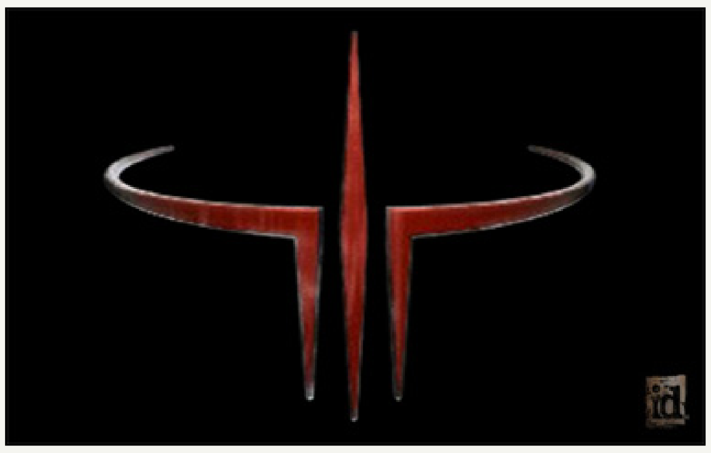
JUNE 30, 2012

QUAKE 3 SOURCE CODE REVIEW: RENDERER (PART 2 OF 5) >>

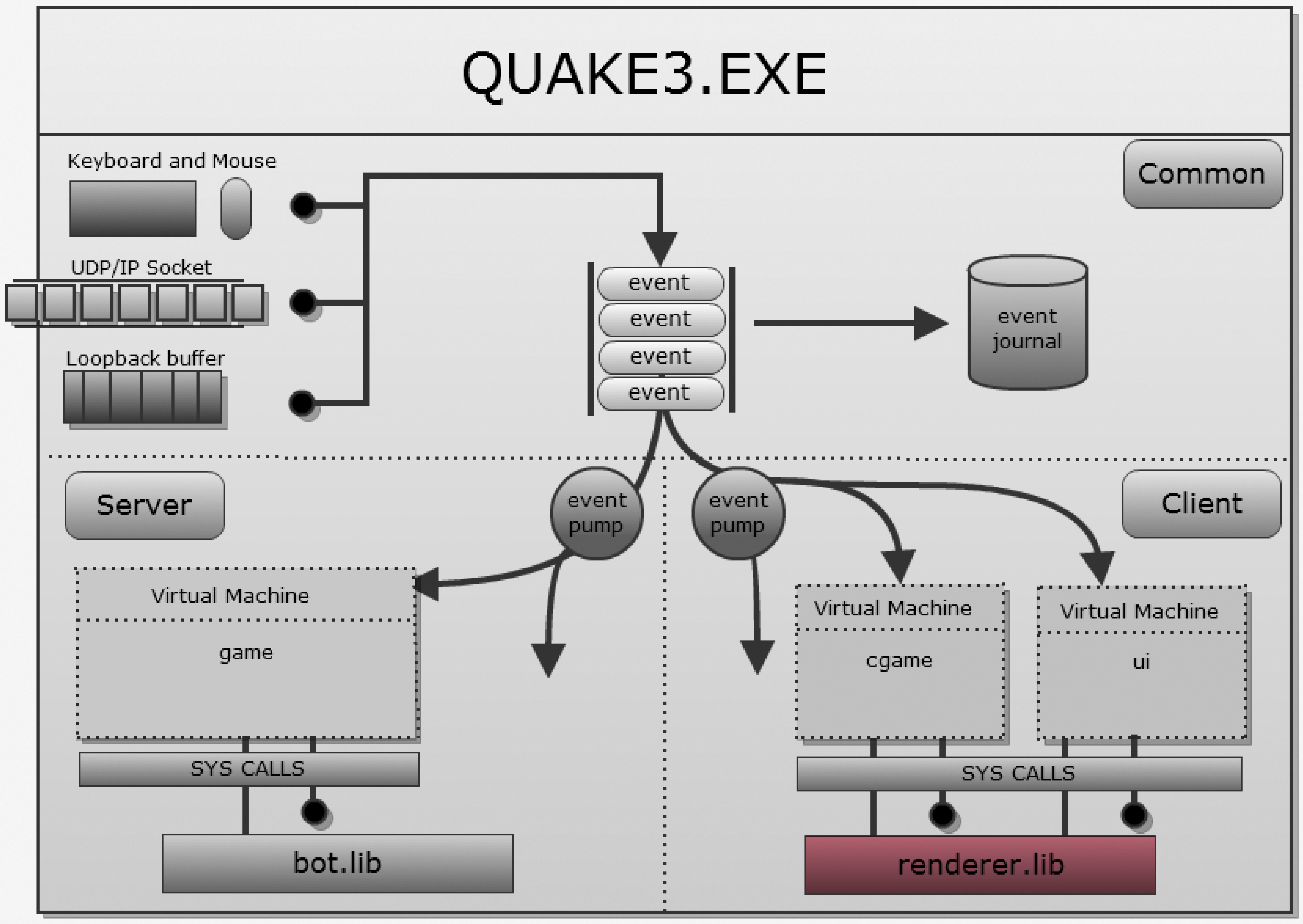
The Quake III renderer is an evolution of the Quake II hardware accelerated renderer: The classic part is that it is built on a "Binary Partition"/"Potential Visible Set" architecture but two new key aspects are noticeable:

- A Shader system built on top of the OpenGL 1.X fixed pipeline. This was quite an accomplishment in 1999. It provided a lot of space for innovation in an era before the now ubiquitous vertex,geometry and fragment shaders.
- Support for multicore architecture: The OpenGL client/server model is blocking on some methods and a system of threads partially eliminates this issue.



Architecture

The renderer is entirely contained in renderer.lib and statically linked against quake3.exe:



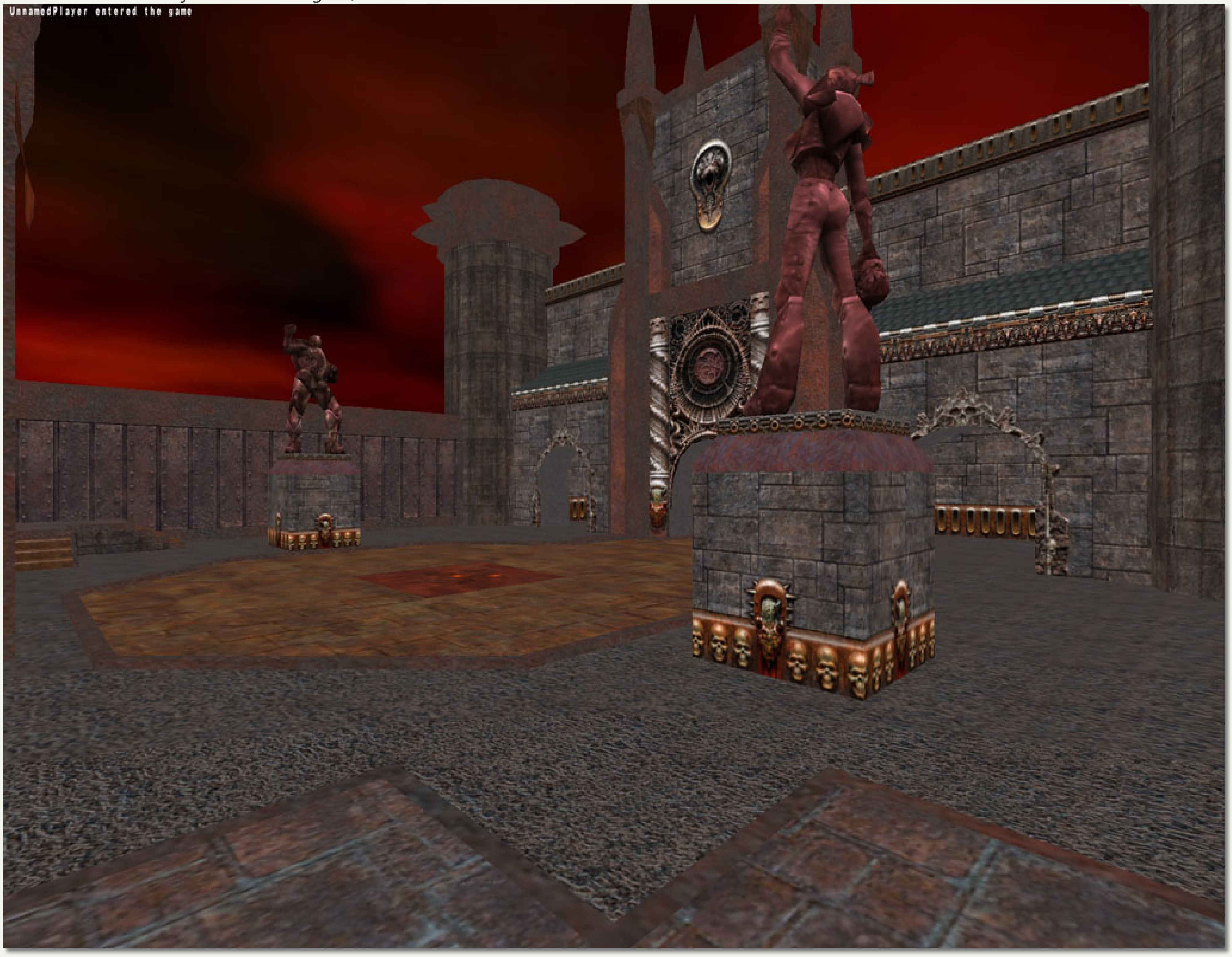
The renderer overall architecure is the Quake Classic: It relies on the famous BSP/PVS/Lightmap combo:

- Preprocessing:
  1. The game designer create and save a .map using QRadiant.
  2. q3bsp.exe slices the map via Binary Space Partitioning. I wrote about this in Quake1 renderer review.
  3. Out of the BSP a portal system is generated: I wrote about this in Doom3 Dmap tool.
  4. q3vis.exe uses the portal system and generates a PVS (Potentially Visible Set) for each leave. Each PVS is compressed and stored in a bsp file as I described it in a previous article.
  5. The portal system is discarded.
  6. q3light.exe calculate all illuminations for each polygon in the map and store the result as lightmap textures in the bsp file.
  7. At this point all the preprocessing (PVSs and Lightmaps) is stored in the .bsp file.
- Runtime:
  1. The engine loads the map and the bsp.
  2. When rendition is requested:
  3. The engine decompress the PVS for the current leave and determine was is actually visible.

