



REALIZING RAPID CONCEPTUAL DESIGN WITH KITBASHING

Use kitbashing to get your game off the ground quickly.



With its origins in real-world modelmaking and special effects, kitbashing is the process of combining different assets or elements to create something original and new. The idea comes from modeling hobbyists, who would mix together (bash) a variety of model kits to build their own custom projects like the models seen in some of your favorite blockbuster sci-fi movies.

In the context of game development, kitbashing harnesses these principles to enable small teams and solo developers to produce prototypes and polished final projects much more quickly than they could by building everything from scratch. Even experienced creators, such as those behind *Lost in Random* or *The Falconeer*, leverage the Unity Asset Store to speed up development and artistry processes.

In this e-book, we'll take you from the ground up as we create a fresh take on a sci-fi environment by mixing a gloomy, dark, and wet space at the ground level, then shifting into a warm, dry skyline as the camera ascends. We achieved this dramatic mood in Unity's [High Definition Render Pipeline \(HDRP\)](#) by kitbashing assets from the [Unity Asset Store](#). Throughout this guide, we'll share an overview of our production processes and highlight some of the assets, tools, and effects we used. We hope you'll walk away with a few ideas to take your projects and prototypes to the next level by kitbashing assets.

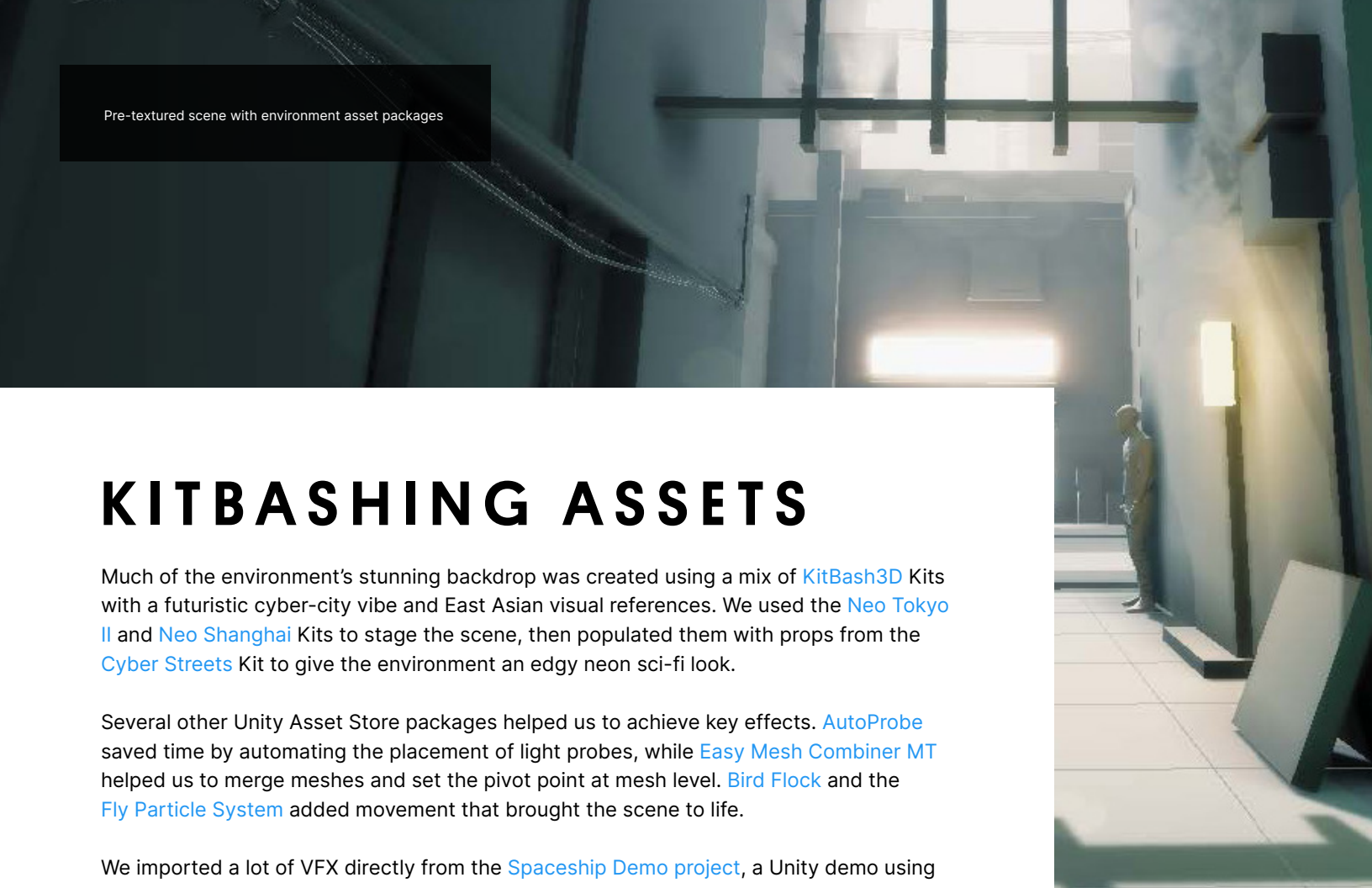


THE KITBASHING PIPELINE

Our production unfolded in three phases. First, we developed a creative concept and scoped assets to support our art direction. We added these packages to a library scene for quick development, then mocked up a greybox with basic lighting. We split the project into three scenes, which allowed team members to work independently on the backdrop, lighting, and core environment. In the second phase, we started with our greybox blockout, then subbed in the assets. Finally, we polished the demo by adding lighting, VFX, and shader updates.

For this project, we wanted to create the look and feel of an in-engine captured game trailer or an AAA game opening cinematic. This goal allowed our team to push all the visual features and settings to the max to achieve quality and fidelity.

The level of detail (LOD) levels are pushed out or set to 1, and we used a lot of VFX, lighting, and planar reflection volumes. [Forward rendering](#) is turned on, which lets us use [Multisampling anti-aliasing](#) (MSAA) and additional temporal anti-aliasing (TAA). For prototyping, these options are easy to implement and allow for fast iteration. Keeping our scene clean also lets us turn off expensive elements like VFX and planar reflections during production, so we could continue developing and iterating quickly.



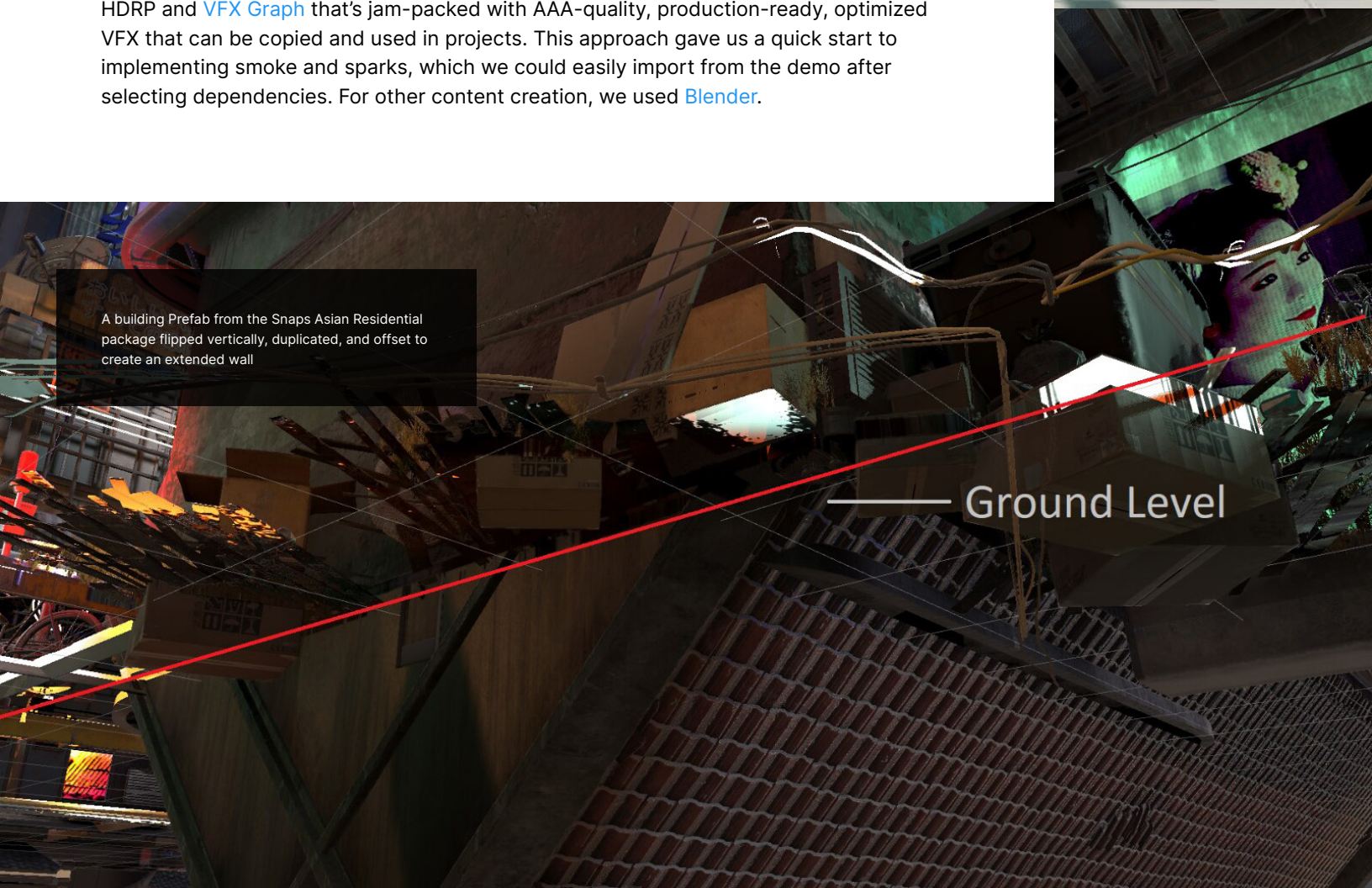
Pre-textured scene with environment asset packages

