3DMARK®

# Technical Guide

Updated January 14, 2021

**3DMark – The Gamer's Benchmark** .......... **5**
3DMark benchmarks at a glance .......... 7
3DMark features by edition .......... 8
Test compatibility .......... 10
Legacy benchmark tests .......... 11
Good testing guide .......... 13
Options .......... 14
Guide to 3DMark results .......... 17
Custom Benchmark settings .......... 20
Notes on DirectX 11.1 .......... 21
**Time Spy** .......... **23**
DirectX 12 .......... 24
Direct3D feature levels .......... 25
System requirements .......... 26
Graphics test 1 .......... 27
Graphics test 2 .......... 28
Time Spy CPU test .......... 29
Time Spy Extreme CPU test .......... 30
Scoring .......... 32
DirectX 12 features in Time Spy .......... 35
Time Spy engine .......... 43
Post-processing .......... 48
Time Spy version history .......... 49
**Port Royal** .......... **51**
Microsoft DirectX Raytracing .......... 52
How to measure ray tracing performance .......... 53
System requirements .......... 54
Graphics test .......... 55
Scoring .......... 56
Port Royal engine .......... 57
Port Royal version history .......... 66
**Night Raid** .......... **68**
Native Support for Windows 10 on ARM .......... 69
System requirements .......... 70
Graphics test 1 .......... 71

Graphics test 2 .......... 72
CPU test .......... 73
Scoring .......... 74
Night Raid engine .......... 76
Night Raid version history .......... 81
**Wild Life .......... 83**
System requirements .......... 84
Wild Life .......... 85
Graphics test .......... 86
Scoring .......... 87
Wild Life engine .......... 88
Wild Life version history .......... 90
**Fire Strike .......... 92**
System requirements .......... 93
Default settings .......... 94
Graphics test 1 .......... 95
Graphics test 2 .......... 96
Physics test .......... 97
Combined test .......... 98
Scoring .......... 99
Fire Strike engine .......... 101
Post-processing .......... 104
Fire Strike version history .......... 106
**DirectX Raytracing feature test .......... 108**
System requirements .......... 109
Technical details .......... 110
Scoring .......... 112
Interactive mode .......... 113
DirectX Raytracing feature test history .......... 114
**PCI Express feature test .......... 116**
System requirements .......... 117
Technical details .......... 118
Scoring .......... 119
PCI Express feature test history .......... 120
**VRS feature test .......... 122**

System requirements ........ 123
Variable-Rate Shading ........ 124
Technical details ........ 125
Scoring ........ 127
Interactive mode ........ 128
VRS feature test history ........ 132
**NVIDIA DLSS feature test ........ 134**
System requirements ........ 135
Deep Learning Super Sampling ........ 136
How to test DLSS performance ........ 138
How to compare DLSS image quality ........ 141
Result screen ........ 142
Graphics engine ........ 144
NVIDIA DLSS feature test history ........ 145
**Stress Tests ........ 146**
Options ........ 147
Technical details ........ 148
Scoring ........ 149
**How to report scores ........ 150**
**Release notes ........ 152**
**About UL ........ 153**

# 3DMark – The Gamer's Benchmark

3DMark is a tool for measuring the performance of PCs and mobile devices. It includes many different benchmarks, each designed for a specific class of hardware from smartphones to laptops to high-performance gaming PCs.

> ⚠ This guide is for the Windows version. There are separate guides for the Android version and the iOS version.

3DMark works by running intensive graphical and computational tests. The more powerful your hardware, the smoother the tests will run. Don't be surprised if frame rates are low. 3DMark benchmarks are very demanding.

Each benchmark gives a score that you can use to compare similar systems. When testing devices or components, be sure to use the most appropriate test for the hardware's capabilities and report your results using the full name of the benchmark test, for example:

✓ "Video card scores 5,800 in 3DMark Fire Strike benchmark."
✕ "Video card scores 5,800 in 3DMark benchmark."

3DMark is used by millions of gamers, hundreds of hardware review sites and many of the world's leading manufacturers. We are proud to say that 3DMark is the world's most popular and widely used benchmark.

## The right test every time

We've made it easy to find the right test for your hardware. When you open the 3DMark app, the Home screen will recommend the most suitable benchmark. You can find and run other tests on the Benchmarks screen.

## Choose your tests

3DMark grows bigger every year with new tests. When you buy 3DMark from Steam, you can choose to install only the tests you need. In 3DMark Advanced and Professional Editions, tests can be installed and updated independently.

## Complete Windows benchmarking toolkit

3DMark includes benchmarks for DirectX 12 and DirectX 11compatible hardware. All tests are powered by modern graphics engines that use Direct3D feature levels to target compatible hardware.

## Compare scores

See how your 3DMark score compares with results from other systems with the same combination of CPU and GPU.

## Estimated game performance

3DMark helps you relate your score to real-world game performance by estimating the frame rates you can expect in a selection of popular games.

## Cross-platform benchmarking

You can measure the performance of Windows, Android, and iOS devices and compare scores across platforms.

# 3DMark benchmarks at a glance

3DMark includes many benchmarks, each designed for a specific class of hardware. You will get the most useful and relevant results by choosing the most appropriate test for your system.

| BENCHMARK | TARGET HARDWARE | ENGINE | RENDERING RESOLUTION[1] |
|---|---|---|---|
| Time Spy Extreme | 4K gaming with DirectX 12 | DirectX 12 feature level 11 | 3840 × 2160 (4K UHD) |
| Time Spy | High-performance DirectX 12 gaming PCs | DirectX 12 feature level 11 | 2560 × 1440 |
| Port Royal | Graphics cards with Microsoft DirectX Raytracing support | DirectX 12 feature level 12_1 | 2560 × 1440 |
| Night Raid | PCs with integrated graphics | DirectX 12 feature level 11 | 1920 × 1080 |
| Wild Life | Lightweight notebooks and tablets | Vulkan 1.1 | 2560 × 1440 |
| Fire Strike Ultra | 4K gaming with DirectX 11 | DirectX 11 feature level 11 | 3840 × 2160 (4K UHD) |
| Fire Strike Extreme | High-performance DirectX 11 gaming PCs | DirectX 11 feature level 11 | 2560 × 1440 |
| Fire Strike | DirectX 11 gaming PCs | DirectX 11 feature level 11 | 1920 × 1080 |

1 The resolution shown in the table is the resolution used to render the Graphics tests. In most cases, the Physics test or CPU test will use a lower rendering resolution to ensure that GPU performance is not a limiting factor.

# 3DMark features by edition

| | BASIC EDITION | ADVANCED EDITION | PROFESSIONAL EDITION[2] |
|---|---|---|---|
| Time Spy Extreme | × | ● | ● |
| Time Spy | ● | ● | ● |
| Port Royal | × | ● | ● |
| Night Raid | ● | ● | ● |
| Wild Life | × | ● | ● |
| Fire Strike Ultra | × | ● | ● |
| Fire Strike Extreme | × | ● | ● |
| Fire Strike | ● | ● | ● |
| DirectX Raytracing feature test | × | ● | ● |
| PCI Express feature test | × | ● | ● |
| VRS feature test | × | ● | ● |
| NVIDIA DLSS feature test | × | ● | ● |
| Stress Tests | × | ● | ● |
| Score comparison | ● | ● | ● |
| Estimated game performance | × | ● | ● |
| Hardware monitoring | ● | ● | ● |
| Custom benchmark settings | × | ● | ● |

[2] Professional Edition annual license required to access the full list of tests and features.

| | BASIC EDITION | ADVANCED EDITION | PROFESSIONAL EDITION[2] |
|---|---|---|---|
| Install tests independently | × | ● | ● |
| Skip demo option | × | ● | ● |
| Save results offline | × | ● | ● |
| Private, offline results option | × | × | ● |
| Command-line automation | × | × | ● |
| Image Quality Tool | × | × | ● |
| Export result data as XML | × | × | ● |
| Compatible with Testdriver® | × | × | ● |
| Licensed for commercial use | × | × | ● |

# Test compatibility

| | WINDOWS | ANDROID | IOS |
|---|---|---|---|
| TIME SPY EXTREME | ● | × | × |
| TIME SPY | ● | × | × |
| PORT ROYAL | ● | × | × |
| NIGHT RAID | ● | × | × |
| WILD LIFE | ● | ● | ● |
| FIRE STRIKE ULTRA | ● | × | × |
| FIRE STRIKE EXTREME | ● | × | × |
| FIRE STRIKE | ● | × | × |
| SLING SHOT | × | ● | ● |
| DIRECTX RAYTRACING | ● | × | × |
| PCI EXPRESS | ● | × | × |
| VRS | ● | × | × |
| NVIDIA DLSS | ● | × | × |

# Legacy benchmark tests

Benchmarks have a natural lifespan that ends when they no longer provide meaningful results on modern hardware. When old benchmarks are used with new hardware, the results can be skewed or limited in ways that reduce their accuracy and relevance.

Unsupported tests are hidden in the app by default. If you need to run a legacy test for some reason, you can find them by going to the **Benchmarks** screen and using the filter option to "Show unsupported tests."

## 3DMark Sky Diver

Support for 3DMark Sky Diver ended on January 14, 2021.

3DMark Sky Diver is a DirectX 11 benchmark for PCs with integrated graphics. Released in 2014, it is now too lightweight for modern PCs.

UL recommends using 3DMark Night Raid (DirectX 12) or 3DMark Wild Life (Vulkan) to benchmark laptops, notebooks and other systems with integrated graphics.

## 3DMark API Overhead feature test

Support ended on January 14, 2021.

The 3DMark API Overhead feature test was released in 2015 to coincide with the launch of Windows 10 and DirectX 12. The test compares the draw call performance of graphics APIs. At launch, this was the most significant difference between older APIs and new, low-level APIs like DirectX 12, Metal and Vulkan.

Today, developers are more likely to choose a graphics API based on feature support and compatibility. Draw call performance is no longer the deciding factor. UL has no plans to update or replace this test.

## 3DMark Cloud Gate

Support for 3DMark Cloud Gate ended on January 14, 2020.

Cloud Gate is a DirectX 11 feature level 10 benchmark that was released in 2013. It was designed to test basic Windows notebooks and home PCs with integrated graphics. It is now too lightweight for modern hardware.

We recommend using 3DMark Wild Life (Vulkan) or 3DMark Night Raid (DirectX 12) to benchmark PCs with integrated graphics.

## 3DMark Ice Storm Extreme

Support for 3DMark Ice Storm Extreme ended on January 14, 2020.

Ice Storm Extreme is a cross-platform benchmark that was released in 2013. On Windows tablets, the benchmark uses DirectX 11 feature level 9. On Android and iOS devices, Ice Storm Extreme uses OpenGL ES 2.0. This benchmark is now too lightweight for modern mobile devices.

We recommend using 3DMark Wild Life for benchmarking smartphones and tablets. Wild Life is a cross-platform benchmark that is available for Windows, Android and iOS.

## 3DMark Ice Storm

Support for 3DMark Ice Storm benchmarks ended on January 14, 2020.

Ice Storm is a cross-platform benchmark that was released in 2013. On Windows tablets, the benchmark uses DirectX 11 feature level 9. On Android and iOS devices, Ice Storm uses OpenGL ES 2.0. This benchmark is now too lightweight for modern mobile devices.

We recommend using 3DMark Wild Life for benchmarking smartphones and tablets. Wild Life is a cross-platform benchmark that is available for Windows, Android and iOS.

# Good testing guide

To get accurate and consistent benchmark results you should test clean systems without third party software installed. When that is not possible, you should close other background tasks, especially automatic updates or tasks that feature pop-up alerts such as email and messaging programs.

- Running other programs during the benchmark can affect the results.
- Don't touch the mouse or keyboard while running tests.
- Do not change the window focus while the benchmark is running.
- You can cancel a test by pressing the ESC key.

### Recommended process

1. Install all critical updates to ensure your operating system is up to date.
2. Install the latest approved drivers for your hardware.
3. Close other programs.
4. Run the benchmark.

### Expert process

1. Install all critical updates to ensure your operating system is up to date.
2. Install the latest approved drivers for your hardware.
3. Restart the computer or device.
4. Wait 2 minutes for start-up to complete.
5. Close other programs, including those running in the background.
6. Wait for 15 minutes.
7. Run the benchmark.
8. Repeat from step 3 at least three times to verify your results.

# Options

The settings on the Options screen apply to all available benchmark tests.

## License

### Register / Unregister

If you have a 3DMark Advanced or Professional Edition upgrade key, copy it into the box and press the Register button. If you wish to unregister your key, so you can move your license to a different machine for example, press the Unregister button.

## Version details

Here you see the current version number and status of the various benchmark tests available in 3DMark. If a newer version is available, you will be able to update from this screen.

## General

### Language

Use this drop down to change the display language. The choices are:

- English
- German
- Japanese
- Korean
- Russian
- Simplified Chinese
- Traditional Chinese
- Spanish

### GPU count

You can use this drop down to tell 3DMark how many GPUs are present in the system you are testing. The default choice, Automatic, is fine in most cases and should only be changed in the rare instances when SystemInfo is unable to correctly identify the system's hardware.

### Scaling mode

This option controls how the rendered output of each test, which is at a fixed resolution regardless of hardware, is scaled to fit the system's Windows desktop resolution.

The default option is Centered, which maintains the aspect ratio of the rendered output and, if needed, adds bars around the image to fill the remainder of the screen.

Selecting Stretched will stretch the rendered output to fill the screen without preserving the original aspect ratio. This option does not affect the test score.

### Output resolution

3DMark tests are rendered at a fixed resolution regardless of hardware – the *rendering resolution.* The resulting frames are then scaled to fit the system's Windows desktop resolution – *the output resolution.* The default option is automatic, which sets the output resolution to the Windows desktop resolution. Change this option if you wish to display the benchmark at some other resolution. This option does not affect the test score.

### Demo audio

Uncheck this box if you wish to turn off the soundtrack while a demo is running. This option is selected by default.

## Result

### Validate result online

This option is only available in 3DMark Professional Edition where it is disabled by default. In 3DMark Basic and Advanced Editions, all results are validated online automatically.

### Automatically hide results online

Check this box if you wish to keep your 3DMark test scores private. Hidden results are not visible to other users and do not appear in search results. Hidden results are not eligible for competitions or the Hall of Fame.

- 3DMark Basic Edition, disabled by default and cannot be selected.
- 3DMark Advanced Edition, disabled by default.
- 3DMark Professional Edition, selected by default.

## SystemInfo

### Scan SystemInfo

SystemInfo is a component used by UL benchmarks to identify the hardware in your system or device. It does not collect any personally identifiable information. This option is selected by default and is required to get a valid benchmark test score.

### SystemInfo hardware monitoring

This option controls whether SystemInfo monitors your CPU temperature, clock speed, power, and other hardware information during the benchmark run. This option is selected by default.

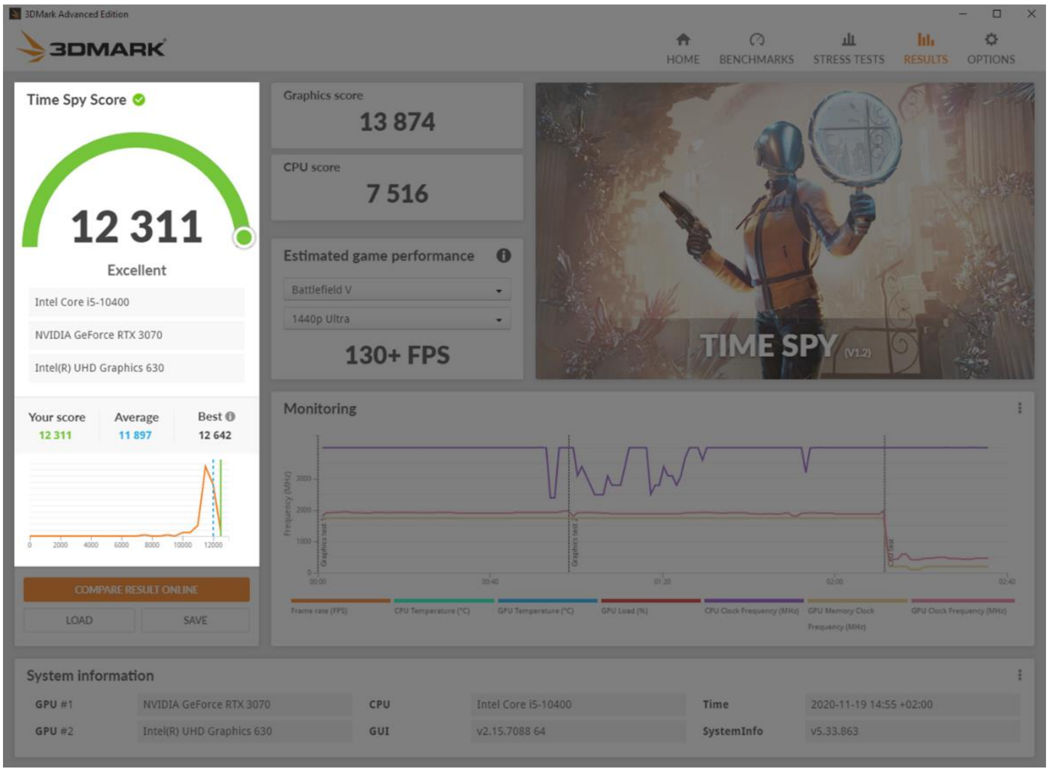
# Guide to 3DMark results

3DMark helps you learn more about your PC's performance. The benchmark result screen helps you answer,

1. Is this a good 3DMark score for my PC?
2. How does my 3DMark score relate to game performance?

## Score comparison

3DMark shows you how your benchmark score compares with results from other systems with the same components. This makes it easy to see if your PC is performing correctly.




*3DMark shows how your score compares with other results from the same hardware.*

This chart shows the range of scores submitted by other 3DMark users with the same combination of CPU and GPU.

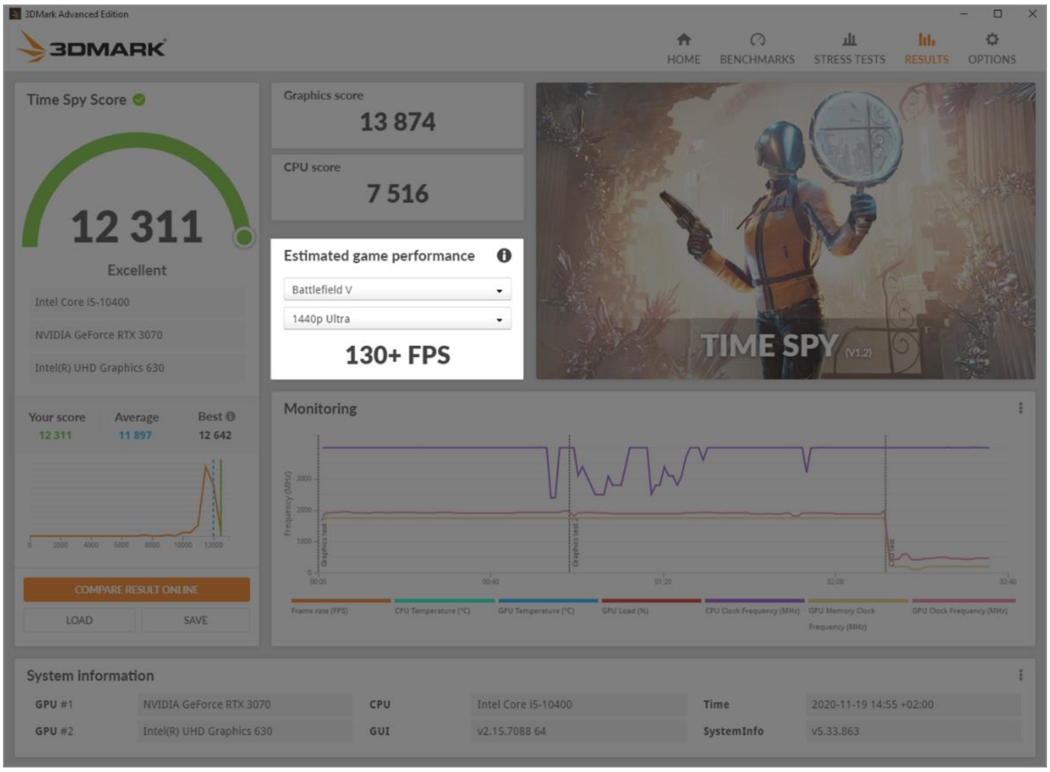
The peaks in the chart show the most common scores. The green line shows your score. The blue, dotted line shows the average score for your combination of components.

If your score is close to the average, it means your PC is working as it should. A score well below the average could indicate a hardware or

configuration problem. The best score gives you an idea of the overclocking potential of your setup.

## Estimated game performance

3DMark helps you relate your 3DMark score to real-world game performance by estimating the frame rates you can expect in a selection of popular games.




