Reducing Fortnite's Power Consumption

Contents

Introduction

In this paper, we explain how we reduced the power consumption of *Fortnite* on Xbox, PlayStation, and PC, cutting energy costs for players and reducing environmental impact. We describe how we achieved these savings with minimal noticeable impact to players and how we measured the impact of the changes in the real world. We also provide recommendations for developers who are interested in improving the energy efficiency of their own games.

The methods demonstrated can significantly improve the energy efficiency of a game with minimal development cost. We show how even simple configuration changes can make an impact on power draw.

For this work, we focused primarily on reducing the game's GPU load since the GPU is a significant contributor to overall power consumption. The solutions presented required a small amount of game-side C++ code. We did not make any significant changes to engine code beyond some minor bug fixes which will be included in the upcoming 5.3 release. We focus on Unreal Engine, but these techniques are applicable to other engines, too.

The intended audience for this paper is software engineers and technical artists.

Dynamic resolution and energy efficiency

Epic Games recommends using Unreal Engine's dynamic resolution to keep a consistent frame rate while maximizing available GPU computing power, and we make heavy use of this in *Fortnite*. Dynamic resolution works by adjusting the resolution based on the current GPU frame time in order to hit a specified budget in milliseconds. Temporal Super Resolution (TSR)—or Temporal Anti-Aliasing Upscaling (TAAU) on lower-end platforms—is then used to upscale the current frame to the final resolution (typically to 4K on PS5 or Xbox Series X).

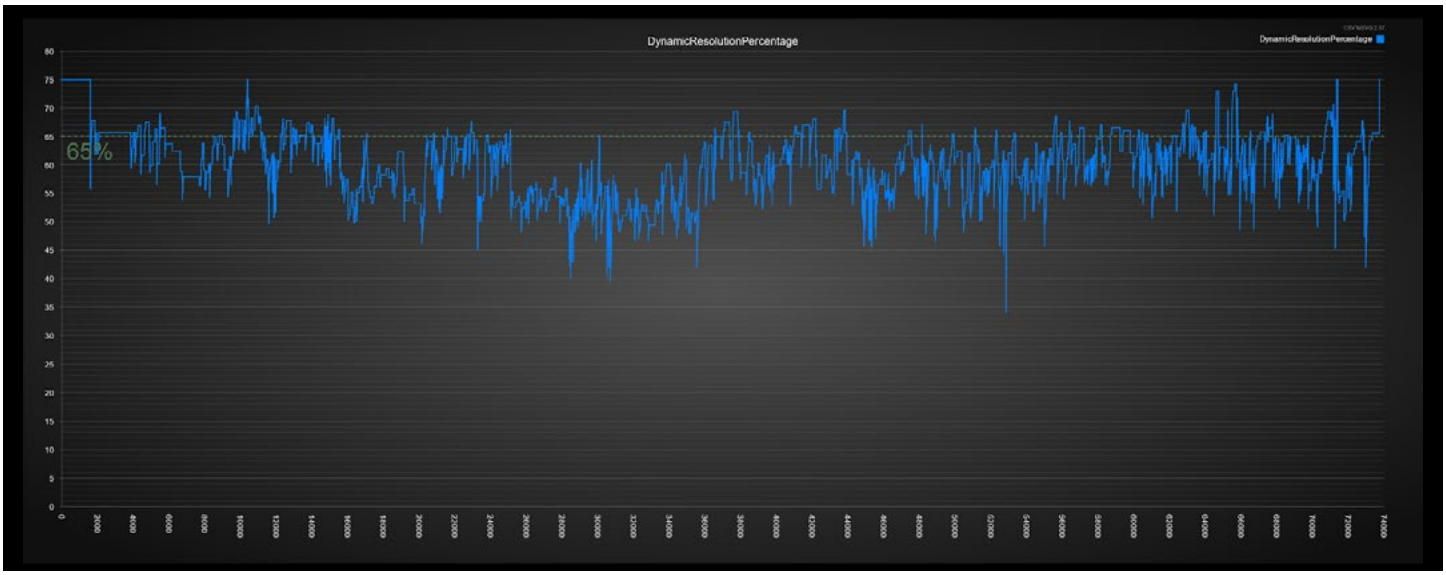
Dynamic resolution is a useful tool for maximizing GPU performance, but care is needed to ensure that it doesn't increase power load beyond what is necessary. If a game's dynamic resolution rarely or never hits the configured maximum value, the GPU load will be close to 100% most of the time, and this has a corresponding impact on power consumption. Reducing the maximum resolution enables the GPU load to decrease when performance is good and also brings some performance benefits. If you're using a high-quality upscaler such as TSR, it's often possible to reduce the maximum resolution without a perceptible difference in visual quality.