KITBASHING ASSETS

Much of the environment's stunning backdrop was created using a mix of [KitBash3D](#) Kits with a futuristic cyber-city vibe and East Asian visual references. We used the [Neo Tokyo II](#) and [Neo Shanghai](#) Kits to stage the scene, then populated them with props from the [Cyber Streets](#) Kit to give the environment an edgy neon sci-fi look.

Several other Unity Asset Store packages helped us to achieve key effects. [AutoProbe](#) saved time by automating the placement of light probes, while [Easy Mesh Combiner MT](#) helped us to merge meshes and set the pivot point at mesh level. [Bird Flock](#) and the [Fly Particle System](#) added movement that brought the scene to life.

We imported a lot of VFX directly from the [Spaceship Demo project](#), a Unity demo using HDRP and [VFX Graph](#) that's jam-packed with AAA-quality, production-ready, optimized VFX that can be copied and used in projects. This approach gave us a quick start to implementing smoke and sparks, which we could easily import from the demo after selecting dependencies. For other content creation, we used [Blender](#).



A building Prefab from the [Shaps Asian Residential](#) package flipped vertically, duplicated, and offset to create an extended wall

— Ground Level

SHADERS AND DECALS

We used the Lit Shader, Layered Lit Shader, and Decal Shader on nearly every material in this project, and we created simple custom shaders for emissive signs and decals.

