

# Game dev: Framerate

Ricard Pillosu - UPC



# The past

- Computer graphics was late (1980) and initially for the business:
- [Silicon Graphics](#)
- Later they founded Pixar :)



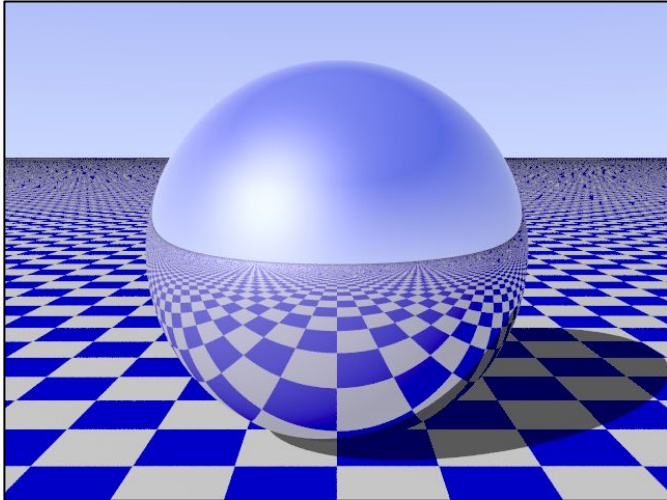
# The past

- Home computers with multimedia was way later (1985):
- [Commodore Amiga](#)
- It features graphical multitasking OS
- ... in 1.4 Mb!



# The past

- The PC took market advantage (1992). Still rendering was expensive!



# The past

- The revolution of the GPUs (1995):

## 1995

NVIDIA LAUNCHES ITS FIRST PRODUCT, NV1



NVIDIA NV1, the company's first product, is launched. The PCI card was sold as the Diamond Edge 3D, featuring a 2D/3D graphics core based on quadratic texture mapping.

Sega, the leader in arcade games, ports Virtua Fighter to be the first 3D game to run on NVIDIA graphics.



# 3dfx Voodoo!

More info on history of real time graphics on video games [here](#).



# Real Time applications

- Rendering-computation fast enough, so that the series of rendered images induce the illusion of movement in the human brain of the user.
- This illusion allows for the interaction with the software doing the calculations taking into account user input.
- The unit used for measuring the frame rate in a series of images is frames per second (fps).

Lot's of wisdom here!





# Frame rate

- We exploit a limitation in the Visual Cortex called [Persistence of Vision](#)
- This optical illusion is created when images cycle of less than 60-80 ms
- This means that motion is perceived starting at 12-16 fps
- Movie industry uses the 24 fps standard for [historical reasons](#)
- Frame rate also applies to sound and input!
- Humans are better at detecting low sound framerate.

