Coherent GT Unity3D Guide

2.3.0

Contents

Chapter 1

Introduction

Coherent GT is a modern user interface middleware solution that allows you to integrate HTML pages built with CSS and JavaScript in your game.

Each HTML page is represented by a `Coherent.UIGT.View` object. The `Coherent.UIGT.View` is wrapped by a *Coherent GT View* component that allows you to add views to to your scenes easily.

You can perform operations on the page via the *View*, such as resizing, navigating to a different URL, sending input events, executing custom JavaScript code and so on.

Views can be created both by adding a *Coherent GT View* component to a *game object* in the Unity scene or attaching them through code.

You can customize the *View* via different initialization parameters, such as width, height, initial URL, etc.

View notifications, for example when the URL is changed, are received via the *ViewListener* class. The `View↩Listener` is the class that receives such notifications for a specific view - when the URL was changed, the page you're trying to open requires authentication details, etc.

Chapter 2

Getting Started

2.1 In this guide

- How to add Coherent GT to your Unity3D Project
- What can you find in the Coherent GT package
- Add a Coherent GT view to your scene
- Distributing games with Coherent GT

2.2 What do you need before we start

Before starting development with Coherent GT, make sure you have the following:

1. Unity3D 5.4 or higher
2. Coherent GT unitypackage
3. License key, if you're using Coherent GT Pro version.

2.3 Add CoherentGT in your Unity3D project

First, open your project in Unity3D Editor, then:

1. From the menu bar choose `Assets -> Import package -> Custom package`
2. Navigate in the file chooser dialog to the folder with your Coherent GT package.
3. Click `Open` and a Unity3D dialog with all files of the package files listed should open.
4. Click `Import`.

If you're using a **Coherent GT Pro** package, there's an additional step - you should navigate to `Assets\Standard Assets\Detail` and copy your license key in the LicenseGT.cs file.

2.4 What's there in the Coherent GT package

Here is the structure of the Coherent GT package explained briefly, assuming `Assets` is the root folder:

- *CoherentUIGT* - this folder contains useful resources.
  - *Documentation* - folder containing the documentation.
  - *Editor* - folder containing scripts used by the Unity3D Editor, the binaries for the Coherent Editor and for the debugger.
  - *Resources* - contains the shader files used to render `CoherentUIGTView`s in Unity3D.
  - *Samples* - folder containing samples with Coherent GT.
  - *CoherentEditor.zip* - archived standalone version of Coherent Editor.
  - *Debugger.zip* - archived Coherent GT debugger binaries.
  - *Inspector.zip* - archived files needed for the Coherent GT debugger.
  - *JavaScriptBinding.zip* - contains the JavaScript engine used by Coherent GT for triggering events in JavaScript.
- *Plugins* - the Coherent GT library files and their dependencies (split into two sub folders containing the files needed for x86 and x86_64 builds respectively).
- *Standard Assets* - contains the Coherent GT components that you'll need for UI development.
  - `CoherentUIGTView.cs` - component used for displaying web pages.
  - `CoherentUIGTSystem.cs`- component used to initialize and manage the UI.
  - `CoherentUIGTLiveGameView.cs` - component used for easy creation of live views.
  - `CoherentUIGTLocalization.cs` - component used for localization of web pages.
  - *Detail* - scripts used by the above components, you do not need to add them directly to your scene.
- *WebPlayerTemplates/uiresources* - contains HTML, CSS and JavaScript UI resources for the samples.
- *WebPlayerTemplates/inspector* - contains the files needed for the Coherent GT debugger.
- *WebPlayerTemplates/editor* - contains the files needed for the Coherent Editor.
- *WebPlayerTemplates/DX11Shaders* - contains the shaders that Coherent GT uses when rendering the web page with DirectX11 and DirectX9.
- *WebPlayerTemplates/OpenGLShaders* - contains the shaders that Coherent GT uses when rendering the web page with OpenGL.

