



+

JUST FOOTBALL

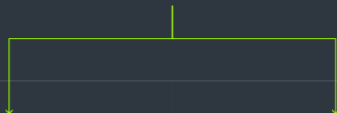


GOALS



FOR THE PLAYER

+



+

GAMEPLAY FIRST

| FREE TO PLAY

| CROSS PLATFORM

|

GOALS FTP

GOALS FTP

GOALS FTP

GOALS FTP

GOALS FTP

GOALS FTP

GOALS FTP

GOALS FTP

GOALS



FTP

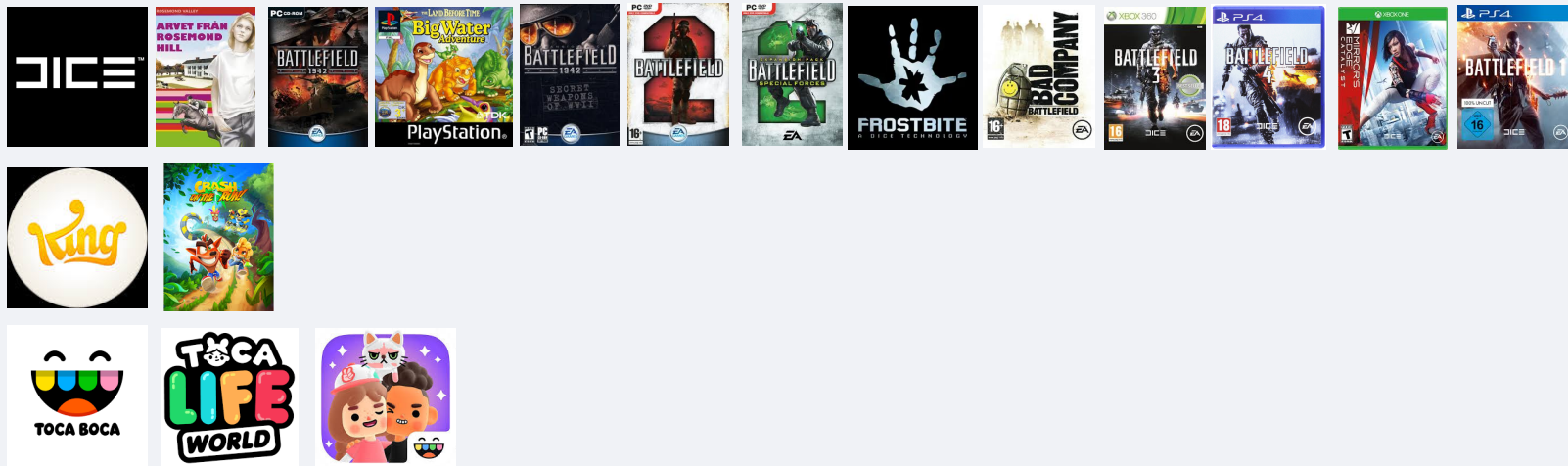
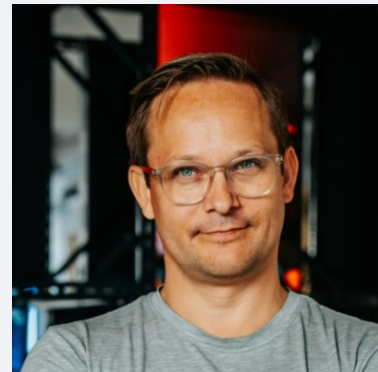
GLS

SPLIT FOR SPEED



Scalable Rendering with Smart Build Delivery and GPU-Aware Throttling in
UE5

Torbjörn Söderman



Luisb

1



FAST-PACED // COMPETITIVE // FREE TO PLAY

THE ESPORTS-READY FOOTBALL GAME



PC



XBOX



PLAYSTATION



52:10

Luis	0
龜RockmanJ龜	4



VAN DER LAAN

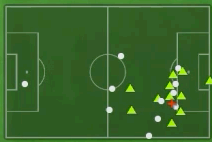
HERRERA

85	PAC	90	DRI
61	SHO	80	DEF
72	PAS	67	PHY

01
WB

HERRERA

11



94	PAC	86	DRI
76	SHO	42	DEF
90	PAS	66	PHY

11

TOSKA

85
WF

The premise



GOALS



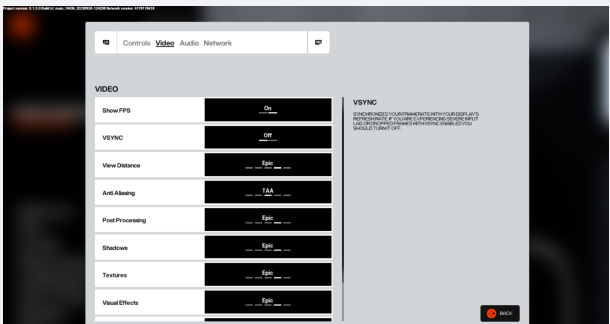
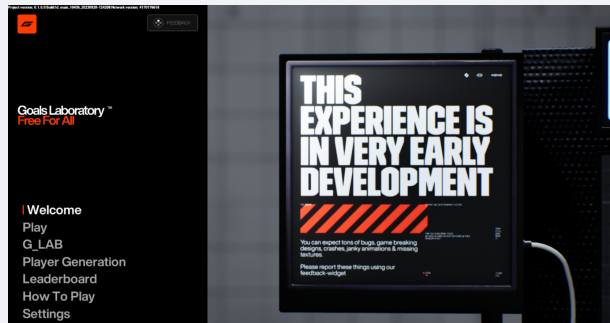
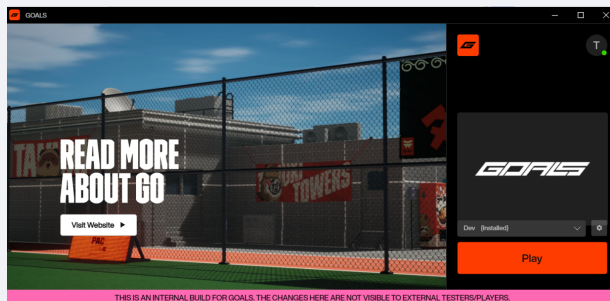
EVERYONE'S
GAME



“EVERYONE’S GAME”



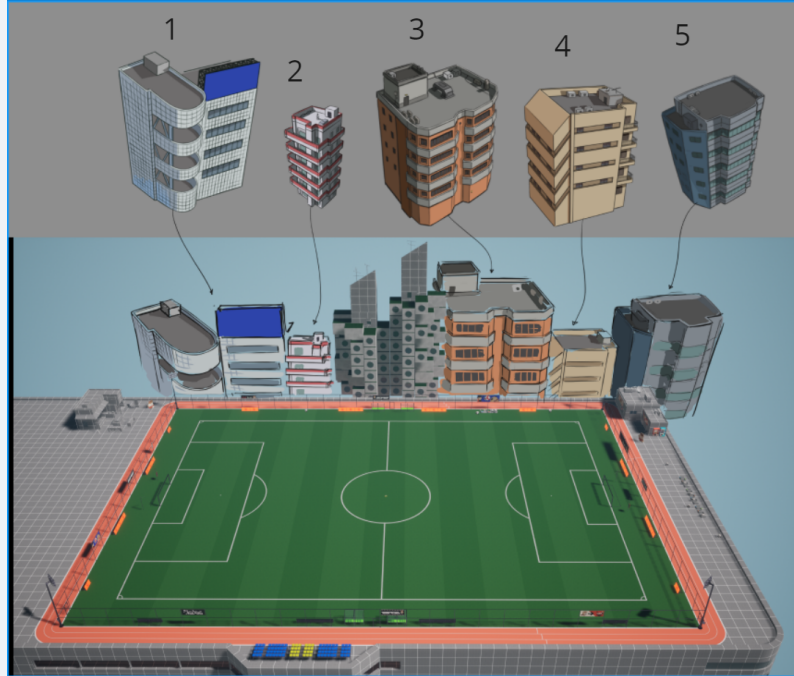
2023 Q3



Graphics Programming Conference, November 18-20, Breda

2025

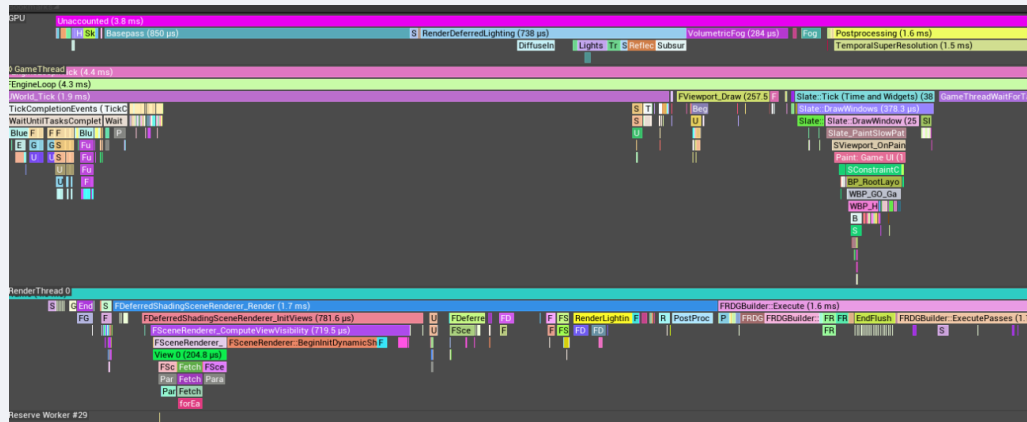
Determining min-spec (Lighting)



