Lingheng Tony Tao

TOOL PROGRAMMER / TECHNICAL ARTIST, PORTFOLIO 2025

About

Who I Am.

Lingheng Tony Tao

Programmer & Artist.

Ready to program.

Love to draw.

Wish to help.

Pittsburgh, PA

+1 412 328 9641

ltt@taotamago.com

www.taotamago.com

www.linkedin.com/in/lingheng-tao

Projects

Light Baking Tool for AlterStaff

A tool for help baking static lightmaps for procedurally generated maps.

ShaderGraph Comparison App for

A tool for comparing nodes in Unity Shader Graph, Amplify Shader Editor, and Reality Composer Pro ShaderGraph.

Otter Shaders for CMU ETC Beyond Beyond Touch

A tool for future developers to use 2D SwiftUI Shaders.

Unity Shaders for CMU ETC Building Virtual Worlds

A tool for artists to smoothly use shaders in our projects.

Simplify to **A Single Click.**

Light Baking Tool

AlterStaff Inc, Jan 2025

BACKGROUND & ABSTRACT

Replaced complex, manual baking steps with one-click automation in Unity.

TODO

AlterStaff Inc. is developing *AI2U: With You 'Til The End*, a game currently with three levels.

Levels 1 and 2 are procedurally generated, whereas Level 3 contains four predefined rooms — one of which will randomly lose power during gameplay.

We need **a dedicated tool** to streamline the lightmap baking workflow.

Problems

There were 3 different workflows for different levels.

① Trivial Levels

These levels only require Unity's built-in **Generate Lighting** button — no custom tool needed.

② Multiple Lightmap Groups

GameObjects that consistently appear together — such as items within a room type — are marked as a **`MultipleLightmapGroup`**. Each level includes around a dozen of these groups, with a random subset selected during runtime.

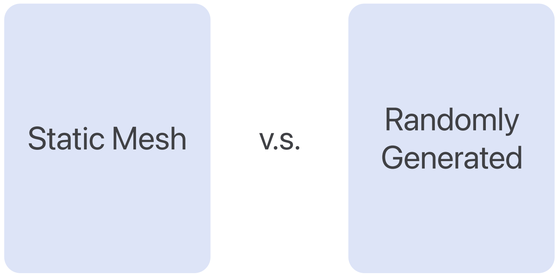
③ Different Configurations

The process involves enabling one group of GameObjects, baking, then disabling it and enabling the next group—repeating this cycle for all required groups.

Each time we enter a level, the correct workflow must be chosen. Type ② levels require adding an `Auto Baker` with strict setup and baking steps, while other levels use a `LightManager`, which even involves manually placing baked maps into the correct folder. The workflow is tightly coupled and difficult to manage.

CHALLENGES

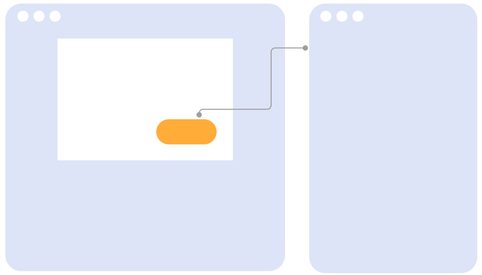
Static Light



Lightmap baking requires static geometry.

However, in procedurally generated maps, rooms are assembled at runtime and their positions may vary randomly.

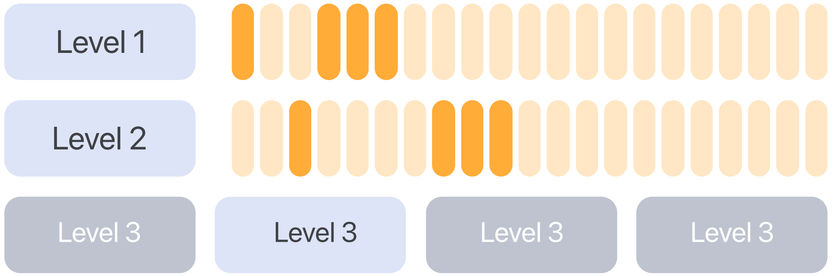
Unity Editor Windows



Since the baking process relies on scene references, it requires a `MonoBehaviour` Game Object in the scene to serve as the delegate.

This creates challenges for serialization and the design of data structures.

Unification



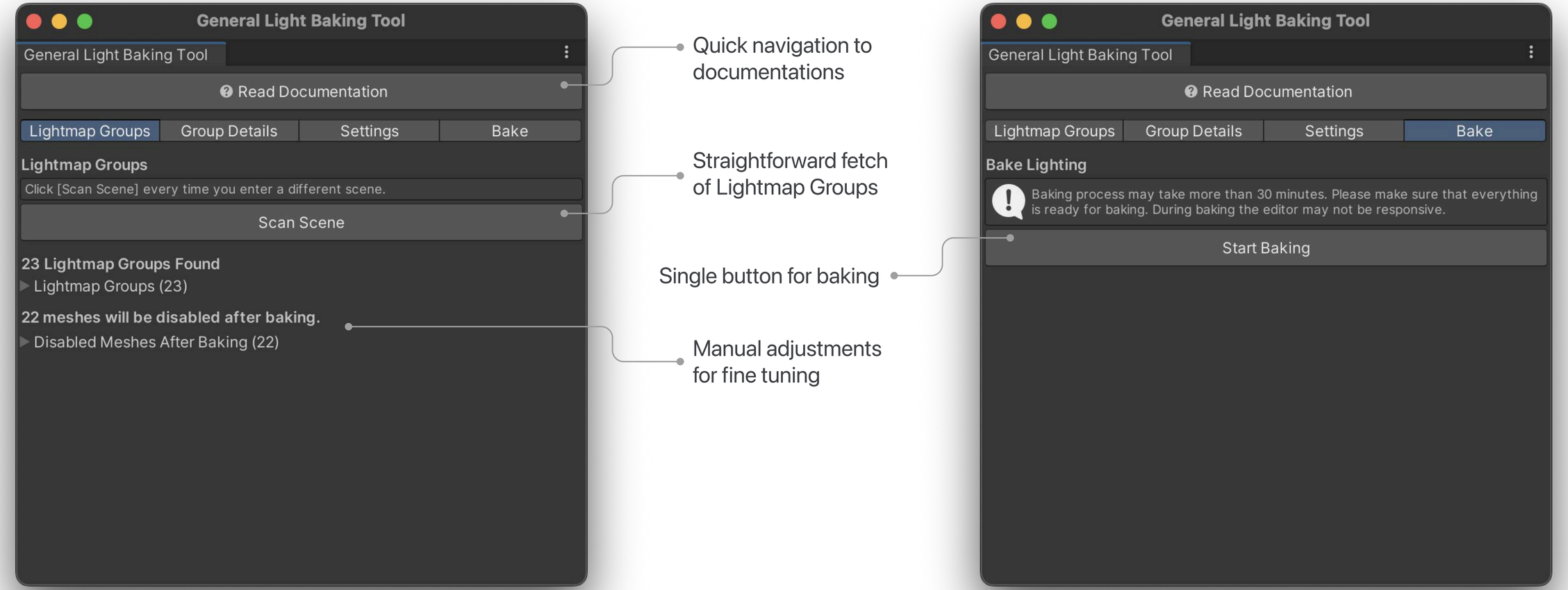
The need for a light baking tool varies by level.

In Levels 1 and 2, each of the four room slots can host different types of rooms — effectively selecting 4 from a larger pool.

In Level 3, there are four fixed room compositions, so instead of assembling lightmaps, we simply enable the appropriate setup.