Note: More info on dynamic resolution can be found in the Unreal Engine 5 documentation.

Analyzing dynamic resolution in *Fortnite*

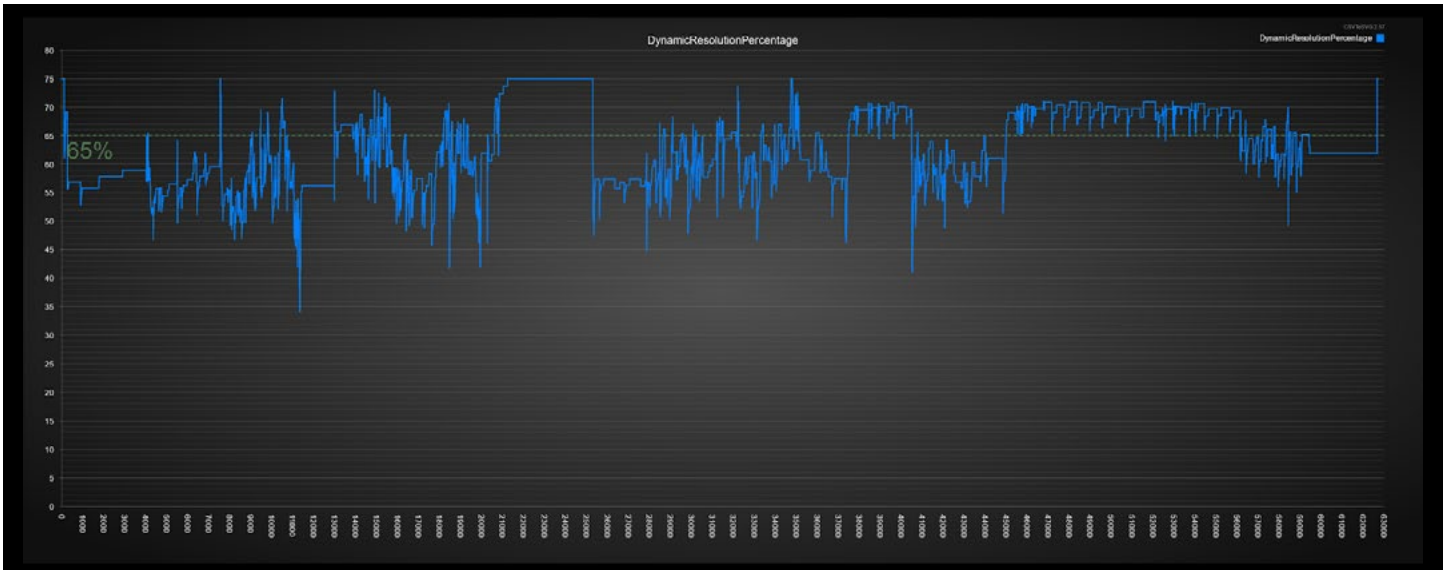
We looked at detailed performance data to see how dynamic resolution changed based on scene complexity in different *Fortnite* game modes on PS5 and Xbox Series X. We found that the game was rarely running at the configured 75% maximum resolution in Battle Royale matches. In other game modes, such as Creative, resolution was higher, but the visual benefit wasn't noticeable.