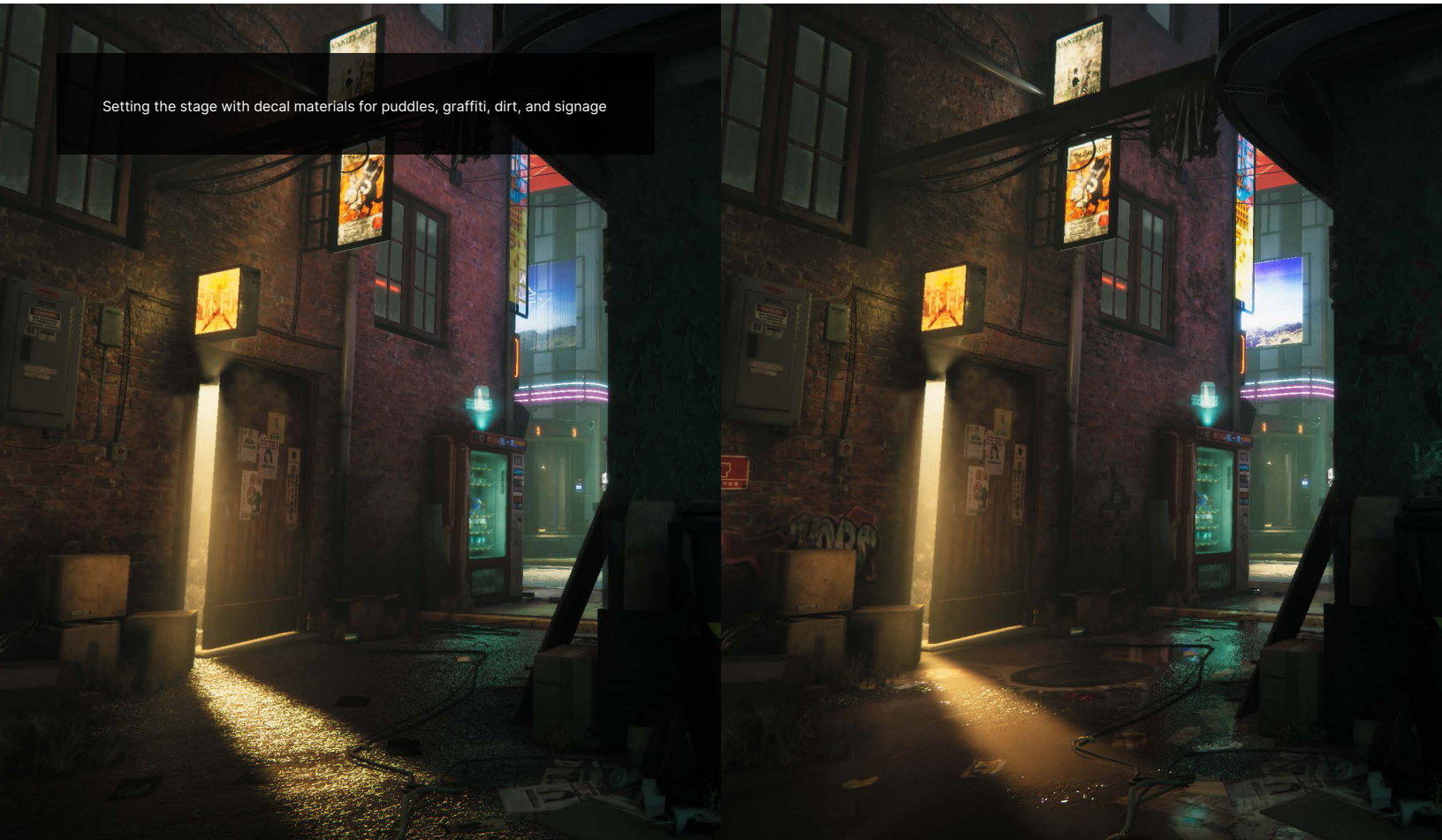
Decals add an extra layer of polish to our project. Many of the effects we used them for could also be achieved within shaders, such as dirt build-up or crack details, but the decals were a faster solution for our tight production timeline, and they gave us more control over placement.

Several of the environment packages we used include graphic elements like graffiti, utility signs, and ads, which can be turned into decals. Package materials came with normals and alpha channels, so creating projected decals was just a matter of plugging textures into the [Decal shader](#).

We created a few different decal materials to give our scene greater depth. We used them for puddles, graffiti, dirt, and signage, sometimes playing with opacity levels and sort order to allow graffiti and dirt to render above a decal layer.

Puddles were created as decals using the [Shader Graph](#). This required placement control to introduce randomness and break up repetitive tiling. The shader would take three generated puddle shapes, then lerp between each mask based on its world position to give the puddles a random shape, and there were additional controls to let us tweak the values for puddles' smoothness, shape, color, and opacity.

We added more life to the scene by pulling VFX from demo tools. Smoke and sparks were adapted from the [Spaceship Demo](#), while we grabbed the hologram's core from Unity's [Visual Effect Graph Samples project](#). The hologram model uses a depth-based camera to render out, so you can project any correctly tagged asset – we created some basic characters and a quick looping animation.



