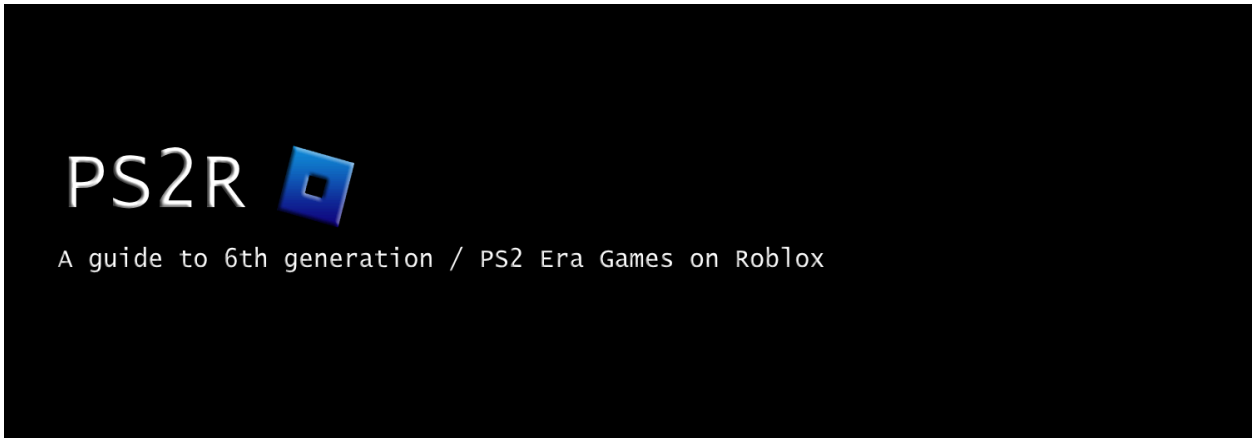


I see a lot of games trying to replicate the ps1/ps2 era gameplay and graphics but fail miserably. This guide will go into how those games, specifically Playstation 2 games were made at the time and how we can use a similar structure in Roblox Studio.

Before we start make sure you have:

- Basic knowledge of Roblox Studio
- Some scripting knowledge
- Blender
- Roblox Studio

I suggest you read EVERYTHING and not skip a chapter.

INTRODUCTION

Most people try to jump in, creating primarily horror games, the whole popularization of old games is because of horror games that started using the style. I believe horror games are amazing but they're so milked nowadays especially on roblox.

Some PS2 picks of mine that are NOT horror games:

- Socom 3: US NAVY SEALS (2005)
- Van Helsing (2004) *(horror-adventure)*
- Peter Jackson's King Kong (2005)

The reason i used these three games as example is because all of them have an extremely different gameplay and "vibe". This will be important when thinking of an idea for your own PS2 Game.

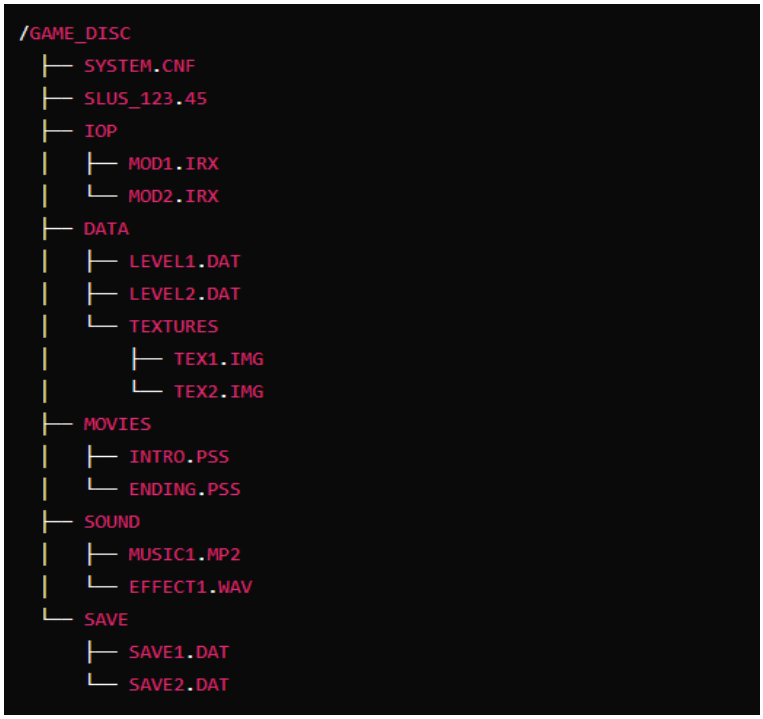
The first game, Socom 3 is a 3rd person shooter with multiple storyline missions,at the end of every mission you are prompted with a save file popup, to sum it up: *3rd Person, Mission System, Datastores.*

