GXLib（仮称） ― DirectX 12 ゲームフレームワーク 完全計画書

> **DXライブラリ完全上位互換 + モダンレンダリング + XML-GUI + ポストエフェクト標準搭載**

---

0. プロジェクト概要

0.1 動機と目的

DXライブラリは教育・ホビー向けとして広く使われているが、以下の根本的な問題を抱えている：

- DirectX 9/11 止まりで DirectX 12 に非対応
- 描画表現が貧弱でプロレベルのビジュアルに到達できない
- ポストエフェクトが標準搭載されておらず全て自前実装が必要
- 描画レイヤー管理の仕組みがなく、描画順制御が煩雑
- GUIシステムが存在しない

本プロジェクトでは、DXライブラリの全機能を網羅しつつ、上記の問題を根本解決した **DirectX 12 ベースの完全上位互換フレームワーク** を構築する。

0.2 設計理念

1. **DXライブラリ互換**: 既存の DXLib ユーザーが違和感なく移行できる API 設計
2. **モダンレンダリング**: PBR、HDR、ポストエフェクトを標準搭載
3. **レイヤードアーキテクチャ**: 低レベル API と高レベル API の両方を提供
4. **宣言的 GUI**: XML/スクリプトによる UI 定義
5. **拡張性**: プラグインやカスタムシェーダーで自由に拡張可能

0.3 技術スタック

項目	選定
グラフィックス API	DirectX 12 (D3D12)

シェーダー言語	HLSL (Shader Model 6.x)
シェーダーコンパイラ	DXC (DirectX Shader Compiler)
ビルドシステム	CMake + vcpkg
言語	C++20 (モジュール対応準備)
オーディオ	XAudio2 / WASAPI
入力	XInput + Raw Input + DirectInput(後方互換)
物理	内製 2D + Jolt Physics(3D、オプション)
GUI記述	XML + CSS-like スタイルシート
スクリプト	Lua（オプション）
テスト	Google Test + GPU ベースの回帰テスト

0.4 ゴール定義

完成条件 = 以下の全てを満たすこと：

1. DXライブラリの全 API カテゴリを網羅（後述の機能マッピング表を100%カバー）
2. ポストエフェクトパイプラインが標準搭載され、設定ファイルで ON/OFF 可能
3. 描画レイヤーシステムが動作し、レイヤー単位でのブレンド・エフェクトが可能
4. XML ベースの GUI システムでメニュー・HUD・エディタ UI が構築可能
5. サンプルプロジェクト群（2Dゲーム、3Dゲーム、GUIデモ）が動作する

---

1. DXライブラリ機能マッピング表

DXライブラリの全機能カテゴリを洗い出し、本フレームワークでの対応方針を定義する。

1.1 システム系

DXLib 機能	対応関数例	GXLib 対応方針
ウィンドウ管理	`DxLib_Init`, `SetGraphMode`, `ChangeWindowMode`	Win32 ウィンドウ + DXGI SwapChain

メッセージ処理	`ProcessMessage`	内部メッセージループ + イベントコールバック
フルスクリーン切替	`ChangeWindowMode`	DXGI フルスクリーン + ボーダーレス
解像度変更	`SetGraphMode`	動的スワップチェーンリサイズ
DPI対応	（なし）	**新規**: Per-Monitor DPI V2 対応
マルチウィンドウ	（なし）	**新規**: マルチウィンドウレンダリング
タイマー	`GetNowCount`, `GetNowHiPerformanceCount`	`QueryPerformanceCounter` ベース
FPS制御	`SetWaitVSyncFlag`	VSync + フレームレートリミッター
ログ出力	`printfDx`, `ErrorLogAdd`	構造化ログシステム (レベル別、ファイル出力)