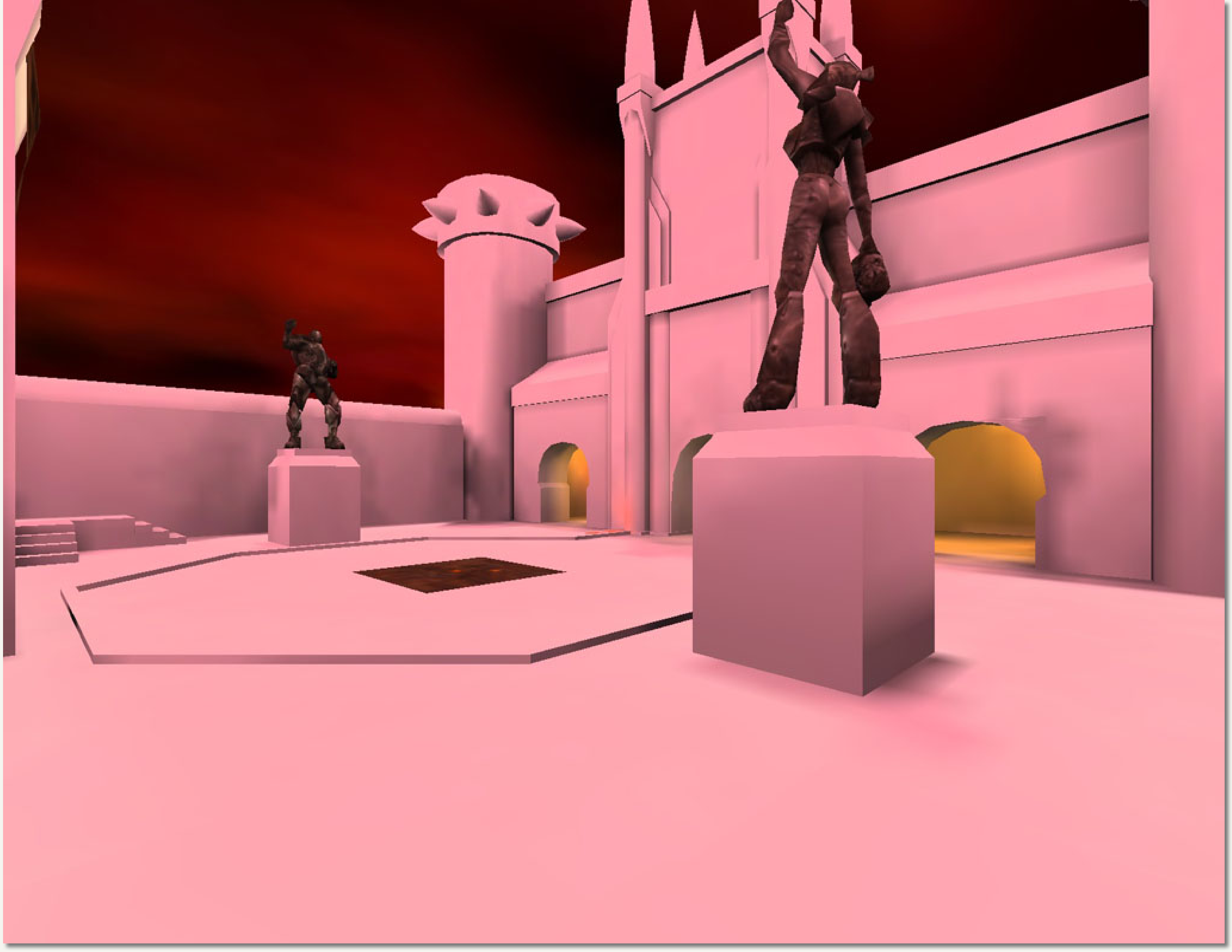
4. For each polygon it combines the lightmap with the color via multitexturing.

The multitexturing and lightmap step is clearly visible when the engine is modified to display only one or the other:

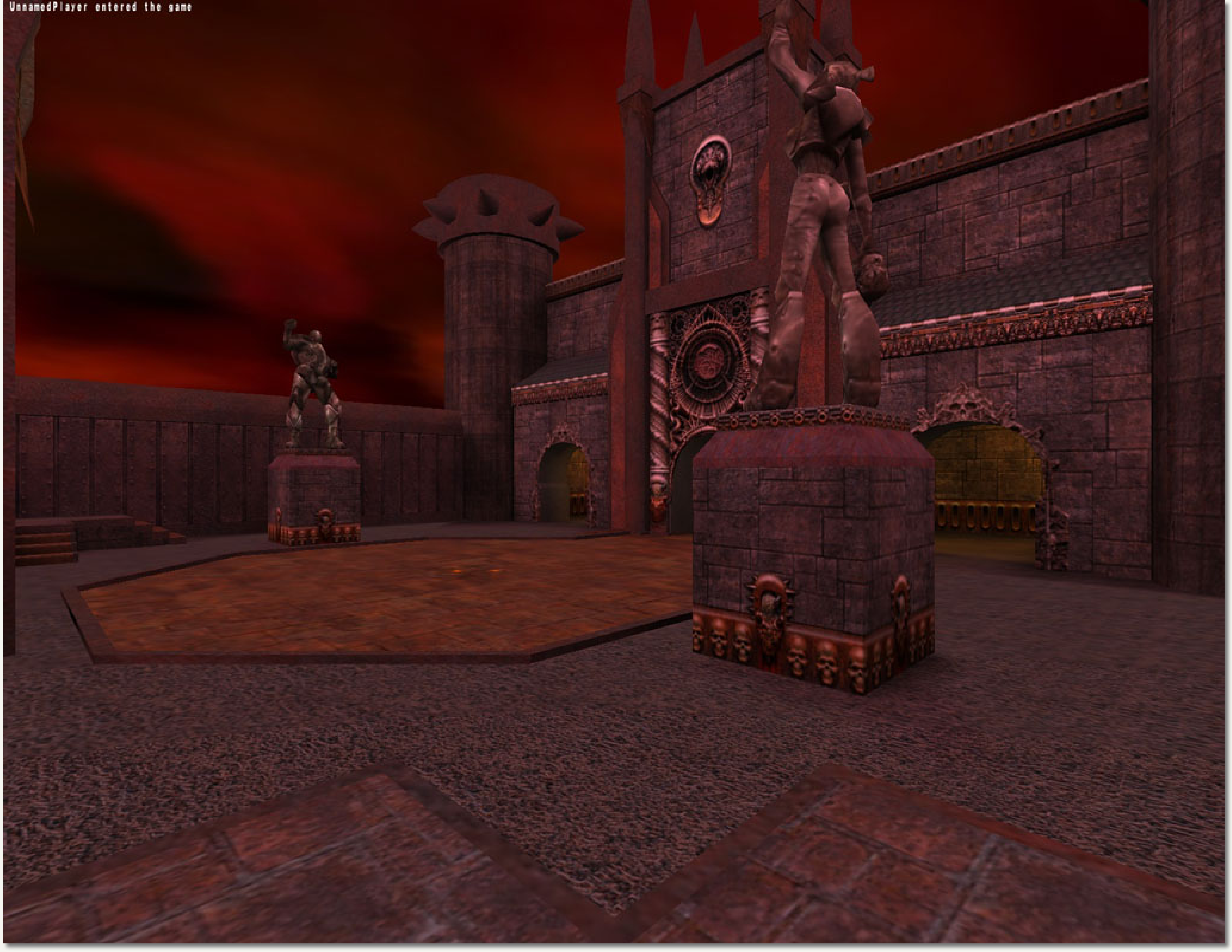
The texture drawn by the level designer/artists:



The lightmap generated by `q3light.exe` :



The final result when combined at runtime via multi-texturing :



The rendering architecture was discussed by Brian Hook at the 1999 Game Developer Conference.

Shaders

The shader system is build on top of OpenGL 1.X fixed pipeline and is hence very costy. Developers can program vertex modifications but also add texture passes. This is detailled in the Quake 3 Shader bible:

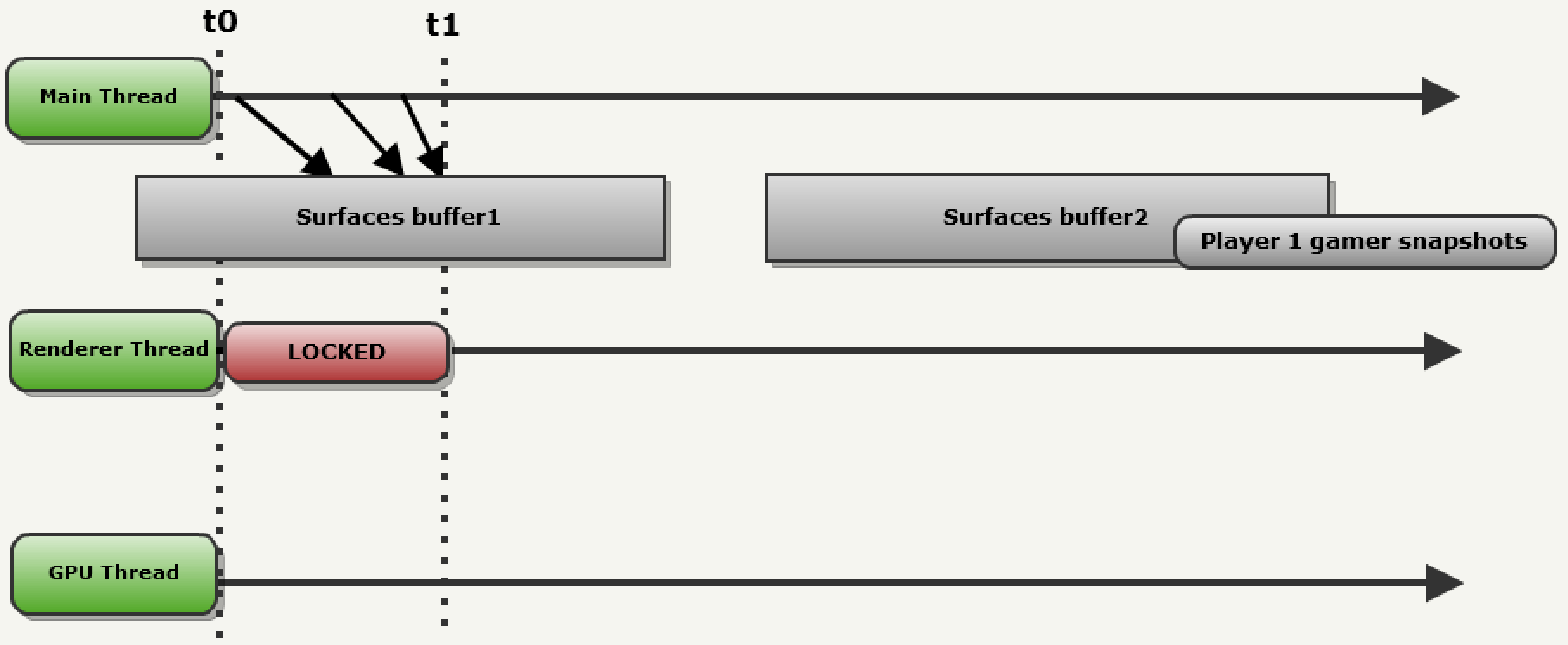
Multicore Renderer and SMP (Symmetric multiprocessing)

Unknown to a lot of people: Quake III Arena shipped with SMP support via `r_smp` cvariable. The frontend and the backend communicate via a standard Producer/Consumer design. When `r_smp` is set to 1 drawing surfaces are alternatively stored in a double buffer located in RAM. The frontend (that is called **Main thread** in this example) alternatively write to one of the buffer while the other is read by the backend (called **Renderer thread** in this example).

An example to illustrate how things are working:

From t0 to t1 :

- The Main thread decides what to draw and write surfaces to the surfacebuffer1.
- The Renderer thread is starving and hence locked.
- The GPU thread is also doing nothing.