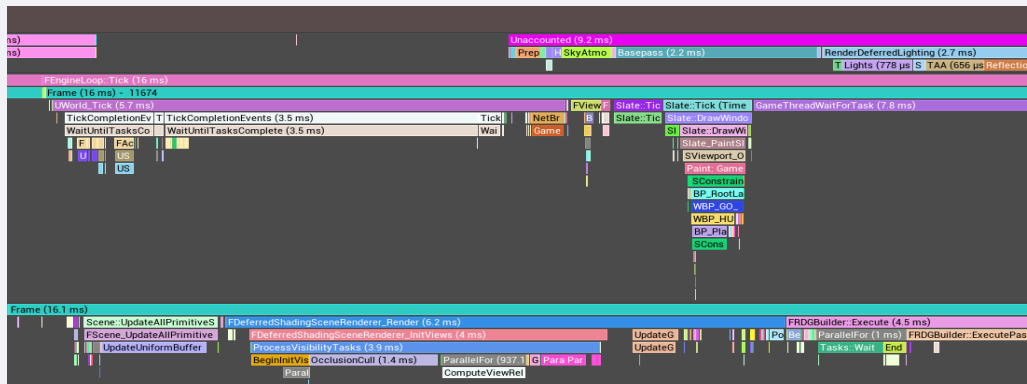
Esport focus - “Everyone’s game”

- Decided to use baked lighting
- Unreal Lightmass for environment
- Lightprobes for characters
 - Unreal Indirect Lighting Cache
- Custom lighting on close-ups and in menus

Determining min-spec (Profiling)



200 FPS



60 FPS



Determining min-spec (Profiling takeaways)

Intel Celeron G5905T, Intel UHD Graphics 610

AMD Ryzen 9 7950X, NVIDIA GeForce RTX 4070 Ti

Forward is 30% faster

Deferred not a real issue

Deferred is a memory hog

Can push visuals and still reach high FPS

Memory will be a limiting factor

Memory is not a limiting factor

High end settings should be hidden

Everyone's game needs deferred shading

Cannot have both deferred and forward



A game for everyone needs both forward and deferred



Determining min-spec (Lowest common denominator)



Acer Veriton N4680GT
CPU: Intel(R) Celeron(R) G5905T (2 CPUs), ~3.3GHz
GPU: Intel(R) UHD Graphics 610
RAM: 4096MB RAM



Amd Am5
CPU: AMD Ryzen 9 7950X (32 CPUs), ~4.5GHz
GPU: NVIDIA GeForce RTX 4070 Ti
RAM: 65536MB RAM

Classic approach

- HW Benchmark
- Low, Medium, High

Need to draw the line on what HW and rendering features to support



“You can’t have your
cake and eat it too”

Or can you?

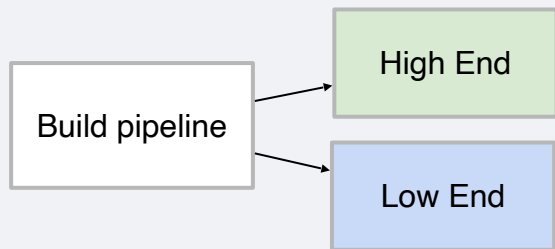
What if you could have the best of two worlds...

No lowest common denominator

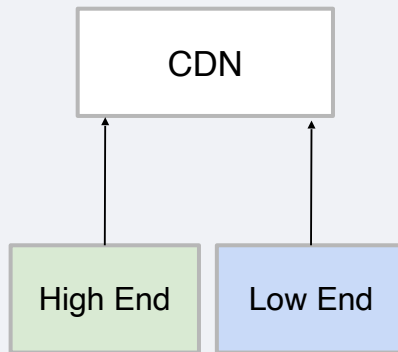
->Split the problem in 2 versions

Eat the cake and have it too!

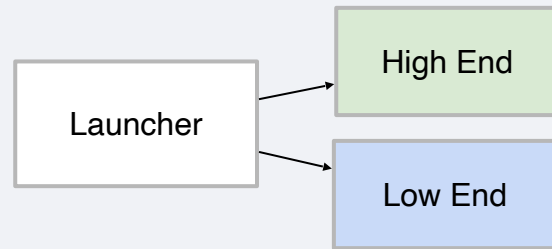
Split content build pipeline



Split package distribution



Split execution



The present



Launcher



Headless launcher

Detect HW

Download

Install

Launch the game

Solves the problem for games with very wide scalability in performance. Lowend to support:

- Dual core celeron
- Integrated GPU
- 30 FPS

Highend version (Player Experience)

- Doesn't have to compromise with lowest common denominator.
- Can use much of the Unreal bling and high end extensions

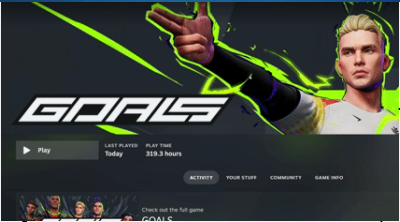
Lowend version (Player Experience)

- Forward shading ~30% faster
- Specific optimization for low end
- Faster load times
- Smaller download size

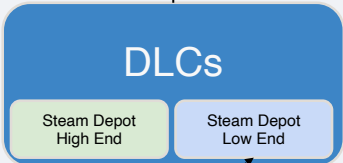
Reach (Profit/Power)

- Being able to play the game on less powerful HW increases the reach by a lot.
- Steam binary patches keeps download size to a minimum

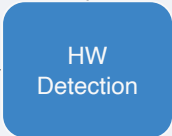




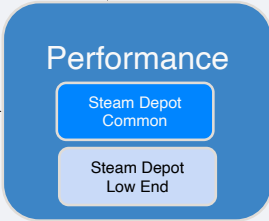
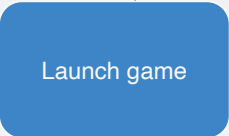
Headless launcher



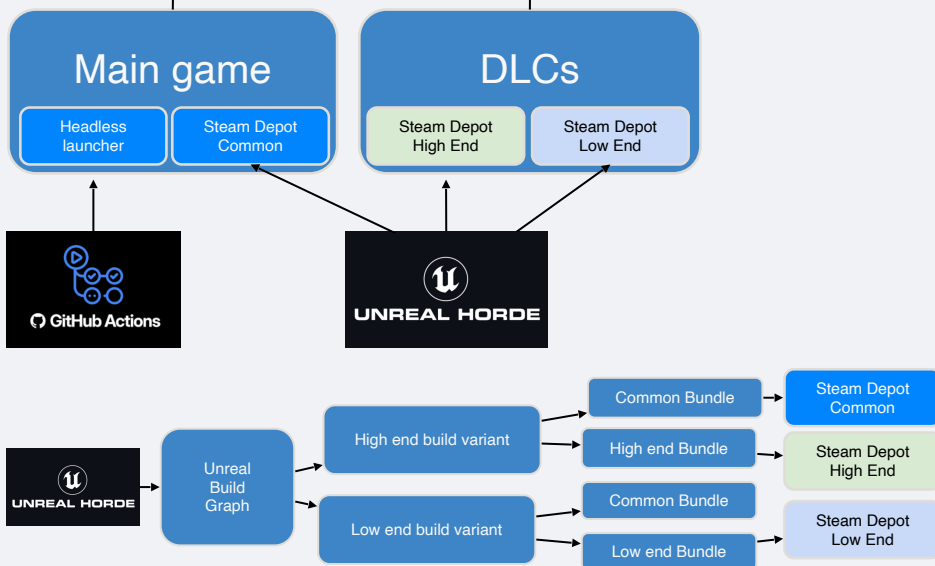
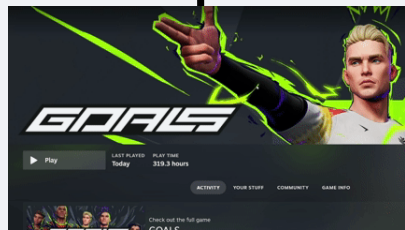
Steam Depot High End



Steam Depot Low End



Infra deep dive



Steam DLC Setup

Build pipeline

- Divergence, Unreal Build Tool (UBT) knows about low end variant.
- Build & Cook 2 builds
 - High end version and Low end version
- Have a list of classes and splits them into chunks
 - 0 is the common one
 - 1..4 is either high end or low end DLC bundle
- Copy common bundle from High end build
- Have 2 executables, one in each DLC.

