2020182031 이서연 졸업작품 12주차 진행

1. Object ID개편 시도

현재 Object ID는 Object Shader에서 Constant Buffer로 Object하나 당 1씩 값을 올려서 넘겨주고 있다.

문들 이런 방법이 아니라 Shader코드에서 이를 처리할 수 있지 않을까? 하는 마음에 사용하던 Object ID를 지워 두고 SV\_InstanceID를 사용하여 Vertex Shader에서 Instance값을 넘겨주어 보았다.

내 머릿속에서 이 값은 각 객체마다 1씩 값이 올라가면서 전달되기에 내 생각대로 구현이 될 것이라고 생각했으나 결과는 달랐다.

텍스트이(가) 표시된 사진

자동 생성된 설명가장 바깥 테두리에만 외곽선이 생김

왜 이렇게 다른 결과가 나올까? 생각해봤는데 아무래도 이 육면체들이 모두 같은 모델을 사용하는 Instancing 객체들이기 때문인 것 같다.

육면체 Vertex를 생성할 때 한번 Instance ID가 늘어나고, 그 이후 같은 값을 계속 사용하므로 하나의 Instance ID를 가지고 있다고 판단이 되어, 이 방법을 사용하지 않기로 하였다.

1. Debug Shader

저번 주에 Mipmap관련하여 수정을 한 부분의 설명이 부족한 것 같아 추가설명을 적어보려 한다.

Texture의 크기가 1:1에서 1:2인 절반이 될 때, LOD가 1 오르게 된다.

즉, 1:1 ~ 1:2 사이의 크기에 있으면 LOD 0과 LOD 1결과를 비율적으로 계산하여 결과가 나오게 되는데, 나 같은 경우 그래서 Vertex 범위를 -1~+1에서 -1~0으로 줄일수록 점점 화면이 어두워지고, 0부터는 아예 검정색이 나오는 것을 확인할 수 있었다.

그 이유는 Mipmap이 어떻게 설정이 되는지는 모르겠지만, LOD 1의 결과값이 들어가지 않고 있기 때문에 검정색 값이 섞여 들여가 비율에 따라 점점 어두워지는 것 이였다.

이런 Mipmap을 왜 굳이 사용하나 생각이 들었지만 멀리 있는 물체의 해상도가 선명하면 노이즈가 발생하는 등의 Mipmap관련 문제들이 존재한다는 것을 기억해냈다.

아무튼 이렇게 문제를 수정했으니, 이제 Debug Shader를 진짜로 만들어보려 한다.

일단 먼저 내가 선택한 방법은 간단한 방법이었다.

Screen Shader에서 화면을 그릴 때 Draw Option이라는 값을 바꾸어 주면서 그 값에 따라 그려지는 화면이 다르게 해 줄 생각이었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이게 바꾼 Render함수이고 Update Shader Variables의 두번째 인자는 Draw Option의 값이다.

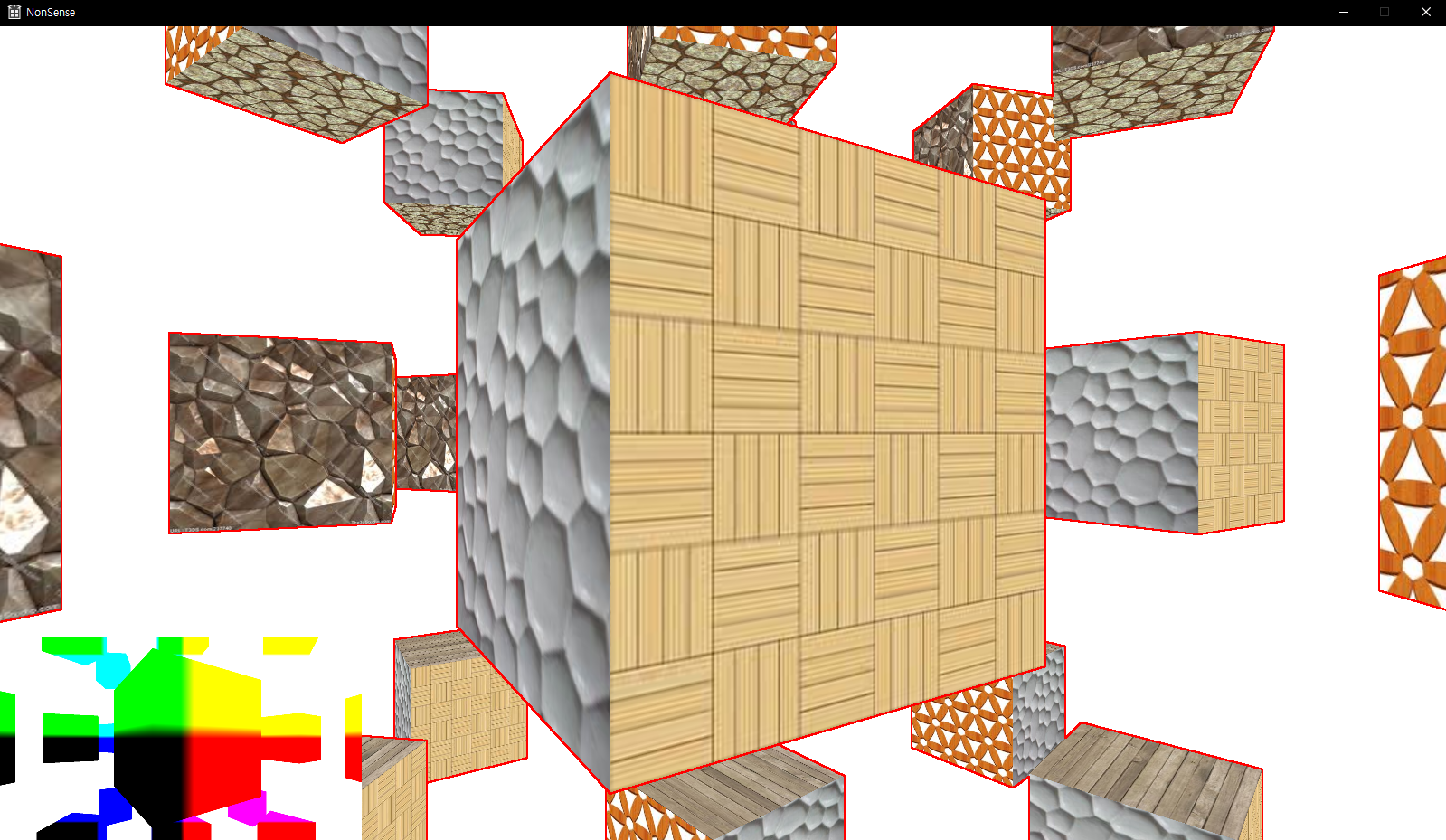
그러나 내 예상과 결과는 달랐다.

전의 Screen들은 모두 사라지고 Draw Option의 값이 MRT의 마지막 번호가 될 때의 Screen만 화면에 뜨는 것 이였다

난 Draw Instanced를 하면 Draw Option 0 ~ MRT까지 차곡차곡 쌓아서 그려질 줄 알았는데 그게 아니라 그냥 가장 마지막에 그린 화면만 그려지고 있었다.

그래서 전에 만들어 뒀던 Debug Shader를 다시 가지고 왔다.

원래는 Texture가 제대로 출력되지 않아 사용을 못하고 있었는데 오늘 Render함수를 상속받지 않고 따로 만들어주니 갑자기 동작이 잘 되기 시작했다.

Position을 출력하는 모습이다.

문제는 이 Debug Shader를 여러 개 출력하는 것이었는데 원래 내 생각은 Shader Variable을 그릴 때 마다 Version을 바꿔서 Shader Code로 넘겨지는 방법을 생각했으나, 쓸데없는 데이터 전달인 것 같다고 생각이 들어 Shader Code내에서 해결이 되지 않을까? 하고 고민을 해보았다.

텍스트, 야외, 가까이이(가) 표시된 사진

자동 생성된 설명

현재 Vertex Shader의 상태인데 이 부분을 보고 힌트를 얻을 수 있었다.

nVertexID는 SV\_VertexID, 즉 Vertex가 들어올 때마다 1씩 증가하는 수인데 Vertex Shader에 따로 Input을 해주지 않고 Draw Instance로 6개의 점을 넣어 Vertex Shader에서 수동으로 값을 넣어주고 있었는데 여기서 Vertex를 더 넣어줘서 그려줄 수 있지 않을까? 하는 생각이 들어 수정해 보았다.

일단 Debug Shader의 Render함수를 다음과 같이 바꾸어 주었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 Vertex Shader를 다음과 같이 바꾸었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이게 무슨 코드인가? 싶을 수 있지만 사실 0~24 모든 Vertex의 값을 넣어주었다가 간략화 한 것이다.

처음부터 설명해 보자면 int s는 Vertex ID를 6으로 나눈 값, 즉 몇 번째 화면인지를 알려준다.

그리고 Vertex ID를 6으로 나눈 나머지는 매 화면마다의 몇 번째 Vertex인지를 알려주는데 결국 각 화면의 중점 기준 위치는 같으므로 이를 계산해서 s와 함께 계산해 주었다 (4개의 화면이라고 생각하고 계산)

그리고 output에 num이라는 변수를 추가해주었는데 이는 Pixel Shader에서 어떤 Texture를 불러줄 지 알려주기 위해서 넘겨주었다.

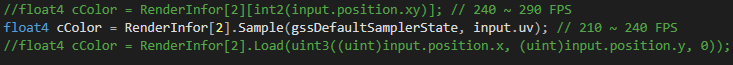
텍스트이(가) 표시된 사진

자동 생성된 설명

이것이 Pixel Shader인데 3번만 따로 빼 둔 이유는 3번이 Depth인데 Depth를 Sampling하여 그대로 출력하면 빨간 배경이 되어 너무 눈이 아파서 r값만 빼 와서 출력해 주었다.