1.2 描画系 — 2D

DXLib 機能	対応関数例	GXLib 対応方針
画像読込・描画	`LoadGraph`, `DrawGraph`	テクスチャ管理 + スプライトバッチ
画像分割読込	`LoadDivGraph`	スプライトシート + アトラスパッカー
画像回転拡大	`DrawRotaGraph`, `DrawExtendGraph`	Transform2D 指定描画
画像ブレンド	`SetDrawBlendMode`	ブレンドステート (加算、乗算、αなど)
画像輝度設定	`SetDrawBright`	シェーダーパラメータ / カラーマスク
図形描画	`DrawLine`, `DrawBox`, `DrawCircle`, `DrawTriangle`	プリミティブバッチレンダラー
アンチエイリアス図形	`DrawLineAA`, `DrawCircleAA`	MSAA + シェーダーベース AA
ピクセル操作	`GetPixelSoftImage`,	CPU側ソフトイメージ + Readback

	`DrawPixelSoftImage`	
グラフィックフィルタ	`GraphFilter` (モノ、ガウス、明度等)	**ポストエフェクトパイプラインで代替**
描画先変更	`SetDrawScreen`, `MakeScreen`	RenderTarget システム
描画レイヤー	（なし）	**新規**: レイヤースタック (Z-order、ブレンド、エフェクト)
スプライトアニメーション	（なし）	**新規**: アニメーションコントローラー

1.3 描画系 — 3D

DXLib 機能	対応関数例	GXLib 対応方針
3Dモデル読込	`MV1LoadModel`	glTF / FBX / OBJ ローダー
モデル描画	`MV1DrawModel`	メッシュレンダラー + インスタンシング
モデルアニメーション	`MV1AttachAnim`, `MV1SetAttachAnimTime`	スケルタルアニメーション + ブレンドツリー
モデル衝突判定	`MV1CollCheck_Sphere`, `MV1CollCheck_Line`	メッシュコリジョン
カメラ制御	`SetCameraPositionAndTarget_UpVecY`	Camera コンポーネント (Perspective / Ortho)
ライティング	`SetLightDirection`, `SetLightDifColor`	**PBR ライティング** (Directional, Point, Spot, Area)
マテリアル	`MV1SetMaterialDifColor`	**PBR マテリアル** (Albedo, Normal, Metallic, Roughness, AO)
シャドウマップ	`SetShadowMapDrawArea`	**CSM (Cascaded Shadow Maps)** + PCF/VSM
フォグ	`SetFogEnable`, `SetFogColor`	ボリューメトリックフォグ / 距離フォグ
3D図形	`DrawSphere3D`, `DrawCone3D`	プリミティブメッシュ生成

Zバッファ	`SetUseZBuffer3D`	深度ステート制御
環境マップ	（なし）	**新規**: キューブマップ / IBL 反射
PBR	（なし）	**新規**: Cook-Torrance BRDF
スカイボックス	（なし）	**新規**: HDR スカイボックス / プロシージャル空
地形	（なし）	**新規**: ハイトマップ地形 + LOD

1.4 ポストエフェクト（全て新規）

エフェクト	実装方式
Bloom	Dual Kawase / ガウシアン ダウンサンプリング
Tonemapping	ACES / Reinhard / Uncharted 2 / AgX
HDR	浮動小数テクスチャ (R16G16B16A16_FLOAT)
SSAO	GTAO / HBAO
被写界深度 (DoF)	Bokeh DoF (六角形/円形)
モーションブラー	Per-Object / Camera ベース
カラーグレーディング	3D LUT
FXAA / TAA	FXAA 3.11 / TAA
ビネット	シェーダーベース
色収差	シェーダーベース
スクリーンスペースリフレクション	SSR (Hi-Z trace)
ボリューメトリックライト	レイマーチング
輪郭線 (トゥーン)	Sobel / 法線・深度エッジ検出

1.5 シェーダー

DXLib 機能	対応関数例	GXLib 対応方針

頂点シェーダー	`LoadVertexShader`	HLSL SM 6.x コンパイル + ホットリロード
ピクセルシェーダー	`LoadPixelShader`	同上
定数バッファ	`SetShaderConstantReg`	CBV 自動バインド
シェーダー描画	`SetUseVertexShader`	マテリアルシステム統合
コンピュートシェーダー	（なし）	**新規**: CS パイプライン
シェーダーホットリロード	（なし）	**新規**: ファイル監視による即時反映
シェーダーバリアント	（なし）	**新規**: プリプロセッサ定義による分岐管理