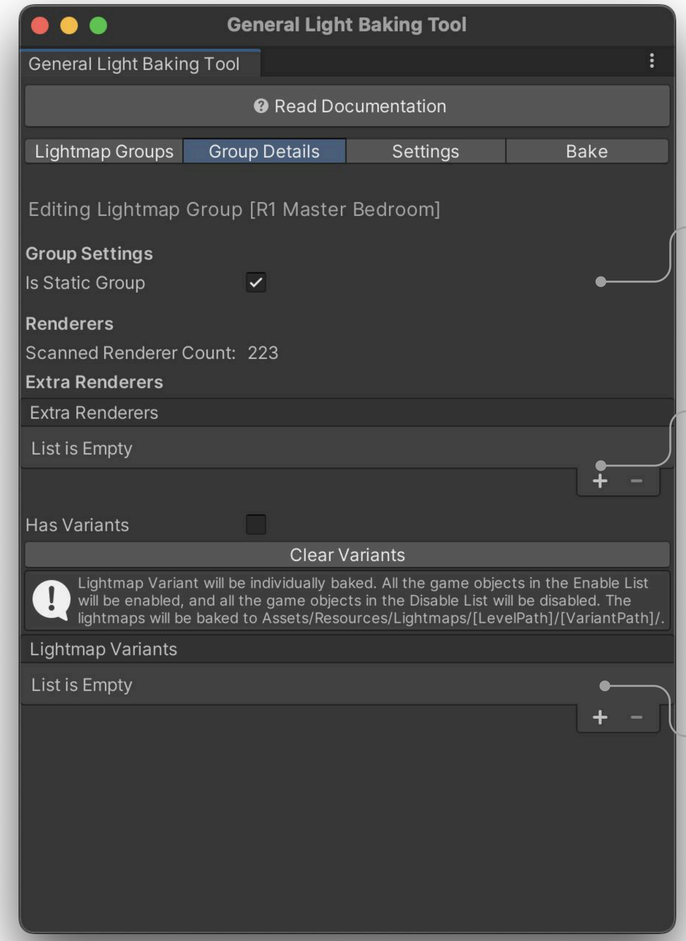
SOLUTIONS

General Light Baking Tool Editor Window



SOLUTIONS

General Light Baking Tool Editor Window



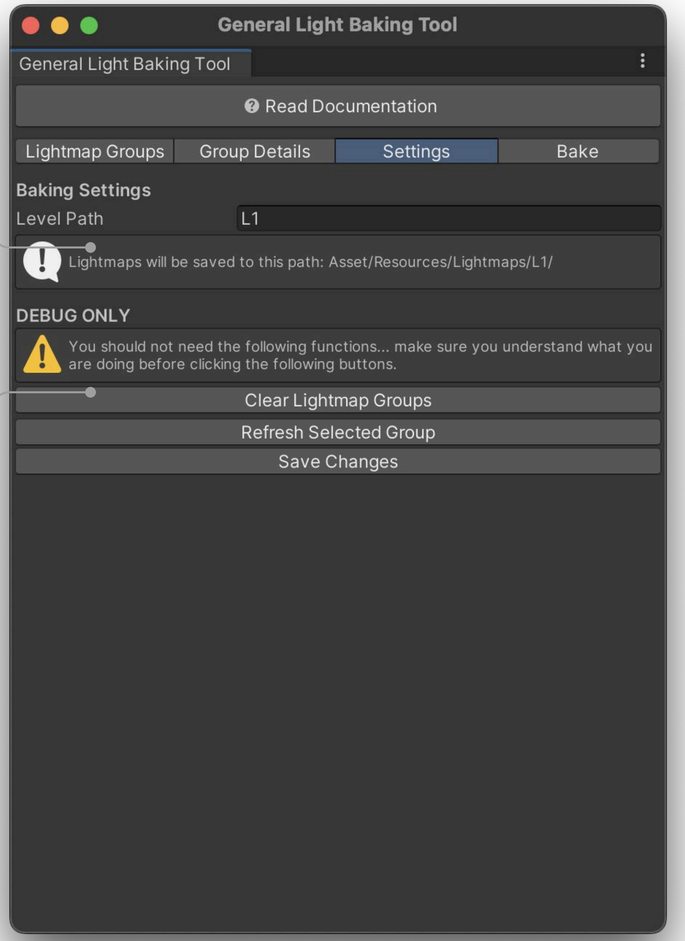
Mark Static for the Lightmap Group. This means the group will always be turning on for Light Probes to work properly.

Extra Renderers to allow the lightmap group to select exterior Renderers.

Add variants to setup different compositions of the current Lightmap Group. This allows a manual on and off for the game objects in the group.

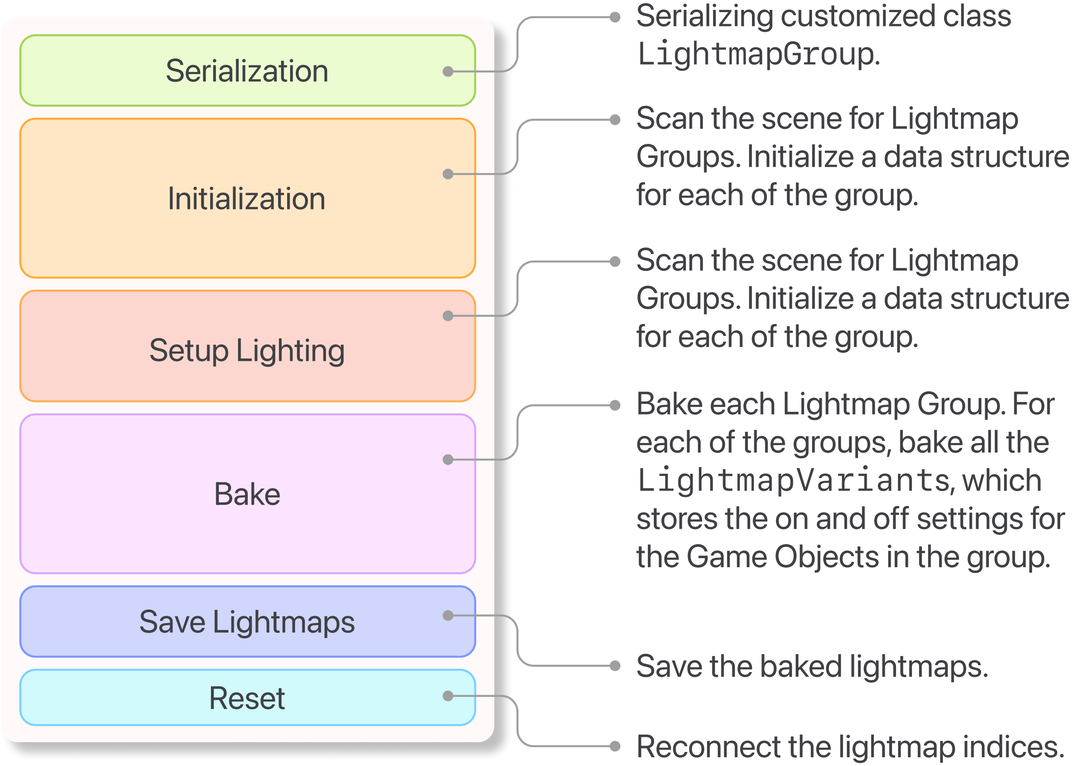
Allows manual setting of level path. This is because of our version control workflow — team members may work on their own level.

DEBUG ONLY section with warning: functions for testing and resetting.



