The Jespa3D engine

Introduction

We present a 3D Engine designed and developed for the Game Boy Advance console, Game Boy SP and Gameboy Micro (in general, GBA, see figure 1) as a final undergraduate project of technical engineering career at the University of Girona by Jordi Espada Brau (project tutor: Gustavo Patow, PhD). The GBA console has a slow processor (ARMTDMI 32bit to 16,78 MHz), and it does not dispose of a floating point coprocessor, and because of this it does not allow 3D hardware operations, so we had to develop optimal algorithms to obtain an interactive tour on a 3D world (game engine).

The code has been developed in a combination of C and ARM assembler to achieve high performance. The code has been also adapted to the SDL library to be able to execute the engine on WINDOWS and UNIX platforms and, on other hand, to shorten development cycles for the engine.

Figure 1. *GameBoy Advance*

Characteristics of the engine

This engine **includes**:

- Render of world and sprites.
- Organization of the world structure in portals and sectors (to obtain an interactive tour).
- Complex maps with different levels (floors plants).
- Mapping of non-flat floors and ceilings.
- External landscapes by environment mapping.
- Mapping with local lighting.
- Mip-mapping of textures for produce views of great quality.
- Transparency effects.
- Sprites.
- Player collision with walls, floors and ceilings.
- Several optimizations.

This engine **does not** include:

- Map and object editor.
- Script System.
- Object management.

World data structure

The data structure of the world is 2D, based on the portals and sectors technique (figure 2). Every sector is convex and it is formed by a series of lines connected in clockwise order. In the figure, the solid lines represent walls and dotted lines represent portals that connect adjacent sectors. Every sector takes its own configuration, as intensity of lights, general height of ceilings and floors (for flat sectors), heights for each individual vertex (for non-flat sectors), etc.

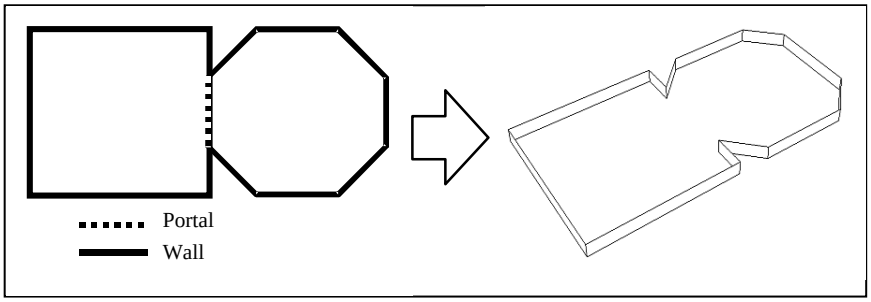

Figure 2. *Example of map 2D (left) and resulting 3D world (right)*

It is also possible to connect several 2D maps (floor plants), by changing the floor of a sector from solid to a portal. This way, we obtain a 3D data structure in layered floor plants (figures 3).

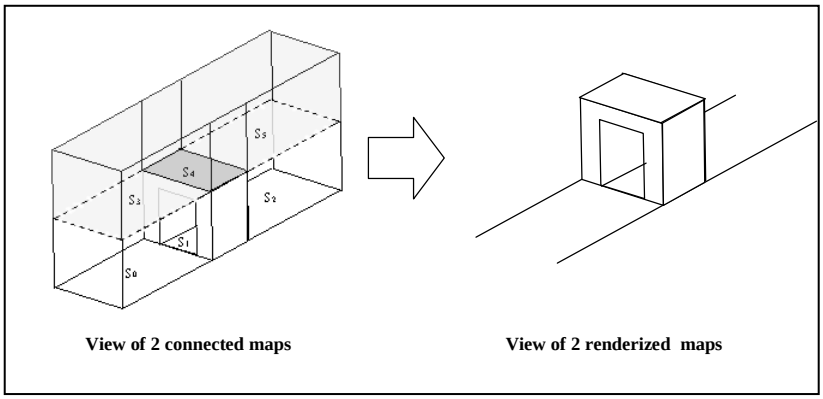

Figure 3. *Example connection of two 2D maps (left) and resulting 3D world (right)*

Rendering

The engine uses an 8 bits per pixel (bpp), and, consequently, it saves considerable graphics space. The engine renderer allows mapping of walls, planes and external landscapes (figure 5). Because of optimization aspects, the mapping of walls and exteriors has been carried out by vertical strips and the mapping of floors and ceilings by horizontal strips (figure 6).

