Bringing *Anno 1800** to Laptop Gamers Around the World: The Long Road From 10 to 30 FPS

By Philipp Gerasimov (Intel), Frank Hoffmann (Ubisoft), Marcel Hatam (Ubisoft)



Figure 1: Anno 1800 in-game scene

About the Anno Series and *Anno 1800

Anno 1800—Lead the Industrial Revolution!

Welcome to the dawn of the Industrial Age. The path you choose will define your world. Are you an innovator or an exploiter? A conqueror or a liberator? How the world remembers your name is up to you.

First launched in 1998 during the golden age of real-time strategy gaming to great critical acclaim, *Anno 1602** quickly went on to become one of the best-selling strategy games at the time. Players could not help falling in love with its unique mix of city building, economic simulation, and highly detailed graphics (trademarks that the series has managed to maintain over the ensuing 20 years). Following a sequel building on these strengths (*Anno 1503**), the series underwent its first major change with the third entry, *Anno 1701** (or 1701 A.D., depending on what port you may call home).

Not only did the game make the technically, extremely challenging jump from highly detailed 2D art to lovingly rendered full 3D art, it also underwent a change of developers. While the first two games hailed from rural Austria, *Anno 1701* was the first game in the series to be developed by Related Designs in the heart of Germany. And while names may change (Related Designs having been fully acquired by Ubisoft

in 2013 to become Ubisoft Mainz), it is worth pointing out that many of those same developers who brought the series into the third dimension in 2006 are still working on it 13 years later. The new team quickly came into their own with 2009's seminal *Anno 1404** (*Dawn of Discovery** for international audiences; thankfully this was the last title in the series with differing names in regions), which is widely considered the most beloved game in the series.

Having delivered the peak of historically inspired Anno games, the team next turned to new frontiers, as 2011's *Anno 2070** not only brought the series into a sci-fi setting that allowed players to build futuristic metropolises on the ocean floor, but, thanks to the rising popularity of digital distribution, it also managed to attract a wider international audience. This was followed up by 2015's *Anno 2205**, which turned out to be another visual benchmark in the city-building genre.

After that, the team once again felt the call of history and returned to its roots, both in historic atmosphere and gameplay depth. 2019's *Anno 1800** plunges players into the turbulent age of the industrial revolution, bringing all the iconic charm of a pivotal era in human history to lovingly crafted 3D life. It also set new records for the franchise, boasting to become the fastest-selling Anno game, further proving that the PC is today's most dynamic and diverse gaming platform.

Why It is Great and Important to Support More Gamers

As previously mentioned, one of the hallmarks of the Anno series since day one has been its ambition to deliver best-in-genre graphics, with loving attention to even the tiniest detail—from the faithful workers in your factories to dogs enthusiastically chasing cats across the bustling marketplaces. This has always posed an optimization and performance challenge, given Anno's massive scale, as players expect to be able to build hundreds of buildings across several islands (and in recent games, even in several parallel maps, or sessions). At the same time, the game is handling very complex calculations under the hood as well, to keep track of the output of dozens of production chains and sometimes hundreds of thousands of citizens.

Ensuring that the games run across the widest possible range of hardware configurations is not only a matter of professional pride for our R&D team but is also an economic necessity.

Many Anno players fall outside of what would commonly be referred to as hardcore gamers (as defined by those gamers who always keep up with the latest trends in games in hardware), and many of them play no other games apart from Anno, or other building and management titles. Anecdotal evidence of this can be seen in our community when some players are surprised to find that the PC that ran *Anno 1404* well in 2009 is suddenly struggling with *Anno 1800* a decade later.

Not many of these players own any dedicated gaming hardware or even a desktop PC anymore but would still love to play a new Anno for nostalgia's sake. And even many hardcore gamers find Anno, with its relaxing pace and ability to be played completely offline in single-player mode, the ideal game to take along on a laptop for some evening entertainment while travelling.

Among all of these factors, when the opportunity to work alongside Intel to optimize *Anno 1800* for their integrated GPUs came along, we immediately jumped on it, and we are extremely happy how the

result of that cooperation has allowed us to welcome a wider range of players than ever before into the Anno community.

Initial Performance Analysis

The optimization work started at the Ubisoft Mainz office with a joint team of Ubisoft and Intel engineers. The initial tests were showing that on Intel's latest integrated graphics the game was running under 10 frames per second (fps) with lowest quality. That was more than 3x slower than it should be for a good gaming experience on a mainstream laptop computer!

The engineering team decided to dive into identifying what the performance issues were, and how to fix those. One of the first steps to understanding the performance of a 3D application is to discover whether it is GPU- or CPU-limited. Games can be bottlenecked by both, and there are multiple methods to evaluate that.

Intel has a comprehensive graphics performance analysis toolkit: Intel® Graphics Performance Analyzers (Intel® GPA) includes powerful, agile tools that enable game developers to use the full performance potential of their gaming platform, including, though not limited to, Intel® Core™ processors and Intel® Processor Graphics. Intel GPA tools visualize performance data from your application, enabling you to understand issues from system-level to individual frame performance.

You can find more information about Intel GPA at the Intel® Developer Zone.

Vegetation Rendering

Using the Intel GPA system analyzer experiments it was clear that the game is totally GPU-limited, so the team dived into a frame analysis. The tool quickly highlighted that most of the frame time was spent rendering vegetation. Diving into these draw calls, we identified that the cause of the low performance was a hardware-specific issue.

