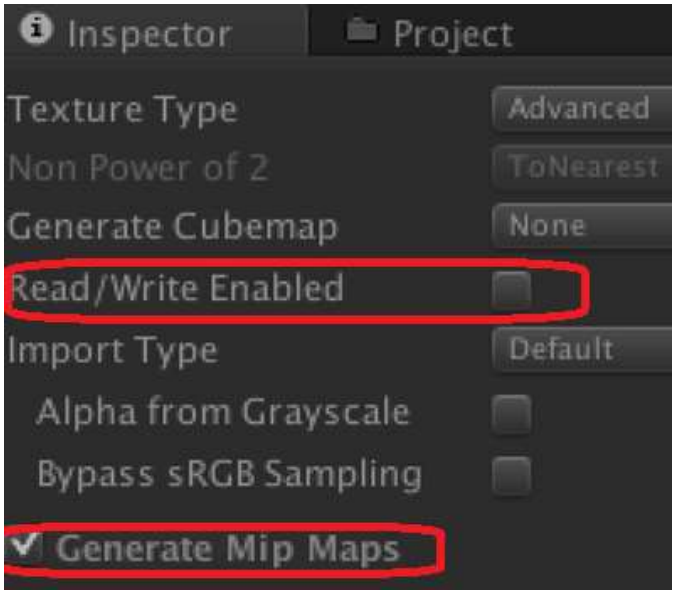
Unity* 設定のヒント: メモリー、オーディオ、テクスチャー

この記事は、インテル® デベロッパー・ゾーンに公開されている「Unity Configuration Tips: Memory, Audio, and Textures」の日本語参考訳です。

ここでは、インテルのビジュアル・コンピューティングのアプリケーション・エンジニアである Steve Hughes とともに、メモリーの最適化およびテクスチャーに関するいくつかのヒントとコツについて簡単に紹介します。詳細は、Steve と Cristiano の記事 (英語) (紹介されているヒントの多くは Windows* と Android* にも適用可能です) および「Tips for using Unity* with the Intel® RealSense™ SDK」(英語) を参照してください。

テクスチャーに関するヒント

- 必要な場合を除き、テクスチャーにミップマップを使用しないようにします。
- 圧縮テクスチャーを使用します。
- 必要な場合を除き、テクスチャーを読み取り可能 (CPU R/W) に設定しないようにします。そうすることで、余分な CPU コピー、マーカーなどを回避できます。
- 画面解像度に応じてテクスチャーのサイズを減らします。
- Android* および小さな画面:
    - モデルでトップの MIP をスキップします ([Quality Settings] > [Rendering] > [Texture Quality])。
    - アルファチャネル (非圧縮) と RGB チャネル (圧縮) を切り離します。



デザインに関するヒント

- 通常のマップのスケールは、デフューズマップと 1:1 である必要はありません。
- アニメーションから余分なキーフレームを削除します (グラフィックの変更のみをリフレッシュします)。

メッシュモデルに関するヒント

- メッシュから使用していないチャネルを削除します (Player 設定にある Optimized Mesh Data を有効にします)。
- RAM 容量が低いシステムでは、詳細なメッシュに対して Quality で maxLOD を使用するように設定します。

Unity* でのグラフィックスの最適化については、「How To Plan Optimizations with Unity」(英語) を参照してください。

オーディオに関するヒント

- オーディオ再生には 200K バッファーを利用します。
    - 非圧縮のショートクリップをバッファーにロードします。
    - または、AudioClip.GetData を利用して圧縮し、圧縮したものは解放します。

メモリーの最適化に関するヒント

- RAM の使用状況を確認する方法は 2 つあります。
    - Profiler.GetRuntimeMemorySize() は RAM 使用量が高いオブジェクトの特定に役立ちます (デバッグログに出力することを推奨します)。
    - Mono CIL (共通中間言語) で 3 つの割り当てタイプ (Newarr、Newobj、Box) を確認します。
- 起動時に (アセットの使用履歴に応じて) メモリー空間を割り当てて、手動でヒープを拡張します。
- 実行中に必要になった場合は、Resources.UnloadAsset (アセットを参照する必要がある場合) または Resources.UnloadUnusedAssets (アセットを参照する必要がない場合) を呼び出します。
- Unity* の自動ガベージ・コレクションは、通常、ヒープが一杯の場合または空き容量が足りない場合にのみ呼び出されるため、レベルのロード (またはタイマーへの配置) の前後に (System.GC..Collect) を呼び出すか、切り替え時にクリーンアップすることを検討してください。
- 大量の隠匿された割り当てを含む関数は使用しないようにします。
- クラス (ヒープに格納される) の代わりに構造体 (スタックに格納される) を使用してメモリーの断片化を回避します。
- 列挙子は RAM を割り当てます。(Foreach) はコードブロックに再処理され、このコードブロックも列挙子を割り当てます。
- 同様の理由から、匿名メソッドとラムダも避けるべきです。
- アセットをストリームして RAM の使用量を減らします。ビルド時に関連するアセットをアセットバンドルにまとめて、実行時にストリームします。(次のシーンまたは次のアセット・バンドル・ストリームの前に必ず解放してください。)
- 文字列を一緒に追加しないようにします。フレームごとに文字列を操作しないで、代わりにトラップを追加して、値が変更した場合のみ更新します。
- StringBuilder クラスを使用して文字列を作成します。ループの反復ごとにダンプと再割り当てが生じるため、文字列を返す関数に配列を渡さないようにします。
- 複数回新規作成する代わりに、次のようにアセット・プール・クラスを作成して、オブジェクトを格納します。

```
MyClass m - poolOfMyClass
// 処理を実行
poolOfMyClass.Store(m);
```

参考資料 (英語): この記事の一部は、次の資料を参考にしました。

- Unity3.com 上のメモリー関連のリソース
- Andrew Frey - Reducing Memory Usage In UNITY, C# AND .NET/MONO
- Gamasutra: C# Memory Management for Unity Developers (part 1 of 3)

ここでは概要を紹介しました。詳しくは、以下の記事を参照してください。

- https://software.intel.com/en-us/node/542152 (英語) (紹介されているヒントの多くは Windows* と Android* にも適用可能です)
- Tips for using Unity* with the Intel® RealSense™ SDK (英語)

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください