우리 코드는 왜 더러운가 version001

2025-12-24

한국공학대

PulseCity 팀

2021180007 노훈철

코드가 더럽다고 느끼는 부분

1. 코드를 딱 봤을때, 이 코드가 뭐하는지 모를때
   - 더 들어가면 코드를 파악하려 노력을 해도 알기가 쉽지 않을때.

2. 코드가 기기에서 비효율적으로 실행될때.

3. 코드 수정할때의 어려움이 있을때
   - 해당 코드를 만든 사람이 그 코드의 수정방법을 제공해야 하고,
   - 이 이상의 수정을 할때 어려운 것은 이 경우가 아니다.

4. 아직 구현될 부분이 구현되지 않음. 결국 최종적인 구조는 이런게 아닌데, 여기에 뭘 게속 올리고 있으니 부담이 될 수 밖에 없음.

1-1 개선사항 : 각자 다른 코드를 쓰지만, 서로 이해할 시간은 없음.

결국 다수가 코드를 작성하게 되면, 각자 다른 부분을 만들 수 밖에 없지만,
결국 그 결과들이 맞물려 게임이 돌아가기 때문에, 각자 다른 작업을 하지만,
내가 하지 않은 작업에 대한 이해가 필요하다.

하지만 그곳에 시간을 투자하는 것을 사람들은 꺼려하는 현상이 있다.
때문에 새로운 기능이나 물체를 추가하는 것에서
쓸데없는 고생을 하거나, 뭔가 많이 해매일 수 밖에 없다.
그럼 비효율적인 개발이 되고 뭐 하나 추가하는 데도 시간이 많이 걸린다.

이를 보완하기 위해 어떻게 할 것인가? 여기에 너무 많은 시간을 쏟으면 안된다.
그리고 너무 빡빡하게 지키려고 하거나, 작업에 방해가 되면 안된다.
단지 가볍게 생각날때 보완하고, 다음달 넘어가기 전에 해당 부분을 마무리를 하면 된다.

보완책1 :
하루 15분.(너무 길게 하면 안될듯)
각자 작성한 코드에 대한 코드리뷰 시간을 가지고,
나중에 볼 수 있도록 영상 기록으로 남기기
코드에서 패치노트 같은 것임. 이제 이 기능 만들어졌으니까 이렇게 쓰면 되고 ~~

보완책2 : **한달에 한번. 대대적인 주석 공사.**
먼저 코드 쓸때 주석을 쓰는 습관을 가지는 것이 사실 먼저이지만,
(그런 습관을 만들어 내야 함.)
습관은 한번에 바뀌기 쉽지 않다. 잘 되지 않을 가능성이 높고,
그래서 달에 한번씩 주석 공사를 하자.
모든 함수나 클래스, 변수들의 역할과 하는 일등을 주석으로 남기는 일.
함수의 경우 주석을 남기고 실제로 그렇게 작동하는지 단위테스트를 진행한다.

보완책3 : 주석법칙을 만든다.
주석을 쓸때 지켜야할 주석의 법칙을 만든다.

보완책4 : 전체 흐름도를 그려보자. (High Level Design)
개발할때 옆에 띄워두거나, 프린트 해두면 많은 도움이 될 것이다.
무엇이 어디에서 어떻게 처리되는지 한눈에 볼 수 있게 하자.

1-1-3 주석의 법칙 1p

[불확실] 주석은 **한글로 일단 써보자**. Github가 잘 불러오는지 봐야 할 것 같다.
근데 Github 사이트에서 볼때나 다른 환경에서 돌릴때는 깨지는 현상이 있는 듯?

모든 <코드작성의 마무리> 에는, 주석 수정을 해야 한다.
주석은 접혀있는 상태가 기본상태이다. (볼 필요 없다면 숨겨라.)

1. AI가 쓴 코드가 코드내에 들어간 경우
//AI Code Start : <AI 이름>
<AI가 쓴 코드>
//AI Code End : <AI 이름>

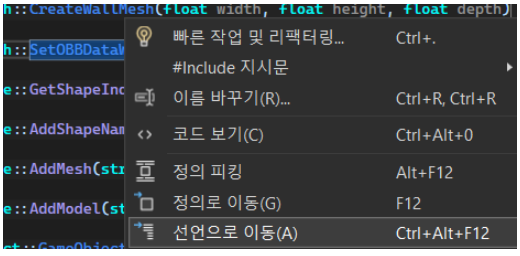
2. AI 코드가 실제 코드에 들어가진 않았지만,
핵심 아이디어가 AI가 관여한 코드의 경우
//AI involved Code Start : <AI 이름>
//AI involved Code End : <AI 이름>