Vegetation was writing in the Stencil buffer as well as using the clip() instruction in the pixel shader for alpha testing. This combination of states and instructions is not recommended for Intel® HD Graphics before Gen11. It was found that it is possible to achieve the same vegetation rendering quality without using the Stencil buffer. After some minor code changes the game is now up and running, this time with about 20 fps performance; twice what it was before!



Figure 2: Highest-detailed tree mesh shown when the game runs in low-quality settings (458 triangles).

We further had to reduce the vegetation's performance impact by lowering the number of trees in low-quality settings. This proved especially useful in the game's New World setting, a South America-inspired map, where most parts of the islands are densely covered with jungle trees. We also needed to disable the grass auto-generator that normally scatters thousands of grass and bush meshes onto empty areas and around forest borders, depending on the camera view. These many small objects pressed too hard on the GPUs triangle throughput and pixel fill rate.

Object Culling

Anno 1800 is all about many small visual details on the screen. Our players create and manage their own cities from the ground up, but regularly they want to put management aside and immerse into their creation. For our engine, this poses a serious challenge, as tens of thousands of dynamic objects need to be ticked and rendered each frame. Almost everything on screen is dynamic: The game's buildings consist of dozens of smaller sub objects, needed for visual variations, ease of use for our artists, and gameplay-dependent animations and effects. Same for the inhabitants, wild life, and vehicles moving around the game world. We need to handle trees, bushes, and grass individually too, as they have to be dynamically added or removed while the player's city evolves. That sheer mass of objects presses heavily on our game's performance, both CPU and GPU-wise.

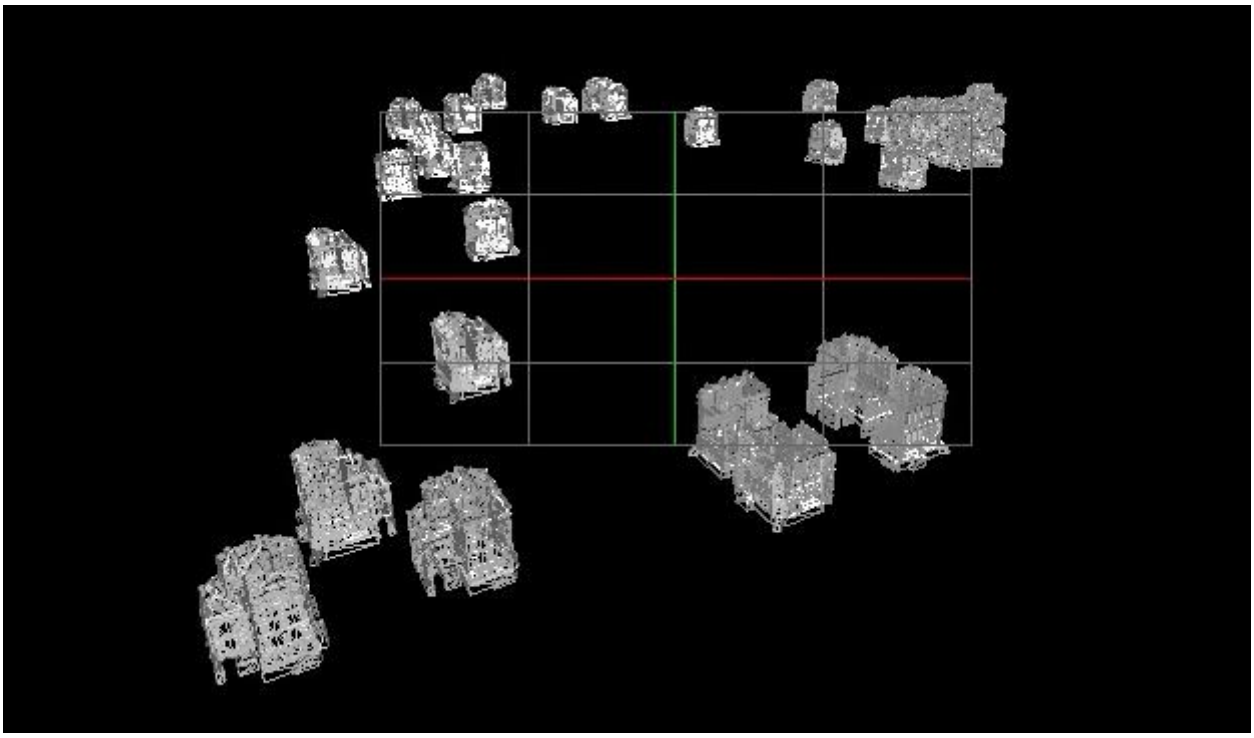


Figure 3: A warehouse building with its sub objects colorized. Most of the sub objects are culled by camera distance.

Many of these objects are only noticeable to the player on close view. We aggressively cull objects as early as possible by their size. For each object, we estimate its final size on the screen based on its bounding sphere, scale, camera distance, and the camera field of view (FOV). A hard threshold culls objects that are small enough. Objects that almost reach that threshold are faded out via alpha blending for a soft look, as popping would be just too noticeable and distracting. Object shadows are faded by dithered rendering into the shadow map.

The game actively utilizes geometry instancing to rendering multiple object instances in one draw call. That allows it to render tens of thousands of objects with a much smaller number of actual draw calls; for example, it could be all the buildings of the same time in a village. That also brings another challenge; these objects are not always spatially close to each other. Without an aggressive frustum culling, there could be cases where some of the rendered objects are completely off screen, but still being sent to and transformed by the GPU.