Build size/Patching

- Size is around 2.6/2.7 GB combined Common+DLC
- DLC: Low end = 277 MB , High end = 346 MB
- Delta is based on Steam binary patching
- Biggest patch was 1 Gb, new level added
- Regular patch size is 200Mb
- Using chunks -> smaller delta



Headless Launcher



Launcher App

- C# Executable
- Started from Steam
- Does HW detection based on
 - GPU,
 - CPU,
 - Memory
 - CPU/GPU feature level
- Calculates score
- Determines if the HW should run the Performance or Experience version
- Initiates download of the correct DLC bundle
- Launches the game with the correct cmd-line

Experience/High End

Deferred shading

D3D12

- FL_12_0
- Shader Model 6.0
- Feature level:
 - Resource Binding Tier 3+
 - Type UAV Load Additional Formats
 - Wave Ops
 - Atomic64



GENERAL

CONTROLS

VIDEO

AUDIO

NETWORK

ACCOUNT

GENERAL

SHOW FPS

ON

VSYNC

OFF

RENDERING QUALITY

EPIC

ANTI ALIASING

DLSS

RESOLUTION SCALE

DLAA

THERMAL SETTINGS

THERMAL MODE

OFF

ACCESSIBILITY

COLOR VISION DEFICIENCY TYPE

NORMAL VISION

ANTI ALIASING

OFF - NO ANTI-ALIASING

FXAA - FAST APPROXIMATE ANTI-ALIASING

TAA - TEMPORAL ANTI-ALIASING

TSR - TEMPORAL SUPER-RESOLUTION

XESS - INTEL SUPER RESOLUTION

FSR3 - AMD SUPER RESOLUTION

DLSS - NVIDIA SUPER SAMPLING

Performance/Low End



Forward Shading

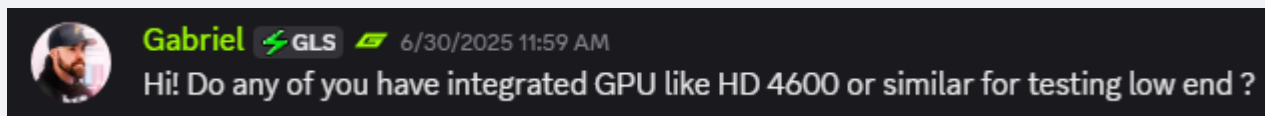
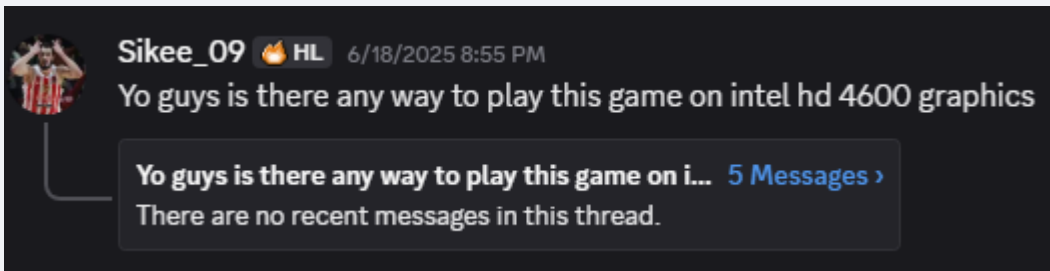
- Performance
 - D3D12
 - FL_11_0
 - Shader Model 5.1
- Performance11
 - D3D11
 - FL_11_0
 - Shader Model 5.0
- SuperLowEnd
 - ES31
 - Mobile rendering
- Reduced feature set
 - No AVX

Performance/Low End

“Half-platform” variant

```
// GOALS-DIVERGENCE - laedre - 66-14894 - Add engine 'half-platform' for LowEnd variants - BEGIN
//
// This allows for the '-GoalsLowEnd' argument to be used consistently across all build, cook and runs including boths client types and the editor.
//
#if defined(GOALS_LOW_END_BUILD) && GOALS_LOW_END_BUILD
    static const bool bGoalsSuperLowEndParam = FParse::Param(FCommandLine::Get(), TEXT("GoalsSuperLowEnd")) || FParse::Param(FCommandLine::Get(), TEXT("extraLow"));
    if (bGoalsSuperLowEndParam)
    {
        ProfileName += TEXT("SuperLowEnd");
        UE_LOG(LogTemp, Log, TEXT("Windows Device Profile: '%s' selected from a combination of compile-time 'GOALS_LOW_END_BUILD' and command-line argument '-GoalsSuperLowEnd'."), *ProfileName);
    }
else
{
    ProfileName += TEXT("LowEnd");
    UE_LOG(LogTemp, Log, TEXT("Windows Device Profile: '%s' selected from compile-time 'GOALS_LOW_END_BUILD'."), *ProfileName);
}
#else
    static const bool bGoalsLowEndParam = FParse::Param(FCommandLine::Get(), TEXT("GoalsLowEnd"));
    if (bGoalsLowEndParam)
    {
        static const bool bGoalsSuperLowEndParam = FParse::Param(FCommandLine::Get(), TEXT("GoalsSuperLowEnd")) || FParse::Param(FCommandLine::Get(), TEXT("extraLow"));
        if (bGoalsSuperLowEndParam)
        {
            ProfileName += TEXT("SuperLowEnd");
            UE_LOG(LogTemp, Log, TEXT("Windows Device Profile: '%s' selected from command-line arguments '-GoalsLowEnd' and '-GoalsSuperLowEnd' passed to editor build."), *ProfileName);
        }
        else
        {
            ProfileName += TEXT("LowEnd");
            UE_LOG(LogTemp, Log, TEXT("Windows Device Profile: '%s' selected from command-line arguments '-GoalsLowEnd' and '-GoalsSuperLowEnd' passed to game build."), *ProfileName);
        }
    }
#endif
```

Playtesting with the community

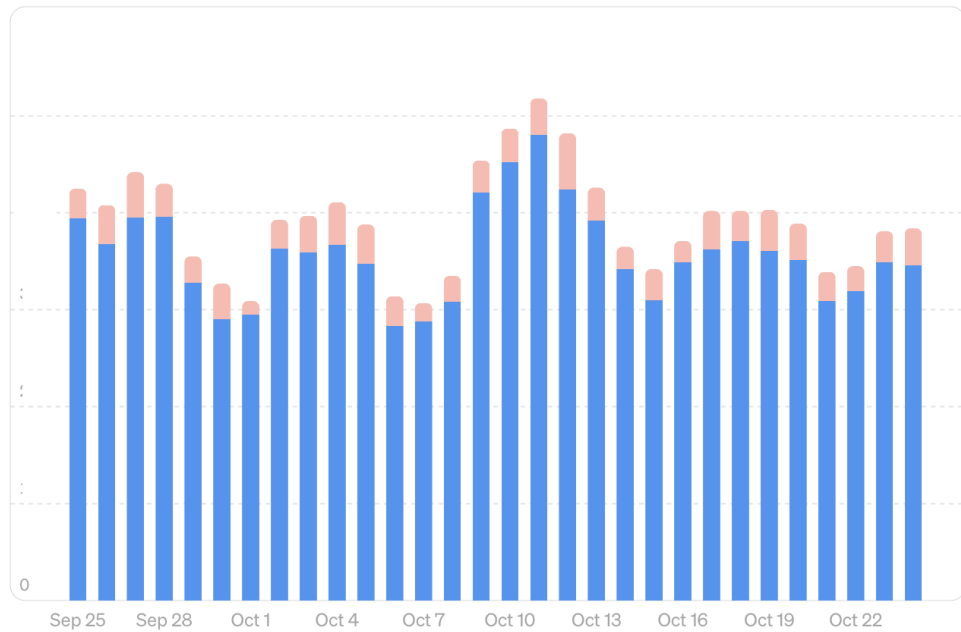


Is it worth the investment?

Active Users Game

Last 30 days

● Windows High-end ● Windows Low-end



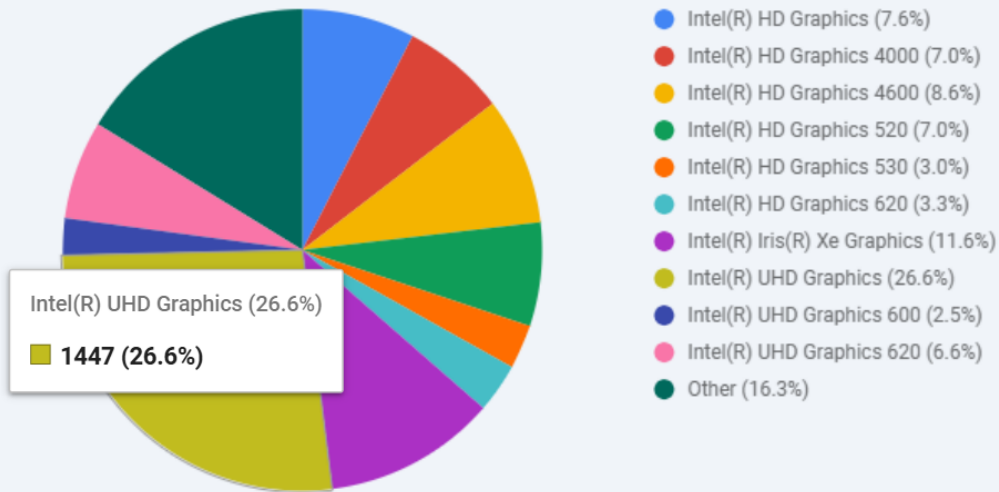
Everyone's game

- >9% low end
- F2P game
- ~30% Low end by launch
- Optimizations will improve everything



What is the current distribution

Top 10 GPUs by Logins with 'Other' Category



A mixed bag

- Really old (2010) 6.2%
 - Sandy bridge
- Old architecture (2012) 24,7%
 - Ivy bridge
 - Haswell, Broadwell
- It's been a while (2015) 15,8%
 - Sky lake
 - Kaby lake
- Before Corona (2017) 22,7%
 - Coffee Lake
 - Ice Lake, Comet Lake
- Corona (2020) 19,8%
 - Tiger Lake
 - Alder Lake
- Post Corona (2022+) 10.8%
 - Raptor Lake
 - Meteor Lake

But we need more low end HW

“A 3.3 Ghz Celeron isn't really low end!”