1.6 サウンド

DXLib 機能	対応関数例	GXLib 対応方針
サウンド読込・再生	`LoadSoundMem`, `PlaySoundMem`	XAudio2 ベース
BGM ストリーミング	`PlayMusic`	ストリーミングデコード
音量制御	`ChangeVolumeSoundMem`	デシベルベース音量制御
パン制御	`ChangePanSoundMem`	ステレオパン
再生速度	`SetFrequencySoundMem`	ピッチシフト
3Dサウンド	`Set3DPositionSoundMem`	**3D 空間音響** (HRTF オプション)
対応フォーマット	WAV, OGG, MP3	WAV, OGG, MP3, FLAC, OPUS
サウンドミキサー	（なし）	**新規**: バス・エフェクトチェーン
リアルタイムエフェクト	（なし）	**新規**: リバーブ、EQ、コンプレッサー

1.7 入力

DXLib 機能	対応関数例	GXLib 対応方針
	`CheckHitKey`,	

キーボード	`GetHitKeyStateAll`	Raw Input
マウス	`GetMousePoint` , `GetMouseInput`	Raw Input + カーソル管理
マウスホイール	`GetMouseWheelRotVol`	WM_MOUSEWHEEL
ジョイパッド	`GetJoypadInputState` , `GetJoypadAnalogInput`	XInput + DirectInput フォールバック
ジョイパッド振動	`StartJoypadVibration`	XInput バイブレーション
タッチ	`GetTouchInputNum`	WM_TOUCH / WM_POINTER
入力マッピング	（なし）	**新規**: アクションマップ (設定ファイルで再マップ可能)
入力バッファリング	（なし）	**新規**: 格闘ゲーム向けコマンドバッファ
デッドゾーン設定	（なし）	**新規**: アナログスティック デッドゾーン

1.8 文字描画

DXLib 機能	対応関数例	GXLib 対応方針
文字列描画	`DrawString` , `DrawFormatString`	**SDF フォントレンダリング**
フォント作成	`CreateFontToHandle`	DirectWrite + フォントアトラス
文字列幅取得	`GetDrawStringWidth`	テキスト計測 API
文字コード	`SetUseCharCodeFormat`	UTF-8 / UTF-16
リッチテキスト	（なし）	**新規**: カラー、サイズ混在テキスト
テキストレイアウト	（なし）	**新規**: ワードラップ、行間、カーニング
ビットマップフォント	（なし）	**新規**: BMFont 形式対応

1.9 ネットワーク

DXLib 機能	対応関数例	GXLib 対応方針
TCP接続	`ConnectNetWork`, `NetWorkSend`	Winsock2 非同期 TCP
UDP	`MakeUDPSocket`, `NetWorkSendUDP`	Winsock2 UDP
HTTP	`GetHTTP`, `GetHTTPRes`	WinHTTP / libcurl
WebSocket	（なし）	**新規**: WebSocket クライアント
非同期IO	（なし）	**新規**: IOCP ベース非同期

1.10 ファイル・アーカイブ

DXLib 機能	対応関数例	GXLib 対応方針
ファイル読み書き	`FileRead_open`, `FileRead_gets`	std::filesystem + 非同期IO
DXA アーカイブ	`SetDXArchiveExtension`	**カスタムアーカイブ** (AES暗号化対応)
メモリ上読込	`CreateGraphFromMem`	メモリストリーム対応
アセットホットリロード	（なし）	**新規**: ファイル監視による即時反映
非同期アセット読込	（なし）	**新規**: バックグラウンドロード + プログレス

1.11 算術・衝突判定

DXLib 機能	対応関数例	GXLib 対応方針
ベクトル演算	`VGet`, `VAdd`, `VCross`	SIMD 最適化 数学ライブラリ (DirectXMath ラップ)
行列演算	`MGetIdent`, `MMult`	Matrix4x4 クラス
衝突判定	`HitCheck_Sphere_Sphere`, `HitCheck_Line_Triangle`	衝突判定ユーティリティ