2.5 Add CoherentGTView to your scene

You can add views both to Camera objects and to regular game objects in world. The first type, are suitable for HUDs and menus, while the latter can be used for interaction between the player and the game.

Let's have a simple HTML page, HelloHTML.html

Place it in your *UI Resources* folder. By default, this is the ∗Assets∗ folder, unless you change it from `Coherent GT -> Setup -> Select UI Resources folder`.

Then, choose a GameObject in the scene, where the page should be placed. Let's say we want to display it instead of HUD - in this case click the **MainCamera** object in the scene and then `Add Component`, afterwards choose `Coherent GT View`.

Go to the `Page` property of the view and set the relative URL of the HelloHTML page here, for example: **coui↩ ://uiresources/HelloHTML.html**. Click `Play` and you'll see the result.

2.6 Distributing games with Coherent GT

CoherentUIGTDevelopment.dll is the library where the debugger is implemented. It is copied only when you build with *Development Build* option. If the library is missing, the debugging functionality is disabled. You can also copy it manually, to enable the debugger. Just make sure that you specified a valid port in the system settings.

Chapter 3

Transitioning from Coherent UI 2.x

Coherent GT makes transitioning the whole UI or just some Views from Coherent UI 2.x very easy. The API provided is very similar, with classes residing in the Coherent.UIGT and Coherent.UI namespaces respectively. You should only change the components used.

Note

If you've been using multiple contexts with CoherentUI 2.x, you should switch to using a single one in GT since multiple UISystems are not supported

When transitioning, only the initialization and rendering parts for a View require changes. The Binding and Input APIs are 100% compatible, so no changes there should be done.

Content created for Coherent UI 2.x should run without changes on *Coherent GT*. Note however that the 'coherent.js' file may contain differences depending on your version.

Always use the correct file for the runtime that is currently rendering your View.

Chapter 4

Coherent GT Components

4.1 Creating a Coherent GT view

Coherent GT View is added to the scene just like a regular Component in Unity3D:

1. In the Unity3D Editor, via attaching the Coherent GT View component to an existing Game Object.
2. Programatically, by calling `AddComponent` on the desired Game Object.

All the views are managed by a `Coherent.UIGT.UISystem` (wrapped in a `Coherent GT System`). By default, if there's no user-created Coherent GT System present, at the time when a Coherent GT View is created, a default Coherent GT System is added automatically to the scene.

If you want your *Views* to use a non-default `UISystem` you should, simply add a `Coherent GT System` component in your scene with the desired settings.

Note

You can have only one `Coherent GT System`. If you try to create more than one, the creation of every system but the first, will fail.

4.2 Coherent GT View properties

General section:

- **URL** - Indicates the URL that will be initially loaded.
- **Width** - The width of the Coherent GT View.
- **Height** - The height of the Coherent GT View.
- **AudioSource** - The audio source for this View. Audio from the page is send to this AudioSource. If no audio source is selected, a default one will be. created and attached to the same game object.
- **Auto Refresh** - When this is ticked, the view will monitor for changes in the loaded page and will reload it automatically upon change.
- **IsAudio3D** - This is visible only in Unity 4.x. It is indicating whether the sound from the view should be 3D or 2D. In Unity 5.x this is controlled by the Audio Source component. By default the sound is 2D.

Rendering section:

- **Post Effect** - Defines whether the View is drawn. before or after the post-effects. Available only for Views attached to cameras.
- **Flip Y** - Flips the drawn image vertically.
- **Is transparent** - Defines if the View supports transparency.
- **Match camera size** - The View will be automatically resized to always match the size of the camera.
- **Click-through threshold** - A value in the range [0-1] inclusive that determines whether a pixel is transparent. A pixel is transparent if its alpha value is below or equal to the threshold.

