DirectXTK関連（Commonstateってなんぞ）

☒ 主な機能

CommonStates は以下のような「よくある組み合わせの状態オブジェクト」を提供します。 GitHub+1

- ブレンド状態（BlendState）
  - `Opaque()`：ブレンドなし（不透明）
  - `AlphaBlend()`：アルファ付き（通常の透過）
  - `Additive()`：加算ブレンド（光・パーティクル用）
  - `NonPremultiplied()`：プレマルチプライドでないアルファ用
- 深度／ステンシル状態（DepthStencilState）
  - `DepthDefault()`：通常の深度テスト＆書き込み
  - `DepthNone()`：深度テスト・書き込みとも無効
  - `DepthRead()`：深度確認はするが深度バッファ更新しない／透過用など
  - `DepthReverseZ()`／`DepthReadReverseZ()`：反転Zバッファ使用時用
- ラスタライザ状態（RasterizerState）
  - `CullNone()`：カリングなし（両面描画）
  - `CullClockwise()`, `CullCounterClockwise()`：カリング方向指定
  - `Wireframe()`：ワイヤーフレームモード
- サンプラ状態（SamplerState）
  - `PointWrap()`, `PointClamp()`, `LinearWrap()`, `LinearClamp()`, `AnisotropicWrap()`, `AnisotropicClamp()`：テクスチャサンプラの典型

☒ 使い方（例）

```
std::unique_ptr<DirectX::CommonStates> states =
    std::make_unique<DirectX::CommonStates>(device);

deviceContext->OMSetBlendState(states->AlphaBlend(), nullptr, 0xFFFFFFFF);
deviceContext->OMSetDepthStencilState(states->DepthDefault(), 0);
deviceContext->RSSetState(states->CullCounterClockwise());
auto sampler = states->LinearWrap();
deviceContext->PSSetSamplers(0, 1, &sampler);
```

このように、状態オブジェクトを毎回自作せず「典型パターン」を `states` から取得して使えます。 GitHub

☒ なぜ使うといいか

- 状態オブジェクト（BlendState 等）の設定を毎回細かく書くと煩雑。CommonStates によって手軽に済む。
- 状態オブジェクトを重複して生成するのを防げる。状態生成は比較的コストがあるため。
- 読みやすく、間違えにくい。例えば「透過用に深度書き込みを止める」など典型パターンが明確。
- プロジェクトが大きくなっても「この状態セットを使えばOK」という共通の仕様にできる。

⚠ 注意すべきこと

- CommonStates が提供する「典型の状態」がすべてのケースに最適とは限らない。カスタムなラスタ／ブレンド／深度設定が必要な場合は自分で構築する必要あり。
- 状態オブジェクトを頻繁に切り替えるとパフォーマンスに影響出る可能性あり。CommonStates とは別に描画順／バッチング／状態切り替えを整理するべき。
- サンプラ状態や深度モードを変更することで、既存の描画が崩れる可能性あり（例：カリングなし＋両面描画＝意図しない裏面が見える、深度無効＝描画順が重要になる）。
- ライブラリをリンク／設定ミスすると、「CommonStates の関数が未定義（Linker error）」になることが実際に質問として出ています

> ℹ よく使う設定集

`states` は `std::make_unique<CommonStates>(device)` 済み前提です。

---

1) 不透明3D（デフォルト）

- 用途：普通の3Dモデル（不透明）

```
ctx->OMSetBlendState(states->Opaque(), nullptr, 0xFFFFFFFF);
ctx->OMSetDepthStencilState(states->DepthDefault(), 0);
ctx->RSSetState(states->CullCounterClockwise()); // 右手系ならCCW, 左手系なら適宜
```

- メモ：最初はこれ。描画順の考慮が少なくて済む。

---

2) 透明（アルファブレンド）

- 用途：ガラス/フェンス/半透明UI 等

```
ctx->OMSetBlendState(states->AlphaBlend(), nullptr, 0xFFFFFFFF);
ctx->OMSetDepthStencilState(states->DepthRead(), 0); // 深度テストON/書き込みOFF
ctx->RSSetState(states->CullNone()); // 両面必要なら
```