Dynamic Resolution: Battle Royale example



Dynamic resolution in a typical Battle Royale play session (PS5)

Dynamic Resolution: Creative Mode example



Dynamic resolution in a typical Creative Mode play session (PS5)

We did some visual comparisons of the game at different resolutions both in static scenes and in motion. We found very little difference between 65% and 75% resolution when using TSR for upscaling.

Zoomed-in image with 75% resolution (L) vs 65% resolution (R) in the Fortnite Lobby (PS5)

As a result of this analysis, we dropped the maximum resolution to 65% on both PS5 and Xbox Series X. This reduced power consumption both in game and also in the front-end screens.

Recommendations

Developers can adjust their game's maximum resolution setting using the `r.DynamicRes.MaxScreenPercentage` console variable in engine or device profile ini files. Note that this defaults to 100.

We recommend doing some visual comparisons of different scenes, both in motion and static, trying different resolutions using the console variable `r.DynamicRes.TestScreenPercentage`. Try to find the threshold at which raising the resolution is no longer perceptible. If you're using Temporal Super Resolution, it's likely you can reduce the maximum resolution without any perceptible visual differences.

Developers can also look at the dynamic resolution stat during gameplay using Unreal Insights or the CSV Profiler to see the typical ranges across different scenes and game modes. If your game is almost never rendering at the maximum resolution, it's likely that the GPU is close to 100% load all the time; the maximum resolution should be configured to avoid this.

In addition to reducing power consumption, you may see some performance benefits from lowering your maximum resolution. Internal render targets are allocated according to the maximum resolution, so reducing this improves the performance of fullscreen resolves and clears and also brings cache benefits. Frame rate will also be more consistent if you're hitting the maximum resolution more often, because there is more headroom to absorb variance in GPU load.

Developers can also try increasing the console variable `r.DynamicRes.TargetedGPUHeadRoomPercentage` (default: 10). By tuning this parameter, you are effectively telling the engine what fraction of time you would like the GPU to be idle. This is a more direct way of reducing power consumption but at the cost of visual fidelity. As an example, if you really wanted to be aggressive, you could use this to indicate you want to target 12 ms/frame, leaving the GPU idle for ~25% of the time on average and reducing GPU power consumption by a corresponding amount. The engine will lower resolution to try to hit that target.

Frontend—Inactivity-based energy saving

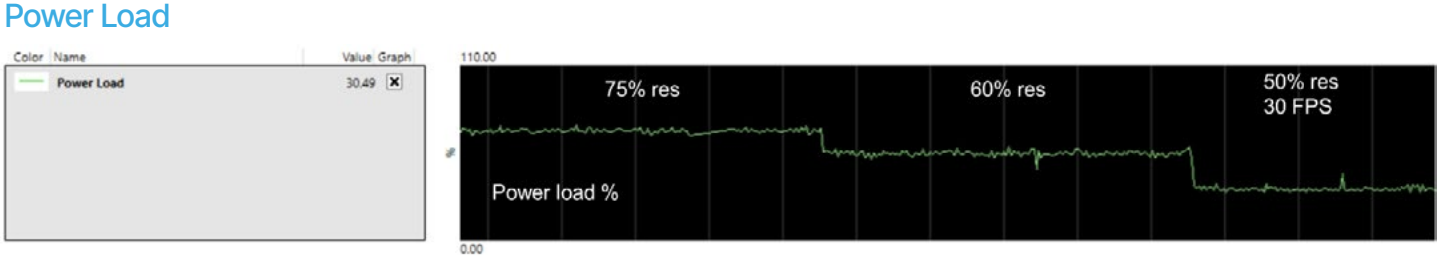
In *Fortnite*, players can spend a significant amount of time in the front-end screens such as the Lobby and the Item Shop.

