1st Oral

From

Team

Romain Biessy Renaud Gaubert Aenora Tye Erwan Vasseure

Contents

Chapter 1

Introduction

Chapter 2

Communications

2.1 Twitter, Facebook and YouTube

2.2 Web site

Chapter 3

Licencing

Chapter 4

Terrain display

4.1 What was done

For the 1st oral presentation, the display system was the biggest problem we had, FPS were very low, around 30, even though we only displayed little Islands (5*5*5 chunks, which are 16*16*16 blocks tall). Therefore the game was laggy and unpleasant to play
This system was based on the idea that each blocks are composed of 6 faces that are vsible or not (Not visible means not added to the scene). When we detected visible faces, we added a unique entity to the scene. However, MOGRE as any other 3d engines had troubles supporting thousands of faces at a time (around 10 000 entities).

4.2 Optimizations

So we came up with another idea, which was a bit risky. Basically creating an entity in Mogre is very simple :

```
ManualObject block = new ManualObject("name");
block.Begin("texture here", RenderOperation.OperationTypes.OT_TRIANGLE_LIST);

    block.Position(new Vector3(0,   0,     0));
    block.Position(new Vector3(0,   100,   0));
    block.Position(new Vector3(100, 0,     0));
    block.Position(new Vector3(100, 100,   0));

    block.Quad(0, 1, 2, 3);
block.End();
```

As you can see, we only have to give the object position and to tell to draw a quad (which is actually a shortcut to creating 2 triangles). Basically, the idea we had was to gather all visible blocks of the same material in an instance of an object (which was called multiblock). However this was not simple for 2 reasons :

- We cannot tell the material to delete a face at a certain position neither can we add new faces to it
- We wanted blocks which had more than one texture on it (for example the grass cube doesn't have the same texture on the the top face than on the bottom face)

First of all you need to understand our architecture :

As you can see here each chunk contains an array of block. At first we stored in each cell a new instance of an object (The cube object) but !we made some changes and we now !instanciet! once blocks of

each material and the store the references to the corresponding block in the array.
¡explain¿ However with this model, we can only have one unique texture per block. Therefore, using inheritance, we came up with the following idea : in a base class, we create an abstract method called getComposingFaces which returns the material. In the derived class, if it needs multiple texture than it'll have to override this method, returning an array of string composed of the name of it's faces which would be ¡/explain¿

This system may seem a bit complicated, and it was, but it was totally worth the cost, we increased the FPS rate by 20, it now turns around 400. And, the Islands size were trippled, getting from 4*4 to 12*12 (on the x, z coordinates with y the height)

4.3 Results

4.4 Sky

At the first oral, we presented you a sky using the lilbrary Caelum but then we wanted to try another well-known library : SkyX. Once again we actually used a wrapper of the original library coded in C++. We wanted to try another library so that we could choose the best for our game.

Unfortunately it appeared that even if SkyX is a great library in C++ its wrapper is much less interesting. Indeed many functions weren't implemented in the wrapper. This lack leads to a far less awesome render than in the C++ version. Besides we had lots of troubles integrating this library to our project because of the different existing versions of Mogre.

That's why we ended up choosing Caelum. But we didn't made this choice because it was the first we saw but because this is the best one we can have.

Chapter 5

Player actions

5.1 Adding blocks

One of the downside of gathering all faces in one object is that you can't add faces to the object. Therefore we had to think a bit outside the object. What we did was the following : When the player adds cube to the terrain, we display the face as indepedent objects each and remember the position he added a block. When he has added 30 blocks, we launch a new thread whose role is to recreate each multiblocks that were modified.

First he must change the list of Vector3 the instance of Multiblock has and add the blocks the player added. Then he must recreate the object and when it's done remove the old one and replace it with a new one.

C# having a thread system which is easy to handle, this part was pretty easy.

5.2 Removing them

On the other hand removing blocks from the scene was a tricky part. Because the function we needed was not implemented in C# we had to get it from the C++ libraries. And, what's more, the function needed pointers so we had to use the unsafe option.
The basic idea is that a Manual Object is just an abstract layer to Vertex and Vertices. Thus we had to get the vertex buffer, Lock it, get the position of the first element and from there on tell what was the face number we wanted to access, and multiply this number by the "size" of a Vector3 this would set the pointer to the right position. From there on we only had to set for a face, it's 4 coords to 0 and check how many faces were visible.

However 2 problems arose : the first block with multiple structures, had to be dealt with in a separate way.
The second was that when removing a block, we had to refresh the surrounding blocks, thus most of the time creating faces which were not in the manual object (that was the cause of a bug). So we had to be careful not to suppress

Chapter 6

Terran generation

6.1 A new structure

6.2 Perlin 3d algorithm

6.3 Biomes

Chapter 7

Buildings

7.1 Constructions blocks

Unlike in Minecraft, our player can build some presaved buildings. This is the basic of a Starcraft-like game. These buildings will allow the players to spawn some bots for his army and also to gather few basic resources.

We don't want the player to build all of the constructions block per block, all he will have to do is to place a "construction block". Then when he clicks this block, a menu opens and the player will have to give the required resources to start the construction. For now we haven't the inventory system with the actual resources of the player that's why the construction menu is very simple.

7.2 Saving and loading the structure

Chapter 8

Physics

8.1 Collisions

As we want the best of our game, we tried to implement the library MogreNewt. This library provides a physic engine and handle collisions detection. It is a powerful tool but it implies to add each cube one by one as a ”Body” that is to say about 80 000 blocks. This leads to an important loss of FPS which is definitely what we don't want. Here you can see all the wires of the bodies including the character which was automatically created using its mesh file.

Eventually, we kept our system using the height points of each corner since we consider our character as a parallelepiped rectangle. This isn't an AABB collision algorithm because we look at the type of block there are next to the points.

8.2 Jump

As promised our characters can now jump. Once the collisions with the floor worked this wasn't a hard task. We use a second order polynomial to determine the actual speed of the jump depending on the time since the character has jumped. Then our gravity system takes cares to make him fall.

Chapter 9

Character

Chapter 10

AI

10.1 Pathfinding

10.2 A star (A*)

Chapter 11

Conclusion