>>왜 AI가 쓴 코드를 표시해야 하는가?
그것은 그 코드를 적은 사람이 해당 코드를 이해하지 못할 우려가 있기 때문이다. 쓴 사람이 그것을 이해할 수 없다면, 그것에 영향을 주는 혹은 영향을 받는 다른 부분을 짜는 작업자가 해당 코드를 이해해야 하는데, 결국 그 작업을 할 수 없다.
AI가 짰다는 사실을 알 수 있다면,
해당 AI 대화 문맥에 들어가 맥락을 물어볼 수는 있다.

3. 아직 단위 테스트를 하지 않아 제대로 작동하는지 알 수 없는 코드
//sus

4. 교수님에게 질문이 필요한 코드 / 혹은 추가 이해가 필요한 경우
//Question <질문내용>

1-1-3 주석의 법칙 2p



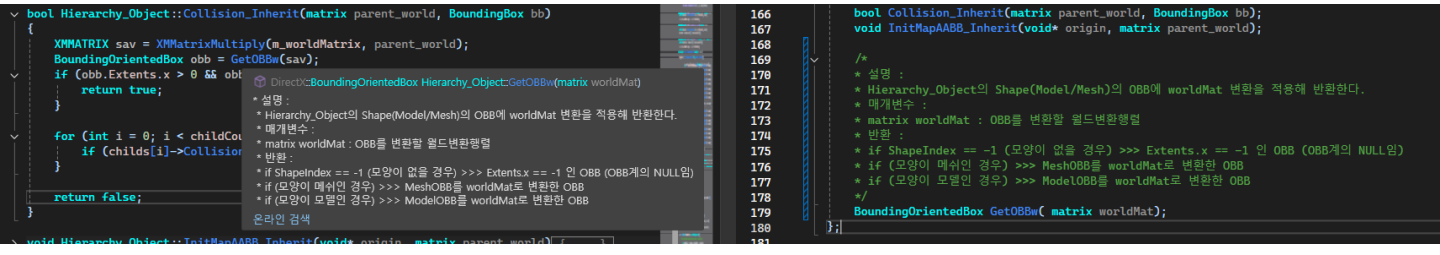
5. 함수주석 남기기
함수의 주석은 함수의 선언에 남긴다.
참조된 함수를 우클릭하여 <선언으로 이동> 을 누르면 해당 함수의 주석을 바로 볼 수 있다.

약속

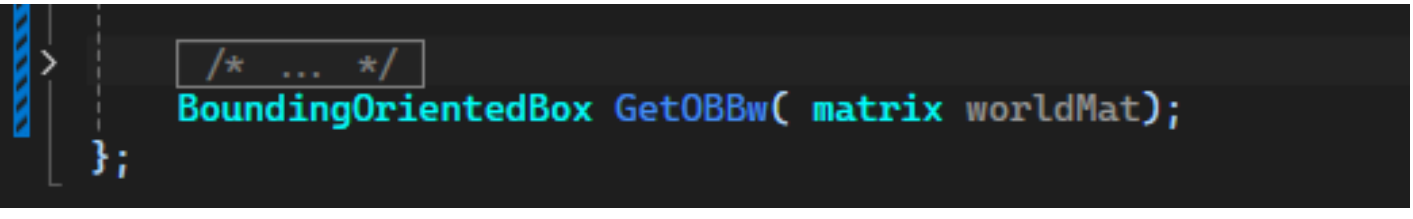
```
/*
<함수 목적, 설명>
<첫번째 매개변수> : <첫번째 매개변수 설명>
<두번째 매개변수> : <두번째 매개변수 설명>
...
<n번째 매개변수> : <n번째 매개변수 설명>

return
if <조건> -> <리턴하는 값을 설명>
if <조건> -> <리턴하는 값을 설명>
*/
```

예시
왼쪽 : 해당 함수를 사용하는 쪽에서 주석을 읽을 수 있음.
오른쪽 : 함수 선언에서 주석을 작성.



주석을 접을 수도 있다.



보면 매개변수 worldMat을 검사를 하지는 않는다는 걸 볼 수 있다.
>> 그럼 worldMat이 이상한게 들어간 경우 검출을 못하겠네? 라는걸 알 수 있고,
코드 작성시 이를 고려해 worldMat을 잘 들어가게 한다거나,
아님 함수를 수정해 검사하게 만들고 주석을 수정할 수 있다.

