|  |  |  |  |
| --- | --- | --- | --- |
| MORPHOSIS 개발일지 23차 | | | |
| 기간 | 2019-10-08 ~ 2019-10-22 | 작성자 | 신재욱 |
| 작업 내용 | | | |
| 작업일지 요약   * GPU HLSL에서 사용할 리소스를 어떤 양식으로 올릴 건지 적어두는 그래픽 루트 시그니처를 이후 내용 작성을 위해 수정. * 그림자(Shadow Map 방식) 렌더링을 위해 필요한 내용들 확인   + 텍스처에 렌더링 하기(Shadow Map)   + 그 텍스처를 내 화면에 투영하여 광원과 내 카메라에서 렌더링한 픽셀의 거리 구하기   + 거리를 비교하여 그림자 안인지, 밖인지 구분하기 * 텍스처에 렌더링하기(Render To Texture)를 하기 위해선 컴퓨트 쉐이더를 사용할 수 있어야 함.   + 컴퓨트 쉐이더는 쉽게 말하면 GPU를 이용한 멀티스레드 계산 단계라고 볼 수 있음. * 컴퓨트 쉐이더를 사용하기 위해 필요한 코드들을 작성.   + 기존의 그래픽 루트 시그니처와 다른 컴퓨트 루트 시그니처 사용   + 기존의 그래픽 파이프라인 스테이트와 다른 컴퓨트 파이프라인 스테이트 사용   + GPU에서 멀티스레드 계산 이후 결과값을 다른 곳에 저장하고, 그 값을 다시 사용해서 한 번 더 렌더링 패스를 돌려야 하므로 이를 위해   Unordered Access View, UAV 를 만들고 사용해야 함.   * + 리소스 배리어를 통해 이를 어떤 렌더 타겟에 렌더링할 것인지, 리소스 카피를 할 것인지 등 자잘한 내용들 작성   + 추가로 필요한 디스크립터 힙 등을 생성.   Root Parameter에서는 텍스처를 담기 위해 Descriptor Table을 사용한다. (Descriptor == 서술자 == View, 다 같은 의미이다)  Descriptor Table은 Descriptor Range 정보가 필요하다. 간단하게 말하면 ‘이러이러하게 쓸거다’라고 적어둔 구조체.  Descriptor Range 에는 이게 쉐이더 자원인지(SRV), 상수 버퍼 자원인지(CBV) 아니면 뭐 UAV 자원인지 RangeType을 정해줘야 하고,  Descriptor는 몇 개를 쓸건지(쉽게 말하면 텍스처 배열 할당할 때 몇 개로 할당할건지), 레지스터 첫 숫자는 뭘로 할건지(t4, t5 이런거) 등을 정할 수 있다. (ResisterSpace 등은 거의 쓸 일 없어서 설명 안 함)    요게 RootParameter. 맨 아래 요소는 어떤 쉐이더에서만 접근가능하게 할건지 정하는 값. (최적화 가능성이 있다지만 잘 모르겠다. 체감도 안 되고)    요게 DescriptorRange. 4번 레지스터 주소에 올라가는 버퍼의 값은 텍스처를 넣을 예정이므로 텍스처로 간주하고 해석해주세요~ 하는 의미이다. 근데 4번 레지스터 주소에 서술자 배열을 두고 그 배열의 길이는 5로 하겠다는 것이므로 그렇게 해석하면 된다.  이렇게 되면 t4 레지스터에는 Texture2D gtxtTexture[5]가 있을 예정이고, **실제로 사용해보니 gtxtTexture[0]으로 첫 번째 텍스처, gtxtTexture[1]로 두 번째 텍스처가 샘플러에 잘 들어오는걸 볼 수 있었다.** <- 이것이 궁금했던 부분.  근데 웃긴건 나는 텍스처들을 로드하고 그 텍스처들의 리소스뷰(==Descriptor)를 만들어주긴 했는데 실제 Render를 할 때는 첫 번째 텍스처만 SetGraphicsRootDescriptorTable() 해줬다는 점이다.  별다른 무언가 없이도 첫 텍스처의 다음 텍스처가 잡혀서 올라왔다는 건데..    텍스처 리소스 서술자를 만드는 부분이다.   * 지금까지 HANDLE 자료형을 그저 포인터 자료형 정도로 알고 있었는데 살짝 달랐다. 자세한 내용은   <https://blog.naver.com/PostView.nhn?blogId=tipsware&logNo=221065382244> 를 참고.  쉐이더 리소스 뷰의 CPU 서술자 주소와 GPU 서술자 주소(GPU와 연결고리)를 받아온다.  텍스처 타입을 저장한다.  다이렉트 X에서 사용할 수 있게 만들어진 리소스의 주소를 받아온다.  리소스의 정보와 쉐이더 리소스 뷰의 정보를 받아온다.  그걸 바탕으로 다이렉트X 디바이스에서 쉐이더 리소스 뷰를 생성하게 한다.  쉐이더 리소스 뷰의 CPU 서술자 시작 주소와 GPU 서술자 시작 주소를 서술자 사이즈만큼 증가시킨다.  텍스처 클래스의 루트 파라미터 인덱스와 GPU 서술자 주소를 저장한다.   * SetRootArgument() 는 해당 텍스처 리소스를 사용하기 위해 업로드 할 때 루트 파라미터의 몇 번째인지, 자신의 쉐이더 리소스 뷰 GPU 서술자 주소는 어디인지를 텍스처 클래스에 저장해주는 함수이다. 이를 통해 추후에 루트 디스크립터 테이블을 정할 때 사용할 수 있다.     일단 이를 통해 텍스처 리소스 디스크립터들이 CPU든, GPU든 일정한 Offset을 두고 저장됨을 알 수 있다. Offset으로 건너뛰면서 접근하면 다음 인덱스의 값에도 접근할 수는 있어보인다.  아니 그럼! Descriptor Range를 따로 두고 Descriptor를 하나씩 줘도 되지 않을까?!      이렇게 변경을 해보았으나 생각해보니 SetGraphicsRootDescriptorTable 에서 루트 파라미터 인덱스로 4를 넣으면  그럼 DescriptorRange 두 번째에는 어떻게 하지?? <- 이걸 해결 못 해서 포기했다. 돌려 돌려 롤백~  기존 방식대로 하면 인덱스 터지고 할 가능성이 있다. 루트 파라미터를 추가해주자.        디퓨즈 맵과 노멀맵의 루트 파라미터를 구분한 모습이다.  위와 같이 변경하고 잘 실행되는 것을 확인하였다.  이제 UAV 를 만들 준비가 됐다.  먼저 HLSL에 RWTexture2D 자료형을 추가해준다.    앞의 RW는 read-write를 의미한다. 컴퓨트 쉐이더에서 계산한 결과물을 저기에 쓸 것.  그 다음에는 unordered access view, UAV를 만들어야 한다. 그러나 그 전에 텍스처 자원을 만들어야 하는데  기존 방식과 달라지는 부분이 있다.    이게 원래 텍스처를 읽어올 때 텍스처 리소스를 만드는 방식이고    이게 UAV 용으로 쓰기 위한 텍스처 리소스 생성 방식이다. 위에서 <float4> 라고 한건 저기 Format이 R8G8B8A8\_UNORM이기 때문.  CreateCommittedResource()가 실패한다. 뭔가 인자가 잘못 들어간 모양.    Width랑 Height는 클라이언트 값이 맞는데  책 보고 해봤는데도 답이 없다! 검색해보자!    업로드 힙으로 생성했더니 아래와 같은 오류가 생긴다.  책에서도 디폴트 힙으로 생성하고 있는데 왜 업로드 힙으로 했지?  디폴트 힙으로 하면 문제 없음.    마지막에 생성한 pd3dBuffer가 텍스처 자원이 된다. 저것에 대한 SRV와 UAV를 생성해야 한다.  다른 곳에서 SRV를 언제 생성하는가? CreateTextureResourceView()에서 쓴다.  해당 함수에서 ID3D12Resource \*pd3dBuffer의 역할은 CTexture가 가지고 있는 리소스 자원이 맡는다.  일단은 임시로 생성해주자..    UAV 추가 했으니까 힙 하나 추가해줘야 함. 안 그러면 힙손상 에러가 다시 날 것.    M\_d3dUavCPUDescriptorStartHandle을 추가해줌. 이제 루트 파라미터를 건드려보자.      일단은 잘 몰라서 ShaderVisibility를 All로 했다. 이제는 Dispatch를 해줄 차례.  아니 루트시그니처를 두 개 만들어서 하네  책의 예제파일과 인터넷에 올라와있는 코드들을 보고 어떻게든 위치 변경    아 드디어~~~~~  문제 수정!  노멀 텍스처가 업로드 되지 않던 문제  - 서술자를 + 1에서 + 2로 변경하고 m\_d3dSrvCPUDescriptorStartHandle 를 올려줌  PS에서 Depth값을 얻어와서 그리는걸 해보자.      픽셀과 카메라의 거리를 구해서 반영.  그림자를 만들려면 일단 여러 가지 선행되야 하는 것이 있는데   * 텍스처에 렌더링을 할 수 있어야 함. * 텍스처 투영을 할 수 있어야 함. * 위를 바탕으로 섀도우 맵을 만들고 텍스처 투영을 할 수 있어야 함.   위의 세 가지이다.  텍스처에 렌더링 하는 것(Render To Texture, RTT)는 컴퓨트 쉐이더를 사용하여 텍스처 리소스에 렌더링을 하는 것을 의미한다. 컴퓨트 쉐이더를 사용하기 위해서는 먼저 기존의 PSO와 다른 Compute PSO를 사용해야 한다. 그것부터 만들어주자.    굿.  이제는 입력 텍스처 자원과 출력 텍스처 자원을 써야 한다. 입력은 기존 텍스처 사용방식과 동일하게 SRV로 생성해주면 된다. 출력은 조금 다른데 GPU에서 읽고 쓸 수 있어야 하기 때문에 읽기-쓰기를 지원한다는 의미의 ‘RW’가 붙는 자료형을 쓴다.  위의 자원들을 사용하려면 루트 파라미터를 수정해줘야 한다. | | | |