유니티 3D 빌드 사이즈 최적화 체크 리스트

1.최적화 계획

최적화 작업 순서와 일정을 세우기 위해서는 어느 애셋이 전체 빌드에서 얼마나 차지하는지 측정할 필요가 있다. 유니티에서는 매 빌드 시마다 이 정보를 에디터 로그로서 파일에 기록한다. 아래 <그림 1>이 에디터 로그 중 애셋 별 빌드 사이즈를 표시한 줄이다.

```
Unloading 2 unused Assets to reduc
Total: 2.630412 ms (FindLiveObject
¬
¬
Textures ..... 10.8 kb∆  0.2% ¬
Meshes ..... 0.0 kb∆  0.0% ¬
Animations .... 0.0 kb∆  0.0% ¬
Sounds ..... 0.0 kb∆  0.0% ¬
Shaders ..... 0.0 kb∆  0.0% ¬
Other Assets .. 241.4 kb∆  5.1% ¬
Levels ..... 16.3 kb∆  0.3% ¬
Scripts ..... 429.0 kb∆  9.0% ¬
Included DLLs 3.9 mb∆  84.9% ¬
File headers .. 20.5 kb∆  0.4% ¬
Complete size 4.6 mb∆  100.0% ¬
```

그림 1

최적화는 애셋 중 가장 큰 비중인 것에 우선적으로 시도하되, 쉽게 사이즈를 줄일 수 있는 텍스처와 사운드 같은 원본 리소스를 먼저 고려하고, 사용하지 않는 애셋은 최우선으로 삭제해야 한다.

유니티 에디터 로그는 OS 마다 서로 다른 위치에 저장된다. 간단히 로그 파일을 열려면 로그 코솔 창 우상단의 'Open Editor Log' 버튼을 누르면 된다.

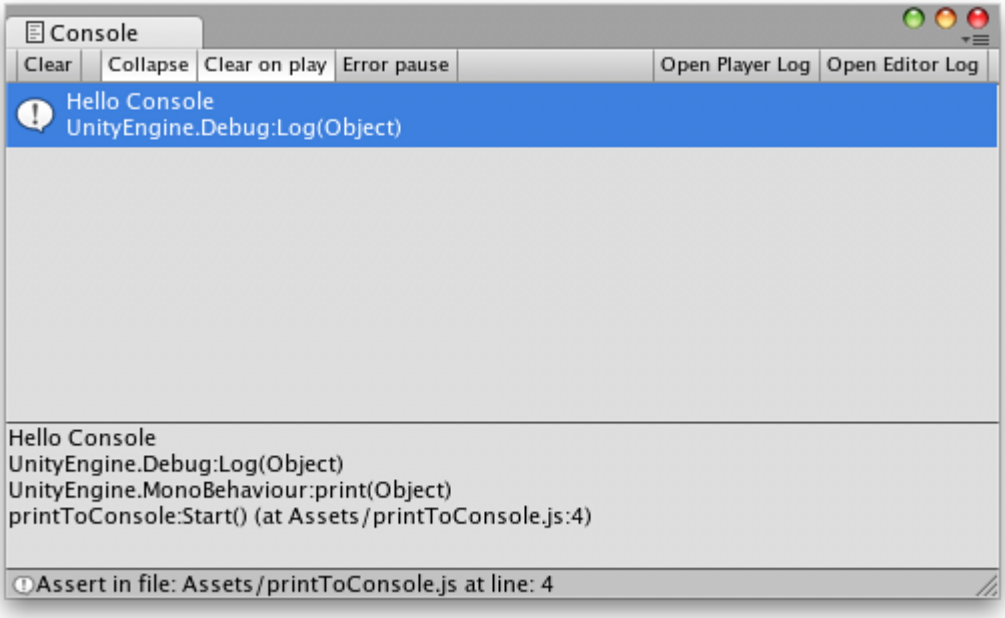

그림 2

2.에디터에서 할 수 있는 각종 설정

직접적으로 리소스들을 최적화하기 전에 플레이어 및 빌드 설정에서 빠트린 것은 없는지 먼저 체크한다. 설정에서 변경한 것이 있으면 빌드를 통해 확인하고, 사이즈 및 성능을 다시 측정한다.