2D物理	（なし）	**新規**: 簡易 2D 物理エンジン (AABB, Circle, Polygon)
空間分割	（なし）	**新規**: Quadtree / Octree / BVH

1.12 動画

DXLib 機能	対応関数例	GXLib 対応方針
動画再生	`PlayMovie`, `OpenMovieToGraph`	Media Foundation デコード → テクスチャ
動画フレーム取得	`SeekMovieToGraph`, `TellMovieToGraph`	フレーム単位シーク

1.13 マスク

DXLib 機能	対応関数例	GXLib 対応方針
マスク描画	`CreateMaskScreen`, `DrawMask`	ステンシルバッファ + マスクテクスチャ
マスク図形	`DrawFillMask`, `DrawCircleMask`	ステンシル描画

---

2. 新規システム詳細設計

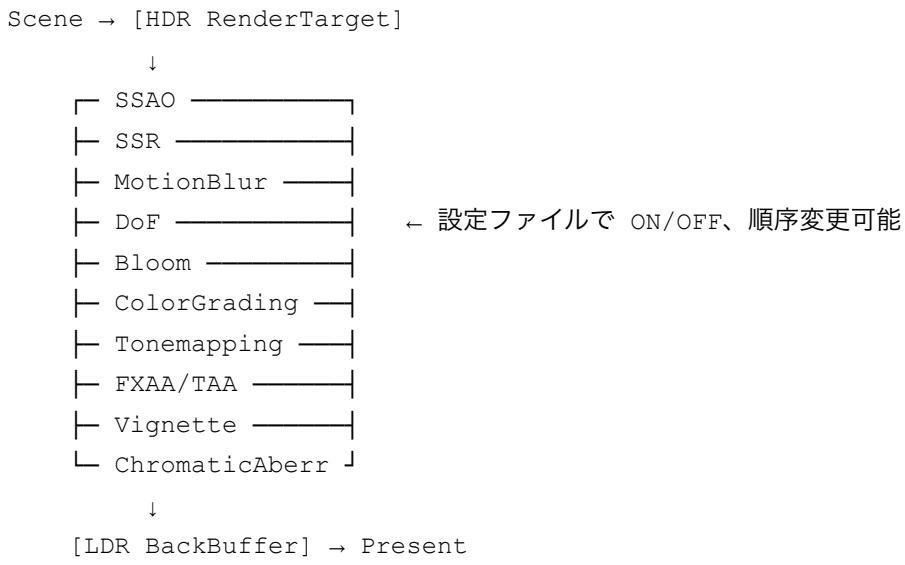
2.1 描画レイヤーシステム

```
┌─────────────────────────────────────┐
│           最終合成出力              │
├─────────────────────────────────────┤
│  Layer: UI (Z: 1000)                │ ← ポストエフェクト適用外
│  Layer: HUD (Z: 900)                │ ← ポストエフェクト適用外
│  Layer: PostFX                      │ ← ポストエフェクトパイプライン
│  Layer: Particles (Z: 500)          │
│  Layer: Characters (Z: 400)         │
│  Layer: World (Z: 100)              │ ← PBR レンダリング
│  Layer: Background (Z: 0)           │
│  Layer: Skybox (Z: -1000)           │
└─────────────────────────────────────┘
```

レイヤーの機能:

- 個別の RenderTarget を保持
- レイヤー単位のブレンドモード設定（通常、加算、乗算、スクリーン...）
- レイヤー単位の不透明度制御
- レイヤー単位のポストエフェクト適用可否
- レイヤーのカメラ独立設定（パララックススクロール等）
- レイヤーのソートモード設定（Z-sort, Y-sort, 挿入順）

2.2 ポストエフェクトパイプライン



設定例 (JSON):

```
{
  "postEffects": {
    "bloom": { "enabled": true, "threshold": 1.0, "intensity": 0.8, "radius": 4 }
    "tonemapping": { "enabled": true, "operator": "ACES", "exposure": 1.2 },
    "ssao": { "enabled": true, "radius": 0.5, "bias": 0.025, "samples": 32 },
    "dof": { "enabled": false },
    "fxaa": { "enabled": true }
  }
}
```