Advanced rendering section:

- **Always on top** - Indicates whether this value is z-buffer independent. If it is set to true, the view is rendered on top of everything.
- **Texture format** - This refers to the color space conversion mode of the RenderTexture used to display the view. You can read more about it in `Unity3D documentation`

Input section:

- **Lockable focus** - When enabled, the View takes the input focus when clicked and releases it when you click outside the View. See Click to focus Views

Scripting section:

- **Pre-load script** - The script will be executed before any other code in the UI View.
- **Auto UI messages** - When enabled, any event triggered in *JavaScript* on the `engine` object is forwarded to the game object containing the View.
- **Enable [CoherentMethodAttribute]** - Enables the usage of the [Coherent Method attribute]. (Coherent↩ MethodAttribute)

4.3 Coherent GT System properties

General section:

- **Debugger port** - The port where the system will listen for the debugger. Invalid values disable the debugger.
- **Disk Cache size** - The maximum size of the disk cache.
- **Run Asynchronously** - Enables the asynchronous mode. See Asynchronous mode
- **Enable Localization** - Enables localization for all views. Keep that unticked when not using localization for better performance. See Localization

4.4 Coherent Live Game View properties

General section:

- **Name** - The name of the Live View - must be the same as in the HTML page.
- **Width** - Width of the Live View.
- **Height** - Height of the Live View.

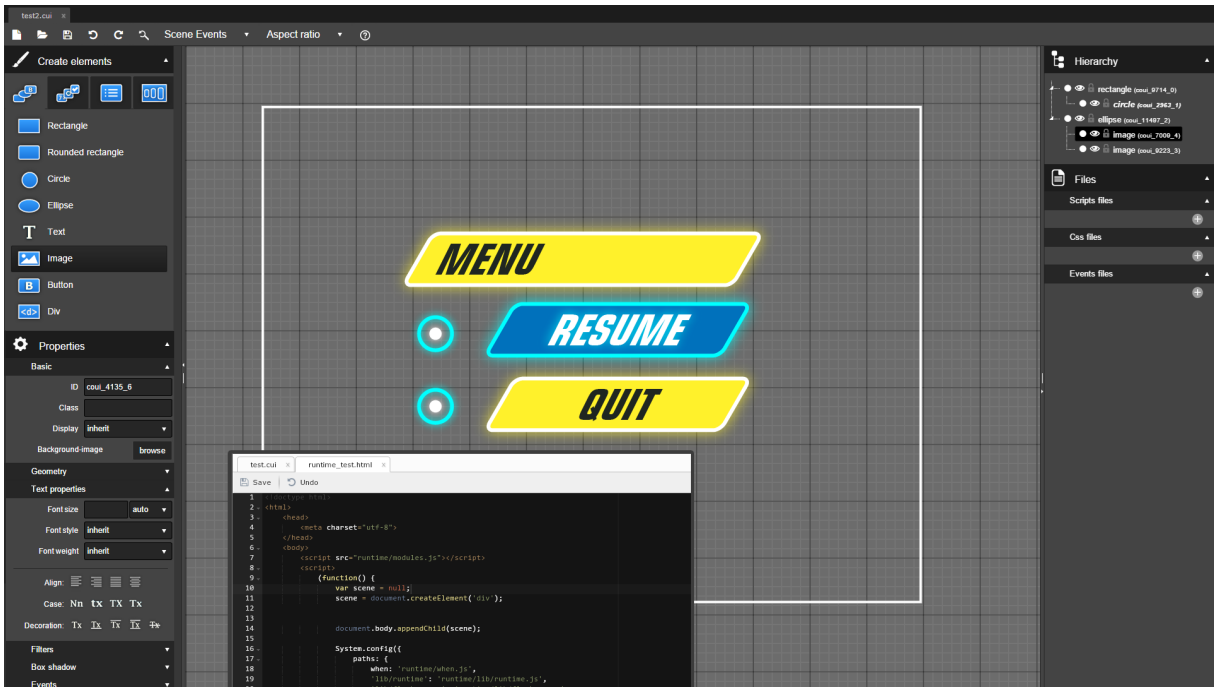
- **Target View** - The Coherent GT View displaying the HTML page with this Live View name.
- **Source Camera** - The camera used as source for this live view. By default the component will use the camera on the same gameobject.
- **Source Texture** - A RenderTexture used as source.