*3DMark estimates game performance from your benchmark score.*

3DMark estimates the game performance for five popular games:

- Apex Legends
- Battlefield V
- Fortnite
- GTA V
- Red Dead Redemption 2

For each game, you can choose either 1080p Ultra or 1440p Ultra settings. Ultra means using the game's highest visual quality settings. Ray tracing options, if available, are turned off.

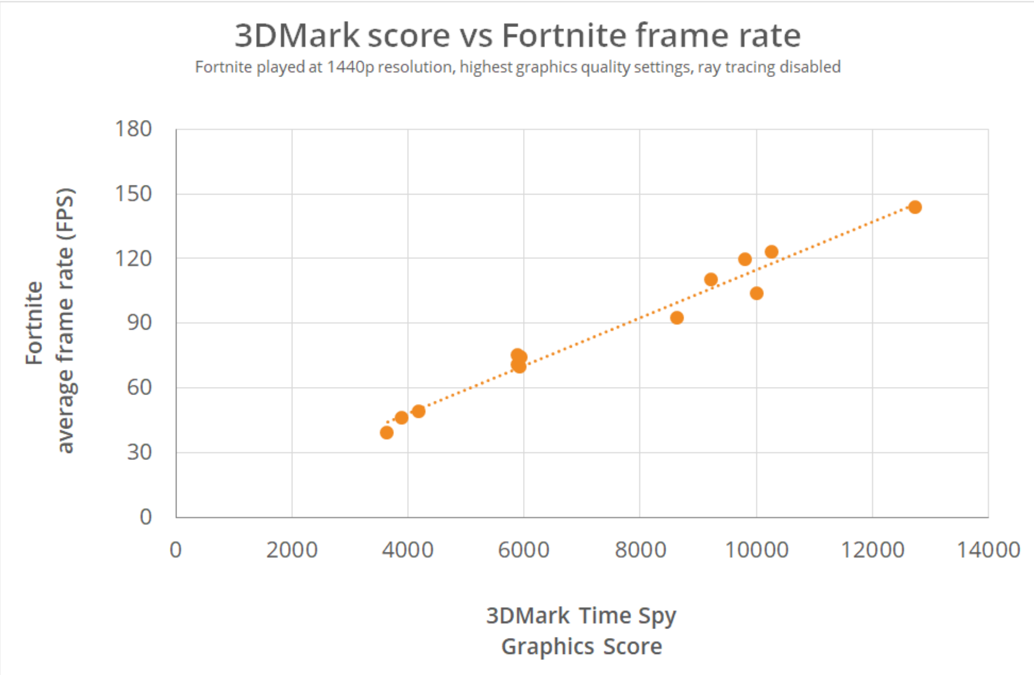
3DMark shows the average frame rate you can expect in the selected game. This frame rate is an estimate based on your 3DMark score and game testing performed by UL. Frame rates can vary within a game depending on

the level or mode. Game updates, driver updates and system updates may also affect performance.

### Estimating game performance from 3DMark scores

The game performance feature is powered by extensive data from UL's in-house testing. UL works with consumer electronics retailers to test and categorize PCs of all types. Every year, UL tests hundreds of PC systems for its retail partners. UL tests the performance of each system with a selection of benchmarks and popular games. Using this data, UL can model the relationship between 3DMark scores and game frame rates.

The chart below shows the relationship between 3DMark Time Spy Graphics score and average frame rate in Fortnite. Each orange dot shows the performance of a different PC system tested by UL.




*There is a strong correlation between 3DMark Time Spy Graphics Score and average frame rate in Fortnite.*

You can see from the chart that a PC that scores 4,000 in the 3DMark Time Spy Graphics test can run Fortnite at 50 FPS on average at 1440p with the highest graphics quality settings. A 3DMark score of 6,000 translates to an average of 70 FPS in Fortnight. A score of 12,000 translates to 140 FPS.

It's these correlations that enable 3DMark to estimate game frame rates. You can read more about estimating game performance from 3DMark scores on the UL Benchmarks website.

# Custom Benchmark settings

Each benchmark test has its own settings, found on the Custom Run tab on the Test Details screen. Use custom settings to explore the limits of your PC's performance by making tests more or less demanding.

Custom settings are only available in the Advanced and Professional Editions.

You will only get an official 3DMark test score when you run a test with the default settings. When using custom settings, you will still get the results from individual sub-tests as well as hardware performance monitoring information.

# Notes on DirectX 11.1

3DMark does use DirectX 11.1, but only in a minor way and with a fall-back for DirectX 11 to ensure compatibility with the widest range of hardware and to ensure that all tests work with Windows 7 and Windows 8.

DirectX 11.1 API features were evaluated and those that could be utilized to accelerate the rendering techniques in the tests designed to run on DirectX 11.0 were used.

## Discard resources and resource views

In cases where subsequent Direct3D draw calls will overwrite the entire resource or resource view and the application knows this, but it is not possible for the display driver to deduce it, a discard call is made to help the driver in optimizing resource usage. If DirectX 11.1 is not supported, a clear call or no call at all is made instead, depending on the exact situation. This DX11.1 optimization may have a performance effect with multi-GPU setups or with hardware featuring tile-based rendering, which is found in some tablets and entry-level notebooks.

## 16 bpp texture formats

The 16 bpp texture formats supported by DirectX 11.1 are used on Ice Storm game tests to store intermediate rendering results during post processing steps. If support for those formats is not found, 32 bpp formats are used instead. This optimization gives a noticeable performance effect on hardware such as tablets, entry-level notebooks for which the Ice Storm tests provide a suitable benchmark.

There are no visual differences between the tests when using DirectX 11 or DirectX 11.1 in 3DMark and the practical performance difference from these optimizations is limited to Ice Storm on very low-end Windows hardware.

# Time Spy

Time Spy is a DirectX 12 benchmark test for high-performance gaming PCs running Windows 10. Time Spy includes two Graphics tests, a CPU test, and a demo. The demo is for entertainment only and does not influence the score.

With its pure DirectX 12 engine, which supports features like asynchronous compute, explicit multi-adapter, and multi-threading, Time Spy is the ideal benchmark for testing the DirectX 12 performance of modern graphics cards.

3DMark Advanced and Professional Editions include Time Spy Extreme, a more demanding 4K benchmark test designed for the latest graphics cards and multi-core processors.

Scores from 3DMark Time Spy and Time Spy Extreme should not be compared with each other - they are separate tests with their own scores, even though they share similar content.

Time Spy benchmarks are only available in the Windows editions of 3DMark.

## Time Spy

Time Spy is a DirectX 12 benchmark test for Windows 10 gaming PCs. The Graphics tests are rendered at 2560 × 1440 resolution.

## Time Spy Extreme

Time Spy Extreme is a 4K gaming benchmark that raises the rendering resolution to 3840 × 2160. A 4K monitor is not required, but your graphics card must have at least 4 GB of memory. The enhanced CPU test is ideal for processors with 8 or more cores.

# DirectX 12

DirectX 12, introduced with Windows 10, is a low-level graphics API that reduces processor overhead. With less overhead and better utilization of modern GPU hardware, a DirectX 12 game engine can draw more objects, textures and effects to the screen. How much more? Take a look at the table below that compares Time Spy with Fire Strike, a high-end DirectX 11 test.

## Average amount of processing per frame

| | Vertices | Triangles | Tessellation patches | Compute shader invocations |
|---|---|---|---|---|
| 3DMark Fire Strike Graphics test 1 | 3,900,000 | 5,100,000 | 500,000 | 1,500,000 |
| 3DMark Fire Strike Graphics test 2 | 2,600,000 | 5,800,000 | 240,000 | 8,100,000 |
| 3DMark Time Spy Graphics test 1 | 30,000,000 | 13,500,000 | 800,000 | 29,000,000 |
| 3DMark Time Spy Graphics text 2 | 40,000,000 | 14,000,000 | 2,400,000 | 31,000,000 |

With DirectX 12, developers can significantly improve the multi-thread scaling and hardware utilization of their titles. But it requires a considerable amount of graphics expertise and memory-level programming skill. The programming investment is significant and must be considered from the start of a project.

3DMark Time Spy was developed with expert input from AMD, Intel, Microsoft, NVIDIA, and the other members of the UL Benchmark Development Program. It is one of the first DirectX 12 apps to be built "the right way" from the ground up to fully realize the performance gains that DirectX 12 offers.

# Direct3D feature levels

DirectX 11 introduced a paradigm called Direct3D feature levels. A feature level is a well-defined set of GPU functionality. For instance, the 9_1 feature level implements the functionality in DirectX 9.

With feature levels, 3DMark tests can use modern DirectX 12 and DirectX 11 engines and yet still target older DirectX 10 and DirectX 9 level hardware. For example, 3DMark Cloud Gate uses a DirectX 11 feature level 10 engine to target DirectX 10 compatible hardware.

Time Spy uses DirectX 12 feature level 11_0. This lets Time Spy leverage the most significant performance benefits of the DirectX 12 API while ensuring wide compatibility with DirectX 11 hardware through DirectX 12 drivers.

Game developers creating DirectX 12 titles are also likely to use this approach since it offers the best combination of performance and compatibility.

# System requirements

| | TIME SPY | TIME SPY EXTREME |
|---|---|---|
| OS[3] | Windows 10, 64-bit | Windows 10, 64-bit |
| PROCESSOR | 1.8 GHz dual-core CPU with SSSE3 support | 1.8 GHz dual-core CPU with SSSE3 support |
| STORAGE | 2 GB free disk space | 2 GB free disk space |
| GPU | DirectX 12 | DirectX 12 |
| VIDEO MEMORY | 1.7 GB<br>(2 GB or more recommended) | 4 GB |

[3] Time Spy will not run on multi-GPU systems with Windows 10 build 10240, but this is due to an issue with Windows. You must use Windows 10 build 10586 ("November Update") or later to enable multi-GPU configurations to work.

# Graphics test 1

Graphics tests are designed to stress the GPU while minimizing the CPU workload to ensure that CPU performance is not a limiting factor.

Graphics test 1 focuses more on rendering of transparent elements. It utilizes the A-buffer heavily to render transparent geometries and big particles in an order-independent manner. Graphics test 1 draws particle shadows for selected light sources. Ray-marched volumetric illumination is enabled only for the directional light. All post-processing effects are enabled.

### Processing performed in an average frame

| | VERTICES | TESSELLATION PATCHES | TRIANGLES | PIXEL SHADER INVOCATIONS[4] | COMPUTE SHADER INVOCATIONS |
|---|---|---|---|---|---|
| TIME SPY | 30 million | 0.8 million | 13.5 million | 80 million | 29 million |
| TIME SPY EXTREME | 30 million | 0.9 million | 13.5 million | 220 million | 63 million |

[4] This figure is the average number of pixels processed per frame before the image is scaled to fit the native resolution of the device being tested. If the device's display resolution is greater than the test's rendering resolution, the actual number of pixels processed per frame will be even greater.

# Graphics test 2

Graphics tests are designed to stress the GPU while minimizing the CPU workload to ensure that CPU performance is not a limiting factor.

Graphics test 2 focuses more on ray-marched volume illumination with hundreds of shadowed and unshadowed spot lights. The A-buffer is used to render glass sheets in an order-independent manner. Also, lots of small particles are simulated and drawn into the A-buffer. All post-processing effects are enabled.

## Processing performed in an average frame

| | VERTICES | TESSELLATION PATCHES | TRIANGLES | PIXEL SHADER INVOCATIONS[5] | COMPUTE SHADER INVOCATIONS |
|---|---|---|---|---|---|
| TIME SPY | 40 million | 2.4 million | 14 million | 50 million | 31 million |
| TIME SPY EXTREME | 40 million | 2.4 million | 14 million | 220 million | 68 million |

[5] This figure is the average number of pixels processed per frame before the image is scaled to fit the native resolution of the device being tested. If the device's display resolution is greater than the test's rendering resolution, the actual number of pixels processed per frame will be even greater.

# Time Spy CPU test

The CPU test measures processor performance using a combination of physics computations and custom simulations. It is designed to stress the CPU while minimizing GPU load to ensure that GPU performance is not a limiting factor.

The CPU test uses a fixed time step. This means that the speed at which the timeline advances is constant. As a result, the same frames are simulated and rendered on every system but the time taken to complete the test will vary.

The two main components of the test workload are an implementation of a boid system to simulate flocking behaviour and a physics simulation. The boids use a simple, highly optimized simulation whereas the physics simulation is performed with the x86 path of the Bullet Open Source Physics library (v2.83) using rigid bodies and a Featherstone solver. Of the two, the boids are more dominant and make up between 40% and 70% of the workload.

In the Time Spy CPU test, the boids are implemented with SSSE3 vectorization, which is common practice in games.

The test metric is the average frame rate reported in frames per second. A higher value means better performance.

# Time Spy Extreme CPU test

In 2017, both AMD and Intel introduced new processors with more cores than had ever been seen in a consumer-level CPU before.

The Time Spy CPU test does not scale well on processors with 10 or more threads. It simply doesn't have enough workload for the large-scale parallelization that high-end CPUs provide. A new test is needed.

## Enhanced test design

The Time Spy Extreme CPU test also features a combination of physics computations and custom simulations, but it is three times more demanding than the Time Spy CPU test.

Adding more simulation requires more visualization, however, which can make rendering the bottleneck in some cases. This issue was solved by changing the metric for the test.

Instead of calculating the time taken to execute an entire frame, in the Extreme CPU test we only measure the time taken to complete the simulation work. The rendering work in each frame is done before the simulation and doesn't affect the score.

The test metric is average simulation time per frame reported in milliseconds. Unlike frame rate, with this metric a lower number means better performance.

## CPU instruction sets

In the Time Spy test, the boids simulation is implemented with SSSE3.

In the Extreme CPU test, half of the boids systems can use more advanced CPU instruction sets, up to AVX2 if supported by the processor. The remaining half use the SSSE3 code path.

The split makes the test more realistic since games typically have several types of simulation or similar tasks running at once and would be unlikely to use a single instruction set for all of them.

## Custom run

With Custom run settings, you can choose which CPU instruction set to use, up to AVX512. The selected set will be used for all boid systems, provided it is supported by the processor under test.

You can evaluate the performance gains of different instruction sets by comparing custom run scores, but note that the choice of set doesn't affect

the physics simulations, which always use SSSE3 and are 15-30% of the workload.

# Scoring

Time Spy produces an overall Time Spy score, a Graphics test sub-score, and a CPU test sub-score. The scores are rounded to the nearest integer. The better a system's performance, the higher the score.

## Overall Time Spy score

The 3DMark Time Spy score formula uses a weighted harmonic mean to calculate the overall score from the Graphics and CPU test scores.

$$Time\ Spy\ score = \frac{Wgraphics + Wcpu}{\frac{Wgraphics}{Sgraphics} + \frac{Wcpu}{Scpu}}$$

Where:

| | | |
|---|---|---|
| $Wgraphics$ | = | The Graphics score weight, equal to 0.85 |
| $Wcpu$ | = | The CPU score weight, equal to 0.15 |
| $Sgraphics$ | = | Graphics test score |
| $Scpu$ | = | CPU test score |

For a balanced system, the weights reflect the ratio of the effects of GPU and CPU performance on the overall score. Balanced in this sense means the Graphics and CPU test scores are roughly the same magnitude.

For a system where either the Graphics or CPU score is substantially higher than the other, the harmonic mean rewards boosting the lower score. This reflects the reality of the user experience. For example, doubling the CPU speed in a system with an entry-level graphics card doesn't help much in games since the system is already limited by the GPU. Likewise for a system with a high-end graphics card paired with an underpowered CPU.

## Graphics test scoring

Each Graphics test produces a raw performance result in frames per second (FPS). We take a harmonic mean of these raw results and multiply it by a scaling constant to reach a Graphics score ($Sgraphics$) as follows:

$$Sgraphics = 164\ \times \frac{2}{\frac{1}{Fgt1} + \frac{1}{Fgt2}}$$

Where:

| | | |
|---|---|---|
| $Fgt1$ | = | The average FPS result from Graphics test 1 |
| $Fgt2$ | = | The average FPS result from Graphics test 2 |

The scaling constant is used to bring the score in line with traditional 3DMark score levels.

## Time Spy CPU test scoring

The CPU test consists of three increasingly heavy levels, each of which has a ten second timeline. The third, and heaviest, level produces a raw performance result in frames per second (FPS) which is multiplied by a scaling constant to give a CPU score ($Scpu$) as follows:

$$Scpu = 298 \times Fcpu_3$$

Where:

| | | |
|---|---|---|
| $Fcpu_3$ | = | The average FPS from the CPU test's third level |

The scaling constant is used to bring the score in line with traditional 3DMark score levels.

## Time Spy Extreme CPU test scoring

In the Extreme CPU test we only measure the time taken to complete the simulation work. The rendering work in each frame is done before the simulation and does not affect the score.[6]

The CPU score ($Scpu$) is calculated from the average simulation time per frame reported in milliseconds.

$$S_{CPU} = \frac{T_{Reference} \times S_{Reference}}{T_{Simulation}}$$

6 Note that Time Spy Extreme is not a suitable test for systems with integrated graphics. The rendering will affect the simulation time on such systems due to shared resources.

Where:

| | | |
|---|---|---|
| $T_{Reference}$ | = | Reference time constant set to 70 |
| $S_{Reference}$ | = | Reference score constant set to 5,000 |
| $T_{Reference}$ | = | The average simulation time per frame |

The scaling constants are used to bring the score in line with traditional 3DMark score levels.

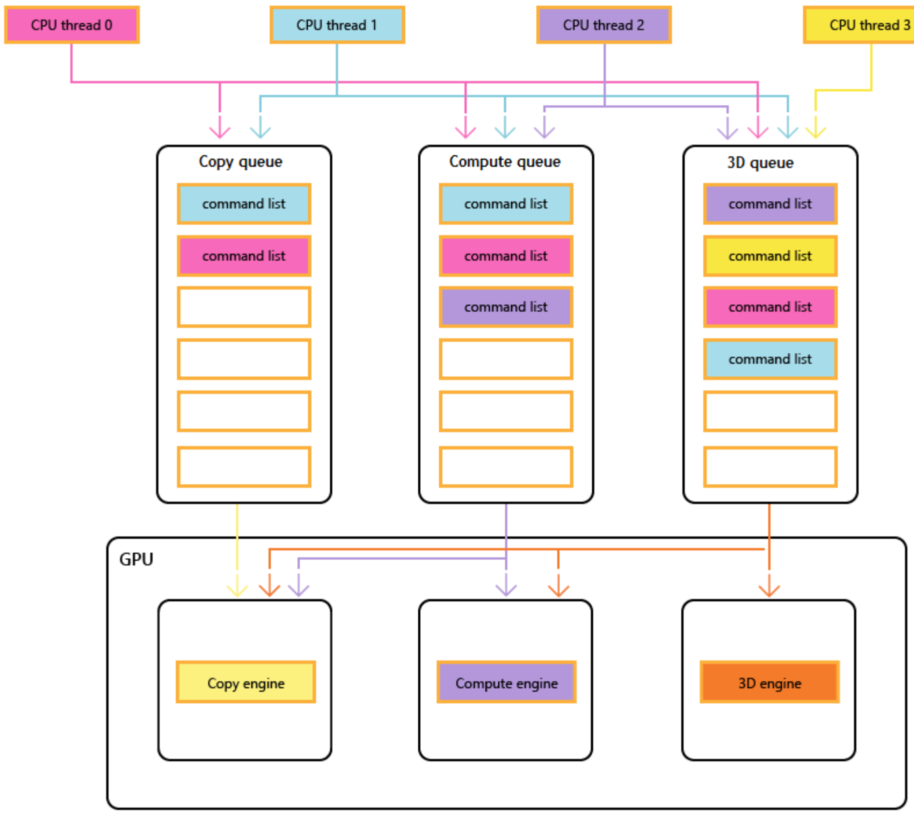
# DirectX 12 features in Time Spy

## Command lists and asynchronous compute

Unlike the Draw/Dispatch calls in DirectX 11 (with immediate context), In DirectX 12, the recording and execution of command lists are decoupled operations. There is no thread limitation on recording command lists. Recording can happen as soon as the required information is available.

Quoting from MSDN:

> *"Most modern GPUs contain multiple independent engines that provide specialized functionality. Many have one or more dedicated copy engines, and a compute engine, usually distinct from the 3D engine. Each of these engines can execute commands in parallel with each other. Direct3D 12 provides granular access to the 3D, compute and copy engines, using queues and command lists.*
>
> *"The following diagram shows a title's CPU threads, each populating one or more of the copy, compute and 3D queues. The 3D queue can drive all three GPU engines, the compute queue can drive the compute and copy engines, and the copy queue simply the copy engine.*




### Command list execution

For GPU work to happen, command lists are executed on queues, which come in variants called DIRECT (commonly known as graphics or 3D as in the diagram above), COMPUTE and COPY. Submission of a command list to a queue can happen on any thread. The D3D runtime serializes and orders the lists within a queue.

| | |
|---|---|
| DIRECT command list | This command list type supports all types of commands including Draw calls, compute Dispatches and Copies. |
| COMPUTE command list | This command list type supports compute Dispatch and Copy commands. |
| DIRECT queue | This queue can be used for executing all types of command lists supported by DirectX 12. |
| COMPUTE queue | This queue accepts compute and copy command lists. |
| COPY command list and queues | This command list and queue type accepts only copy commands and lists respectively. |

Once initiated, multiple queues can execute in parallel. This parallelism is commonly known as 'asynchronous compute' when COMPUTE queue work is performed at the same time as DIRECT queue work.

