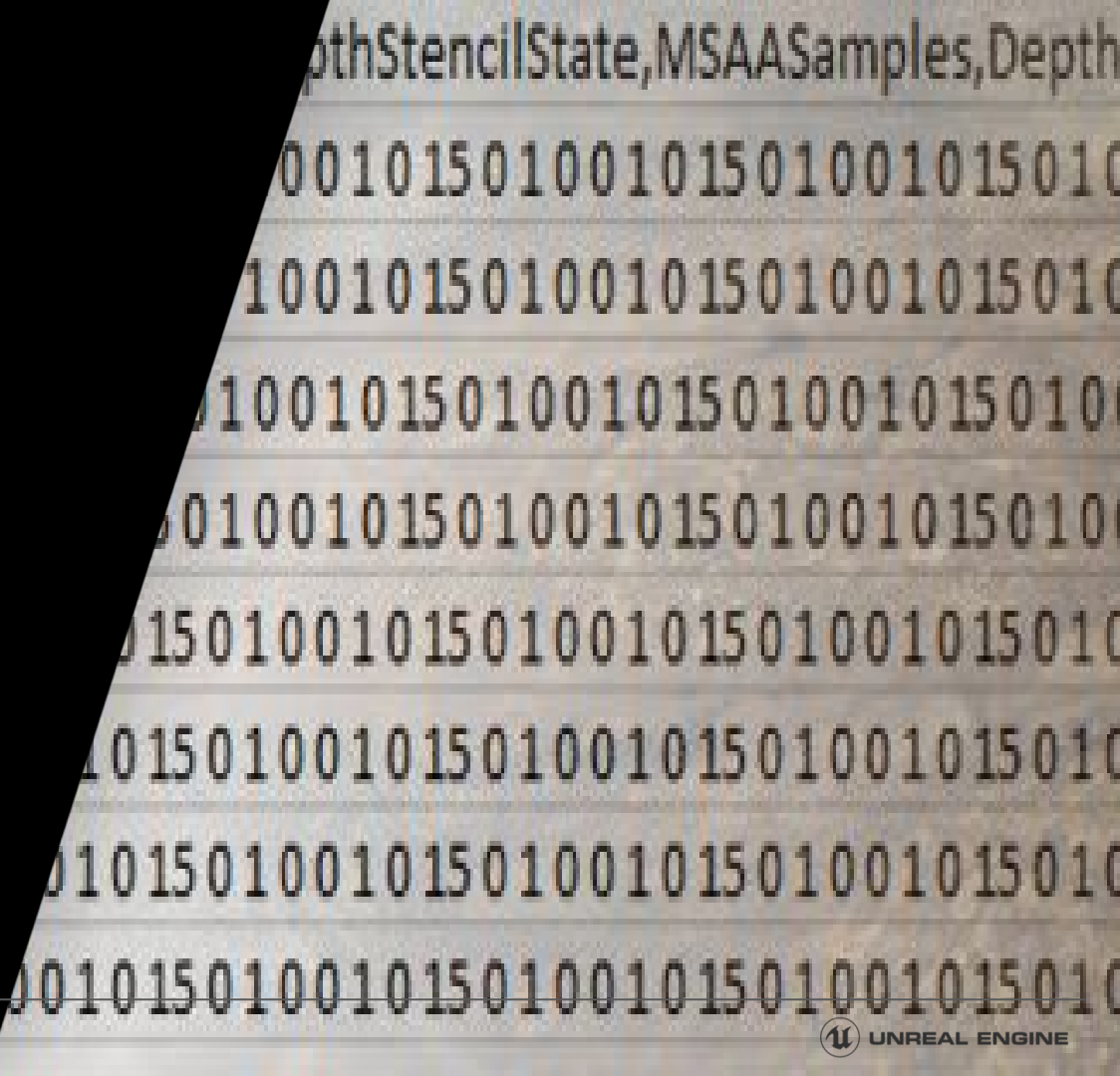
셰이더 파이프라인 캐시 실전 사용법

엔진 서포트 엔지니어 박창현

목차

- 파이프라인 스테이트 오브젝트
- 로우 레벨 캐시
- 셰이더 파이프라인 캐시
- PSO 캐시 사용 가이드



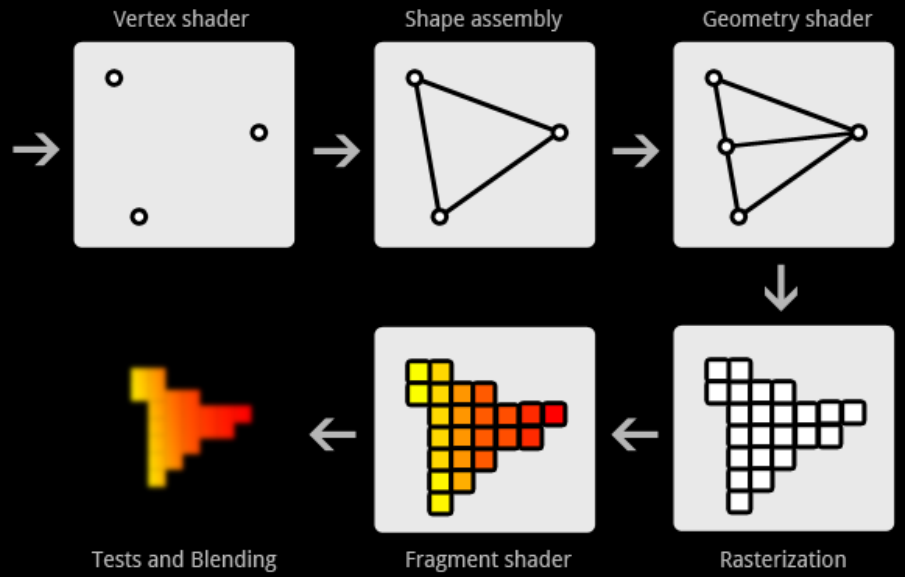
파이프라인 스테이트 오브젝트

파이프라인?