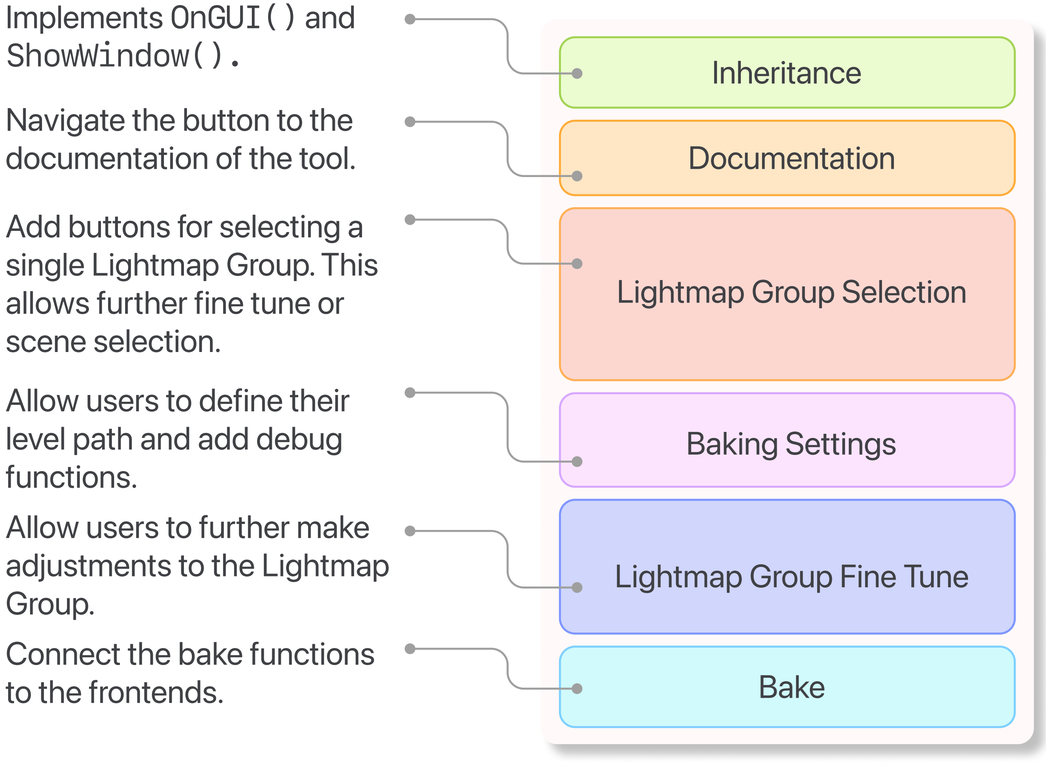
IMPLEMENTATIONS

`LightBakingTool.cs`



Tool Backends

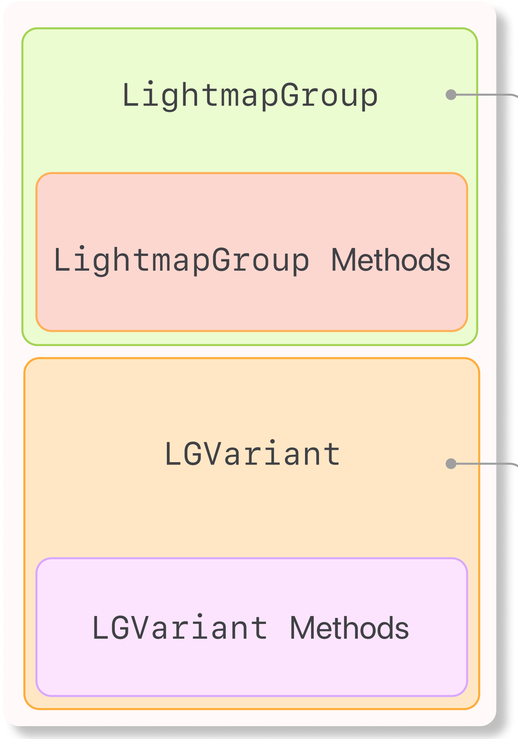
`Editor_LightBakingTool.cs`



Tool Frontends

IMPLEMENTATIONS

LightmapConfig.cs



The data structure for storing the information for a Lightmap Group, including its renderers, variants, lightmap parameters and current variant applied.

It literally means Lightmap Group Variant, which means it stores the information of the objects to enable and disable under this lighting state. The tool will bake a set of lightmaps for each of the LGVariant for a group.

Data Structures

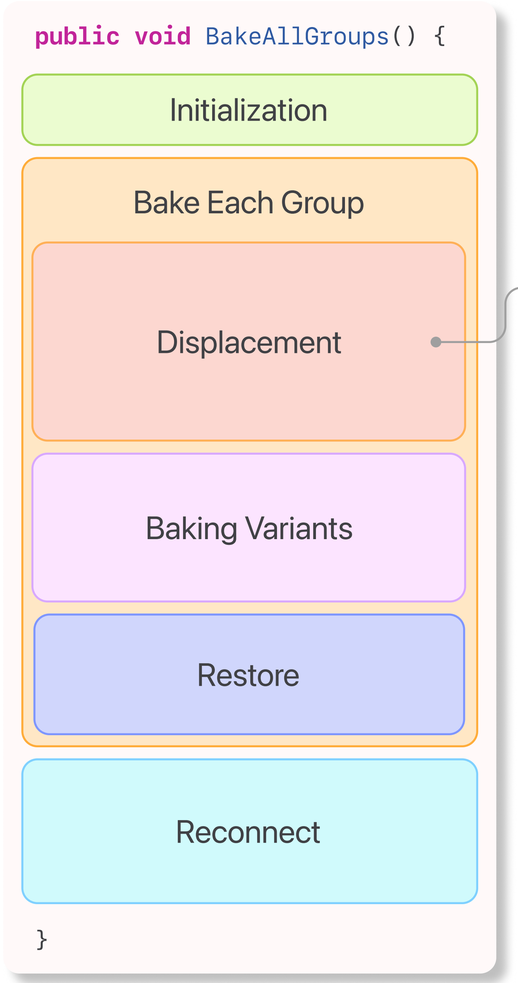
`LightmapGroup::SetVariant()`

Called when setting up the current variant for the group. It will be called O(V) times, where V is the number of variants a group possesses.

```
public void SetVariant(LGVariant variant)
{
    // make sure that variant is in the list
    if (!variants.Contains(variant))
    {
        Debug.LogWarning("No variant at: " + groupName);
        return;
    }

    // enable all the objects marked to be on in this variant
    foreach (var obj in variant.objectsToEnable)
    {
        obj.SetActive(true);
    }

    // disable all the objects marked to be off in this variant
    foreach (var obj in variant.objectsToDisable)
    {
        obj.SetActive(false);
    }

    Debug.Log($"Switching to variant [{variant.variantName}]");
    currentVariant = variant;
}
```

IMPLEMENTATIONS

`LightBakingTool::BakeAllGroups()`



Displacement

To prevent interference, each Lightmap Group is temporarily displaced vertically — except static groups, whose dominant light sources must remain untouched to ensure accurate lighting conditions.

```
foreach (var group in groups.Values)
{
    // if room is not a static room
    if (!group.isStaticGroup)
    {
        OverlappingRoomOriginalPositions.Add(group.container.transform.position);
        OverlappingRoomGroup.Add(group);
        group.container.transform.position += bakingDisplacement;
        bakingDisplacement += Vector3.up * 100;
    }

    if (!group.container.activeInHierarchy)
    {
        Debug.Log("Turning On:" + group.container.name);
        turnedOffGO.Add(group.container);
        group.container.SetActive(true);
    }

    currentIndex++;
}
```

IMPLEMENTATIONS

`LightBakingTool::BakeAllGroups()`

Lightmap Group Variant Baking

If a group has variants, we iterate through them to ensure each one gets baked.After baking, the group is restored to its default variant.

```
foreach (var group in groups.Values)
{
    // if there is only one variant of the lightmap group
    if (!group.hasVariants)
    {
        Lightmapping.Bake();
        SaveBakedLightmaps(group);
    }
    else
    {
        // we might have multiple variants
        foreach (var variant in group.variants)
        {
            group.SetVariant(variant);
            Lightmapping.Bake();
            SaveBakedLightmaps(group);
            group.SetDefault();
        }
    }

    currentIndex++;
}
```

FEEDBACK

Peer Feedback

> “It is really easy to use.”