It is up to the driver and the hardware to decide how to execute the command lists. The application cannot affect this decision through the DirectX 12 API.

Please see MSDN for an introduction to the Design Philosophy of Command Queues and Command Lists, and for more information on Executing and Synchronizing Command Lists.

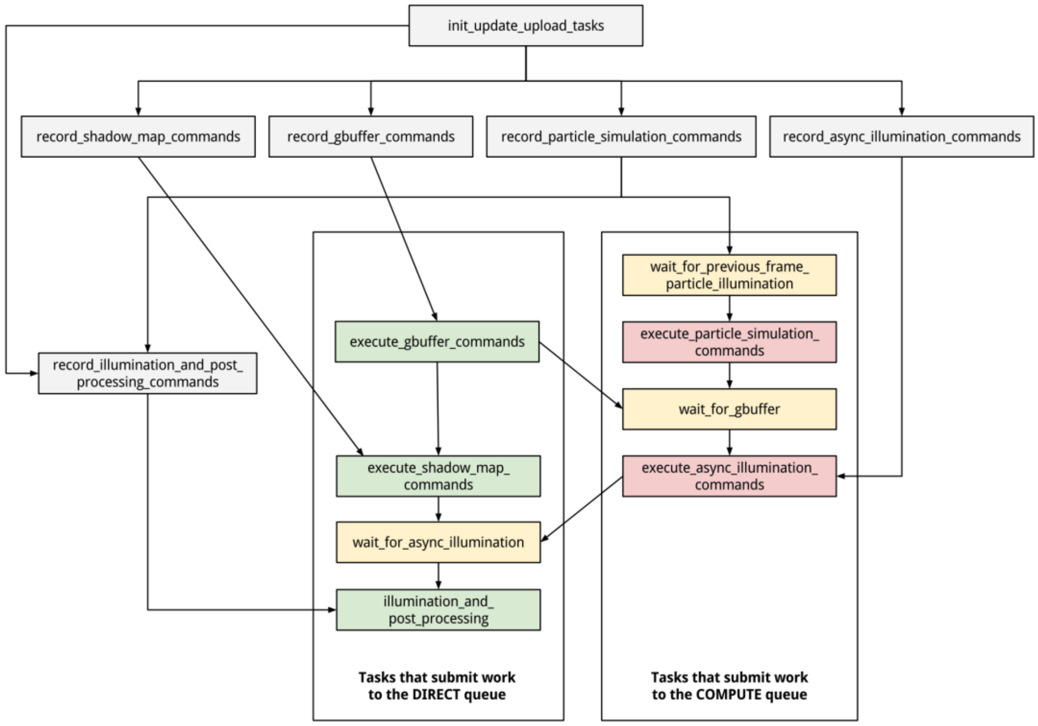
**In Time Spy,** the engine uses two command queues: a DIRECT queue for graphics and compute and a COMPUTE queue for asynchronous compute. [7]

The implementation is the same regardless of the capabilities of the hardware being tested. It is ultimately the decision of the underlying driver whether the work in the COMPUTE queue is executed in parallel or in serial.

There is a large amount of command lists as many tasks have their own command lists, (several copies so that frames can be pre-recorded).

[7] The COPY queue is generally used for streaming assets. It is not needed in Time Spy as we load all assets before the benchmark run begins to ensure the test does not gain a dependency on storage or main memory.

### Simplified DAG[8] of 3DMark Time Spy queue usage




Each task encapsulates a complex task substructure that is omitted in this simplified graph for clarity. If there are no dependencies, tasks are executed on the CPU in parallel.

**Grey** tasks are CPU tasks. The async_illumination_commands task contains light culling and tiling, environment reflections, HBAO, and unshadowed surface illumination.

**Green** tasks are submissions to the DIRECT (graphics) queue. G-buffer draws, shadow map draws, shadowed illumination resolve, and post-processing are executed on the DIRECT queue. G-buffer draws, shadow maps and some parts of the post-processing are done with graphics shaders, while illumination resolve and the rest of the post processing is done in compute shaders.

**Red** tasks are submissions to the COMPUTE queue. Particle simulation, light culling and tiling, environment reflections, HBAO and unshadowed surface illumination resolve are executed on the COMPUTE queue. All tasks in the compute queue must be done in compute shaders.

8 Directed Acyclic Graph (DAG), see https://en.wikipedia.org/wiki/Directed_acyclic_graph.

**Yellow** tasks are submissions of synchronization points. The significance of these can be seen by noting that execute_async_illumination_commands cannot be executed on the GPU before execute_gbuffer_commands is completed, but the submission happens ahead of the execution, (unless we are CPU bound). The GPU needs to know that it should wait for a task to complete execution before a dependent task can begin executing. When the execution is split between queues then this operation should be done by the engine otherwise a RAW hazard occurs. There is another dependency between particle simulation and completion of particle illumination in the previous frame. The simulation happens on the compute queue, which will cause a WAR hazard if it is not synchronized with the Present occurring on the graphics queue.

The order of submission can be obtained from the dependency graph. However, it is entirely up to the driver and the hardware to decide when to actually execute the given list as long as it is executed in order in its queue.

### Compute queue work items (in order of submission)

1. Particle simulation

   This pass is recorded and executed at the beginning of a frame because it doesn't depend on the G-buffer. Thus its recording and submission is done in parallel with recording and submission of geometry draws (G-Buffer construction).

2. Light culling and tiling
3. Environment reflections
4. Horizon based ambient occlusion
5. Unshadowed surface illumination

   These passes are recorded and submitted in parallel with G-Buffer recording and submission, but executed only after the G-Buffer is finished executing and in parallel with shadow maps execution. This is because they depend on the G-Buffer, but not on the shadow maps.

### Disabling asynchronous compute in benchmark settings

The asynchronous compute workload per frame in Time Spy varies between 10% and 20%. To observe the benefit on your own hardware, you can optionally choose to disable asynchronous compute using the Custom run settings in 3DMark Advanced and Professional Editions.

Running with asynchronous compute disabled in the benchmark forces all work items usually associated with the COMPUTE queue to instead be put in the DIRECT queue.

## Explicit multi-adapter

In DirectX 11, control of GPU adapters is implicit - the drivers use multiple GPUs on behalf of an application.

In DirectX 12, control of multiple GPUs is explicit. The developer can control what work is done on each GPU and when. With explicit multi-adapter control, one can implement more complex multi-GPU models, for example choosing to execute partial workloads for a frame across different GPUs.

A GPU adapter can be any graphics adapter, from any manufacturer, that supports D3D12. Each adapter is referred to as a node. There are two multi-adapter modes called linked-node adapter and multi-node adapter.

With linked-node (LDA) the programmer has access to and control over an SLI/Crossfire configuration of similar GPUs through one device interface. LDA enables some extra features over multi-node, such as faster transfers between GPUs, cross-node resource sharing and shared swap-chain (back-buffer).

With multi-node (MDA) each GPU appears as a separate device, even if they are similar and linked. With MDA, the programmer can control any and all GPUs available in the system. But the programmer must explicitly declare which GPU should execute the recorded work. MDA allows much more fine-grained control over rendering and work submission, allowing you to divide work between a discrete graphics card and an integrated GPU for example.

**Time Spy** uses explicit alternate frame rendering on linked-node configurations to improve performance on the most common multi-GPU setups used by gamers today. MDA configurations of heterogeneous adapters are not supported.

## Multi-threaded GPU work recording and submission

DirectX 11 offers multi-threaded (deferred) context support, but not all vendors implement it in hardware, so it is slow. And overall, it is quite limited.

DirectX 12 really takes multi-threaded rendering to the next level. With DirectX 12, the programmer is in the control of everything. There are a few operations that cannot be executed at the same time on multiple threads, but otherwise, there are not many rules.

Resources must be manually transitioned to the correct states, progress within a frame must be tracked explicitly, and any potential hazards must be handled explicitly. All synchronization of CPU and GPU workloads must be

done using fences and barriers, as there is no validation or checks in the driver.

**In Time Spy**, the rendering is heavily multithreaded. Command lists are recorded on all logical cores.

## Improved resource allocation, explicit state tracking, and persistent mapping

In DirectX 11, there are no heaps. The driver manages everything, including all states. Transfers to GPU memory must go through the API layer.

In DirectX 12, there are multiple ways to allocate resources. Programmers can create heaps, big piles of data that can later be filled with textures and buffers. Heaps also save memory by allowing resources to be placed on top of each other, for example render target surfaces.

All resource states must be explicitly declared. Resources have an initial state, and they must be transitioned to the correct state before the rendering commands are executed. For example, if a resource is going to be written to, it must be transitioned to a write state. The same applies for all other operations.

Since all state is explicit, the driver no longer has 'guess' the intent of the programmer, which allows faster execution. State can be changed across different work packets (command lists).

Some buffers can be persistently mapped to CPU memory to mirror the same buffer in GPU memory. This allows transfers to GPU memory with less stalls and also removes the need to invalidate buffers. But on the other hand, it puts the responsibility of managing the buffer on the programmer.

**In Time Spy**, all features are used, including heaps with overlapping resources to save memory. States are explicitly handled as they should be. Persistently mapped (streaming) buffers are used for all dynamic data with custom resource hazard prevention using fences.

## Pre-built GPU state objects

In DirectX 11, individual states (like bound shaders) can be changed at any time. There are no limitations. But the driver must optimize during runtime if necessary, which can lead to stalled rendering.

In DirectX 12, the GPU pipeline state is managed by separate pipeline state objects that encapsulate the whole state of the graphics/compute engine. In the graphics case, this encompasses things like the rasterizer state, different shaders (e.g. vertex and pixel shader), and the blending mode. State switching is done in one step by replacing the whole pipeline at once.

Since pipelines are pre-built before they are bound, the driver can optimize them beforehand. During runtime, only the GPU state reconfiguration is required based on the already optimized state. This allows very fast state switching. It removes the need for 'warm-up' before rendering, since the drivers don't cache state as often as with DirectX 11.

Pipelines can also be compiled during runtime, of course. Games can compile only the necessary pipelines during startup. If a new pipeline object is required later, it can be created easily in a separate thread without halting any of the application logic threads.

**In Time Spy**, all pipelines are built during startup. State changes are minimized by sorting by pipeline state object during rendering.

## Resource binding

As mentioned in the previous section on pipelines, when a new state is bound to the GPU everything about it is already known. This also applies for resource bindings. Pipeline state objects also contain information about the resources that will be bound to the shader and how they will reside in the GPU memory.

DirectX 12 uses descriptors and descriptor tables to bind resources. Descriptors are very lightweight objects that contain information about the resource that is to be bound. Descriptors can be arranged in tables for easy binding of multiple resources at once. This operation is also very fast, as the table can be described by binding only one pointer.

**In Time Spy**, resource binding is used as it should be to optimize performance.

## Explicit synchronization between CPU, GPU, multiple GPUs, and multiple GPU queues

In DirectX 12, synchronization won't happen without programmer intervention. All possible resource hazards must be handled by the programmer by using various synchronization objects.

And since multiple GPU queues are supported, fences must also be used on the GPU side to make sure queues execute work when they should. It's programmer's responsibility to handle all synchronization.

**In Time Spy**, synchronization is used as it should be to optimize performance.

# Time Spy engine

To fully take advantage of the performance improvements that DirectX 12 offers, Time Spy uses a custom game engine developed in-house from the ground up. The engine was created with the input and expertise of AMD, Intel, Microsoft, NVIDIA, and the other members of the UL Benchmark Development Program.

## Multi-threading

The rendering, including scene update, visibility evaluation, and command list building, is done with multiple CPU threads using one thread per available logical CPU core. This reduces CPU load by utilizing multiple cores.

## Multi-GPU support

The engine supports the most common type of multi-GPU configuration, i.e. two identical GPU adapters in Crossfire/SLI, by using explicit multi-adapter with a linked-node configuration to implement explicit alternate frame rendering. Heterogeneous adapters are not supported.

## Visibility solution

The Umbra occlusion library (version 3.3.17 or newer) is used to accelerate and optimize object visibility evaluation for all cameras, including the main camera and light views used for shadow map rendering. The culling runs on the CPU and does not consume GPU resources.

## Descriptor heaps

One descriptor heap is created for each descriptor type when the scene is loaded. Hardware Tier 1 is sufficient for containing all the required descriptors in the heaps. Root signature constants and descriptors are used when suitable.

## Resource heaps

Implicit resource heaps created by `ID3D12Device::CreateCommittedResource()` are used for most resources. Explicitly created heaps are used for some target resources to reduce memory consumption by placing resources that not needed at the same time on top of each other.

## Asynchronous compute

Asynchronous compute is utilized heavily to overlap multiple rendering passes for maximum utilization of the GPU. Async compute workload per frame varies between 10-20%.

## Tessellation

The engine supports Phong tessellation and displacement-map-based detail tessellation.

Tessellation factors are adjusted to achieve the desired edge length for the output geometry on the render target (G-buffer, shadow map or other). Additionally, patches that are back-facing and patches that are outside of the view frustum are culled by setting the tessellation factor to zero.

Tessellation is turned entirely off by disabling hull and domain shaders when the size of an object's bounding box on the render target drops below a given threshold.

If an object has several geometry LODs, tessellation is used on the most detailed LOD.

## Geometry rendering

Objects are rendered in two steps. First, all opaque objects are drawn into the G-buffer. In the second step, transparent objects are rendered to an A-buffer, which is then resolved on top of surface illumination later on.

Geometry rendering uses a LOD system to reduce the number of vertices and triangles for objects that are far away. This also results in bigger on-screen triangle size.

The material system uses physically based materials. The following textures can be used as input to materials. Not all textures are used on all materials.

| MATERIAL TEXTURE | FORMAT |
|---|---|
| Albedo (RGB) + metalness (A) | BC3 or BC7 |
| Roughness (R) + Cavity (G) | BC5 |
| Normal (RG) | BC5 |
| Ambient Occlusion (R) | BC4 |
| Displacement | BC4 |
| Luminance | BC1 or BC7 |

| MATERIAL TEXTURE | FORMAT |
|---|---|
| Blend | BC4, BC5 or BC3 |
| Opacity | BC4 |

## Opaque objects

Opaque objects are rendered directly to the G-buffer. The G-buffer is composed of textures shown in the table below. A material might not use all target textures. For example, a luminance texture is only written into when drawing geometries with luminous materials.

| G-BUFFER TEXTURE | FORMAT |
|---|---|
| Depth | D24_UNORM_S8_UINT |
| Normal | R10G10B10A2_UNORM |
| Albedo | R8G8B8A8_UNORM_SRGB |
| Material Attributes | R10G10B10A2_UNORM |
| Luminance | R11G11B10_FLOAT |

## Transparent objects

For rendering transparent geometries, the engine uses a variant of an order-independent transparency technique called Adaptive Transparency (Salvi et al. 2011). Simply put, a per-pixel list of fragments is created for which a visibility function (accumulated transparency) is approximated. The fragments are blended according to the visibility function and illuminated in the lighting pass to allow them to be rendered in any order. The A-buffer is drawn after the G-buffer to fully take advantage of early depth tests.

In addition to the per-pixel lists of fragments, per 2x2 quad lists of fragments are created. The per-quad lists can be used for selected renderables instead of the per pixel lists. This saves memory when per pixel information is not required for a visually satisfying result. When rendering to per quad lists, a half resolution viewport and depth texture is used to ignore fragments behind opaque surfaces. When resolving the A-buffer fragments for each pixel, both per pixel list and per quad list are read and

blended in the correct order. Each per quad list is read for four pixels in the resolve pass.

## Lighting

Lighting is evaluated using a tiled method in multiple separate passes.

Before the main illumination passes, asynchronous compute shaders are used to cull lights, evaluate illumination from prebaked environment reflections, compute screen-space ambient occlusion, and calculate unshadowed surface illumination. These tasks are started right after G-buffer rendering has finished and are executed alongside shadow rendering. All frustum lights, omni-lights and reflection capture probes are culled to small tiles (16x16 pixels) and written to an intermediate buffer. Reflection illumination is evaluated for the opaque surfaces by sampling the precomputed reflection cubes. The results are written out to a separate texture. Ambient occlusion and unshadowed illumination results are written out to their respective targets.

Second, illumination from all lights and GI data is evaluated for the surface. The A-buffer is also resolved in a separate pass and then composed on top of surface illumination. This produces the final illumination that is sampled in the screen space reflection step, which also blends in previously computed environment illumination based on SSR quality. Reflections are applied on top of surface illumination. Surface illumination is also masked with SSAO results.

Third, volume illumination is computed. This includes two passes. The first one evaluates volume illumination from global illumination data and the second one calculates illumination from direct lights. The evaluation is done by raymarching the light ranges.

Finally, surface illumination, GI volume illumination, and direct volume illumination are composed into one final texture with some blurring, which is then fed to post-processing stages.

Shadows are sampled in both surface and volume illumination shaders. For shadow casting lights, the textures in the table below can be rendered.

| SHADOW TEXTURE | FORMAT |
| --- | --- |
| Shadow Depth | D16_UNORM |
| Particle Transmittance | R8G8B8A8_UNORM |

## Particles

Particles are simulated on the GPU using asynchronous compute queue. Simulation work is submitted to the asynchronous queue while G-buffer and shadow map rendering commands are submitted to the main command queue.

### Particle illumination

Particles are rendered by inserting particle fragments into an A-buffer. The engine utilizes a separate half-resolution A-buffer for low-frequency particles to allow more of them to be visible in the scene at once. They are blended together with the main A-buffer in the combination step. Particles can be illuminated with scene lights or they can be self-illuminated. The output buffers of the GPU light-culling pass and the global illumination probes are used as inputs for illuminated particles. The illuminated particles are drawn without tessellation and they are illuminated in the pixel shader.

### Particle shadows

Particles can cast shadows. Shadow casting particles are rendered into transmittance 3D textures for lights that have particle shadows enabled. Before being used as an input to illumination shaders, an accumulated version of the transmittance texture is created. If typed UAV loads are supported, the transmittance texture is accumulated in-place. Otherwise the accumulated result is written to an additional texture. The accumulated transmittance texture is sampled when rendering surface, particle and volume illumination by taking one sample with bilinear filtering per pixel or per ray marching step. Resolution of the transmittance texture for each spotlight is evaluated on each frame based on screen coverage of the light. For directional light, fixed resolution textures are used.

# Post-processing

## Depth of field

The effect is computed by scattering the illumination in the out-of-focus parts of the input image using the following procedure.

1. Using CS, circle of confusion radius is computed for all screen pixels based on depth texture. The information is additionally reduced to half and quarter resolutions. In the same CS pass, a splatting primitive (position, radius and color) for out-of-focus pixels whose circle of confusion radius exceeds a predefined threshold is appended to a buffer. For pixel quads and 4x4 tiles that are strongly out of focus, a splatting primitive per quad or tile is appended to the buffer instead of per pixel primitives.
2. The buffer with splatting primitives for the out-of-focus pixels is used as point primitive vertex data and, using Geometry Shader, an image of a bokeh is splatted to the positions of these primitives. Splatting is done to a texture that is divided into regions with different resolutions using multiple viewports. First region is screen resolution and the rest are a series of halved regions down to 1x1 texel resolution. The screen space radius of the splatted bokeh determines the used resolution. The larger the radius the smaller the used splatting resolution.
3. The different regions of the splatting texture are combined by up-scaling the data in the smaller resolution regions step by step to the screen resolution region.
4. Finally, the out-of-focus illumination is combined with the original illumination.

## Bloom

Bloom is based on a compute shader FFT that evaluates several effects with one filter kernel. The effects are blur, streaks, anamorphic flare and lenticular halo.

## Lens Reflections

The effect is computed by first applying a filter to the computed illumination in frequency domain like in the bloom effect. The filtered result is then splatted in several scales and intensities on top of the input image using additive blending. The effect is computed in the same resolution as the bloom effect and therefore the forward FFT needs to be performed only once for both effects. The filtering and inverse FFT are performed using the CS and floating point textures.

# Time Spy version history

| VERSION | Windows | Android | Apple | NOTES |
|---|---|---|---|---|
| 1.2 | ● | × | × | Added the GPU and monitor selector |
| 1.1 | ● | × | × | Added Time Spy Extreme |
| 1.0 | ● | × | × | Launch version |

# Port Royal

Port Royal is a graphics card benchmark for testing real-time ray tracing performance. Port Royal complements Time Spy by providing a dedicated test that focuses on ray tracing performance.