You can supply either Camera or a Texture as source.

Chapter 5

User Interface creation and editing

The Coherent Editor is WYSIWYG HTML visual editor specifically designed for Game UI. Using its visual tools and code editor you can quickly create amazing looking UI for your game.



Key features:

- WYSIWYG visual editor that anybody in the team can use to quickly create amazing looking Game UI
- Unmatched feature set through the power of HTML, CSS and JavaScript
- Easy data binding that enables quick communication between the game and the UI
- Powerful animations options through CSS animations and JavaScript libraries
- Performance optimized output code

The Coherent Editor's documentation is available online - http://coherent-labs.com/editor/documentation

If you like you can also check its online demo - http://coherent-labs.com/editor/

Of course for content creation apart from the Coherent Editor you can also use standard web development tools such as Adobe Edge, Dreamweaver, Google Web Designer, Sencha Animate, etc.

To get started using the **Coherent Editor**, launch it from the Coherent GT menu or you can use the standalone version from the *CoherentEditor.zip* package in the *CoherentUIGT* folder. Just extract it to a location of your choice. Then start the CoherentEditor executable and follow its `Quick Start` guide.

Chapter 6

UI Debugging and Live Editing

Coherent GT supports the whole WebKit Inspector protocol and allows for live editing and debugging of the UI even when the client application is running. This increases the productivity of UI programmers and designers immensely as can directly work on the live interface without stopping the game, recompiling or restarting.

6.1 Debugging in Unity Editor

You can launch the Debugger from Unity Editor `Coherent GT -> Launch Debugger`, then navigate to `http://127.0.0.1:19999` (or another port you've selected in the Coherent GT System component)

6.2 Debugging remotely

1. In the "Coherent GT" package you'll find the *debugger.zip* archive. Extract it to a location of your choosing.
2. Start the application for the current platform you're on.
3. Run the game on the remote machine.
4. Navigate to `http://w.x.y.z:19999` (or another port you've selected in the Coherent GT System component) where w.x.y.z is the IP of the remote machine.

6.3 Debugging on built application

1. Build your application in `Development Mode`.
2. Start your application.
3. Debug through Unity Editor or remotely.

6.4 Debugger in detail

It is advisable to use the debugging functionality only during development. It is automatically enabled when building in `Development Mode` and in the editor. Leaving it active on a shipped product leaves the UI accessible to users. This might be a desired feature as it aids UI modding but should be something to keep in mind when deploying a final product. The debugger is implemented in the CoherentUIGTDevelopment library.

The Debugger uses the resources located in the `WebPlayerTemplates/inspector` folder for its user interface.

The Debugger works over the network, so you can attach it either to you local machine or to a remote one. The debugging port is selected in the `Coherent GT System` component. If you pass -1, debugging will be disabled.

Debugging is done live, so start your application and have Coherent GT running.

You'll be presented with a list of Coherent GT Views that can be explored, debugged and edited.

6.5 Using the debugger

The debugger allows live viewing/editing of CSS styles, JavaScript debugging (setting breakpoints, watches, editing variables), resource load inspection, profiling.

The tool uses the WebKit Inspector UI, so for more information please refer to: https://developer.apple.com/library/safari/documentation/AppleApplications/Conceptual/Safari_Developer_Guide/Introduction/Introduction.html

6.6 Changing elements live

With the Inspector you can manually change all aspects of the elements within the page as well as adding/removing whole sections of it. This makes prototyping and iterating on the UI very easy. Go to the "Elements" tab. The page with all its current state will be shown. You can hover elements and they will be highlighted in the View in the application. All properties of the elements are editable and changes will be immediately mirrored in the live application.