그래픽스 파이프라인

- 데이터를 이미지로 만드는 일련의 과정
  - 입력: Virtual camera, Objects(vertices / material / etc), Light sources, Textures
  - 출력: Framebuffer의 픽셀별 색

- 그래픽스 하드웨어 지원
  - 그래픽스 API : DirectX, OpenGL (ES), Metal, Vulkan



이미지 출처: https://open.gl

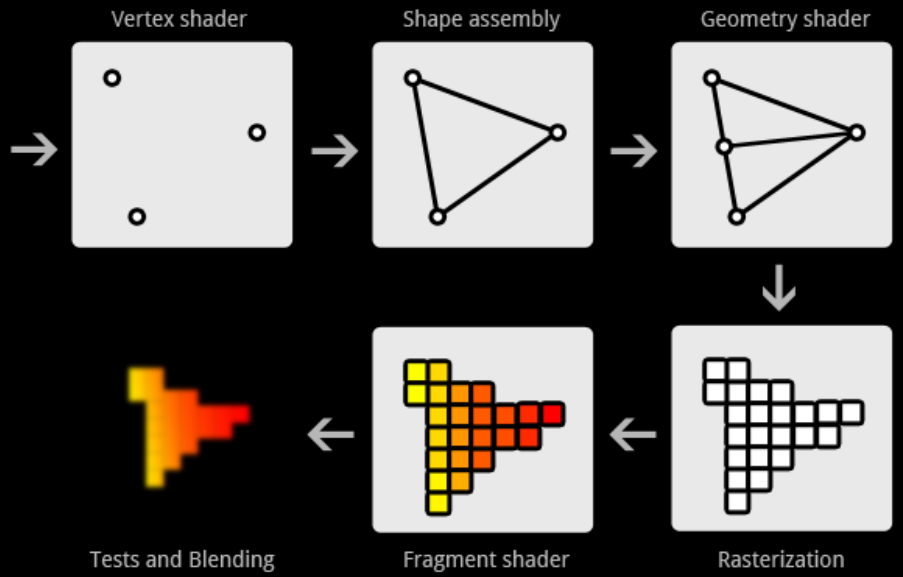
파이프라인?

그래픽스 파이프라인

- Programmable
  - Vertex / Geometry / Fragment Shader 등

Compute 파이프라인..?

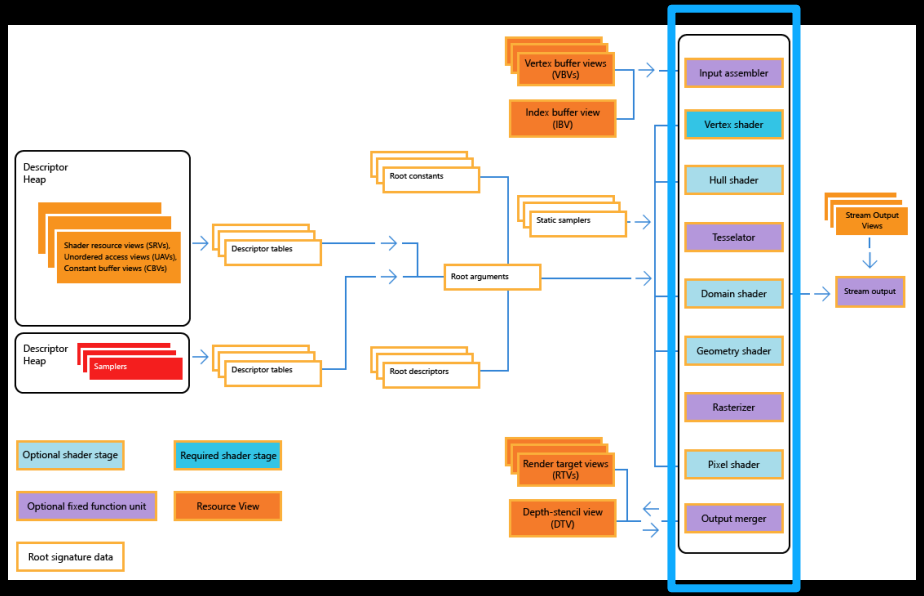
- 데이터를 계산 후 반환하는 일련의 과정
- 그래픽스 하드웨어 지원
- Programmable Compute Shader



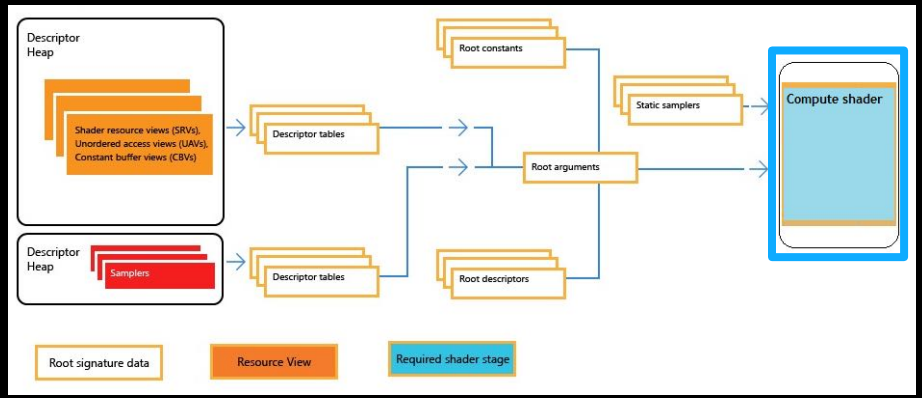
이미지 출저: https://open.gl

파이프라인?