Acer Aspire Lite 15

CPU: Intel(R) Celeron(R) N4500 (2 CPUs), ~1.1GHz
GPU: Intel(R) UHD Graphics
RAM: 4096MB RAM



Lenovo IdeaPad 1

CPU: AMD 3020e (2 CPUs), ~1.2GHz
GPU: AMD Radeon(TM) Vega 3 Graphics
RAM: 4096MB RAM



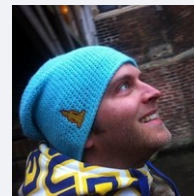
Lenovo G500

CPU: Intel(R) Core(TM) i5-3230M (4 CPUs), ~2.6GHz
GPU: Intel(R) HD Graphics 4000
RAM: 4096MB RAM



Narrow Box Ultra HD Compact v5

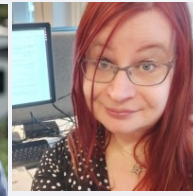
CPU: Intel(R) Celeron(R) N5100 (4 CPUs), ~1.1GHz
GPU: Intel(R) UHD Graphics
RAM: 4096MB RAM



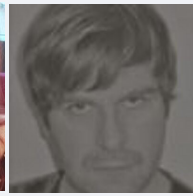
Mario



David

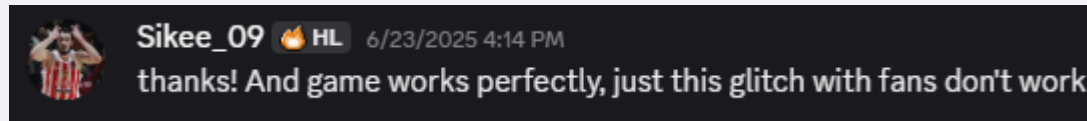
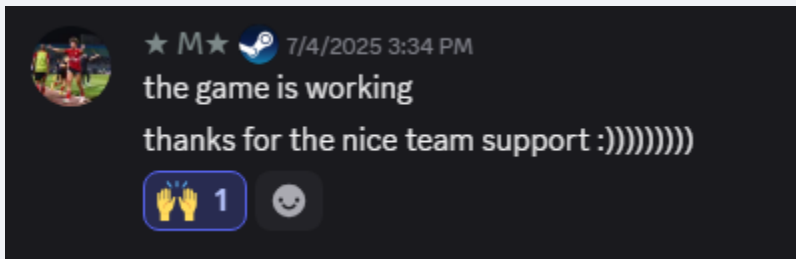


Sofie



Torbjörn

Playtesting with the community



SuperLowEnd



Lenovo G500

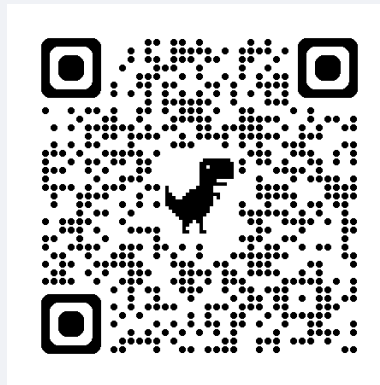
CPU: Intel(R) Core(TM) i5-3230M (4HT CPUs), ~2.6GHz