> “Now we only need one tool for baking the light!!”

Ship

The tool has **already been shipped** internally for baking the lightmaps in the level.

Robustness

Up to now, 0 tickets related to bake errors by this tool.

My **Swift In A Week.**
Apple likes it.

ShaderGraph Comparison App

Apple Inc, May 2024

BACKGROUND & ABSTRACT

Adding nodes to Reality Composer Pro and document them with a macOS application.

TODO

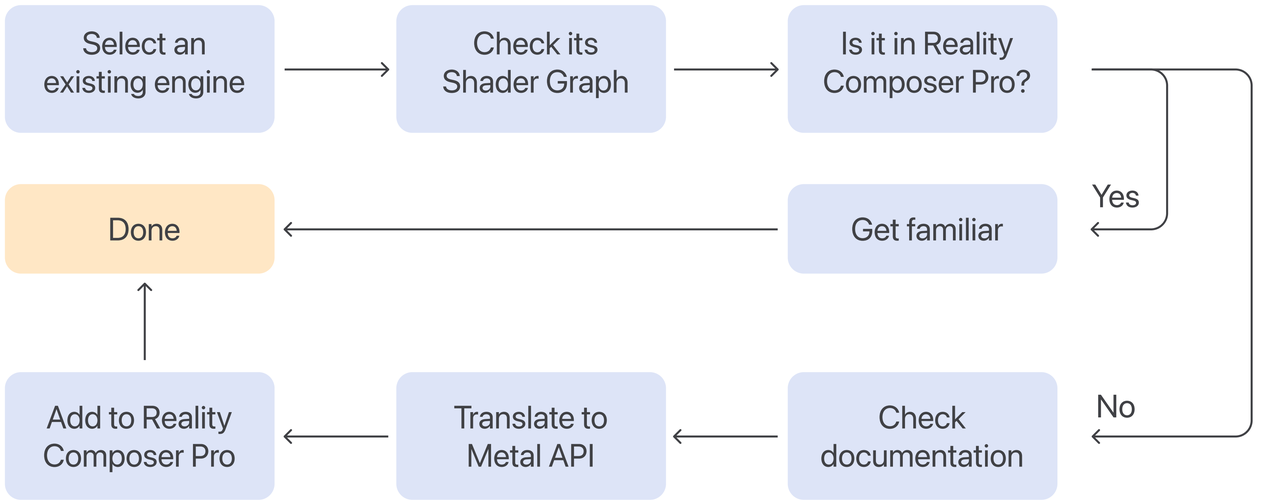
Apple Inc. is developing its own ShaderGraph in the Graphics Engine Reality Composer Pro, prepared for Apple Vision Pro development.

Multiple important features were missing when I joined Apple.

Complete **the nodes missing in the ShaderGraph** of Reality Composer Pro.

Problems

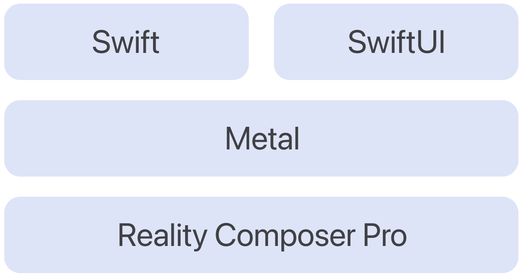
The following diagram shows the workflow.



Checking if a node exists in Reality Composer Pro is time-consuming — UI names often differ from codebase identifiers. The codebase is modular, requiring cross-referencing UI modules and their implementations. Adding new nodes involves similar effort.

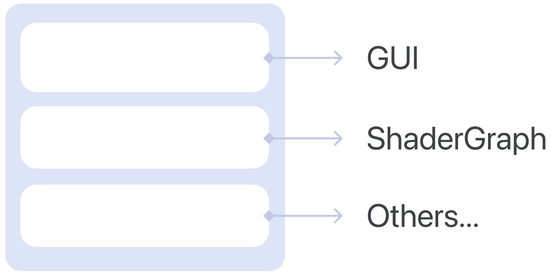
CHALLENGES

New to Apple



Apple's entire stack — language, UI framework, graphics API, and engine — is new to me. I'm starting from scratch.

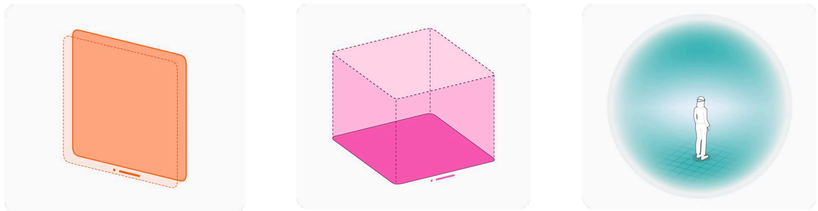
Modules



Reality Composer Pro's scale makes ShaderGraph changes systematic and complex, posing a steep learning curve for new team members.

Nodes are defined in a separate language, adding another layer of complexity to the workflow.

Vision Pro's Limitation



Tailored for Vision Pro, Reality Composer Pro enforces its own specs, limiting node compatibility. Its customized rendering pipeline adds complexity — vertex-related nodes may show unexpected behavior.

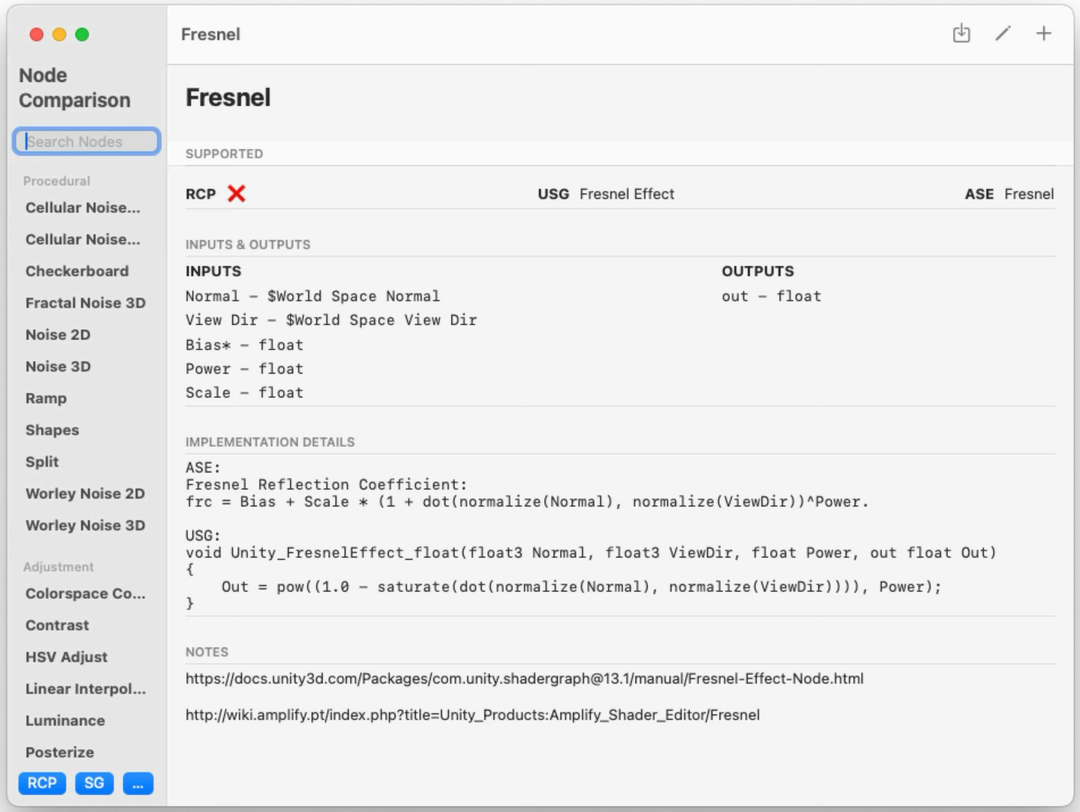
Comprehensive testing is required for each node — whether in 2D on Windows, 3D within Volumes, or interactively in Immersive Spaces.