D3D12 Graphics Pipeline



D3D12 Compute Pipeline



https://docs.microsoft.com/en-us/windows/desktop/direct3d12/pipelines-and-shaders-with-directx-12

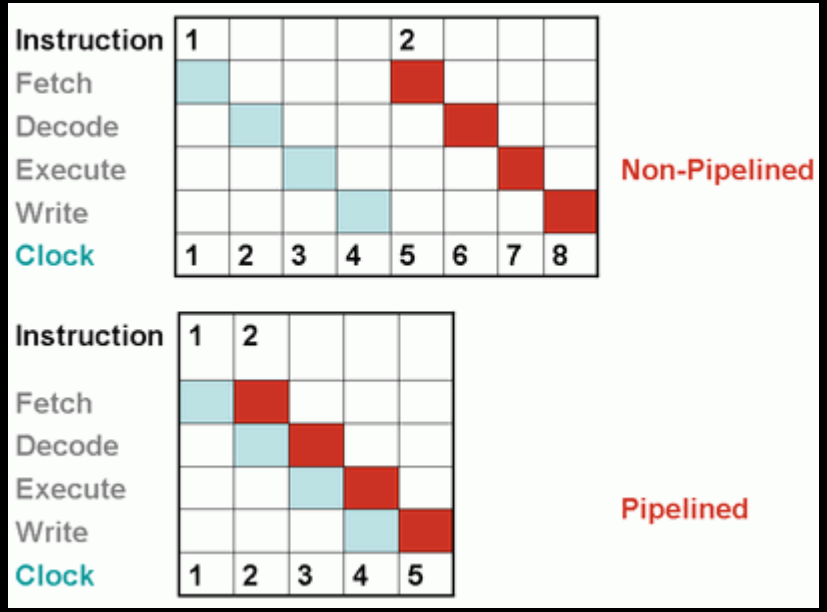
파이프라인?

그래픽스 하드웨어 지원

- 각 단계 Stage별로 최적화된 하드웨어 유닛 할당
- Stage 로 구분되어 있어 중첩 가능: 효율성 극대화

파이프라인 실행시, 조금씩 다른 동작을 하고 싶다면?

- Ex. ABCDE 파이프라인이 있을 경우
  - ㄱ: AB--E만 / ㄴ: A—DE만 / ㄷ: AB`C-E



CPU Instruction Pipeline - https://stackpointer.io/hardware/how-pipelining-improves-cpu-performance

스테이트?

파이프라인은 여러 Stage로 구성

- 각 Stage는 미리 세팅된 상태 정보(State)를 토대로 다른 동작

스테이트 State

- Ex. Blend State, DepthStencil State, Rasterizer State, Sampler State, …
- 파이프라인 사용시 스테이트 설정 필요.
  - State 변경 자체의 부하 있음.
  - State 변경을 하지 않으면? 이전 State를 그대로 사용. 성능 최적화에 활용.

스테이트?

과거에는 모든 스테이트들을 하나씩.

- Ex. D3D9 Render state: Alpha Blending State, Texture Stage State

개선: 연관된 몇가지 세팅을 한번에 할 수 있도록

- Ex. ID3D11BlendState: alpha-to-coverage, independent blending, render targets
- 연관된 다른 세팅도 함께 세팅하여, 스테이트 변경에 들이는 부하를 줄이자는 목적
- 렌더링 시점에 생성 및 세팅 가능

최신 하드웨어

- 하드웨어 유닛간 종속성 존재
- Ex. 하드웨어의 블렌드 설정시 Blend State외에 Raster State도 영향을 준다

파이프라인 스테이트

개념: 파이프라인 단위로 한.번.에. 스테이트를 세팅하자.

파이프라인 스테이트 Pipeline state

- 입력 데이터가 어떻게 해석되어 그려질지에 대한 하드웨어 설정
- 셰이더와 렌더 스테이트(Blend, Depth Stencil, Rasterizer, …)와 기타
- 파이프라인 스테이트 오브젝트 PSO를 통해 파이프라인 스테이트를 관리

파이프라인 스테이트 오브젝트

파이프라인 스테이트 정보를 담은 객체
Pipeline State Object == PSO

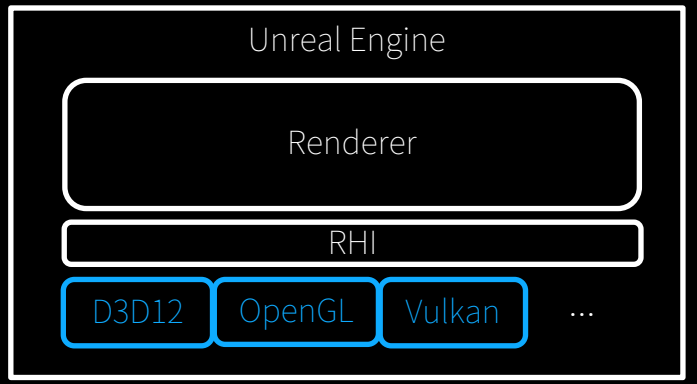
- 지원 그래픽스 API: Direct3D 12 / Vulkan / Metal
- 파이프라인 스테이트 관리에 활용.
- 미리 State의 유효성 판단을 해두어
- 렌더 타임에는 파이프라인 스테이트의 교체를 더 빠르게 할 수 있도록.

파이프라인 스테이트 오브젝트 PSO를 통해 대부분의 파이프라인 스테이트를 설정함

- PSO로 설정
  - 모든 셰이더 바이트 코드, Blend 스테이트, Rasterizer 스테이트, DepthStencil 스테이트, Multi-sampling 정보 등
- 커맨드 리스트로 설정
  - 리소스 바인딩, 뷰포트 정보, Blend 팩터, Scissor rects, DepthStencil 레퍼런스 값 등

로우 레벨 캐시

로우 레벨 캐시



D3D12 / Vulkan / Metal

- 런타임 생성된 PSO에 대해 캐시 지원

D3D12 / Vulkan

- 디스크로 PSO를 파일 아웃하여 로드 타임 PSO 생성

OpenGL

- ProgramBinary 지원 디바이스(OpenGL ES 3.0 이상)

