

Game Dev: Profiling

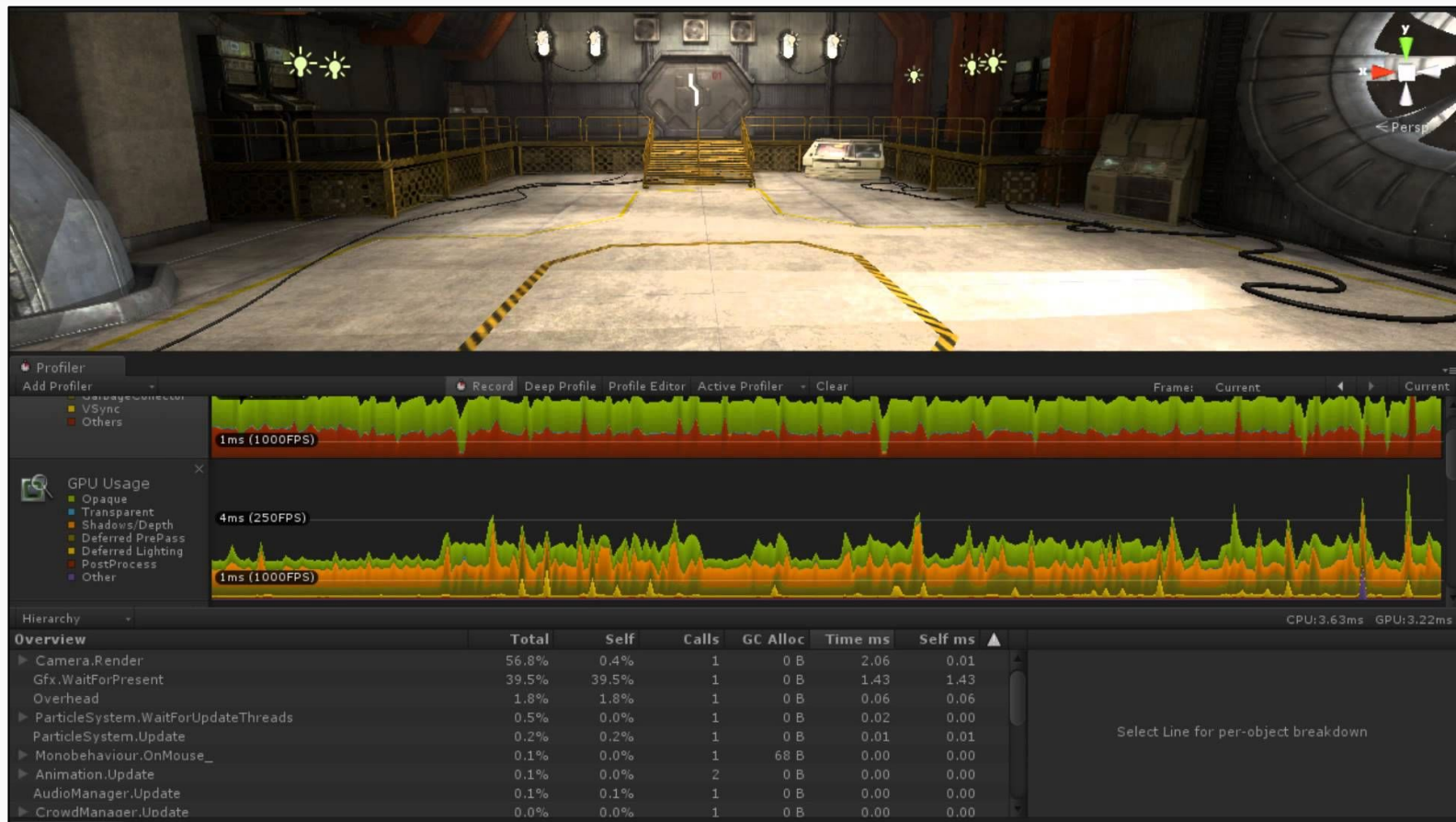
Ricard Pillosu - UPC



Profiling in Video Games

- Complexity and variability of game code makes it unpredictable
- And we want to use 100% of the hardware resources efficiently
- Coders / Artist / Designers use **profiling** to understand the consequences in **performance** of their choices for content of the video game
- We track many things like memory, cpu load, gpu load, etc ...

