

Multimedia and computer animation

API Unreal Engine 5

Basic information

Programming UE5 using C++:

- Language standard
 - C++20
- Standard library
 - Use UE equivalent types and functions (TMap, TArray, TPair, ...)
 - Some exceptions to this rule
 - <cmath>
 - <regex>
 - <limits>
 - <atomic>
 - ... see the source site below
- Garbage collector
 - UPROPERTY
 - Nulls pointers for destroyed objects

Sources:

- https://dev.epicgames.com/documentation/en-us/unreal-engine/epic-cplusplus-coding-standard-for-unreal-engine?application_version=5.6



Basic information

UE5 conventions:

- Class name prefixes:
 - A = class derived from AActor
 - U = class derived from UObject - more generic than AActor
 - T = template classes (i.e. TArray)
 - E = enumerations
 - F = most other classes with a couple of exceptions (i.e. ConstructorHelpers)
 - ... and some other prefixes, see sources for more

Sources:

- https://dev.epicgames.com/documentation/en-us/unreal-engine/epic-cplusplus-coding-standard-for-unreal-engine?application_version=5.6



Variables and functions

UPROPERTY specifiers:

- **EditAnywhere / VisibleAnywhere** - allows to edit or show a variable in the editor
- **BlueprintReadWrite / BlueprintReadOnly** - controls access to variables from blueprints
- **EditDefaultsOnly / EditInstanceOnly** (or Visible*) - to allow editing the variable only for an instance of the class, or only for the archetype (default object)

UFUNCTION specifiers:

- **BlueprintCallable** - allows for blueprints to call a function

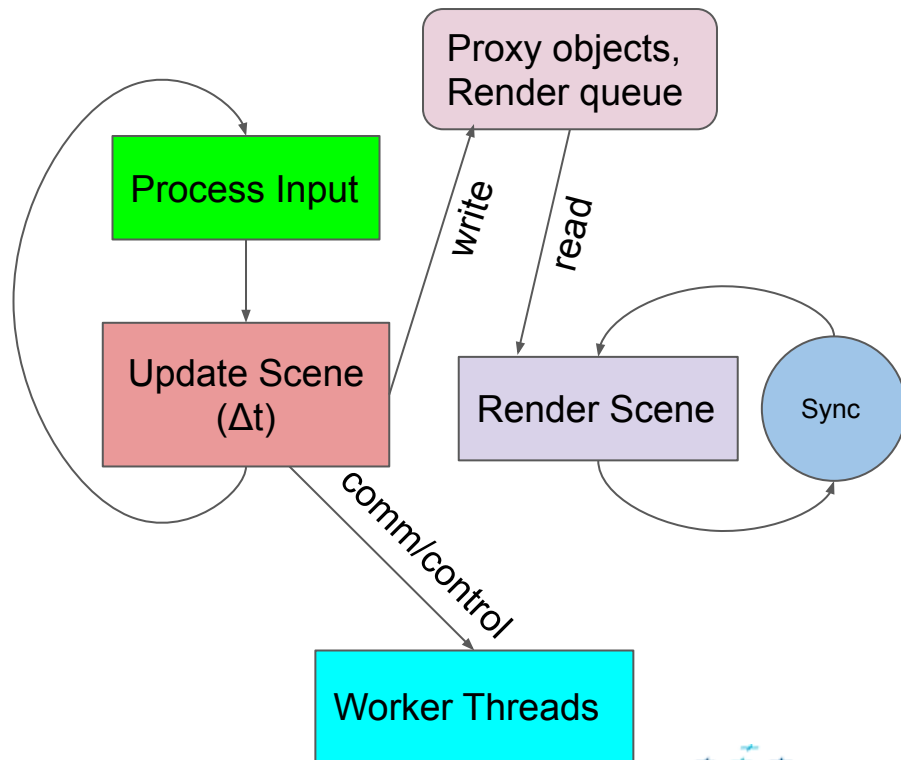
Usable for both:

- **Category** - category tree in which the variable/function can be found (uses “|” pipe as separator)



Game loop

- Game thread
 - Gathers user inputs
 - Processes game objects
 - UI
 - Creates render command queue
- Render thread (+ RHI thread)
 - Uses proxy objects (scene data)
 - Processes and runs commands from the main thread
- Worker threads
 - Animation
 - Audio
 - TaskGraph, FRunnable, ...