RHI

- 플랫폼 전용 그래픽스 API 위의 얇은 레이어. 플랫폼 독립적인 코드로 모든 동작에 이상 없도록.
- 로우 레벨에서 생성된 PSO를 렌더 리소스 형태로 보관
- 이 PSO가 들어있는 Map 컨테이너를 활용해 캐시 검색에 관여

로우 레벨 캐시 – D3D12

런타임 캐시와 파일에서 로드된 캐시 동시 사용

- Runtime 캐시 = RHI로부터 “GraphicsPipelineStateInitializer”을 건내받아 검색
- Loaded 캐시 = Low Level Description 정보를 이용하여 검색
  - Low Level Description: 플랫폼 종속적인 파이프라인 스테이트 Descriptor
  - Ex. ShaderBytecodeHash, D3D12_SHADER_BYTECODE, D3D12_BLEND_DESC, D3D12_RASTERIZER_DESC, …

- 더 빠른 Runtime 캐시 부터 검색
  - RHI로부터 플랫폼 독립적인 파이프라인 정보(GraphicsPipelineStateInitializer)만 건내받음.
  - 플랫폼 종속적인 형태(Low Level Description)로 Translation하지 않고 바로 검색 시도.

로우 레벨 캐시 – Vulkan

D3D12와 동일 + Pipeline LRU Cache 지원

LRU Cache?

- LRU = Least Recently Used
- 최근 사용되지 않은 데이터를 캐시에서 내려, 새로운 데이터에 대한 캐시 공간 확보

Pipeline LRU Cache

- 로우 레벨 캐시에 LRU 지원
- 셰이더 메모리가 부족한 Android Vulkan 플랫폼의 경우, 아주 유용.

관련 설정

- #define VULKAN_ENABLE_LRU_CACHE 1
- r.Vulkan.EnablePipelineLRUCache = 1
- r.Vulkan.PipelineLRUSize = 10*1024*1024 / r.Vulkan.PipelineLRUCacheEvictBinary = 1

로우 레벨 캐시 – OpenGL

PSO를 지원하지 않음.

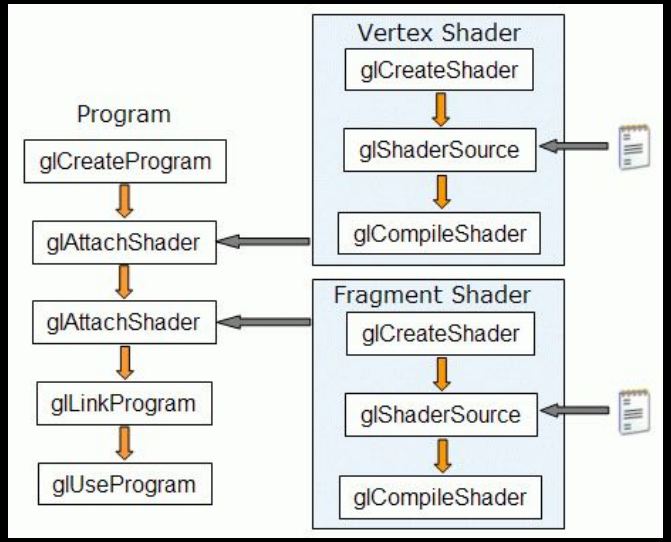
- 파이프라인 스테이트를 통한 일괄 변경이 아닌
- 셰이더 스테이트 + 렌더 스테이트 각각 갱신
  - 바인딩된 셰이더 스테이트(BoundShaderState, BSS)에 대해 로우 레벨 캐시 지원

```
virtual void RHISetGraphicsPipelineState(FGraphicsPipelineStateRHIParamRef GraphicsState)
{
    FRHIGraphicsPipelineStateFallBack* FallbackGraphicsState = static_cast<FRHIGraphicsPipelineStateFallBack*>(GraphicsState);

    auto& PsoInit = FallbackGraphicsState->Initializer;

    RHISetBoundShaderState(
        RHICreateBoundShaderState(
            PsoInit.BoundShaderState.VertexDeclarationRHI,
            PsoInit.BoundShaderState.VertexShaderRHI,
            PsoInit.BoundShaderState.HullShaderRHI,
            PsoInit.BoundShaderState.DomainShaderRHI,
            PsoInit.BoundShaderState.PixelShaderRHI,
            PsoInit.BoundShaderState.GeometryShaderRHI
            ).GetReference()
        );

    RHISetDepthStencilState(FallbackGraphicsState->Initializer.DepthStencilState, 0);
    RHISetRasterizerState(FallbackGraphicsState->Initializer.RasterizerState);
    RHISetBlendState(FallbackGraphicsState->Initializer.BlendState, FLinearColor(1.0f, 1.0f, 1.0f));
}
```

로우 레벨 캐시 – OpenGL

Program Binary Cache

- OpenGL은 셰이더들을 개별 컴파일 후 링크하여 Program Object 생성
- 프로그램 객체를 재컴파일 하지 않도록, 파일로 쓰고 나중에 로드하여 재사용할 수 있도록 하는 기능
- Separate Shader Object 지원



이미지 출저: http://cse.csusb.edu/tongyu/courses/cs520/notes/glsl.php

LRU 알고리즘 지원

- 셰이더 메모리가 부족한 OpenGL ES 플랫폼의 경우, 아주 유용.
- Mali의 경우, 드라이버가 허락한 최대 셰이더 메모리 힙 사이즈가 작다.