We did some profiling and saw that GPU load could be almost as high in the Lobby as it is in game. The front-end scenes are simpler and so require less GPU time at a given resolution, but dynamic resolution increases to compensate for this, keeping the power consumption high.

We decided to implement a solution based on inactivity. If a player is in the front-end screens and hasn't touched an input device for an extended period of time, we can reduce the resolution and/or frame rate without it being noticeable.

Experiments

Microsoft provides a "power percentage" metric in their PIX profiling tool. This estimates the power draw of key hardware components on an Xbox Series X dev kit. We tried reducing resolution from 75% to 50% and halving the frame rate (to 30 fps) in the *Fortnite* Lobby and measured the impact in PIX. The adjusted settings reduced the power metric by around half.



The PIX power percentage metric showing different energy saving configurations on an Xbox Series X

It's worth noting that the metric is not proportional to overall power consumption at the socket. It measures the power draw of certain key components (such as the CPU and GPU) since these are the main components that a game can influence. The metric is a useful tool for iteration, but if you're interested in measuring overall power consumption, we recommend using a socket-based watt meter.

Energy mode settings

We targeted the biggest savings at the high-end current generation consoles (PS5 and Xbox Series X). These platforms render at 4K output resolution and use TSR for upscaling, which provides some flexibility to reduce the input resolution. We did a visual review dropping to 50% resolution in the Lobby after a period of inactivity and the difference was hard to perceive. This type of scene with a fixed camera and slow-moving animation is an ideal case for TSR upscaling.

The table below shows the final settings we shipped with for the various consoles and their corresponding power consumption reduction.

The inactivity time threshold we used on all consoles was 30 seconds. Energy-saving mode would become active after this time.

Console	Upscale target resolution	Frame rate (default)	Frame rate (energy saving)	Max resolution (default)	Max resolution (energy saving)	Energy savings
Xbox Series X	4K (TSR)	60	**30**	**75%***	**50%**	**36%**
Xbox Series S	1080p (TAAU)	60	**30**	85%	*unchanged*	**19%**
PS5	4K (TSR)	60	**30**	**75%***	**50%**	**36%**
PS4	1080p (TAAU)	60	**30**	100%	*unchanged*	**17%**
Xbox One	900p (TAAU)	60	**30**	100%	*unchanged*	**19%**

As an example, we measured an Xbox Series X at the wall socket; when running in energy-saving mode, we measured 117 W, versus a baseline of 184 W. In cases where we were unable to drop resolution, relative savings were lower. For example, Xbox One measured 79 W in energy-saving mode compared to a baseline of 98 W.

**The 75% maximum resolution here predated our change to globally reduce the maximum resolution to 65%, as described in the previous section.*

Implementation example

We provide some example code for handling inactivity-based energy saving in a UE5 game in front-end screens. This is intended as a starting point for a game-specific implementation. Developers may wish to expand this and incorporate support for their own game-specific logic.

The inactivity time threshold and the frame rate and resolution can be configured via console variables. This enables you to override per platform via DeviceProfile ini files:

```
C/C++
float GEnergySavingInactivityTime = 30;
static FAutoConsoleVariableRef CVarEnergySavingInactivityTime(
        TEXT("EnergySaving.InactivityTime"),
        GEnergySavingInactivityTime,
        TEXT("Idle time threshold at which energy saving kicks in (seconds). Set to 0 to disable"),
        ECVF_Default);

int32 GEnergySavingMaxFps = 0;
static FAutoConsoleVariableRef CVarEnergySavingMaxFps(
        TEXT("EnergySaving.MaxFps"),
        GEnergySavingMaxFps,
        TEXT("Max FPS for the energy saving mode. Set to 0 to disable"),
        ECVF_Default);

int32 GEnergySavingMaxScreenPercentage = 0;
static FAutoConsoleVariableRef CVarEnergySavingScreenPercentage(
        TEXT("EnergySaving.ScreenPercentage"),
        GEnergySavingScreenPercentage,
        TEXT("Max resolution percentage for the energy saving mode. Set to 0 to disable"),
        ECVF_Default);
```

We can get the last interaction time using the method `FSlateApplication::GetLastUserInteractionTime()`. By comparing this to the current time, we can determine how long it's been since a player interacted with the game. If the elapsed duration is over our configured threshold, we enable energy saving. The code below does this, but also overrides the last interaction time if the player is in game. This ensures that energy saving settings are only enabled in the front-end screens.