Second game, Van Helsing is a 3rd person game that uses a different approach to the 3d experience, *using scene-specific cameras, autosaving your progress and letting you open a "Armory" section at the end of each mission where you can upgrade your weapons and buy more.*

Third game, King Kong 2005, is a 1st person adventure game, *scripted dialog, sequences, lots of cutscenes, manual saving*

Basically i'd suggest picking a game style from one of these or exploring other games on the internet.

HOW DID IT WORK (1/2)



This is the file structure of a PS2 game.

We're gonna try to replicate this to the best of roblox studio's capability.

SYSTEM.CNF: This is a configuration file that contains the system settings for the game, such as the entry point for the executable and the disc's region code.
SLUS_XXX.XX / SLES_XXX.XX / SCUS_XXX.XX / SCES_XXX.XX: These files are the main executable files for the game. The naming convention varies based on the region:

- SLUS: North America
- SLES: Europe
- SCUS: North America (some early games)
- SCES: Europe (some early games)

IOP: This folder typically contains drivers and modules for the Input/Output Processor (IOP) of the PS2.
DATA: This is a general folder where game data is stored. It can include subfolders and files for assets like graphics, sounds, and levels.
IRX: These are drivers and modules specific to the game, often related to the IOP.
MODULES: This folder contains additional modules needed by the game.
MOVIES: Some games have a folder for cutscenes or in-game movies.
SOUND: This folder stores the sound effects, music, and other audio files used in the game.
TEXTURES: Graphics and texture files used by the game.
SAVE: This folder can contain save files or settings specific to the game.
ELF Files: Executable and Linkable Format files, which are used for the main game executable and other auxiliary executables.

HOW DID IT WORK (2/2)

The PS2's limitations in texture memory, texture size, polygon count, and rendering capabilities required developers to be creative and efficient in their use of resources. Despite these limitations, many games on the PS2 achieved impressive graphics for the time

We’re gonna stick to these limitations as if our lives depend on it.

Texture Limitations

1. **Texture Memory**:
    - The PS2 had 4 MB of dedicated Video RAM (VRAM), which was quite limited compared to modern standards. Developers had to manage texture memory very carefully.
2. **Texture Size**:
    - Maximum texture size was generally 256x256 pixels, although larger textures could be used by tiling multiple smaller textures.
    - Mipmapping and texture compression (such as S3TC) were sometimes used to manage memory more efficiently.
    -

3. **Texture Formats**:
    - The PS2 supported several texture formats, including 4-bit, 8-bit, and 24-bit color depths. Higher color depths used more memory and bandwidth.
4. **Texture Filtering**:
    - The PS2 supported bilinear and trilinear filtering, which could improve the appearance of textures when viewed at various distances and angles.
5. **Texture Mapping**:
    - Limited by the PS2's rendering capabilities, complex texture mapping techniques like bump mapping or parallax mapping were challenging to implement efficiently.