You can use Port Royal to test and compare the real-time ray tracing performance of any graphics card that supports Microsoft DirectX Raytracing—including multi-GPU systems.

Port Royal includes a Graphics test and a Demo. The Graphics Test combines real-time ray tracing and traditional rendering techniques to test GPU performance. The demo is for entertainment only and does not affect the benchmark score.

Port Royal is only available in 3DMark Advanced and Professional Editions for Windows PCs.

⚠ To run Port Royal, you need the Windows 10 October 2018 Update (1809) and a graphics card with drivers that support Microsoft DirectX Raytracing.

# Microsoft DirectX Raytracing

Real-time ray tracing promises to bring new levels of realism to in-game graphics.

Ray tracing is not a new technique, but until recently it has been too computationally demanding to use in real-time.

With modern GPUs, it's now possible to use rasterization for most of the rendering while using ray tracing selectively to enhance reflections, shadows, and other effects that are difficult to achieve with traditional techniques.

Microsoft DirectX Raytracing is a new DirectX component that enables developer to use ray tracing in DirectX 12 applications. For more details, please see the following posts on the Microsoft DirectX Developer Blog:

- Announcing Microsoft DirectX Raytracing!
- DirectX Raytracing and the Windows 10 October 2018 Update

## Ray tracing in Port Royal

Port Royal uses DirectX Raytracing for reflections and shadows.

Port Royal uses DirectX Raytracing to produce realistic specular reflections with correct perspective. Ray tracing overcomes a limitation of traditional techniques by accurately reflecting objects that appear outside of the screen space and those that are occluded by other objects in the view.

Port Royal uses DirectX Raytracing to render pixel-perfect hard shadows.

# How to measure ray tracing performance

Ray tracing is very computationally demanding. You can also use Port Royal **Custom run** settings to make the test more, or less, demanding by changing the rendering resolution and other quality settings.

You can also disable ray tracing effects to see how it affects performance.

Set **Reflection mode** to **Traditional** or **Disabled** to turn off ray traced reflections.

**Traditional** reflection mode disables the DirectX Raytracing part of the reflection pipeline and keeps the rest of the pipeline as is. The performance and quality of the traditional reflections is not directly comparable with the state-of-the-art in games. This feature is only meant to let you compare performance when the DirectX Raytracing part is removed.

When reflections are **Disabled**, all parts of the reflection pipeline are removed from execution, including the traditional reflection parts.

Set **Disable ray traced shadows** to **Yes** to turn off ray traced shadows. When DirectX Raytracing is disabled, a traditional shadow map is used for the sunlight.

Custom benchmark runs do not produce an overall score, but you can use the Graphics test score to compare performance with ray tracing on and off.

# System requirements

| | |
|---|---|
| OS | Windows 10, 64-bit with October 2018 Update (version 1809) |
| PROCESSOR | 1.8 GHz dual-core with SSSE3 support |
| GPU | DirectX 12 with DirectX Raytracing support[9] |
| MEMORY | 4 GB RAM |
| VIDEO MEMORY | 6 GB |
| STORAGE | 0.8 GB free disk space |

[9] To run Port Royal, you must have a graphics card with drivers that support Microsoft DirectX Raytracing. Compatible cards include NVIDIA GeForce RTX series, NVIDIA Quadro RTX series, NVIDIA TITAN X, XP, V, and RTX, and NVIDIA GeForce GTX series cards starting with the GeForce GTX 1060 6GB and all higher models.

# Graphics test

Port Royal is a graphics card benchmark. The test measures graphics card performance with a combination of real-time ray tracing and traditional rendering techniques. Port Royal does not have a CPU test.

The scene features ray traced reflections, shadows (ray traced and shadow mapped), transparent surfaces with ray traced reflections, volumetric lighting, particles, and post-processing effects. The rendering resolution is 2560 × 1440.

## CPU factors

The main role of the CPU in the rendering process is to compose the command lists that the GPU executes. The Port Royal engine is multi-threaded and divides the work between all logical cores.

If the CPU cannot submit work to the GPU fast enough, it will be the limiting factor in the benchmark, effectively invalidating the result of the Graphics test. Port Royal requires a multi-core CPU to run at high frame rates.

The exact point at which the benchmark becomes bound by the CPU depends on the configuration of the system.

# Scoring

3DMark Port Royal produces an overall Port Royal score and a Graphics test score. These scores are the same given that the benchmark is based on a single Graphics test. The scores are rounded to the nearest integer. The better a system's performance, the higher the score.

## Overall Port Royal score

The overall 3DMark Port Royal score is the same as the Graphics test score.

$$S_{3DMark} = S_{graphics}$$

Where:

| | | |
|---|---|---|
| $S_{graphics}$ | = | The Port Royal Graphics score |

## Graphics test scoring

The Graphics Test produces a raw performance result in frames per second. We multiply this result by a scaling constant to produce the Graphics score ($Sgraphics$) as follows:

$$S_{graphics} = C_{graphics} \times Fgt$$

Where:

| | | |
|---|---|---|
| $C_{graphics}$ | = | Scaling constant set to 216 |
| $Fgt$ | = | The average FPS result from the Graphics test |

The scaling constant is used to bring the score in line with traditional 3DMark score levels.

# Port Royal engine

3DMark Port Royal uses a custom engine developed in-house with input from UL Benchmark Development Program members including AMD, Intel, and NVIDIA. We worked especially closely with Microsoft to create a first-class implementation of the DirectX Raytracing API.

Port Royal improves on the Time Spy/Night Raid rendering engine by implementing new effects and integrating DirectX Raytracing.

## CPU side

Since Port Royal does not have a CPU test, the main role of the CPU in the test is to compose command lists for the GPU to execute. The task system allows heavy parallel execution. The rendering— including scene update, visibility evaluation and command list building—is done with multiple CPU threads using one thread per available logical CPU core. This shortens the CPU rendering time and reduces the chance of the CPU becoming a bottleneck.

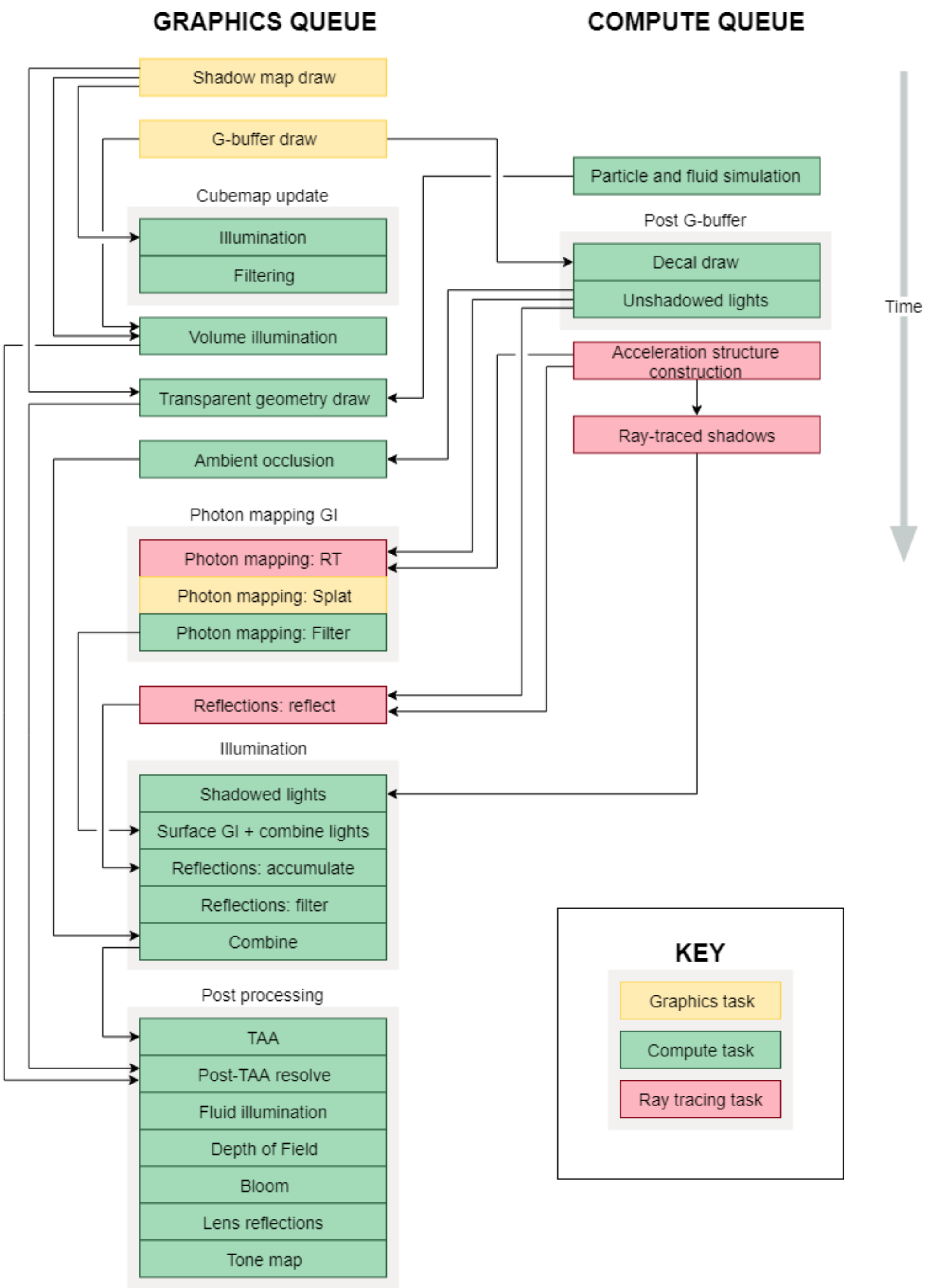
## GPU side

The GPU side of the rendering is composed of multiple rasterization, compute, and ray tracing passes. Some passes run in an asynchronous compute queue.

The engine supports multi-GPU in the form of alternate frame rendering for linked node setups (homogenous adapters).

# Rendering passes

The image below shows the high-level construction of a typical frame in the Port Royal benchmark. Tasks are color-coded by work type. Arrows show task relationships. The position of each task indicates the queue in which the task is executed.



# Shadow map draw

Frustum lights can be shadowed. For each shadowed light, a shadow map is allocated for each frame based on heuristics that determine which resolution is required. The maximum resolution is 1k. The shadow map is used for surface illumination and for generating light shafts for volume illumination.

Shadows are sampled in illumination, cube rendering, and ray tracing shaders.

# G-buffer draw

Opaque objects are drawn into the G-buffer in two passes, separating luminous and non-luminous geometries.

The material system uses physically based materials. The system supports the following material textures: Albedo (RGB) + metalness (A), Normal (RG) + Roughness (B) + Height (A), Luminance, Blend, Opacity, and Light Map. A material might not use all these textures.

The G-buffer is composed of five textures: depth, albedo + metalness, normal + roughness, luminance, and motion vectors for TAA.

# Volume illumination

Volume illumination is computed using the tessellated light volume approach. The volume mesh of a light is computed by extruding the shadow map of a light, using the tessellation pipeline and adaptation heuristics to reduce the amount of mesh data. The fragment shader then computes the volumetric illumination using additive blending to sum up the airlight integral for the view ray corresponding to the pixel.

This is only used for frustum lights, and each fragment computes its own contribution to the airlight integral numerically to include the influence of the radial mask and attenuation of the light. The algorithm is explained in this paper.

This pass is the only pass that uses tessellation in this benchmark. The normal geometry pipeline does not use tessellation.

# Cubemap update

Cubemaps are used to cache the radiance for both perspective-correct and traditional specular reflections for static geometries.

### Illumination

Cubemaps include static geometries that are drawn once into a G-buffer. The illumination of the cubes is dynamically updated in a compute pass similar to the normal surface illumination each frame. The lights are queried from the world-space clusters in the illumination pass. The view ray direction is set to the main camera view direction to match the specular highlights to the screen space illumination texture that is also sampled for reflections.

### Filtering

The mip levels of the cube maps are calculated in compute passes by taking the average of each quad in the lower mip. Each cube is halved in this way until the highest possible mip level is computed.

## Transparent geometry draw

For rendering transparent geometries we use a variant of an order-independent transparency technique called Order-Independent Transparency Approximation with Raster Order Views.  Simply put, transparent geometry is rendered and a per-pixel visibility function (accumulated transparency) is approximated by merging pixels into the compressed function. Then the transparent geometry is re-rendered, illuminated and additively blended according to the visibility function.

## Ambient occlusion

Ambient occlusion uses an adaptive screen-space technique. It is computed using a group of compute shader passes.

## Lighting

All frustum lights, omni lights, reflection probes, and decals are clustered in the world space to a uniform grid. This is done CPU side and then transferred to the GPU in advance.

The main camera lighting is evaluated using a tiled method in multiple, separate passes. Dynamic light evaluation is split into shadowed and unshadowed parts and computed separately.

Before the main illumination passes, asynchronous compute shaders are used to compute screen-space ambient occlusion and calculate unshadowed surface illumination. These tasks are started right after G-buffer rendering has finished and are executed alongside shadow and

environment rendering. Ambient occlusion and unshadowed illumination results are written out to their respective targets.

# Reflections

Reflection rendering is a combination of multiple rendering passes containing cube rendering, ray tracing of reflection rays, reflection sampling of the cubes, and filtering of the reflection result. The illumination of the reflection cubes is updated in a compute pass as explained earlier.

We cast rays to the importance sampled direction for each screen space pixel that is over a roughness threshold. The resulting hitpoints (or one if only a single ray is used per pixel) are stored and the results are used to sample reflections from the environment maps. In cases of pixels being non-visible from the cubes or with mirror like surfaces that require pixel-perfect reflections, we compute the reflection separately to render a correct reflection (re-shade). For reflections of glass, we always run the full shading.

### Reflect

The reflect pass uses the ray tracing pipeline to generate a reflection ray for each pixel above a predetermined roughness threshold. The direction is importance sampled according to the same specular BRDF as used in direct illumination. The hit shader writes the ray length, instance ID, primitive index and barycentric coordinates of the hit. The ray generation shader then stores these into textures.

### Cube sampling

The reflection cubes are used for glossy reflections to find the radiance of a ray intersection generated by the ray tracing pass. The intersection point is reconstructed using the same importance sampling routine as in the reflect pass and reading the ray length stored by the reflect pass. This position is then projected into the reflection cubes. The world space position of the projected point in each cube is tested to determine if it corresponds to the same intersection point, or if it was occluded by another geometry.

In case the point is not found from any cubes or screen space illumination texture, the instance ID, primitive index and barycentric coordinates stored by the reflection pass are read and used to recompute the radiance for the given ray.

If the roughness of the surface is above a certain threshold, the cube sampling is skipped since the resolution is generally not enough for sharp reflections.

### Filtering

Finally, we execute a spatial-temporal filtering pass for the reflection result and combine it with illumination.

# Pre-TAA combine pass

This pass, shown in the GPU task structure as “Combine” inside the “Illumination” block, evaluates the reflection illumination by evaluating a preintegrated specular BRDF, and modulating the reflection filtering results with it. The reflection is then added with surface illumination. Ambient occlusion is also applied here since it has to be before TAA and after the reflection sampling phase as the screen space illumination texture is also sampled there.

# Decals

The Port Royal engine implements a deferred decal system for increased visual quality and easier scene variation.

Decals are skewed prisms that are applied on top of the rendered G-buffer using a compute shader in the asynchronous compute queue. Decals are clustered similarly to lights to speed up the apply pass. For each pixel, active decals are fetched from the matching cluster and applied on top of the G-buffer using one of the implemented blending modes. Various modes allow changing different attributes in the G-buffer (such as normal only or all channels).

# Ray-traced shadows

Ray traced shadows are implemented in a separate pass running in an async compute queue. For each fragment, there is a shadow ray cast from this fragment in the world space towards the direction of the light source. The any-hit shader is then used to detect whether the ray has been occluded on its way from light towards the fragment.

The output of the ray generation shader is shadow modulation map, which is a float32 texture filled with values ranging from 0 to 1. The values are generated per-fragment by dividing the energy flux that has reached this fragment by the total energy flux present in the scene (i.e. from all the lights).

One shadow ray is cast from each fragment towards the light source. Post-process filter is not employed for the shadow mask, so the implementation only supports hard-edged shadows.

# Particles

Particles are simulated on the GPU using the asynchronous compute queue. Simulation work is submitted to the asynchronous queue while G-buffer and shadow map rendering commands are submitted to the main command queue.

### Particle illumination

Particles are rendered as transparent surfaces with approximated visibility.

# Fluids

> ⚠ Fluid simulation is only used in the Port Royal demo. It does not contribute to the Graphics Test score.

### Simulation

Fluids are simulated on the GPU using the asynchronous compute queue. The simulation is based on the Position Based Fluids method. Radix sort is used in each step to order the fluid particles using the Z-order curve to achieve locality of memory access when calculating interactions. Additionally, spatial hashing is used to accelerate the neighbor search.

### Illumination

The liquid surface is constructed in screen-space by splatting ellipsoids, doing most of the computation in vertex shader, and smoothing the result. The illumination of the surface is done in a compute pass after the surface is illuminated, and the surface illumination is used to apply approximate screen space refractions.

# Post-processing

### Temporal anti-aliasing

Temporal anti-aliasing (TAA) is applied for the surface illumination texture that already has reflections applied. The projection matrix used for the G-buffer is jittered for each frame so that the sampled subpixel position varies according to a determined pattern. TAA then blends these subpixel-jittered samples together using exponential average. To fetch a sample from a previous frame, motion vectors written by the G-buffer pass are used. Additionally, variance clipping is used to reduce ghosting.

### Post-TAA resolve

This pass applies parts that do not use TAA on top of the illumination resolved by TAA. Since TAA only applies to opaque objects, transparent elements within the scene such as volumetric illumination, particles and transparent meshes are directly resolved in this pass, on top of the TAA results.

### Depth of field

The effect is computed by scattering the illumination in the out of focus parts of the input image by using multiple passes. First, a compute shader is used to compute confusion radiuses based on depth texture, and splatting primitives are added to a buffer. Then, these primitives are rendered to various resolution textures using the normal rasterization pipeline. Last, the out-of-focus illumination is combined with the original illumination.

### Bloom

Bloom is based on a compute shader FFT that evaluates several effects with one filter kernel. The effects are blur, streaks, anamorphic flare and lenticular halo.

### Lens Reflections

The effect is computed by first applying a filter to the computed illumination in frequency domain like in the bloom effect. The filtered result is then splatted in several scales and intensities on top of the input image using additive blending. The effect is computed in the same resolution as the bloom effect and therefore the forward FFT needs to be performed only once for both effects. The filtering and inverse FFT are performed using the CS and floating point textures.

### Tone mapping

Tone mapping is executed as the last pass of the rendering pipeline. It applies various two-dimensional camera effects (such as vignette) to the final texture and controls the tone reproduction.

## Dynamic Global Illumination: Ray traced photon mapping

> ⚠ Dynamic Global Illumination is only used in the Port Royal demo. It does not contribute to the Graphics Test score.

We have implemented a dynamic global illumination solution using real-time photon mapping. This is a multi-pass algorithm with components of

rasterization, ray tracing and compute work. The main passes of the algorithm are sample generation from reflective shadow maps, photon tracing, photon splatting and irradiance filtering.

# Port Royal version history

| VERSION | | | | NOTES |
|---|---|---|---|---|
| 1.2 | ● | × | × | Added the GPU and monitor selector |
| 1.1 | ● | × | × | Improved multi-GPU support. Scores improve on multi-GPU systems. |
| 1.0 | ● | × | × | Launch version |


06
06

# Night Raid

3DMark Night Raid is a DirectX 12 benchmark for laptops, notebooks, tablets and other mobile computing devices with integrated graphics.

You can also use Night Raid to benchmark and compare the performance of Always Connected PCs, a new category of devices that aim to combine the performance and functionality of a PC, with the all-day battery life, and always-on connectivity of a smartphone.

3DMark Night Raid has native ARM support, which means you can benchmark and compare Always Connected PCs powered by Qualcomm Snapdragon processors.

3DMark Night Raid includes two Graphics tests, a CPU test, and a Demo. The Graphics tests measure GPU performance. The CPU test measures CPU performance. The demo is for entertainment. It does not affect the score.

Scores from Night Raid should not be compared with scores from other 3DMark tests.

Night Raid is only available in the Windows editions of 3DMark.

⚠ Night Raid is a benchmark for PCs with integrated graphics hardware. For testing PCs with discrete graphics cards, you should use Time Spy or Time Spy Extreme.

# Native Support for Windows 10 on ARM

Night Raid has native ARM support for devices with ARM processors.

3DMark Night Raid scores from devices powered by Windows 10 on ARM are comparable with scores from traditional PCs running Windows 10.

On PCs running on Windows 10, the Night Raid CPU Test uses advanced instructions sets, up to AVX2 if supported, and the SSSE3 code path.

