COMP350: Algorithms & Optimisation

2: PS4 Dev Kit & Profilers

Learning outcomes

By the end of today's session, you will be able to:

- **Develop** games for the PS4
- **Understand** the usage of a profiler
- **Profile** your own code base¡/li¿

Developing for the PS4

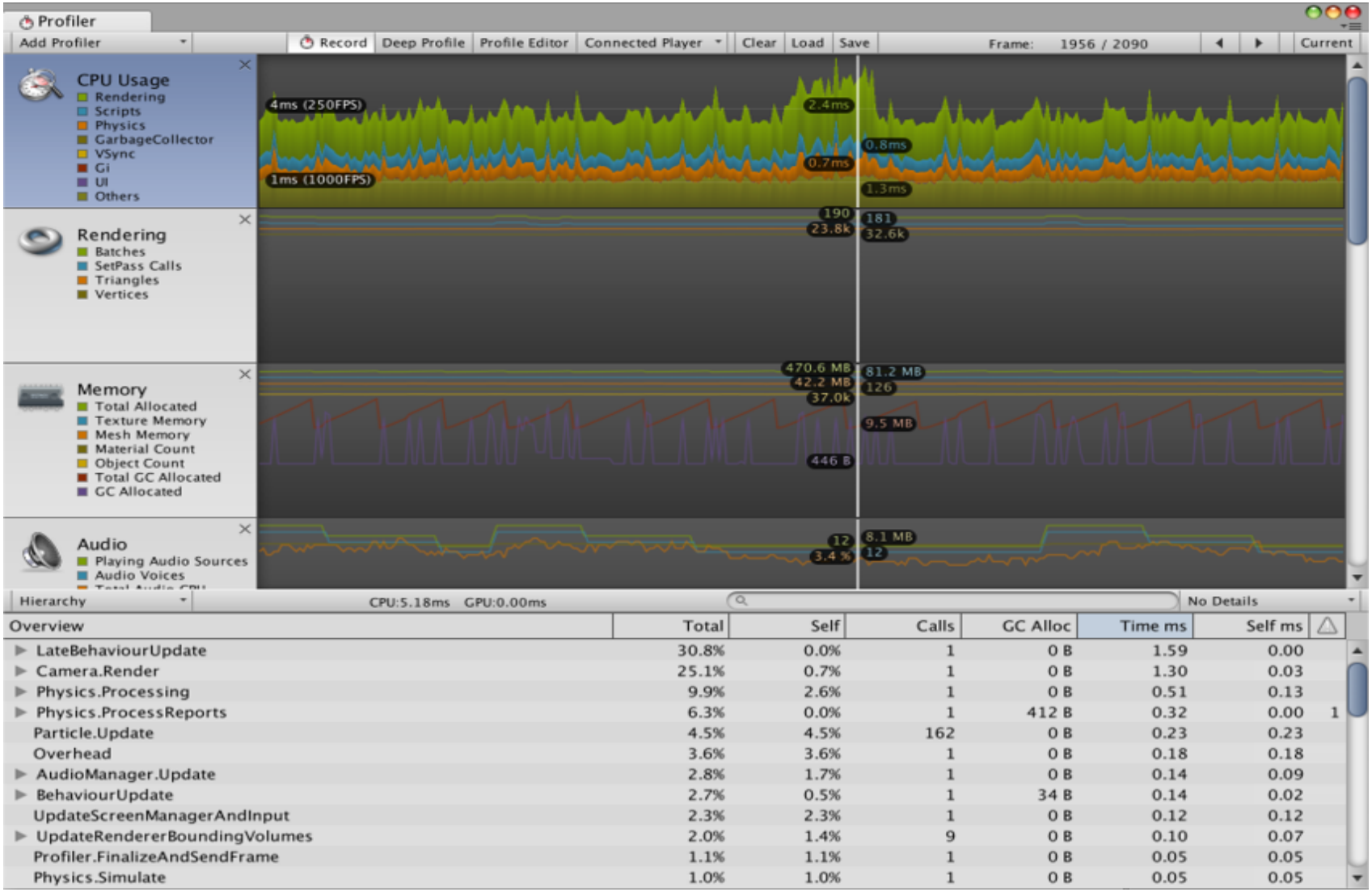
Coffee Break

Profilers

Unity Profiler

- The Unity Profiler is built into the engine
- It can be accessed via the **Window ¿ Profiler**
- This allows you to profile the following
  - CPU Usage - Scripts, Physics, UI etc
  - Rendering - Batches, Triangles, Vertices
  - Memory - Total, Texture, Mesh, Garbage Collection
  - Audio - Number of Sources, Audio Memory
  - GPU - Deferred Lighting, Transparent, Post Processing

Unity Profiler



Overview	Total	Self	Calls	GC Alloc	Time ms	Self ms
▶ LateBehaviourUpdate	30.8%	0.0%	1	0 B	1.59	0.00
▶ Camera.Render	25.1%	0.7%	1	0 B	1.30	0.03
▶ Physics.Processing	9.9%	2.6%	1	0 B	0.51	0.13
▶ Physics.ProcessReports	6.3%	0.0%	1	412 B	0.32	0.00
Particle.Update	4.5%	4.5%	162	0 B	0.23	0.23
Overhead	3.6%	3.6%	1	0 B	0.18	0.18
▶ AudioManager.Update	2.8%	1.7%	1	0 B	0.14	0.09
▶ BehaviourUpdate	2.7%	0.5%	1	34 B	0.14	0.02
UpdateScreenManagerAndInput	2.3%	2.3%	1	0 B	0.12	0.12
▶ UpdateRendererBoundingVolumes	2.0%	1.4%	9	0 B	0.10	0.07
Profiler.FinalizeAndSendFrame	1.1%	1.1%	1	0 B	0.05	0.05
Physics.Simulate	1.0%	1.0%	1	0 B	0.05	0.05

