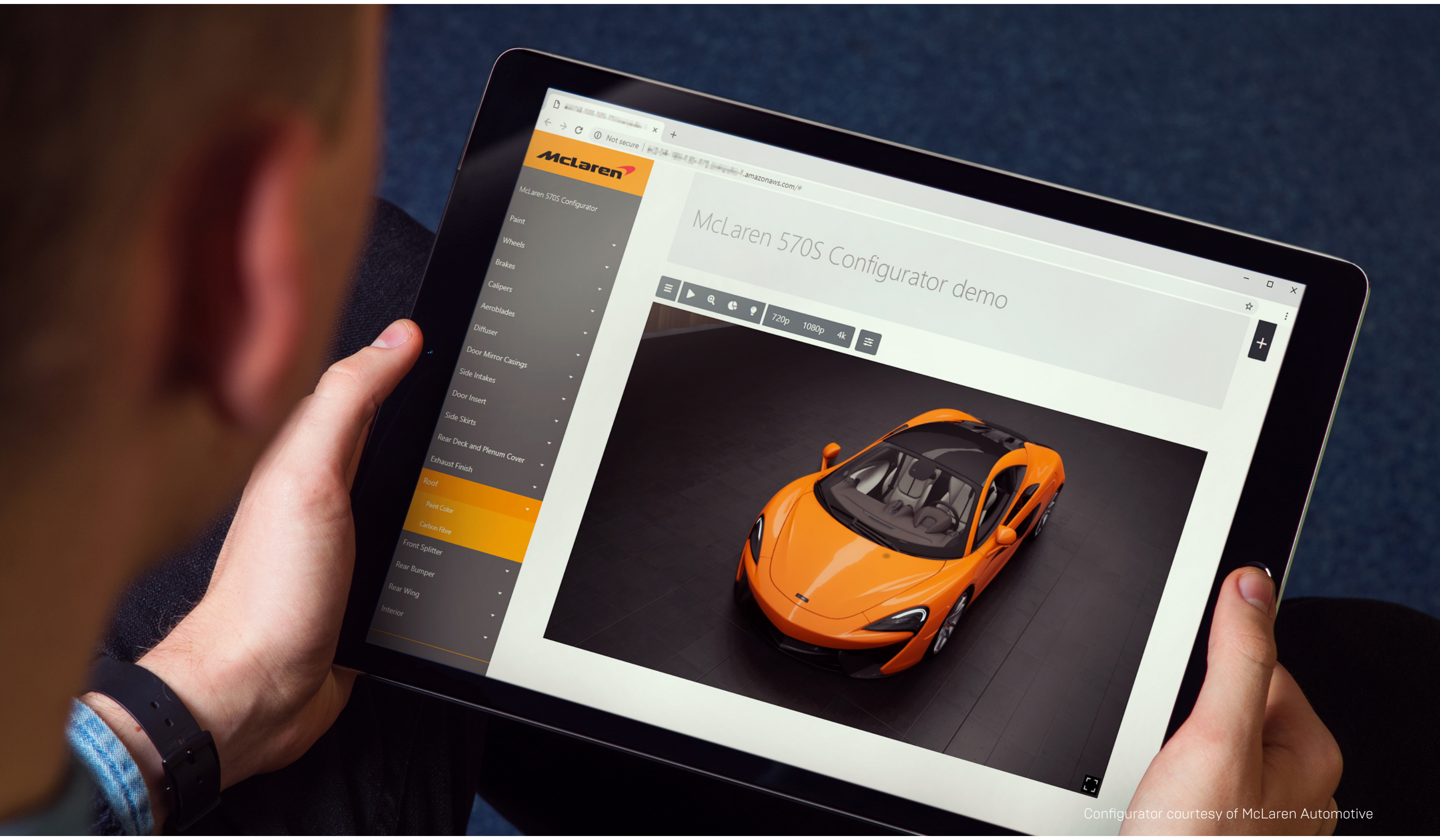


Configurator courtesy of McLaren Automotive

Streaming Unreal Engine content to multiple platforms

Comparison of HTML5, WebGL, and Pixel Streaming

Contents

Introduction

In developing a connected user experience, the challenge of sharing content is always a key decision point in collaboration, production, and publishing. If users with widely varying devices—PCs, tablets, smartphones—will be consuming and interacting with the shared content, a core question arises: how to publish to a diverse field of platforms and hardware capabilities.

The usual approach is to identify the lowest point of entry and acceptable quality for the target audience, and use that as the defining platform. However, this approach limits the quality of the experience for all end users.

Also, in such a deployment, the device on which the end user consumes the real-time content is the same device that contains the data and logic, and which renders the results to the screen. Download of data to the device raises several issues, such as the need to limit quality in the interest of smaller file sizes and faster downloads.

This paper evaluates the options and tools available to share an Unreal Engine-based experience through Pixel Streaming in comparison to other available options. The goal is to maintain the highest quality and capability in the experience for non-games applications of Unreal Engine.

Image courtesy of Zaha Hadid Architects, Line Creative, and Epic Games

Challenges of distributing content

We have moved beyond the Information Age to the Experience Age, where consumers respond best to high-quality, engaging, immersive content. An increasing amount of this content includes an interactive experience, and is based on 3D models as well as AR or XR overlays. To reach the most consumers, producers need to share this content across various platforms, from cell phones and tablets to PCs and interactive displays.

With traditional or current solutions that are based on WebGL or HTML5, the fidelity and interactivity of the displayed content are dependent on the consumer's device, specifically its hardware, display mechanism, and operating system. This means that in order for content to be consumable by the maximum number of users, either the lowest acceptable device defines the shared application, or multiple versions of the application have to be created to address the different user groups.