6.7 Debugging JavaScript

A complete JavaScript debugger is included in the Inspector. To use it go to the "Sources" tab, click "Enable Debugging". On the left side a list of the loaded JS sources is given. Users can add breakpoints, watches, step through expressions and catch exceptions.

6.8 Checking page performance

It is very important to keep track of the computational resources used by the UI each frame. The time it takes for the interface to update itself (usually through JavaScript triggered by the application), re-layout the content and re-paint it are particularly important. The Inspector provides a Timeline that is fully integrated with Coherent GT. Click on the Timeline tab. Now you can record some frames via the "Record" button (a dot in the low-left corner). Event happening in the UI will be registered along with their duration. Important events include:

- Event - outside events like mouse over, clicks, keyboard events etc.
- Timer fired - when a timer in JS fires
- Request animation frame fired - what a requestAnimationFrame handler is called
- Recalculate style - a CSS style recalculation caused by some event or JavaScript code
- Time Start/End - events triggered in native code via the Binding API
- Paint - parts of the View that get re-painted. Note that only the CPU time is recorded, not the actual time it took the GPU to perform the draw actions.

The Timeline tends to slow the application a bit, especially if many events happen. Numbers thus reported in it are usually somewhat larger than when it is not enabled.

The Timeline is invaluable when profiling the performance in the UI and makes spotting eventual bottlenecks very easy.

6.9 Detailed JavaScript profiling

If more detailed JavaScript profiling is required, use the "Profiles" tab and start a JavaScript CPU profile. It will time all JavsScipt code executed and give timing statistics on all functions.

6.10 Checking what gets re-drawn

Coherent GT re-draws only parts of the View that have changed between frames. You can check visually check what gets re-drawn by clicking the "Settings" button (the small *gear* icon in the low-left corner) and checking "Show paint rectanges". It is very important to keep re-drawn regions to the bare minimum. Redundant changes in the D↩ OM like setting CSS styles with tiny differences can cause no visual change but still trigger a re-paint. Users should regularly check the behaviour of the page and make sure that no redundant and avoidable re-draws are triggered.

6.11 Checking the UI layers

Some elements in the page will get their own backing layers. A complete guide to the working of UI layers is available in the Performance guide. You can see what layers are currently on-screen and their re-paint counters and sizes clicking the "Settings" button (the small *gear* icon in the low-left corner) and checking "Show composting layers".

6.12 Optimizing performance

For a detailed guide on performance best practices, please refer to the "Performance Best Practices" section of the documentation.

Chapter 7

Bindings

Bindings in *Coherent GT* are almost 100% identical with these in *Coherent UI 2.x*, and reside in the Coherent.UIGT namespace. You should only change the namespace used.

Note

Currently *Coherent GT* does not support BoundObjects, so you'll have to remove them in your projects with GT.

You can check out the "BindingGT" sample about example usage of bindings in GT.

Chapter 8

Asynchronous mode

Coherent GT has support for multithreading which can greatly enhance your game's performance at an added complexity cost.

When using the async mode, most calls to Coherent GT are deferred and executed on another thread. We guarantee that your callbacks and event listeners will still be run on the main thread (the thread that they are registered on, that is).

Furthermore, Coherent GT will never deadlock but race conditions are unavoidable and as such the user must protect against them by himself.

8.1 Using the debugger

Enabling the async mode is as simple as checking the Run Asynchronously option in the CoherentGTSystem component.

Note that some methods are always synchronous and will block your main thread no matter what the current mode is. These methods are:

- `Coherent.UIGT.View.Destroy` and destroying the view component.
- any method that can change the currently loaded URL of a view (`Coherent.UIGT.View.GoForward`, `Coherent.UIGT.View.LoadURL`, etc.)
- IME methods (`Coherent.UIGT.View.IMESetComposition`, etc.) Additionaly the initialization and uninitialization of the UI System are also synchronous.