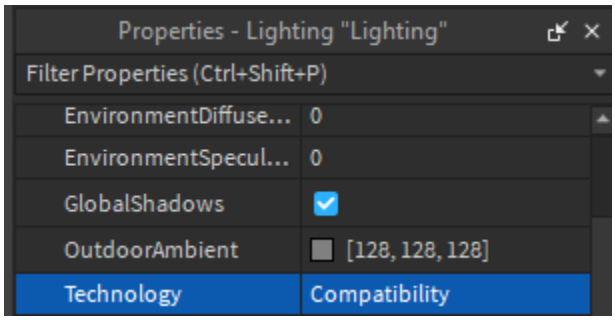
Polygon and Model Limitations

1. **Polygon Count**:
    - Theoretical maximum polygon count was around 66 million polygons per second, but this number was under ideal conditions without any shading, texturing, or other processing.
    - In practical game scenarios, the polygon count was much lower. Typical games ran at around 1-2 million polygons per second to maintain a balance between graphical fidelity and performance.
2. **Vertex Processing**:
    - The PS2's Graphics Synthesizer (GS) and Emotion Engine (EE) worked together to process vertices. The GS was responsible for rendering, while the EE handled transformations and lighting.
    - Vertex processing was limited by the EE’s processing power. Complex models with high vertex counts required efficient use of the EE’s capabilities.
3. **Lighting and Shading**:
    - The PS2 supported basic lighting and shading techniques, including Gouraud shading and Phong shading. Advanced effects like per-pixel lighting were challenging to implement.
    - Hardware support for real-time shadowing was limited. Many games used simpler shadow techniques, such as blob shadows or shadow maps with low resolution.
4. **Model Complexity**:
    - Models had to be optimized to balance detail and performance. Developers often used level-of-detail (LOD) techniques to switch to simpler models at greater distances.
    - Skeletal animation was supported, but complex animations could be CPU-intensive. Efficient use of bone influences and keyframes was necessary.
5. **Rendering Techniques**:
    - The PS2 supported various rendering techniques, including alpha blending for transparency effects and multitexturing for combining multiple textures on a single model.
    - Particle effects and special effects like reflections and refractions were possible but required careful optimization to maintain performance.

SETUP

1

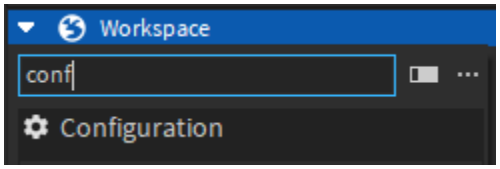
Create a new Game, Classic Baseplate.
Go into Lighting and change the Technology to Compatibility.



This will give us that crappy lighting that we need.

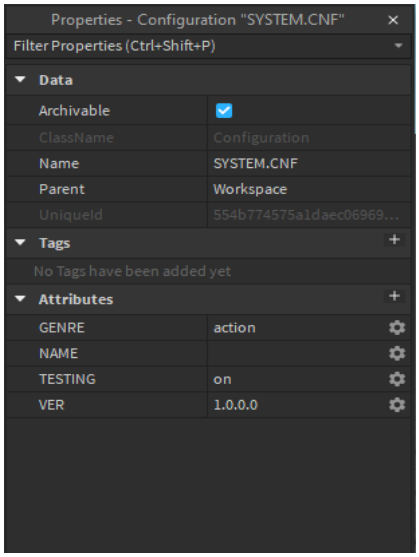
2

In Workspace, add “Configuration” this is usually used as a folder for tools and their configurations (damage, animations, etc) however we will use this as our SYSTEM.CNF file.



3

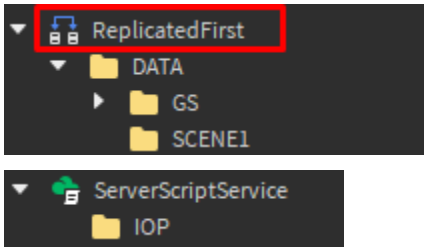
Go into it’s Properties, and add a few attributes, this is for testing purposes and video logging of the project.



Add GENRE, NAME, TESTING, VER.