`LastInteractionTime` and `bEnergySavingModeEnabled` are member variables used to track the timestamp of the last interaction and whether energy saving is enabled, respectively.

```
C/C++
void FEnergySaving::Tick()
{
  if ( bIsInFrontend )
  {
    LastInteractionTime = FMath::Max( FSlateApplication::Get().GetLastUserInteractionTime(),
LastInteractionTime);
  }
  else
  {
    // If we're not in the frontend, override the time, resetting the countdown
    LastInteractionTime = FSlateApplication::Get().GetCurrentTime();
  }

  double TimeSinceLastInteraction = FSlateApplication::Get().GetCurrentTime() -
LastInteractionTime;

  bPrevEnergySavingModeEnabled = bEnergySavingModeEnabled;
  bEnergySavingModeEnabled = TimeSinceLastInteraction > GEnergySavingInactivityTime;
```

When `bEnergySavingModeEnabled` changes, we update the frame rate and resolution accordingly, storing the old value when the mode is enabled, and restoring it when it's disabled again. The member variables `MaxScreenPercentageToRestore` and `MaxFpsToRestore` are used to store the old values.

We set the console variable `t.maxfps` in order to limit the frame rate. This throttles on the game thread, ensuring the frame rate doesn't exceed the specified value. The console variable `r.DynamicRes.MaxScreenPercentage` is used to limit the maximum resolution (Note: There is a better alternative to this in UE 5.3—see below).

```C/C++
static IConsoleVariable* CvarMaxScreenPercentage = IConsoleManager::Get().FindConsoleVariable(TEXT("r.DynamicRes.MaxScreenPercentage"));
static IConsoleVariable* CvarMaxFps = IConsoleManager::Get().FindConsoleVariable(TEXT("t.maxfps"));

if ( bEnergySavingModeEnabled != bPrevEnergySavingEnabled )
{
  if ( bEnergySavingModeEnabled )
  {
    if ( GEnergySavingMaxScreenPercentage > 0 )
    {
       MaxScreenPercentageToRestore = CvarMaxScreenPercentage->GetFloat();
       CvarMaxScreenPercentage->AsVariable()->SetWithCurrentPriority(GEnergySavingMaxScreenPercentage);
    }
    if ( GEnergySavingMaxFps > 0 )
    {
      MaxFpsToRestore = CvarMaxScreenPercentage->GetFloat();
      CvarMaxFps->AsVariable()->SetWithCurrentPriority(MaxScreenPercentageToRestore);
    }
  }
  else
  {
    if ( GEnergySavingMaxScreenPercentage > 0 )
    {
      CvarMaxScreenPercentage->AsVariable()->SetWithCurrentPriority(MaxScreenPercentageToRestore);
    }
    if ( GEnergySavingMaxFps > 0 )
    {
      CvarMaxScreenPercentage->AsVariable()->SetWithCurrentPriority(MaxFpsToRestore);
    }
  }
}
```

UE 5.3 improvements

There are a number of improvements and fixes in the upcoming UE 5.3 release which provide better support for energy saving.