관련 세팅

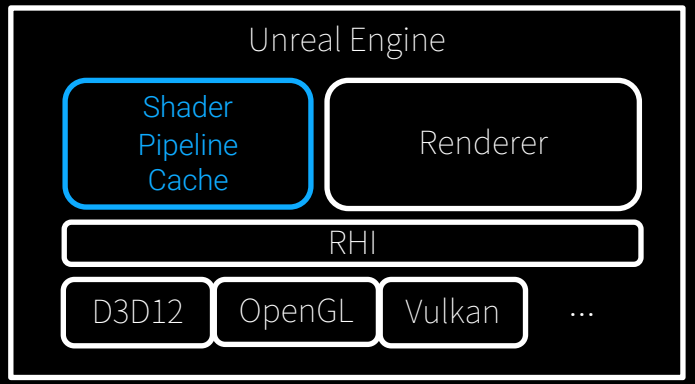
- r.ProgramBinaryCache.Enable=1
- r.OpenGL.EnableProgramLRUCache = 1
- r.OpenGL.ProgramLRUCount = 700 / r.OpenGL.ProgramLRUBinarySize = 35*1024*1024

셰이더 파이프라인 캐시

셰이더 파이프라인 캐시

셰이더 파이프라인 캐시

- RHI 레벨의 API를 활용해 로우 레벨 캐시 활용을 돕는 객체
- PSO 생성 시점/생성 방식에 초점



- PSO 캐시라고 부른다.

- 과거 존재하던 Shader Cache를 대체
- 목적: 처음 앱을 실행하는 유저“도” 런타임 히치 없이 플레이할 수 있도록 만들자.
  - 배포용 빌드에 PSO 생성에 필요한 데이터를 미리 세팅해두고
  - 배포용 빌드 실행시 해당 데이터를 통해 유저가 눈치채지 못하는 시점에 PSO를 컴파일

셰이더 파이프라인 캐시

PSO 캐시 동작 흐름

- 테스트 빌드로 런타임에 PSO 생성하기
  - 생성된 PSO를 바이너리 PSO로 변환하여 파일 저장.
- 여러 플레이 결과를 쌓기 위해, 바이너리 PSO를 파이프라인 메타데이터에 Merge.
- 배포 빌드 쿠킹시 파이프라인 메타데이터를 다시 바이너리 PSO로 변환
- 배포 빌드에서 바이너리 PSO를 활용하여 초기화 시점에 PSO 생성. 로우 레벨 캐시에 등록.
- 배포 빌드에서 PSO 사용하기

※ PSO 캐시 사용

- r.ShaderPipelineCache.Enabled = 1 / 커맨드라인 “-psocache”
- Share Material Shader Code (셰이더 코드 라이브러리) 활성화

셰이더 파이프라인 캐시 – 동작

테스트 빌드

- 테스트 빌드 쿠킹
- 플레이시 생성된 PSO를 바이너리 PSO로 저장
- 바이너리 PSO를 파이프라인 메타데이터에 Merge

배포 빌드

- 쿠킹 시 파이프라인 메타데이터를 바이너리 PSO로 변환
- 초기화 시점에 PSO 생성. 로우 레벨 캐시에 등록
- 런타임에 PSO 사용하기

셰이더 파이프라인 캐시 – 동작

테스트 빌드

- 테스트 빌드 쿠킹
- 플레이시 생성된 PSO를 바이너리 PSO로 저장
- 바이너리 PSO를 파이프라인 메타데이터에 Merge

배포 빌드

- 쿠킹 시 파이프라인 메타데이터를 바이너리 PSO로 변환
- 초기화 시점에 PSO 생성. 로우 레벨 캐시에 등록
- 런타임에 PSO 사용하기

셰이더 파이프라인 캐시 – 테스트 빌드

테스트 빌드 쿠킹

- 콘텐츠의 모든 머티리얼의 Stable Shader 정보를 생성한다 = scl.csv에 저장
  - 저장 정보: ClassNameAndObjectPath, ShaderType, ShaderClass, MaterialDomain, FeatureLevel, QualityLevel, TargetFrequency, TargetPlatform, VFType, Permutation, OutputHash
  - OutputHash: Share Material Shader Code 가 필요한 이유
- 해당 정보는 후에 “파이프라인 메타데이터” 생성시 사용
- 어떤 정보가 저장되는지는 뒤 페이지에.

셰이더 파이프라인 캐시 – 테스트 빌드

Stable Shader 정보

Key	Value
ClassNameAndObjectPath	Material /Engine/EngineMaterials/DefaultTextMaterialOpaque.DefaultTextMaterialOpaque
ShaderType	TMobileBasePassPSFMobileMovableDirectionalLightCSMAndSHIndirectPolicyINT32_MAXHDRLinear64Skylight
ShaderClass	MeshMaterial
MaterialDomain	MD_Surface
FeatureLevel	ES3_1
QualityLevel	High
TargetFrequency	SF_Pixel
TargetPlatform	GLSL_ES3_1_ANDROID
VFType	FLocalVertexFactory
Permutation	Perm_0
OutputHash	EEA38C7B6E88DC7334755EAFBEE33D0CEB9DC3D3

셰이더 파이프라인 캐시 – 테스트 빌드

플레이시 생성된 PSO를 바이너리 PSO로 저장

- 로우 레벨 캐시의 파일을 직접 배포하는 것이 아님.
- 플랫폼 독립적인 정보(GraphicsPipelineInitializer)를 활용 = ushaderpipeline 생성
  - 바인딩된 셰이더 BoundShaderState: VertexDeclarationRHI, VertexShaderRHI, PixelShaderRHI, GeometryShaderRHI, DomainShaderRHI, HullShaderRHI
  - 렌더 스테이트들: BlendState, RasterizerState, DepthStencilState, ImmutableSamplerState
  - DepthStencil 관련 정보: DepthStencilTargetFormat, DepthStencilTargetFlag, DepthTargetLoadAction, DepthTargetStoreAction, StencilTargetLoadAction, StencilTargetStoreAction, DepthStencilAccess
  - 기타: bDepthBounds, PrimitiveType, RenderTargetsEnabled, RenderTargetFormats, RenderTargetFlags
  - 멀티 샘플링 정보: NumSamples

※ 바이너리 PSO 저장

- r.ShaderPipelineCache.LogPSO = 1 / 커맨드라인 “-logPSO”

