Golden Rules

* Use the same style you’re seeing when modifying an existing file.
* No Singletons!
* Use least global variables as possible.

Class Organization

* Separate definitions and implementations into .h and .cpp.
  + Unless it’s a template class, which can’t separate implement from header file.
* Only use private function if private member is going to be accessed.
  + Otherwise, we can define the function in cpp file locally.
  + Another exception is friend class.
* Always initialize values in constructor instead of inline declaration.
  + So that if we change our minds for initial value, it doesn’t need to compile all files which include the header.
* Order the definitions and functions as public -> protected -> private

Folder Organization

* Put debug only code in Editor/
* Put Runtime code in Runtime/
  + Put components in Components/
  + Put core implementation in Engine/
  + Put classes defines in Classes/
  + Put rendering related stuffs in Renderer/
* Put HLSL in Shaders/
* Put raw fbx in RawAssets/Meshes/
* Put raw texture in RawAssets/Textures/
* UHAssets and asset caches will be generated into Assets/ and AssetCaches/

Naming Conventions

* The first letter of each word in a name (such as type name or variable name) is capitalized, and there is usually no underscore between words. For example, Health and UPrimitiveComponent are correct, but not lastMouseCoordinates or delta\_coordinates.
* All classes/structs/enums are started with a prefix UH.
  + UHEngine, UHMaterialConstant, UHRenderPassType for example.
* Variable, method, and class names should be clear, unambiguous, and descriptive. The greater the scope of the name, the greater the importance of a good, descriptive name. Avoid over-abbreviation.
* Put “b” as a prefix in the start of every Boolean variables
  + E.g. bIsThisGood.

Type / Container

* Always use the integer defines in <cstdint>
  + E.g. uint32\_t, int16\_t
* Always use std::string or std::wstring instead of char array. Unless it’s API limitation.
* Always use std::array for static array, and std::vector for dynamic array.
  + If the static array dimension is larger than 1, for example a 2D array. Then it’s okay to use C-style static array.
* Always cast variable with static\_cast<T>
  + If the code piece is verbose, then it’s fine to use C-style cast.
  + Of course, it’s okay to use reinterprest\_cast<T> for special cases.
* Always call clear() for std containers except static array despite C++ does it for you.

Const Correctness / Formatting

* Always use const if the variables aren’t going to be changed.
* Always use parentheses in if-else/for loop/switch blocks.
* Use guard clause for blocks with verbose if statement.

Ranged For Loop

* Always use ranged for loop if index information isn’t needed.
* Note the const correctness in the ranged loop.

Pointer Management

* Use nullptr for custom pointers.
* Use VK\_NULL\_HANDLE for Vulkan pointers.
* Always use smart pointer for new instance.
* Always use std::unique\_ptr as default.
* Always use raw pointer for cache only.
* Always call reset() for smart pointer in release function despite C++ does it for you.

Comments / Maintenance

* Always attach reference website or source if implementation isn’t completely original.
* Always write comment if something really needs to be explained. Otherwise, keep good habit for naming variables.
* Always revisit smelly code if available.
* Always try to reuse code piece which are likely duplicating multiple times.

Debug / Release

* Always wrap code with WITH\_DEBUG if it’s debug only.

DirectXMath

* Always use XMFLOAT, XMFLOAT4X4..etc as class member
* Only convert to XMVETOR/XMMATRIX if computation is needed

Compute / Ray Tracing Shader

* Always remember (+0.5f) when converting dispatch index to UV.