- We added a console variable `r.DynamicRes.ThrottlingMaxScreenPercentage` which avoids reallocation of render targets in order to avoid a visual discontinuity (aka "pop") when the maximum resolution changes. We recommend using this instead of `r.DynamicRes.MaxScreenPercentage` to limit resolution in energy-saving scenarios.
- For PS5, we added the console variable `Slate.Input.MotionFiresUserInteractionEvents`, which prevents spurious motion events from affecting `FSlateApplication::GetLastUserInteractionTime()`.
  Important: This still defaults to true currently for compatibility, so titles should set this to false if they wish to implement idle energy saving on PS5.
- We fixed a bug in Lumen reflections which caused a visual pop when the resolution changed rapidly in some scenarios.
- We added fixes for deadzone handling on PlayStation consoles, which could cause energy-saving mode to disable due to spurious input events.

Multiple energy-saving modes

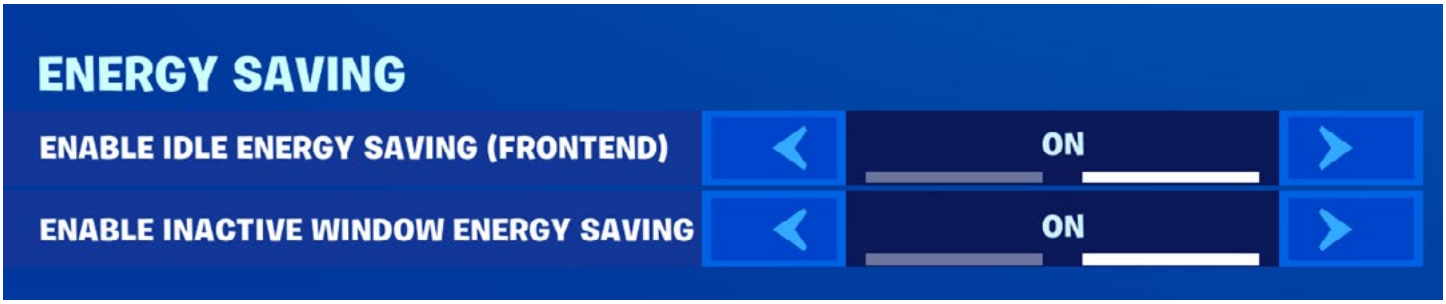
Games may also wish to implement multiple levels of energy-saving support. *Fortnite*'s initial energy-saving release had two modes: a 60 fps *low* mode which reduced only resolution, and a 30 fps *high* mode which reduced resolution and frame rate.

Adding PC support

PC support was added soon after the console release. This worked in a similar way to consoles, but we also added focus detection, so we could enable energy saving when the game was in the background. This enabled us to reduce power consumption both in the front end and in game.

Note: UE5 developers can call the `FApp::HasFocus()` to detect the current focus state of the game.

We defaulted idle and focus detection to on, but to enable content creation workflows we enabled players to opt out, with separate settings for idle detection and focus detection.



Energy efficiency options in the PC settings menu

Because PC players have a variety of anti-aliasing options to choose from, we could not guarantee a consistent quality when reducing the resolution. For this reason, PC energy saving does not affect resolution, only frame rate. This is something we may explore in future for players with TSR enabled, however.

PCs include a wide variety of hardware, so the total impact on power consumption is much harder to quantify than on consoles. However, the GPU remains one of the main contributors to power draw across all hardware, and so it's reasonable to assume that these changes will make a significant difference to energy use.

Experimenting with the live game

Fortnite's final settings were reached after a number of iterations on the live game, and we rolled it out in a staged manner, starting with Xbox Series consoles.