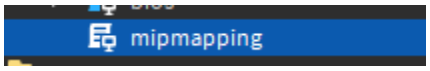
4

Make the following folders:
DATA -> GS -> SCENES
IOP

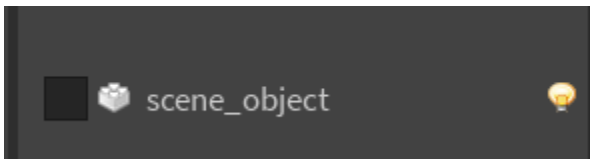


5

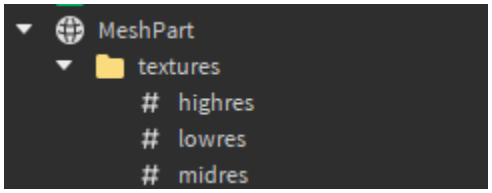
Install the TagEditor Plugin and enable it.
We're gonna make our own MIP Mapping.
Make a locaScript and place it inside of StarterPlayerScripts, rename it to "mipmapping" or whatever you prefer.



Open up TagEditor and make a tag called scene_object.



Now to choose your first model, take any MESH from the toolbox you want to test this on,
Once you have it in the Workspace, make a folder inside of it and put 3 IntValues in it,
Name them highres, midres, lowres.
Make sure the checkbox next to scene_object is checked (TagEditor window)



If the mesh already has a texture, use:
https://assetdelivery.roblox.com/v1/asset?id=0
To download the texture of the mesh (Replace the 0 at the end of the link with the ID of the textureID and paste it into your browser and hit enter, once it's downloaded drag it over into:

https://www.photopea.com/

Once the texture is open in photopea you can use the topbar button that says “Image” and click “Image Size” and change the Height and Width Pixels within the parameters of the texture limitations we mentioned above.
For the highres i suggest 128x128
Midres - 64x64
Lowres - 32x32
(For objects)

Upload these textures to the textureid and make sure the texture is set to the highest quality one

Then paste this code into the mipmapping localscript: https://pastebin.com/n7jbwHJJ

Now make a new localscript and rename it to bios, paste in: https://pastebin.com/8MaULJhk

Make a localscript IN bios and rename it to buffer, we’re gonna use this script to cap the FPS at 30 as many ps2 games were back in the day.

```
local RunService = game:GetService("RunService")
local MaxFPS = 30

while true do
	local t0 = tick()
	RunService.RenderStepped:Wait()
	repeat until (t0 + 1/MaxFPS) < tick()
end
```

CUTSCENES

If you’re making a story game you’re gonna want to use cutscenes, what separates the ps2 in this field from modern games is that they utilized pre-renders which made the cutscenes look higher in quality and allowed complete freedom over animating the character however they wanted without taking up too much memory from the system. As i am writing this (5.7.24) Roblox allows you to upload videos for a robux fee. I highly suggest to use this for your game. The maximum video quality you should go for is 480p.

SOUND

If you want an authentic ps2 experience you need sound effects for a lot of stuff, the main being the material footstep sounds, an audio file's details for a footstep sound would look something like this:
Format: ADPCM
Sampling Rate: 22 kHz
Bit Depth: 8-bit
Duration: 0.5 seconds
File Size: Approximately 5-10 KB (depending on compression efficiency)

Basically whatever sound you want to use, compress it and make sure the compression is HEARD. And especially avoid using a different % of compression for sound effects, you want the game to feel the same throughout, except for the footstep sfx (you can compress them more to save memory)

Use SoundService and Sound groups to store your audio

LEVEL/SCENE DESIGN

1. Prebaked Textures and Lighting

Prebaked Textures:

- Static Environments: To save processing power, textures for static environmental objects like buildings, terrain, and other scenery elements were often prebaked. This means that lighting, shadows, and details were baked into the texture maps during the development process.
- Lightmaps: A common technique was to use lightmaps, where lighting information (including shadows and ambient occlusion) was baked into a secondary texture applied over the environment. This allowed for more complex lighting effects without runtime computation.
-

Lighting for Characters:

- **Dynamic Lighting**: Characters and other dynamic objects typically used real-time lighting to interact properly with the environment and other light sources.

https://craigsnedeker.itch.io/classic64-asset-library

For your textures 🙂

An example of visually appealing level design:

Van helsing 2004

That concludes part 1 of this tutorial, in part 2 we will dive deep into the coding and texturing and so on

- Made by @juicewaree (roblox)
- Contact me via the devforum or discord (@juiceware)