<모바일 게임 성능 최적화>

***01. 최적화 정의***

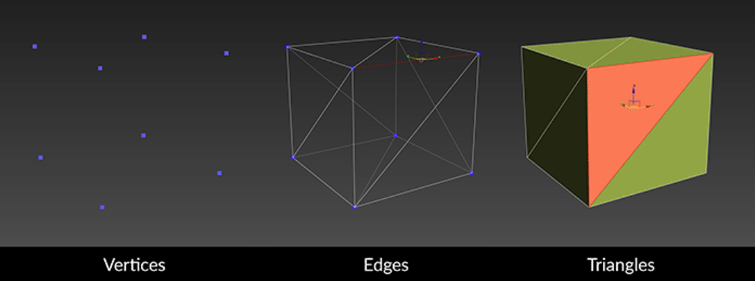
최적화란 소프트웨어의 성능을 향상하는 프로세스. 모바일 애플리케이션에서는 대개 애플리케이션을 수정하여 리소스가 한정적인 모바일 기기에서 빠르게 실행되도록 하는 것.

***02. 최적화 단계와 예시***

1. 단계
   1. 프로파일러로 애플리케이션을 프로파일링하거나 측정함
   2. 데이터를 분석해 병목 현상 찾음
   3. 유효한 최적화를 결정하여 적용
   4. 최적화가 됐는지 확인
   5. 성능이 수긍할 수 없는 수준이라면 1단계로 돌아가서 프로세스를 반복
2. 예시
   1. 게임을 테스트하면서 그래픽상의 문제나 성능 저하를 확인. Unity 프로파일러로 측정해 문제 찾음
   2. 프로파일러에서 나온 측정 결과를 분석함으로써 성능 문제의 원인을 구별하고 파악할 수 있음.
   3. 이를 확인한 다음 다른 조건에서 테스트하여 특정 모델이 뷰에 보일 때 게임에서 지나치게 많은 버텍스를 렌더링 하는 것은 아닌지 확인
   4. 팀에 관련 내용을 알리고 일부 3D 모델에 LOD를 사용하여 모델이 뷰에서 멀어질 때 버텍스 수를 줄이도록 함
   5. 모델에 LOD를 적용한 뒤 게임을 다시 테스트하여 최적화가 됐는지 확인.

***03. 지오메트리***

지오메트리(폴리곤 메시)는 3D 오브젝트의 셰이프를 구성하는 Vertex, Edge, Triangles의 집합체. 이런 메시는 자동차나 환경 오브젝트, 캐릭터 등 3D 애플리케이션에 표시되는 모든 종류의 에셋이 될 수 있음.



* Vertex 3D 오브젝트의 표면을 구성하는 점
* Edge는 두 vertex를 일직선으로 연결한 것
* Triangles는 세 개의 모서리로 연결된 세 개의 vertex로 구성됨. 삼각형을 폴리곤과 면으로도 부름

***04. 삼각형과 폴리곤***

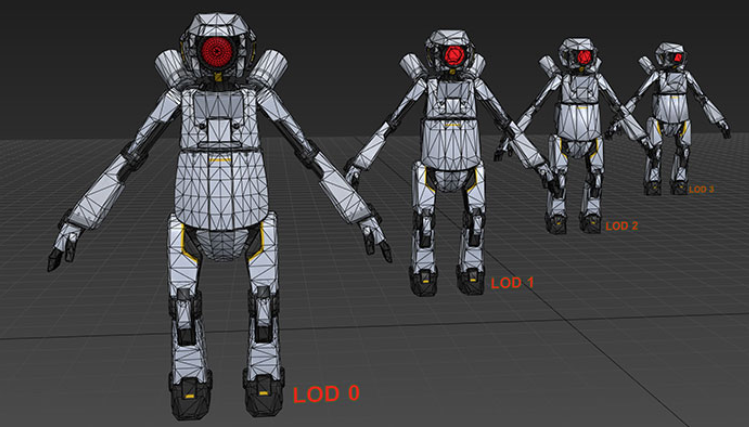
애플리케이션의 성능 최적화를 위해서는 특정 시점에 화면에 몇 개의 삼각형이 표시되는지를 알아야함. 삼각형을 적게 사용할수록 GPU에서 처리할 버텍스 수도 적어짐. 이는 모바일 플랫폼용 콘텐츠를 제작할 때는 특히 중요한 부분임. 버텍스를 처리하려면 컴퓨터 리소스가 많이 들기 때문임.