On devices running Windows 10 on ARM, the CPU Test uses the NEON instruction set.

# System requirements

| | |
|---|---|
| OS | Windows 10 |
| PROCESSOR | 1.8 GHz dual-core CPU with SSSE3 or NEON support |
| STORAGE | 2 GB free disk space |
| GPU | DirectX 12 |
| VIDEO MEMORY | 1 GB |

> ⚠ Windows 10 64-bit is strongly recommended to run Night Raid. To benchmark on a Windows 10 32-bit system, you need to enable the 3 GB option by running `bcdedit /set IncreaseUserVa 3072` in the Administrator Command Prompt. Reboot the system after the command. To revert, run `bcdedit /deletevalue IncreaseUserVa` in the Administrator Command Prompt.

# Graphics test 1

Graphics tests are designed to stress the GPU while minimizing the CPU workload to ensure that CPU performance is not a limiting factor.

Night Raid Graphics Test 1 uses deferred rendering. The main source of illumination is the shadowed directional light shining in through the windows. There are a few dynamic frustum lights. Unshadowed omni lights contribute to illumination as well. The scene contains tiny, scattered particle systems. Screen-space dynamic reflection and ambient occlusion are enabled. Post-processing effects include lens reflections and bloom.

### Processing performed in an average frame

| | VERTICES | TESSELLATION PATCHES | TRIANGLES | PIXEL SHADER INVOCATIONS[10] | COMPUTE SHADER INVOCATIONS |
|---|---|---|---|---|---|
| NIGHT RAID | 5.4 million | - | 1.8 million | 9.2 million | 9.3 million |

10 This figure is the average number of pixels processed per frame before the image is scaled to fit the native resolution of the device being tested. If the device's display resolution is greater than the test's rendering resolution, the actual number of pixels processed per frame will be even greater.

# Graphics test 2

Graphics tests are designed to stress the GPU while minimizing the CPU workload to ensure that CPU performance is not a limiting factor.

Night Raid Graphics Test 2 uses forward rendering. Tessellated objects appear in almost all frames. There are a few shadowed frustum lights and a small number of point lights. The scene contains large particle systems with depth complexity. Post-processing adds a depth of field effect.

## Processing performed in an average frame

| | VERTICES | TESSELLATION PATCHES | TRIANGLES | PIXEL SHADER INVOCATIONS[11] | COMPUTE SHADER INVOCATIONS |
|---|---|---|---|---|---|
| NIGTH RAID | 2.0 million | 0.032 million | 0.7 million | 19.6 million | 0.3 million |

[11] This figure is the average number of pixels processed per frame before the image is scaled to fit the native resolution of the device being tested. If the device's display resolution is greater than the test's rendering resolution, the actual number of pixels processed per frame will be even greater.

# CPU test

The CPU test measures processor performance. It is designed to stress the CPU while minimizing GPU load to ensure that GPU performance is not a limiting factor.

The Night Raid CPU test features a combination of physics computations and custom simulations.

The simulations require visualization, which can make rendering a bottleneck in some cases. To avoid this, the test only measures the time taken to complete the simulation work. The rendering work in each frame is done before the simulation and doesn't affect the score.

The result of the test is the average simulation time per frame reported in milliseconds. A lower number means better performance.

## CPU instruction sets

On Windows 10 devices, half of the boids systems in the Night Raid CPU use advanced CPU instruction sets, up to AVX2 if supported. The remaining half use the SSSE3 code path. This split makes the test more realistic since games typically have several types of simulation or similar tasks running at once and would be unlikely to use a single instruction set for all of them.

On devices powered by Windows 10 on ARM, the CPU test always uses the NEON instruction set.

## Custom run

With Custom run settings, you can choose which CPU instruction set to use, up to AVX512. The selected set will be used for all boids systems, provided it is supported by the processor under test.

You can evaluate the performance gains of different instruction sets by comparing custom run scores. Note that the choice of set does not affect the physics simulations, which always use SSSE3 and are 15-30% of the workload.

This settings is not available on devices powered by Windows 10 on ARM.

# Scoring

3DMark Night Raid produces an overall Night Raid score, a Graphics test sub-score, and a CPU test sub-score. The scores are rounded to the nearest integer. The better a system's performance, the higher the score.

## Overall Night Raid score

The 3DMark Night Raid score formula uses a weighted harmonic mean to calculate the overall score from the Graphics and CPU test scores.

$$Night\ Raid\ score = floor\left(\frac{1}{\frac{Wgraphics}{Sgraphics} + \frac{Wcpu}{Scpu}}\right)$$

Where:

| | | |
|---|---|---|
| $Wgraphics$ | = | The Graphics score weight, equal to 0.85 |
| $Wcpu$ | = | The CPU score weight, equal to 0.15 |
| $Sgraphics$ | = | Graphics test score |
| $Scpu$ | = | CPU test score |

For a balanced system, the weights reflect the ratio of the effects of GPU and CPU performance on the overall score. Balanced in this sense means the Graphics and CPU test scores are roughly the same magnitude.

For a system where either the Graphics or CPU score is substantially higher than the other, the harmonic mean rewards boosting the lower score. This reflects the reality of the user experience. For example, doubling the CPU speed in a system with an entry-level graphics processor doesn't help much in games since the system is already limited by the GPU. Likewise, for a system with a high-end GPU paired with an underpowered CPU.

## Graphics test scoring

Each Graphics test produces a raw performance result in frames per second (FPS). We take a harmonic mean of these raw results and multiply it by a scaling constant to reach a Graphics score ($Sgraphics$) as follows:

$$S_{graphics} = floor\left(C_{graphics} \times \frac{2}{\frac{1}{Fgt1} + \frac{1}{Fgt2}}\right)$$

Where:

| | | |
|---|---|---|
| $C_{graphics}$ | = | Scaling constant set to 208.33 |
| $Fgt1$ | = | The average FPS result from Graphics test 1 |
| $Fgt2$ | = | The average FPS result from Graphics test 2 |

The scaling constant is used to bring the score in line with traditional 3DMark score levels.

## CPU test scoring

The Night Raid CPU test performs rendering and simulation, but only the simulation time affects the score. The time is measured for Bullet Physics and boid simulations, from start to finish of all simulations. Task priorities are set so that only simulations are executed when measuring time, thus eliminating other factors except the minor overhead of the task system.

Note that on systems with integrated GPUs the rendering will affect simulation time due to shared resources. On systems with discrete GPUs rendering should not affect scores except marginally.

$$Scpu = floor\,(\frac{T_{Reference} \times S_{Reference}}{T_{Simulation}})$$

Where:

| | | |
|---|---|---|
| $T_{Reference}$ | = | Reference time constant set to 115 |
| $S_{Reference}$ | = | Reference score constant set to 5,000 |
| $S_{Reference}$ | = | The average simulation time per frame |

The scaling constant is used to bring the score in line with traditional 3DMark score levels.

# Night Raid engine

3DMark Night Raid uses a DirectX 12 graphics engine that is optimized for integrated graphics hardware. The engine was developed in-house with input from members of the UL Benchmark Development Program.

## Engine features

### Multi-threading

The rendering, including scene update, visibility evaluation, and command list building, is done with multiple CPU threads using one thread per available logical CPU core. This reduces CPU load by utilizing multiple cores.

### Multi-GPU support

The engine implements multi-GPU support using explicit alternate frame rendering on linked-node configuration. Heterogeneous adapters are not supported.

### Visibility solution

The Umbra occlusion library (version 3.3.17 or newer) is used to accelerate and optimize object visibility evaluation for all cameras, including the main camera and light views used for shadow map rendering. The culling runs on the CPU and does not consume GPU resources.

### Descriptor heaps

One descriptor heap is created for each descriptor type when the scene is loaded. Hardware Tier 1 is sufficient for containing all the required descriptors in the heaps.

### Resource heaps

Implicit resource heaps are used for most resources. Explicitly created heaps are used for some resources to reduce memory consumption by placing resources that are not needed at the same time on top of each other.

### Asynchronous compute

Asynchronous compute is used heavily to overlap multiple rendering passes for maximum utilization of the GPU. Async compute workload per frame varies between 10-20%. The forward-rendering path uses less async compute as there are fewer compute passes to run along the shadow map and G-buffer passes.

### Tessellation

The engine supports Phong tessellation and displacement-map-based detail tessellation.

Tessellation factors are adjusted to achieve the desired edge length for the output geometry on the render target (G-buffer, shadow map or other). For shadow maps, edge length is also calculated from the main camera to reduce aliasing due to different tessellation factors between the main camera and shadow map camera.

Additionally, patches that are back-facing and patches that are outside of the view frustum are culled by setting the tessellation factor to zero.

Tessellation is turned entirely off by disabling hull and domain shaders when the size of an object's bounding box on the render target drops below a given threshold.

If an object has several geometry LODs, tessellation is used on the most detailed LOD.

## Deferred rendering

Graphics Test 1 uses a deferred rendering pipeline. Objects are first rendered into a G-buffer that contains all the geometry attributes that are required for the illumination. Illumination is computed in multiple passes and the final result is blended with transparents and fed to the post-processing stages.

### Geometry rendering

Objects are rendered in two steps depending on the attributes of the geometries. First, all non-transparent objects are drawn into the G-buffer. In the second step, transparent objects are rendered using an order-independent transparency algorithm to another target, which is then resolved on top of surface illumination later on.

Geometry rendering uses a LOD system to reduce the number of vertices and triangles for objects that are far away. This also results in bigger on-screen triangle size.

The material system uses physically based materials. The system supports the following material textures: Albedo (RGB) + metalness (A), Roughness (R) + Cavity (G), Normal (RG), Ambient Occlusion (R), Displacement, Luminance, Blend, and Opacity. A material might not use all these textures.

### Opaque objects

Opaque objects are rendered directly to the G-buffer. The G-buffer is composed of textures for Depth, Normal, Albedo, Material Attributes, and Luminance. A material might not use all these textures.

### Transparent objects

When rendering transparent geometries, the engine uses a technique called "Weighted Order-Independent Transparency" (McGuire & Bavoil, 2013). The technique only requires two render targets and the special blending settings to achieve a good approximation of real transparency. Transparents are blended on top of the final surface illumination.

### Illumination

Lighting is evaluated using a tiled method in multiple separate passes.

Before the main illumination passes, asynchronous compute shaders are used to cull lights, compute screen-space ambient occlusion and evaluate unshadowed illumination. These tasks are started right after G-buffer rendering has finished and are executed alongside shadow rendering. All omni-lights are culled to small tiles (16x16 pixels) and written to an intermediate buffer. Frustum lights and environment cubes are culled for every pixel, because there are only a couple of them. Ambient occlusion and unshadowed illumination results are written out to their respective textures.

Illumination for shadowed lights is calculated after the completion of the shadow map rendering. This is also written out to its respective texture.

These results are combined in the global illumination pass while adding probe-based global illumination for objects that do not use light maps.

Reflection illumination is evaluated for the opaque surfaces by combining Screen Space Reflections (SSR) and sampling the precomputed reflection cubes for those surfaces that are rough (above a fixed threshold). Reflections are blended into the illumination in the SSR combination pass.

Final illumination is passed into post-processing.

## Forward rendering

Graphics Test 2 uses a forward rendering pipeline.

In forward rendering mode the geometry is rendered in the same order as in the deferred mode. The same input textures are used and the illumination is computed similarly. The difference is that the outputs do not contain all material information, but rather the results of the illumination which is done in the same pixel shader. There is only one color render

target where the illumination information is stored and a depth target which is used for post-processing effects. There is no depth pre-pass. All the lights in the scene are iterated and there is no culling step.

## Particles

Particles are simulated on the GPU using the asynchronous compute queue. Rendering is performed using indirect draw calls with inputs coming from the simulation buffers.

### Particle simulation

Simulation is executed with multiple compute shader passes in the asynchronous queue alongside shadow map rendering. The following steps are executed per frame for each particle system:

- Alive count of particles is cleared
- New particles are emitted
- Particles are simulated
- Particles that are alive are counted and the count is written into a buffer that is used as indirect argument buffer in the draw phase.

### Particle illumination

Particles can be illuminated with scene lights or they can be self-illuminated. The output buffers of the GPU light culling pass are used as inputs for illuminated particles. The illuminated particles are drawn without tessellation and they are illuminated in either the vertex or pixel shader. Particles are blended together with the same order-independent technique as transparent geometries.

## Post-processing

### Depth of field

The effect is based on a separable blur filter that is used to create an out-of-focus texture in the following manner.

1. Circle of confusion radius is computed for all screen pixels based on the half-resolution depth. Output texture is obtained by multiplying the illumination with the corresponding radii. Average radius is stored to output alpha channel.
2. The result of the previous step is blurred in two passes using a separable filter and two work textures so that we get hexagonal bokehs when the outputs are combined.
3. Upon summing the work textures together in the combination step, they are divided by the stored average radii to renormalize the illumination.

4. The final result is obtained by linearly interpolating between the original illumination and the out-of-focus illumination based on the radius calculated from the full-resolution depth.

## Bloom

Bloom is based on a compute shader FFT that evaluates several effects with one filter kernel. The effects are blur, streaks, anamorphic flare and lenticular halo. Bloom is computed in half resolution to make it faster.

## Lens Reflections

The effect is computed by first applying a filter to the computed illumination in frequency domain like in the bloom effect. The filtered result is then splatted in several scales and intensities on top of the input image using additive blending. The effect is computed in the same resolution as the bloom effect and therefore the forward FFT needs to be performed only once for both effects. The filtering and inverse FFT are performed using compute shaders.

# Night Raid version history

| VERSION | | | | NOTES |
|---|---|---|---|---|
| 1.1 | ● | × | × | Added the GPU and monitor selector |
| 1.0 | ● | × | × | Launch version |

# Wild Life

3DMark Wild Life is a cross-platform benchmark for notebook computers, tablets and smartphones. Use 3DMark Wild Life to test and compare the GPU performance of the latest notebook computers, tablets and smartphones.

Wild Life uses the Vulkan API on Android devices and Windows PCs. On iOS devices, it uses Metal.

Wild Life scores can be compared across Windows, Android and iOS.

On Windows, Wild Life is only available in the 3DMark Advanced Edition and Professional Edition. For Android, download the 3DMark app from Google Play. For iOS, download the standalone app from the Apple App Store.

⚠ As Wild Life is designed for benchmarking mobile devices, it does not have multi-GPU support. It runs on a single GPU.

# System requirements

| | |
|---|---|
| OS | Windows 10, 64-bit |
| PROCESSOR | 1.8 GHz dual-core CPU |
| GPU | Compatible with Vulkan 1.1 |
| MEMORY | 4 GB RAM |
| STORAGE | 6 GB free disk space |

⚠ Wild Life supports Windows 10, 64-bit. For testing devices on Windows 10 on ARM, you should use Night Raid instead.

# Wild Life

| | |
|---|---|
| RENDERING RESOLUTION | 2560 × 1440 |
| GPU MEMORY BUDGET | 1024 MB |

Use Wild Life to test and compare the graphics performance of the latest smartphones, tablets and notebooks. The test uses a 2560 × 1440 rendering resolution before scaling the content to the display.

**Wild Life Unlimited mode**

Use Wild Life Unlimited mode to make chip-to-chip comparisons. In Unlimited mode, the test renders exactly the same frames in every run on every device. The display is updated with frame thumbnails every 100 frames to show progress.

Use Unlimited mode to make chip-to-chip comparisons without vertical sync, display resolution scaling and other operating system factors affecting the result.

# Graphics test

Wild Life is a GPU benchmark that runs for one minute.

On Windows, the benchmark shows the advanced rendering techniques and post-processing effects that are possible on lightweight notebooks when using the Vulkan 1.1 graphics API. The test uses a 2560 × 1440 rendering resolution before scaling the content to the display.

The Wild Life graphics test consists of multiple scenes with variations in the amount of geometry, lights and post-processing effects. The backbone of the test is a deferred renderer with clustered light culling. The scenes utilize transparent geometry in the form of clouds, dust and bright self-illuminated particles. The main source of illumination is the shadowed directional light. Unshadowed omni lights contribute to illumination as well, while a single non-shadowed frustum light is used in a few scenes. The post-processing effects include bloom, heat distortion, volume illumination and depth of field.

# Scoring

3DMark Wild Life produces an overall Graphics score and a Graphics test score. The overall Graphics scores are rounded to the nearest integer.

Wild Life renders a demanding game-like scene in real time. The faster the scene runs, the higher your benchmark score.

## Overall Wild Life Graphics score

The overall 3DMark Wild Life score is based on the Graphics test score.

$$S_{3DMark} = F_{Graphics} * C_{Reference}$$

Where:

$F_{Graphics}$ = The average frame rate across the timeline

$C_{Reference}$ = Scaling constant set to 167

## Graphics test scoring

The Graphics Test score is a raw performance result in frames per second.

$$F_{Graphics} = \frac{S_{Frames}}{T_{Timeline}}$$

Where:

$S_{Frames}$ = The amount of rendered frames ignoring frames from the first and last second in the timeline

$T_{Timeline}$ = The timeline runtime in seconds minus two seconds for the ignored first and last second.

# Wild Life engine

3DMark Wild Life uses a cross-platform graphics engine optimized for mobile devices and lightweight notebooks. The engine was developed in-house with input from members of the [UL Benchmark Development Program](#).

The rendering, including scene update, visibility evaluation and command recording is done with multiple CPU threads using one thread per available logical CPU core. The purpose is to reduce CPU load by utilizing multiple cores.

## Graphics features

### Clustered Light Culling

The scene lights are culled and stored in a three-dimensional screen space grid. The light culling is done using the CPU before the rendering passes.

### Geometry Rendering

Opaque objects are rendered using a deferred rendering method in the graphics pipeline using temporary G-Buffer targets for PBR material parameters. The shading is done using the clustered light information in linear HDR color space utilizing temporary G-Buffer data. In addition to the final lighting result, the deferred rendering pass outputs depth information for other subsequent rendering effects.

Transparent objects are rendered in an order-independent way using two passes. The first pass uses temporary targets to store accumulated color contribution and transparency weighted based on linear depth and the total transparency. The second pass calculates the final color from the accumulated color and transparency. The result of the transparent objects pass is blended on top of the final surface illumination.

Environment reflections are based on a single cube map. Geometry shaders and tessellation are not supported.

### Particles

Particles are simulated on the GPU using compute shaders. The particles are self-illuminated. The particles are rendered at the same time with transparent geometries using the same order-independent technique.

## Post-Processing

### Heat Distortion

The heat distortion effect is generated with the use of particles. For particles that generate the effect, a distortion field is rendered to a texture using a 3D

noise texture as input. This field is then used to distort the input image in post processing phase.

### Bloom

Bloom is based on a compute shader FFT that evaluates several effects with one filter kernel. The effects are blur, streaks, anamorphic flare and lenticular halo. Bloom is computed in reduced resolution to make it faster.

### Volume illumination

Volume illumination is computed by approximating the light scattered towards the viewer by the medium between eye and the visible surface on each lit pixel. The approximation is based on volume ray casting and simple scattering and attenuation model. One ray is cast on each lit pixel for each light. The cast ray is sampled at several depth levels. The achieved result is blurred before combining the volume illumination with the surface illumination.

### Depth of Field

The depth of field effect is computed by filtering rendered illumination in half resolution with three separable skewed box filters that form hexagonal bokeh pattern when combined. The filtering is performed in two passes that exploit similarities in the three filters to avoid duplicate work.

The first pass renders to two render targets and the second pass the one target combining results of the three filters. Before filtering, a circle of confusion radius is evaluated for each pixel and the illumination is premultiplied with the radius.

After filtering, illumination is reconstructed by dividing the result with the radius. This makes the filter gather out of focus illumination and prevents it from bleeding in focus illumination to neighbor pixels.

# Wild Life version history

| VERSION | | | | NOTES |
|---|---|---|---|---|
| 1.0 | ● | ● | ● | Launch version |

# Fire Strike

Fire Strike is a DirectX 11 benchmark for gaming PCs. Fire Strike includes two graphics tests, a physics test and a combined test that stresses both the CPU and GPU.

3DMark Advanced and Professional Editions include Fire Strike Extreme and Fire Strike Ultra, two benchmarks designed for high-end systems with multiple GPUs (SLI / Crossfire).

Scores from 3DMark Fire Strike, Fire Strike Extreme and Fire Strike Ultra should not be compared to each other - they are separate tests with their own scores, even though they share the same content.

Fire Strike benchmarks are only available in the Windows editions of 3DMark.

## Fire Strike

Fire Strike is a DirectX 11 benchmark for gaming PCs. If your system scores less than 2800 in Fire Strike you should run Sky Diver instead.

## Fire Strike Extreme

Fire Strike Extreme is designed for testing PCs with multiple GPUs (minimum 1.5 GB graphics card memory required). It raises the rendering resolution from 1920 ×1080 to 2560 ×1440 and improves the visual quality.