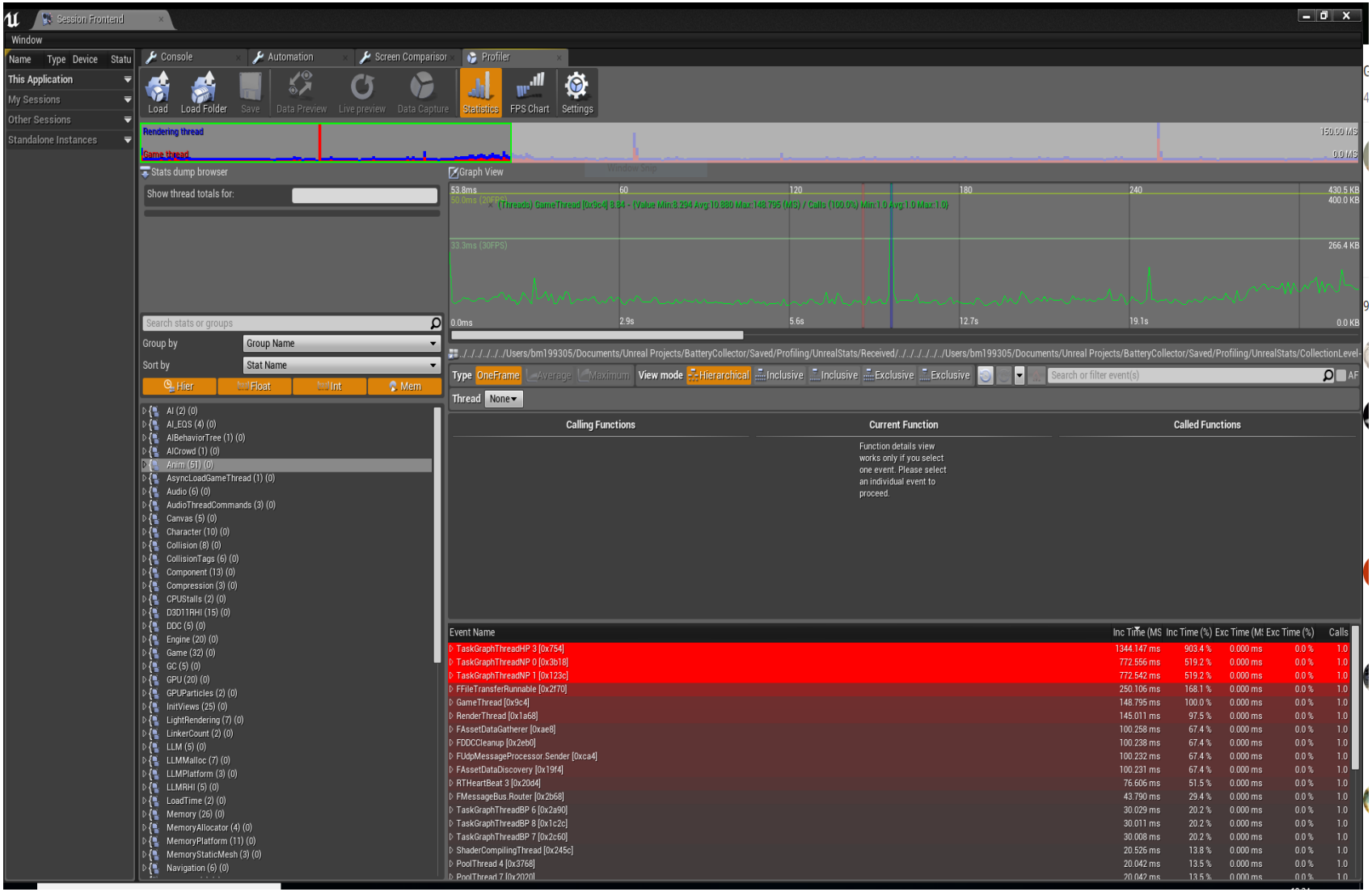
Unity Profiler: Hints & Tips

- You can remove items from the profiler graph by click on the colour box
- Enabling **Deep Profile** will add a significant overhead to larger games
  - Surround you code with **Profiler.BeginSample** & **Profiler.EndSample** this will appear in the Profiler
- You should consider Profiling a development build as the Editor adds significant overheard

Unreal Profiler

- The Unreal Profiler is built into the engine
- It can accessed via **Window ¿ Developer Tools ¿ Session Frontend**
- Allows us to profile all major systems including CPU (code) and GPU