GPU: Intel(R) HD Graphics 4000

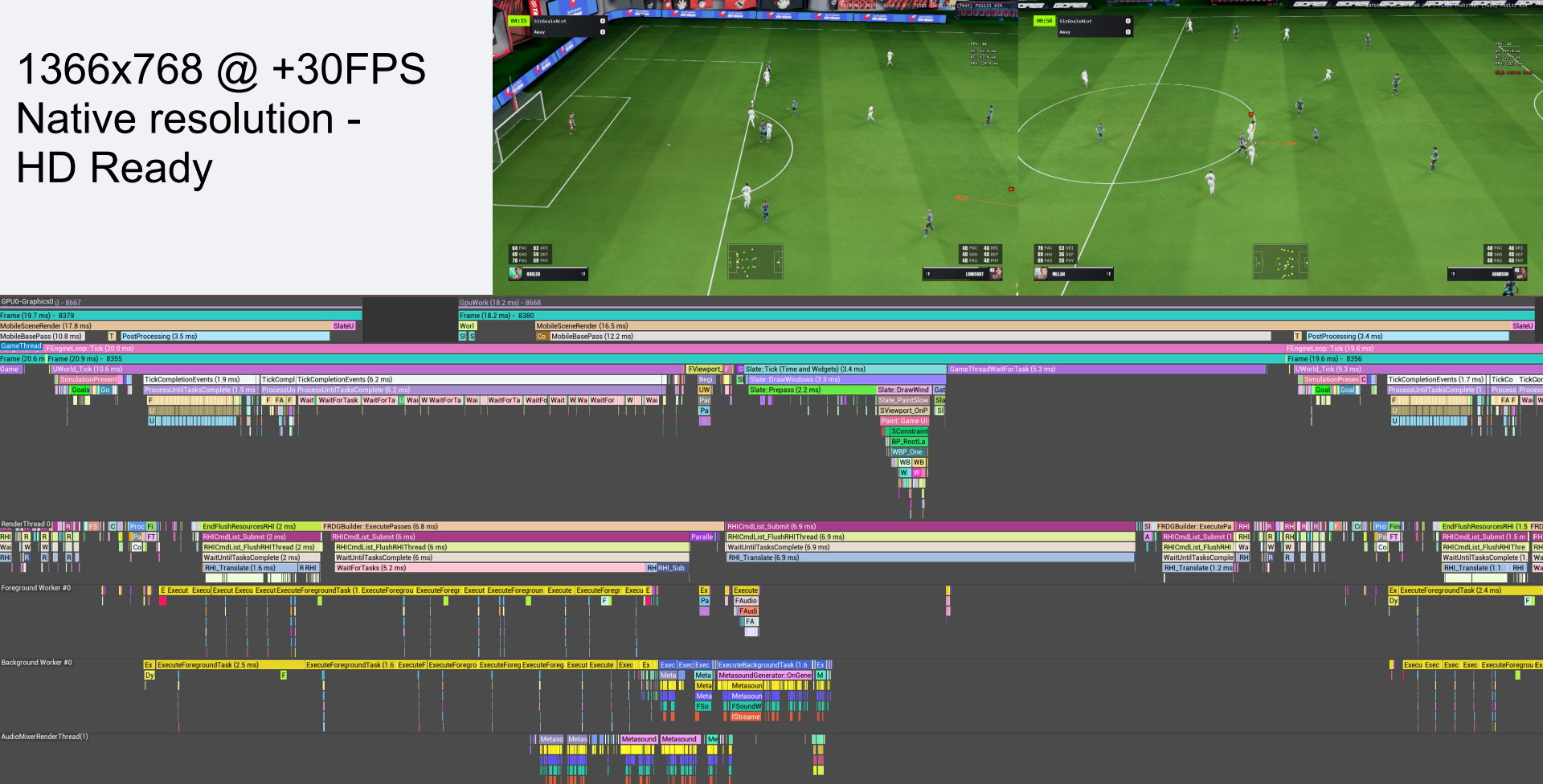
RAM: 4096MB RAM

ES31 Rendering path

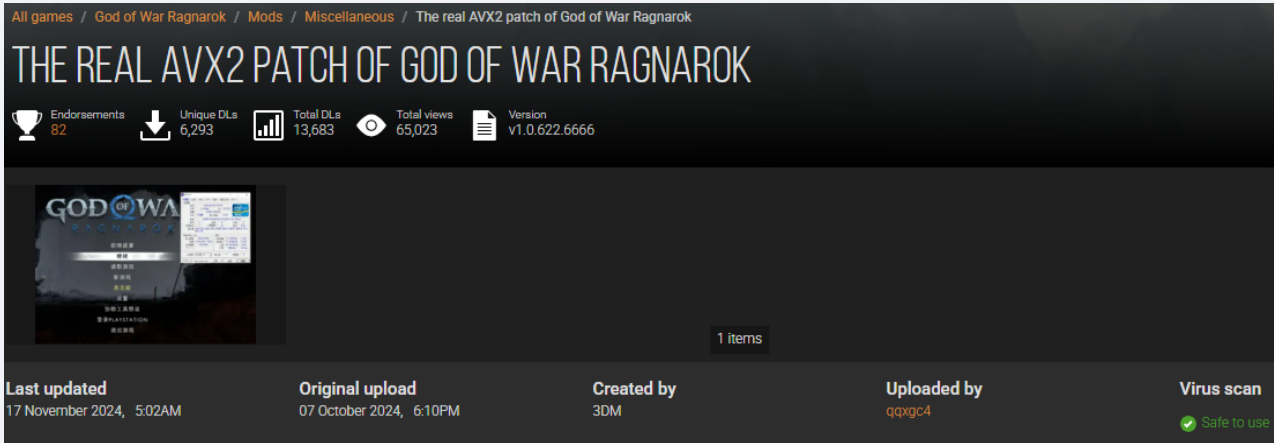
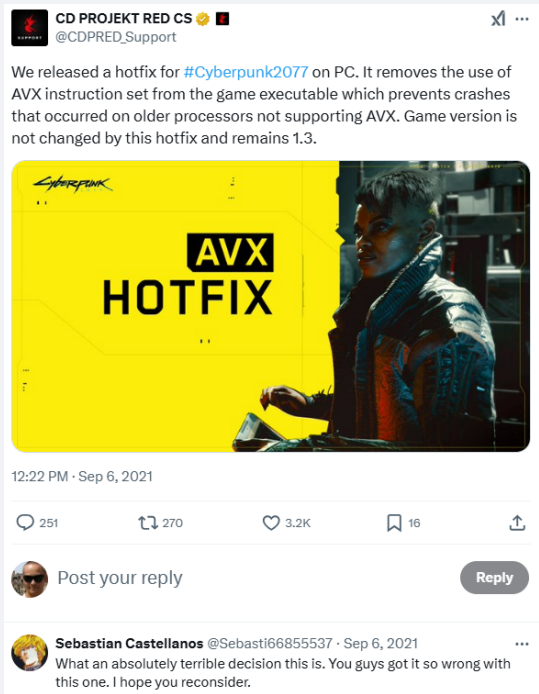
- Emulated mobile forward shading
- 4-6 ms/frame faster
- Pushing potato +30 FPS
- Thanks to Epic for the tip



1366x768 @ +30FPS
Native resolution -
HD Ready



Pros: With 2 builds



Autodetection downloads
the correct version of the
game



Future work



More platforms



But what happens
when you have
optimized so much
that you almost have
100% resource
utilization?

THROTTLING

How you control the temperature of your gaming device



Background



Why should we care?

- Player Experience
- Cost of power
- CO2 output

The higher the power usage the hotter the device gets

- Highend Desktops uses 200-500W on average for gaming
- Laptops uses on average 30-70W
- As a comparison mobile phone systems uses 3-6W



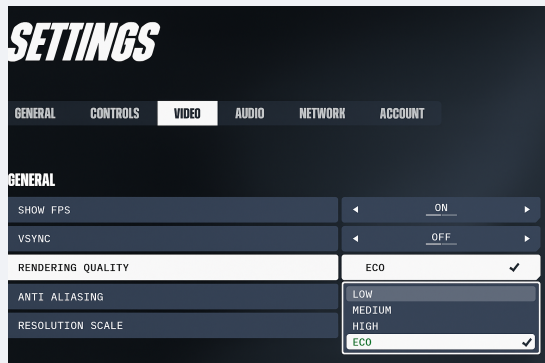
3 ways to throttle/reduce heat



- Reduce shader complexity
- Reduce bandwidth
- Reduce FPS



Reduce shader complexity



- Implement scalability settings in shaders
- Make it easy to change between modes
- HW benchmarks good quality level baseline
- Introduce ECO/Green scalability level that uses less HW resources



Reduce bandwidth



- Reached FPS Target?
 - Look at optimizing bandwidth
 - Will reduce power consumption & heat
- Investigate bandwidth optimization techniques
 - Render the same quality
 - With better bandwidth performance.

Reduce FPS

ANTGAMER teases 1000Hz monitor with 2026 release plans



You have 3 phases where you can throttle the FPS of the game:

- Menus
 - Fixed frame rate, say 60 or 30 FPS
 - In Unreal use SetMaxFPS
- Idle
 - Lower FPS even more
- Gameplay
 - This is the hardest one
 - VSYNC doesn't cap 1000 Hz frequency
 - Fixed frame rate menu option is better
 - Throttling based on temperature?

E-sport players
wants high FPS, but
no-one wants loud
fan noise and a
computer that almost
burns up!

Quality settings and
clamping FPS is how
you can reduce
thermal pressure

Clint 4/18/2024 11:54 AM
GPU/CPU thermal heat + fan noise issues

When reporting please state:

- Your hardware - CPU + GPU
- Game settings which you were using (please share data with VSYNC on and VSYNC off)
- Any external factors which may have caused the increase in temperature - e.g. Twitch/Youtube open in the background
- Also provide any screenshots or videos

Note: Please share data from the menus and also whilst in a game (edited)

Hellguardian MTHN 4/18/2024 3:14 PM
results with vsync on

Hellguardian MTHN 4/18/2024 3:01 PM
in the menu it is around 50° but as soon as i start to play a game it just goes to 80°

cinqseptxd 4/19/2024 1:48 PM
Tried with vsync both on & off, not much of a difference for me other than the fan, temps are more or less the same, worth mentioning that i restarted the game after turning on vsync.

@Hellguardian MTHN in the menus i have zero problems

Torbjörn Söderman GLS 4/22/2024 7:15 PM
Thanks, when the temp is on 80 do you get a lot of noise, heat issues?
In the menus we have already clamped to 60 FPS

Hellguardian MTHN 4/22/2024 7:21 PM
yes , i hear a lot of noise and regarding the heat i didn't have any problem like the game crashing or the pc turning off because of that

the fan noise isn't too bad, but my GPU has coil whine and it's really annoying

perculated 4/19/2024 6:22 PM
CPU: AMD Ryzen 5 5600H
GPU: AMD Radeon(TM) Graphics + NVIDIA GeForce RTX 3050 Laptop
Vsync ON = 88 °C , VSYNC off is only 2 °C less

kingkun10 GLS 10/14/2025 6:55 PM
Played after a while , had riva tuner enabled and experienced very High cpu temperature around 85-90°C , is this issue known or its my machine that needs a new cooler?



SETTINGS

Solving gameplay thermal pressure

GENERAL

CONTROLS

VIDEO

AUDIO

NETWORK

ACCOUNT

GENERAL

SHOW FPS

ON

VSYNC

OFF

RENDERING QUALITY

EPIC

ANTI ALIASING

DLSS

RESOLUTION SCALE

DLAA

THERMAL SETTINGS

THERMAL MODE

TEMPERATURE

TEMPERATURE

69

ACCESSIBILITY

COLOR VISION DEFICIENCY TYPE

NORMAL VISION

GPUThrottle

Reads GPU data
(nvml, adlx, igcl)

THERMAL MODE

OFF - NO THROTTLING

FIXED FRAMERATE - SETS A FIXED FRAMERATE

SILENT - DYNAMICALLY ADJUSTS FPS TO MAINTAIN A TARGET GPU TEMPERATURE AS WELL AS FAN RPM. HELPS REDUCE FAN NOISE AND HEAT BUILDUP FOR A QUIETER, COOLER EXPERIENCE. (EXPERIMENTAL FEATURE)

TEMPERATURE - DYNAMICALLY ADJUSTS FPS TO MAINTAIN A TARGET GPU TEMPERATURE. HELPS REDUCE FAN NOISE AND HEAT BUILDUP FOR A QUIETER, COOLER EXPERIENCE. (EXPERIMENTAL FEATURE)