2.3 XML-GUI システム

設計コンセプト: Web のような宣言的 UI を C++ ゲームで

XML レイアウト定義

```
<!-- ui/main_menu.xml -->
<Window id="mainMenu" width="100%" height="100%">
  <Panel id="centerPanel" layout="vertical" align="center" valign="center"
         background="#00000080" padding="20" cornerRadius="8">

    <Text id="title" text="My Game" fontSize="48" fontFamily="GameFont"
          color="#FFFFFF" shadow="2,2,#000000" />

    <Spacer height="40" />

    <Button id="btnStart" width="300" height="60" text="ゲーム開始"
            class="menuButton" onClick="onStartGame" />
    <Button id="btnOption" width="300" height="60" text="オプション"
            class="menuButton" onClick="onOpenOptions" />
    <Button id="btnExit" width="300" height="60" text="終了"
            class="menuButton" onClick="onExit" />
  </Panel>
</Window>
```

CSS-like スタイルシート

```
/* ui/styles/menu.gss */
.menuButton {
    background: linear-gradient(#4A90D9, #357ABD);
    color: #FFFFFF;
    fontSize: 24;
    fontFamily: "GameFont";
    cornerRadius: 6;
    border: 2px solid #2A5A8E;
    transition: background 0.2s;
}

.menuButton:hover {
    background: linear-gradient(#5BA0E9, #4590DD);
    transform: scale(1.05);
}

.menuButton:pressed {
    background: linear-gradient(#2A5A8E, #1A4A7E);
    transform: scale(0.95);
}
```

C++ バインド

```
// GUI の読み込みと操作
auto ui = GX::GUI::Load("ui/main_menu.xml", "ui/styles/menu.gss");

// イベントバインド
ui->Bind("onStartGame", [&]() { scene.TransitionTo<GameScene>(); });
ui->Bind("onOpenOptions", [&]() { ui->Show("optionsPanel"); });
ui->Bind("onExit", [&]() { GX::System::Exit(); });

// 動的操作
ui->Find<GX::GUI::Text>("title")->SetText("Updated Title");
ui->Find<GX::GUI::Button>("btnStart")->SetEnabled(false);
```

GUI ウィジェット一覧

ウィジェット	説明
Window	ルートコンテナ
Panel	レイアウトコンテナ (horizontal/vertical/grid/absolute)
Text	テキスト表示
Button	ボタン
Image	画像表示
TextInput	テキスト入力欄
Slider	スライダー
CheckBox	チェックボックス
RadioGroup / Radio	ラジオボタン
DropDown	ドロップダウン
ListView	スクロール可能リスト
ScrollView	スクロール可能コンテナ
ProgressBar	プログレスバー
TabView	タブ切替
Dialog	モーダルダイアログ
Canvas	カスタム描画エリア