Figures 4 and 5 from the Intel GPA Frame Analyzer highlights some of these issue with culling. They show two draw calls, which render building. The visible viewport is displayed by the square in the middle with red and green lines. It is clearly visible that multiple instances are completely outside the viewport. These instances will be culled during the clipping rendering stage, but all previous stages, including geometry fetching and VS processing will be executed on the GPU. Many GPU clock cycles could be saved with optimizing the instances and doing better CPU culling on per instance bounding boxes, before even being sent to the GPU.



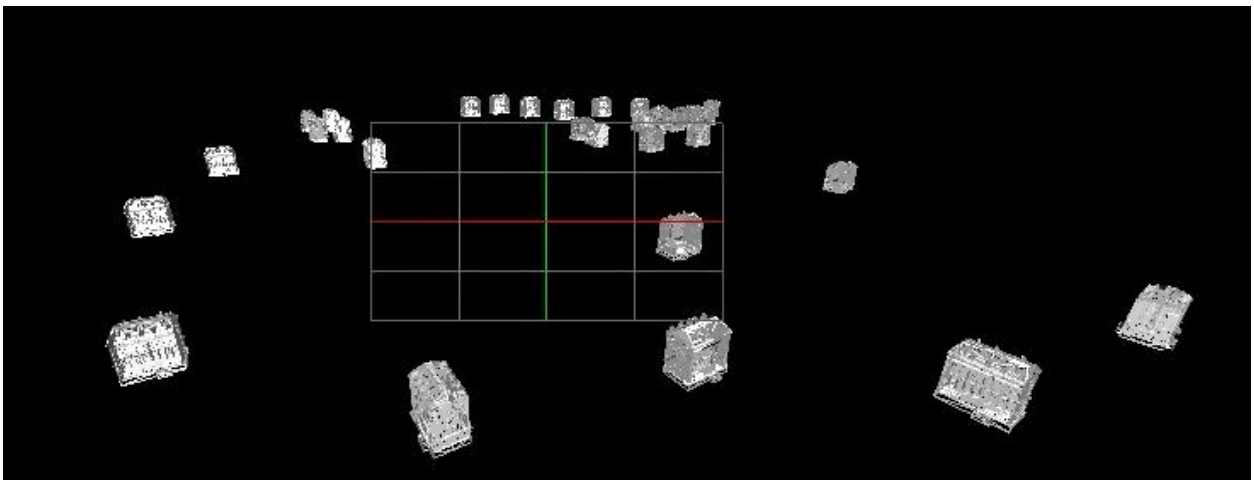


Figure 4 and 5: Initial Intel GPA geometry view, which shows sub optimal object culling.

3D Pipe/Rasterizer		
Clipper Invocations	190k	190k
Post-Clip Primitives	39.9k	39.9k

Figure 6: 3D Pipe/Rasterizer hardware metrics

Figure 6 shows Clipper hardware metrics, which are available for each draw call in the Intel GPA Frame Analyzer. The first metric shows total number of Clipper invocations and the second shows how many primitives are left after. In the example provided in Figure 5, there are only about 40 thousands

polygons left from the initial 190 K. It is a little bit more than 20 percent, with 80 percent of the work submitted being culled.

Shadows Optimizations

(1) In previous Anno titles we used cascaded shadow maps, which we classically managed by CPU code, carefully tweaked by hand for worst-case scenarios. For *Anno 1800* we switched to Sample Distributed Shadow Maps (SDSM) [Lauritzen2010], where a compute shader dynamically adjusts the shadow cascades each frame by analyzing the depth buffer after the Z prepass is finished. This results in much sharper shadows in almost all cases. For example, in our previous titles the shadows were always a little bit offset and disconnected from small casters like the inhabitants, and caused by the many tweaks and biases that we needed for handling other problematic situations. SDSM greatly improved these cases and connected the shadows to their casters.

Switching to SDSM led us to a serious problem though: We were no longer able to partition objects into shadow cascades on the CPU, as we no longer had access to the cascades' split plane positions. Initially, we just postponed this problem and always rendered all shadow-casting objects into each cascade, resulting in a severe performance slowdown, especially on the GPU. We tried reading the split plane positions back from video random access memory (VRAM) and using them for CPU partitioning. However, this did not show satisfying results, as the data lags several frames behind and is just too outdated.