모바일 기기 화면은 일반적인 컴퓨터 모니터보다 작기 때문에 수많은 삼각형을 사용한 디테일이 모바일 기기에서는 보이지 않을 수 있음. 그렇기 때문에 카메라에 가까운 3D 오브젝트에는 삼각형을 많이 사용하고, 배경의 3D 오브젝트에는 삼각형을 적게 사용하는 것이 좋음.

***05. LOD(Level Of Detail)***

1. LOD의 사용

LOD를 사용하면 카메라에서 오브젝트가 떨어진 거리에 따라 씬에서 각 오브젝트에 렌더링 되는 버텍스 수를 제어할 수 있음.



* 삼각형의 수를 줄일 때는 삼각형이 오브젝트의 실루엣에 미치는 영향에 주의할 것
* 평평한 영역에서 폴리곤을 더 제거
* 오브젝트가 카메라로부터 멀어짐에 따라 오브젝트에서 사용하는 텍스처 수를 줄일 수 있음

1. LOD를 사용하면 안 되는 경우
   1. 카메라 뷰와 오브젝트가 모두 정적인 경우
   2. 오브젝트가 이미 폴리곤 수를 적게 사용할 경우

***06. 텍스처 압축***

텍스처는 3D 모델의 표면에 적용되어 디테일과 사실성을 더하는 이미지.

텍스처는 표면의 요철이나 반사도와 같은 정보도 제공함.

텍스처 압축은 데이터 크기를 줄이는 동시에 화질을 유지하기 위해 적용하는 이미지 압축임

Android에는 ASTC(Adaptive Scalable Texture Compression), ETC(Crcsson Texture Compression) , ETC2등 다양한 옵션이 있음.

***07. 텍스처 아틀라스***

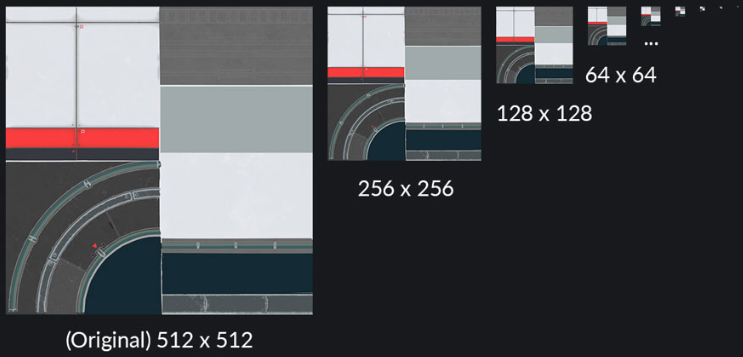
텍스처 아틀라스는 함께 패킹된 여러 작은 이미지의 데이터를 포함하는 이미지.

텍스처 아틀라스를 사용하면 하나의 메시에 하나의 텍스처를 사용하는 대신 하나의 큰 텍스처를 여러 메시에서 활용하게 됨.

텍스처 아틀라스를 통해 동일한 텍스처 아틀라스와 머터리얼을 공유하는 여러 정적 오브젝트를 배칭할 수 있음. 배칭하면 드로우콜 수가 줄어들고, 드로우 콜이 줄어들면 CPU의 영향을 받는 게임의 성능이 향상됨.

***08. 밉맵***

밉맵은 해상도를 줄여서 저장한 텍스처 사본. 밉매핑은 텍스처용 디테일 수준(LOD)이라고 생각해도 됨.



밉맵이 있는 텍스처가 렌더링 되면 스크린 공간에 텍스처 공간, 즉 픽셀 프래그먼트가 차지하는 공간의 크기에 따라 적합한 샘플링 수준이 선택됨. 오브젝트가 카메라로부터 멀리 있으면 저해상도, 가까이 있으면 고해상도 텍스처가 적용됨. 그렇기 때문에 모바일 애플리케이션의 성능과 품질을 높일 수 있음.