3. アーキテクチャ

3.1 モジュール構成

```
GXLib/
├── Core/                    # コアシステム
│   ├── Application.h/cpp    # アプリケーションライフサイクル
│   ├── Window.h/cpp         # ウィンドウ管理
│   ├── Timer.h/cpp          # 高精度タイマー
│   ├── Logger.h/cpp         # ログシステム
│   ├── Memory/              # メモリ管理
│   │   ├── Allocator.h      # カスタムアロケータ
│   │   ├── PoolAllocator.h  # プールアロケータ
│   │   └── StackAllocator.h # スタックアロケータ
│   ├── Event/               # イベントシステム
│   │   ├── EventBus.h       # パブリッシュ・サブスクライブ
│   │   └── Delegate.h       # 型安全コールバック
│   └── Config/              # 設定管理
│       └── ConfigManager.h  # JSON/INI 設定読み書き
│
├── Graphics/                # 描画エンジン
│   ├── Device/              # D3D12 デバイス管理
│   │   ├── GraphicsDevice.h # デバイス初期化・管理
│   │   ├── SwapChain.h      # スワップチェーン
│   │   ├── CommandQueue.h   # コマンドキュー
│   │   ├── CommandList.h    # コマンドリスト
│   │   ├── DescriptorHeap.h # デスクリプタヒープ管理
│   │   ├── Fence.h          # GPU 同期
│   │   └── GPUResource.h    # リソース基底クラス
│   │
│   ├── Pipeline/            # パイプライン管理
│   │   ├── PipelineState.h  # PSO 管理
│   │   ├── RootSignature.h  # ルートシグネチャ
│   │   ├── Shader.h         # シェーダーコンパイル・管理
│   │   └── ShaderLibrary.h  # シェーダーライブラリ + ホットリロード
│   │
│   ├── Resource/            # GPU リソース
│   │   ├── Texture.h        # テクスチャ
│   │   ├── Buffer.h         # 頂点/インデックス/定数バッファ
│   │   ├── RenderTarget.h   # レンダーターゲット
│   │   ├── DepthBuffer.h    # 深度バッファ
│   │   └── SamplerState.h   # サンプラー
```

```
│   │
│   ├── 2D/                    # 2D 描画
│   │   ├── SpriteBatch.h      # スプライトバッチ
│   │   ├── SpriteSheet.h      # スプライトシート
│   │   ├── PrimitiveBatch.h   # 図形バッチ
│   │   ├── TextRenderer.h     # テキスト描画 (SDF)
│   │   ├── FontManager.h      # フォント管理
│   │   └── Animation2D.h      # スプライトアニメーション
│   │
│   ├── 3D/                    # 3D 描画
│   │   ├── Mesh.h             # メッシュ
│   │   ├── Model.h            # 3Dモデル (glTF/FBX)
│   │   ├── SkeletalAnim.h     # スケルタルアニメーション
│   │   ├── Camera.h           # カメラ
│   │   ├── Light.h            # ライト (Directional/Point/Spot)
│   │   ├── Material.h         # PBR マテリアル
│   │   ├── ShadowMap.h        # CSM シャドウ
│   │   ├── Skybox.h           # スカイボックス
│   │   ├── Terrain.h          # 地形
│   │   └── Primitive3D.h      # 3Dプリミティブ
│   │
│   ├── Layer/                 # 描画レイヤー
│   │   ├── RenderLayer.h      # レイヤー定義
│   │   ├── LayerStack.h       # レイヤースタック管理
│   │   └── LayerCompositor.h  # レイヤー合成
│   │
│   ├── PostFX/                # ポストエフェクト
│   │   ├── PostEffectPipeline.h  # パイプライン管理
│   │   ├── PostEffect.h          # エフェクト基底
│   │   ├── Bloom.h               # Bloom
│   │   ├── Tonemapping.h         # トーンマッピング
│   │   ├── SSAO.h                # SSAO
│   │   ├── DoF.h                 # 被写界深度
│   │   ├── MotionBlur.h          # モーションブラー
│   │   ├── ColorGrading.h        # カラーグレーディング
│   │   ├── FXAA.h                # FXAA
│   │   ├── TAA.h                 # TAA
│   │   ├── Vignette.h            # ビネット
│   │   ├── ChromaticAberration.h # 色収差
│   │   ├── SSR.h                 # スクリーンスペースリフレクション
│   │   ├── VolumetricLight.h     # ボリューメトリックライト
│   │   └── OutlineEffect.h      # 輪郭線
│   │
│   └── Renderer.h             # レンダラー統合 (2D/3D/PostFX/Layer合成)
│
├── Audio/                     # オーディオ
│   ├── AudioDevice.h          # XAudio2 デバイス
```

```
│   ├── Sound.h              # サウンドリソース