## Fire Strike Ultra

Fire Strike Ultra is a dedicated test for 4K gaming. It raises the rendering resolution to 3840 × 2160 (4K UHD), four times larger than 1080p. A 4K monitor is not required, but your graphics card must have at least 3GB of memory.

# System requirements

| | FIRE STRIKE | FIRE STRIKE EXTREME | FIRE STRIKE ULTRA |
|---|---|---|---|
| OS[12] | Windows 7 or later | Windows 7 or later | Windows 7 or later |
| PROCESSOR | 1.8 GHz dual-core Intel or AMD CPU | 1.8 GHz dual-core Intel or AMD CPU | 1.8 GHz dual-core Intel or AMD CPU |
| STORAGE | 6 GB free space | 6 GB free space | 6 GB free space |
| GPU | DirectX 11 | DirectX 11 | DirectX 11 |
| FOR SYSTEMS WITH INTEGRATED GRAPHICS | 3 GB RAM | 5.5 GB RAM | 7 GB RAM |
| FOR SYSTEMS WITH A DISCRETE GRAPHICS CARD | 2 GB RAM<br>1 GB video card memory | 4 GB RAM<br>1.5 GB video card memory | 4 GB RAM<br>3 GB video card memory |

[12] Windows 7 users must install Service Pack 1.

# Default settings

| | FIRE STRIKE | EXTREME | ULTRA |
|---|---|---|---|
| RESOLUTION | 1920 × 1080 | 2560 × 1440 | 3840 × 2160 |
| GPU MEMORY BUDGET | 1 GB | 1.5 GB | 3 GB |
| TESSELLATION DETAIL | Medium | High | High |
| SURFACE SHADOW SAMPLE COUNT | 8 | 16 | 16 |
| SHADOW MAP RESOLUTION | 1024 | 2048 | 2048 |
| VOLUME ILLUMINATION QUALITY | Medium | High | High |
| PARTICLE ILLUMINATION QUALITY | Medium | High | High |
| AMBIENT OCCLUSION QUALITY | Medium | High | High |
| DEPTH OF FIELD QUALITY | Medium | High | High |
| BLOOM RESOLUTION | 1/4 | 1/4 | 1/4 |

# Graphics test 1

3DMark Fire Strike Graphics test 1 focuses on geometry and illumination. Particles are drawn at half resolution and dynamic particle illumination is disabled. There are 100 shadow casting spot lights and 140 non-shadow casting point lights in the scene. Compute shaders are used for particle simulations and post processing. Pixel processing is lower than in Graphics test 2 as there is no depth of field effect.

### Processing performed in an average frame

| | VERTICES | TESSELLATION PATCHES | TRIANGLES | PIXELS[13] | COMPUTE SHADER INVOCATIONS |
|---|---|---|---|---|---|
| FIRE STRIKE | 3.9 million | 500,000 | 5.1 million | 80 million | 1.5 million |
| FIRE STRIKE EXTREME | 3.9 million | 560,000 | 9.9 million | 150 million | 3.4 million |
| FIRE STRIKE ULTRA | 3.7 million | 650,000 | 12.4 million | 330 million | 3.4 million |

13 This figure is the average number of pixels processed per frame before the image is scaled to fit the native resolution of the device being tested. If the device's display resolution is greater than the test's rendering resolution, the actual number of pixels processed per frame will be even greater.

# Graphics test 2

3DMark Fire Strike Graphics test 2 focuses on particles and GPU simulations. Particles are drawn at full resolution and dynamic particle illumination is enabled. There are two smoke fields simulated on GPU. Six shadow casting spot lights and 65 non-shadow casting point lights are present. Compute shaders are used for particle and fluid simulations and for post processing steps. Post processing includes a depth of field effect.

### Processing performed in an average frame

| | VERTICES | TESSELLATION PATCHES | TRIANGLES | PIXELS[14] | COMPUTE SHADER INVOCATIONS |
|---|---|---|---|---|---|
| FIRE STRIKE | 2.6 million | 240,000 | 5.8 million | 170 million | 8.1 million |
| FIRE STRIKE EXTREME | 3.9 million | 260,000 | 12.9 million | 400 million | 10.4 million |
| FIRE STRIKE ULTRA | 6.0 million | 260,000 | 17.6 million | 1100 million | 10.4 million |

[14] This figure is the average number of pixels processed per frame before the image is scaled to fit the native resolution of the device being tested. If the device's display resolution is greater than the test's rendering resolution, the actual number of pixels processed per frame will be even greater.

# Physics test

3DMark Fire Strike Physics test benchmarks the hardware's ability to run gameplay physics simulations on the CPU. The GPU load is kept as low as possible to ensure that only the CPU is stressed. The Bullet Open Source Physics Library is used as the physics library for the test.

The test has 32 simulated worlds. One thread per available logical CPU core is used to run simulations. All physics are computed on CPU with soft body vertex data updated to GPU each frame.

# Combined test

3DMark Fire Strike Combined test stresses both the GPU and CPU simultaneously. The GPU load combines elements from Graphics test 1 and 2 using tessellation, volumetric illumination, fluid simulation, particle simulation, FFT based bloom and depth of field.

The CPU load comes from the rigid body physics of the breaking statues in the background. There are 32 simulation worlds running in separate threads each containing one statue decomposing into 113 parts. Additionally there are 16 invisible rigid bodies in each world except the one closest to camera to push the decomposed elements apart. The simulations run on one thread per available CPU core.

The 3DMark Fire Strike Combined test uses the Bullet Open Source Physics Library.

### Processing performed in an average frame

| | VERTICES | TESSELLATION PATCHES | TRIANGLES | PIXELS[15] | COMPUTE SHADER INVOCATIONS |
|---|---|---|---|---|---|
| FIRE STRIKE | 7.5 million | 530,000 | 7.9 million | 150 million | 110 million |
| FIRE STRIKE EXTREME | 9.2 million | 540,000 | 14.8 million | 390 million | 110 million |
| FIRE STRIKE ULTRA | 10.8 million | 540,000 | 19.6 million | 960 million | 120 million |

[15] This figure is the average number of pixels processed per frame before the image is scaled to fit the native resolution of the device being tested. If the device's display resolution is greater than the test's rendering resolution, the actual number of pixels processed per frame will be even greater.

# Scoring

Scores from different benchmarks should not be compared to each other. Fire Strike, Fire Strike Extreme, and Fire Strike Ultra are separate tests with their own scores, even though they share the same content.

## Overall Fire Strike score

The 3DMark Fire Strike score formula uses a weighted harmonic mean to calculate the overall score from the Graphics, Physics, and Combined scores.

$$Fire\ Strike\ score = \frac{Wgraphics + Wphysics + Wcombined}{\frac{Wgraphics}{Sgraphics} + \frac{Wphysics}{Sphysics} + \frac{Wcombined}{Scombined}}$$

Where:

| | | |
|---|---|---|
| $Wgraphics$ | = | The Graphics score weight, equal to 0.75 |
| $Wphysics$ | = | The Physics score weight, equal to 0.15 |
| $Wcombined$ | = | The Combined score weight, equal to 0.10 |
| $Sgraphics$ | = | Graphics score |
| $Sphysics$ | = | Physics score |
| $Scombined$ | = | Combined score |

For a balanced system, the weights reflect the ratio of the effects of GPU and CPU performance on the overall score. Balanced in this sense means the Graphics, Physics and Combined scores are roughly the same magnitude.

For a system where either the Graphics or Physics score is substantially higher than the other, the harmonic mean rewards boosting the lower score. This reflects the reality of the user experience. For example, doubling the CPU speed in a system with an entry-level graphics card doesn't help much in games since the system is already limited by the GPU. Likewise for a system with a high-end graphics card paired with an underpowered CPU.

## Graphics score

Each Graphics test produces a raw performance result in frames per second (FPS). We take a harmonic mean of these raw results and multiply it by a scaling constant to reach a Graphics score ($Sgraphics$) as follows:

$$Sgraphics = 230 \times \frac{2}{\frac{1}{Fgt1} + \frac{1}{Fgt2}}$$

Where:

| | | |
|---|---|---|
| $Fgt1$ | = | The average FPS result from Graphics test 1 |
| $Fgt2$ | = | The average FPS result from Graphics test 2 |

The scaling constant is used to bring the score in line with traditional 3DMark score levels.

## Physics score

$$Sphysics = 315 \times Fphysics$$

Where:

| | | |
|---|---|---|
| $Fphysics$ | = | The average FPS result from the Physics Test |

The scaling constant is used to bring the score in line with traditional 3DMark score levels.

## Combined score

$$Scombined = 215 \times Fcombined$$

Where:

| | | |
|---|---|---|
| $Fcombined$ | = | The average FPS result from the Combined Test |

The scaling constant is used to bring the score in line with traditional 3DMark score levels.

# Fire Strike engine

Fire Strike benchmarks require graphics hardware with full DirectX 11 feature level 11 support.

## Multithreading

The multithreading model is based on DX11 deferred device contexts and command lists. The engine utilizes one thread per available logical CPU core. One of the threads is considered as the main thread, which uses both immediate device context and deferred device context. The other threads are worker threads, which use only deferred device contexts.

Rendering workload is distributed between the threads by distributing items (e.g. geometries and lights) in the rendered scene to the threads. Each thread is assigned roughly equal amount of scene items.

When rendering a frame, each thread does the work associated to items assigned to the thread. That includes, for example, computation of transformation matrix hierarchies, computation of shader parameters (constants buffer contents and dynamic vertex data) and recording of DX API calls to a command list. When the main thread is finished with the tasks associated to its own items, it executes the command lists recorded by worker threads.

## Tessellation

The engine supports rendering with and without tessellation. The supported tessellation techniques are PN Triangles, Phong, and displacement map based detail tessellation. Both triangle and quad based tessellation is supported.

Tessellation factors are adjusted to achieve desired edge length for output geometry on the render target. Additionally, patches that are back facing and patches that are outside of the view frustum are culled by setting the tessellation factor to zero.

Tessellation is turned entirely off by disabling hull and domain shaders when size of object's bounding box on render target drops below a given threshold. This applies both to g-buffer and shadow map drawing.

## Lighting

Lighting is done in deferred style. Geometry attributes are first rendered to a set of render targets. Ambient occlusion is then computed from depth and normal data. Finally illumination is rendered based on those attributes.

## Surface illumination

Two different surface shading models and g-buffer compositions are supported. The more complex model uses four textures and depth texture as the g-buffer. The simpler model uses two textures and depth texture.

Surface illumination model is either combination of Oren-Nayar diffuse reflectance and Cook-Torrance specular reflectance or basic Blinn Phong reflectance model. Simple surface shading model is used on Feature Level 10 demo and tests while the complex model is used on Feature Level 11 demo and tests. Optionally atmospheric attenuation is also computed.

Horizon based screen space ambient occlusion can be applied to the surface illumination.

Point, spot and directional lights are supported. Spot and directional lights can be shadowed. For spot lights, shadow texture size is selected based on size of the light volume in screen space. Shadow maps are sampled using best candidate sample distribution. Sample pattern is dithered with 4 × 4 pixel pattern.

## Volumetric illumination

The renderer supports volume illumination. It is computed by approximating the light scattered towards the viewer by the medium between eye and the visible surface on each lit pixel. The approximation is based on volume ray casting and the Rayleigh-Mie scattering and attenuation model.

One ray is cast on each lit pixel for each light. The cast ray is sampled at several depth levels. Sampling quality is improved by dithering sampling depths with a 4 × 4 pixel pattern. The achieved result is blurred to combine the different sampling depths on neighboring pixels before combining the volume illumination with the surface illumination.

When rendering illumination, there are two high dynamic range render targets. One is for surface illumination and the other for volume illumination.

## Particle illumination

Particle effects are rendered on top of opaque surface illumination with additive or alpha blending. Particles are simulated on the GPU. Particles can be either simply self-illuminated or receive illumination from scene lights.

Lights that participate in particle illumination can be individually selected. To illuminate particles, the selected lights are rendered to three volume

textures that are fitted into view frustum. The textures contain incident radiance in each texel stored as spherical harmonics. Each of the three textures holds data for one color channel storing four coefficients. Incident radiance from each light is rendered to these volume textures as part of light rendering.

When rendering illuminated particles, hull and domain shaders are enabled. Incident radiance volume texture sampling is done in the domain shader. Tessellation factors are set to produce fixed size triangles in screen pixels. Tessellation is used to avoid sampling incident radiance textures in the pixel shader.

Particles can cast shadows on opaque surface and on other particles. For generating particle shadows, particle transmittance is first rendered to a 3D texture. The transmittance texture is rendered from the shadow casting light like a shadow map. After particles have been rendered to the texture, an accumulated transmittance 3D texture is generated by accumulating values of each depth slice in the transmittance texture. The accumulated transmittance texture can then be sampled when rendering illumination or incident radiance that is used to illuminate particles.

# Post-processing

## Particle based distortion

Particles can be used to generate a distortion effect. For particles that generate the effect, a distortion field is rendered to a texture using a 3D noise texture as input. This field is then used to distort the input image in post processing phase.

## Depth of field

The effect is computed using the following procedure:

1. Circle of confusion radius is computed for all screen pixels and stored in a full resolution texture.
2. Half and quarter resolution versions are made from the radius texture and the original illumination texture.
3. Positions of out-of-focus pixels whose circle of confusion radius exceeds a predefined threshold are appended to a buffer.
4. The position buffer is used as point primitive vertex data and, using Geometry Shaders, the image of a hexagon-shaped bokeh is splatted to the positions of these vertices. Splatting is done to a texture that is divided into regions with different resolutions using multiple viewports. First region is screen resolution and the rest are a series of halved regions down to 1x1 texel resolution. The screen space radius of the splatted bokeh determines the used resolution. The larger the radius the smaller the used splatting resolution.
5. Steps 3 and 4 are performed separately for half and quarter resolution image data with different radius thresholds. Larger bokehs are generated from lower resolution image data.
6. The different regions of the splatting texture are combined by up-scaling the data in the smaller resolution regions step by step to the screen resolution region.
7. The out-of-focus illumination is combined with the original illumination.

## Lens reflections

The effect is computed by first applying a filter to the computed illumination in frequency domain like in the bloom effect. The filtered result is then splatted in several scales and intensities on top of the input image using additive blending. The effect is computed in the same resolution as the bloom effect and therefore the forward FFT needs to be performed only once for both effects. As in the bloom effect, the forward and inverse FFTs are performed using the CS and 32bit floating point textures.

## Bloom

The effect is computed by transforming the computed illumination to frequency domain using Fast Fourier Transform (FFT) and applying bloom filter to the input in that domain. An inverse FFT is then applied to the filtered image. The forward FFT, applying the bloom filter and inverse FFT are done with the CS. The effect is computed in reduced resolution. The input image resolution is halved two or three times depending on settings and then rounded up to nearest power of two. The FFTs are computed using 32bit floating point textures. A procedurally pre-computed texture is used as the bloom filter. The filter combines blur, streak, lenticular halo and anamorphic flare effects.

## Anti-aliasing

MSAA and FXAA anti-aliasing methods are supported.

In MSAA method G-buffer textures are multisampled with the chosen sample count. Edge mask is generated based on differences in G-buffer sample values. The mask is used in illumination phase to select for which pixels illumination is evaluated for all G-buffer samples. For pixels that are not considered edge pixels, illumination is evaluated only for the first G-buffer sample. Volume illumination is always evaluated only for the first G-buffer sample due to its low frequency nature.

FXAA is applied after tone mapping making it the final step in post processing.

## Smoke simulation

The implementation of the smoke simulation is based on Ronald Fedkiw's paper "Visual Simulation of Smoke" with the addition of viscous term as in Jos Stam's "Stable Fluids" but without a temperature simulation. Thus the smoke is simulated in a uniform grid where velocity is modeled with incompressible Euler equations. Advection is solved with a semi-Lagrangian method.

Vorticity confinement method is then applied to the velocity field to reinforce vortices. Diffusion and projection is then computed by the Jacobi iteration method. The simulation is done entirely with Compute Shaders. Cylinders that interact with the smoke are implicit objects which are voxelized into the velocity and density field in Compute Shaders.

# Fire Strike version history

| VERSION | | | | NOTES |
|---|---|---|---|---|
| 1.1 | ● | × | × | Fixed issues when benchmarking systems with multiple GPUs. Scores improve significantly on systems with multiple GPUs. |
| 1.0 | ● | × | × | Launch version |

Fire Strike Ultra, added in 3DMark v1.4.775, uses the Fire Strike v1.1.0 workload.

3DMark v2.1.2852, released July 14, 2016, used an incorrect setting for Fire Strike Custom runs that resulted in slightly lower than expected scores. Results from Fire Strike Custom runs using that version should not be compared with any other version of 3DMark. The issue was fixed in 3DMark v2.1.2969 released August 18, 2016. The standard Fire Strike benchmark was not affected, nor were Fire Strike Extreme and Fire Strike Ultra.

# DirectX Raytracing feature test

3DMark feature tests are special tests designed to highlight specific techniques, functions or capabilities.

The DirectX Raytracing feature test measures pure ray-tracing performance. Use this test to compare the performance of the dedicated ray-tracing hardware in the latest graphics cards.

The test also offers an interactive mode that lets you move freely around the scene and take screenshots.

The DirectX Raytracing feature test is available in 3DMark Advanced Edition and 3DMark Professional Edition (annual license only).

# System requirements

| | |
|---|---|
| OS | Windows 10, 64-bit with May 2020 Update (version 2004) |
| PROCESSOR | Dual-core processor |
| GPU | DirectX 12 compatible with driver support for DirectX Raytracing Tier 1.1 |
| VIDEO MEMORY | 6 GB |
| STORAGE | 1 GB |

# Technical details

The 3DMark DirectX Raytracing feature test is designed to make ray-tracing performance the limiting factor. Instead of relying on traditional rendering techniques, the whole scene is ray-traced and drawn in one pass.

Camera rays are traced across the field of view with small random offsets to simulate a depth of field effect. The frame rate is determined by the time taken to trace and shade a set number of samples for each pixel, combine the results with previous samples and present the output on the screen. The rendering resolution is 2560 × 1440.

## DirectX Raytracing Tier 1.1

DirectX Raytracing helps developers create realistic reflections, shadows, and other effects that are difficult to achieve with other techniques.

DirectX Raytracing Tier 1.1 introduces new features and capabilities that improve efficiency and give developers more flexibility and control. While the details of these improvements are beyond the scope of this guide, you can read more about DXR Tier 1.1 on the Microsoft DirectX Developer Blog and in the DirectX Raytracing Functional Spec.

This test uses features from DirectX Raytracing Tier 1.1 to create a realistic ray-traced depth of field effect.

## Rendering

There is a minimal amount of traditional rendering in this test. Instead of drawing a GBuffer or using a rasterizer at all, camera rays are traced in a compute shader with random offsets to simulate a depth of field effect.

To keep light computation to a minimum, image-based lighting is used in addition to a baked light map.

### Sampling

Camera rays are randomized with per-pixel offsets. There are 12 samples for each pixel when running the test with default settings.

When the camera is stationary, samples are accumulated at a rate of 12 samples per pixel per frame. This improves the appearance of the depth of field effect from slightly grainy to smooth over the span of several frames.

When the camera moves, a light motion blur is applied to reduce the noise that is a natural result of this ray-tracing technique.

### Implementation

The test measures the peak ray-traversal performance of the GPU. All other work, such as illumination and post processing, is kept to a minimum.

The ray tracing acceleration structure is built only once. As the scene is static and non-animated, there is no need to update the acceleration structure during the test.

The test casts primary rays only. The rays are approximately sorted by direction on the CPU during the test initialization, which is possible because the sampling pattern in screen space is known beforehand. Generating the optimal ray order during initialization allows more coherent ray traversal for out-of-focus areas without the run-time cost of sorting.

## Settings

### Sample count

The sample count determines how many samples are taken per pixel. Higher sample counts produce better-quality images but are more demanding. The test will run faster with a lower sample count, but the image quality will be grainier. The default value is 12 samples per pixel.

### Rendering device

On systems with multiple GPUs, you can choose which GPU you want to use for the test.

### Display

On systems with multiple displays, you can choose which display you want to use for the test.

⚠ If the chosen rendering device is not connected directly to the selected display then the frame rate may be limited by PCI Express bandwidth and other factors.

## Scoring

The result of the test is the average frame rate in frames per second.

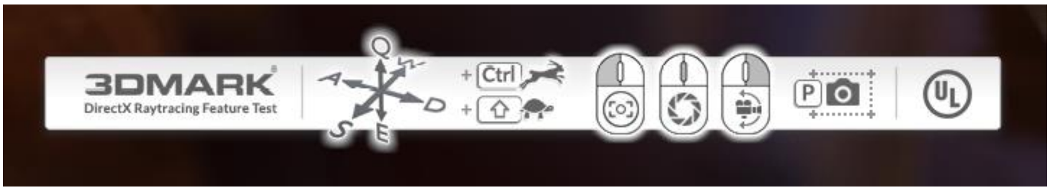
# Interactive mode

The test includes an interactive mode that lets you move freely around the scene and take screenshots. You can control the focus point and aperture of the camera to explore different depth of field effects using ray tracing.