In creating a streaming solution, the challenge is to deliver this content over multiple communication channels while maintaining fidelity and interactive features along with the branding's look and feel, regardless of the end user's device. This is accomplished by separating the shared application itself, and the high-end hardware needed to run it, from the user's display device.

UX and performance considerations

Development of a shared application over multiple user devices requires consideration of several factors in both the user's experience and the technology that drives it.

Interaction - For a successful deployment, the interaction between user input in the browser and the reaction on the host has to be within a tolerable time space for the experience to be enjoyable. This includes the wait time for initial startup. In general, consumers of non-game content will have a higher patience threshold than those playing a fast-paced game. For example, in an automotive configurator, users might be willing to wait a second or two to see a reaction, and then wait 10 seconds to see a final-quality update.

Playback speed - The content itself might need to be optimized for real-time performance. First, you will need to define what is considered an acceptable playback speed for that particular application and its audience. While 60 fps is the accepted norm for human immersive perception, some applications might require only 5 fps, while others might call for 90 fps.

Image quality - The level of quality, and the fidelity of what the viewer will see and experience, should be determined at the application end by content setup (including considerations of real-time playback), encoding quality, and the hardware's GPU. Quality expectations and cost considerations can define the business case and thus the final output of the setup.

Technical considerations

Hosting and data - Technically, a distributed experience can run from either the client or application side. However, to avoid the quality limitations of client platforms and the time, storage, and security issues inherent in transferring data to the client, streaming from the application side is ideal. For a simple application with a limited number of users, streaming from a single, accessible workstation might be a sufficient solution. For a high number of users, each of whom has full control over a unique version of the application, a cloud solution provides more availability and scalability. A complex application with a low number of users is also a good candidate for cloud hosting. If a cloud solution is used, the content's data size must be considered with regard to storage cost.

User load - The number of users and the complexity of the application will also define what a single instance of the host can deliver and what load balancing is necessary to react to concurrent users. In a high-fidelity, fully interactive car configurator, 10,000 instances of the same setup might be necessary. Depending on how fast the first interaction needs to happen, the number of servers is a key aspect.

Stream sizes - If the target group of users is using a specific platform, for example mobile devices, you might need to specify different video stream sizes to conform with bandwidth restrictions in that field.

Updates - How will updates and bug fixes will be deployed? Will the whole application always be replaced, or is there a way to run live updates? Consider how these updates and fixes translate to a higher number of instances.

Security - Access and security between the host and client might define the use case in the first place. If the system calls for confidential data for design reviews to be shared and discussed, or if new information has to be deployed by a certain time frame but can't be accessed until a later date, it will be necessary to implement security measures on the network or access sites.

Metrics - If user behavior and scene statistics are KPIs, you will need to define, activate, and connect the events that need to be tracked and stored to analytics software.

Comparison of distribution solutions

When Epic Games set out to provide a distribution solution for Unreal Engine (UE4) content, the development team considered using existing technology as the backbone of the solution. After looking at two possible candidates, WebGL and HTML5, Epic decided to create the Pixel Streaming plugin as a new, separate solution.

Here we describe the team's research into WebGL and HTML5 for distributing content, specifically as it pertains to meeting the needs discussed earlier, and then we take a look at the reasons for developing Pixel Streaming technology.

WebGL

WebGL is a JavaScript API that can render interactive 2D and 3D graphics and display the results in a web browser without the user having to download an application or plugin. WebGL is designed and maintained by Khronos Group, a non-profit organization that creates royalty-free open standards.

WebGL is based on OpenGL ES (OpenGL for Embedded Systems), which was designed for embedded systems like smartphones, tablets, video game consoles, and PDAs. WebGL works via the HTML5 Canvas element, which is used to draw graphics on a web page.

WebGL supports GPU-accelerated usage of physics, image processing, and effects.

WebGL deployments consist of two parts:

- Control code written in JavaScript
- Shader code written in OpenGL ES Shading Language (OpenGL ES SL)

By definition, WebGL does not require compiling before deployment.

Authoring for WebGL

The most efficient way to build content for WebGL is to use authoring software that shows the final result. This usually means using a tool that tests online through a browser.

PlayCanvas - The PlayCanvas Editor is an advanced WebGL authoring environment—basically a WebGL game engine. JavaScript is used to program 2D or 3D graphics simulations. All code is written in standards-compliant, cross-platform HTML5 for every major browser and device.

Sketchfab - Sketchfab is an online marketplace for sharing and distributing 3D content. The 3D viewer offers plugin-free deployment across browsers with support for VR and AR. All content must be hosted on the Sketchfab server and distributed from there.

WebGL limitations

The main limitations of WebGL, when considered as a broad distribution tool, have to do with content creation and delivery. Content has to be produced specifically to work in the OpenGL ES SL shader language, which in its current state is usable only for interactive web deployment of applications. To deploy content created with a real-time engine like Unreal Engine, the content and interactive elements must be set up to work within the WebGL framework.

In addition, WebGL relies entirely on the capabilities of the receiving browser and hardware on the client side; the quality of the graphics depends on the browser's display capabilities. Data must be downloaded to the client side, and the time to download defines the wait time for the start of the experience. For a complex experience with a large dataset, the client must have sufficient space to store the data.

If the data is sensitive or confidential, additional steps are required to secure it on the client side.