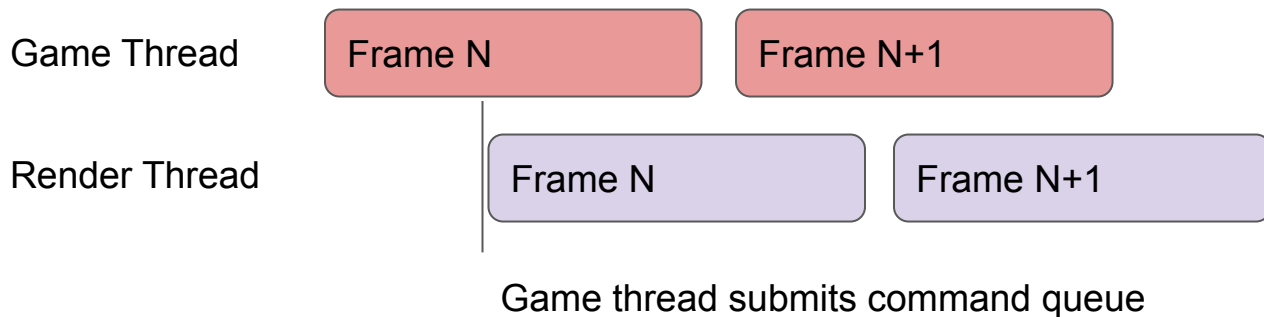


Sources:

- https://dev.epicgames.com/documentation/en-us/unreal-engine/epic-cplusplus-coding-standard-for-unreal-engine?application_version=5.6

Game Thread vs. Render Thread

- Game thread handles tick synchronization
 - Render thread stays about 1-2 frames behind
 - Separates rendering commands and their execution



Sources:

- https://dev.epicgames.com/documentation/en-us/unreal-engine/epic-cplusplus-coding-standard-for-unreal-engine?application_version=5.6

TaskGraph

- Short tasks
- Handles task dependencies
- Simple execution - AsyncTask (header "Async/Async.h")

```
void AsyncTask  
(  
    ENamedThreads::Type Thread,  
    TUniqueFunction < void ()> Function  
)
```

FRunnable

- Longer tasks and computation
- Ideally over several frames
- Three main methods:
 - Init() - executed at the start of the thread
 - Run() - executed after Init, place for the actual computation
 - Stop() - executed when Run finishes, to free up resources, etc.
- Tutorial:
 - <https://unrealcommunity.wiki/multithreading-with-frunnable-2a4xuf68>

ParallelFor

- Simple parallelization
- Be careful about dependencies on other indices
 - Ideally the task for each index is independent from others, dependent i.e. only on the previous simulation state

- Function in the second argument is either a lambda (like in this example) or a pointer to a function.

```
for(int i = 0; i < Particles.Num(); ++i)
{
    auto ForceToApply = ComputeParticleForce (i);
    Particles [i]->GetStaticMeshComponent ()->AddForce (ForceToApply);
}
```



```
ParallelFor (Particles.Num(), [this](int32 i) -> void const
{
    auto ForceToApply = ComputeParticleForce (i);
    Particles [i]->GetStaticMeshComponent ()->AddForce (ForceToApply);
});
```



Thread-safety

- Typical Unreal Engine objects are not thread-safe
 - UObject
 - AActor
 - TimeManager
 - ...and others
- Use the usual constructs when working with multiple threads, like mutexes or atomic variables