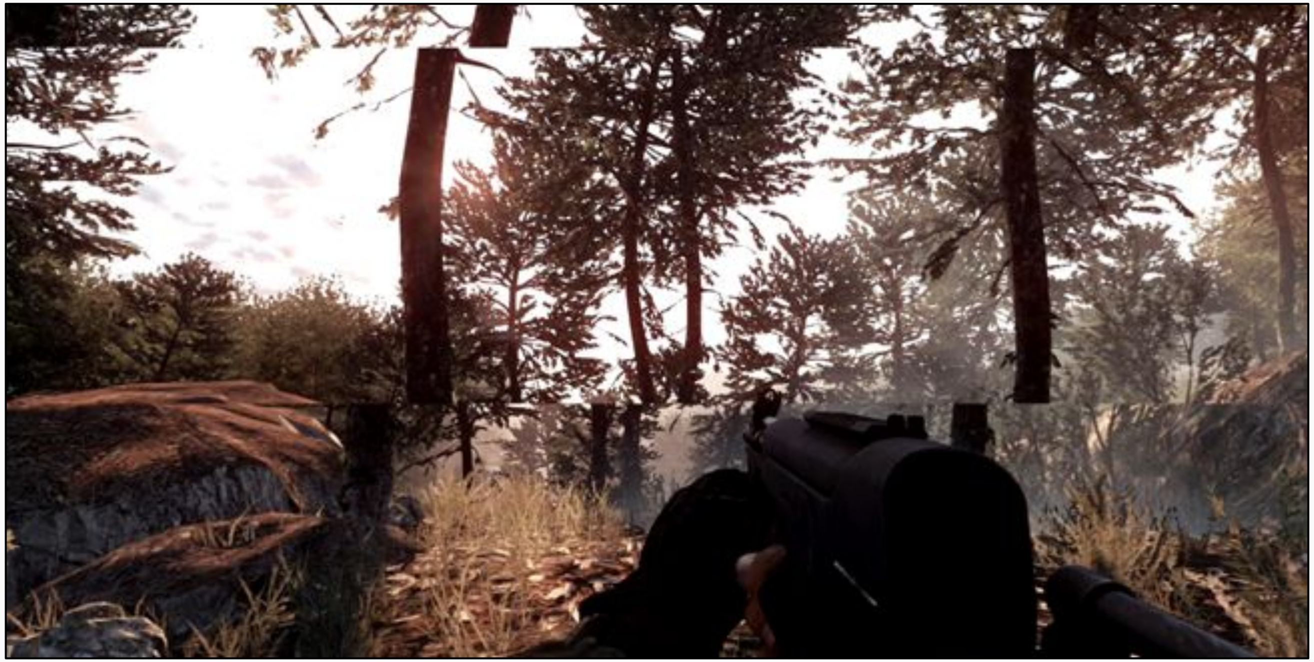
Limits, ticks and dt

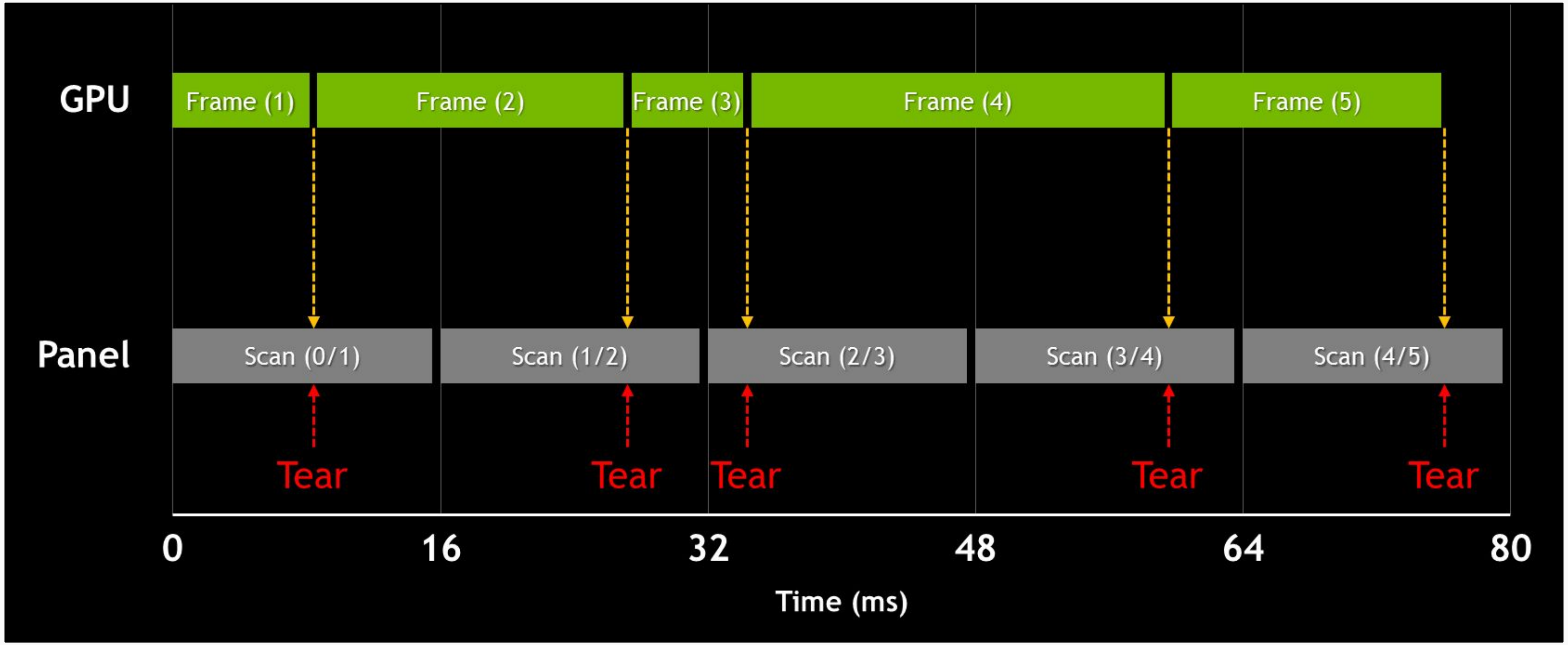
Game Development

Back in the days ...