Figure 5. *Example of a renderized floor, wall and external landscape*

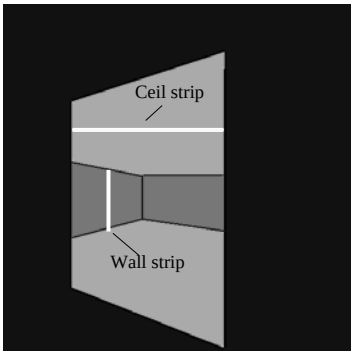

Figure 6. *The strip paints*

Lighting Mapping

Also, lighting effects have been implemented. Every strip has associated an intensity factor based on the strip-observer distance plus the sector own lighting (figure 7).

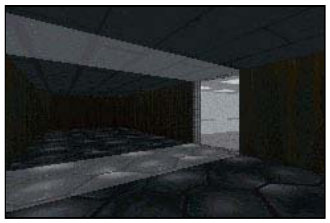

Figure 7. *Mapping with lighting*

Mapping portals with transparency

The engine is able to map portals with textures with sprites (figure 8).

Figure 8. *Mapping a portal*

Mipmap filter

In order to get better visual quality, the engine provides support for Mip-mapping filtering (figure 9).

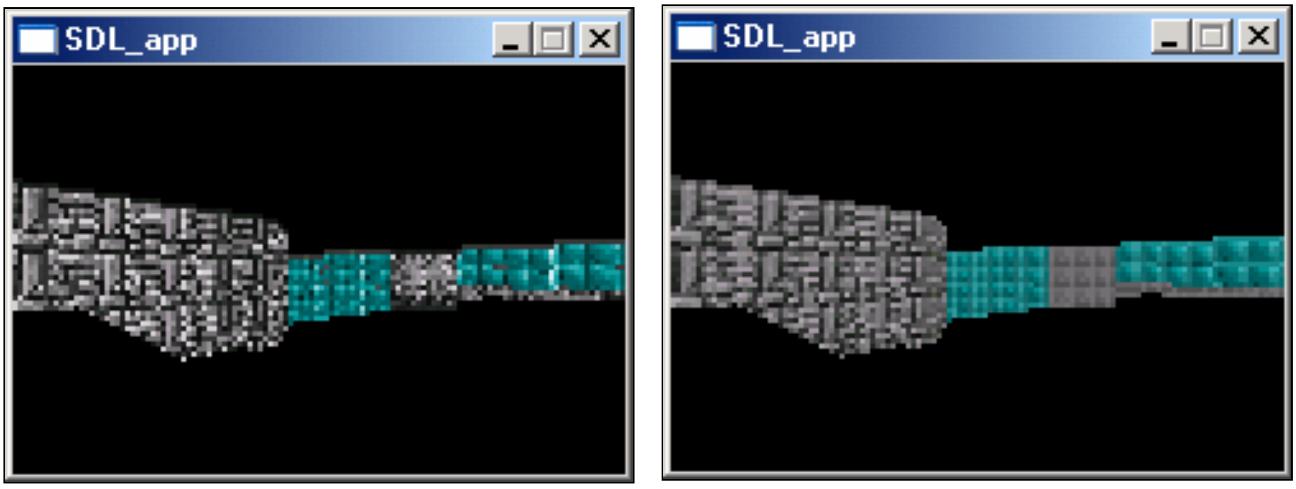

Figure 9. *Renderized without MipMapping (left) and with Mipmapping (right)*

Sprite Rendering

The engine supports animated sprites, with lighting and transparency effects, scaled according to the observer-sprite distance (figure 10)

Figure 10. A r*enderized sprite.*

Physics

Collisions

Detection and management collision of player with wall, ceil and floor.

Player actions

The player can move freely over the 2D plane, and it can jump, fly and crouch.

Rendering optimization

To obtain a faster execution in the rendering stage, the following features have been added:

- Use of Fixed Point Math.
- Use of assembler on time-critic functions like the painting of strips.
- Avoiding the use of high-cost operations like division, square roots, etc.
- Pre-calculated operations (stored in look-up tables, LUT)