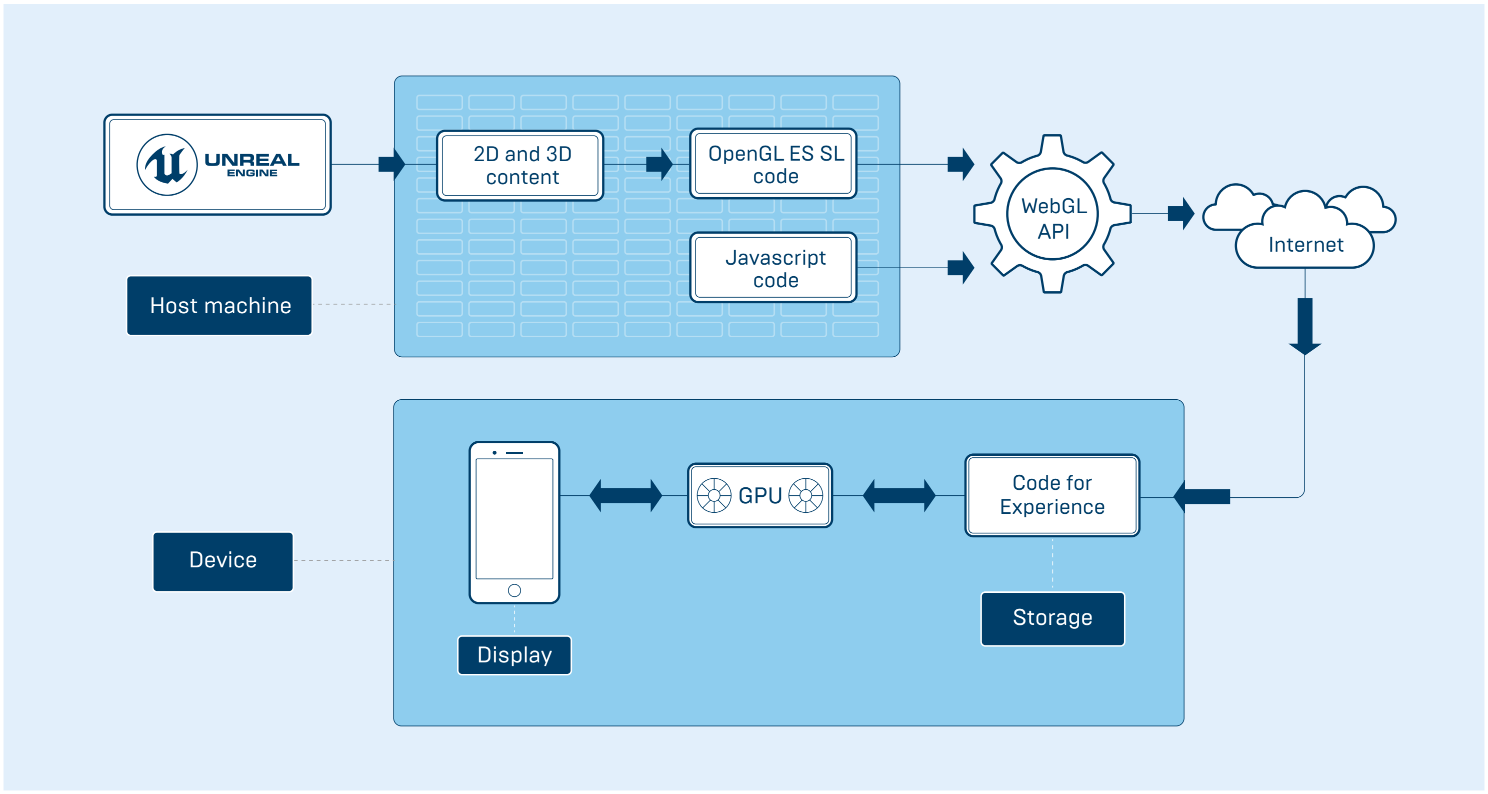


Figure 1: Data flow for WebGL deployment

In an environment where WebGL content is mixed with other produced content, the WebGL quality will be the defining standard of the experience. In a wider platform perspective, this requires the system to maintain at least two data sets, shader models, and creative setups when working with different end channels.

As for metrics, login and session time data can be collected via the usual channels for websites.

Despite its limitations, WebGL is a very simple deployment option suitable for experiences with small datasets, low security concerns, and basic quality requirements. To support these types of experiences, Epic is working on an Unreal Engine option to export to WebGL directly from UE4, including files in glTF format and a supporting library of exchange shaders. We expect to have this exporter ready in Q1 2020.

For Unreal Engine experiences with higher demands, Epic continued its research and development for a more suitable solution.

HTML5

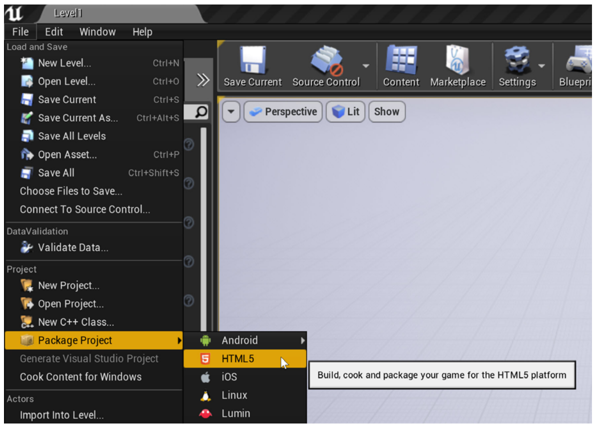
HTML5 is the latest version of HTML, the markup language for web page development. There are various solutions within HTML5 itself to build 3D experiences with the Canvas element, and without using WebGL at all. All these solutions require some knowledge of GPU programming and the packaging of extensive data for the end client to download, and result in an experience limited in quality and functionality.

