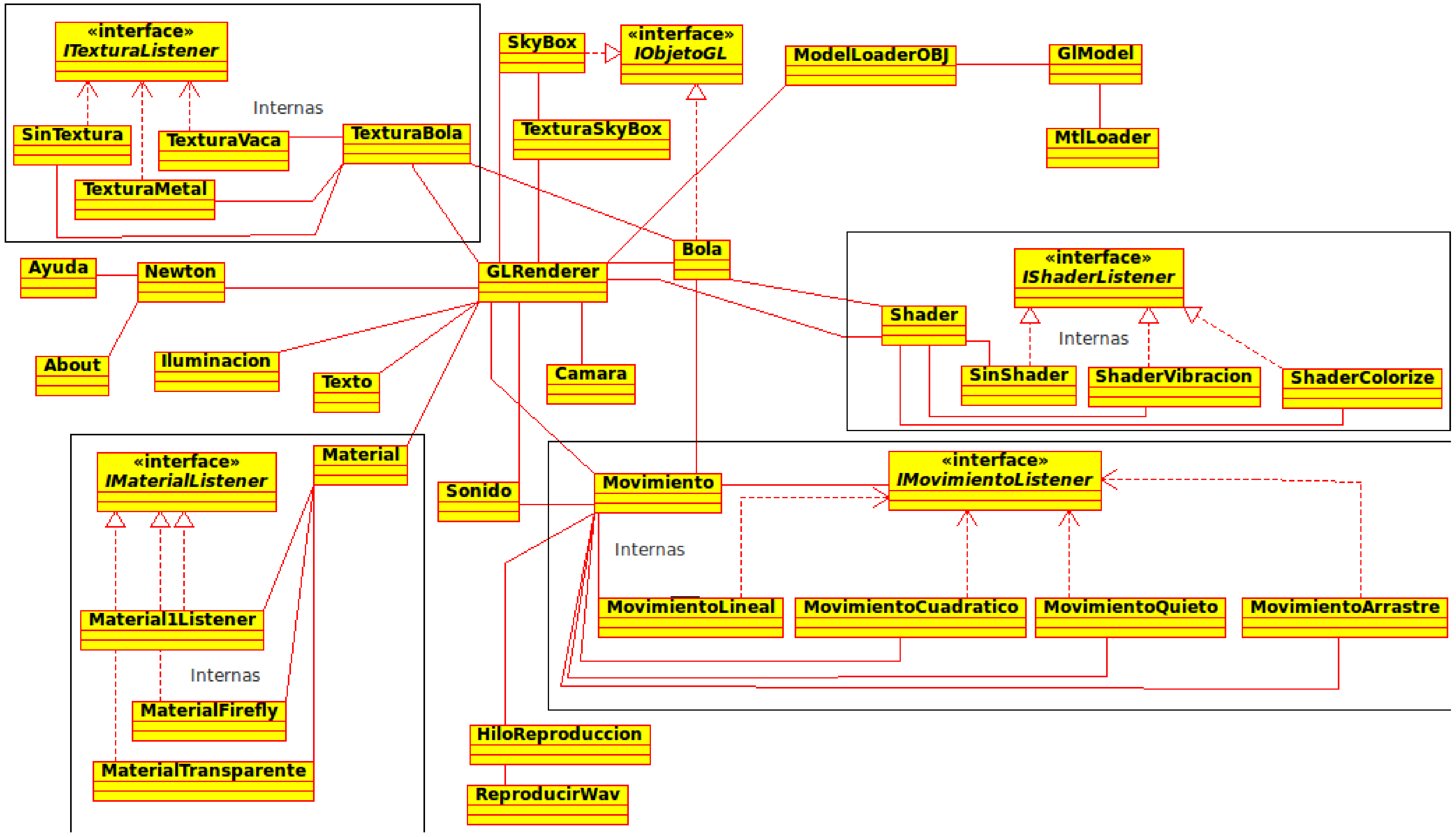
- Take advantage of the environmental mapping, making the balls reflect the surrounding world.