SOLUTIONS

ShaderGraph Comparison App

Quick search

Filter to view currently supported nodes in Reality Composer Pro.



Manual addition and edits to the node library

Support in different race products and corresponding names

Inputs and outputs

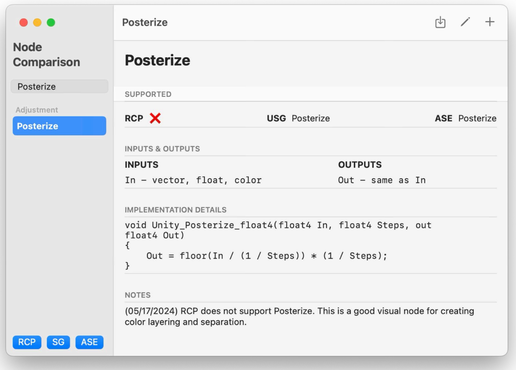
HLSL Implementation details; will be used to translate to Metal

Additional notes

A macOS Application developed based on SwiftUI

SOLUTIONS

New Workflow



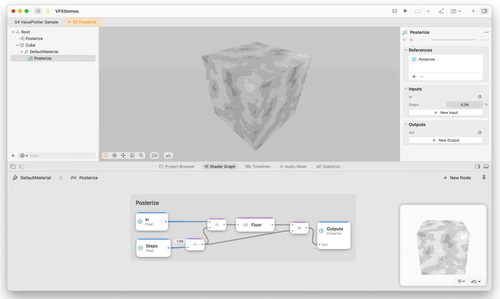
Check if it exists in Reality Composer Pro

Yes

Got familiar by reading the information in the app.

Done

No



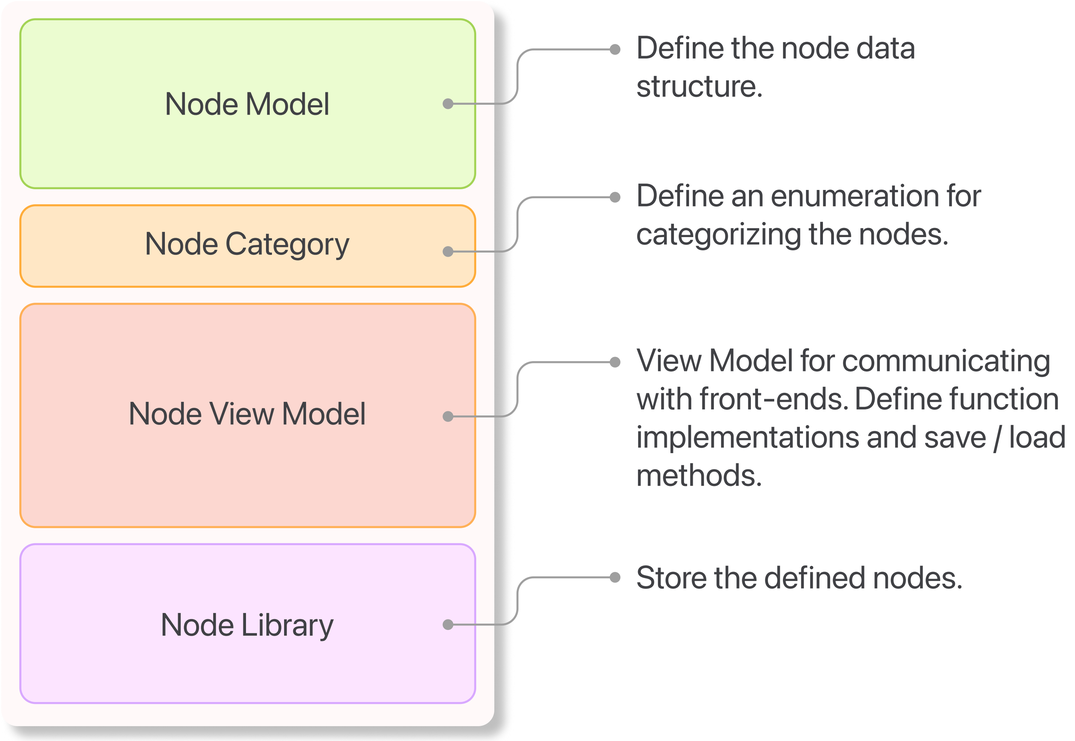
Implement it using NodeGraph or Metal with the help of **Implementation Details** section of the app.

Translate to internal syntax (using another command line tool I created with a single line)

Compile

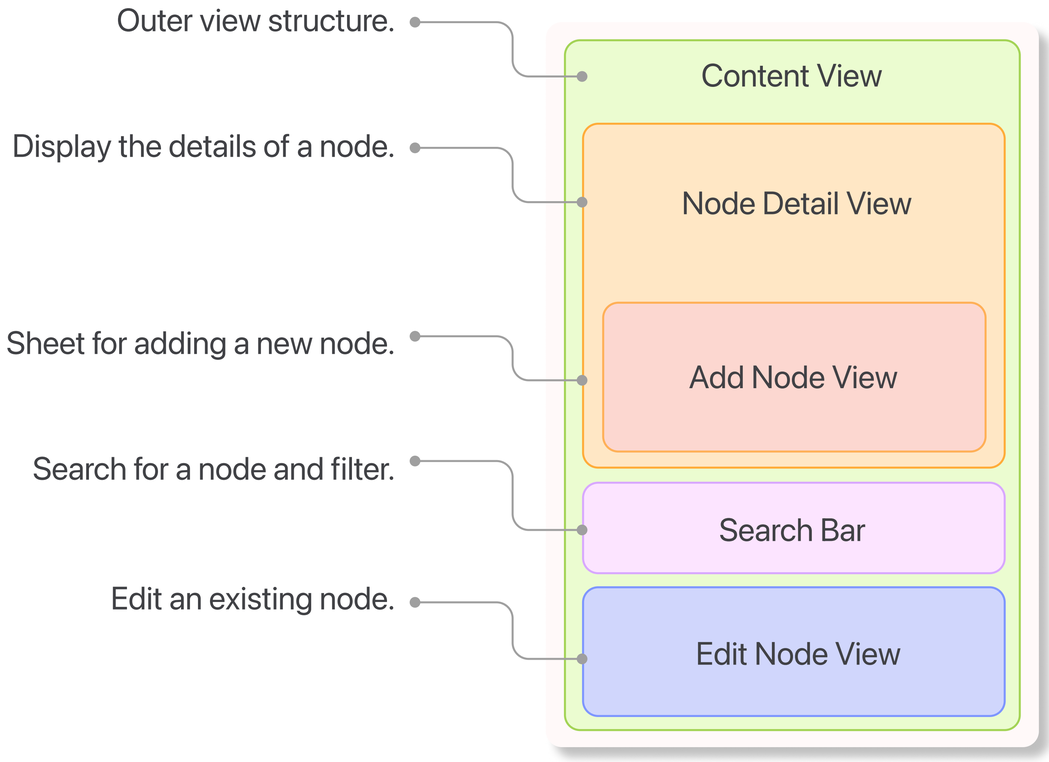
IMPLEMENTATIONS

ShaderGraphComparisonApp



Tool Backends

ShaderGraphComparisonApp



Tool Frontends

FEEDBACK

Managers Feedback

My Manager

"We are really delighted to see you get yourself familiar through making an app that is not only useful for you, but also for our team."