HTML5 limitations

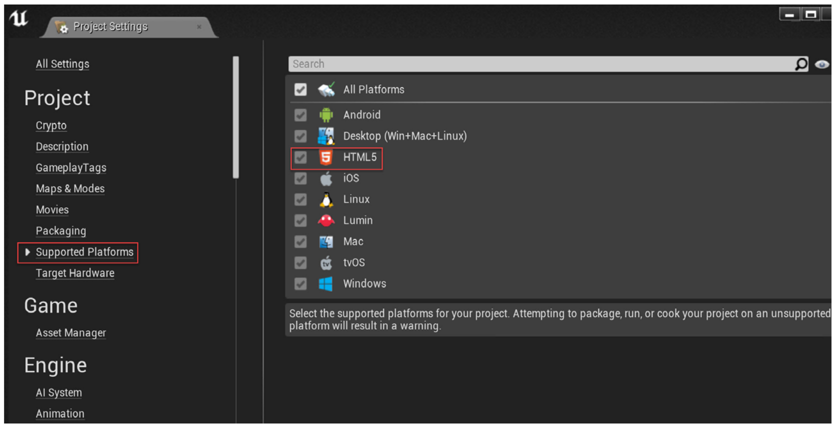
This setup runs the application on the client hardware, which requires the client to download the entire dataset before the experience can start. Download speed and available space are key factors in such a deployment—for a rich experience, the client might need to download several GB of data, which is not feasible with regard to wait time and storage space, particularly on mobile devices.

As with WebGL, this methodology requires the system to maintain two data standards, and can't guarantee a consistent user experience between the original content and the distributed content. Distributed content can be updated and maintained only by completely replacing it, which can become troublesome with highly variable content. Concerns about security with HTML5 are the same as for WebGL.

Unreal Engine includes tools to publish a project to the HTML5 platform. However, as of UE 4.24, Epic migrated HTML5 support to GitHub as a community-supported platform extension, and it is no longer officially supported.



Build, cook and package your game for the HTML5 platform



Select the supported platforms for your project. Attempting to package, run, or cook your project on an unsupported platform will result in a warning.

Figure 2: HTML5 selection in Unreal Engine interface

Pixel Streaming

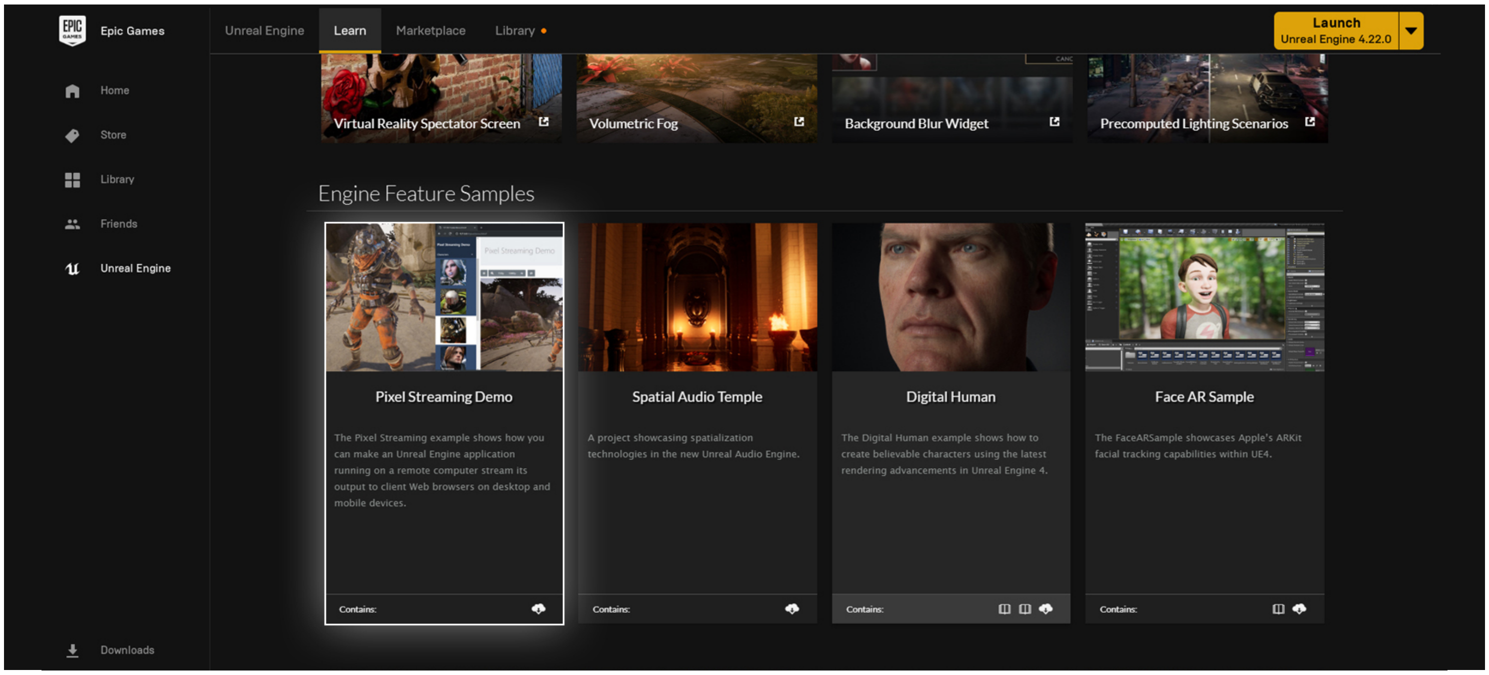
In looking for the best solution for distributing real-time content to multiple types of devices, Epic Games considered the tools and features offered by both WebGL and HTML5. After reviewing the limitations of these technologies, Epic decided to develop its own technology for this purpose, called Pixel Streaming.