2.1.빌드 설정

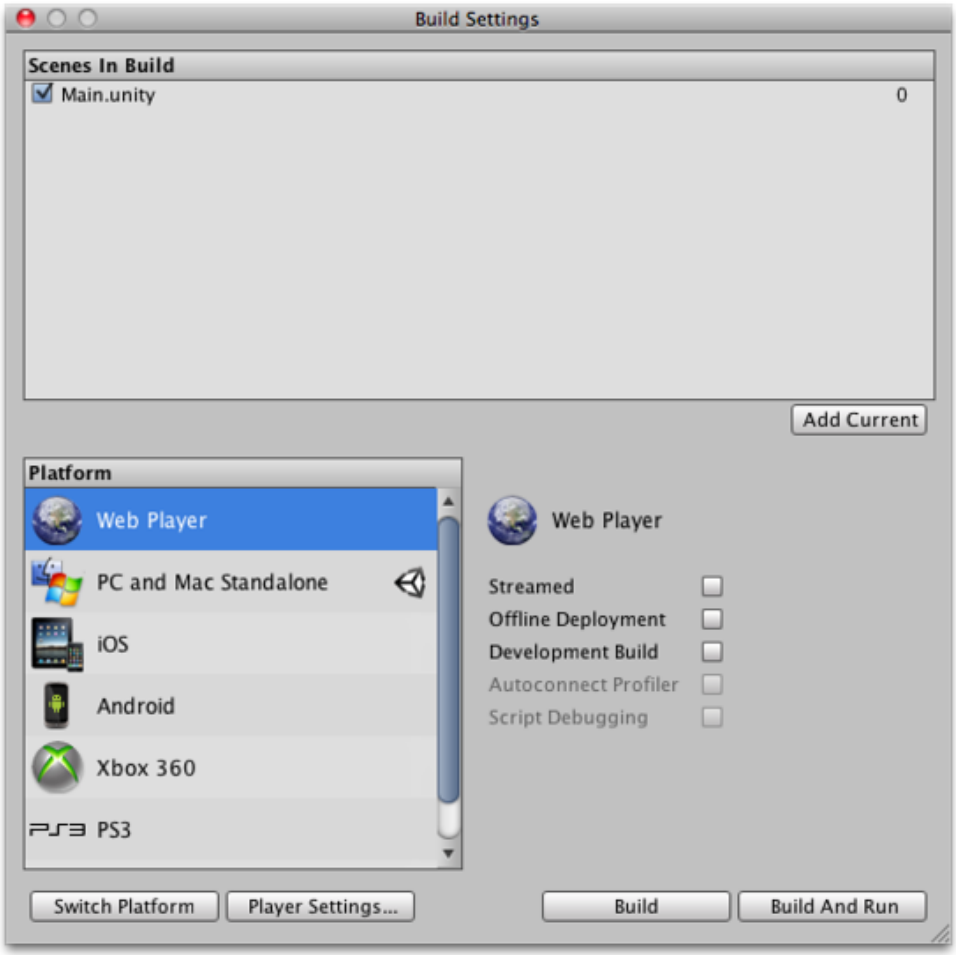

그림 3

2.1.1.씬 리스트에서 필요하지 않거나, 로드되지 않는 씬은 삭제한다.

2.1.2.'Development Build' 체크나 'Symlink Unity libraries'는 끈다.

2.1.3.iOS 에서 XCode 빌드 시 릴리즈 모드 빌드인지, 디버그 정보 삭제 옵션이 켜져있지 않은지 확인한다.

2.2.퀄리티 설정

그림 4

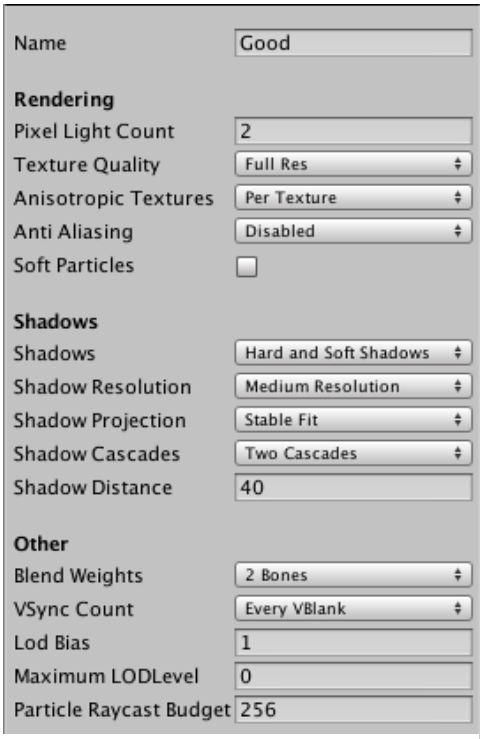

그림 5

2.2.1.플랫폼 별로 원하는 퀄리티 레벨이 선택돼 있는지 확인한다.

2.2.2.'Texture Quality'가 'Half Res'라면 'Full Res'로 변경하고 텍스처의 크기를 반으로 줄인다.
'Half Res' 설정에서는 Mip Map 이 있는 텍스처의 경우 레벨 0 이 아닌 레벨 1 부터 로드하게 된다.
따라서 표시되는 텍스처가 항상 원본 이미지의 1/2 사이즈로 나오게 된다. Mip Map 이 없는 텍스처는 이 설정의 영향을 받지 않는다.

