Introduction

Integrating web content, such as Shopify's Storefront Web Components, into a Unity application offers a powerful way to create dynamic e-commerce experiences within your games and apps. The method for this integration fundamentally depends on your target platform. This guide provides a comprehensive overview of the two primary approaches:

1. **WebGL Integration via JavaScript Interoperability**: For projects targeting WebGL, you can directly interface with the browser's JavaScript environment. This is the most direct and performant method for web-based builds.
2. **Standalone and Mobile Integration via WebView**: For standalone (Windows, macOS) and mobile (iOS, Android) applications, a WebView plugin is necessary. This component embeds a web browser within your Unity application, allowing you to render web content in a native environment.

Choosing the Right Method

The following table outlines the key differences between the two methods to help you decide which is best suited for your project:

Feature	WebGL (JavaScript Interop)	Standalone/Mobile (WebView)
Target Platforms	Web Browsers (via WebGL)	Windows, macOS, iOS, Android
Core Technology	Direct C# to JavaScript calls (`.jslib`)	Embedded browser (e.g., Chromium)
External Assets	None (built-in Unity feature)	Requires a third-party plugin (e.g., Vuplex 3D WebView, UniWebView)
Performance	High (direct communication)	Moderate (overhead from the embedded browser)
Setup Complexity	Moderate (requires manual JavaScript and HTML template modification)	Easy to Moderate (prefab-based setup, but requires plugin installation)

Rendering	Renders outside the Unity canvas, in the HTML DOM	Renders to a texture within the Unity scene (3D or 2D UI)
Best For	Web-based games, applications where the Unity content and web content can coexist on the same page.	Native applications that need to display web content, in-game stores, news feeds, or complex UI that is easier to build with web technologies.

Method 1: WebGL Integration via JavaScript Interoperability

This method is for projects targeting the WebGL platform. It uses Unity's built-in JavaScript interoperability to communicate between your C# scripts and the Shopify components running in the browser.

Prerequisites

- Unity Hub and a recent version of the Unity Editor (2020.1 or newer recommended).
- A Shopify store with the Headless channel installed (optional, for full functionality).

Step 1: Project Setup

1. **Create a new Unity Project**: Open Unity Hub and create a new 3D project.
2. **Switch to WebGL Platform**: In Unity, go to **File > Build Settings**, select **WebGL**, and click **Switch Platform**.
3. **Create Plugin Folder**: In the **Project** window, create a new folder structure: `Assets/Plugins/WebGL`.

Step 2: Create the JavaScript Plugin (`.jslib`)

Inside the `Assets/Plugins/WebGL` folder, create a new file named `ShopifyIntegration.jslib`. This file will contain the JavaScript code that loads the Shopify web components and provides functions that can be called from your C# scripts.

JavaScript

```
mergeInto(LibraryManager.library, {

  Shopify_Init: function (storeDomain, publicAccessToken) {
    // Convert parameters to JavaScript strings
    var storeDomainStr = UTF8ToString(storeDomain);
```

```
    var publicAccessTokenStr = UTF8ToString(publicAccessToken);

    // Check if the script is already loaded
    if (document.getElementById('shopify-web-components-script')) {
        console.log('Shopify script already loaded.');
        return;
    }

    // Create and append the Shopify web components script tag
    var script = document.createElement('script');
    script.id = 'shopify-web-components-script';
    script.src = 'https://cdn.shopify.com/storefront/web-components.js';
    script.onload = function() {
        console.log('Shopify web components script loaded.');
        // Create and configure the shopify-store element
        var shopifyStore = document.createElement('shopify-store');
        shopifyStore.setAttribute('store-domain', storeDomainStr);
        if (publicAccessTokenStr) {
            shopifyStore.setAttribute('public-access-token',
publicAccessTokenStr);
        }
        document.body.appendChild(shopifyStore);
    };
    document.head.appendChild(script);
  },

  Shopify_CreateProductCard: function (productHandle, parentElementId) {
    var productHandleStr = UTF8ToString(productHandle);
    var parentElementIdStr = UTF8ToString(parentElementId);

    var parentElement = document.getElementById(parentElementIdStr);
    if (!parentElement) {
        console.error('Parent element not found: ' + parentElementIdStr);
        return;
    }

    // Create the shopify-context and template for the product card
    var context = document.createElement('shopify-context');
    context.setAttribute('type', 'product');
    context.setAttribute('handle', productHandleStr);

    var template = document.createElement('template');
    template.innerHTML = `
        <div style="border: 1px solid #ccc; padding: 16px; margin: 16px; max-
width: 300px;">
            <h1><shopify-data query="product.title"></shopify-data></h1>
            <shopify-media query="product.featuredImage" style="max-width:
100%;"></shopify-media>
```

```
            <p><shopify-data query="product.descriptionHtml"></shopify-data>
</p>
            <p>Price: <shopify-money
query="product.selectedOrFirstAvailableVariant.price"></shopify-money></p>
            <button onclick="document.querySelector('shopify-
store').buyNow(event)">Buy Now</button>
        </div>
    `;

    context.appendChild(template);
    parentElement.appendChild(context);
  }

});
```

Step 3: Create the C# Script

Now, create a C# script in your Unity project (e.g., `Assets/Scripts/ShopifyController.cs`) to call the JavaScript functions defined in your `.jslib` file.