8.1.1 Debugging JavaScript

Asynchronous mode implies that the JavaScript execution is also deferred on the UI thread. This may require some changes if your existing code depends on the fact that up to now, callbacks were executed in the same stack.

Consider the following communication:

C#:

```
void Bar()
{
    // Code
}

// During initialization
view.BindCall("Bar", (System.Action)this.Bar);

// During your main loop
view.TriggerEvent("foo");
```

JS:

```
engine.on("foo", function fooHandler() {
    // Code
    engine.trigger("bar");
});
```

Calling `view.TriggerEvent("foo")` in the synchronous mode will execute the methods in the following simplified callstack:

Callstack
Bar
JavaScript fooHandler
View TriggerEvent
GameLoop

In the async mode you get no such guarantee. `fooHandler` will be executed whenever the UI thread gets to it and `Bar` will be executed during the first `View.Layout` or `View.ExecuteJSTimers` after `fooHandler` has completed.

8.1.2 Remove implicit dependencies on synchronous mode

Let's look at a more concrete example

C#:

```
void StartGame()
{
    // Initialize the game
}

// During initialization
view.BindCall("StartGame", (System.Action)this.StartGame);
```

JS:

```
startButton.addEventListener("click", function () {
    engine.trigger("StartGame");
    doSomethingAssumingTheGameHasStarted();
});
```

If you are running in a synchronous mode, this code will work fine. When the button is clicked, the engine will call `StartGame` immediately, which will in turn initialize the game. Thus, by the time `engine.call("Start↩Game")` returns you can assume that everything is ready.

This assumption does not hold in asynchronous mode as the execution of `StartGame` will be delayed. To cope with this, use another event to notify the JS that everything is ready:

C#:

```
void StartGame()
{
    // Initialize the game
    view.TriggerEvent(GameStarted);
}

// During initialization
view.BindCall("StartGame", (System.Action)this.StartGame);
```

JS:

```
startButton.addEventListener("click", function () {
    engine.call("StartGame");
```

```
});

engine.on("GameStarted", function () {
    doSomethingAssumingTheGameHasStarted();
});
```

8.1.3 Usage Summary

- If your code depends on the assumption that Coherent GT works synchronously, small changes to it will be required.
- Working with the asynchronous mode resembles the model that most web applications use. You might think about the game as the *server* and your UI as its *client*
- Multithreading requires thorough understanding of the threading model in order to minimize the errors and maximize the profits.

8.2 Debugger in detail

Knowing what Coherent GT does under the hood might be useful if you run into any problems.

If you are to call `Coherent.UIGT.View.Layout` for example, instead of running the layout immediately, Coherent GT will now post a job task to another thread (*the UI thread*) via a queue. The UI thread waits for any tasks to be posted and executes them in order. Note that Coherent will wait for the previous call to layout for this view to complete before issuing the next command in the interest of preventing overtasking the UI thread.

The UI thread can communicate to the main thread via another task queue. For example, when the time for execution of your event handlers (registered via `Coherent.UIGT.View.RegisterForEvent` or through your custom view listener) comes, the UI thread will ask the main thread to run the callback. This guarantees that your code runs only on the main thread. Most of these events will be triggered during the call to `Coherent.UIGT.View.Layout`.

Note that there are no changes to `Coherent.UIGT.ViewRenderer` and you can still paint the view on your rendering thread without complications.

Chapter 9

Live Views (Picture-in-picture)

Coherent GT supports a powerful feature called Live Views. With it developers can embed dynamic content rendered by the 3D engine in their UI. The feature is ideal for 3D player portraits, equipment screens, rear view mirrors in racing games, but also advanced effects like addingparticle effects or complex 3D content inside the interface.

The Live View will act as any other HTML element and you'll be able to animate it, put elements on it etc.

