# Renderer Vulkan stile PS1 (low-poly)

Confronto interattivo *Moderno vs Retro*

Hermann Tamilia Matteo Vitale

Corso di Computer Graphics – A.A. 2024/2025

8 settembre 2025

**Sommario**

Realizzeremo una piccola applicazione **C++/Vulkan** capace di renderizzare una scena 3D *low-poly* con due pipeline selezionabili: una moderna e una che simula lo *stile PlayStation 1* (PS1). Il progetto evidenzia come scelte e limitazioni tecniche (affine texture mapping, bassa risoluzione, quantizzazione colore, dithering, precisione ridotta dei vertici, assenza di mipmapping) influenzino l'estetica finale. La demo includerà un *toggle* per passare *on-the-fly* tra i due stili.

## 1 Introduzione

Obiettivo: costruire un *renderer didattico* che, pur mantenendo un'architettura moderna e pulita, consenta di attivare intenzionalmente artefatti tipici dell'era PS1 per comprendere il nesso tra hardware/tecniche e risultato visivo.

## 2 Obiettivi

1. Implementare un renderer Vulkan minimale ma robusto (swapchain, render pass, depth, pipeline, sincronizzazione).
2. Caricare e visualizzare asset *low-poly* (OBJ + texture low-res) con camera interattiva.
3. Simulare gli artefatti PS1 (sez. 6) e fornire un confronto interattivo *Moderno vs PS1*.
4. Redigere documentazione tecnica con scelte progettuali, metriche e riferimenti.

## 3 Requisiti

### Funzionali

- Rendering di una scena dimostrativa (3–5 oggetti, skybox opzionale).
- Camera FPS/third-person o *fixed angles* in stile survival anni '90.
- UI debug: FPS, parametri, *toggle* effetti (`1..9`).

### Non funzionali

- Target prestazioni: $\geq$ 60 FPS in *offscreen* 320×240 con upscaling.
- Portabilità: Windows 11 (sviluppo) + Linux nativo (test facoltativo).
- Qualità del codice: CMake, RAII, *validation layers* attivi in debug.

## 4 Stack tecnologico e toolchain

- **Linguaggio**: C++20.
- **API grafica**: Vulkan 1.2 (SDK LunarG).
- **Windowing/Input**: GLFW 3.x (o SDL2).
- **Math**: GLM.
- **Allocazione GPU**: Vulkan Memory Allocator (VMA).
- **Asset I/O**: tinyobjloader (OBJ), `stb_image` (PNG/JPG), opz. KTX.
- **Shader**: GLSL → SPIR-V (glslc/shaderc; opz. DXC).
- **Debug/Profiling**: RenderDoc; opz. Nsight/Radeon GPU Profiler.
- **Tool artistici**: Blender (modelli low-poly), GIMP/Aseprite (texture).
- **Versionamento**: Git + Git LFS per asset binari.

## 5 Architettura software (alto livello)

- **Core Vulkan**: instance, device/queues, swapchain, image views, depth buffer, render pass, framebuffer.
- **Risorse**: VMA (buffer/immagini), staging buffer, descriptor set layout (set0=global, set1=materiale), sampler.
- **Pipeline grafica**: vertex input (pos/norm/uv), rasterizer, depth test, blending semplice; 2–3 *frames-in-flight*.
- **Scene layer**: `Mesh`, `Material`, `Texture`, `Camera`, lista draw; simple scene graph.
- **Post-process**: *offscreen* low-res + fullscreen quad per upscaling/quantizzazione/dithering.
- **Input/Camera**: FPS o third-person minima; optional *fixed angles*.
- **UI debug**: overlay minimale (testo) con parametri runtime.

## 6 Tecniche grafiche PS1

- **Affine texture mapping**: UV interpolati in screen-space con `noperspective` (warp su piani inclinati).
- **Bassa risoluzione di rendering**: *offscreen* a 320×240/400×300, upscaling *nearest*.
- **Filtri/mipmapping disabilitati**: sampler `NEAREST`, `maxLod=0`.
- **Quantizzazione colore 15-bit**: RGB 5:5:5 (o 5:6:5) in fragment shader.
- **Dithering Bayer 4×4/8×8**: applicato prima della quantizzazione.
- **Vertex jitter/snap**: quantizzazione posizioni/NDC alla griglia pixel *low-res*.
- **Profondità limitata**: depth buffer 16-bit (o simulazione errori di sorting & z-fighting).
- **Fog / depth cueing**: nebbia lineare marcata per mascherare draw distance ridotta.
- **Palette/CLUT** (opz.): texture indicizzate + lookup palette.

## 7 Piano di lavoro (milestone)

| Settimana | Attività |
|---|---|
| 1 | Setup (CMake, SDK Vulkan, GLFW); finestra; *validation layers*; *hello triangle*. |
| 2 | Mesh/texture loader; camera; depth; lighting semplice; struttura scene. |
| 3 | *Offscreen* low-res; upscaling *nearest*; affine UV; quantizzazione colore; dithering. |
| 4 | Vertex snap; fog; UI debug; *toggle* Moderno/PS1; rifiniture; documentazione e demo. |

## 8 Collaborazione e gestione progetto

**Ruoli suggeriti.**

- *Developer A (engine)*: init Vulkan, memoria/descriptor/pipeline, I/O asset, post-process.
- *Developer B (shader & assets)*: shader base e PS1, guidelines asset low-poly, supporto tool artistici.

**Strumenti.** GitHub/GitLab, Git LFS, Kanban (Projects/Trello), issue per feature/bug, *code review*.

## 9 Metriche di qualità e testing

- **Prestazioni**: FPS medio e 1% low; risoluzione *offscreen*; tempi di CPU/GPU per pass.
- **Stabilità**: resize, alt-tab, perdita focus, *device lost*; catture RenderDoc.
- **Qualità visiva**: confronti A/B (Modern vs PS1), screenshot comparativi.

## 10 Rischi e mitigazioni

- **Sincronizzazione (stutter/deadlock)** → 2–3 *frames-in-flight*, fence/semaphore chiari.
- **Swapchain recreation** → funzione dedicata, test aggressivi su resize/minimize.
- **Gestione memoria** → VMA; tracciamento risorse; distruttori RAII.
- **Asset incoerenti con look** → linee guida asset (tris bassi, texture 128/256, palette limitate).

## 11 Struttura repository (suggerita)

```
/assets        # models/ textures/ (LFS)
 /models
 /textures
/docs          # relazione, note, immagini
/shaders       # *.vert, *.frag (GLSL -> SPIR-V)
/src
  app/         # init, loop
```

```
  gfx/          # device, swapchain, renderpass, pipeline, descriptors, vma
  scene/        # mesh, material, texture, camera
  io/           # loader OBJ/PNG
  util/         # timer, logger
CMakeLists.txt
README.md
```

## 12 Conclusioni

Il progetto offre un percorso pratico per comprendere come le scelte di pipeline e le limitazioni intenzionali plasmino lo stile visivo. La struttura modulare facilita estensioni (deferred, SSAO, particles) e riuso in contesti didattici.

## Riferimenti bibliografici


[1] D. Colson, *PS1 Style Rendering* (2021). Disponibile online: `https://www.david-colson.com/2021/11/30/ps1-style-renderer.html`.

[2] PIKUMA, *How to Make PS1 Graphics* (2024). Disponibile online: `https://pikuma.com/blog/how-to-make-ps1-graphics`.

[3] Wikipedia, *Texture Mapping – Affine vs Perspective Correct*. Disponibile online: `https://en.wikipedia.org/wiki/Texture_mapping`.