2.2.3.'Anti Aliasing'은 성능에 여유가 있는게 아니라면 'Disable' 한다.

2.2.4.기타 설정은 'Fastest' 또는 'Fast' 설정의 기본값을 고려한다.

2.3.플레이어 설정

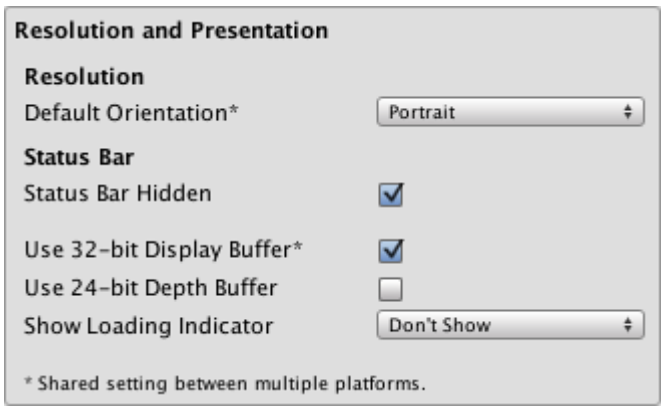

그림 6

2.3.1.'Resolution and Presentation' 항목에서 'Use 32-bit Display Buffer'와 'Use 24-bit Depth Buffer' 체크가 켜져 있지 않은지 확인한다.
만약 꺼져 있다면 현재 사용하는 모든 텍스처를 16 비트로 다운그레이드 하는 것을

고려한다. 켜져 있다면 정말 32 비트 품질이 필요한지 비교해보고, 성능이 더 중요하다면 16 비트로 변경한다.

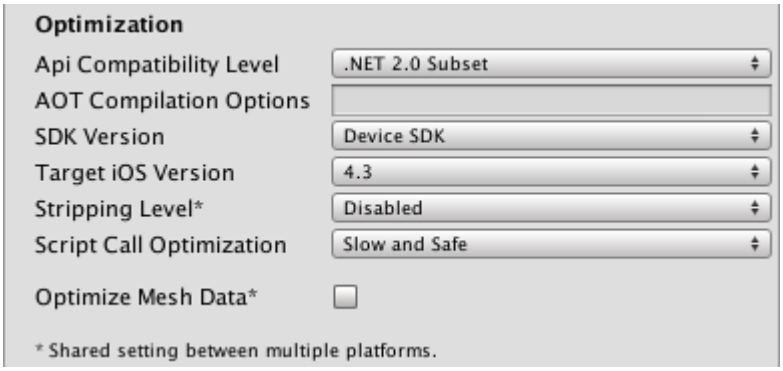

그림 7

2.3.2.'Api Compatibility Level'은 특별히 사용하려는 라이브러리가 없는 한 '.NET 2.0 Subset'을 사용한다. 사용하려는 클래스나 라이브러리가 현재 설정에서 호환 가능한지 여부는 아래 페이지에서 확인할 수 있다.
http://docs.unity3d.com/410/Documentation/ScriptReference/MonoCompatibility.html

2.3.3.'Stripping Level'은 가능한 한 최고 수준으로 설정한다. 단, 'Use micro mscorlib'이나 'Strip Byte Code(*iOS only*)' 레벨은 경우에 따라 프로그램을 크래쉬시킬 수 있으므로, 개발 초기부터 외부 라이브러리는 물론, 시스템 라이브러리라 하더라도 새로운 클래스를 도입하는 시점에 적용하여 문제가 없는지 확인하여야 한다.
각 레벨에서 사용 가능한 시스템 라이브러리 클래스 역시 위 항목의 링크에서 확인할 수 있다.

2.3.4.iOS 의 'Script Call Optimization'은 'Fast but no Exception'이 성능이나 빌드 사이즈 면에서는 이득이나, 런타임 C# 크래쉬 로그를 남겨서 수집하고자 하는 경우에는 어쩔 수 없이 'Slow and Safe'로 설정해야 한다.

2.4.오디오 매니저

그림 8

2.4.1.스테레오 사운드가 필요없다면 'Default Speaker Mode'를 'Mono'나 'Raw'로 낮춘다.

2.4.2.사운드 재생이 한꺼번에 나오는 씬에서 성능 저하가 일어난다면 'DSPBuffer Size'를 낮춰서 테스트해본다. 사운드가 성능 저하의 원인이 아니라면 굳이 건드리지 않아도 된다.