Epic determined that the ideal solution for distributing real-time, interactive content to multiple types of devices would include the following features:

- Ability to stream pixels from host rather than requiring client-side data download
- Single setup of data for both high-end and low-end use cases
- Platform-independent deployment
- Deterministic quality on all platforms and devices
- High fidelity and quality
- Ability to share the full range of Unreal Engine features
- Straightforward, simple maintenance and updates
- Fast access at start of experience
- Secure data and setup before, during, and after deployment
- Variety of configurations possible for different use cases
- Scalable; can be deployed from single server or cloud
- Ability to capture metrics on users and sessions

Pixel Streaming is implemented as an Unreal Engine plugin. The plugin encodes the graphics stream on the host server and sends it through the WebRTC protocol to a receiving browser and device. In effect, running Unreal Engine from a high-end host system activates all the Unreal Engine features on end-user devices at the same quality as on the host device.

Because the data stays on the hosting server and only pixels are distributed to the viewing device, a streaming solution like Pixel Streaming is inherently faster and more secure than a client-download solution. In addition, user session data can be captured within UE4 for any needed metrics.



The Pixel Streaming example shows how you can make an Unreal Engine application running on a remote computer stream its output to client Web browsers on desktop and mobile devices.

A project showcasing spatialization technologies in the new Unreal Audio Engine.

The Digital Human example shows how to create believable characters using the latest rendering advancements in Unreal Engine 4.

The FaceARSample showcases Apple's ARKit facial tracking capabilities within UE4.

Figure 3: Pixel Streaming plugin in Launcher interface

WebRTC protocol

WebRTC (Web Real-Time Communication) is a protocol for real-time communication (RTC) through web browsers and mobile applications. With the protocol, audio and video is transmitted via a direct link, with no need for the user to download a plugin or app. Commands for communication are submitted via an API.

Requirements

Pixel Streaming can be run from a single server, or from a GPU cloud environment that allows for dynamic scaling and provisioning of adequate hardware. The key factor in these situations is the analytics of the scale needed, as it directly relates to the cost of the final host environment as well as the speed of the user experience.

The Pixel Streaming plugin communicates through WebRTC on a host server with connected servers or clients. In its simplest form, it's a local host that is accessed through a local IP address and the network ports 80, 8124, and 8888.

Video card

NVIDIA GPUs—beginning with the Kepler generation—contain a hardware-based encoder called NVENC that provides fully accelerated, hardware-based video encoding independent of graphics performance. With the computationally complex task of complete encoding offloaded to NVENC, the graphics engine and the CPU are freed up for other operations. NVENC makes it possible to:

- Encode and stream games and applications at high quality and ultra-low latency without utilizing the CPU
- Encode at very high quality for archiving, OTT streaming, and web videos
- Encode with ultra-low power consumption per stream (watts/stream)

The video encoding hardware NVENC can be accessed using the NVIDIA Video Codec SDK. This dedicated accelerator supports hardware-accelerated encoding of the many popular video codecs on Windows and Linux.

For more information, see the NVIDIA Video Codec SDK documentation.

With the development of standards and greater availability of GPUs and drivers, Unreal Engine supports AMD options for Pixel Streaming as of Unreal Engine 4.24. We will continue to explore additional hardware support options for future releases.

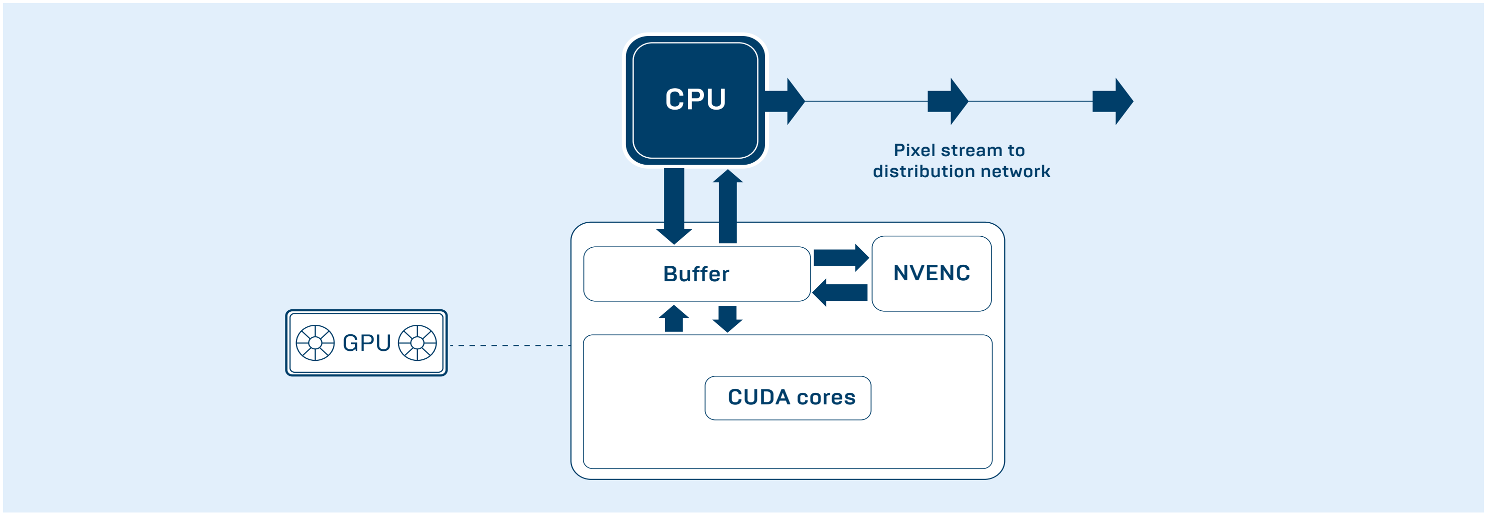


Figure 4: Hardware-based encoding and generation of pixel stream. The NVENC encoder works with the CPU and CUDA cores via the buffer to generate the pixel stream, which is then pushed out via the CPU to the distribution network.