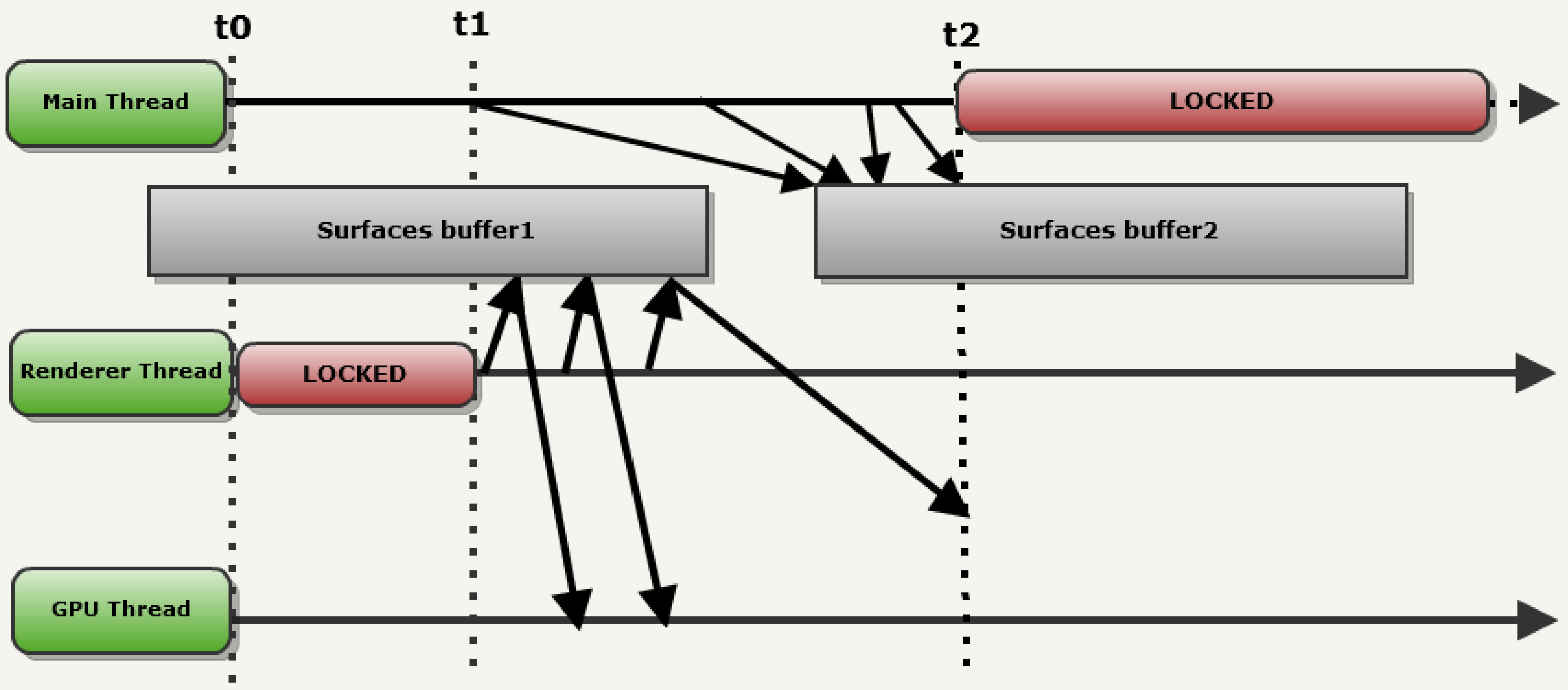
From t1 to t2 : Things start to move all over the place:

- The Main thread is deciding what is visible for next frame. It writes surface to surfacebuffer2: This is a typical example of Double Buffering.
- Meanwhile, the Renderer thread makes OpenGL call and patiently wait until the GPU thread has copied everything in a secure space.
- The GPU thread read the surface from where the Renderer thread is pointing to.