셰이더 파이프라인 캐시 – 테스트 빌드

바이너리 PSO를 파이프라인 메타데이터에 Merge

- 제외되는 파이프라인 정보가 없도록, 테스트 빌드로 최대한 모든 콘텐츠를 다 실행/렌더링 하는 것이 중요.
- 바이너리 PSO와 Stable Shader 정보를 합하여, 파이프라인 메타데이터를 생성 = stablepc.csv 저장
  - Blend 스테이트 : Multiple RenderTarget(ColorBlendOp/Src/Dest, AlphaBlendOp/Src/Dest, ColorWriteMask) + bUseIndependentRenderTargetBlendStates = <0 4 5 0 1 5 15 / 0 1 0 0 1 0 15 / 0 1 0 0 1 0 15 / 0 1 0 0 1 0 15 / 0 1 0 0 1 0 15 / 0 1 0 0 1 0 15 / 0 1 0 0 1 0 15 / 0 1 0 0 1 0 15 / 0 >
  - Rasterizer 스테이트 : DepthBias, SlopeScaleDepthBias, FillMode, CullMode, bAllowMSAA, bEnableLineAA = < 0.000000 0.000000 2 0 0 0 >
  - DepthStencil 스테이트 : DepthWrite/Test, FrontFaceStencil, BackFaceStencil, StencilReadMask, StencilWriteMask = < 0 7 / 0 7 0 0 0 / 0 7 0 0 0 / 255 255 >
  - MSAASamples : 1 // DepthStencilFormat : 0
  - …
  - Shaderslot0~4 : ClassNameAndObjectPath, ShaderType, ShaderClass, MaterialDomain, FeatureLevel, QualityLevel, TargetFrequency, TargetPlatform, VFType, Permutation, OutputHash : "Material Game/UI/Material/M_UI_IntroBG02.M_UI_IntroBG02, TMobileBasePassVSFMobileDistanceFieldShadowsLightMapAndCSMLightingPolicyHDRLinear64,MeshMaterial, MD_Surface, ES3_1, High, SF_Vertex, GLSL_ES3_1_ANDROID, FLocalVertexFactory, Perm_0, 0000000000000000000000000000000000000000"

셰이더 파이프라인 캐시 – 동작

테스트 빌드

- 테스트 빌드 쿠킹
- 플레이시 생성된 PSO를 바이너리 PSO로 저장
- 바이너리 PSO를 파이프라인 메타데이터에 Merge

배포 빌드

- 쿠킹 시 파이프라인 메타데이터를 바이너리 PSO로 변환
- 초기화 시점에 PSO 생성. 로우 레벨 캐시에 등록
- 런타임에 PSO 사용하기

셰이더 파이프라인 캐시 – 배포 빌드

쿠킹 시 파이프라인 메타데이터를 바이너리 PSO로 변환

- stablepc.csv가 충분한 커버리지를 가졌다면, 배포 빌드 제작 = stable.upipelinecache 생성
- 바이너리 PSO 생성시 플랫폼별 불필요한 정보들은 제거
  - OpenGL의 경우, 바인딩 셰이더 스테이트(BSS)만 고려.

```
LogCook: Display: ---- Running UShaderPipelineCacheToolsCommandlet for platform Android_ETC1  shader format GLSL_ES3_1_ANDROID
…
LogShaderPipelineCacheTools: Display: Loading ../../../../../Workspaces/Projects-4.21/ActionRPG/Build/Android/PipelineCaches/ActionRPG_GLSL_ES3_1_ANDROID.stablepc.csv....
LogShaderPipelineCacheTools: Display: Loaded 448 stable PSO lines.
LogShaderPipelineCacheTools: Display: Re-deduplicated into 101 binary PSOs.
LogShaderPipelineCacheTools: Display: OpenGL detected, reducing PSOs to be BSS only as OpenGL doesn't care about the state at all when compiling shaders.
LogShaderPipelineCacheTools: Display: BSS only reduction produced 91 binary PSOs.
LogShaderPipelineCacheTools: Display: Wrote binary PSOs, (44KB) to D:/Workspaces/Projects-4.21/ActionRPG/Saved/Cooked/Android_ETC1/ActionRPG/Content/PipelineCaches/Android/ActionRPG_GLSL_ES3_1_ANDROID.stable.upipelinecache
LogCook: Display: ---- Done running UShaderPipelineCacheToolsCommandlet for platform Android_ETC1
```

셰이더 파이프라인 캐시 – 배포 빌드

초기화 시점에 PSO 생성. 로우 레벨 캐시에 등록

- 엔진 초기 시점 혹은 임의의 시점에 미리 PSO를 생성해두어, PSO가 재활용되도록 만든다.
- PSO캐시의 Precompile 과정을 통해 실제 사용될 PSO 생성
    - 모든 바이너리 PSO에 대해 각각 GraphicsPipelineInitializer 생성

      -> SetGraphicsPipelineState(…) 호출

      -> PipelineStateCache::GetAndOrCreateGraphicsPipelineState(…) 통해

      -> GGraphicsPipelineCache.Find(…) 검색 실패

      -> RHICreateGraphicsPipelineState(…)

셰이더 파이프라인 캐시 – 배포 빌드

LogRHI: Opened FPipelineCacheFile: ../../../ActionRPG/Content/PipelineCaches/Android/ActionRPG_GLSL_ES3_1_ANDROID.stable.upipelinecache (GUID: 85E2662143839EF38C1DDA81587A11C1) with 91 entries.
LogRHI: Scanning Binary program cache, using Shader Pipeline Cache version 85E2662143839EF38C1DDA81587A11C1
LogRHI: OnShaderScanProgramCacheFile
LogRHI: OnShaderScanProgramCacheFile : Failed to open /storage/emulated/0/Android/data/com.EpicLRT.ActionRPGSample/files/ProgramBinaryCache/GLSL_ES3_1_ANDROID_F63BDD0EFCE0271C12DC164CAFBAA6C26053BFE3
LogRHI: Deleting binary program cache folder: /storage/emulated/0/Android/data/com.EpicLRT.ActionRPGSample/files/ProgramBinaryCache
LogRHI: Display: Opened pipeline cache and enqueued 91 of 91 tasks for precompile.
LogRHI: Using OpenGL program LRU cache: 0
LogRHI: AddProgramFileEntryToMap : Adding program: Program V_7E7320CE9F491AEB8090A265F7D6E2B367481817_P_0D18C4AA803AFAA57BE8370BB420A557CD9A46A5
LogRHI: AddProgramFileEntryToMap : Adding program: Program V_001B6271AD76D6646F455AF7EB1E162E526C94C8_P_5F7D5C230B33E2D4AAF5CC4495C2C890A7223B51
…
LogRHI: Warning: FShaderPipelineCache completed 91 tasks in 6.91s (6.94s wall time since intial open).
LogRHI: OnShaderPipelineCachePrecompilationComplete: 91 shaders

