

Hardware
- CPU
- GPU
- Memory
(- Network Card)

CPU

- process logical workflow
- managing the other hardware
- loading/releasing assets
- preparing the render thread
- has cores
- working in threads, one thread per core
- also the data is mostly handled by one thread
- is specialized on integer calculations

What can cause a CPU bound?

- heavy calculation on a cpu core
- overheating
- non optimized code, e.g. allocating a lot of memory for every iteration

When the CPU is bound, what effects does this have on our game?

- crash
- it becomes very slow
- frame drops

How can we optimize such situations?

- reduce objects, e.g. active players in a multiplayer game
- object pooling
- multithreading
- optimizing code

GPU

- process graphical data
- rendering pixels
- processing model data (e.g vertices, triangles, textures, etc.)
- has Shader Units (a lot of them, e.g 5090 has over 10k)
- works in parallel, e.g. model vertices are processed all at a time
- is specialized on floating point calculations

What can cause a GPU bound?

- heavy graphics, e.g. a lot of vertices, triangles, materials, draw calls
- too big textures
- high resolution, Full HD vs. 4K
- overdrawing (touch a pixel multiple times)
- Fullscreen Post Processing
- overheating
- Shader is not optimized

When the GPU is bound, what effects does this have on our game?

- frame drops
- application is not starting or is crashing immediatly
- sometimes frames artifacts

How can we optimize such situations?

- use LODs or reduce mesh complexity in general (LOD bias)
- use smaller textures
- use less materials
- optimize shader code

Hardware

- CPU
- GPU
- Memory
- (- Network Card)

Memory

- cpu memory vs. video memory

What happen if we are out of memory depending on cpu or gpu?

- out of memory on cpu
 - most times the app is crashing
 - sometimes it become very slow because of swapping
- out of memory on gpu
 - pixel artifacts
 - big bottleneck because, the gpu has swap assets very often
- you also have to identify where we are out of memory

Draw Calls

- every operation on the gpu or every command executed on the gpu

Time vs. Frames Per Second (fps)

- fps is very volatile & unprecise
- fps can be used as an indicator
- you should always look at average fps, but also for 1% & 0.1%
- time is better for optimizing
- time can be meseasured for every calculation, e.g. frames, methods, blocks
- always optimize time instead of frames

Memory Allocation

- try to move the allocation the beginning, e.g. start of the game or start of the level
- also deallocation takes time, you don't when the memory is deallocated (Garbage Collector)
- use object pooling instead of (de-)allocating objects
- try to use Unity methods with preallocated data
- try to avoid using methods which are calling GetEnumerator, e.g. foreach or LINQ

Loading Times

- making a loading screen more enjoyable by
 - some useful informations
 - making it dynamic, e.g. with a loading bar or loading corridors
- load scenes async
- try to load your gameobjects in chunks
- try to load your asset in batches with AssetBundles and/or Addressables