1-1-3 주석의 법칙 3p

5. 함수주석 남기기
다만 어떤 함수는 실행되는 알고리즘이 이해하기 너무나 복잡한 경우가 있다.
그럴 경우, 해당 주석의 설명은 문서화를 통해 따로 빼내는 작업이 필요하다.

해당 함수를 설명하는 문서는 따로 빼내어 Github의 함수설명문서 폴더에 pdf의 형태로 (링크만 따라가면 바로 볼 수 있도록) 저장한다.
그리고 함수의 주석은 기존의 양식을 그대로 쓰되, 자세한 설명에 해당 링크를 붙여넣는다.

예시 :

```
/*
* 설명 :
* Hierarchy_Object의 Shape(Model/Mesh)의 OBB에 worldMat 변환을 적용해 반환한다.
* 매개변수 :
* matrix worldMat : OBB를 변환할 월드변환행렬
* 반환 :
* if ShapeIndex == -1 (모양이 없을 경우) >>> Extents.x == -1 인 OBB (OBB계의 NULL임)
* if (모양이 메쉬인 경우) >>> MeshOBB를 worldMat로 변환한 OBB
* if (모양이 모델인 경우) >>> ModelOBB를 worldMat로 변환한 OBB
* 자세한 사항 : https://github.com/nohc2001/PulseCity/blob/main/%EC%9E%91%EC%97%85%EC%
  9D%BC%EC%A7%80/%EC%B5%9C%EC%A2%85%20%EA%B8%B0%ED%9A%8D%EC%84%9C.pdf
*/
BoundingOrientedBox GetOBBw( matrix worldMat);
```

다음과 같이 작성하면 Ctrl +클릭으로 해당 링크를 들어가 pdf를 직접 볼 수 있다.

1-1-3 주석의 법칙 4p

6. 변수의 주석
변수 또한 주석이 필요하다. 코드를 작성할때 이 변수가 무엇을 하는 변수인지 명확하게 알 수 있도록 하는 과정이 필요하다.

변수의 주석은 선언 앞에 위치한다.
// <설명>
<변수선언문>
예시

7. 구조체/클래스의 주석
구조체 정의 앞에 주석을 추가한다.
형식은 이렇다.
/*
설명 :
<설명>
Sentinel Value (Optional)
<Sentinel Value> : (<조건>)
***/**

```
/*
* 설명 :
* GPU에 올렸거나 올릴 리소스의 디스크립터 핸들을 저장한다.
* Sentinel Value
* NULL = (hcpu == 0 && hgpu == 0)
*/
struct DescHandle {
    D3D12_CPU_DESCRIPTOR_HANDLE hcpu;
    D3D12_GPU_DESCRIPTOR_HANDLE hgpu;

    DescHandle(D3D12_CPU_DESCRIPTOR_HANDLE _hcpu, D3D12_GPU_DESCRIPTOR_HANDLE _hgpu) {
        hcpu = _hcpu;
        hgpu = _hgpu;
    }
    DescHandle() :
        hcpu{ 0 }, hgpu{ 0 }
    {
    }

    __forceinline void operator+=(unsigned long long inc) { ... }
};
```

1-1-3 주석의 법칙 5p

8. 가끔 주석을 기존 코드를 내용은 유지하지만, 실행에서 비활성화 하고 싶을때 주석을 사용하기도 한다.

문제는 이 코드들은 내용이 길어 코드에서 상당 부분을 차지하게 된다.
그러면 코드들간의 간격이 넓어져 가독성이 떨어진다.

이럴때는 다른 주석과 구분되게 이렇게 하자.

//temp <날짜> <왜 비활성화 했는지에 대한 힌트>
/*
<기존 코드>
*/
날짜를 적는 것이 나중에 코드를 작성할때 많은 도움이 될 것이다.
언제 그 코드가 비활성화 됐는지 몰라 지우기가 어려운데, 날짜가 오래 지났다면 지워도 좋다는 판단을 할 것이다.

정확한 시간까지 적는 건 무의미하다 판단된다.
24시간 내에는 그걸 비활성화를 왜 했는지 충분히 기억할 것이다.

추가적으로 비활성화의 이유도 힌트로 주면 좋다.
이걸 제외해도 되는지 판단하는데 도움이 된다.