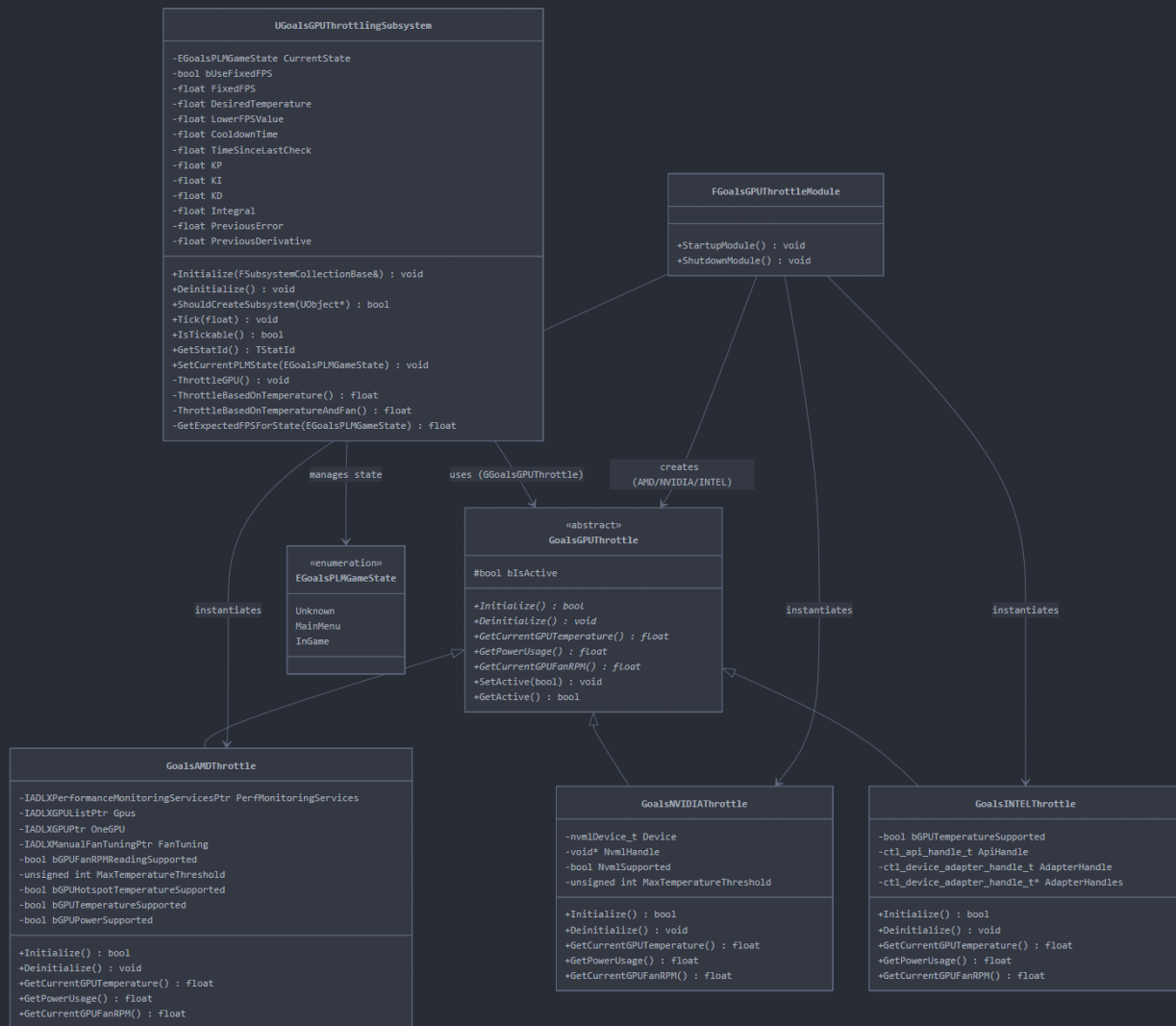
GPUThrottle: Unreal Plug-in

GoalsGPUThrottle:

- Reads GPU Temperature
- Reads GPU Power usage
- Reads GPU Fan speed
- Implements:
 - NVML (NVIDIA GPUs)
 - ADLX (AMD GPUs)
 - IGCL (Intel GPUs)

GoalsGPUThrottlingSubsystem:

- Handles state changes
 - Ingame
 - Menu
- Handles mode changes
 - Fixed Framerate
 - Silent
 - Temperature



NVIDIA Management Library (NVML)

The NVIDIA Management Library (NVML) is a C-based programmatic interface for monitoring and managing various states within Data Center GPUs. It is intended to be a platform for building 3rd party applications

```
float GoalsNVIDIAThrottle::GetCurrentGPUTemperature()
{
    if (!IsActive || !NvmlSupported)
        return 0.f;

#ifdef !UE_SERVER && PLATFORM_WINDOWS
    unsigned int temp;
    nvmlReturn_t result = NvmlDeviceGetTemperature(Device, NVML_TEMPERATURE_GPU, &temp);
    if (result != NVML_SUCCESS)
    {
        UE_LOG(LogGPUDhrottle, Warning, TEXT(InFormat: "Failed to get GPU temperature: %s"),
            UTF8_TO_TCHAR(NvmlErrorString(result)));
        return 0.f;
    }

    return static_cast<float>(temp);
#else... #if !UE_SERVER && PLATFORM_WINDOWS #else
}
}
```

```
float GoalsNVIDIAThrottle::GetPowerUsage()
{
    if (!NvmlSupported)
        return 0.f;

#ifdef !UE_SERVER && PLATFORM_WINDOWS
    unsigned int PowerMilliWatts = 0;
    nvmlReturn_t result = NvmlDeviceGetPowerUsage(Device, &PowerMilliWatts);
    if (result != NVML_SUCCESS)
    {
        UE_LOG(LogGPUDhrottle, Warning, TEXT(InFormat: "Failed to get GPU power usage: %s"),
            UTF8_TO_TCHAR(NvmlErrorString(result)));
        return 0.f;
    }

    return PowerMilliWatts / 1000.0f;
#else... #if !UE_SERVER && PLATFORM_WINDOWS #else
}
}
```



AMD Device Library eXtra (ADLX)

ADLX is a modern library and SDK designed to access features and functionality of AMD systems in the categories of Display, 3D graphics, Performance Monitoring, GPU Tuning, and more.

```
float GoalsAMDThrottle::GetCurrentGPUTemperature()
{
    if (!IsActive)
        return 0.f;

    adlx_double Temp = 0;
    IADLXGPUMetricsPtr GpuMetrics = nullptr;

    if (!PerfMonitoringServices || !OneGPU || !bGPUPowerSupported)
        return 0.f;

    if ((bGPUHotspotTemperatureSupported || bGPUTemperatureSupported) &&
        ADLX_SUCCEEDED(PerfMonitoringServices->GetCurrentGPUMetrics(OneGPU, &GpuMetrics)))
    {
        if (bGPUHotspotTemperatureSupported)
        {
            GpuMetrics->GPUHotspotTemperature(&Temp);
        }
        else
        {
            GpuMetrics->GPUTemperature(&Temp);
        }

        if (!GpuMetrics)
        {
            GpuMetrics = nullptr;
        }
    }

    return static_cast<float>(Temp);
}
```

```
float GoalsAMDThrottle::GetPowerUsage()
{
    if (!PerfMonitoringServices || !OneGPU || !bGPUPowerSupported)
        return 0.f;

    adlx_double Power = 0;
    IADLXGPUMetricsPtr GpuMetrics = nullptr;
    if (ADLX_SUCCEEDED(PerfMonitoringServices->GetCurrentGPUMetrics(OneGPU, &GpuMetrics)))
    {
        GpuMetrics->GPUPower(&Power);
        GpuMetrics = nullptr;
    }

    return static_cast<float>(Power);
}
```



Intel Graphics Control Library (IGCL)

IGCL is meant to be a collection of high level APIs for all control aspects of hardware, primarily graphics.

