HighlightActor Plugin Documentation

This document provides a comprehensive guide to the **HighlightActor** Unreal Engine plugin. It covers installation, setup, API reference, advanced usage, and extension points.

Table of Contents

Overview

The HighlightActor plugin provides a modular system for highlighting actors in your scene based on different input modalities (mouse cursor, crosshair, touch, VR pointer, or custom). It uses a strategy

pattern to encapsulate tracing logic and a single ActorComponent (`UHighlightInteraction`) to coordinate highlighting.

Key features:

- Modular tracing: Swap trace methods by choosing an enum (`EHighlightDetectionMode`) in the editor.

- UInterface-based highlight callbacks: Actors implement `IHighlightInterface` to respond to highlight/unhighlight events.

- Blueprintable strategies: All trace strategies derive from `UHighlightTraceStrategy`, allowing Blueprint overrides if desired.

- Custom mode: Hook into `PerformCustomTrace` in Blueprint for fully custom logic.

Installation

1. Clone or copy the `HighlightActor` plugin folder into your project's `Plugins/` directory.

2. Open your project in Unreal Editor.

3. Enable the `HighlightActor` plugin under **Edit → Plugins**.

4. Restart the editor when prompted.

Setup

Collision Channel

To ensure traces only hit highlightable actors:

1. **Edit → Project Settings → Collision**.

2. Under **Object Channels**, click **+** to add a new channel:

- Name: `Highlightable`

- Default Response: `Overlap` or `Block`

- Channel index assigned automatically.

3. In your actor's **Collision** settings, set the object type to `Highlightable`, or handle channel filtering in code.

Enabling Plugin

Ensure your module's `*.Build.cs` includes:

```csharp
PublicDependencyModuleNames.AddRange(new[] { "HighlightActor", /* ... */ });
```

Core Concepts

HighlightInterface

File: `HighlightInterface.h`

Location: `Public/Interaction/HighlightInterface.h`

```cpp
UINTERFACE(Blueprintable)
class UHighlightInterface : public UInterface { GENERATED_BODY() };
class IHighlightInterface {
GENERATED_BODY()
public:
UFUNCTION(BlueprintNativeEvent)
void HighlightActor();
UFUNCTION(BlueprintNativeEvent)
void UnHighlightActor();
};
```

- Implement this interface on any actor you want to be highlightable.
- Override `HighlightActor_Implementation` and `UnHighlightActor_Implementation` in C++ or Blueprint to change mesh outline, material, etc.

HighlightTraceStrategy

File: `HighlightTraceStrategy.h`

Location: `Public/Interaction/HighlightTraceStrategy.h`

```cpp
UCLASS(Abstract, Blueprintable)
class UHighlightTraceStrategy : public UObject {
GENERATED_BODY()
public:
UFUNCTION(BlueprintNativeEvent)
bool PerformTrace(APlayerController* PC, FHitResult& OutHit);
};
```

- Base class for all trace strategies. Implements `PerformTrace` as a BlueprintNativeEvent.
- Derive concrete strategies (C++ or Blueprint) and implement `PerformTrace_Implementation`.

HighlightInteraction Component

Files:

- `HighlightInteraction.h`
- `HighlightInteraction.cpp`

Location: `Private/Interaction/HighlightInteraction.*`

This component:

1. Instantiates the correct `UHighlightTraceStrategy` based on `HighlightMode`.
2. Ticks every frame and calls `PerformTrace`.

3. Compares hit actor to the previously highlighted one.
4. Calls `IHighlightInterface::Execute_HighlightActor` or `UnHighlightActor` as needed.

Configurable properties:
- **HighlightMode** (`EHighlightDetectionMode`): choose Mouse, Crosshair, Touch, VR, or Custom.
- **PerformCustomTrace**: BlueprintImplementableEvent for fully custom trace logic.

EHighlightDetectionMode Enum
File: `HighlightTypes.h`
Location: `Public/Interaction/HighlightTypes.h`
```cpp
UENUM(BlueprintType)
enum class EHighlightDetectionMode : uint8 {
MouseUnderCursor,
CrosshairCenter,
TouchInput,
VRPointer,
Custom
};
```

Using the Component

C++ Integration
1. Add the component to your PlayerController or Pawn:
```cpp
HighlightInteraction = CreateDefaultSubobject(TEXT("HighlightInteraction"));
```
2. Set `HighlightMode` in C++ constructor or `BeginPlay`.

Blueprint Integration
1. Open your PlayerController or Character Blueprint.
2. Click **Add Component** → **HighlightInteraction**.
3. Select the component, choose `HighlightMode`, and optionally implement `PerformCustomTrace` in the Events graph.

Available Strategies

CursorTraceStrategy
- File: `CursorTraceStrategy.*`

- Logic: Deprojects mouse cursor to world, performs line trace along direction.

CrosshairTraceStrategy
- File: `CrosshairTraceStrategy.*`
- Logic: Finds viewport center, deprojects screen position, line trace.

TouchTraceStrategy
- File: `TouchTraceStrategy.*`
- Logic: Reads first touch index, deprojects screen pos when pressed.

VRControllerTraceStrategy
- File: `VRControllerTraceStrategy.*`
- Logic: Finds `UMotionControllerComponent` on pawn, traces forward from controller.

Custom Tracing

- Custom Mode: Set `HighlightMode = Custom`.
- Implement the `PerformCustomTrace` event in Blueprint or override in C++:

```bp
Event PerformCustomTrace(HitResult OutHit) → return true/false
```

Extending the Plugin

Creating New Strategies
1. Subclass `UHighlightTraceStrategy`.
2. Implement `PerformTrace_Implementation` with your logic.
3. Include your new strategy in `InitializeStrategy` or expose it via Blueprints.
4. Add a new enum entry to `EHighlightDetectionMode` if desired.

Modifying Highlight Logic
- The core logic lives in `PerformHighlight()` within `UHighlightInteraction.cpp`.
- You can override highlight transitions, change timing, or add smoothing between highlights.

Troubleshooting

- No highlight: Verify your actors implement `IHighlightInterface` and collision is set to `Highlightable` channel.
- Strategy not created: Ensure `HighlightMode` matches the case in `InitializeStrategy()`.

- BP not firing: Ensure `PerformCustomTrace` is marked `BlueprintImplementableEvent` and your Blueprint child implements it.

FAQ

Q: Can I use multiple `HighlightInteraction` components on different actors?

A: Yes—each component independently traces and highlights based on its owner's context.

Q: How costly are these traces?

A: Traces run every tick. Use efficient collision channels and disable the component when not needed.

Q: Can I smooth transitions (fade outlines)?

A: Yes—implement interpolation in your `HighlightActor_Implementation` on the actor side.

For further questions or contributions, contact the Heathrow (Derman) engineering team.