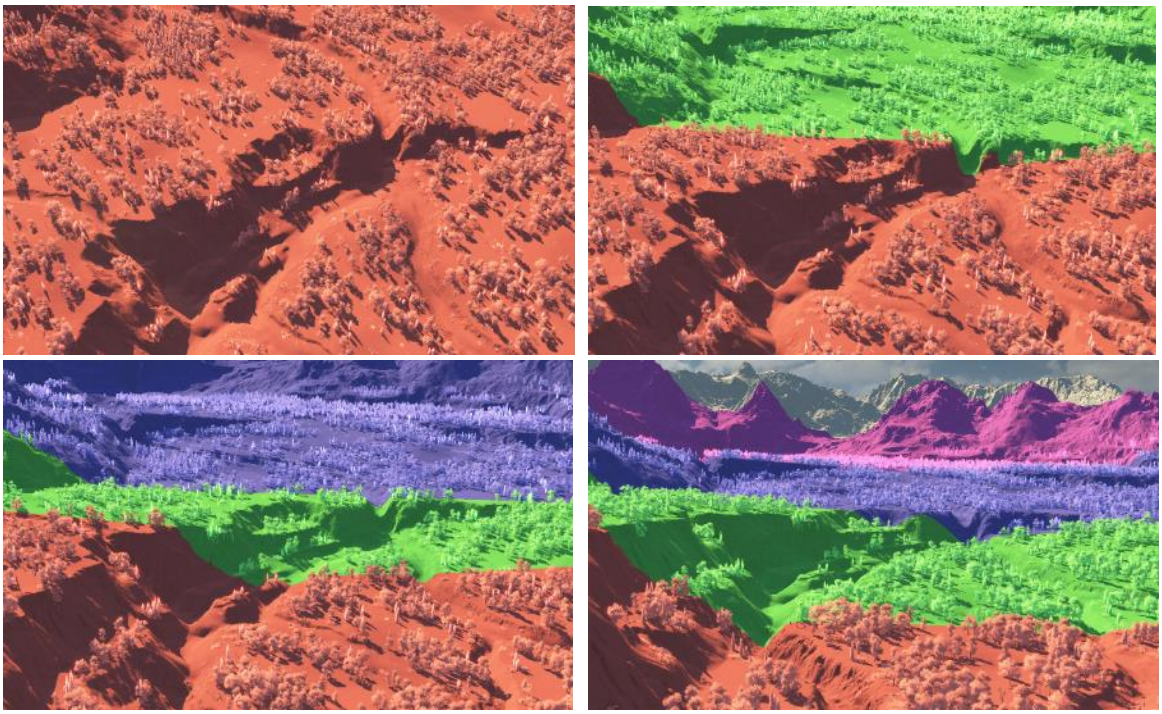
In the end, we decided to do the partitioning on the GPU with compute shaders. We filled a structured buffer with the relevant information about all shadow-casting objects: their Axis-Aligned-Bounding Boxes (AABB), transformation, and shading parameters. The compute shader consumes that buffer, tests each object to the shadow cascade split planes, and writes into two output buffers. The objects' transformation and shading parameters are appended to a second structured buffer, while a byte address buffer is filled with the draw arguments. Afterwards, we simply replace our usual DrawInstancedIndexed() calls with DrawIndexedInstancedIndirect() for shadow map rendering, consuming the byte address buffer's draw arguments, and reading the per object parameters in the shaders from the structured buffer instead of constants buffers.

(2) We originally only scaled down the shadow map resolution on lower quality settings, but always used a full set of four shadow cascades in all situations. This introduced several performance problems on low-end machines. First, all shadow-casting objects needed to be checked into which of these cascades they need to be rendered. Second, all objects at cascade boundaries also needed to be rendered twice, once for each cascade they touch. As *Anno 1800* is mostly played from a top-down perspective, such a large number of cascades are just not needed in most situations: A single cascade is sufficient in a top-down view. Only when the camera is looking to the horizon do we need multiple cascades for a proper distribution of the shadow map texels.

We adapted the number of shadow cascades dynamically to the current camera pitch (Figures 7-10). Additionally, we limited the maximum number of cascades, using up to four cascades only on the

highest quality settings. For low quality we clamped to a maximum of two cascades, which showed to be good enough in all situations.

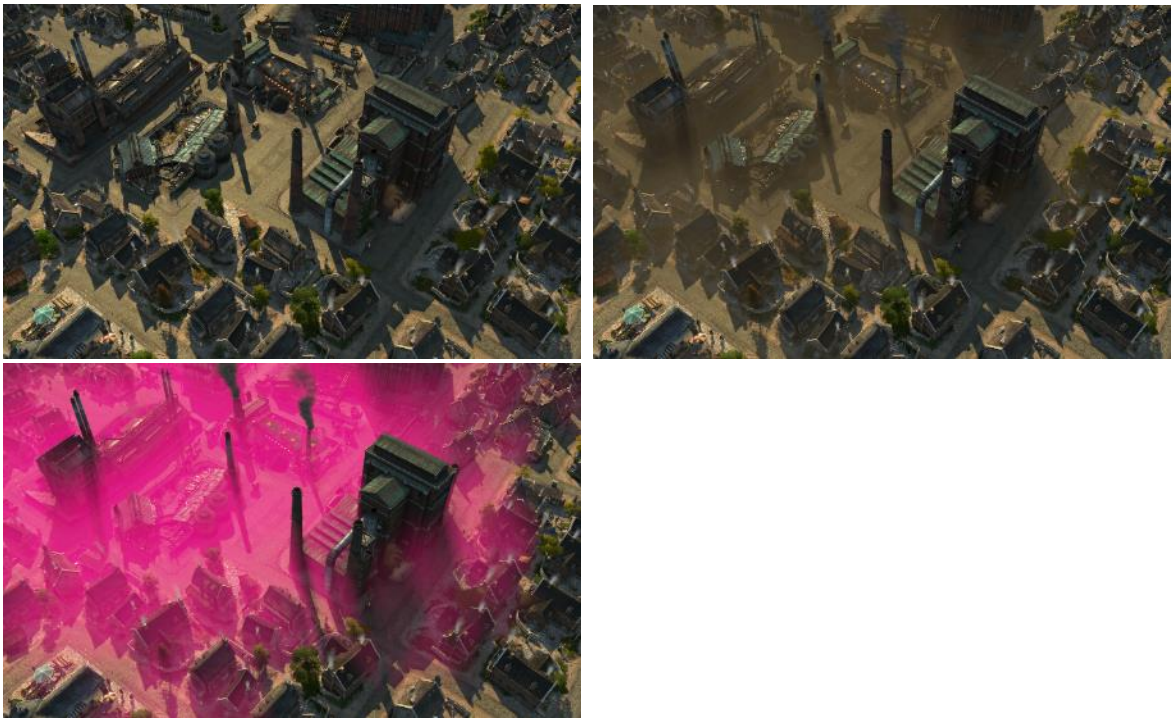
We also disabled the soft blending between adjacent cascades on low quality. On higher quality settings, a blue noise dithering pattern blends the crossing line between the cascades that is occasionally noticeable. On low-quality settings, we felt these additional shader instructions are not worth their cost.