- メモ：**後ろから前**の順に描く。深度書き込みOFFがコツ。

---

3) 加算（パーティクル/発光）

- 用途：火花・魔法・グローっぽいもの

```
ctx->OMSetBlendState(states->Additive(), nullptr, 0xFFFFFFFF);
ctx->OMSetDepthStencilState(states->DepthRead(), 0);
ctx->RSSetState(states->CullNone());
```

- メモ：重なるほど明るくなる。順番は透明同様、奥→手前。

---

4) 2D/UI/スプライト

- 用途：テクスチャをそのまま貼るUI/スプライト

```
ctx->OMSetBlendState(states->AlphaBlend(), nullptr, 0xFFFFFFFF);
ctx->OMSetDepthStencilState(states->DepthNone(), 0);
ctx->RSSetState(states->CullNone());
auto smp = states->LinearClamp(); // UIはClampが無難
ctx->PSSetSamplers(0, 1, &smp);
```

- メモ：**DepthNone** にしておくのが楽。拡大縮小が多いなら Linear。

---

5) スカイボックス

- 用途：立方体の内側に空を描く

```
ctx->OMSetBlendState(states->Opaque(), nullptr, 0xFFFFFFFF);
ctx->OMSetDepthStencilState(states->DepthRead(), 0); // 背景なので書かない
ctx->RSSetState(states->CullNone()); // 内側を描くため
```

- メモ：ビュー行列の平行移動成分を消すのも忘れずに。

---

6) デバッグ（ワイヤーフレーム）

- 用途：ジオメトリ確認

```
ctx->OMSetBlendState(states->Opaque(), nullptr, 0xFFFFFFFF);
ctx->OMSetDepthStencilState(states->DepthDefault(), 0);
ctx->RSSetState(states->Wireframe());
```

---

7) シャドウ用（深度のみパスの基本）

- 用途：単純な深度プリパスや影マップ（※深度バイアスは別途）

```
ctx->OMSetBlendState(states->Opaque(), nullptr, 0xFFFFFFFF);
ctx->OMSetDepthStencilState(states->DepthDefault(), 0);
ctx->RSSetState(states->CullCounterClockwise());
// ピクセルシェーダを外す or 深度書き込み専用PSに
```

- メモ：CommonStates には深度バイアスは含まれないので、必要なら自作 RasterizerState を用意。

---

8) ポストプロセス（フルスクリーン）

- 用途：トーンマップ/ブラーなど

```
ctx->OMSetBlendState(states->Opaque(), nullptr, 0xFFFFFFFF);
ctx->OMSetDepthStencilState(states->DepthNone(), 0);
ctx->RSSetState(states->CullNone());
auto smp = states->LinearClamp(); // スクリーンスペースはClampが基本
ctx->PSSetSamplers(0, 1, &smp);
```

---

9) サンプラの目安（用途別）

- ドット絵/ピクセルパーフェクト：`PointClamp()` / `PointWrap()`
- 一般/テクスチャ：`LinearWrap()`（タイル） or `LinearClamp()`（UI/ポスト）
- 高品質リニア：`AnisotropicWrap()` / `AnisotropicClamp()`（斜めに見る床など）

---

10) Reverse-Z を使う場合（上級）

- 用途：遠距離のZ精度改善（プロジェクションと深度比較を反転）

```
ctx->OMSetDepthStencilState(states->DepthReverseZ(), 0);       // 不透明
// or 透過時
ctx->OMSetDepthStencilState(states->DepthReadReverseZ(), 0);
```

- メモ：**射影行列のZも反転**が必要（アプリ側対応）。慣れてからでOK。

使い分けのコツ（超要約）

- **不透明は DepthDefault / Opaque、透過は AlphaBlend / DepthRead**
- **UI/ポストは DepthNone / CullNone**
- **粒子は Additive or AlphaBlend**（内容で使い分け）
- **スカイボックスは DepthRead / CullNone**
- サンプラは **UI=Clamp、タイル=Wrap、品質=Anisotropic**

Revision #1
★Created 26 October 2025 17:19:26 by youe2
Updated 17 June 2026 02:32:06 by youe2