Profiling in Unity



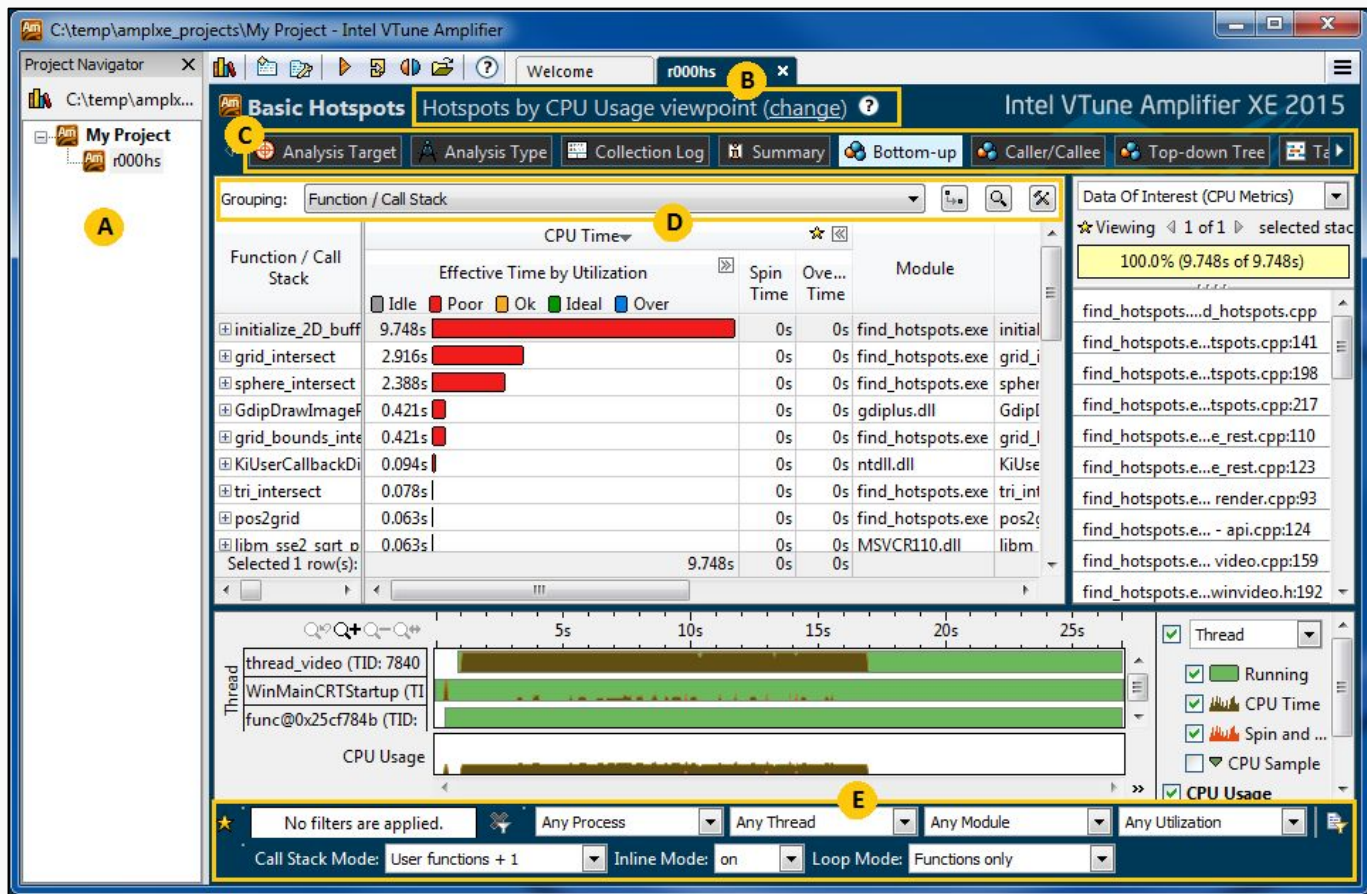
Profiling in Unreal Engine



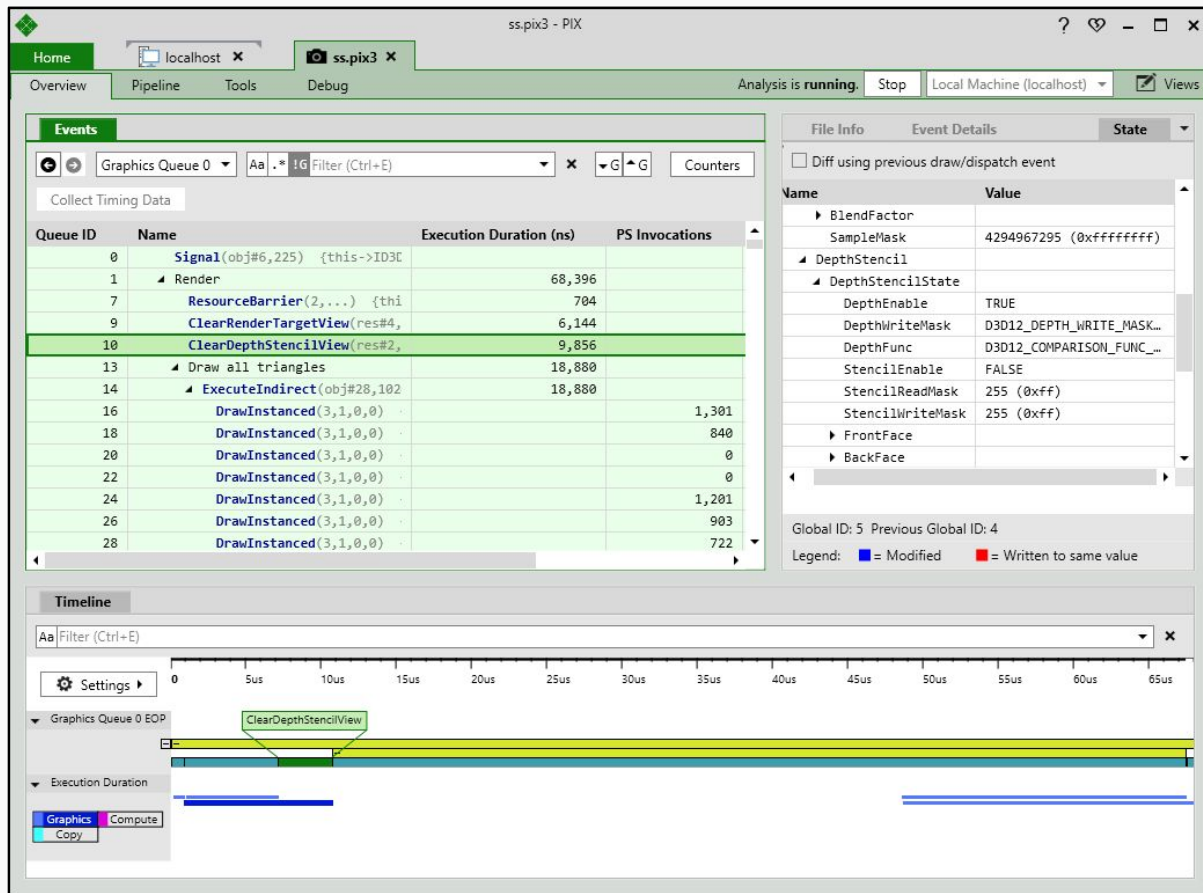
Profiling your own code



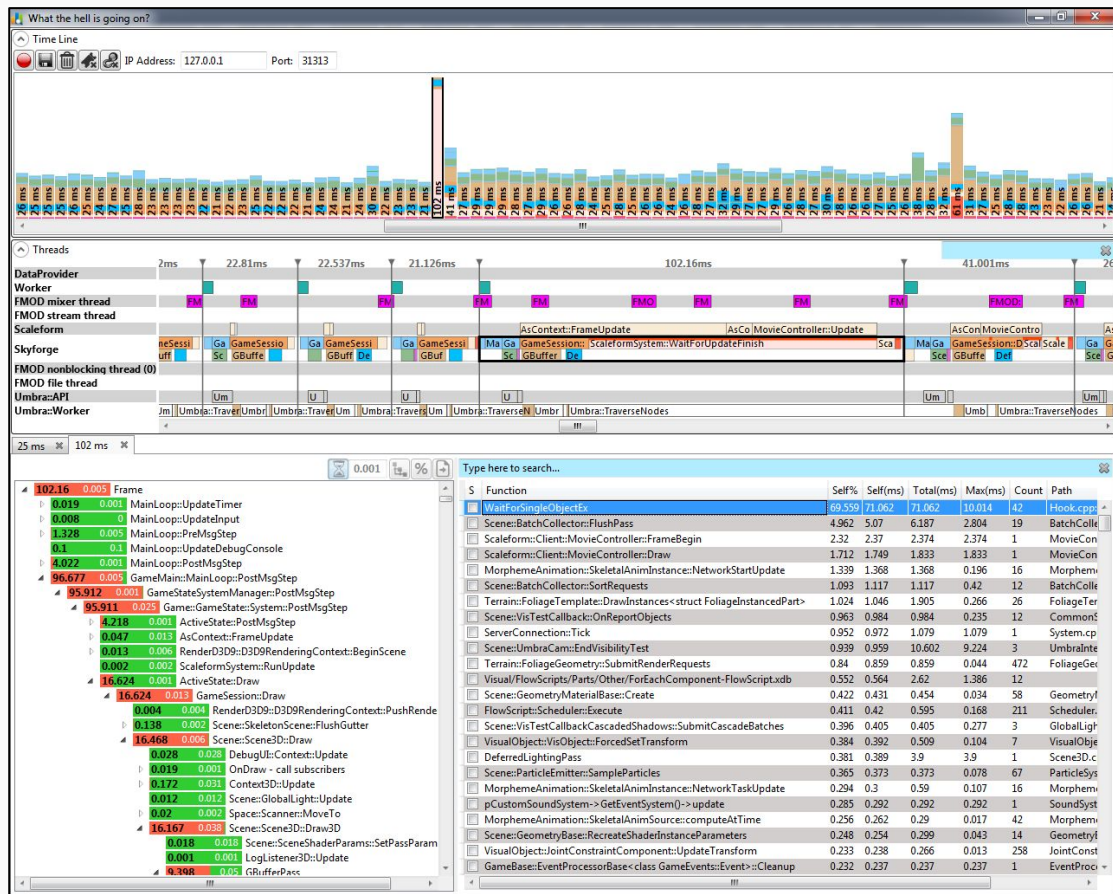
Profiling with Intel's VTune



Profiling with Microsoft's PIX



Profiling with Brofiler



Brofiler

- We will integrate Brofiler
- It requires [code instrumentation](#)
- This means that we need to use its library to inject C++ code in our game

```
////////////////////////////////////  
void Engine::UpdateInput()  
{ BROFILER_CATEGORY( "UpdateInput", Profiler::Color::SteelBlue )  
  SlowFunction2();  
}  
////////////////////////////////////  
void Engine::UpdateMessages()  
{ BROFILER_CATEGORY( "UpdateMessages", Profiler::Color::Orange )  
  SlowFunction<REPEAT_COUNT>();  
}  
////////////////////////////////////  
void Engine::UpdateLogic()  
{ BROFILER_CATEGORY( "UpdateLogic", Profiler::Color::Orchid )  
  SlowFunction<REPEAT_COUNT>();  
}  
////////////////////////////////////  
void Engine::UpdateScene()  
{ BROFILER_CATEGORY( "UpdateScene", Profiler::Color::SkyBlue )  
  SlowFunction<REPEAT_COUNT>();  
}  
////////////////////////////////////  
void Engine::Draw()  
{ BROFILER_CATEGORY( "Draw", Profiler::Color::Salmon )  
  SlowFunction<REPEAT_COUNT>();  
}  
////////////////////////////////////  
void Engine::UpdatePhysics()  
{ BROFILER_CATEGORY( "UpdatePhysics", Profiler::Color::Wheat )  
  int64 time = Profiler::GetTimeMicroSeconds();  
  while (Profiler::GetTimeMicroSeconds() - time < 20 * 1000) {}  
}
```

Brofiler: integration steps

- Include Brofiler header and lib
- Mark the beginning of the main loop using:
 - `BROFILER_FRAME("YourThreadName")`
- Mark any function that you want measured:
 - `BROFILER_CATEGORY("UpdateLogic", Profiler::Color::Orchid)`

```
////////////////////////////////////  
void Engine::UpdateInput()  
{ BROFILER_CATEGORY( "UpdateInput", Profiler::Color::SteelBlue )  
  SlowFunction2();  
}  
////////////////////////////////////  
void Engine::UpdateMessages()  
{ BROFILER_CATEGORY( "UpdateMessages", Profiler::Color::Orange )  
  SlowFunction<REPEAT_COUNT>();  
}  
////////////////////////////////////  
void Engine::UpdateLogic()  
{ BROFILER_CATEGORY( "UpdateLogic", Profiler::Color::Orchid )  
  SlowFunction<REPEAT_COUNT>();  
}  
////////////////////////////////////  
void Engine::UpdateScene()  
{ BROFILER_CATEGORY( "UpdateScene", Profiler::Color::SkyBlue )  
  SlowFunction<REPEAT_COUNT>();  
}  
////////////////////////////////////  
void Engine::Draw()  
{ BROFILER_CATEGORY( "Draw", Profiler::Color::Salmon )  
  SlowFunction<REPEAT_COUNT>();  
}  
////////////////////////////////////  
void Engine::UpdatePhysics()  
{ BROFILER_CATEGORY( "UpdatePhysics", Profiler::Color::Wheat )  
  int64 time = Profiler::GetTimeMicroSeconds();  
  while (Profiler::GetTimeMicroSeconds() - time < 20 * 1000) {}  
}
```