Figures 7 - 10: Colorized view of the active shadow cascades, depending on camera pitch. The fourth picture's far gray area does not receive shadows.

Particle Effects and Fog

Anno 1800 is a game about the industrial revolution, characterized by smog and pollution in heavy industry areas. This effect is realized by rendering simple boxes into a render target with 1/16th resolution. The pixel shader ray marches through a 2D density texture, adds volumetric noise, and takes the depth buffer into account. Afterwards, the render target is upscaled and blended into the scene (Figures 11-13). Even though we calculate this at a reduced resolution, the ray marching initially took 8 milliseconds (ms) on our Intel® NUC (next unit of computing). As this effect is an important visual hint to the player, we could not simply disable it on low-quality settings. Luckily, the shader showed a lot of potential for optimizations: We increased the ray marching's step distances and limited the maximum step count even further. We disabled the volumetric noise computations, which saved two 2D texture samples and several arithmetic-logic unit (ALU) instructions per ray march step. Last but not least, we disabled the volumetric shadows, which saved two shadow map samples and even more ALU instructions per ray march step.



Figures 11 – 13: Industrial factory scene without (figure 11) and with (figure 12) pollution fog. Figure 13 shows the fog exaggerated with magenta color.

Adding LODs

As previously shown, individual objects in the game can be quite detailed and have thousands of polygons. This adds great detail when people are looking closely at these objects at the highest zoom level. Beautiful building and nature details make gamers feel they are inside these cities. But when players zoom the view out to see more area, and as a result more objects, it creates significant challenges for performance. It could even be that each object has more vertices than pixels on the screen. But there is a well-known technique to resolve this challenge—level of detail (LOD). The game should use different variants of the same model, depending on view distance.

For quality reasons, our artists generated all LODs manually, by hand, instead of using auto-generator tools. Until the end of production, most buildings had no full LOD chain, as artists first wanted to nail down each asset's final look before putting effort into the LOD creation. Although reasonable from a production standpoint, this made early performance tests a bit problematic.

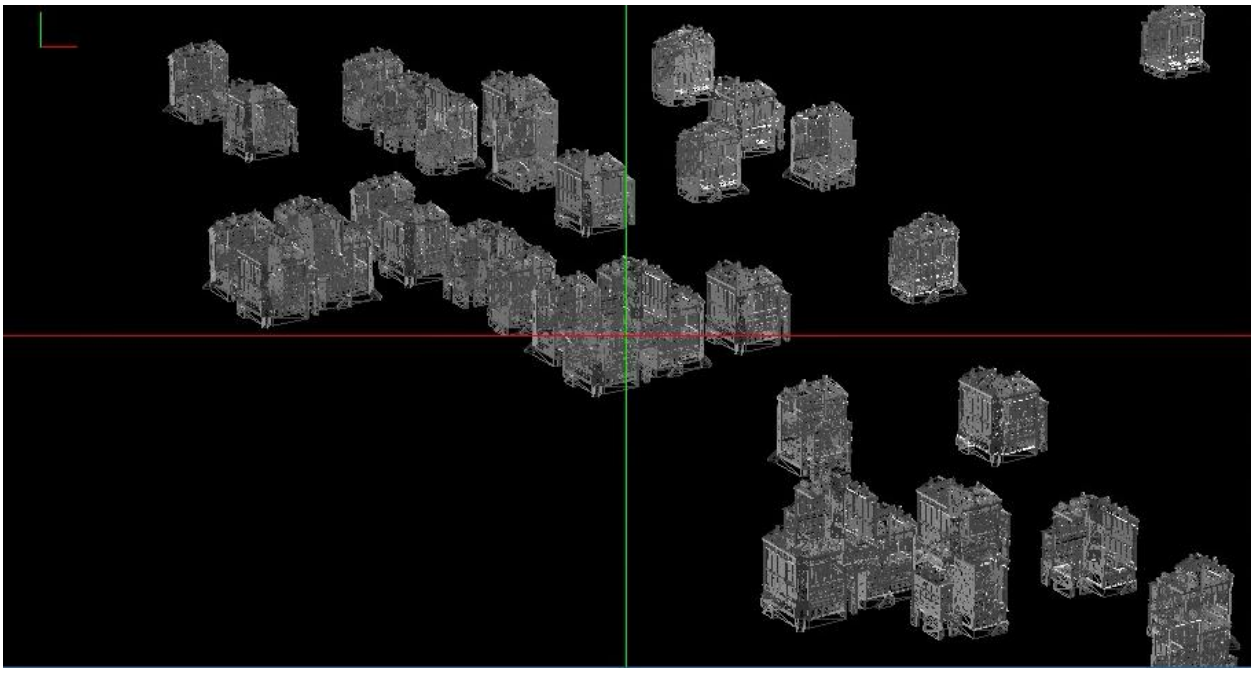


Figure 14: One of the big draw calls showing a high number of building instances.