***09. UV***

U는 좌우, V는 위아래, W는 앞뒤 방향으로 이루어져 있는데, UV 맵이라는 것은 2D로 이루어져 있어 깊이감(W방향)이 빠진 U와 V축만 남아있게 되어 UV라고 불림. UV맵은 2D 텍스처를 3D 모델의 표면에 투사함.

***10. 셰이더와 머티리얼***

셰이더는 화면에 오브젝트를 드로우하는 방식과 오브젝트를 드로우하기 위해 이루어져야 하는 모든 계산을 GPU에 전달하는 작은 프로그램. 셰이더를 작성할 수 있는 일반적인 스크립팅 언어로는 HLSL(High-Level Shading Language)과 GLSL(OpenGL Shading Language)이 있음.

셰이더는 오브제트의 룩을 정의하는 머티리얼을 통해 게임 오브젝트에 적용됨. 머티리얼은 셰이더에서 사용할 수 있는 파라미터를 설정하는데 사용됨. 예를 들어 머티리얼은 셰이더에서 참조하는 색상, 텍스처, 숫자 값을 지정할 수 있음.

***11. 라이트맵과 텍셀***

1. 라이트맵

라이트맵은 빛의 방향과 세기에 따라 생성되는 빛을 받는 오브젝트의 그림자, 반사면, 면의 밝기 차이 등의 라이팅 정보를 사전에 저장하는 텍스처를 뜻함

1. 텍셀

텍스처 맵의 개별 픽셀. 텍셀은 라이트맵에서 조명이 오브젝트를 비추는 지점마다 조명 정보를 저장함.

1. 라이트맵 최적화

필요한 라이트맵의 데이터 양을 최소화하려면 라이트 매핑 설정에서 베이크 단위별로 텍셀 수를 조정해야함.

***12. 프로젝트 설정 최적화 예시***

(1) Accelerometer Frequency감소 또는 비활성화(iOS)

Unity iOS애플리케이션은 기본적으로 가속 센서를 초당 60번 샘플링..

그렇기 때문에 가속 센서 샘플링 빈도를 낮추면 성능이 다소 향상 될 수 있고, 가속 센서 입력을 사용하지 않는 게임의 경우 빈도를 0으로 설정할 수 있음.

(2) 불필요한 플레이어 설정 또는 품질 비활성화

Auto Graphics API를 비활성화 하면 셰이더 배리언트가 과도하게 생성되지 않도록 방지할 수 있음

애플리케이션이 오래된 CPU를 지원하지 않을 경우 Target Architectures를 비활성화.

텍스트이(가) 표시된 사진

자동 생성된 설명

(3) 불필요한 물리 비활성화

게임에서 물리를 사용하지 않는다면 Auto Simulation과 Auto Sync Transforms를 비활성화

텍스트이(가) 표시된 사진

자동 생성된 설명

(4) 올바른 프레임 속도 선택

기기의 한계인 60fps까지 밀어붙이기 보다는 30fps정도에서 타협해 실행하는 것이 좋음

Application.targetFrameRate를 활용해 런타임 중에 프레임 속도를 동적으로 조정할 수 있음.

예를 들어 속도가 느리거나 비교적 정적인 씬에는 30fps아래로, 게임 플레이 중에는 더 높은 fps설정을 유지할 수 있음

텍스트이(가) 표시된 사진

자동 생성된 설명

(5) 대규모 계층 구조 사용 지양

계층 구조가 단순하면 씬에서 트랜스폼을 새로 고침 할 때 멀티 스레딩의 이점을 누릴 수 있음.

계층 구조가 복잡하면 불필요한 트랜스폼 연산과 높은 가비지 컬렉션 비용이 발생함.

(6) 트랜스폼 한 번에 이동

트랜스폼 이동 시, Transform.SetPositionAndRotation을 사용해 위치와 회전을 한 번에 하면 트랜스폼을 두 번 수정함으로써 발생하는 오버헤드를 지할 수 있음

***13. 에셋 최적화 예시***

에셋 파이프라인은 애플리케이션 성능에 지대한 영향을 줄 수 있음. 플랫폼별 오버라이드 탭을 사용하면 텍스처, 메시 지오메트리와 같은 에셋을 최적화 할 수 있음.