- Use bump mapping to make the balls look irregular
- Make the friction increase when the ball is higher, and decrease when the ball is in a lower position
- Make crystal balls which will be broken into pieces when colliding
- A possible new interface for the application could be showing different worlds from the start (little skyboxes) and then allow the user to select one of them to access it. In each skybox a different mode will be shown.

Appendix: diagrams

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	⊟ **Newtons Cradle**	**40 días**	**jue 28/04/11**	**mié 22/06/11**	
2	⊟ **Inicio**	**5 días**	**jue 28/04/11**	**mié 04/05/11**	
3	⊟ **Analisis**	**3 días**	**jue 28/04/11**	**lun 02/05/11**	
4	Casos de uso	3 días	jue 28/04/11	lun 02/05/11	
5	Requisitos	3 días	jue 28/04/11	lun 02/05/11	
6	Riesgos	3 días	jue 28/04/11	lun 02/05/11	
7	⊟ **Gestion**	**1 día**	**mar 03/05/11**	**mar 03/05/11**	**3**
8	Riesgos	1 día	mar 03/05/11	mar 03/05/11	
9	⊟ **Diseño**	**1 día**	**mié 04/05/11**	**mié 04/05/11**	**7**
10	Arquitectura candidata	1 día	mié 04/05/11	mié 04/05/11	
11					
12	⊟ **Elaboracion**	**3 días**	**jue 05/05/11**	**lun 09/05/11**	**2**
13	⊟ **Analisis**	**2 días**	**jue 05/05/11**	**vie 06/05/11**	
14	Casos de uso	2 días	jue 05/05/11	vie 06/05/11	
15	Requisitos	2 días	jue 05/05/11	vie 06/05/11	
16	Riesgos	2 días	jue 05/05/11	vie 06/05/11	
17	⊟ **Diseño**	**3 días**	**jue 05/05/11**	**lun 09/05/11**	
18	Linea base de la arquitectura	3 días	jue 05/05/11	lun 09/05/11	
19					
20	⊟ **Construccion**	**27 días**	**mar 10/05/11**	**mié 15/06/11**	**12**
21	⊟ **Iteración1**	**15 días**	**mar 10/05/11**	**lun 30/05/11**	
22	⊟ **Implementacion**	**15 días**	**mar 10/05/11**	**lun 30/05/11**	
23	Implementar casos de uso base	15 días	mar 10/05/11	lun 30/05/11	
24	⊟ **Iteracion2**	**12 días**	**mar 31/05/11**	**mié 15/06/11**	**21**
25	⊟ **Gestion**	**12 días**	**mar 31/05/11**	**mié 15/06/11**	
26	Documentacion	12 días	mar 31/05/11	mié 15/06/11	
27	⊟ **Implementacion**	**12 días**	**mar 31/05/11**	**mié 15/06/11**	
28	Implementacion de CU complementarios	12 días	mar 31/05/11	mié 15/06/11	
29					
30	⊟ **Transicion**	**5 días**	**jue 16/06/11**	**mié 22/06/11**	**20**
31	⊟ **Implementacion**	**3 días**	**jue 16/06/11**	**lun 20/06/11**	
32	Extras	3 días	jue 16/06/11	lun 20/06/11	
33	Version final del software	3 días	jue 16/06/11	lun 20/06/11	
34	⊟ **Revision**	**2 días**	**mar 21/06/11**	**mié 22/06/11**	**31**




«interface»
ITexturaListener
Internas
SinTextura
TexturaVaca
TexturaBola
TexturaMetal
SkyBox
«interface»
IObjetoGL
ModelLoaderOBJ
GlModel
TexturaSkyBox
MtlLoader
Bola
Ayuda
Newton
GLRenderer
«interface»
IShaderListener
Shader
Internas
About
Iluminacion
Texto
Camara
SinShader
ShaderVibracion
ShaderColorize
«interface»
IMaterialListener
Material
Sonido
Movimiento
«interface»
IMovimientoListener
Internas
Material1Listener
MovimientoLineal
MovimientoCuadratico
MovimientoQuieto
MovimientoArrastre
Internas
MaterialFirefly
MaterialTransparente
HiloReproduccion
ReproducirWav