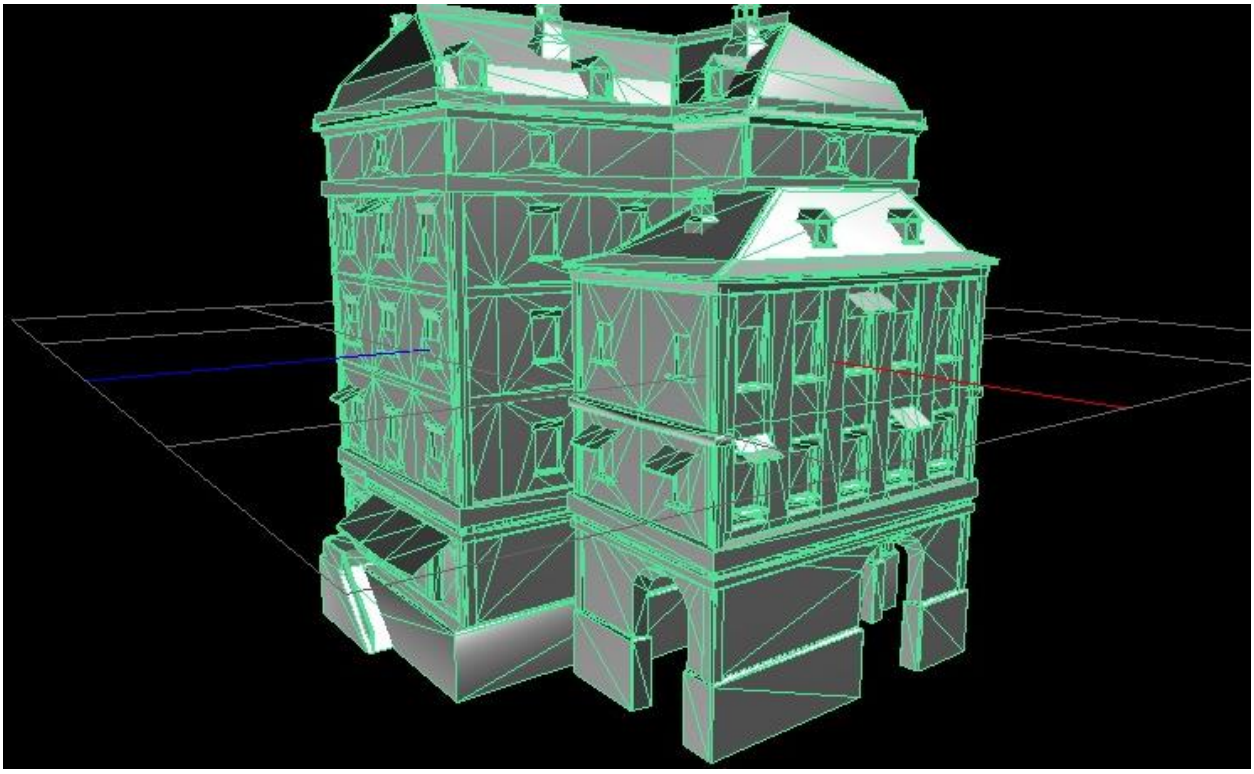


Figure 15: Geometry for the one building.

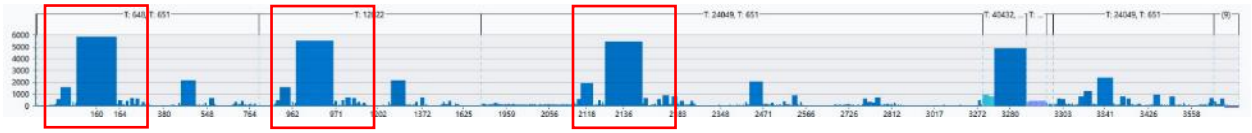


Figure 16: Intel GPA Frame Analyzer bar chart, showing a group of large draw calls.

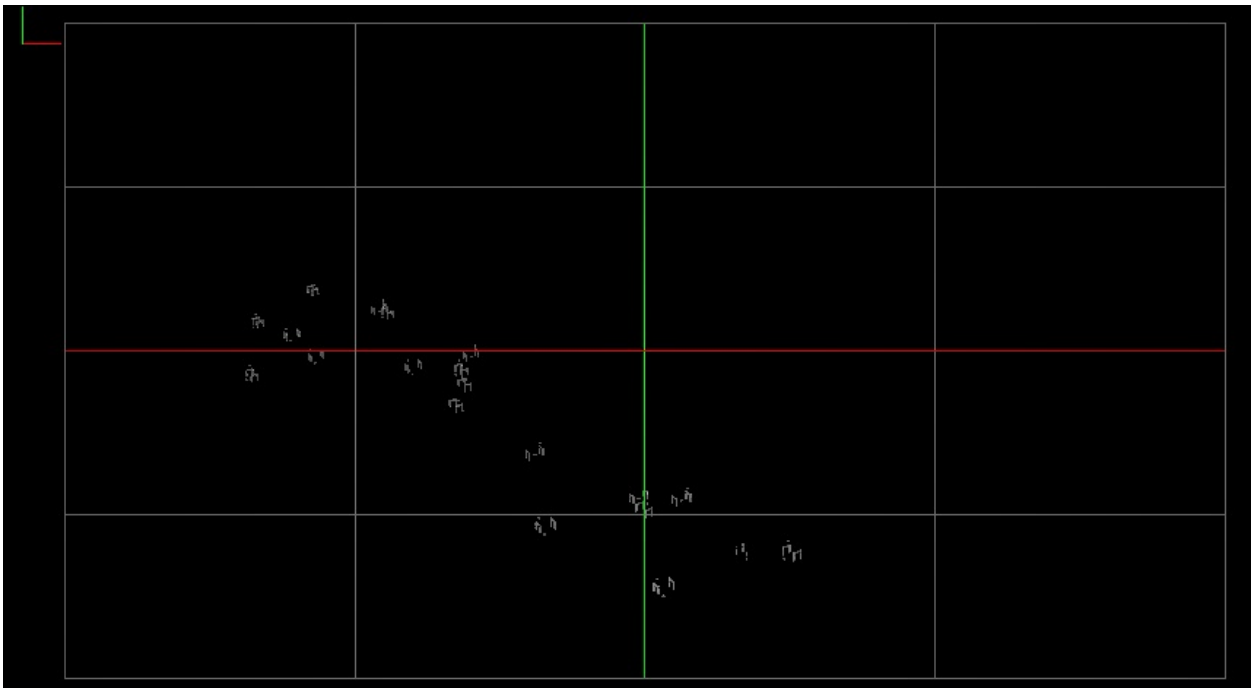


Figure 17: Geometry view for each call.

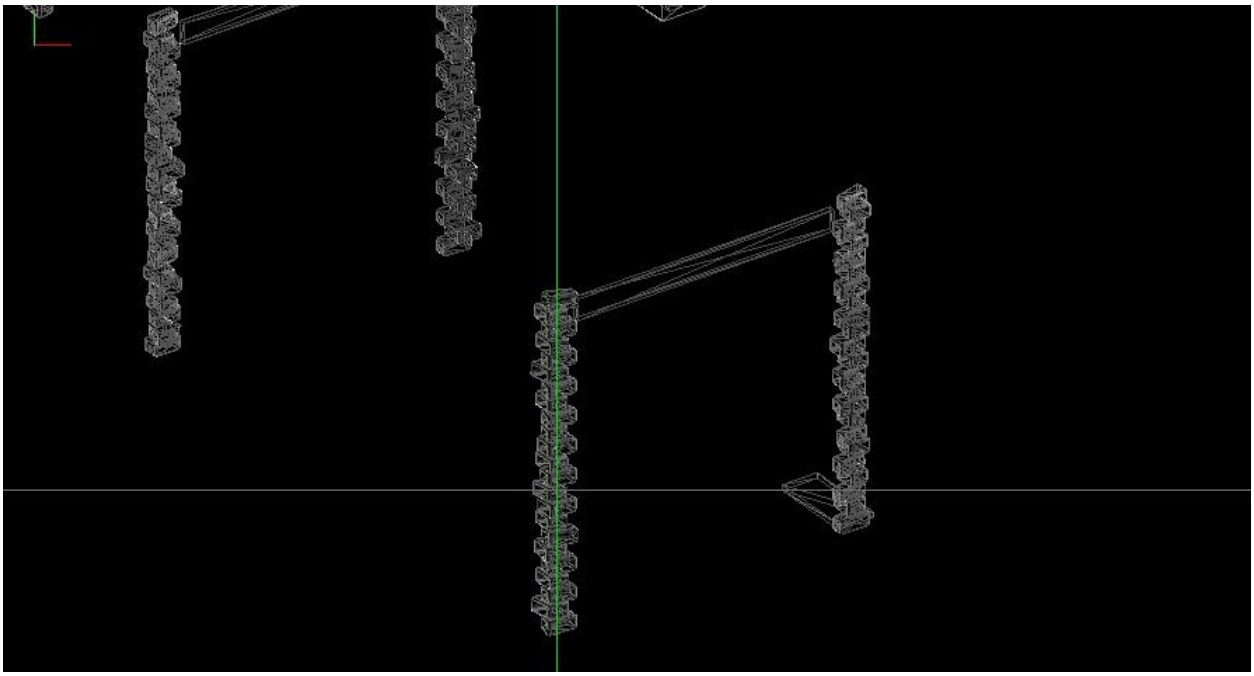


Figure 18: Zooming in to view one distance.

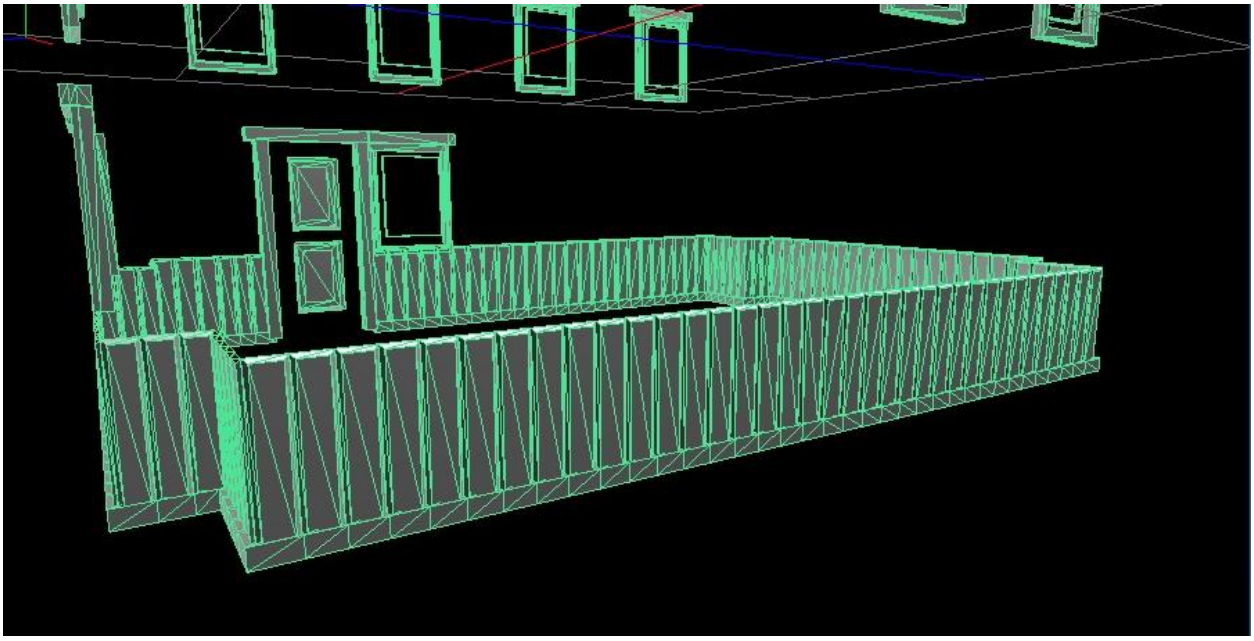


Figure 19. Another example of high polygon building.