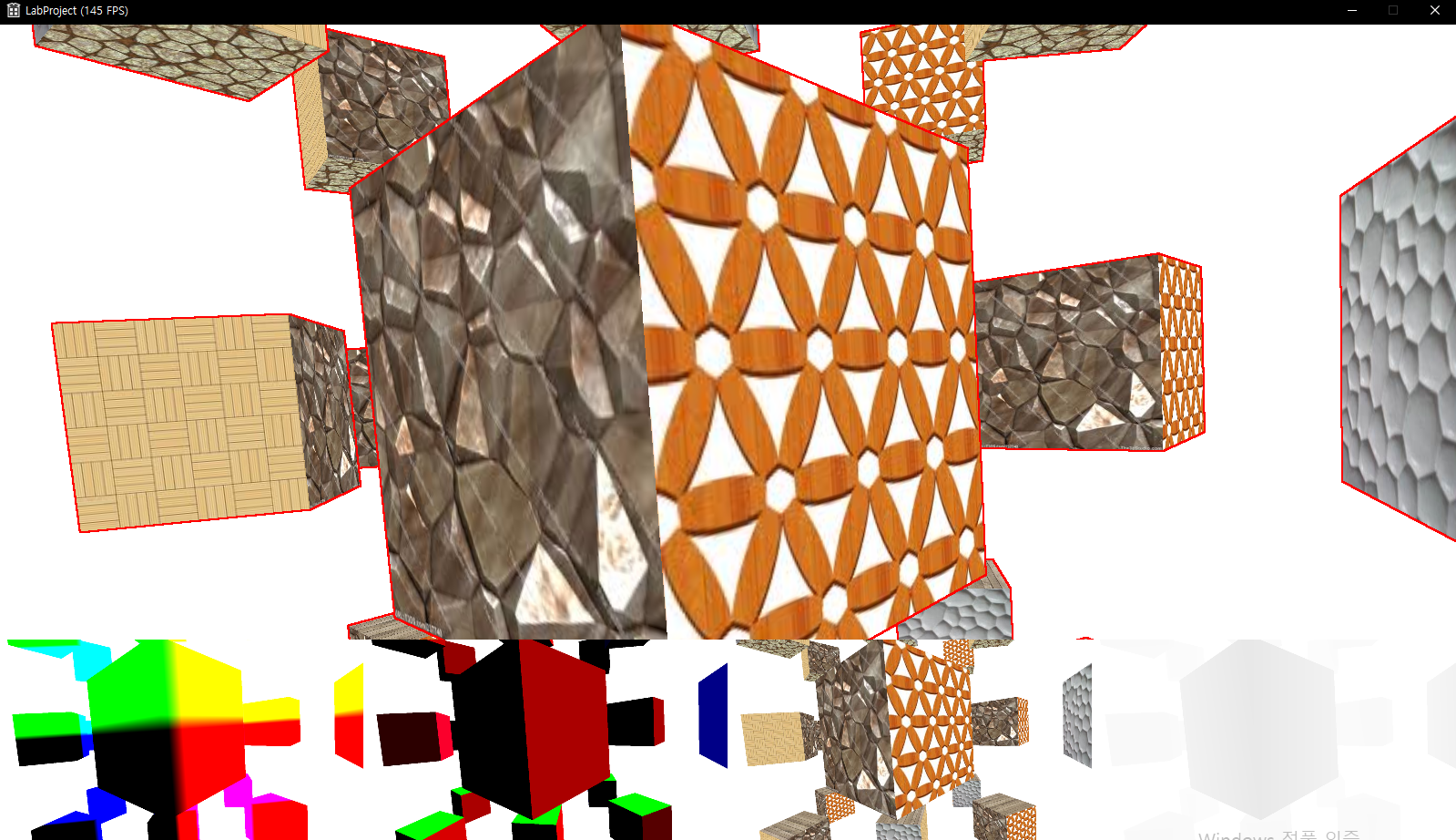
추가적으로 Texture를 Load하는 방법에 대하여 생각해보았다.

Texture를 불러올 때 Sample함수를 사용하다가 문득 어차피 Pixel Shader는 화면 픽셀 수만큼 불릴 것인데 굳이 Sample로 모든 Texture를 Load할 필요가 있을까? 싶어서 Road함수로 한 픽셀의 값을 불러오는 방법과 2차배열로 직접 Texture값을 가져오는 방법을 생각했다.



다음과 같이 FPS를 확인하며 불러보았는데 매 실행마다 서로 다른 결과가 나오는 것을 확인하고 미미한 차이라서 알 수 없구나 하는 생각이 들었다.

머릿속 이론상으로 Sample함수는 확실히 부하가 심할 것 같은데 Load함수와 직접 호출 중엔 뭐가 더 빠를지 감이 안 오는데 찾아보니 직접 호출은 Load2함수라는 것을 사용하는데 큰 차이가 없다고 한다.

이것이 완성된 Debug Shader

아무튼 결론적으로 이렇게 Debug Shader를 완성했다! 추가적으로 키를 설정해주어 F1을 눌러서 Debug Mode를 On/Off 할 수 있도록 설정해 주었다.