3.씬과 런타임 리소스 로드

씬을 어떻게 나누고 언제 어디서 리소스를 로드할 것인지도 최적화의 중요한 요소이다. 처음부터 모든 리소스를 씬이나 프리팹에 참조로 걸지 말고, 다운로드하거나 런타임 로드할 애셋은 계획적으로 선정하여 별도 관리하도록 한다.

그림 9

3.1.씬과 프리팹끼리는 가능한 한 애셋을 공유하지 않는다. 만약 다중 참조되는 머테리얼이나 텍스처가 있다면 (다운로드로 빼지 말고) 그냥 다 패키지에 넣든가, 처음부터 (인스펙터에서 참조를 걸지말고) 해당 공유 애셋만 런타임에 로드하도록 스크립트를 짜놓는다.

3.2.유니티에서 런타임에 애셋을 로드하는 방법에는 크게 3 가지가 있다.

AssetBundle - 어떤 애셋이든 포함할 수 있는 패키지 형식으로, 압축 및 캐쉬 기능을 제공한다.

/Resources - 이 디렉토리에 있는 애셋은 런타임에 Resources.Load() 라는 메소드를 이용하여 로드할 수 있다.

/StreamingAssets - 이 디렉토리의 애셋은 임포트되지 않고 원본 형식 그대로 패키징된다. 런타임에는 Application.streamingAssetsPath 에서 파일 I/O 를 통해 읽어들일 수 있다.

3.3.AssetBundle 을 로드하는 방법에는 다음 3 가지가 있다.

WWW - URL 을 이용하여 WWW 객체를 생성하여 로드를 하거나, WWW.LoadFromCacheOrDownload()를 이용해 유니티에서 제공하는 캐쉬에 저장할 수 있다. 유니티 캐쉬는 번들의 갯수가 많아질 경우 iOS/안드로이드에서 성능이 원하는만큼 안나올 수 있다. 또, 비동기 호출이기 때문에 WWW 객체 생성 후 다음 업데이트 루프로 넘어가기 전에는 완료 여부를 확인할 수 없다.

AssetBundle.LoadFromFile() - 로컬에 저장된 파일에서 즉시 로드할 수 있다. 번들은 다른 방법으로 다운로드되어 저장돼 있거나 /StreamingAssets 디렉토리를 이용해 포함돼 있어야 한다. 가장 성능이 좋은 방법이나, 비압축 번들에 대해서만 사용할 수 있다.

AssetBundle.LoadFromMemory() - 메모리에 파일을 로드한 다음, 번들 객체로 변환한다. 압축 번들에 대해서도 사용할 수 있고, 동기화 호출이기 때문에 로드 하는 루틴에서 바로 애셋을 꺼내어 참조할 수 있다. 압축 번들의 경우는 성능 문제가 있을 수 있으므로, 한 프레임에 너무 많은 번들을 로드하지 않도록 주의한다.

3.4./Resources 나 /StreamingAssets 디렉토리에 사용하지 않는 애셋이 있다면 반드시 삭제한다. 이 애셋들은 비록 로드되지 않더라도 무조건 패키지에 포함되게 된다.

4.모델과 애니메이션

모델이나 애니메이션은 디자이너가 익스포트한 애셋을 그대로 사용하여야 하는 경우가 많으므로, 디자인 시점에 최적화를 고려해 정점(vertex) 갯수나, 애니메이션 골격(bone) 갯수, 삼각형 갯수 등의 최대치를 설정해 놓고 그 범위를 넘지 않는 한도에서 품질을 맞춰야 한다.

4.1.모델 임포트 인스펙터

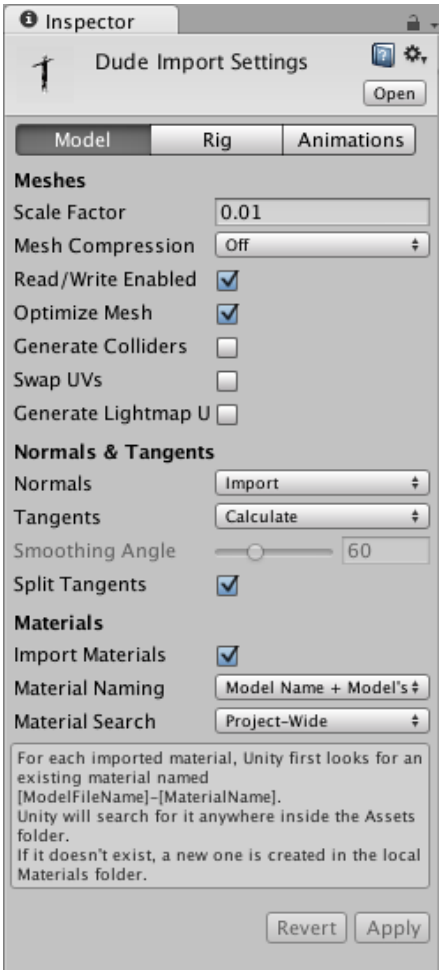

그림 10

4.1.1.'Mesh Compression'이 꺼져 있으면 켠다. 이 값은 모양이 뭉개지지 않는 범위에서 최대값을 선택한다.