Org Manager

"This app should be incorporated to our workflow. Thanks for the contribution."

Ship

The tool has **already been shipped** internally in Apple for quick query of availability of a node.

You would also **love to use it.**

Otter Shaders

Beyond Beyond Touch at CMU ETC, Dec 2024

https://github.com/Lockbrains/OtterShaders

BACKGROUND & ABSTRACT

Code snippets that you can call directly in SwiftUI to create visual effects.

TODO

Beyond Beyond Touch is my second-year project at CMU Entertainment Technology Center — an otter-themed fitness app for iPhone. As a card-collection game, we use visual effects in SwiftUI to differentiate card quality.

Created a **shader effect library** designed for easy integration with SwiftUI.

Problems

Adding a SwiftUI Metal Shader Effect requires several steps.

① A Metal Shader

```
[[stitchable]] half4 noise(float2 position, half4 currentColor, float time) {
    float value = fract(sin(dot(position + time, float2(12.9898, 78.233))) * 43758.5453);
    return half4(value, value, value, 1) * currentColor.a;
}
```

② Use it in the View

```
struct ContentView: View {
    let startDate = Date()

    var body: some View {
        TimelineView(.animation) { context in
            Image(systemName: "figure.run.circle.fill")
                .font(.system(size: 300))
                .colorEffect(
                    ShaderLibrary.noise(.float(startDate.timeIntervalSinceNow))
                )
        }
    }
}
```

While the workflow is simple, exposing shader logic to the frontend and relying on manual selection makes it fragile and unsafe.

CHALLENGES

Metal Shading Language

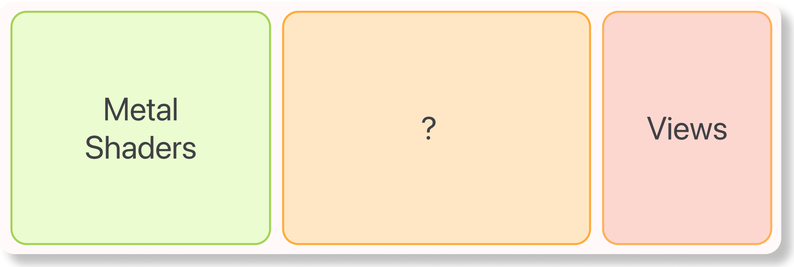
SwiftUI visual effects are powered by the Metal API, with shaders written in Metal Shading Language — requiring strict adherence to its rules.

Shader Effects

```
Image(...)
    .font(.system(size: 300))
    .colorEffect(...)
    .layerEffect(...)
    .distortEffect(...)
```

We need a variety of shader effects to deliver diverse card visuals. Each effect requires flexible parameter tuning and should be composable with others.

Encapsulation



Shaders are written in MSL (Metal Shading Language), while views are built with Swift. To separate concerns, each layer should be encapsulated — frontend and artist developers should only interact with exposed parameters.

SOLUTIONS

Encapsulations

- **Shaders**
  - ColorEffects.metal
  - DistortEffects.metal
  - LayerEffects.metal
  - Common.metal

Shaders written in Metal Shading Language. Frontend developers won't have access to these implementations.

- **ShaderToView**
  - ColorEffectsGenerator.swift
  - DistortEffectsGenerator.swift
  - LayerEffectsGenerator.swift

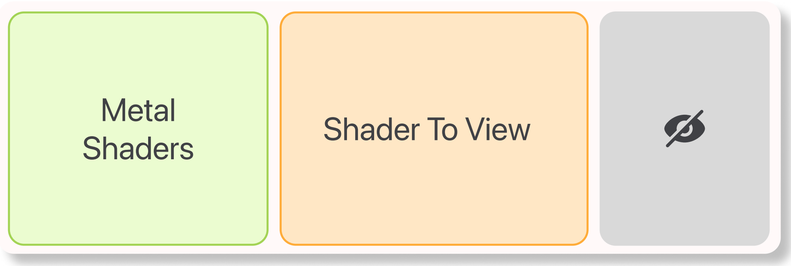
API for frontend developers to call. They will have access to these generator codes — they will generate a modifier for views to use.

- **Views**
  - ContentView.swift
  - SampleShaders.swift

Views where frontend developers work.

- **Documentations**
  - ShaderDocumentation.md

→ **Documentation** for how to use this tool.



In backend developer's perspective

In frontend developer's perspective

SOLUTIONS

Tools I Prepared

Shaders

ColorEffects.metal

```
[[ stitchable ]] half4 starEffect (float2 position,
                                   half4 currentColor,
                                   float2 size,
                                   float time,
                                   float layers,
                                   float intensity,
                                   half4 starColor) {
    float2 uv = position / size;
    float t = time * 0.02;

    uv *= rotateMatrix(t);

    half3 col = half3(0.0);

    for(float i = 0.0; i < 1.0; i+= 1.0 / layers) {
        float depth = fract(i + t);
        float scale = mix(20.0, 0.5, depth);
        float fade = depth * smoothstep(1.0, 0.9, depth);
        col += starLayer(uv * scale + i * 454.23, time, starColor.rgb) * fade;
    }

    half4 finalColor = currentColor;
    finalColor += intensity * half4(col, 0);
    return finalColor * currentColor.a;
}
```

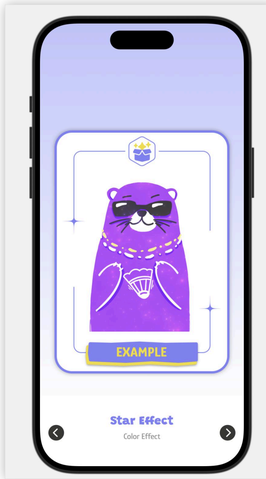
Example Shader written in MSL

ShaderToView

ColorEffectsGenerator.swift

```
// MARK: Star Effect
@available(iOS 17.0, *)
func starEffect(date: Date,
                speed: Double,
                x: CGFloat,
                y: CGFloat,
                layers: Float,
                intensity: Float,
                starColor: Color) -> Shader {
    return ShaderLibrary.starEffect(
        .float2(x, y),
        .float(abs(date.timeIntervalSinceNow) * speed),
        .float(layers),
        .float(intensity),
        .color(starColor)
    )
}
```

Applied in views



`ShaderLibrary` modifier generator

DOCUMENTATION

Follow **How To Use** to get started with the tool. Apple official documentation links are also provided.

- i. `colorEffect` : if the effect ONLY changes color, find the shader in Color Effect Section
- ii. `layerEffect` : if the effect requires information from the background (neighboring pixels, background pixels, etc), find the shader in Layer Effect Section.
- iii. `distortEffect` : if the effect requires the morph of the shape / vertex manipulations, find the shader in the Distort Effect Section.