(1) 올바른 텍스처 임포트

- 최대 크기 줄이기 : 시각적으로 허용 가능한 결과를 내는 최소한의 설정 사용.

- POT(Powers of Two) 사용 : Unity에서 모바일 텍스처 압축 형식(PVRCT or ETC)를 사용하려면 POT 텍스처가 필요함

- 텍스처 아틀라스 사용 :하나의 텍스처에 여러 텍스처를 배치하면 드로우콜을 줄이고 렌더링 속도를 높일 수 있음.

- Read/Write Enabled 옵션 해제

- 불필요한 밉맵 비활성화 : 일정한 크기로 유지되는 텍스처에는 밉맵이 필요하지 않음

(2) 텍스처 압축

압축되지 않은 텍스처에는 더 많은 메모리가 필요함

iOS와 Android 모두에서 ASTC(Adaptive Scalable Texture Compression)를 사용.

(3) 메시 임포트 설정

텍스처와 마찬가지로 메시도 신중하게 임포트 하지 않으면 과도한 메모리 사용으로 이어질 수 있음.

- 메시 압축 : 압축으로 디스크 공간을 줄일 수 있음 (압축 수준을 다양하게 조절해보고 모델에 적합한 수준 파악)

- Read/Write Enabled 비활성화 : 이 옵션을 활성화 하면 각각 메시 사본이 시스템 메모리와 GPU메모리에 유지되므로 메모리에서 메시가 중복됨.

- Rig와 BlendShapes 비활성화 : 메시에 골격 또는 블렌드 셰이프 애니메이션이 필요하지 않다면 이 옵션을 비활성화함.

- Normal과 Tangent 비활성화 : 메시의 머터리얼에 노멀 또는 탄젠트가 필요하지 않은 것이 확실하다면 이 옵션을 선택 해제하여 추가로 메모리를 절감할 수 있음.

(4) 폴리곤 개수 확인(업로드 전 확인 가능)

해상도가 높은 모델은 메모리 사용량이 더 많고 잠재적으로 GPU시간이 더 길 수 있음.

카메라의 시점에서 보이지 않는 폴리곤은 삭제, 고밀도 메시 대신 텍스처와 노멀 맵을 사용.

***14. 그래픽스 및 GPU 최적화 예시***

Unity는 프레임마다 렌더링 해야 하는 오브젝트를 지정한 다음 드로우 콜을 만듦.

(1) 드로우콜 배칭

함께 그릴 오브젝트를 배치로 구성하면 각 오브젝트를 그리는데 필요한 상태 변화가 최소화 됨.

- 동적 배칭 : 작은 메시의 경우 Unity는 CPU에서 버텍스를 그룹화하고 변환한 다음 모두를 한 번에 그림.

- 정적 배칭 : 움직이지 않는 지오메트리의 경우 Unity는 동일한 메테리얼을 공유하는 메시에 대한 드로우 콜을 줄일 수 있음. 동적 배칭에 비해 효율적이지만 메모리 사용량은 늘어남

- GPU 인스턴싱 : 동일한 오브젝트의 수가 많을 때 이 기법을 사용하면 그래픽 하드웨어의 사용을 통해 더 효율적인 배칭이 가능

- SRP 배칭 : 유니버설 렌더 파이프라인 에셋의 Advanced 항목에서 SRP Batcher를 활성화하면 씬에 따라 CPU 렌더링 시간이 크게 빨라짐

(2) 프레임 디버거 사용

프레임 디버거를 사용하면 실행 중인 게임을 특정 프레임에서 중지하고 해당 프레임을 렌더링하는 데 사용되는 개별 드로우 콜을 볼 수 있음.

(3) 동적 광원 수 줄이기

모바일 게임에서는 동적 광원을 지나치게 많이 추가하지 않아야함.

동적 메시에서는 커스텀 셰이더 이펙트나 라이트 프로브, 정적 메시에는 베이크된 광원과 같은 대안을 고려.

(4) 그림자 비활성화

MeshRenderer 및 광원별로 그림자 드리우기를 비활성화 할 수 있음. 가능하면 이것을 줄여 드로우 콜을 줄이기