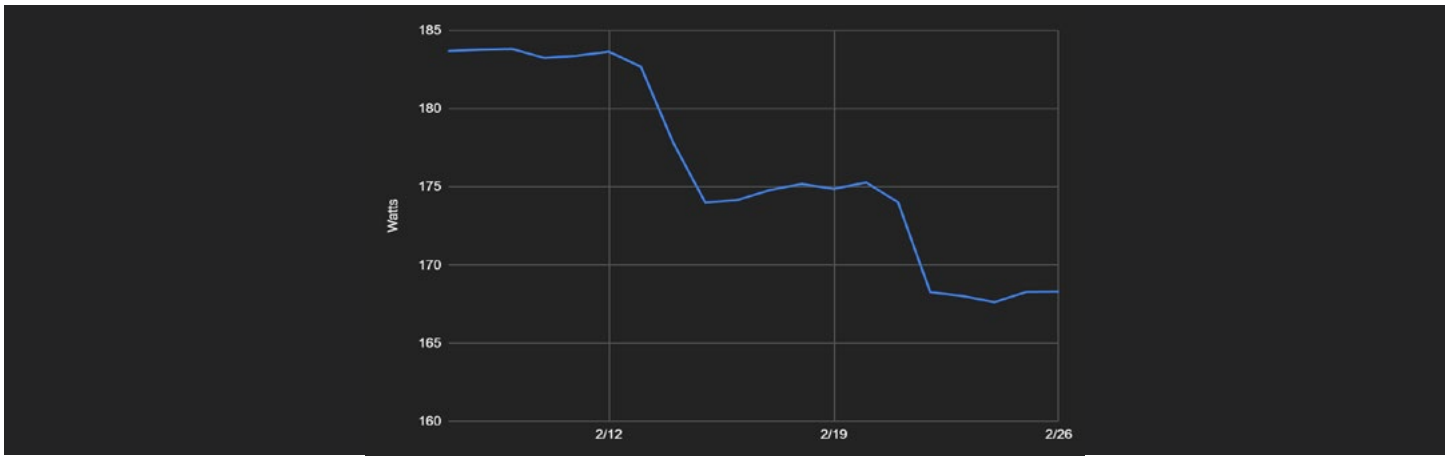
We were interested to gauge player response and also gather data to see how long players were spending in the energy-saving mode in the front end. We added telemetry to our code enabling us to see how long energy-saving mode was enabled.

The Xbox team at Microsoft provided live telemetry showing the average power consumption of our title across the whole Xbox Series player base. This data is based on a sampling of retail consoles in the live environment. This information was extremely useful in order to guide our efforts. The chart below shows a section of the data from Series X consoles.

Our initial energy saving settings were relatively conservative and focussed purely on front-end inactivity: After one minute in the front end, we dropped the maximum resolution to 60% (from 75) and after two minutes, we dropped the resolution to 50% and reduced the frame rate to 30 fps. The first drop in power consumption in the chart below corresponds to these settings.

After analyzing the data and reviewing player sentiment, we decided to take a more aggressive approach. We adjusted the front-end energy saving settings to kick in fully after 30 seconds, dropping to 30 fps and 50% resolution; we also reduced the maximum resolution of the game from 75% to 65% (as described in the section Analyzing Dynamic Resolution in Fortnite). These changes correspond to the second drop in power consumption in the chart below.

Average power consumption



Average power consumption by date on Xbox Series X, showing initial and improved energy-saving configurations

We highly recommend careful testing and measuring to check the impact of your changes in the live environment. We worked closely with Epic's internal Player Support and Community teams to ensure that the changes were imperceptible to players as measured by either team's volume reporting.

Based on our telemetry, we discovered a bug in our code preventing energy saving from working properly on PS5. We were able to fix this in the next release (see UE 5.3 Improvements, above).

Conclusion

As a result of these changes, we estimate around 200 MWh per day of savings across our total player base, or 73 GWh per year (equivalent to 14 wind turbines running for a year). And importantly, we helped to reduce the energy bills of our players.

Our main finding was that it's possible to make a significant difference to a game's energy efficiency without a large development effort. Even small configuration changes can make a noticeable difference to overall power consumption, and significant savings are possible with some careful tuning and logic. We hope that our work will inspire other developers to make energy savings in their own games.

We want to thank our partners at Xbox for their support with this project and we encourage developers to check out the tools documentation and case studies on their Gaming Sustainability learning resources site.

About this document

Author

Ben Woodhouse

Contributors

Nicolas Mercier

Nick Penwarden

Editor

Ross Hogben

Layout

Carys Norfor