4.1.2.'Read/Write Enabled'가 켜져 있으면 꺼본다. 이는 런타임 메모리 사용량을 줄여주지만, 내부에서 런타임에 메쉬 데이터를 읽거나 쓰는 경우, 꼭 스크립트에서 접근하지 않더라도 해당 기능을 사용하면 켜놓아야 한다.

4.1.3.'Normals'나 'Tangents' 설정은 객체의 성질에 따라 다른 설정이 필요하다. 노멀이나 탄젠트를 이용한 쉐이더를 특별히 선택해서 사용하지 않는 경우라면 'None'으로 설정해서 빌드 및 메모리 사이즈를 줄일 수 있다.

4.1.4.'Materials'는 FBX 파일이나 원본 3D 씬 파일에서 머테리얼까지 가져오는 경우가 아니라면, 즉, 머테리얼을 유니티에서 생성해서 별도로 로드하는 경우라면 'Disable' 한다.

4.2.애니메이션 임포트 인스펙터

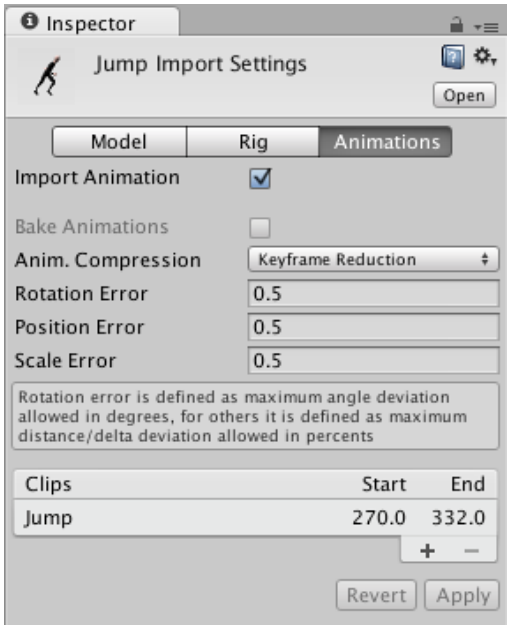

그림 11

4.2.1.애니메이션을 사용하지 않는 단순 메쉬의 경우, 'Import Animation' 체크를 끈다.

4.2.2.'Anim. Compression'은 최소한 'Keyframe Reduction' 이상으로 설정한다. 파일 사이즈가 문제가 될 때는 'Keyframe Reduction and Compression'으로 하여 조금 더 크기를 줄일 수 있다. 압축을 선택하더라도 런타임 메모리 사용량은 줄지 않는다.

5.텍스처

게임에서 가장 많은 용량을 차지하는 애셋은 대부분 텍스처(이미지)이다. 그만큼 텍스처는 작은 변경으로도 큰 효과를 볼 수 있는 최적화 대상이기도 하다. 모바일에서는 압축 텍스처야말로 최적화의 기본이라는 것을 기억하고, 만약 압축 텍스처를 사용하지 않았다면 무조건 그것부터 적용해보도록 한다.

5.1.3D 게임에서 사용하는 이미지는 전부 다 텍스처 애셋이지만 어떤 객체에서 사용되느냐에 따라 그 성질은 무척 다르다. 크게 3D(캐릭터, 지형지물) 텍스처와 2D(스프라이트, UI) 텍스처로 구분되므로 그 둘은 서로 다른 디렉토리에 나누어 놓든가 하여 머테리얼이나 쉐이더 등의 관련 애셋을 일괄적으로 다룰 수 있도록 정리한다.

5.2.압축 이외에 텍스처 사이즈를 줄이는 데는 다음과 같은 방법들을 고려해볼 수 있다.

16 비트 텍스처 - 텍스처 임포트 설정에서 'ARGB 16 bit'를 선택한다. 특히 디스플레이 버퍼가 이미 16 비트일 때는 텍스처가 32 비트라고 해도 품질적으로 차이를 느끼기 힘들다.

JPG 이미지 - 배경 이미지나 알파 채널이 없는 이미지는 JPG 로 패키징함으로써 좀 더 사이즈를 줄일 수 있다. JPG 이미지 로드에 대해서는 5.4 를 참조한다.