On low-quality settings, we limit all assets to their third detail level that misses most small and medium geometrical details. LOD switches occur much more aggressively. To lower the triangle count even more we had to introduce an additional fifth LOD for all assets.



Figures 20 – 22: LODs of a building mesh with 6000, 2100 and 200 triangles.

Lighting and Shading

(1) We normally render the scene into RGBA16F intermediate render targets and apply tone mapping at the end. On low-quality settings, we switch to the RGBA8 format for faster memory access. The artifacts caused by this are negligible: Color banding becomes most noticeable in large gradient areas, like the sky, that the player rarely sees. We had already replaced the game's Filmic tone mapping with a much simpler transformation (compressing only very bright colors, while the largest part of the mapping curve is linear), which kept color precision problems down.

(2) We omit various calculations in our shaders' lighting functions on low quality: Local point and spot lights are completely disabled; only the directional sunlight is taken into account. We use a hard-coded irradiance light term instead of sampling the irradiance cube map. Cloud shadows, which we compose

from noise texture samples, are disabled too. We also tried to simplify our Cook-Torrance specular ALU calculations, but this introduced a very visible lighting degradation and did not show a measurable performance benefit.

Ocean Water

Our ocean waves are assembled from three different types of waves: Large, open-water waves are generated by a Fourier transformation of the Phillips spectrum [Tessendorf2001]. These waves are tiled across the water surface. On a far view distance, we blend them to simpler noise-generated waves to prevent the tiling of the fast Fourier transform (FFT) waves from becoming noticeable. A third type of wave is added on top: Dynamically generated waves on shorelines, ships, and other objects. These are created by a simple, but high resolution 2D wave simulation that is numerically integrated into each frame, and feed with (A) water-object intersections, (B) special wave particles rendered into it, and (C) shader-generated shoreline waves. On low-quality settings, these dynamic waves are disabled, which saves us several milliseconds of GPU time per frame.



Figure 23 and 24: An in-game ocean scene and the scene with colors showing the three different parts forming the ocean surface: tiled noise pattern (red), tiled FFT (green), and wave simulation (blue).

We also fine-tuned our grid of water chunks. Originally, large parts of the ocean water were also rendered in areas where they are hidden by island geometry. This resulted in many unnecessary vertex shader invocations and triangle rasterizations, which were especially costly, given the large number of vertices of fine-grained water surface meshes. Lowering the chunk sizes and optimizing the terrain coverage checks helped us in speeding up the water rendering. We also tried to cull individual triangles that are hidden below terrain by outputting Not a Number (NaN) in the vertex shader, but this did not show a measurable performance improvement, and resulted in minor rendering artifacts in corner cases.



Figure 25: Visualization of the rectangular water chunks.

Conclusion

The time was close to the launch date and Intel and Ubisoft Mainz engineers were doing final performance tests. Performance criteria were met and the game was running at 30 fps on multiple mobile laptop configurations! It was the first time when an Anno series game was able to comfortably run on mobile PCs with integrated graphics. Both teams were very happy with all the work that was done to complete the goals and improve performance more than 3x from the initial evaluation run in the studio!

As a recap, following are the major performance optimizations:

- Optimizing vegetation rendering to run efficiently using integrated graphics
- Improving efficiently of object culling
- Optimizing and scaling shadows
- Scaling lighting and shading complexity
- Optimizing particle effects and fog
- Adding geometry LOD for most complex objects

Now many happy gamers can run the game on their laptops and experience great fun in building cities, driving economies, and enjoying good moments playing Anno.



Figure 26: The same Anno 1800 in-game scene as in the title image, but with lowest quality settings, runnable on modern Intel laptops.

References

[Lauritzen2010] Andrew Lauritzen, *Sample Distribution Shadow Maps*, Advances in Real-Time Rendering in 3D Graphics and Games, SIGGRAPH 2010, July 2010, https://software.intel.com/en-us/articles/sample-distribution-shadow-maps

[Tessendorf2001] Jerry Tessendorf, *Simulating Ocean Water*, 2001

Notices

No computer system can be absolutely secure.

Intel technologies may require enabled hardware, specific software, or services activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. Check with your system manufacturer or retailer or learn more at intel.com.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel, the Intel logo, and Intel Core are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2019 Intel Corporation

About Ubisoft Mainz:

The teams at Ubisoft Mainz are engineering Ubisoft's biggest blockbusters. Operating from a historic city in the heartland of Germany, the studio develops genre pushing games like the acclaimed Anno 1800 and lends decades of expertise to AAA co-development projects like Tom Clancy's Rainbow Six Siege and the upcoming Beyond Good And Evil 2. More information at mainz.ubisoft.com.

© 2018–2019 Ubisoft Entertainment. All Rights Reserved. Anno, Ubisoft, and the Ubisoft logo are registered or unregistered trademarks of Ubisoft Entertainment in the US and/or other countries.

Tweet: Anno 1800 poses an

Anno 1800 tracks output from many production chains and thousands of citizens, ensuring games run in a wide range of hardware configurations.

Summary: *Anno 1800* tracks output from many production chains and thousands of citizens, ensuring games run in a wide range of hardware configurations.