Read Apple Developer's documentation for more details:
https://developer.apple.com/documentation/swiftui/shader

How To Use

To plug the code into your project, you may get the code from the `Shaders` folder (and feel free to edit anything in these shaders written in MSL to meet your own demand!).

There are 3 files that you should take a look:

1. `Shaders\ColorEffects.metal` : All the color effects live here.
2. `Shaders\DistortEffects.metal` : All the distort effects live here
3. `Shaders\LayerEffects.metal` : All the layer effects live here.

If you don't feel like to see the implementation details of these shaders, you may also use the generators for generating a SwiftUI Shader. These generators live in the `ShaderToView` folder. Similar to the above, just find the effect in its category.

Generators

To activate a visual effect on a swift view, you will use the `.colorEffect` or `.layerEffect` or `.distortEffect` modifier, for example:

```
Image("example")
    .resizable()
    .colorEffect(
        // Add the shader here
    )
)
```

You will need to provide a shader and some additional fields for Layer and Distort effects.

For color effects, for example, if you'd like to plug in a star effect, you will simply need to do

```
Image("example")
    .resizable()
    .colorEffect(
        starEffect(
            date: date,
            speed: 1.0,
            x: 200, // the width of the frame
            y: 200, // the height of the frame
            layers: 8,
            intensity: 10,
            starColor: Color.red
        )
    )
```

or, if you feel this codestyle makes your code wordy, you may set up file for saving these shaders as constants, just as what we did in the `Shaders/SampleShaders.swift` .

For layer effects, you will additionally need a `maxSampleOffset` . This is telling the shader how many neighboring pixels' infomation it needs to pack. Please refer to the official documentation for more details:
https://developer.apple.com/documentation/swiftui/view/layereffect(_:maxsampleoffset:isenabled:).

Examples

Example Shaders created using this tool.

FEEDBACK

Team Feedback

Artists

"Happy to see that we can have visual effects on our Swift game."

Frontend

"I'm glad that I don't have to understand how shader works."

Summary

Originally built during *Otter Agent*, a social Apple Watch game where I was a frontend developer, this tool was later extracted into a standalone repository to support future reuse of our code and visual effects in the **Otter Series.**

Now it comes **our artists' showtime.**

Unity Shaders

Building Virtual Worlds at CMU ETC, Sep 2023

https://www.taotamago.com/dissolveeffects

BACKGROUND & ABSTRACT

Shaders with customized editors that artists can easily understand.

TODO

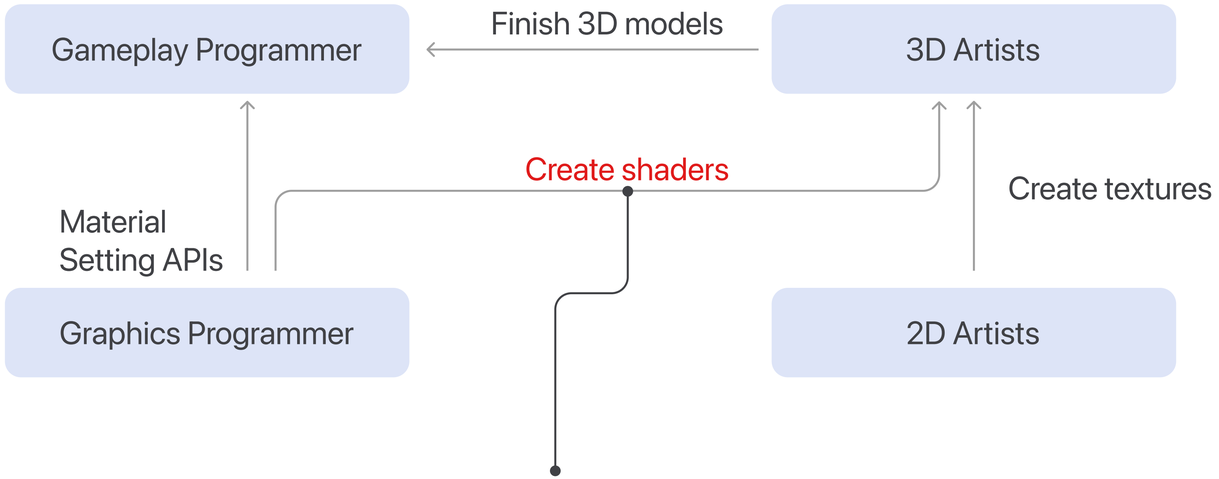
Building Virtual Worlds (BVW) is a fast-paced, team-based course at Carnegie Mellon's Entertainment Technology Center. In each cycle, teams of programmers, artists, and sound designers collaborate to create a playable game or experience within 1–2 weeks.

Create **shaders for artists** to smoothly use and adjust in the scenes.

Problems

Each BVW team has two programmers, two artists and a sound designer.

We have a clear workflow and division of labor.

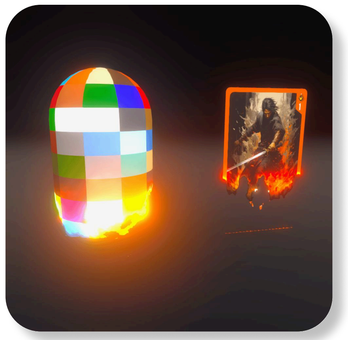


Pain points are here.

Graphics programmers and 3D artists often lack a shared language, making parameter communication frustrating and inefficient.

CHALLENGES

Shaders



As the graphics programmer on my team, I was responsible for writing shaders that aligned with our project scope — even though I was still new to shader programming at the time.

Work with Programmers

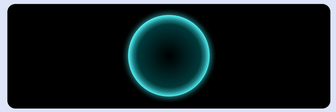
MaterialManager.cs

```
public void SetMaterial();
public void TransitionTo();
...
```

I typically exposed APIs to our gameplay programmers through a `MaterialManager` class. It was important to provide clear documentation on available functions, usage patterns, and potential pitfalls.

Communicate with Artists

M_Sphere



Rim Intensity
Rim Out Color
Rim Inner Color
Intensity
Scale 3.5

What does “RimOutColor” mean?

While I can create highly flexible shaders, artists often struggle to understand how to use them or why certain parameters are needed. As a result, the shaders and materials may feel overly complex and intimidating to use.

SOLUTIONS

Encapsulations

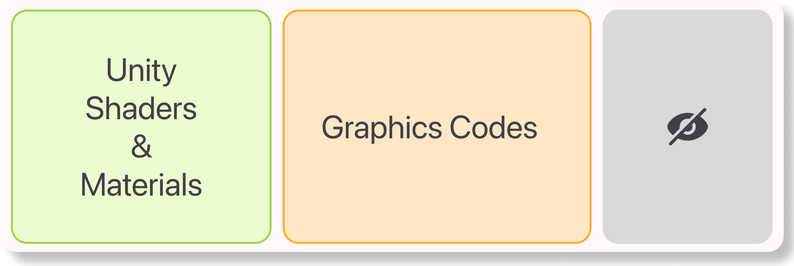
- **Scripts**
  - C# MaterialManager.cs
  - C# MaterialRenderer.cs
  - C# GameManager.cs
  - C# LevelManager.cs
  - ...

Rendering related managers. Gameplay Programmer will only read the interfaces in these files to understand what to call.

Gameplay related managers. Graphics Programmer will not have to understand anything about gameplay logic.