슬라이싱/타일링 - UI 텍스처의 경우, 같은 패턴이 반복되는 배경이나 바, 라인 등의 이미지가 많다. 이런 이미지를 통짜로 큰 사이즈를 사용하지말고, <그림 12>와 같이 테두리를 제외한 반복 패턴을 1 번만 포함한 최소 사이즈의 텍스처를 원래 사이즈의 메쉬에 입힘으로써 작은 텍스처로 큰 이미지를 그리는 효과를 낼 수 있다.

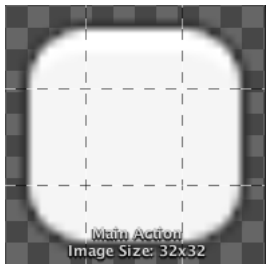

그림 12

5.3.텍스처 임포트 인스펙터

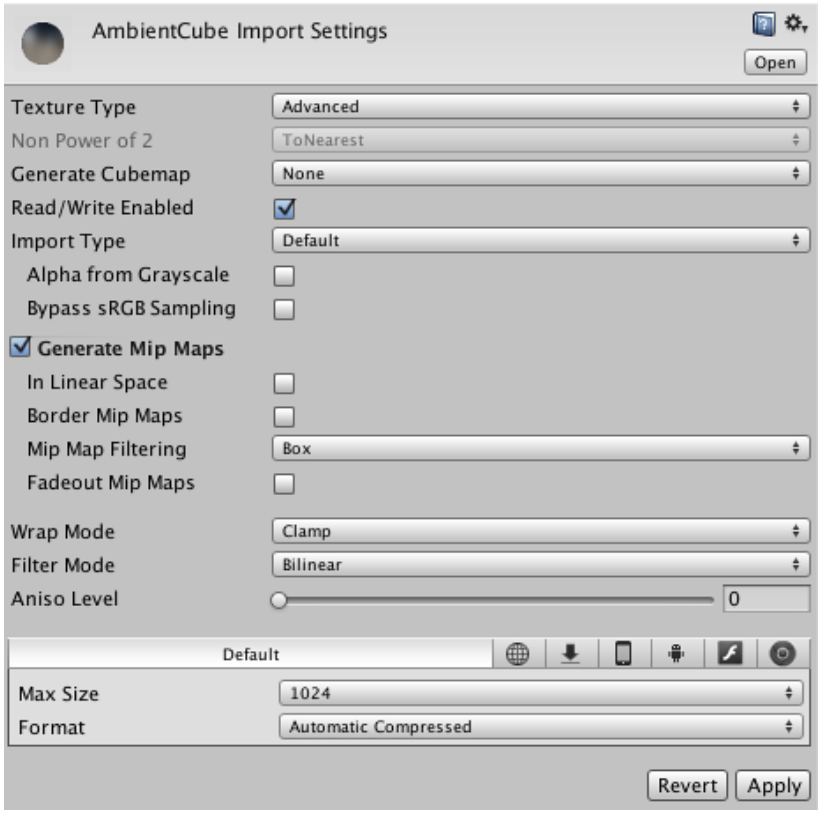

그림 13

5.3.1.대부분 최적화를 위해서는 'Texture Type'은 'Advanced'를 선택한다. 일괄적으로 한 종류의 텍스처 타입을 계속 사용할 때는 상관없지만, 객체 별로 다양한 옵션을 적용해야

하는 시점에는 타입 수준에서 선택하기보다는 Advanced 옵션의 최적 설정을 개별 항목 별로 찾아내는 것이 좀 더 확실히 사이즈를 줄일 수 있다.

5.3.2. 압축 텍스처를 선택하려면 대개 'Non Power of 2'는 비활성화하나, 배경 이미지 같이 1 장만 단독으로 표시하는 경우, 켜서 사이즈가 줄어드는 경우도 있다. 비록 2 제곱 사이즈 이미지가 아니라하더라도 런타임에는 엔진에서 사이즈를 맞춰버리는 경우도 있으므로, 원본 이미지부터 2 제곱 사이즈로 준비하는 것이 편한 경우가 많다.

5.3.3. 'Read/Write Enabled'는 무엇인지 알고 의도적으로 켜는 경우가 아니라면 항상 꺼놓는다.

5.3.4. UI 나 스프라이트 텍스처는 굳이 밉맵(mip map)이 필요하지 않으므로 'Generate Mip Maps'는 끈다. 3D 객체의 경우도 리얼한 3D 묘사라든가 품질적으로 필요한 부분이 있는게 아니라면 꺼놓는게 빌드/로드 사이즈를 줄일 수 있다.

5.3.5. 아래쪽의 'Max Size' 설정은 런타임의 텍스처 사이즈의 상한이다. 원본이 아무리 커도 이 값보다 더 큰 텍스처는 만들지 않게 된다. 만약 더 큰 텍스처를 포함해야 하는 경우는 이 값을 원하는 값보다 더 큰 수로 설정해야 한다.