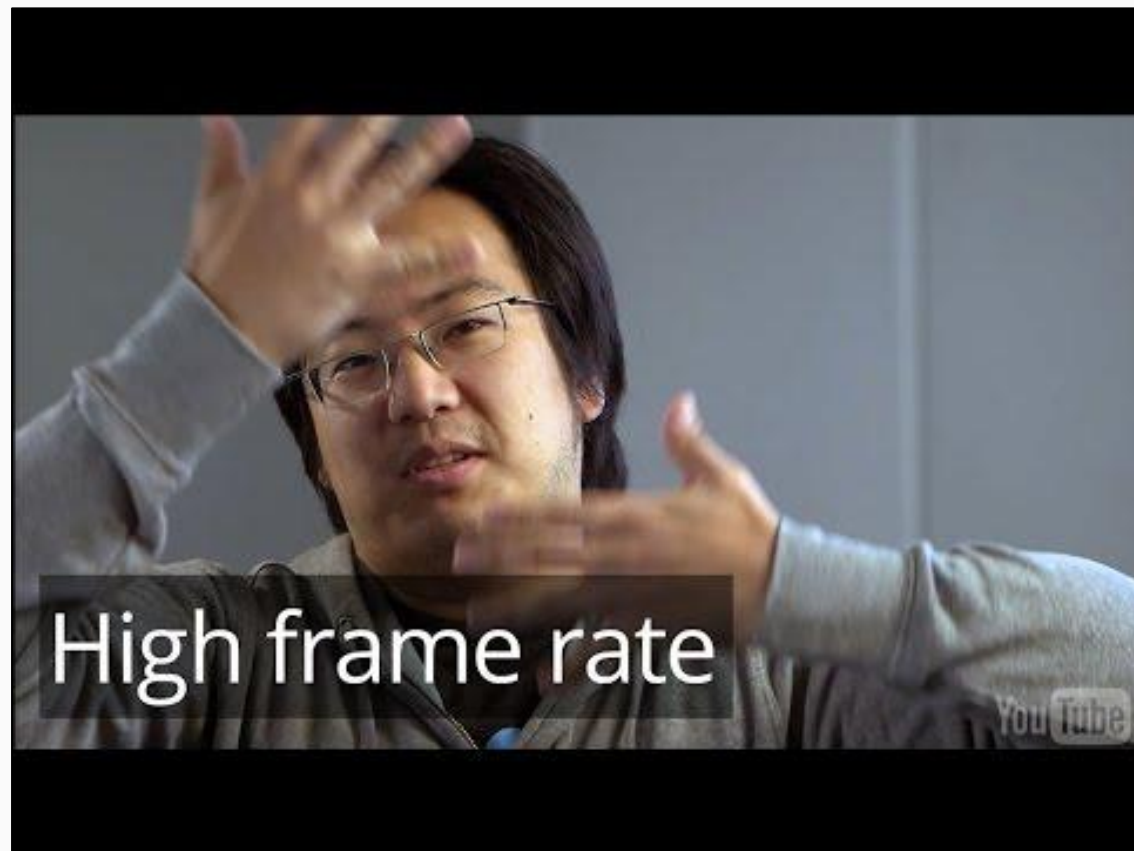
Movement is just an illusion: [30vs60.com](http://30vs60.com)



Movement is just an illusion: [30vs60.com](http://30vs60.com)



Movement is just an illusion: [30vs60.com](http://30vs60.com)



# How do we measure time ?

- We want accurate measurement to the [microsecond \( \$\mu\$ s\)](#)
- This is a [not fully solved problem](#)! ... measuring time takes time :(
- Back in the days, using [rdtsc](#) intrinsic was standard in [windows](#)
- Microsoft discourages it since:
  - We have multiple CPUs sending tasks to each other
  - We have complex power-saving systems that could hibernate CPUs
  - We have [out-of-order](#) execution CPUs

# How do we measure time ?

- Solution came from two functions (windows platform):
  - [QueryPerformanceCounter](#): Returns a high precision timestamp. Only make sense relative to other previous measure.
  - [QueryPerformanceFrequency](#): Returns the frequency of the counter. We can transform intervals to human readable time. It should be constant during execution.
- SDL offers [platform independant versions](#) of this functions.
- Also a simple [SDL\\_GetTicks](#) wich is faster but less accurate (1ms)
  - Enough for **most** profiling

# TODO 1

*“Fill Start() and Read() methods they are simple, one line each!”*

- We will use this class for logic, we do not need precision
- This means that **SDL\_GetTicks()** is enough

# TODO 2

*“Fill Constructor, Start() and Read() methods they are simple, one line each!”*

- **j1PerfTimer** will be for precise measurements of code execution (profiling)
- Frequency is the same across all timers, that's why we use a static
- This means that we need SDL\_GetPerformanceCounter() + SDL\_GetPerformanceFrequency()



# TODO 3

```
ip(26) : Create SDL rendering context  
.05) : j1App::Awake took 981.652598 ms  
.66) : START PHASE =====
```

*“Measure the amount of microseconds that takes to execute: App constructor, Awake, Start and CleanUp. LOG the results”*

- Create a timer in j1App as a property
- Then use it to print in output window (LOG) the amount of ms that each method takes
- Try to guess the times before starting, which will be slowest ?

# TODO 4

 Av.FPS: 129.01 Last Frame Ms: 15 Last sec frames: 112 Last dt: 0.017 Time since startup: 169.466 Frame Count: 21862

*“Now calculate: Amount of frames since startup. Amount of time since game start (use a low resolution timer). Average FPS for the whole game life. Amount of ms took the last update. Amount of frames during the last second”*

- You may need to add code in **PrepareUpdate()** method
- Create as many properties you need in **App** class
- When it works try activating / deactivating vsync
- Check [printf format](#) for zero padding and other cool stuff

# Homework

- Measure Init() and Start() microseconds for each module
- Try producing lag to drop to 30 FPS under vsync