Use case setups for Pixel Streaming

Here we take a look at some common setups for Pixel Streaming, and the companion technologies that make it work. For more information, see the Hosting and Networking Guide topic in the Unreal Engine Pixel Streaming documentation.

Note that if host and client are separated by a network firewall, you might need a STUN (Session Traversal Utilities for NAT) and TURN (Traversal Using Relays around NAT) server to establish communication between the host and participants. Some client firewalls will not allow access to/from the host service due to general restrictions. For such situations, a STUN/TURN server tries a variety of accepted methods to gain a valid communication line between the host and requesting client. In this way, the STUN/TURN server removes a barrier for users behind firewalls, who would otherwise have to contact their system administrator for special permissions every time they want to use the streaming app.

Collaboration inside a LAN

In this most basic sharing setup, the host and two or more parties who need to share a development environment all reside in the same local area network (LAN). The content is streamed through the included server plugin and relies on the WebRTC API for communication between the host and participants.

Note that if the LAN is segmented by firewalls, you might need a STUN/TURN server.

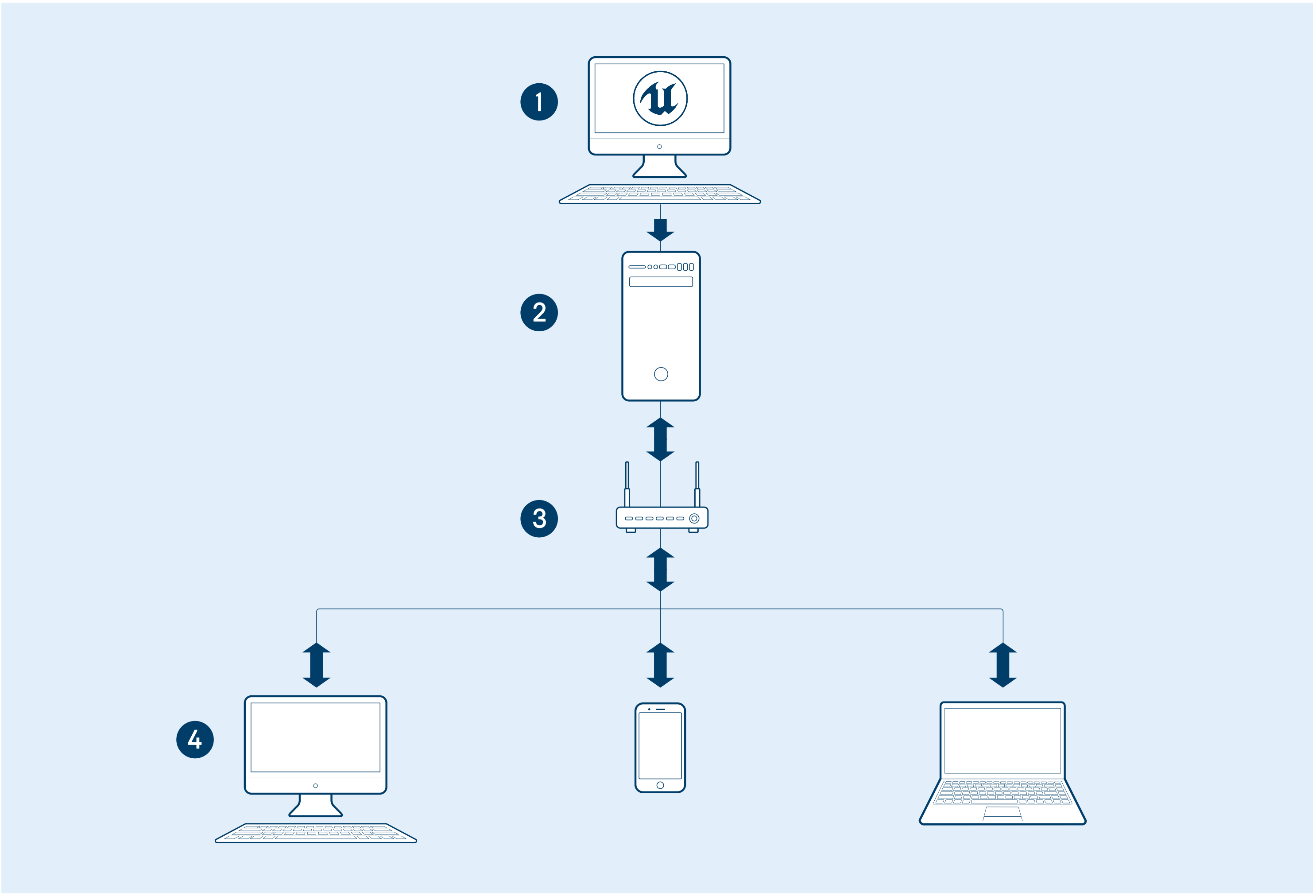


Figure 5: Simple collaboration setup inside a LAN
Key: 1. Unreal Engine development 2. UE4 server app on LAN with WebRTC 3. Router 4. User displays/interaction

Communication, approvals, presentations

For situations where a high-fidelity system is necessary to discuss a project or present to specific, known remote participants in separate LANs behind their own firewalls, Pixel Streaming offers a safe way to communicate (no product data needs to be transferred). In such a setup, a STUN/TURN server is necessary to handle communication between the host and clients.