Please note that the motion blur effect is disabled in interactive mode in order to deliver a higher frame rate.

Screenshots are saved to the Pictures\3DMarkDXRFeatureTest folder.

## Controls




| | |
|---|---|
| W, A, S, D | Move the camera forward, left, right, back |
| Q, E | Move the camera up, down |
| +Ctrl | Move faster |
| +Shift | Move slower |
| Left mouse button | Set the focus distance |
| Mouse wheel | Change the camera aperture |
| Right mouse button | Change the camera pitch |
| P | Take a screenshot |

# DirectX Raytracing feature test history

| VERSION | | | | NOTES |
|---|---|---|---|---|
| 1.0 | ● | × | × | Launch version |

# PCI Express feature test

3DMark feature tests are special tests designed to highlight specific techniques, functions or capabilities.

PCI Express (PCIe) is a standard interface that provides high-bandwidth communication between devices in a computer. New PCI Express 4.0 interfaces provide up to twice the bandwidth of PCI Express 3.0. With more bandwidth, games can transfer more data, reduce loading times, and support more complex scenes.

The 3DMark PCI Express feature test is designed to measure the bandwidth available to your GPU over your computer's PCI Express interface. This is a synthetic test. In real-world use, your PC's gaming performance is extremely unlikely to be limited by PCIe bandwidth.

The PCI Express feature test is available in 3DMark Advanced Edition and 3DMark Professional Edition (annual license only).

# System requirements

| | |
|---|---|
| OS | Windows 10, 64-bit |
| PROCESSOR | Dual-core processor |
| GPU | DirectX 12 dedicated graphics card |
| VIDEO MEMORY | 4 GB |
| MEMORY | 12 GB RAM |
| STORAGE | 170 MB |

⚠ You need a DirectX 12 compatible discrete graphics card to run the 3DMark PCI Express feature test. The test will not run on systems that have only integrated graphics. In multi-GPU systems, the test will run on the primary GPU. External GPU enclosures are not supported.

# Technical details

The 3DMark PCI Express feature test aims to make bandwidth the limiting factor for performance.

For each frame, the test uploads a large amount of vertex and texture data to the GPU. The goal is to transfer enough data to saturate the PCIe 4.0 interface. The geometry streaming load increases from 2,400,000 to 9,600,000 vertices per frame as the test runs. The texture streaming load remains constant.

The test uses a fixed time step between frames. This ensures that every system does the same amount of work when running the test.

## Rendering

The 3DMark PCI Express feature test is rendered at 2560 × 1440 resolution.

The test is forward-rendered with simple Blinn-Phong shading. The background is rendered by blending three textures, two of which are uploaded for each frame. The test uses post-processing to add linear fog, tone mapping, and bloom effects.

## Scoring

The result is the average bandwidth achieved during the test.

The amount of bandwidth available over PCI Express depends on your motherboard, your graphics card, the PCIe slot used by your graphics card, your system's BIOS settings, and other factors.

This is a synthetic test. In real-world use, your PC’s gaming performance is extremely unlikely to be limited by PCIe bandwidth.

# PCI Express feature test history

| VERSION | Windows | Android | Apple | NOTES |
|---|---|---|---|---|
| 1.0 | ● | × | × | Launch version |


乐万
圣节

# VRS feature test

3DMark feature tests are special tests designed to highlight specific techniques, functions or capabilities.

Variable-Rate Shading (VRS) is a DirectX 12 feature that allows developers to improve performance by selectively reducing the shading rate in parts of the frame where there will be little noticeable effect on image quality.

The 3DMark VRS feature test helps you compare differences in performance and image quality when using Tier 1 and Tier 2 variable-rate shading. The test also offers an interactive mode for experimenting with different VRS settings and exporting frames for comparison.

The VRS feature test is available in 3DMark Advanced Edition and 3DMark Professional Edition (annual license only).

# System requirements

| | TIER 1 TEST | TIER 2 TEST |
|---|---|---|
| OS | Windows 10 version 1903 or later, 64-bit | Windows 10 version 1903 or later, 64-bit |
| PROCESSOR | Dual-core processor | Dual-core processor |
| GPU | DirectX 12 GPU that supports Tier 1 Variable-Rate Shading | DirectX 12 GPU that supports Tier 2 Variable-Rate Shading |
| VIDEO MEMORY | 1 GB | 3 GB |
| MEMORY | 4 GB | 4GB |
| STORAGE[16] | 720 MB | 720 MB |

⚠ The 3DMark VRS feature test does not support multiple GPUs. Performance is likely to be lower if SLI/Crossfire is enabled.

16 Both Tier 1 and Tier 2 tests are included in the VRS feature test DLC, which is 720 MB in total.

# Variable-Rate Shading

Variable-Rate Shading (VRS) is a feature in DirectX 12 that allows developers to improve performance by selectively reducing the level of detail in parts of the frame where there will be little noticeable effect on image quality.

Shading rate refers to the number of pixel shader operations called for each pixel. Higher shading rates improve accuracy but are more demanding for the GPU. Lower shading rates improve performance at the cost of visual fidelity.

There are two tiers of VRS support. With Tier 1, developers can specify a different shading rate for each draw call. Tier 2 adds more flexibility and control by allowing different shading rates within each draw call.

By using VRS to lower the shading rate for parts of the frame that are in deep shadow, far the camera, or peripheral to the player's focus, for example, a game can run at a higher frame rate with little perceptible loss in visual quality.

For more details, please see the DirectX Developer Blog and the Windows Dev Center.

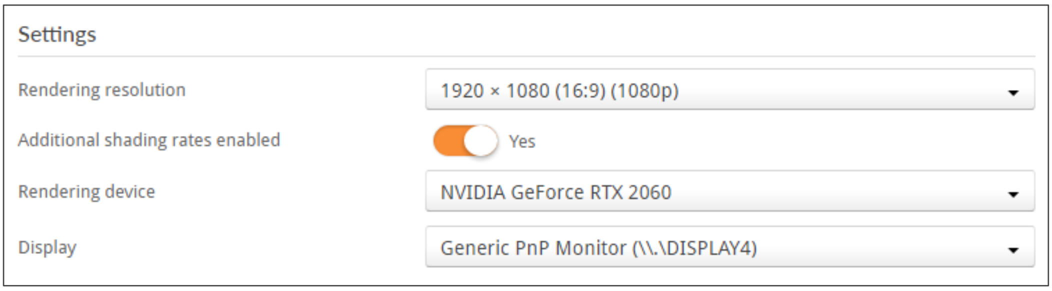
# Technical details

The 3DMark VRS feature test helps you compare differences in frame rate and image quality when using Tier 1 and Tier 2 variable-rate shading. There are separate tests for each tier.

The tests run in two passes to test performance and image quality. VRS is disabled on the first pass to provide a baseline. VRS is enabled for the second pass. The test reports the frame rate for each pass.

## Settings

Each test has additional settings.




### Rendering resolution

The rendering resolution is the resolution that the test uses before scaling the output to your monitor's display resolution. You can change the rendering resolution using the drop-down.

- Default rendering resolution for the Tier 1 test is 1920 × 1080 (1080p).
- Default rendering resolution for the Tier 2 test is 3840 × 2160 (4K).

### Additional shading rates enabled

The DirectX specification for VRS defines a range of shading rates by course pixel size. Support for *additional shading rates*— 2×4, 4×2, and 4×4 —varies across platforms and devices. These shading rates may not be available on some hardware.

On compatible hardware, you can choose whether to use additional shading rates with the Additional shading rates enabled setting.

⚠ The performance gain with VRS is higher when additional shading rates are enabled. When comparing results from different systems, make sure that you compare like-for-like.

In the Tier 1 test, a 1×1 shading rate is used for the foreground and 2×2 for the background. When additional shading rates are enabled, the test will also use 4×4 for the furthest objects in the scene.

In the Tier 2 test, 1×1, 1×2, 2×1, and 2×2 shading rates are used selectively in areas of the frame where there is less detail or contrast. When additional shading rates are enabled, the test will also use 2×4, 4×2, and 4×4.

### Rendering device

On systems with multiple GPUs, you can choose which GPU you want to use for the test.

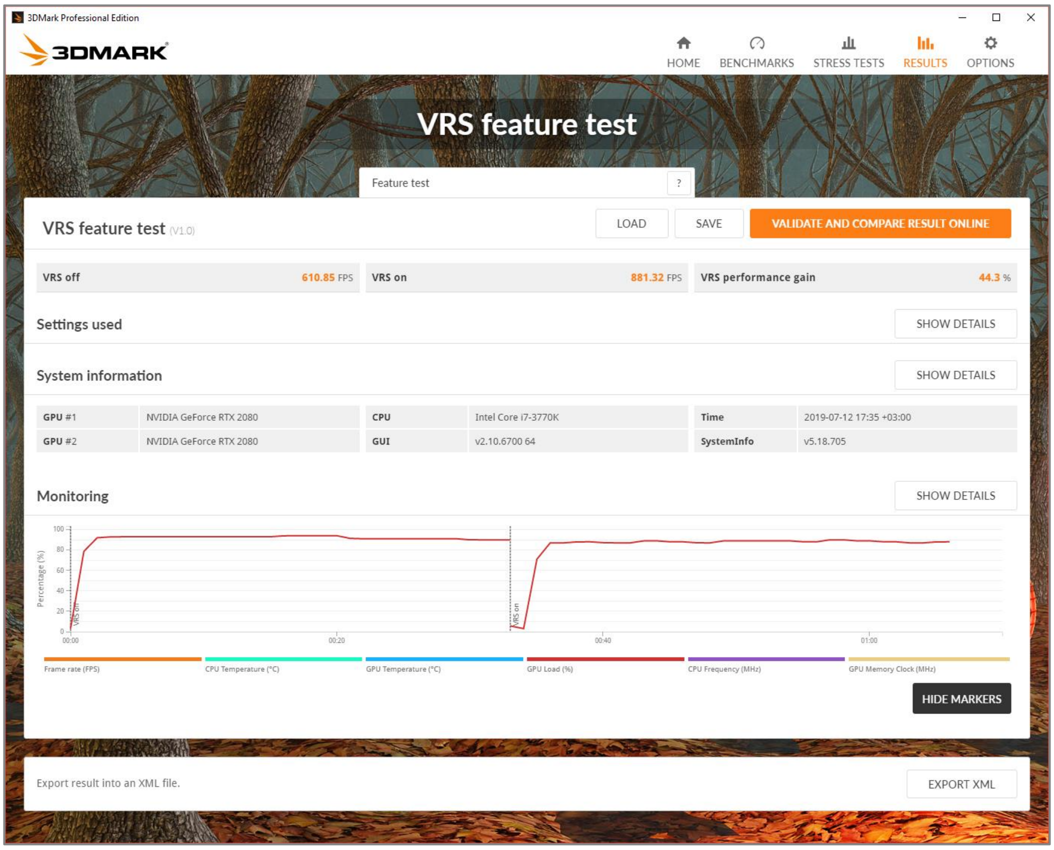
### Display

On systems with multiple displays, you can choose which display you want to use for the test.

> ⚠ If the chosen rendering device is not connected directly to the selected display then the frame rate may be limited by PCI Express bandwidth and other factors.

# Scoring

The test runs in two passes. VRS is disabled on the first pass of the test to provide a baseline for comparison. Variable-Rate Shading is enabled for the second pass. The test then reports the average frame rate for each pass and calculates the performance gained with VRS.

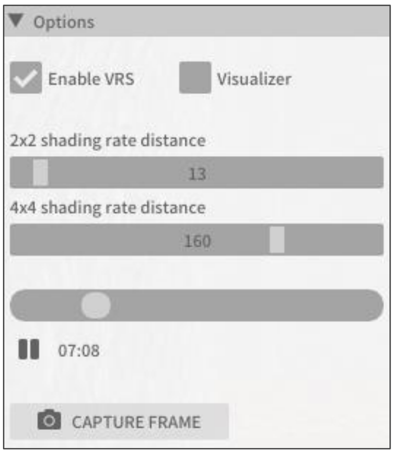


# Interactive mode

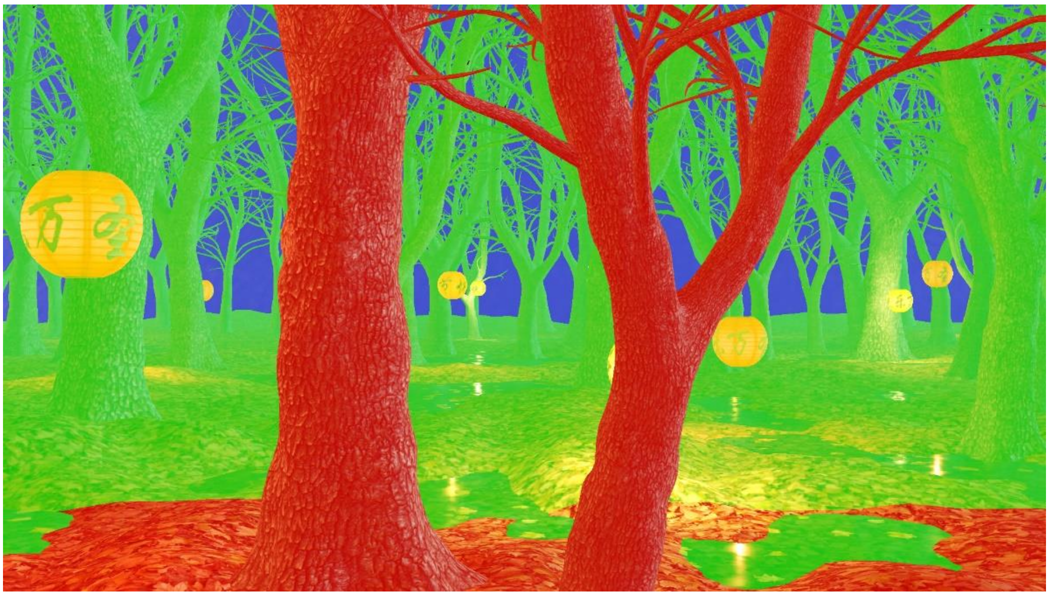
Interactive mode lets you change variable-rate shading settings on the fly to see how they affect the frame rate and image quality.

## Tier 1 Interactive Mode

Use the controls on the overlay to **enable** or **disable VRS**. The **Time** and **Pause** controls let you move through the timeline to specific points. The **Capture frame** button exports the original rendered frame with the current settings for comparison. The captured frames are saved to the Pictures\3DMarkVRSFeatureTest folder.




When VRS is enabled, the shading rate varies with camera distance. You can use the **Visualizer** to see how variable-rate shading is applied in the scene.



| VISUALIZER | SHADING RATE |
|---|---|
| Red | 1 × 1 |
| Green | 2 × 2 |
| Blue | 4 × 4 |

You can change the threshold distance for each shading rate with the sliders. This allows you to experiment with the settings to prioritise performance or visual fidelity.

## Tier 2 Interactive Mode

Use the controls on the overlay to **enable** or **disable VRS**. The **Time** and **Pause** controls let you move through the timeline to specific points. The **Capture frame** button exports the original rendered frame with the current settings for comparison. The captured frames are saved to the Pictures\3DMarkVRSFeatureTest folder.


When VRS is enabled, lower shading rates are used selectively in areas of the frame where there is less detail or contrast. You can use the **Visualizer** to see how variable-rate shading is applied in the scene.

The **Visualizer opacity** slider controls the opacity of the color overlay that shows where each shading rate is used.

| VISUALIZER | SHADING RATE |
|---|---|
| White | 1 × 1 |
| Yellow | 1 × 2 or 2 × 1 |
| Orange | 2 × 2 |

| VISUALIZER | SHADING RATE |
| --- | --- |
| Pink | 2 × 4 or 4 × 2 |
| Purple | 4 × 4 |

The **Quality** setting determines how much variable-rate shading is used.

- Low quality prioritizes performance over visual fidelity.
- Medium quality is the default for the test.
- High quality prioritizes visual fidelity over performance.

### Interactive mode effect on frame rate in the Tier 2 test

The shading rate algorithm used in the Tier 2 test uses data from the previous frame to determine how to apply shading rates in the current frame. Each frame partly informs the next.

When VRS is toggled off in Interactive Mode, this chain of frame-to-frame shading rate data is broken. When VRS is then toggled back on, the next frame is based on a frame that was shaded entirely at the original 1 × 1 rate. While this temporal effect diminishes over subsequent frames, the initial effect is that the frame rate may differ from the rate seen before VRS was toggled off.

In general, it is fine to make FPS comparisons when switching from VRS on to VRS off in Interactive Mode, but not vice-versa due to this temporal effect.

# VRS feature test history

| VERSION | Windows | Android | iOS | NOTES |
|---|---|---|---|---|
| 1.1 | ● | × | × | Added Tier 2 test |
| 1.0 | ● | × | × | Launch version with Tier 1 test |

# NVIDIA DLSS feature test

Feature tests are special tests designed to highlight specific techniques, functions or capabilities.

The NVIDIA DLSS feature test is designed to help you test and compare the performance and image quality of DLSS processing.

DLSS (Deep Learning Super Sampling) is an NVIDIA RTX technology that uses the power of deep learning and AI to improve game performance while maintaining visual quality.

You can choose to run the NVIDIA DLSS feature test using DLSS 2 or DLSS 1. With DLSS 2, you can also choose between three image quality modes—Quality, Performance and Ultra Performance.

The NVIDIA DLSS feature test runs the Port Royal benchmark twice. The first run has DLSS disabled to provide a baseline. The second run uses DLSS processing. The result screen reports the frame rate for each run.

The NVIDIA DLSS feature test is only available in 3DMark Advanced Edition and 3DMark Professional Edition.

⚠ DLSS is a proprietary NVIDIA technology. To run this test, you will need an NVIDIA graphics card with drivers that support DLSS and Microsoft DirectX Raytracing, such as a GeForce RTX series, Quadro RTX series or TITAN RTX. You must also have the latest NVIDIA drivers for your graphics card and the Windows 10 October 2018 Update (1809).

# System requirements

| | |
|---|---|
| OS | Windows 10, 64-bit with October 2018 Update (version 1809) |
| PROCESSOR | 1.8 GHz dual-core with SSSE3 support |
| GPU | DirectX 12 with support for DirectX Raytracing and DLSS[17] |
| VIDEO MEMORY | 6 GB[18] |
| MEMORY | 4 GB RAM |
| STORAGE | 0.8 GB free disk space (shared with Port Royal) |

[17] DLSS is a proprietary NVIDIA technology. To run NVIDIA DLSS feature test, you must have an NVIDIA graphics card that supports DLSS, such as a GeForce RTX series, Quadro RTX series or TITAN RTX. You must also have the latest NVIDIA drivers for your graphics card.

[18] Running the first part of the NVIDIA DLSS feature test with DLSS disabled on 7680 × 4320 (8K) requires at least 10 GB video RAM. If the system does not have the required video RAM space, the test runs only the second part with DLSS enabled.

# Deep Learning Super Sampling

Deep Learning Super Sampling (DLSS) is an NVIDIA RTX technology that uses deep learning and AI to improve game performance while maintaining visual quality.

DLSS uses a deep neural network to extract multidimensional features of the rendered scene and intelligently combine details from multiple frames to construct a high-quality final image. This approach allows NVIDIA RTX GPUs to use fewer samples for rendering and use AI to fill in information to create the final image. The result is a clear, crisp image of a quality similar to traditional rendering but with higher performance.

DLSS 2 is an improved version of DLSS that aims to deliver more performance and image quality than the original DLSS.

## Aliasing

Aliasing—a distracting jagged line on the edge of an object in a scene—is a common artifact in real-time computer graphics.

Increasing the resolution of the entire image is not always practical so a common way to remove the jagged lines is to increase the number of samples on the line which helps to smooth it. Many techniques have been developed which intelligently blend the colors of the jagged edges with the colors of nearby pixels but most of these can lead to a loss of fine detail.

## Temporal Anti-aliasing

The 3DMark Port Royal benchmark uses Temporal Anti-Aliasing (TAA), a popular anti-aliasing technique used in many games today. TAA solves the aliasing problem by accumulating multiple samples temporally—instead of adding more samples to a single frame, it adds a small jitter to a rendered frame, and combines the current samples with matching samples from previous frames. This directly leads to an increased sampling rate. Unfortunately, TAA suffers from flickering and ghosting artifacts. These can be especially seen in dynamic scenes.

## A deep learning solution to aliasing

To develop Deep Learning Super Sampling, NVIDIA researchers trained a neural network to find jagged edges in an image, determine the best color for each pixel, and then apply proper colors to create smoother edges and improved image quality.