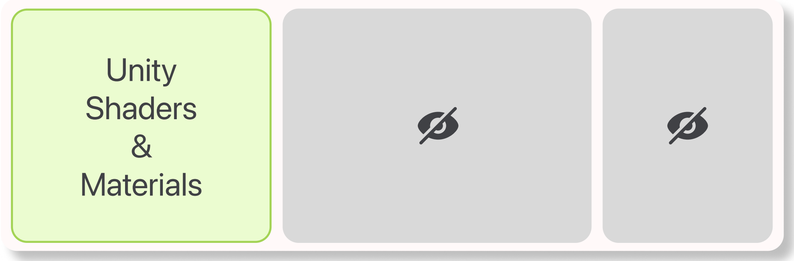
- **Editor**
  - C# MaterialRendererEditor.cs
  - C# TransparentEditor.cs
  - ...

Editor Scripts to hide or organize component variables, preventing artists from misclicking or inputting values in the wrong fields.



In Graphics Programmer's perspective

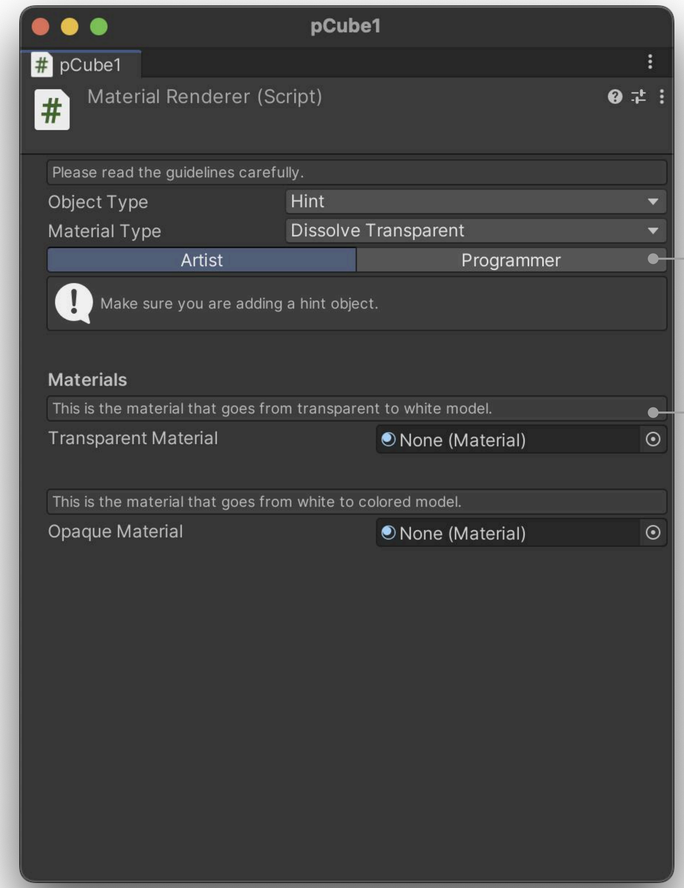
In Gameplay Programmer's perspective



In Artists' perspective

SOLUTIONS

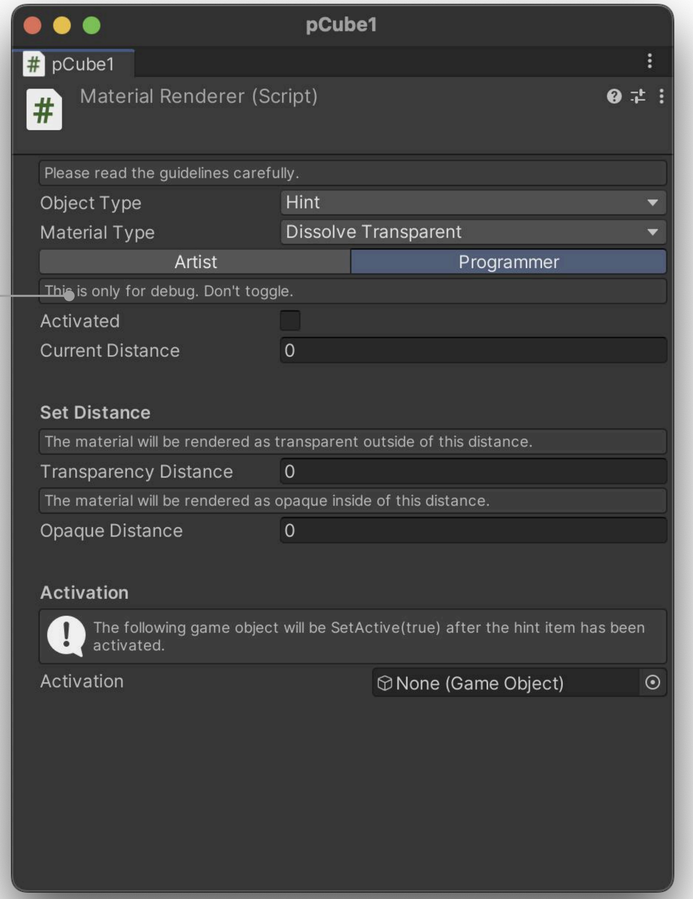
Customized Editor for `MaterialRenderer`



Tabs to distinguish variable exposure to artists and programmers .

Guidelines for artists to understand which materials to select.

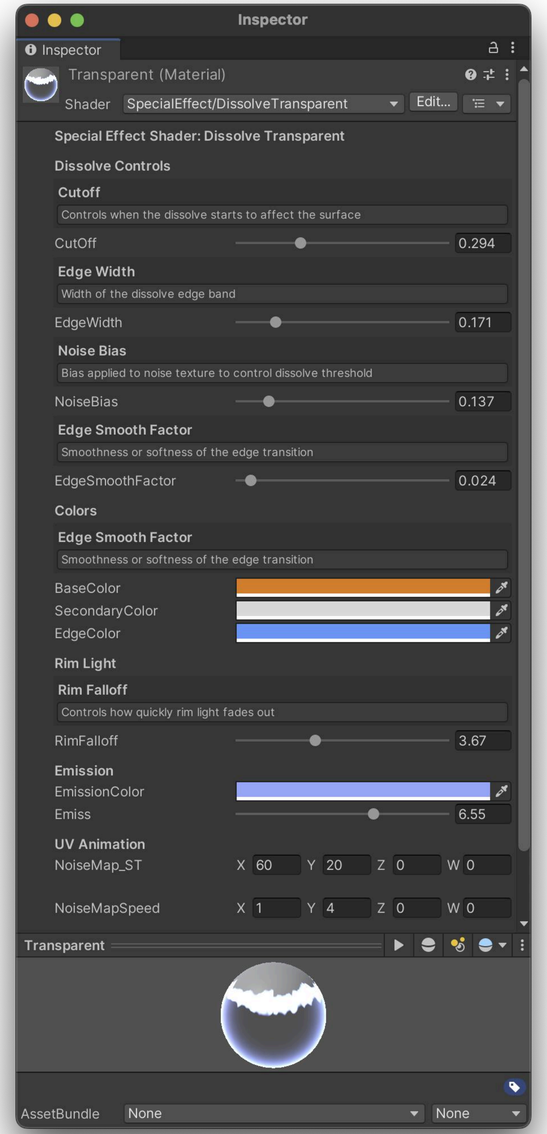
Debug options for the ease of development.



SOLUTIONS

Customized Editor for `DissolveTransparent`

Parameters are labeled with how it will affect the visuals.



FEEDBACK

Team Feedback

Artists

"The procedure of setting up materials are super straightforward."

Gameplay Programmer

"Smooth collaboration with the methods."

Summary

Effective communication between artists and programmers is essential to team morale. The examples shown here are just a glimpse of Building Virtual Worlds, but the collaborative approach stayed consistent. Custom editor tools helped bridge the technical gap, making the engine more accessible and intuitive for artists.

Pittsburgh, PA
+1 412 328 9641
ltt@taotamago.com
www.taotamago.com
www.linkedin.com/in/lingheng-tao