9. 코드의 예정된 업그레이드 사항이나 향상될 사항이 있다면 이렇게 적자
//improve <수정예정사항>

예시

```
// improve : not just wchar, char also. function name is dbglogaN
> dbglogDefines
```

1-1-4 High Level Design

1-2 개선사항 : 기존 작성된 코드가 문제가 있는 경우

1-2-1. 기존 코드가 잘 작동할지 확신이 안되는 부분 사실 다시 짜야 하는 것이다.
해당 사항을 나열하자.
A* algorithm
Recv 잘리는 부분을 복구하는 알고리즘.

1-2 개선사항 : 기존 코드에 불필요한 부분이 있는 경우

해당 경우 코드가 쓸데 없이 길어서 이해가 어렵다.

2. 코드의 성능이 안나올때

2-1. 메쉬나 모델, 텍스쳐 등의 로딩이 비효율적이다.
그냥 한번 커맨드 리스트를 열고 로드 한번에 하고 하면 좋을 것 같은데?

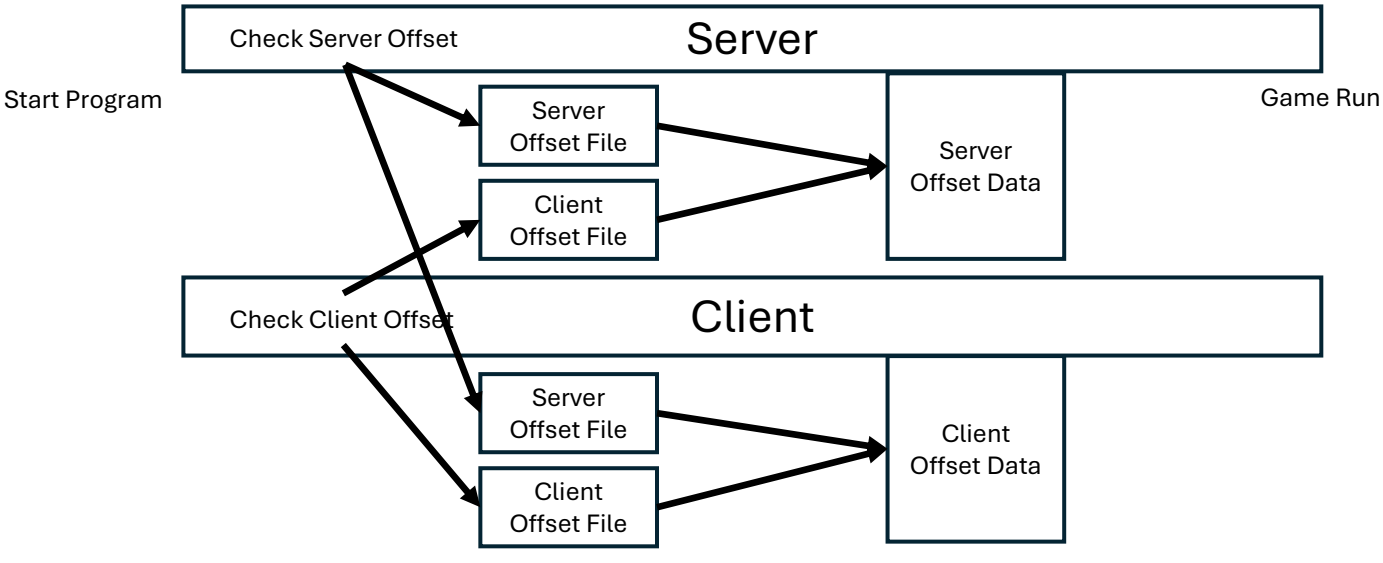
2-2. 너무 많은 물체를 그릴때 프레임이 내려감. 프러스텀 컬링으로 어느정도는 해결했지만 LOD, 인스턴싱을 적극 적용할 필요가 있다.

3-1 개선사항 : ServerOffset과 ClientOffset 수정의 어려움. 1p

Offset 동기화 시스템을 함으로써 얻으려 했던 장점 :
어떤 변수를 동기화하는지에 따라 다른 코드를 작성해야 하는 것은
너무 효율이 안좋다고 생각됨.
그래서 동기화를 원하는 변수의 주소만 전달하면, Offset을 뽑아
다른 호스트의 어떤 변수에 적용해야 할지 결정할 수 있다.

문제의 원인 :
서버와 클라이언트의 동기화에 있어서 Offset을 사용해
어떤 클래스의 맴버변수를 변경하면 나머지 Offset도 모두 변경됨.