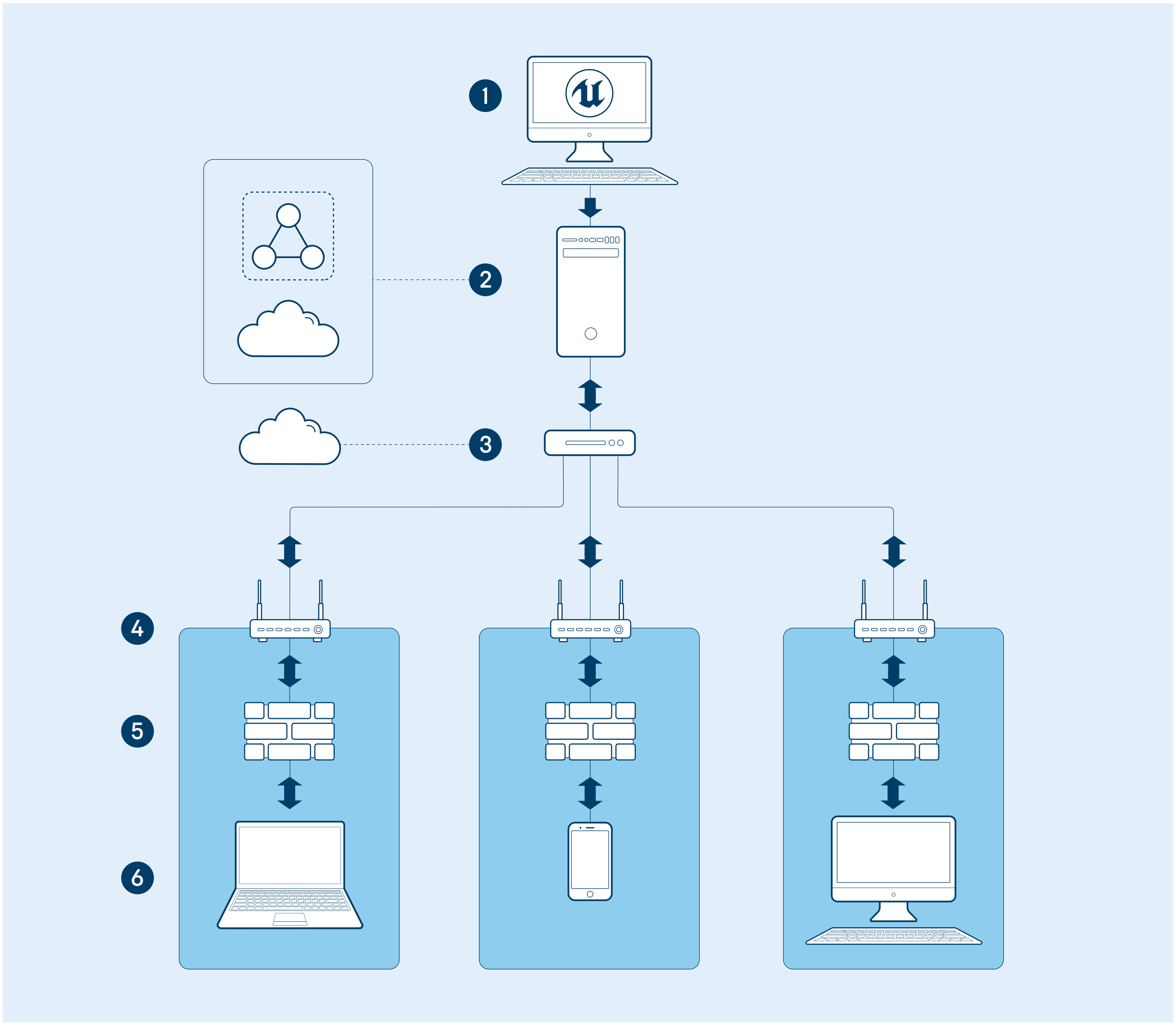


Figure 6: Single interface connection from on-premise network or cloud
Key: 1. UE4 development 2. UE4 server app in on-premise network or cloud 3. STUN/TURN server in cloud 4. Router/entrance to client network 5. Firewall 6. Display/interaction devices

Remote collaborative design, shared experiences, consumer applications, configurators

If an application needs to reach a high number of end users with diverse hardware and software setups, Pixel Streaming can run in scalable cloud environments. This is a typical setup for car configurators or other shared experiences that need safe data access and trackable experiences. This type of setup is also suitable for remote teams working together on a product design.