IGCL allows global control and tweaking of display, media & 3D capabilities.

```
Float GoalsINTELThrottle::GetCurrentGPUTemperature()
{
    if (!bIsActive || !AdapterHandle || !bGPUTemperatureSupported)
        return 0.f;

    uint32_t SensorCount = 0;
    ctl_result_t Res = ctlEnumTemperatureSensors(AdapterHandle, &SensorCount, nullptr);
    if (Res != CTL_RESULT_SUCCESS || SensorCount == 0)
    {
        UE_LOG(LogGPUThrottle, Warning, TEXT(UE_FMT("Temperature component not supported. Error: %hs"), DecodeRetCode(Res).c_str()));
        return 0.f;
    }

    ctl_temp_handle_t* TempHandles = new ctl_temp_handle_t[SensorCount];
    Res = ctlEnumTemperatureSensors(AdapterHandle, &SensorCount, TempHandles);
    double Temperature = 0;

    if (Res == CTL_RESULT_SUCCESS)
    {
        for (uint32_t i = 0; i < SensorCount; ++i)
        {
            ctl_temp_properties_t Props = {};
            Props.Size = sizeof(Props);
            if (ctlTemperatureGetProperties(TempHandles[i], &Props) == CTL_RESULT_SUCCESS && Props.type == CTL_TEMP_SENSORS_GPU)
            {
                ctlTemperatureGetState(TempHandles[i], &Temperature);
            }
        }
    }

    delete[] TempHandles;
    return static_cast<float>(Temperature);
}
```

```
Float GoalsINTELThrottle::GetPowerUsage()
{
    if (!bIsActive || !AdapterHandle)
        return 0.f;

    uint32_t DomainCount = 0;
    ctl_result_t Res = ctlEnumPowerDomains(AdapterHandle, &DomainCount, nullptr);
    if (Res != CTL_RESULT_SUCCESS || DomainCount == 0)
    {
        UE_LOG(LogGPUThrottle, Warning, TEXT(UE_FMT("Power component not supported. Error: %hs"), DecodeRetCode(Res).c_str()));
        return 0.f;
    }

    ctl_pwr_handle_t* PowerHandles = new ctl_pwr_handle_t[DomainCount];
    Res = ctlEnumPowerDomains(AdapterHandle, &DomainCount, PowerHandles);

    static uint64_t PrevEnergy = 0;
    static uint64_t PrevTime = 0;

    uint64_t CurrEnergy = 0, CurrTime = 0;

    if (Res == CTL_RESULT_SUCCESS)
    {
        for (uint32_t i = 0; i < DomainCount; ++i)
        {
            ctl_power_energy_counter_t Counter = {};
            Counter.Size = sizeof(Counter);

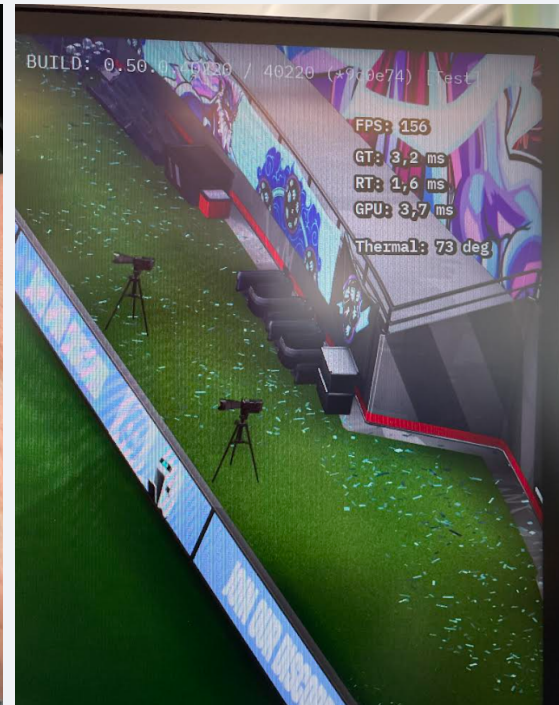
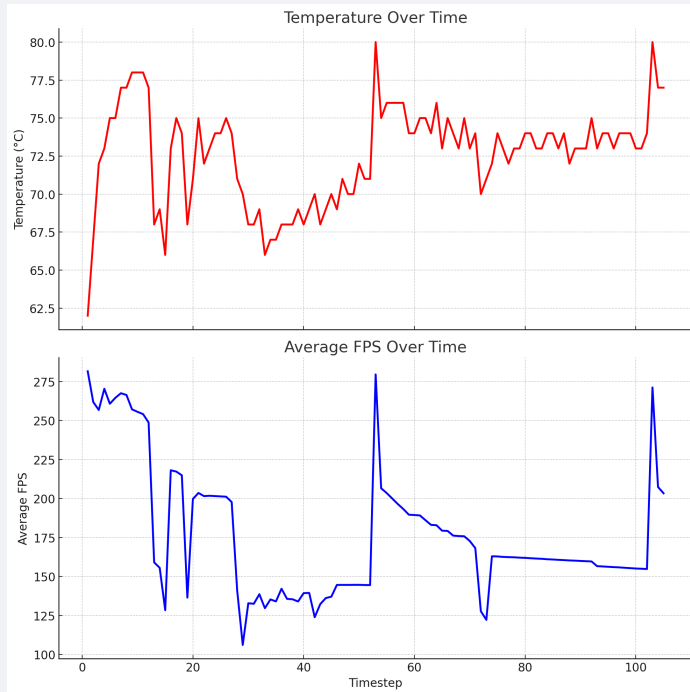
            if (ctlPowerGetEnergyCounter(PowerHandles[i], &Counter) == CTL_RESULT_SUCCESS)
            {
                CurrEnergy = Counter.energy;
                CurrTime = Counter.timestamp;
            }
        }
    }
    else
    {
        UE_LOG(LogGPUThrottle, Warning, TEXT(UE_FMT("Failed to get power handles. Error: %hs"), DecodeRetCode(Res).c_str()));
    }

    delete[] PowerHandles;
    double DeltaTime = static_cast<double>(CurrTime - PrevTime);
    double DeltaEnergy = static_cast<double>(CurrEnergy - PrevEnergy);
    PrevEnergy = CurrEnergy;
    PrevTime = CurrTime;

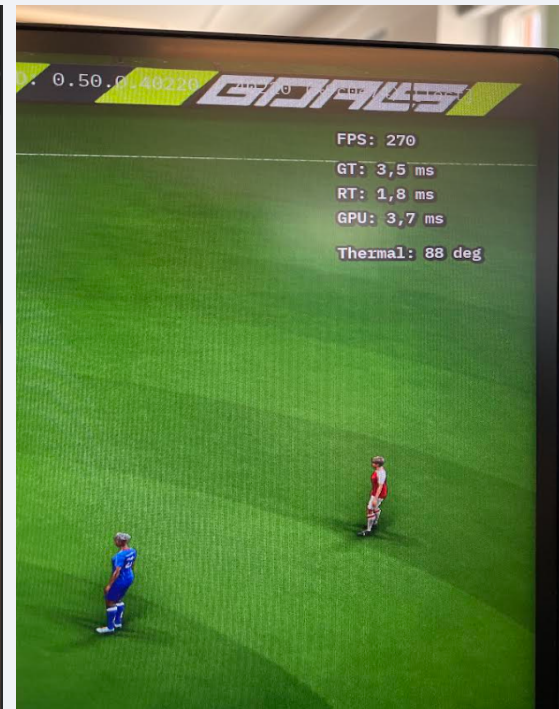
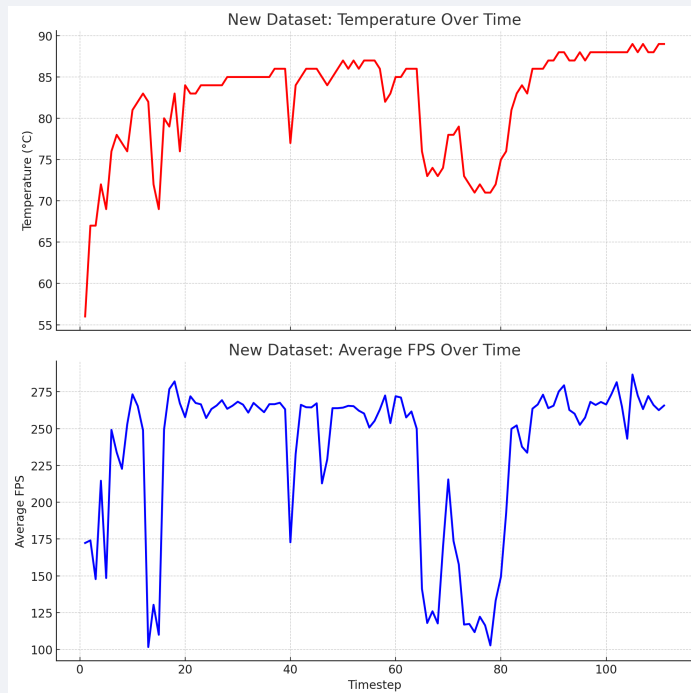
    return (DeltaTime > 0.0) ? static_cast<float>(DeltaEnergy / DeltaTime) : 0.f;
}
```



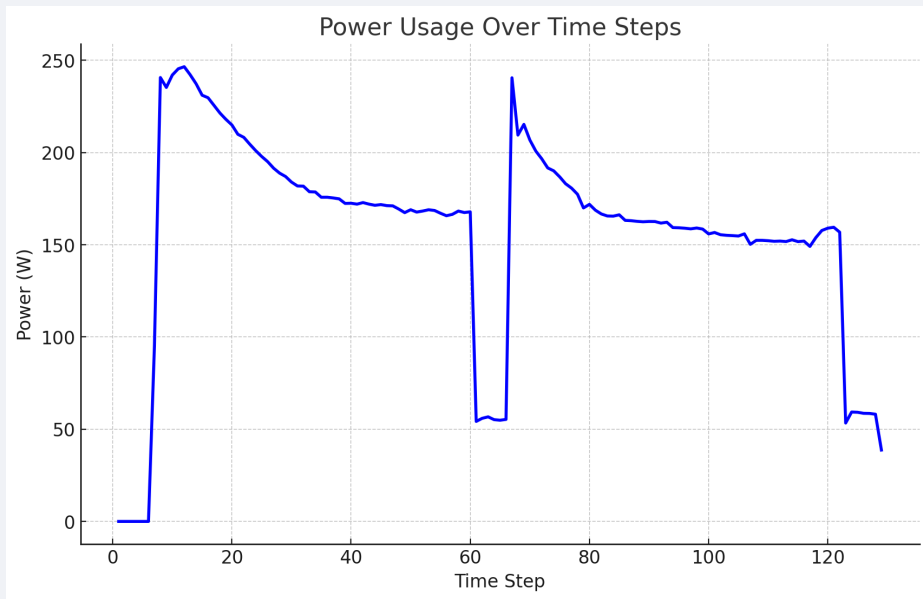
Throttle Enabled



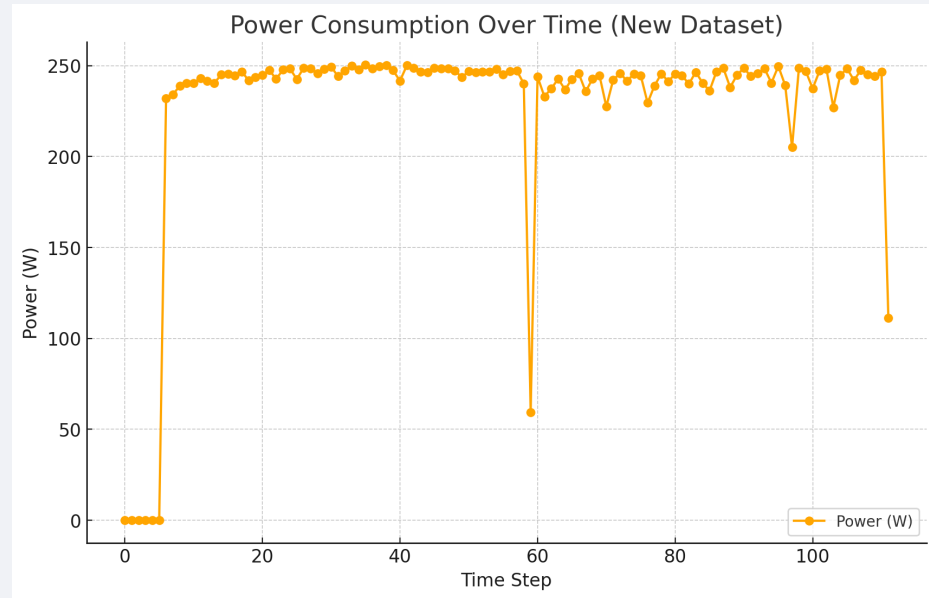
Throttle Disabled



Throttling enabled



Throttling disabled



Spain and Portugal wildfires drive worst EU season on record

28 August 2025

Share [Save](#)