Note that at t2:

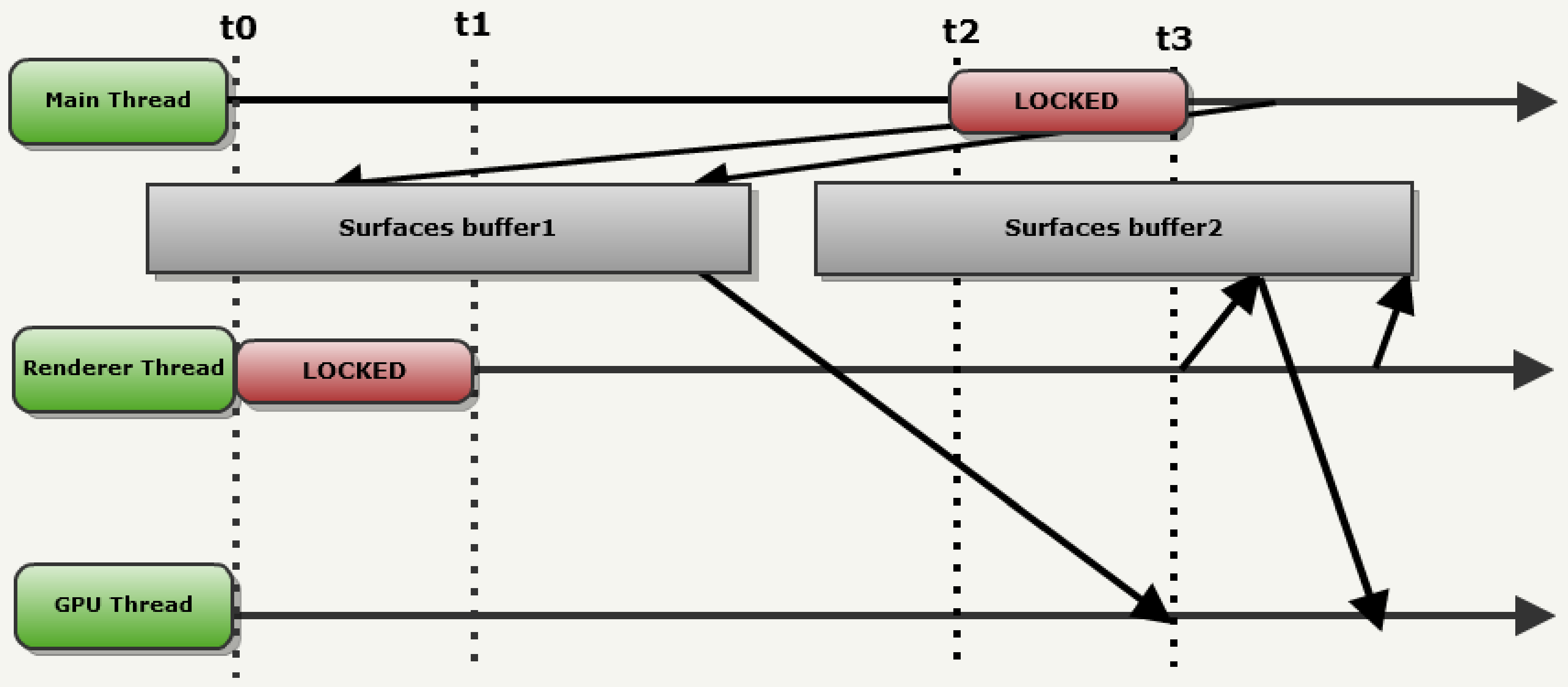
- Renderer thread is still transferring data to the GPU: SurfaceBuffer1 is in use.
- Main thread is done writing to SurfaceBuffer2...but cannot start writing to SurfaceBuffer1: It is locked

This case (where the Renderer thread is locking the Main thread is actually the most common while playing Quake III: It illustrate the blocking limitation of some of the method in the OpenGL API.



After t2 :

- As soon as the Renderer thread is done with SurfaceBuffer1 (t3) it starts pumping surfaces from SurfaceBuffer2.
- As soon as it is unlocked (at t3), the Main thread starts working on next frame, writing to SurfaceBuffer1.
- The GPU is almost never idle with this configuration.



Note : The synchronization is performed via Windows Event Objects in winglimp.c (SMP acceleration section at bottom).

Next part

The Network Model