Setting the stage with decal materials for puddles, graffiti, dirt, and signage

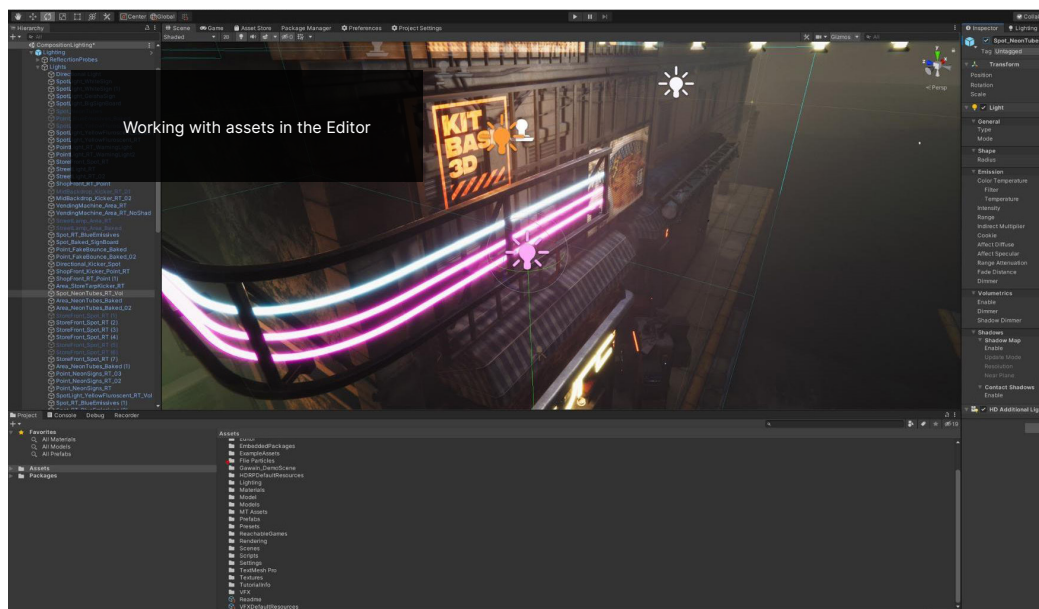


LIGHTING

Lighting is essential to the mood and composition we wanted for this demo. Since the project was rendered as an offline cinematic, we focused on achieving the right atmosphere and color transitions for our storytelling over optimization concerns. We worked on lighting in tandem with creating the environment across numerous iterations, but the compartmentalized structure of the scene let us play with different lighting ideas and iterate on them quickly.

HDRP provides tools and parameters that let us push the scene's realism – details like giving neon signs and billboards emissive material for indirect lighting contributions as well as real-time lights that bring a layer of specular lighting to nearby objects. We used the highest shadow filtering settings available in HDRP alongside screen space effects like [Contact Shadows](#) and [Screen Space Ambient Occlusion \(SSAO\)](#) to flesh out the scene's details.

Volumetric fog and density volumes were used to unify the scene's elements and create depth. HDRP allows for high-quality volumetric fog settings to achieve smooth results, but since this demo was created as an offline rendered cinematic, we also “cheated” by placing point lights near some of the more prominent neon signs to artificially boost the local volumetric effect.



POST-PROCESSING AND POLISHING

These are some of the [HDRP features](#) and [post-processing effects](#) that we used to create our demo's finishing touches.



Creating a wet, gloomy vibe with reflections

Reflections

We placed reflection captures all over the scene to make metallic and glass surfaces reflective, while planar reflections on water puddles and shiny floors mirrored details of the surrounding scene. HDRP and planar reflections allow captures of up to 8K, while layers and cullings systems offer the flexibility to define what will be captured onto those reflections. We used the layers system to create clean results with moving cameras and planar reflections that were updated each frame.

A few specular-only lights were added to create rim highlights around some of the more eye-catching props in the scene, making them more prominent against the background while placing them to subtly lead the eye to key points in the scene.



Volumes and visual effects

We used HDRP's volume system throughout the scene, both to control screen-space effects and for localized color grading. We controlled the main generic overrides in the scene under the [Global Post-Processing Volume](#), so these settings are kept in a single, central location. This global volume setting made the system easier to manage, especially for effects that don't change much in different regions of the scene and can be controlled by a single value, including Chromatic Aberration, Ambient Occlusion, Fog, Bloom, Film Grain, and Vignette.