DLSS provides high-quality anti-aliasing with fewer artifacts and better performance than other types of anti-aliasing. Compared with TAA, DLSS produces sharp images that are temporally stable.

## Neural network training

The specific details and method of the training is NVIDIA's trade secret since DLSS is a proprietary technology.

The original DLSS version required training the AI network with specific data extracted from each supported title. DLSS 2 does not require any input from the workload to train the network. It delivers a generalized network that can work across games.

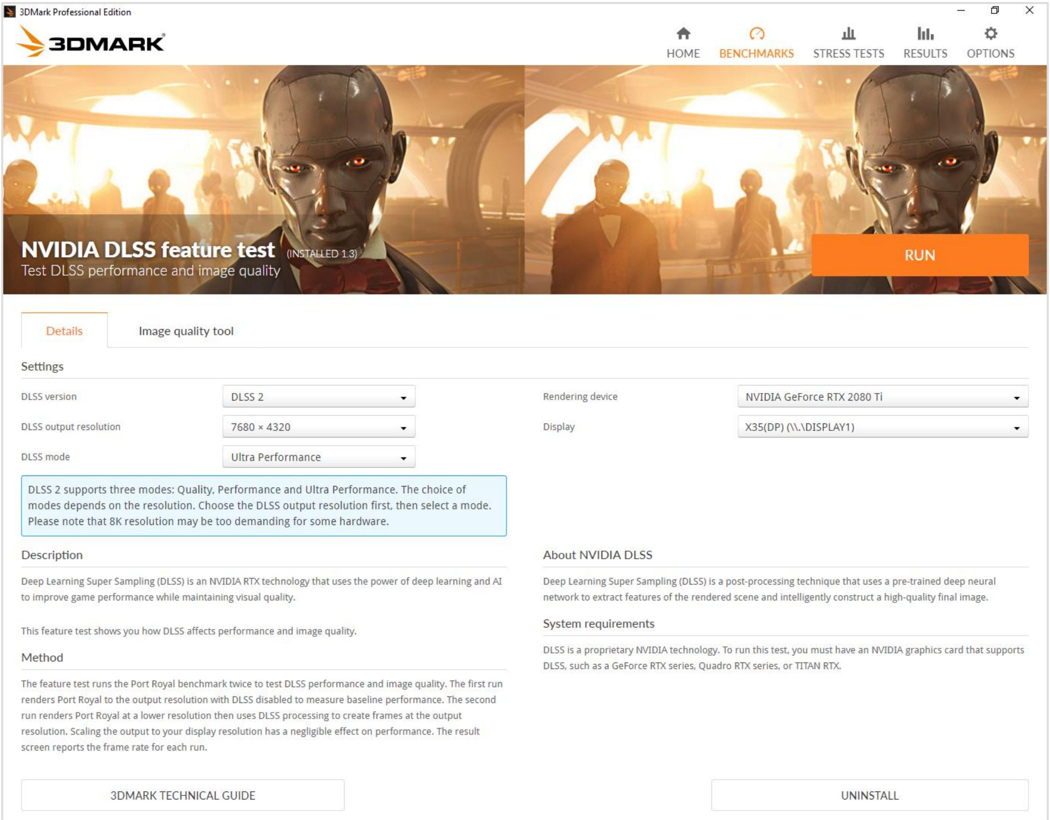
# How to test DLSS performance

The NVIDIA DLSS feature test helps you compare performance and image quality with and without DLSS processing.

The feature test runs the Port Royal benchmark twice. The first run renders Port Royal to the output resolution with DLSS disabled to measure baseline performance.

The second run renders Port Royal at a lower resolution then uses DLSS processing to create frames at the output resolution.

The result screen reports the frame rate for each run. Compare frame rates to see how DLSS affects performance.




## DLSS version

You can choose to run the NVIDIA DLSS feature test using DLSS 2 or DLSS 1.

## DLSS output resolution settings

You can choose an output resolution for the test. By default, the test uses the 2560 × 1440 output resolution. You can select from the following resolutions:

- 1920 × 1080 (1080p)

- 2560 × 1440 (1440p)
- 3840 × 2160 (4K)
- 7680 × 4320 (8K), available only for DLSS 2

The lower the resolution the lower the image quality, but higher performance due to a lighter load on the GPU. You can choose 1920 × 1080 for the highest performance and 7680 × 4320 for the highest image quality.

#### Part 1 — DLSS off

The first part of the test renders the Port Royal benchmark at the output resolution with DLSS disabled. This gives you a baseline reference for Port Royal performance and image quality using TAA.

#### Part 2 — DLSS on

The second part of the test renders Port Royal at a lower resolution then uses DLSS processing to create frames at the output resolution. The table below shows the rendering and output resolution pairs used in the test when DLSS 1 is used.

| DLSS RENDERING RESOLUTION | DLSS OUTPUT RESOLUTION |
|---|---|
| 1440 × 810 | 1920 × 1080 (1080p) |
| 1920 × 1080 | 2560 × 1440 (1440p) |
| 2560 × 1440 | 3840 × 2160 (4K) |

### DLSS mode

With DLSS 2, you can also choose between three image quality modes—Quality, Performance and Ultra Performance.

- The **Quality mode** offers higher image quality than the Performance mode.
- The **Performance mode** offers a higher performance than the Quality mode.
- The **Ultra Performance mode** offers the highest performance increase. It is available for 8K resolution only.

> ⚠ The choice of modes depends on the resolution. Choose the DLSS output resolution first, then select a mode. Please note that 8K resolution may be too demanding for some hardware.

The table below shows the DLSS modes you can choose from for each DLSS output resolution.

| DLSS RENDERING RESOLUTION | DLSS OUTPUT RESOLUTION |
|---|---|
| 1920 × 1080 (1080p) | Quality, Performance |
| 2560 × 1440 (1440p) | Quality, Performance |
| 3840 × 2160 (4K) | Quality, Performance |
| 7680 × 4320 (8K) | Quality, Performance, Ultra Performance |

## Rendering device

On systems with multiple GPUs, you can choose which GPU you want to use for the test.

## Display

On systems with multiple displays, you can choose which display you want to use for the test.

## Display resolution

For both parts of the test, the output resolution is scaled (if required) to the **Scaled resolution** setting on the 3DMark Options screen. By default, this is set to **Automatic**, which scales the output resolution to the display resolution of your Windows desktop. You can override this setting on the Options screen to test other scenarios based on the resolution and scaling capabilities of your monitor.

Scaling from output resolution to your display resolution has negligible effect on performance.

# How to compare DLSS image quality

In 3DMark Professional Edition, you can use the 3DMark Image Quality Tool to output and save frames from each part of the test for further analysis. But please be aware that there is a known issue with the Tool that can affect the quality of the images it generates when using DLSS 1. This issue was fixed for DLSS 2.

Note that rendering a frame right after camera cut will result in noisy images because the temporal accumulation is reset in camera cuts. Warmup will render frames before the camera cut, but this won't have an effect on the frames after the cut.

### Workaround for Image Quality Tool when using DLSS 1

TAA works by using information from previous frames. In **Single frame** mode the tool does not generate enough warmup frames for TAA to work properly. The lack of warmup frames in Single Frame mode results in the Tool producing low quality images.

While we are fixing this for a future update, please use **Frame sequence** mode to capture pixel-perfect comparison images.

Set the frame sequence to run for at least 10 frames either side of the frame you wish to capture. For example, if you want frame 500, set the frame sequence to frames 490 to 510 and use the middle frame for comparison.

# Result screen

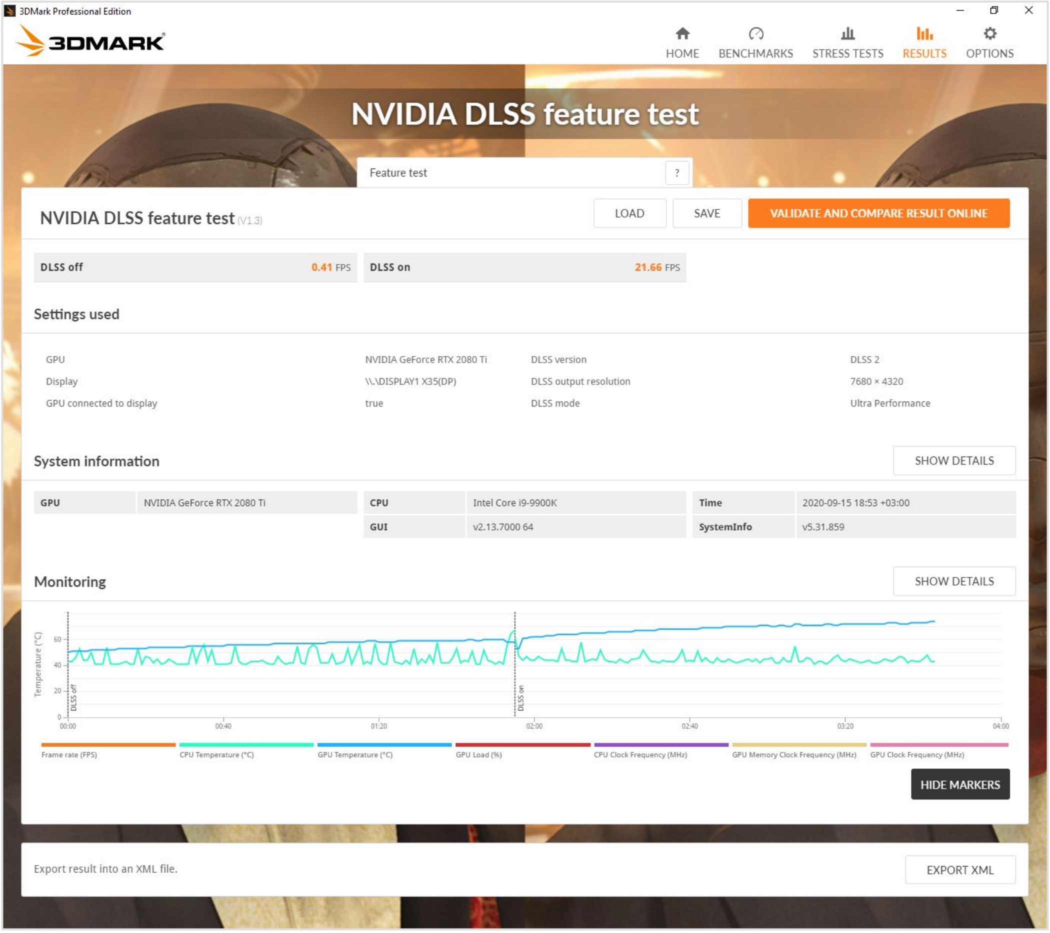
The NVIDIA DLSS feature test produces two frame rate results.

- Average frame rate with DLSS off.
- Average frame rate with DLSS on.

Compare the two results to see how DLSS affects performance.


*Result screen from NVIDIA DLSS feature test using DLSS 1*

When running the NVIDIA DLSS feature test using DLSS 2, the result screen reports the performance for the selected DLSS output resolution and quality mode. Change the quality mode and run the test again to measure the average frame rate with DLSS off and on.




*Result screen from NVIDIA DLSS feature test using DLSS 2*

Hardware monitoring charts show the frame rate and other hardware metrics changed during the test.

# Graphics engine

The NVIDIA DLSS feature test uses the Port Royal engine with an additional DLSS integration. Please read the Port Royal engine section of this guide for more details.

# NVIDIA DLSS feature test history

| VERSION | | | | NOTES |
|---|---|---|---|---|
| 1.3 | ● | × | × | Added the options to run the test using DLSS 2 and select from three image quality modes |
| 1.2 | ● | × | × | Added the GPU and monitor selector |
| 1.1 | ● | × | × | Updated to ensure compatibility with upcoming NVIDIA drivers. |
| 1.0 | ● | × | × | Launch version |

# Stress Tests

Stress testing is a useful way to check the reliability and stability of your system. It can also identify faulty hardware or a need for better cooling. The best time to run the stress test is after buying or building a new PC, upgrading your graphics card, or overclocking your GPU.

If your GPU crashes, hangs, or produces visual artifacts during the test, it may indicate a reliability or stability problem. If it overheats and shuts down, you may need more cooling in your computer.

Stress Tests are not available in 3DMark Basic Edition or the Steam demo.

# Options

### Test Selection

Use this drop down menu to choose which Stress Test to run. 3DMark offers many tests, each designed for a specific class of hardware. You should use the test most suited to the system you are testing.

> ⚠ Note that Fire Strike Ultra requires at least 3 GB of dedicated video card memory. A crash on a system that does not meet this requirement is not a sign of a hardware stability problem.

### Number of loops

In 3DMark Professional Edition, you can use this option to set the number of loops for the test. The minimum number of loops is 2. The maximum is 5000. You can stop the test at any time by pressing the ESC key.

### Enable window mode

In 3DMark Professional Edition, use this option to run the test in a window.

# Technical details

The aim of stress testing is to place a high load on the system for an extended period of time to expose any problems with stability or cooling capability.

3DMark Stress Tests work by looping a benchmark graphics test continuously without pausing for loading screens or other breaks. A Stress Test takes around 20 minutes to run when set to the default 20 loops, which is usually enough to find any significant stability or cooling issues.

| STRESS TEST | TARGET HARDWARE | ENGINE | RENDERING RESOLUTION |
|---|---|---|---|
| TIME SPY EXTREME | PC systems designed for 4K gaming with DirectX 12 | DirectX 12 feature level 11 | 3840 × 2160 (4K UHD) |
| TIME SPY | High-performance gaming PC running Windows 10 | DirectX 12 feature level 11 | 2560 × 1440 |
| NIGHT RAID | PCs with integrated graphics | DirectX 12 feature level 11 | 1920 x 1080 |
| PORT ROYAL | Graphics cards supporting Microsoft DirectX Raytracing | DirectX 12, feature level 12.1, DirectX Raytracing API | 2560 × 1440 |
| FIRE STRIKE ULTRA | PC systems designed for 4K gaming | DirectX 11 feature level 11 | 3840 × 2160 (4K UHD) |
| FIRE STRIKE EXTREME | Multi-GPU systems and overclocked PCs | DirectX 11 feature level 11 | 2560 × 1440 |
| FIRE STRIKE | High-performance gaming PCs | DirectX 11 feature level 11 | 1920 × 1080 |

# Scoring

The main result from the Stress Test is the system's Frame Rate Stability expressed as a percentage.

$$Frame\ Rate\ Stability = \frac{fpsLow}{fpsHigh} * 100$$

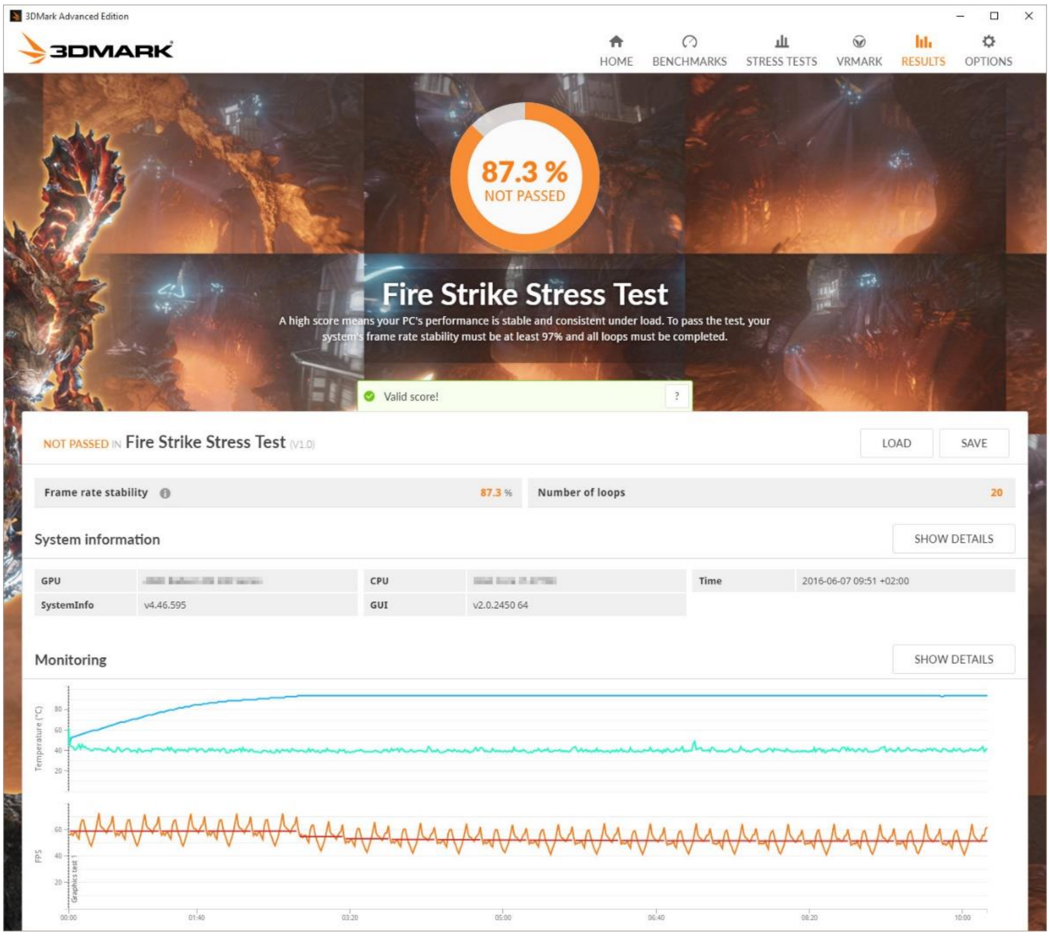
Where:

$fpsHigh$ = The average frame rate from the best performing loop of the test.

$fpsLow$ = The average frame rate from the worst performing loop of the test.

A high score means your PC's performance under load is stable and consistent. To pass the test, your system's frame rate stability must be at least 97% and all loops must be completed.

In the example below, the system failed the test because its average frame rate dropped noticeably after the GPU reaches its peak temperature.



# How to report scores

3DMark includes many tests, each designed for a specific type of hardware. Make sure you use the most appropriate test for the hardware's capabilities.

Each test gives its own score, which you can use to compare similar systems. There is no overall 3DMark score. Scores from different tests are not comparable. Do not use 3DMark as a unit of measurement.

✓ "Video card scores 10,000 in 3DMark Fire Strike benchmark."
× "Video card scores 10,000 3DMarks."

Always include details of the hardware setup you used to obtain the score. Be sure to include the operating system, system hardware and version numbers for relevant drivers.

## World record scores

UL's Hall of Fame is the only source of official 3DMark world record scores. You should not present scores from any other website or leaderboard as world records. In those cases we suggest using alternative wording such as:

"Video card takes the number one spot on [website] leaderboard."

## Using 3DMark scores in reviews

We provide established and reputable publications with complimentary Professional Edition benchmarks. Contact us at UL.BenchmarkPress@ul.com to request keys for your publication.

Press can use our benchmark scores in their hardware reviews. Please include a link to https://benchmarks.ul.com/ whenever you use our benchmarks in a review, feature, or news story.

## Using 3DMark scores in marketing material

For business purposes, a commercial license is granted with the purchase of 3DMark Professional Edition or through our site licensing program.

⚠ You must not disclose or publish 3DMark benchmark test results, nor may you use the UL logo or other UL assets in your sales and marketing materials, without prior, written permission from UL. Please contact UL.BenchmarkSales@ul.com for details.

On the first mention of 3DMark in marketing text, such as an advertisement or product brochure, please write "3DMark benchmark" to protect our trademark. For example:

"We recommend 3DMark® benchmarks from UL."

Please include our legal text in your small print.

3DMark® is a registered trademark of Futuremark Corporation.

# Release notes

You can read the history of 3DMark release notes on our website.

# About UL

UL is helping create a better world by applying science to solve safety, security and sustainability challenges. We empower trust by enabling the safe adoption of innovative new products and technologies. Everyone at UL shares a passion to make the world a safer place. All our work, from independent research and standards development, to testing and certification, to providing analytical and digital solutions, helps improve global well-being. Businesses, industries, governments, regulatory authorities and the public put their trust in us so they can make smarter decisions. To learn more, please visit UL.com.

UL benchmarks help people measure, understand and manage computer hardware performance. Our talented team creates the industry's most trusted and widely used performance tests for desktop computers, notebooks, tablets, smartphones, and VR systems.

We work in cooperation with leading technology companies to develop industry-standard benchmarks that are relevant, accurate, and impartial. As a result, our benchmarks are widely used by the press. UL maintains the world's largest and most comprehensive hardware performance database, using the results submitted by millions of users to drive innovative online solutions designed to help people make informed purchasing decisions.

Our benchmarks are developed in Finland just outside the capital, Helsinki. We also have a performance lab and sales office in Silicon Valley and sales representatives in Germany, China and Taiwan.


| | |
|---|---|
| Press | UL.BenchmarkPress@ul.com |
| Sales | UL.BenchmarkSales@ul.com |
| Support | UL.BenchmarkSupport@ul.com |