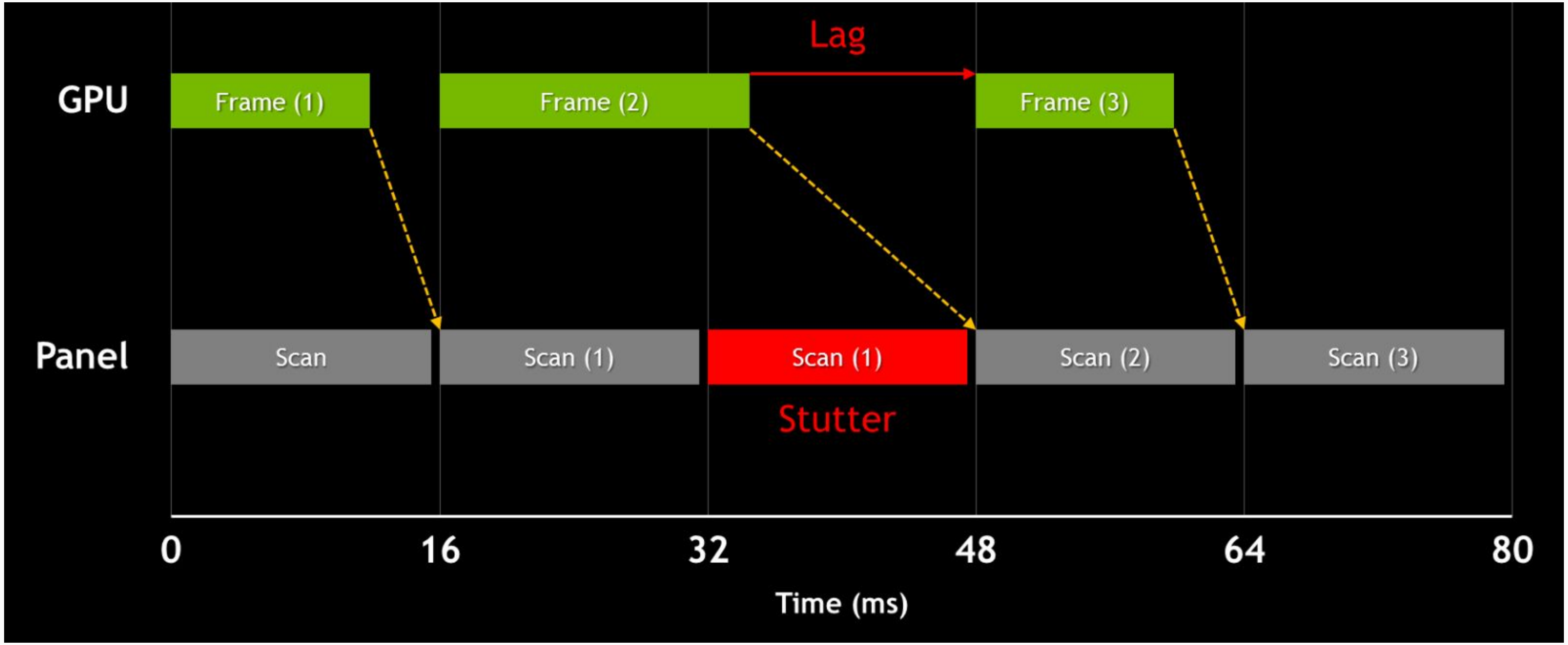


How monitors work

Tearing in action



Vsync in action



Why do we limit the framerate ?

- We want a smooth experience
- We can hide peaks of slow code
- The code becomes more predictable
- Variable time step vs Fixed time step
- My take: a mix of both: limit framerate but use variable time step
- Actually frame rate ALWAYS fluctuates a bit :(

How do we wait

- Every time we wait we give that time back to other OS processes
- It is a bit unpredictable when waiting will finish
- Two main ways:
  - Vsync: it will wait until screen refreshes (very accurate)
    - *“With double-buffered VSync, the framerate can only be equal to a discrete set of values equal to Refresh / N where N is some positive integer. 60Hz: 60, 30, 20, 15, 12, 10, etc.”*
  - SDL_Delay: stop and give time back to the OS (not so accurate)
- Understand that SDL2 automatically uses double buffering

TODO 1

“Read from config file your framerate cap”

- Simply add the variable as an integer in the format “12” or “28”
- This variable is the *new maximum framerate*
- Mind that we could not have that value specified
- That would mean that no cap is going to happen

TODO 2

“Use SDL_Delay to make sure you get your capped framerate”

- You already have current ms in a variable
- You can calculate the amount of milliseconds you need to wait

CAREFUL*: debugging time lapses could be difficult. Everytime we stop as a break point, the time will be increasing creating huge delays.*

TODO 3

“Measure accurately the amount of time SDL_Delay actually waits compared to what was expected”

- Use a j1PerfTimer
- Print your result with LOG
- Test and check the results

You should have something like that ...

```
We waited for 29 milliseconds and got back in 28.845800
We waited for 29 milliseconds and got back in 28.687000
We waited for 32 milliseconds and got back in 32.058500