셰이더 파이프라인 캐시 – 배포 빌드

런타임에 PSO 사용하기

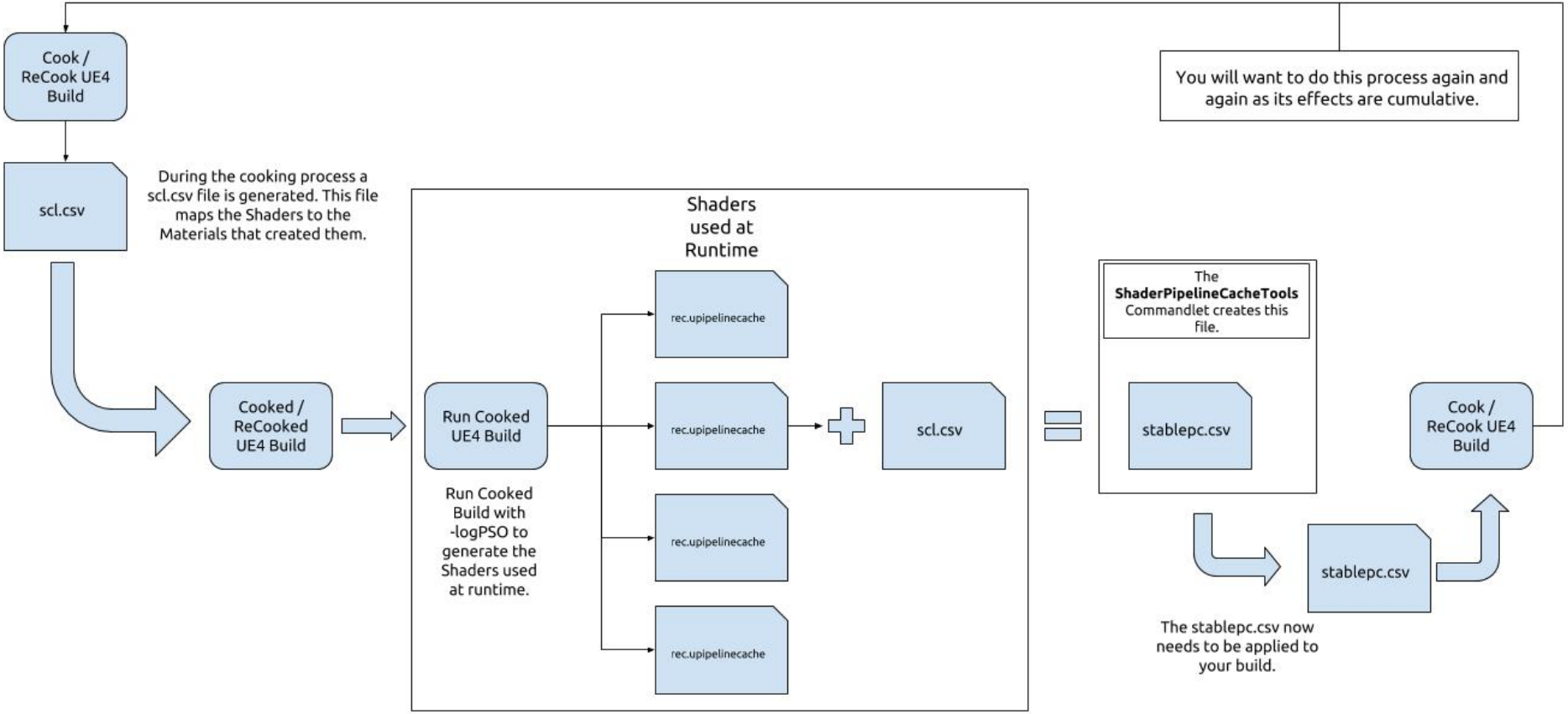
- 드로우콜마다 GraphicsPipelineInitializer 생성

  -> SetGraphicsPipelineState(…) 호출

  -> PipelineStateCache::GetAndOrCreateGraphicsPipelineState(…) 통해

  -> GGraphicsPipelineCache.Find(…) 검색 성공

  -> RHISetGraphicsPipelineState(…)


Cook / ReCook UE4 Build
scl.csv
During the cooking process a scl.csv file is generated. This file maps the Shaders to the Materials that created them.
Cooked / ReCooked UE4 Build
Shaders used at Runtime
Run Cooked UE4 Build
rec.upipelinecache
rec.upipelinecache
rec.upipelinecache
rec.upipelinecache
scl.csv
Run Cooked Build with -logPSO to generate the Shaders used at runtime.
The ShaderPipelineCacheTools Commandlet creates this file.
stablepc.csv
Cook / ReCook UE4 Build
stablepc.csv
The stablepc.csv now needs to be applied to your build.
You will want to do this process again and again as its effects are cumulative.

셰이더 파이프라인 캐시 – 파생 데이터

[테스트 빌드] = -logPSO

- [Cooking output]: scl.csv / ushaderbytecode
- [Execution Output]: rec.upipelinecache
- [Merge output]: stablepc.csv

[배포 빌드]

- [Cooking Input]: stablepc.csv
- [Cooking Output]: stable.upipelinecache / ushaderbytecode

PSO 캐시 사용 가이드

주의사항

※ 제안드리는 내용은 모든 프로젝트에 맞지 않을 수 있습니다.
설명드리는 개념들을 통해 프로젝트에 적합한 방식을 선택하세요.