Coherent GT supports a special syntax in the HTML page that marks a certain image as a Live View. When such a view is linked to a render target texture in the game, this texture will be rendered in the UI and will be dynamic. Any changes on the texture will be immediately reflected in the UI. The link is very high performance. No copying is involved. When Coherent GT has to draw the Live View it uses directly the texture provided to the API by the engine.

Live Views should be used only for dynamic content. If you want a one-time image to be set in the UI please use the "coui://" protocol or the JavaScript API. When content is static, Coherent GT can do certain optimizations that can't be done with Live Views as their content is expected to change every frame.

9.1 Live Views setup

A working sample of Live Views can be seen in the "LiveGameViews" sample distributed in the Coherent GT package.

Setting up and using Live Views is very easy. The Live View is represented by a specially set "img" tag inside the HTML page of the UI.

```
<img id="myLiveView" src="coui://MyLiveView"/>
```

The src should start with the "coui://" protocol and your LiveGameView component should have an unique LiveView name. In the example case above, LiveView name should be set to "MyLiveView". This tells Coherent GT that the img will actually be a 'live' texture updated by the game engine. The name of the Live View is given after the protocol and in this case is "MyLiveView". The name is very important, because you can have multiple Live Views simultaneously in the same page or across multiple pages and the name is a way to identify them in the code.

Now that we have the Live View setup in the HTML page, we have to link it in the game. This is done by the Coherent GT Live Game View component. It is supposed to be attached to a GameObject with a Camera that will be the source for the view. Otherwise you can set the Source Camera manually or set a RenderTexture as the Source Texture for the live view. You must set the Target View field to the Coherent GT View component that is showing the page with the "coui://" protocol. The Name field determines which live view in the page this component manages.

9.2 Live Views performance

Live Views are very high performance. No copying is involved. When Coherent GT has to draw the Live View it uses directly the texture provided to the API by the engine. There is no penalty due to the size of the texture as was

the case in Coherent UI 2.x.

If you have HTML elements that intersect a Live View (for instance text on-top of it or elements in the back), they will be re-drawn each frame. This is because the content of the Live View is expected to be dynamic and the parts of the UI that it intersects have to be re-painted with it every frame. If you have many such elements, it is recommended to move the Live View in its own layer. This will prevent it from causing re-draws of elements in the neighbourhood. Moving elements to their own layers is covered in the "Avoiding re-paints with layers" section of the "Best practices" chapter.

Chapter 10

OpenGL Support for Windows

Unity3D often behaves incorrectly on Windows when using OpenGL and we advise you to use DirectX instead of it.

OpenGL in the editor and in the game

With the editor running, U3D improperly attempts to delete rendering resources that it has no ownership over and this breaks GT's rendering. The issue is absent on OSX and is also absent when using DirectX. This makes GT unusable in this particular configuration so you ***must switch to DirectX while working in the editor***.

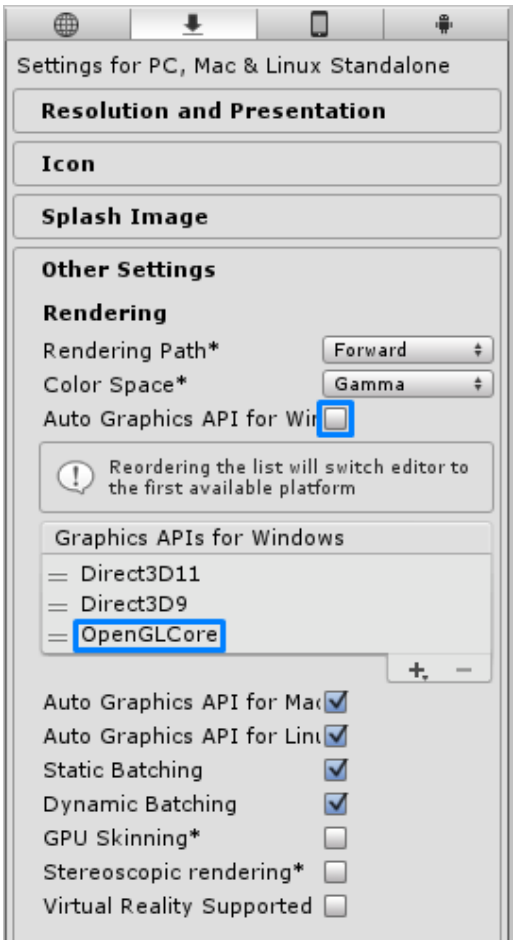
When launching the game outside the editor both Unity3D and GT behave as expected until the game is resized. Resizing causes U3D to break its rendering and this imminently breaks GT rendering too.