We waited for 30 milliseconds and got back in 29.791500
We waited for 32 milliseconds and got back in 36.402700
We waited for 31 milliseconds and got back in 31.195300
We waited for 32 milliseconds and got back in 32.048200
We waited for 32 milliseconds and got back in 32.142200
We waited for 32 milliseconds and got back in 32.217400
We waited for 32 milliseconds and got back in 32.125800
We waited for 32 milliseconds and got back in 31.348000
We waited for 31 milliseconds and got back in 30.200800
We waited for 32 milliseconds and got back in 31.605700
We waited for 32 milliseconds and got back in 32.154500
```

TODO 4

*“Calculate the **dt**: differential time since last frame”*

- We use a float with the amount of seconds since last frame
- We will use it a as a multiplier

TODO 5

*"Send **dt** as an argument to all updates you will need to update the module parent class and all modules that use update"*

- Potentially any Module could use it
- So we distribute to everybody :)
- Actually just SceneModule has an update

TODO 6

“Make the camera movement independent of framerate”

- Most likely won’t move if you keep 1 as the speed (try 100+)
- Note that if you press left and right at the same time there is still some movement! This is due to float precision problems
- Use ceil/floor to solve that
- Try capping to 10 30 and 60 and notice that the speed movement is now stable, independent from frame rate

Documentation

- Game loops can become quite complex. Read carefully this article.
- GPU manufacturers are constantly trying to reinvent the vsync:
  - NVidia’s adaptive vsync and G-Sync
  - AMD enhanced vsync

Homework

- Create a module player that only gets updated 10 times per second
- Use a different method than Update, like UpdateTick()
- We still want Update for graphics!
- But **decisions** can be taken safely 10 times per second