- 자주 사용될만한 케이스 가정
  - Android OpenGL ES3.1
  - 콘텐츠 배포 방식: Minimal APK + DLC w/ HttpChunkInstallData
    - Minimal APK: Android ETC
    - DLC w/ HttpChunkInstallData: Android ASTC
  - Share Material Shader Code = True

바이너리 PSO 저장

r.ShaderPipelineCache.Save
Ex. -logPSO 자동 저장이 원하는 대로 동작하지 않는 경우

- PSO 로깅 필요 조건
  - r.ShaderPipelineCache.Enabled = 1 / r.ShaderPipelineCache.LogPSO = 1 / r.ShaderPipelineCache.SaveBoundPSOLog = 1
- 프로젝트에 따라 디바이스 프로파일 혹은 커맨드라인 혹은 콘솔명령 통해 설정
- r.ShaderPipelineCache.Save 직접 실행: {ProjectDir}\Saved\CollectedPSOs

PSO 생성 시기 조절

엔진 기본 설정: 엔진 Preinit

LaunchEngineLoop.cpp

FEngineLoop.PreInit.{,if

if (FPlatformProperties::RequiresCookedData())

```
2133    if (FPlatformProperties::RequiresCookedData())
2134    {
2135        // Open the game library which contains the material shaders.
2136        FShaderCodeLibrary::OpenLibrary(FApp::GetProjectName(), FPaths::ProjectContentDir());
2137        if (FPaths::HasProjectPersistentDownloadDir())
2138        {
2139            FShaderCodeLibrary::OpenLibrary(FApp::GetProjectName(), FPaths::Combine(*FPaths::ProjectPersiste
2140        }
2141
2142        // Now our shader code main library is opened, kick off the precompile.
2143        FShaderPipelineCache::OpenPipelineFileCache(GMaxRHIShaderPlatform);
2144    }
```

PSO 캐시 Precompile 과정 늦추기

- 바이너리 PSO를 읽어 Batch로 컴파일 진행
- PSO 캐시는 Tick마다 상태에 따라 다르게 동작
- PauseBatching() / ResumeBatching() 으로 Pause 상태 설정

PSO 생성 속도 조절

r.ShaderPipelineCache.SetBatchMode [Pause|Fast|Background]

- 배치 모드
  - Precompile시 렌더스레드 타임 슬라이스로 배치를 처리
  - 한 프레임에 처리할 배치량과 최대 할당 시간 정보 세팅
  - 엔진 기본값: Fast 모드 = 50 PSOs + 16ms / Background 모드 = 1 PSO + 시간 제한 없음.
- 로딩 스크린 중에 컴파일 시도? Fast 모드!
- 게임 플레이 중에 컴파일 시도? Background 모드!

DLC + Shader Code Library

Trouble shooting

- [비어있는 DLC 플러그인 한정] 쿠킹 실패
  - ```
    //FString BasePath = !IsCookingDLC() ? FPaths::ProjectContentDir() : GetContentDirecctoryForDLC();
    FString BasePath = !IsCookingDLC() ? FPaths::ProjectContentDir() : GetBaseDirectoryForDLC() + TEXT("/Content");
    ```

- 런타임 크래시
  - ```
    void LoadShaderCodeLibraryFromPluginAfterMountPak()
    {
        auto Plugins = IPluginManager::Get().GetEnabledPluginsWithContent();
        for (auto Plugin : Plugins)
        {
            if (Plugin.CanContainContent() && Plugin.IsEnabled())
            {
                FShaderCodeLibrary::OpenLibrary(Plugin.GetName(), Plugin.GetBaseDir());
                FShaderCodeLibrary::OpenLibrary(Plugin.GetName(), Plugin.GetContentDir());
            }
        }
    }
    ```

은 경우 크래시

기 작업을 하나, 콘텐츠를

.

DLC + Shader Code Library

PSO 캐시 다시 열기?

- 엔진 기본 동작
  - 엔진 Preinit 시, 프로젝트 이름(Global, Game) 혹은 플러그인 이름(DLC제외)의 Shader Code Library 열기
  - 엔진 Preinit 시, 프로젝트 이름으로 Shader Pipeline Cache도 함께 열기
    - Shader Pipeline Cache와 동일한 GUID를 사용한 Program Binary Cache 생성/로드
- 일반적으로 아래와 같이 진행
  - [엔진 초기화] APK로 엔진 실행 → APK내 ShaderCodeLibary 열기 → PSO 캐시 열기 -> PSO 캐시 Precompile
  - [패치용 레벨] Pak 마운트 → 크래시 제거 위해 DLC내 ShaderCodeLibrary 열기→ PSO 캐시 다시 열기?

DLC + Shader Code Library

PSO 캐시 배포 전략

- 알아두셔야 할 것은, 현재 4.22에서
  - Shader Code Library는 DLC(플러그인) 지원
  - Shader Pipeline Cache는 DLC(플러그인) 미지원

- stable.upipelinecache에 DLC를 포함한 모든 콘텐츠가 들어가도록 하는 것이 중요

- 최대한 많은 정보를 가지고 오랜 시간이 걸리더라도 미리 준비
  - 사용자 경험 훼손 = 게임 오픈시 발생하는 시간 < 게임 플레이 중 발생하는 히치
  - 포트나이트에서 사용하는 방식과 크게 다르지 않음.

포트나이트 사례

포트나이트에서 사용하는 방식과 크게 다르지 않음.

- iOS기준 IPA = 166.3 MB / DLC 다운로드 4.11 GB
- DLC 다운로드 및 인스톨 후 패치용 레벨에서 바로 DLC에 대한 PSO 캐시 생성.
- 모든 캐시 생성 후 게임 재시작 혹은 플레이 레벨 로드

감사합니다
Q&A