For the reasons mentioned above we strongly discourage you from using OpenGl on Windows, both in the editor and in game.

We've submitted tickets to the Unity3D dev team for both problems.

Running with OpenGL

If for whatever reason you insist on using OpenGL, you have to add the OpenGLCore renderer to the list of graphic APIs for Windows.



To do that, go to `Edit -> Project Settings -> Player`. On the `Other Settings` tab make sure that `Auto Graphics API for Windows` option is disabled. In the list `Graphics APIs for Windows` click on the `+` button and select `OpenGLCore`. Now you can build your application and run it with the `-force-glcore` flag.

Chapter 11

Gamepad Support

To enable gamepad input for the UI in Unity3D application you can use the `CoherentUIGTGamepad` component. It automatically gets your settings from the Input Manager and creates mappings for your gamepads. You can also fine tune the mappings from the component.

If you use third party input processing, you can manually register a gamepad and update its state with the following methods found on `Coherent.UIGT.UISystem`:

- `RegisterGamepad` will tell the system to expect events from a new gamepad with the given description.
- `UpdateGamepadState` will give the system the latest state of the gamepad which consists of how much each button has been pressed and each axis moved.
- `UnregisterGamepad` will remove a gamepad from the system.

Keep in mind that gamepads are defined system-wise and not per view. This means that all views share the same gamepads. Although updating the states multiple times per frame is not harmful, there's no reason to do so.

If a view loses focus, it will continue to receive updates. You need to handle that in your JavaScript if this behaviour is undesired.

Chapter 12

Localization

To enable the localization facility in Unity3D you need to attach the `CoherentUIGTLocalization` component to a gameobject in your scene and populate its fields. Then make sure `Enable Localization` box on the `CoherentUIGTSystem` is ticked.

Check out the localization sample scene from the package to see it in action.

`CoherentUIGTLocalization` component supports importing and exporting of CSV files. It accepts the following format: `id, text`. There are couple of .csv's in the localization sample folder you can look at.

More useful things to know:

- In your HTML page you need to add the `data-l10n-id` attribute to any element you need localized.
- To change the language you need to call `ChangeLanguage` on the current system localization manager and pass the language Id.
- Localization is system-wide and not per view. You only need one localization component per scene. You can also call `DontDestroyOnLoad` on its gameobject to make it persistent between the scenes.

Chapter 13

Unity3D uGUI Integration

To use a `Coherent GT View` inside Unity3D's new GUI, you need to draw it onto a texture and forward the input from that texture to the view as page coordinates. There is a sample included in the package that shows how that works.

The scene from the sample contains 4 important components:

- `Raw Image` component to draw onto.
- `RenderViewOnRawImageGT` script that renders the view onto the `Raw Image` component.
- `InputForwardFromUnityGUIGT` script that translates the input coordinates from the `Raw Image` component to page coordinates.
- `Coherent GT View` component.

You can reuse the two scripts from the sample in your scenes if you need to use `Coherent GT View` inside Unity3D's UI. Just add these four components onto a gameobject. If you need to disable the input, just disable the `InputForwardFromUnityGUIGT` component.