Brofiler Colors Reference

AliceBlue AntiqueWhite Aqua Aquamarine Azure Beige Bisque Black BlanchedAlmond Blue BlueViolet
Brown BurlyWood CadetBlue ChartreuseChocolate Coral CornflowerBlue Cornsilk Crimson Cyan
DarkBlue DarkCyan DarkGoldenRod DarkGray DarkGreen DarkKhaki DarkMagentaDarkOliveGreen
DarkOrange DarkOrchid DarkRed DarkSalmon DarkSeaGreen DarkSlateBlue DarkSlateGray
DarkTurquoise DarkViolet DeepPinkDeepSkyBlue DimGray DodgerBlue FireBrick FloralWhite
ForestGreen Fuchsia Gainsboro GhostWhite Gold GoldenRod Gray Green GreenYellowHoneyDew
HotPink IndianRed Indigo Ivory Khaki Lavender LavenderBlush LawnGreen LemonChiffon LightBlue
LightCoral LightCyanLightGoldenRodYellow LightGray LightGreen LightPink LightSalmon LightSeaGreen
LightSkyBlue LightSlateGray LightSteelBlue LightYellow LimeLimeGreen Linen Magenta Maroon
MediumAquaMarine MediumBlue MediumOrchid MediumPurple MediumSeaGreen
MediumSlateBlueMediumSpringGreen MediumTurquoise MediumVioletRed MidnightBlue MintCream
MistyRose Moccasin NavajoWhite Navy OldLace OliveOliveDrab Orange OrangeRed Orchid
PaleGoldenRod PaleGreen PaleTurquoise PaleVioletRed PapayaWhip PeachPuff Peru Pink Plum
PowderBlue Purple Red RosyBrown RoyalBlue SaddleBrown Salmon SandyBrown SeaGreen SeaShell
Sienna Silver SkyBlue SlateBlue SlateGray Snow SteelBlue Tan Teal Thistle Tomato Turquoise Violet
Wheat White WhiteSmoke Yellow YellowGreen

TODO 1 / 2 / 3

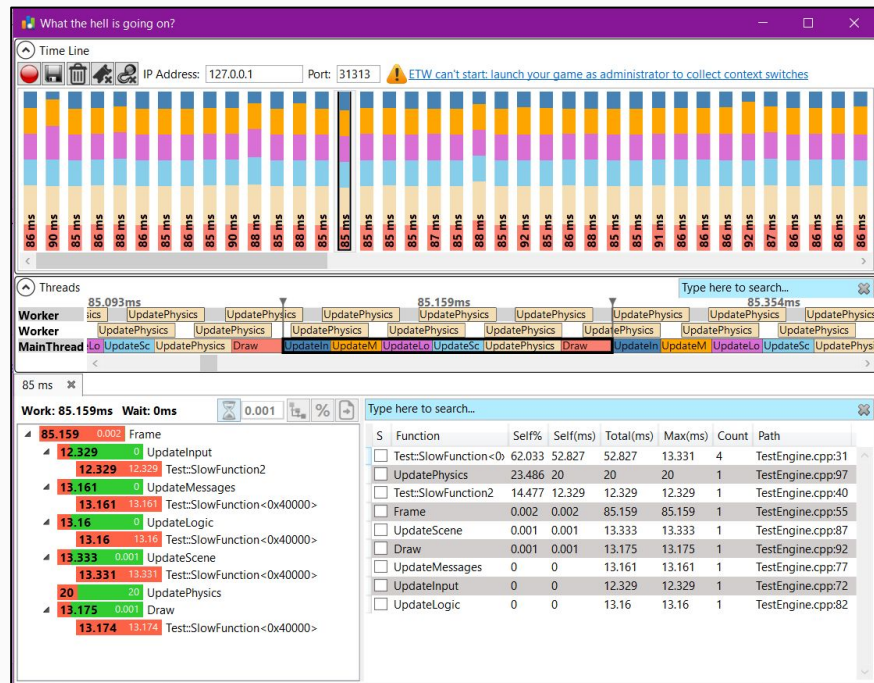
- Add profiler in main.cpp in
 - Add the header
 - Add the library
 - Add the macro to mark the beginning of the main loop
- Now trace App methods:
 - Add the Profile header
 - Add Profiler macros to trace all *Update methods from App

Brofiler GUI

You can see the distribution of use of millisecond on the top panel.

Middle area is for threads

Bottom has the hierarchy and the details of performance usage



References

- [Brofiler](#) is in active development, check it's latest code [here](#)
- Research from students [here](#) and [here](#)
- Will need to follow the [development of PIX](#)

Homework

Add code instrumentalization to measure in ms:

- Cost of your A* / Dijkstra / BFS
- Cost of each module preUpdate / update / postUpdate
- Check cost of Render::PostUpdate with and without vsync
- Check j1Map::Draw