수정 목표 :
서버와 클라이언트를 한번 실행했을때, 모든 Offset관련 설정을 자동으로
제어하는 파일을 자동으로 생성하고, 이후 서버와 클라이언트는 해당 파일을
읽어서 각자의 Offset 설정을 등록하고 사용한다.



해당 그림은 클라이언트와 서버가 한 호스트에서 개발자에 의해 수정될때를
가정하고 있다. (프로그래머는 그냥 맴버를 수정하고 실행만 하면 자동으로)

때문에 실제 게임으로써 실행되는 프로그램은 이미 서버와 클라이언트
동기화를 위한 오프셋은 이미 내장되어있는 상태로 내보내야 한다.

3-1 개선사항 : ServerOffset과 ClientOffset 수정의 어려움. 2p

그럼 그걸 어떻게 구현하는가?
dependency : 일단 c++ 17 이상의 버전이 필요하다.

1. SyncOffset.cpp 코드를 참조하여 각 클래스와 구조체,
   싱크가 필요한 맴버변수들을 설정한다.

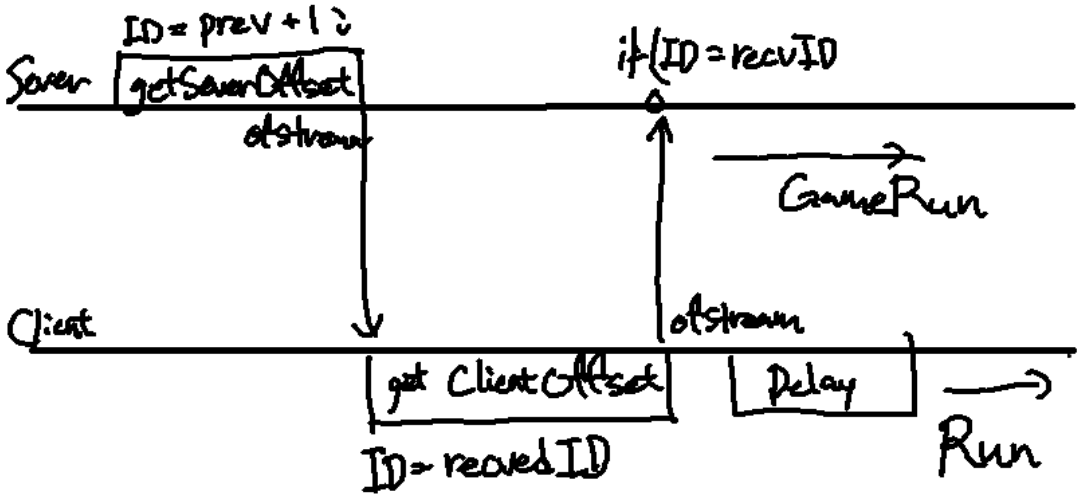
1-1 . 구조체가 정의를 시작하고 바로 다음 매크로를 설정한다.
```
#define CurrentStruct AAA
    STATICINIT_innerStruct;
```

1-2. 동기화가 필요한 맴버변수를 다음과 같은 매크로로 정의한다.
```
    syncDefine(int, a); // 일반 타입
    syncDefineArr(int, arr, 100); // 배열 타입
    syncDefineStdArr(double, stdarr, 50); // std array
```

1-3. 정의가 끝나면 다음과 같은 매크로를 설정한다.
```
    #undef CurrentStruct
```

이를 통해 각 타입의 동기화가 필요한 맴버변수들의 Offset을 바로바로 접근 가능하다.

그러면 ofstream을 통해 클라이언트, 서버쪽에 오프셋 파일을 출력한다.
일단 보통 서버가 먼저 실행되니 서버가 먼저 파일을 보낸다.
그리고 클라이언트가 그 파일을 받아서 분석하고, 서버의 Offset을 적용한다.
그러면 클라이언트가 Offset 파일을 구성해 서버로 전달한다.
서버는 현재 클라이언트의 Offset 파일의 ID가 서버의 Offset ID 와 같음을 확인하고,
같다면, 클라이언트의 Offset을 구성한 후, 게임을 실행한다.



3-2 개선사항 : 아직 고쳐지지 못한 오류

우리 개발 특징 :
아직 고쳐지지 못한 오류들이 있는데 그걸 무시하고 시간에 쫓겨
개발을 시작함.
먼저 고쳐야할 오류들이 쌓여 있다.
그게 충돌할 수 있다.