Elizabeth Dawson BBC News and Erwan Rivault Data Designer, BBC
[Verify](#)



Unprecedented wildfires have scarred northern Spain in recent weeks

A record one million hectares - equivalent to about half the land area of Wales - have burned across the European Union so far this year, making it the worst wildfire season



-400000

-300000

-200000

-100000

Current Year (0 = 2024)

Existing research

Expectations from other industries & businesses

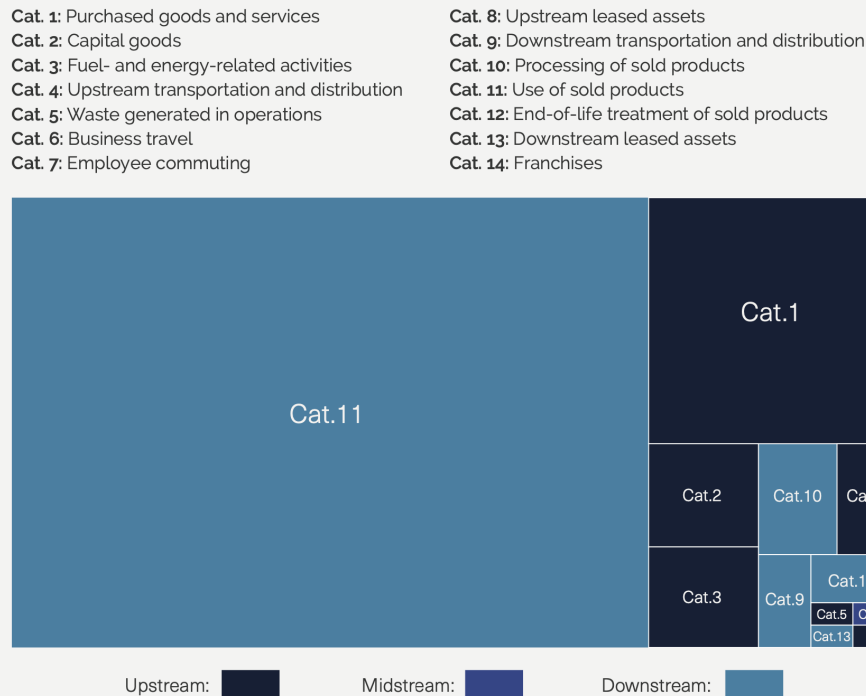
Big category: S3.11 “Use of sold products”
– **gameplay** on mobiles, consoles, PCs

Next: S3.1 “Purchased goods and services”
– Data centres, software, computers, food, desks, office stuff

Often overlooked: S3.9 “downstream T&D”
– **downloads** & **installs**

Indie relevant: S3.6 Business travel –
Flights, ubers, trains, hotels.

Figure 4. Scope 3 emissions by category (excluding category 15)²⁴



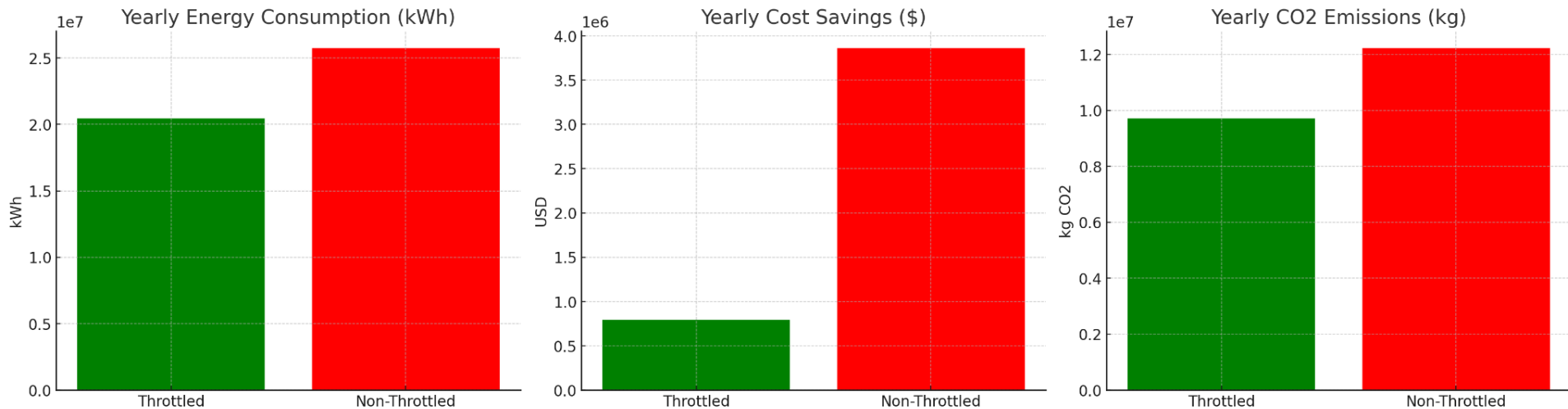
Energy consumption

Assumption of scale (Rocket League):

- ~1,9M PC DAU

- **Yearly Energy Consumption (Throttled):** 20,449,864.78 kWh
- **Yearly Energy Consumption (Non-Throttled):** 25,743,624.11 kWh
- **Yearly Energy Savings:** 5,293,759.32 kWh
- **Yearly Cost Savings:** \$794,063.90 (at \$0.15 per kWh)
- **Yearly CO₂ Savings:** 2,514,535.68 kg CO₂ (global average of 0.475 kg per kWh)

- 🌳 **41,908 trees planted and grown for 10 years** (each tree absorbs ~60 kg CO₂ over a decade).
- 🚗 **544 gas-powered cars off the road for a year** (average car emits ~4.63 metric tons CO₂ annually).
- ✈️ **2,098 one-way flights from New York to Los Angeles** (a flight emits ~1.2 metric tons CO₂ per passenger).
- 🏠 **317 homes' energy use for an entire year** (average home emits ~7.94 metric tons CO₂ annually).



GOALS Telemetry on our own consumption so far

Row	throttle_setting ▼	num_users ▼	power_samples ▼	avg_powe ▼
1	<i>null</i>	18885	24076528	33933.49504102...
2	1	844	1638563	32262.4953364625
3	2	119	206813	31890.54996059...
4	3	144	173587	32168.99027576...

- System works at scale
 - Measure power consumption
- Gamers don't know what it is
 - Uncommon
 - Not in your face
- Fixed Framerate
 - No. 1 in usage
- Throttling reduces power consumption



Future work

Integrating Carbon Awareness into GPUPhrottle

- **Goal:** Extend the Unreal Engine GPUPhrottle system to estimate real-time CO₂ emissions from gameplay sessions.
- **Problem:** Windows `PowerGridForecast` only provides *normalized severity* (0–1) — not actual carbon intensity (gCO₂/kWh).
- **Solution:** Combine GPU power telemetry with live grid data from an external API.

Implementation Overview

- **GPUPhysics:**
 - Already captures GPU power draw via ADLX, NVML, and IGCL.
 - Converts power (W) to energy (kWh) over time.
- **Carbon Data Integration:**
 - Query `Windows PowerGridForecast` → get regional *Severity* of grid carbon intensity.
 - Query `ElectricityMaps API` → get actual *carbon intensity* (gCO₂/kWh) for player's region.
 - Map *Severity* → *CarbonIntensity* using a linear or fitted relationship.
- **Emission Calculation:**

$$\text{CO}_2 \text{ (grams)} = \text{Energy (kWh)} \times \text{Carbon Intensity (gCO}_2\text{/kWh)}$$

- **Telemetry Output:**
 - Adds `CarbonIntensity` and `EmissionsGrams` to existing GPUPhrottle metrics.
 - Enables tracking of *energy efficiency* and *CO₂ footprint* per match/session.

- Gamers don't know what it is
 - Improve UI
 - Add onscreen counters
- Add idle throttle
- Add real-time CO₂ calculations
- Collaborate with SGA



Q&A

