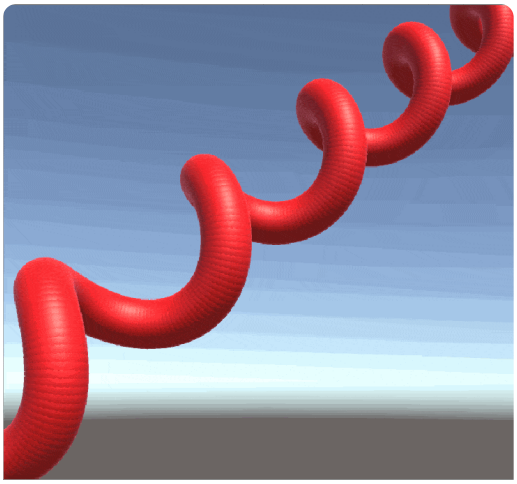
Case Study: Using the Unity Profile Analyzer

In this short case study, we'll make a quick assessment on the impact that the structure of the scene hierarchy causes on game performance.

For this post, we're going to ask ourselves this question: *are plain hierarchies really better than deep hierarchies?*

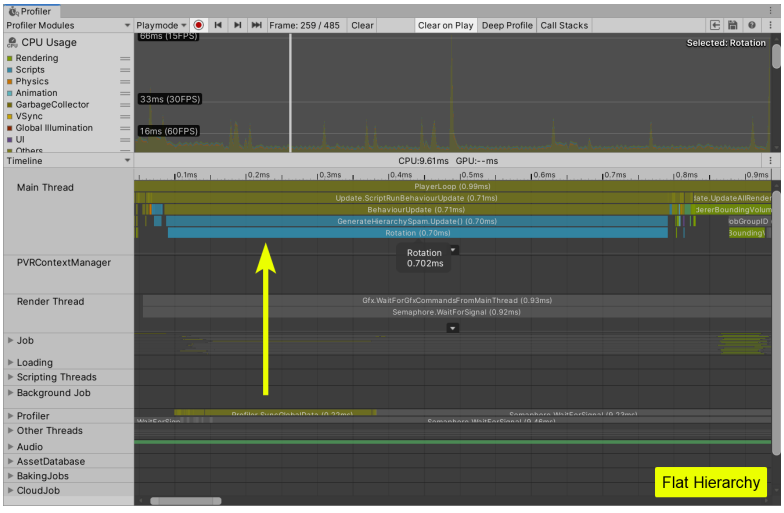
To answer this, I created a small project that spawns a high number of simple shapes that have a sphere collider attached to them. These shapes will do a sim rotation per frame as shown below.



Unity Sample Scene: Testing Side Effects of Hierarchy Changes

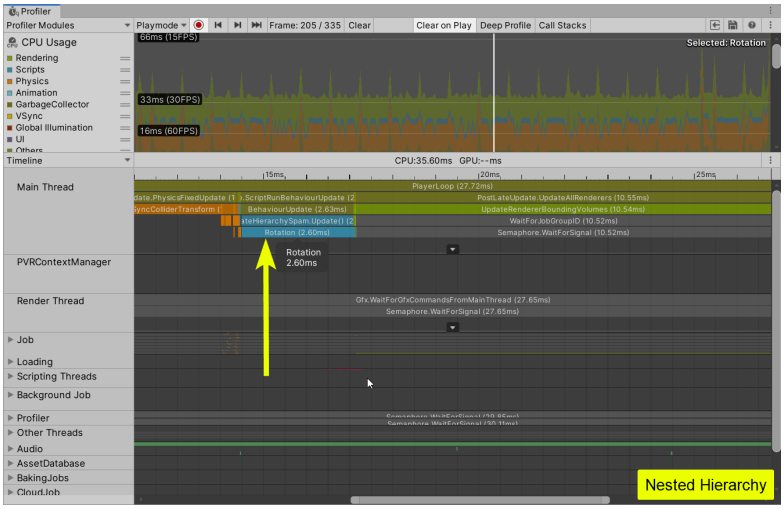
One version will rotate the spheres using a flat hierarchy. The other version will have each sphere nested to each other, forming a long parenting chain.

We start by capturing 300 frames of the flat version using the unity profiler. When we're done, we save the profile to a file named *FlatHierarchy.data*, as you ca below.



Unity Profiler: Flat Hierarchy

Then, we do the same for the nested hierarchy version and we save the profile as *NestedHierarchy.data*:



Unity Profiler: Nested Hierarchy

Now, if we just compared the cost of our **Update** function, we would only be able to explain a difference of 1.90 milliseconds.

But as we saw in the post, that's only one part of the story. The same update loop can have different side-effects on the game performance that we're not able directly.

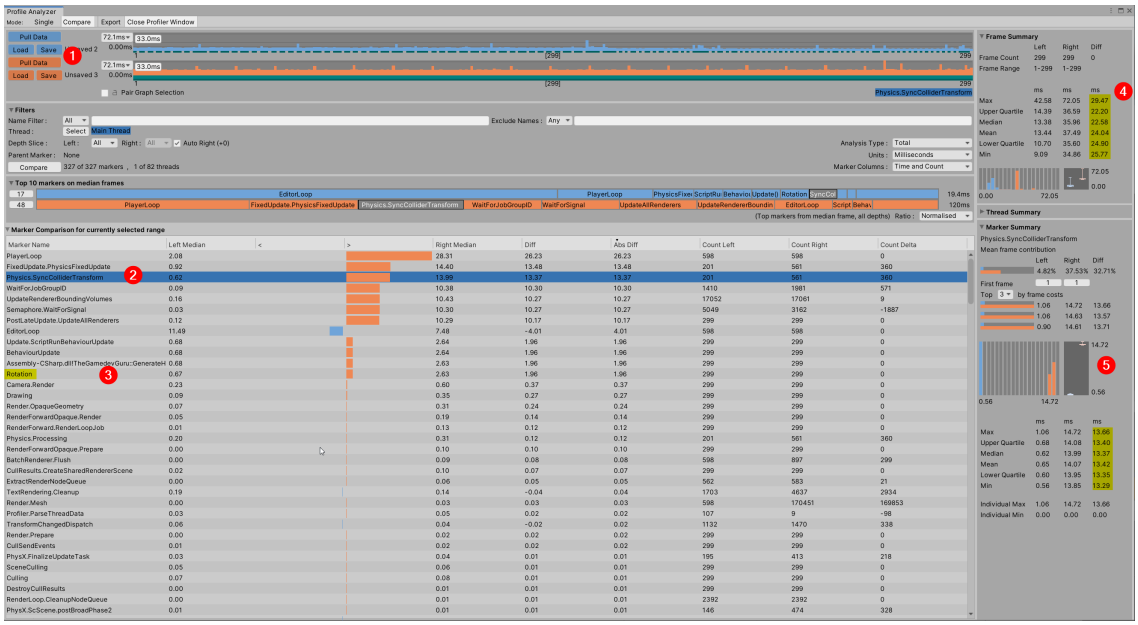
And this is exactly where the Unity Profile Analyze comes in handy.

So we open it through *Window → Analysis → Profile Analyzer*.

As we intend to compare two scenarios, we switch to compare mode on the top part of the window and then we load both profiling sessions by clicking on *Pu*

Pulling data means: load the currently displayed profile in the unity profiler. So for the comparison to work, you must have saved and loaded the profiles as w

Anyway, here's the comparison we were looking for:



Comparing Flat vs Nested Hierarchy Performance Impact

The Unity Profile Analyzer uses the concept of left and right profiles when comparing two profiling sessions. It's an arbitrary choice you can make, but I ofter this pattern: left is my baseline version and right is where I introduce a single change or system.

In this case, the left profile is the flat hierarchy version, while the right one represents the nested hierarchy. This corresponds to the question: how is using a hierarchy going to impact my game performance?

Now that we loaded both profiles, Unity will compute the statistical differences between them.

The most noticeable thing that comes to view is the sharp timing difference on **Physics.SyncColliderTransform**, as I show you in **(2)**.

In the flat hierarchy, this marker takes about 0.6ms/frame. However, with the profile analyzer we see that the same marker steals about 14ms in the nested hi build. That's a huge difference.

Similarly, we see the same trend with other markers such as **UpdateRendererBoundingVolumes**.

And as I've told you in the original post, **most developers stick to comparing the cost of their Update functions**, as you can see in **(3)**. If we stayed there, only be able to explain a difference of 2 milliseconds.

This the biggest reason to analyze the game as a whole, and not just the pure C# logic of specific subsystems.

It all comes down to this: **if your code interacts with the Unity API, chances are high that you'll be introducing side effects that you must measure**.

In **(4)** you can see the aggregated performance difference distributed over quartiles. Upper quartiles (75%) is one of my favorite metrics, as they are very usef smooth gameplay. And this is critical for VR experiences.

As you can see in the comparison image, selecting the Physics.SyncColliderTransform brought up the **marker summary** I show you in **(5)**. This panel gives u statistical summary specific to this marker.

These numbers are self-explanatory if you studied statistics. If you didn't, start analyzing the mean difference and move towards quartiles in the long term.

For this sample test, we can make the following conclusion: a flat hierarchy greatly outperforms a nested hierarchy, therefore you should avoid deep hierarch game as much as possible.

Bonus Content

applying transforms on elements nested in deep hierarchies triggers a cascade of expensive operations, as these elements a interconnected at different levels (transforms, physics, UI, etc.).

Also, Unity can't parallelize these operations with the job system because they're now highly interdependent.

I hope you found this document useful. Feel free to reach out at ruben@thegamedev.guru if you have any question in your mind (you should have).

If you need extra help with your Unity project, apply for my **unity game development coaching** here.

Ruben