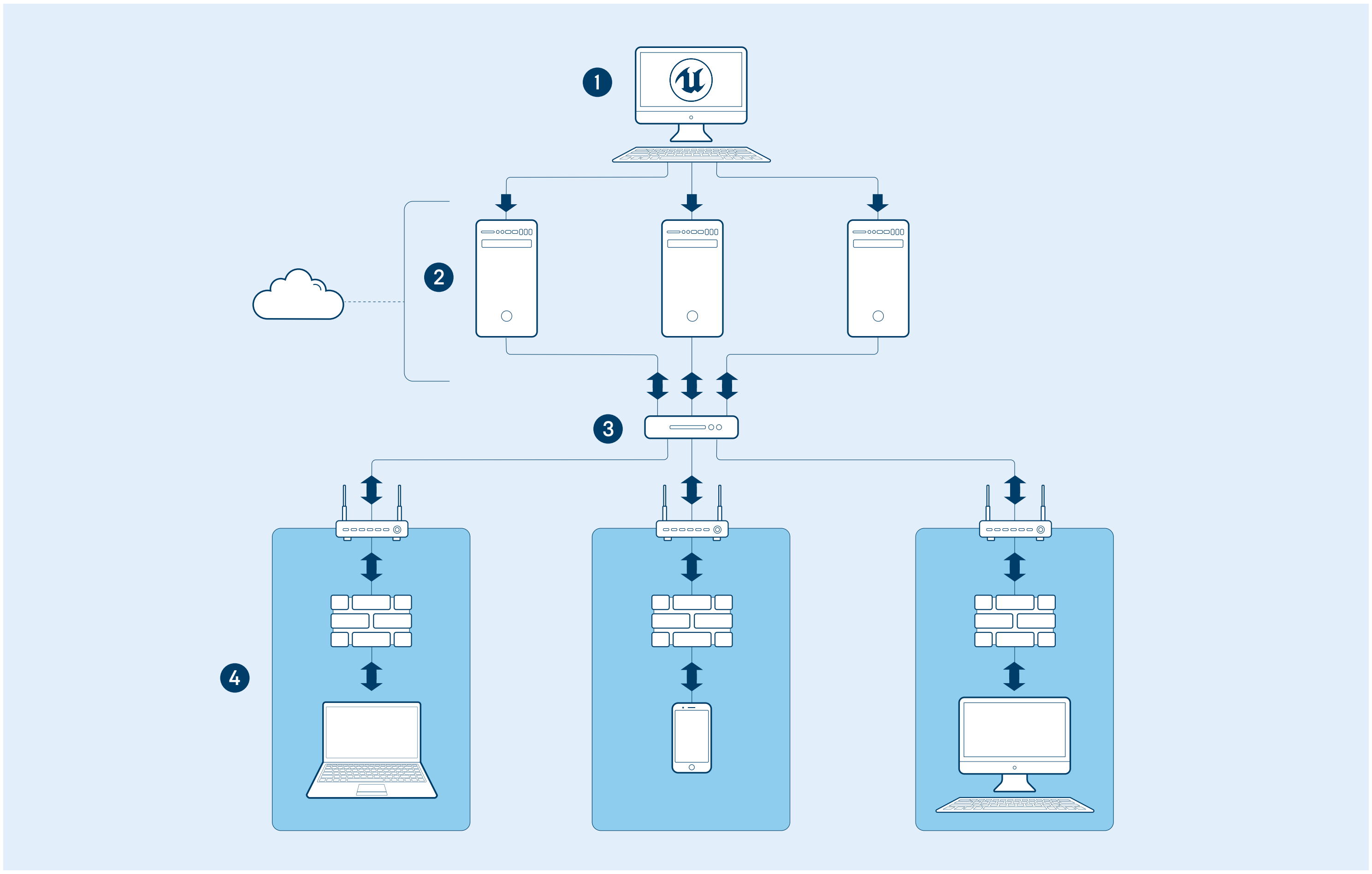


Figure 7: Unique single experience for every user
Key: 1. Unreal Engine development 2. UE4 server apps with Web RTC in cloud 3. STUN/TURN server 4. User displays/interaction

In most commercial use cases, each user will need his or her own interactive experience and stream. The system runs a separate stack of Pixel Streaming components for each user, directing each user to a separate web server and (possibly) host.

Each stack needs a unique identifier and port to control the experience. Many consumer-level graphics cards can only run a maximum of two encoders at the same time, limiting the number of instances that can be run on the computer—professional-grade cards such as the NVIDIA Quadro and Tesla, or cloud-based GPU instances (AWS), do not have the same limitation.

A matchmaker system can take care of redirecting each requester to its own signaling and web server, which sets up the connection between the client and its WebRTC proxy server. The matchmaker maintains the stream to a single user as long as the user stays active on the server. The matchmaker component can be found within Unreal Engine as well as the other server components. For more information, see the *Multiple Full Stacks with Matchmaking* section within the Hosting and Networking Guide topic in the Unreal Engine Pixel Streaming documentation.

Summary

In considering possible solutions for streaming high-quality Unreal Engine content to multiple types of devices, Epic Games researched the options and chose to develop Pixel Streaming.

WebGL is inexpensive when deploying in current host environments, and might be enough for simple use cases. Epic is working on a WebGL exporter for Unreal Engine, but it will be suitable only for simple implementations.

HTML5 solutions are already at the limit of providing benefits, and development in this area is not headed toward commercial use. The deployment of these variations will depend on custom development with tight focus on the target use case and application.

Pixel Streaming addresses all the user and technical considerations for distributing content in the Age of Experience, where the focus is on deployment of the best content at the highest fidelity available. Pixel Streaming can share high-end experiences without consideration of the end client or platform, and will work within larger platform setups that have various channels for both offline or online content.

Next steps

To understand the setup of Pixel Streaming in UE4, see this showcase setup:

https://launcher-website-prod07.ol.epicgames.com/ue/learn/pixel-streaming-demo

The step-by-step instructions are here:

https://docs.unrealengine.com/en-US/Resources/Showcases/PixelStreamingShowcase/index.html

About this document

Author

Heiko Wenczel

Contributors

Sébastien Miglio

Marco Anastasi

Editor

Michele Bousquet