5.3.6. 'Format' 설정은 구체적으로 런타임의 텍스처 형식을 결정하는 중요한 설정으로, 이 값에 따라 압축이나 16 비트 텍스처 등을 선택할 수 있다. 이 항목에 모든 옵션이 보이지 않는다면 텍스처 타입을 'Advanced' 이외로 설정하지 않았는지 확인한다.

5.4. JPG 압축은 큰 품질 손실 없이 이미지 파일 사이즈를 줄일 수 있으나(런타임 사이즈는 변함 없다), 텍스처 포맷에 JPG 옵션이 없으므로, JPG 를 원본 포맷 그대로 패키징할 수는 없다. 따라서, 다운로드하여 로컬 파일 시스템에 저장하든가, /StreamingAssets 디렉토리를 이용한다. /StreamingAssets 의 파일들은 파일 포맷 변경 없이 패키지에 포함되므로, 로컬의 다른 파일들과 동일하게 파일 I/O 를 이용하여 읽을 수 있다. 그렇게 읽어들인 JPG 파일의 바이너리 데이터를 Texture2D.LoadImage() 메소드를 이용하여 텍스처 애셋에 로드하면 된다.

5.5. JPG 나 ETC1 같은 압축 포맷은 알파 채널을 지원하지 않기 때문에, UI 처럼 다량의 반투명 이미지를 포함하는 텍스처의 경우 이러한 압축 포맷을 선택하려면 RGB 와 알파 채널을 분리하여 로드해야 한다.
우선, RGB 만으로 이미지 파일 1 을 만들고, 알파를 단색 채널로 변경하여 또 다른 이미지 파일 2 를 만든다. 이 과정은 포토샵 같은 그래픽 툴을 이용하거나, 유니티 에디터 상에서 간단한 유틸리티를 작성하여 수행할 수 있다. 그런 다음 2 개의 이미지 파일을 2 개의 텍스처로 로드하여, 프래그먼트 쉐이더 단계에서 이미지 파일 2 의 단색 컬러값을 알파값으로 블렌딩하는 쉐이더를 작성한다. 이 쉐이더를 해당 이미지를 사용하는 객체의 머테리얼에 할당하면, 이제 2 개의 텍스처를 각각 압축하거나 JPG 포맷으로 사용할 수 있다.
물론, 이 방식은 1 개의 원본 파일을 2 개로 나누게 되므로 언듯 보기에 용량이 늘어날 거 같으나, 압축에 의한 이득이 오버헤드를 상회하므로 충분히 사이즈를 줄일 수 있다. 또, 알파 채널 이미지의 경우는 원본과 동일한 사이즈 대신 1/2 로도 충분한 경우가 많으므로 더더욱 용량을 줄일 수 있다.

6.사운드

사운드는 최적화에서 간과하기 쉬우나 이미지와 마찬가지 아날로그 데이터를 샘플링한 애셋이므로 포맷이나 옵션에 따라 큰 폭의 최적화 효과를 볼 수 있다.

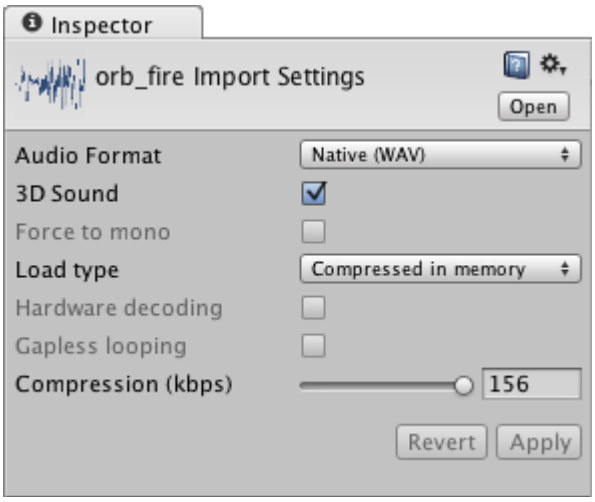

그림 14

6.1.원본 음원 데이터가 스테레오인지 보고, 필요없다면 모노 포맷으로 변경하여 다시 제작한다.

6.2.임포트의 'Audio Format'은 성능 때문에 어쩔 수 없거나 품질 상 필요한 경우가 아니라면 'Compressed'로 한다.

6.3.'Load Type'은 성능과 메모리에 영향을 주는 옵션으로, 짧은 단순 효과음이라면 'Decompress on load', 성우 대사 같은 음성이라면 'Compressed in memory', 계속 반복 플레이되는 긴 배경음악이라면 'Stream from disc' 정도 골라주면 된다.

6.4.품질을 손해 보지 않는 범위에서 비트레이트 'Compression(kbps)' 값은 가장 작은 것을 선택한다. 50 ~ 90 kbps 정도라도 소리에 따라서는 별 차이가 없다.

7.폰트

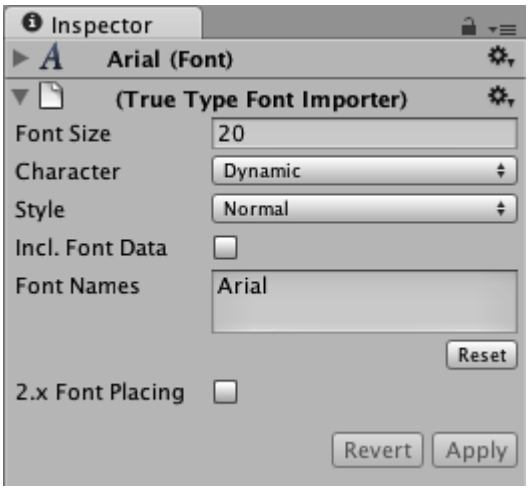

그림 15

7.1.'Character' 항목이 'Dynamic'인지 확인한다. 'Dynamic'으로 할 수 없다면 다른 폰트 솔루션을 찾아본다.

7.2.특정 폰트를 디자인 품질 때문에 사용할 계획이 아니라면 'Incl. Font Data'는 끄도록 한다. 꺼져있는 상태에선 자동으로 시스템에 설치돼 있는 폰트를 사용하게 되며, 따로 폰트 파일을 패키지에 포함하지 않게 되므로 빌드 사이즈에 이득을 볼 수 있다.

8.스크립트와 기타

보통 실행 파일 크기는 안드로이드를 비롯한 다른 시스템에선 문제되지 않으나, iOS 에선 AOT (Ahead Of Time) 컴파일과 스태틱 링크 때문에 C# 코드의 품질/최적화 또한 빌드 사이즈에 큰 부분을 차지하게 된다. 사용하지 않는 소스는 반드시 제거하고 가능한 적은 코드를 짜도록 초기부터 노력할 필요가 있다. 성능적으로도 모바일에서는, 데스크톱에 비해 메모리가 항상 병목이므로 메모리 할당(new 연산)이나 코드 사이즈가 적은 것이 스크립트의 실행 속도 또한 빠르게 할 수 있다.

8.1. 사용하지 않는 소스는 나중을 위해 두지 말고, 언제 어느 때든 필요없어진 시점에 애셋에서 제거하도록 한다. 프로젝트에 다른 백업 디렉토리를 두고 거기다 모아둘지언정, 나중에 골라내려고 생각하면 괜히 일정만 급해진다.

8.2. ConditionalAttribute 를 이용해 디버그 코드를 제거하는 경우, Conditional 에서 지정한 선언자가 플레이어 설정의 'Scripting Define Symbols'에 남아있진 않은지, 소스 상의 #define 구문을 지웠는지 확인한다.

8.3. 제네릭 컬렉션을 사용할 경우 엘리먼트 타입에 밸류 타입은 가급적 피한다. 이는 iOS 에서 AOT 컴파일 시 실행 파일의 빌드 사이즈를 늘릴 수 있기 때문이다. 정 밸류 타입 제네릭 컬렉션이 필요한 경우엔, 밸류 타입 어레이(array)를 먼저 고려해보고, 그래도 안될때만 컬렉션 클래스를 사용한다.

8.4. Reflection 을 이용한 런타임 모듈 로드나, 클래스 생성, 스크립트 컴파일 기능 같은 동적 언어 기능은, iOS 의 AOT 컴파일 시에는 동작하지 않으므로 사용하지 않도록 한다. Reflection 을 통한 필드나 메소드의 동적 호출은 가능하나, 이 경우에도 해당 클래스가 컴파일 타임에 전부 다 포함돼있어야 한다.

8.5. 시스템 라이브러리라 해도 클래스는 이것저것 여러 종류를 사용하지 말고, 필요한 것만 정해서 그 이외의 클래스나 외부 라이브러리는 실험적으로라도 도입하지 않도록 한다. 최적화 시 닷넷 라이브러리 설정이나 스트립핑 설정에 의해 사용하지 못하는 경우가 발생할 수 있다.

8.6. 새로운 클래스나 외부 라이브러리는 아래 호환성 링크에서 최적화 수준에 따라 사용할 수 있는 것들인지 항상 확인하도록 한다.
http://docs.unity3d.com/410/Documentation/ScriptReference/MonoCompatibility.html

(끝)