C#

```
using UnityEngine;
using System.Runtime.InteropServices;

public class ShopifyController : MonoBehaviour
{
    [DllImport("__Internal")]
    private static extern void Shopify_Init(string storeDomain, string
publicAccessToken);

    [DllImport("__Internal")]
    private static extern void Shopify_CreateProductCard(string
productHandle, string parentElementId);

    public string shopifyStoreDomain = "your-store.myshopify.com";
    public string publicAccessToken = "your-public-access-token"; // Optional

    void Start()
    {
#if !UNITY_EDITOR && UNITY_WEBGL
        Shopify_Init(shopifyStoreDomain, publicAccessToken);
#endif
    }

    public void CreateProductCard(string productHandle)
    {
```

```
#if !UNITY_EDITOR && UNITY_WEBGL
        // The parent element ID should match an element in your WebGL
template HTML file
        Shopify_CreateProductCard(productHandle, "product-container");
#endif
    }
}
```

Step 4: Modify the WebGL Template

When you build your WebGL project, Unity uses an HTML template. You need to add a container element to this template where the Shopify product cards will be injected.

1. In your `Assets` folder, create a new folder named `WebGLTemplates`.
2. Copy the default WebGL template from the Unity Editor installation path to your new `WebGLTemplates` folder. You can find the default template at:
   - Windows: `Editor\Data\PlaybackEngines\WebGLSupport\BuildTools\WebGLTemplates\Default`
   - macOS: `/Applications/Unity/Hub/Editor/<version>/PlaybackEngines/WebGLSupport/BuildTools/WebGLTemplates/Default`
3. Rename the copied template folder (e.g., to `ShopifyTemplate`).
4. Open the `index.html` file inside your new template folder.
5. Add a `div` element with the `id="product-container"` inside the `<body>` tag, for example:

HTML

```
<body>
    <div id="unity-container" class="unity-desktop">
        <canvas id="unity-canvas" width=960 height=600></canvas>
        ...
    </div>
    <div id="product-container"></div>
</body>
```

Step 5: Build and Run

1. **Configure Build Settings**: Go to **File > Build Settings**, and under **Player Settings > WebGL > Resolution and Presentation**, select your `ShopifyTemplate` from the **WebGL Template** dropdown.
2. **Build the Project**: Click **Build** and choose a folder to save your build.

3. **Run a Local Server**: Due to browser security restrictions, you need to run the build from a local web server. You can use Python's built-in server for this:
    - Open a terminal or command prompt in your build folder.
    - Run `python -m http.server` (for Python 3) or `python -m SimpleHTTPServer` (for Python 2).
4. **Open in Browser**: Open your web browser and navigate to `http://localhost:8000`.

Step 6: Triggering from Unity

To test the integration, you can create a simple UI button in your Unity scene that calls the `CreateProductCard` method on your `ShopifyController` script. When you click the button in the running WebGL build, it will call the JavaScript function and create the product card on the webpage.

Method 2: Standalone & Mobile Integration via WebView

This method is for projects targeting standalone (Windows, macOS) and mobile (iOS, Android) platforms. It uses a WebView plugin to render web content within your Unity application.

Why Use a WebView?

Unlike WebGL builds that run in a browser, standalone and mobile applications cannot directly execute web scripts like JavaScript. A WebView is a component that renders web content within a native application, essentially embedding a browser window into your Unity scene. This is the standard method for displaying web-based content on these platforms.

Choosing a WebView Plugin

Several WebView plugins are available on the Unity Asset Store. For this guide, we will focus on **Vuplex 3D WebView** due to its cross-platform support and extensive features. Another popular option, especially for mobile-focused projects, is **UniWebView**.

- **Vuplex 3D WebView**: A comprehensive, cross-platform solution for Windows, macOS, iOS, Android, and more.
- **UniWebView**: A modern, mobile-focused WebView component for iOS and Android.

Step 1: Install Vuplex 3D WebView

1. **Purchase and Import**: Purchase the appropriate Vuplex 3D WebView package from the Unity Asset Store (packages are sold per platform). Download and import it into your

Unity project.

2. **Follow Setup Instructions**: Vuplex provides platform-specific setup instructions. Follow the prompts to ensure the plugin is correctly configured for your target platform.

Step 2: Set Up the WebView in Your Scene

Vuplex provides easy-to-use prefabs for both 3D and 2D setups.

1. **For 3D Scenes**: Drag the `WebViewPrefab` from the `Vuplex/WebView/Prefabs` folder into your scene. This will create a 3D plane that renders the web content.
2. **For 2D UI**: If you are working with a Unity UI Canvas, drag the `CanvasWebViewPrefab` into your Canvas. This will create a UI element that renders the web content.

Step 3: Create the C# Controller Script

Create a C# script (e.g., `Assets/Scripts/WebViewShopifyController.cs`) to control the WebView. This script will construct the HTML string and load it into the WebView.

C#

```
using UnityEngine;
using Vuplex.WebView;

public class WebViewShopifyController : MonoBehaviour
{
    public CanvasWebViewPrefab webViewPrefab;

    public string shopifyStoreDomain = "your-store.myshopify.com";
    public string publicAccessToken = "your-public-access-token"; // Optional
    public string productHandle = "your-product-handle";

    async void Start()
    {
        if (webViewPrefab == null)
        {
            Debug.LogError("WebViewPrefab is not assigned!");
            return;
        }

        // Wait for the WebView to be ready
        await webViewPrefab.WaitUntilInitialized();

        // Construct the HTML content
        string htmlContent = GetShopifyHtml(productHandle);

        // Load the HTML into the WebView
```

```
        webViewPrefab.WebView.LoadHtml(htmlContent);
    }

    private string GetShopifyHtml(string productHandle)
    {
        // Note the use of single quotes for attributes within the string
        return $@"<!DOCTYPE html>
<html>
<head>
    <title>Shopify Integration</title>
    <meta name=\"viewport\" content=\"width=device-width, initial-
scale=1.0\">
    <script src=\'https://cdn.shopify.com/storefront/web-components.js\'>
</script>
    <style>
        body {{ font-family: sans-serif; margin: 20px; }}
        .product-card {{ border: 1px solid #ccc; padding: 16px; margin: 16px
auto; max-width: 90%; box-shadow: 0 2px 4px rgba(0,0,0,0.1); }}
        .product-card h1 {{ font-size: 1.5em; }}
        .product-card img {{ max-width: 100%; height: auto; }}
        .product-card button {{ background-color: #4CAF50; color: white;
padding: 10px 15px; border: none; cursor: pointer; font-size: 1em; }}
    </style>
</head>
<body>
    <shopify-store store-domain=\'{shopifyStoreDomain}\' public-access-
token=\'{publicAccessToken}\'>
    </shopify-store>

    <div class=\'product-card\'>
        <shopify-context type=\'product\' handle=\'{productHandle}\'>
            <template>
                <h1><shopify-data query=\'product.title\'></shopify-data>
</h1>
                <shopify-media query=\'product.featuredImage\'></shopify-
media>
                <div><shopify-data query=\'product.descriptionHtml\'>
</shopify-data></div>
                <p>Price: <shopify-money
query=\'product.selectedOrFirstAvailableVariant.price\'></shopify-money></p>
                <button onclick=\"document.querySelector(\'shopify-
store\').buyNow(event)\">Buy Now</button>
            </template>
        </shopify-context>
    </div>
</body>
</html>";
```

```
    }
}
```

Step 4: Configure the Scene

1. **Attach the Script**: Create an empty GameObject in your scene (e.g., "ShopifyManager") and attach the `WebViewShopifyController.cs` script to it.
2. **Assign the WebView Prefab**: In the Inspector for your `ShopifyManager` object, drag the `CanvasWebViewPrefab` from your scene hierarchy into the `WebView Prefab` field of the script.
3. **Enter Your Shopify Details**: Fill in your `Shopify Store Domain`, `Public Access Token` (if needed), and the `Product Handle` of the product you want to display.

Step 5: Build and Run

1. **Build Settings**: Go to **File > Build Settings**, select your target platform (e.g., Windows, Mac, & Linux Standalone, or Android/iOS), and configure your player settings.
2. **Build and Run**: Build the project and run the application. The WebView will initialize, load the HTML, and display the Shopify product card within your Unity application.

Interacting with the WebView

Vuplex provides a rich API for interacting with the WebView, including:

- **Executing JavaScript**: You can call JavaScript functions within the WebView from your C# code.
- **Message Passing**: You can send messages from JavaScript back to your C# code.

This allows for more complex integrations, such as having in-game events trigger changes in the web content, or having web events (like adding an item to the cart) trigger actions in your Unity game.

Conclusion

Integrating Shopify's e-commerce capabilities into Unity opens up a world of possibilities for in-game stores, interactive product showcases, and more. The best integration method depends entirely on your project's target platform.

- For **WebGL** projects, the direct **JavaScript interoperability** offers a lightweight and powerful way to merge Unity and web content seamlessly.

- For **standalone and mobile** applications, a **WebView plugin** like Vuplex 3D WebView is the recommended approach, providing a robust embedded browser that can render complex web content within your native application.

By following the steps outlined in this guide, you can successfully implement either method and bring the power of Shopify into your Unity projects.

References

1. Shopify Storefront Web Components
2. Unity - Manual: Interaction with browser scripting
3. Vuplex 3D WebView Documentation
4. UniWebView Documentation