오류 목록 :
서버가 Release Mode 면 실행이 안됨

4-1 개선사항 : 프로젝트가 최종적으로 어떤 구조여야 하나? 1p

현재까지 생각나는 건 다음과 같음.

1. 클라이언트
    1-1. 그래픽 기본 기능 구현
        Graphics
            비동기 렌더링
            AO 계산 [x]
            Text Rendering
                SDF Text Texture, Mesh Text Rendering with Effects [구현됨]
                고정 텍스트 서브렌더 타겟으로 최적화
            Mesh LOD
            Tesselation Mesh Making Algorithm
        Texture
            DDS, 기본 밉맵(LOD), 기본 블럭압축포맷 정하기 [기술구현됨]
        Geometry Shader & Stream Output
            Stream Output 파티클 시스템 [구현후 CS와 성능 비교가 필요]
            풀/나무 잎 렌더링 [구현됨]
            트레일 효과
            물 표현
            실시간 충돌 디버그 시각화 [이거 해야되나?]
            차폐쿼리 활용?
        Tesselation
            터레인 LOD [구현됨]
            Displacement Map PBR 적용.
        다중 패스
            ShadowMapping [구현됨]
                - Cascading
                - Point[x] / SpotLight[구현됨]
                - Baking
            Light Probe
            Plane Mirror [구현됨]
        Compute Shader
            모션블러 [구현됨]
            포스트 프로세싱
            Compute Shader 파티클 시스템 [구현후 Stream Output과 성능비교 필요.]
        Animation
            키 프레임 애니메이션
            Loot Motion
            State Machine
            Blending
            IK
        Raytracing
            Raytracing 렌더링
            Raytracing 그림자
            Raytracing 반사
            Real Time Denoised Ambient Occulsion
            PBR with Raytracing

4-1 개선사항 : 프로젝트가 최종적으로 어떤 구조여야 하나? 2p

현재까지 생각나는 건 다음과 같음.

1. 클라이언트
    1-1. 그래픽 기본 기능 구현
        기본적으로 지켜져야 하는 것.
            1. F9를 누르면 전체화면 모드 전환이 잘 되어야 한다.
            2. 60fps가 보장되어야 하고, fps가 일정해야 한다.
    1-2. 메모리 관리
    1-3. DirectX 버전 확장성 관리
        - 구식의 버전의 DirectX, 컴퓨터 버전이어도 잘 실행될 수 있게 해야 함.
        - 그렇게 못하면 적어도 원인을 메시지 박스로 나타내서 유저가 알 수 있게 해야 한다.
    1-4. Model Exporter 를 모두의 컴퓨터에서 돌아가게 하기.
    1-5. 클라이언트 설치파일 만들기.
        - Visual C++ Redistributable 패키지 자동 실행하게 하기

4-1 개선사항 : 프로젝트가 최종적으로 어떤 구조여야 하나? 3p

1-2. 게임플레이

- **월드**
  - **맵 만들기**
- **UI**
  - **미니맵**
  - **HP/Shield**
  - **인벤토리**
  - **스탯창**
  - **스킬창**
- **스킬**
- **몬스터**
- **직업**
- **무기**
- **의뢰**
  - **메인의뢰 [던전]**
  - **일반의뢰 [스토리]**
    - **NPC와 대사**

2. 서버

테스트

안정성 : 서버가 게속 돌아도, 어떤 일이 생겨도 죽지 말아야 함.

확장성 : 서버를 얼마나 많이 설치가능한 프로그램인가?

- **NW 라이브러리 사용중지.**
  - **Release Mode, 바이너리로 정상 실행**
  - **언제라도 Naggle 알고리즘을 킬 수 있도록 하기**
  - **논블럭킹 / IOCP를 사용한 대규모 접속 처리.**
  - **멀티스레딩 서버**
  - **여러 개의 서버와 Seemless Zone**
  - **유저가 클라이언트를 업데이트할때, 서버 IP와 포트 데이터를 파일로 업데이트**
- **메모리 관리**
- **GPU 사용 서버?**
- **데이터베이스와 연결**
- **동점 테스트하여 어느정도 성능이 나오는지 측정하기**
- **보안**
  - **DDos 공격방어**
  - **유저 정보·결제 데이터 암호화 저장 및 전송**
  - **실시간 로그 모니터링 및 이상 탐지**

PIX GPU 디버깅

렌더링 하다보면 그래픽 파이프라인 전체와 셰이더에 어떤 값이 들어가는지 알고 싶을 때가 많음.
PIX를 사용해 디버깅을 진행해야 한다.