Unreal Profiler



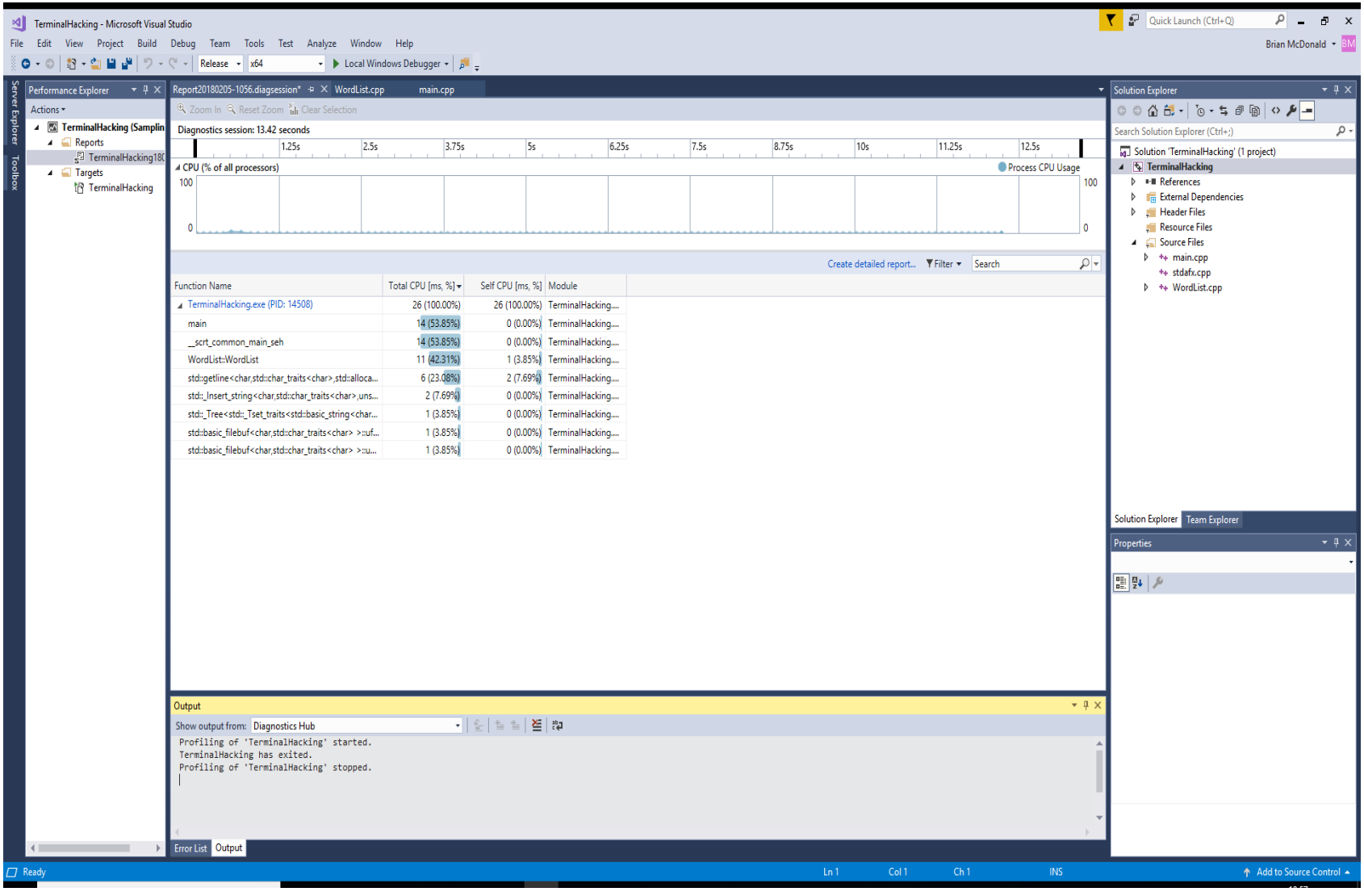
Unreal Profiler: Hints & Tips

- `https://www.unrealengine.com/en-US/blog/how-to-improve-game-thread-cpu-performance`
- A few interesting things from this link
  - To identify the bottleneck (GPU or CPU), run **stat unit** on a **non-debug build**
  - If the **Frame Time** is very close to **GPU Time**, then the GPU is the bottleneck
  - If the **Frame Time** is very close to **Game Time**, then your code is the bottleneck
  - In the profiler **GameThread** entry, find the **FTickFunctionTask** - this shows every actor and component that is ticking
  - Another thing to track is **Blueprint Time**, switch inclusive view and locate it, then switch back to hierarchical view
  - **SkinnedMeshComp Tick** & **TickWidgets** can also be bottleneck

Visual Studio Profiler

- `https://msdn.microsoft.com/en-us/library/ms182372.aspx`
- Switch your application to a release build
- To run the profiler, select **Debug ¿ Performance Profiler** and then click on **Performance Wizard**
- The profiler will run and start collecting data
- Close the application to start analysing the data

Visual Studio Profiler



Visual Studio: Hints & Tips

- Click on **Create Detailed Report** in the summary view, this will generate a report on your application
- In this report **Show Hot Lines** will show the code paths which do the most work
- You will not be able to do much about the *.dll calls, you should look at your own functions in here

Exercises

Profiler Exercise

1. Select a project (sample, past project, etc ,etc)
2. Open up the project and profiler
3. Run the profiler to see if you can find bottlenecks
4. Record all sources you have used

- You may have to do some research on the profiler
- The previous slides contain some links but you may need to find additional sources