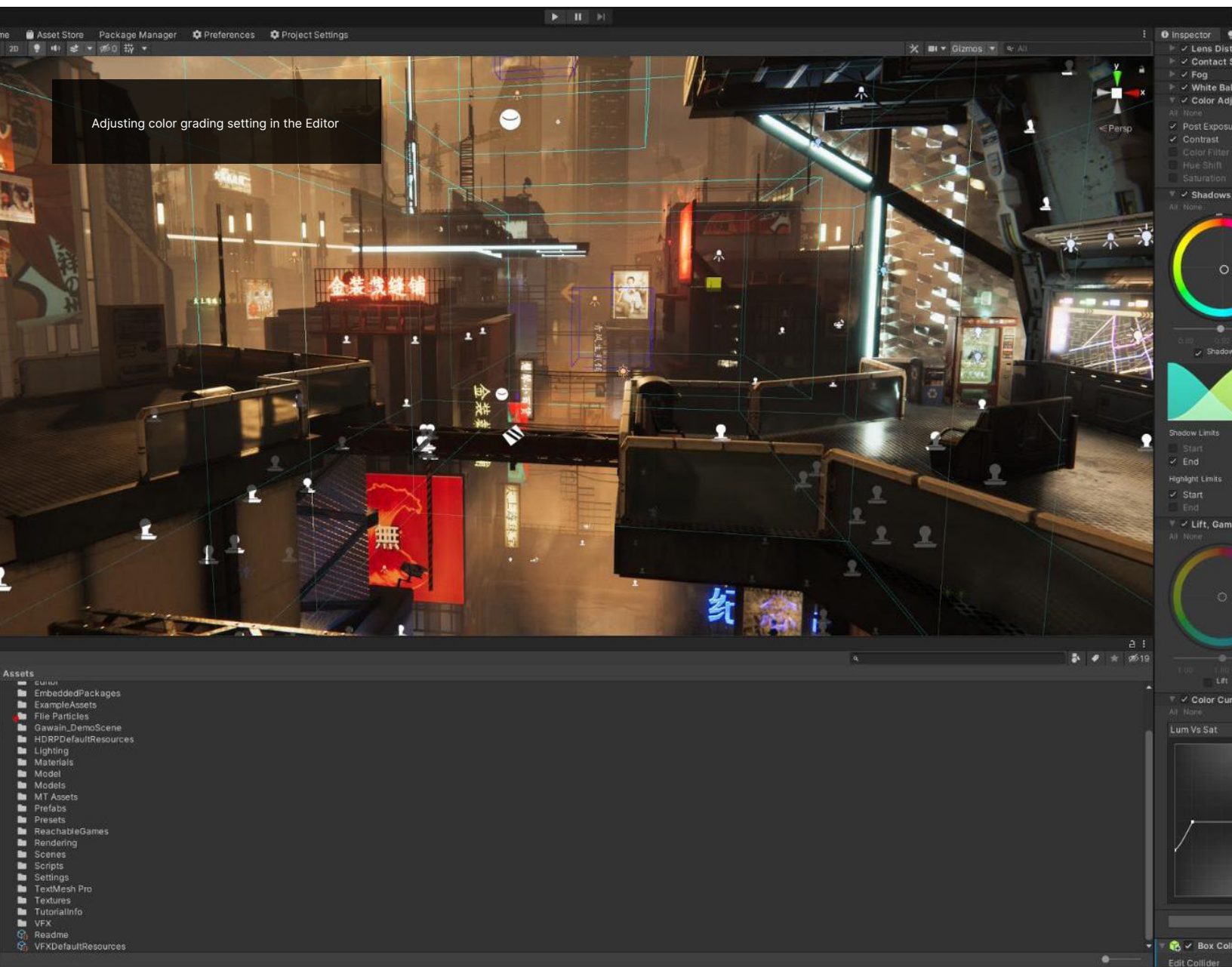
Screen effects like [Film Grain](#), [Vignette](#), and [Chromatic Aberration](#) – all designed to mirror imperfections of a camera lens and film – help CG to look more realistic by preventing the image from looking unnaturally clean. Use very small values in these effects' settings, because a little goes a long way, but these details really help to sell the cinematic feel of the image.

Color grading

Color grading was essential to this demo's art direction. We used localized volumes, one for the ground area and one closer to the top of the scene, and a large blend distance made the transition from one volume to the next seamless.

HDRP's color grading overrides are robust and offer lots of options to fine-tune the colors, saturation and luminance across highlights, midtones, and shadows. In color grading, simplicity is key – only adjust what's needed, rather than overriding everything and ending up with an overly complex grade.

We did most of the color grading using [Color Adjustments](#) for broad saturation and contrast adjustments. Next, we used the [Lift](#), [Gamma](#), [Gain](#) and [Shadows](#), [Midtones](#), [Highlights](#) effects to color shift and adjust the range, finally making more minute adjustments with [Color Curves](#). To create some nice contrast at the ground level, for example, we color-shifted the shadows to a warmer tone compared to the cool hues of the midtones and highlights. We used the color curve to override the Luminance Vs Saturation curve, so the shadows were picked out and desaturated to enhance subtle contrasts.





GET YOUR GAME OFF THE GROUND

By using kitbashing techniques and combining different asset packs, you can create your own unique experiences and immersive worlds, develop your game faster, and focus on creative rather than technical challenges.

Art creation is one of the most time-consuming tasks in game development. Sourcing third-party art assets that work with your project's vision and specifications can be challenging, but the [Unity Asset Store](#) has your back with tens of thousands of community-powered art and tools, including themed collections to give your project or prototype a consistent look and feel.

Find something that inspires you, and get creating!

Thanks to the incredible technical team who made this demo and shared the details of how they did it, particularly art director Elliott Cottell.