│   ├── SoundPlayer.h        # 再生管理
│   ├── MusicPlayer.h        # BGM ストリーミング
│   ├── AudioMixer.h         # ミキサー・バス
│   ├── Audio3D.h            # 3D 空間音響
│   └── AudioEffect.h        # リバーブ等エフェクト
│
├── Input/                   # 入力
│   ├── Keyboard.h           # キーボード
│   ├── Mouse.h              # マウス
│   ├── Gamepad.h            # ゲームパッド (XInput + DInput)
│   ├── Touch.h              # タッチ入力
│   ├── InputManager.h       # 統合入力管理
│   └── ActionMap.h          # アクションマッピング
│
├── GUI/                     # GUI システム
│   ├── GUIManager.h         # GUI 統合管理
│   ├── GUIParser.h          # XML パーサー
│   ├── GUIStyleSheet.h      # スタイルシートエンジン
│   ├── GUIRenderer.h        # GUI 描画
│   ├── GUILayout.h          # レイアウトエンジン (Flexbox-like)
│   ├── GUIAnimation.h       # UI アニメーション・トランジション
│   ├── GUIEvent.h           # UI イベント伝搬
│   └── Widgets/             # ウィジェット
│       ├── Widget.h         # 基底ウィジェット
│       ├── Panel.h
│       ├── Button.h
│       ├── Text.h
│       ├── TextInput.h
│       ├── Image.h
│       ├── Slider.h
│       ├── CheckBox.h
│       ├── RadioButton.h
│       ├── DropDown.h
│       ├── ListView.h
│       ├── ScrollView.h
│       ├── ProgressBar.h
│       ├── TabView.h
│       ├── Dialog.h
│       └── Canvas.h
│
├── IO/                      # ファイル・ネットワーク
│   ├── FileSystem.h         # ファイルシステム抽象化
│   ├── Archive.h            # アーカイブ (暗号化対応)
│   ├── AsyncLoader.h        # 非同期アセットロード
│   ├── Network/             # ネットワーク
│   │   ├── TCPSocket.h
```

```
│   │   ├── UDPSocket.h
│   │   ├── HTTPClient.h
│   │   └── WebSocket.h
│   └── Serialization.h      # シリアライゼーション
│
├── Math/                    # 数学ライブラリ
│   ├── Vector2.h / Vector3.h / Vector4.h
│   ├── Matrix4x4.h
│   ├── Quaternion.h
│   ├── Color.h
│   ├── MathUtil.h           # Lerp, Clamp, SmoothStep 等
│   ├── Random.h             # 乱数生成
│   └── Collision/           # 衝突判定
│       ├── AABB.h
│       ├── Sphere.h
│       ├── Ray.h
│       ├── Frustum.h
│       └── CollisionUtil.h
│
├── Physics/                 # 物理（オプショナル）
│   ├── Physics2D.h          # 簡易 2D 物理
│   └── PhysicsWorld.h       # Jolt 統合 (3D)
│
├── Movie/                   # 動画再生
│   └── MoviePlayer.h        # Media Foundation ベース
│
├── Compat/                  # DXライブラリ互換レイヤー
│   ├── DxLibCompat.h        # DXLib 関数名互換 API
│   └── DxLibTypes.h         # 型変換
│
└── Utility/                 # ユーティリティ
    ├── StringUtil.h         # 文字列ユーティリティ
    ├── PathUtil.h           # パス操作
    ├── Hash.h               # ハッシュ関数
    ├── ThreadPool.h         # スレッドプール
    └── Profiler.h           # パフォーマンスプロファイラ
```

3.2 DXライブラリ互換レイヤー

既存の DXLib ユーザーが段階的に移行できるよう、互換 API を提供する：

```
// Compat/DxLibCompat.h — DXLib スタイルの関数を GXLib にマッピング
namespace DxLibCompat {
    inline int DxLib_Init()         { return GX::Application::Init(); }
    inline int DxLib_End()          { return GX::Application::Shutdown(); }
    inline int ProcessMessage()     { return GX::Application::ProcessMessages();
```

```
    inline int DrawGraph(int x, int y, int handle, int transFlag) {
        return GX::Graphics2D::DrawSprite(handle, {x, y}, transFlag);
    }
    // ... 全 DXLib 関数をマッピング
}
```

---

4. 実装フェーズ

フェーズ 0: 基盤構築（目安: 4-6 週）

ゴール: ウィンドウ表示 + D3D12 初期化 + 三角形描画

- CMake プロジェクト構成 + vcpkg 依存管理
- Win32 ウィンドウ作成・メッセージループ
- D3D12 デバイス初期化
  - Factory, Device, CommandQueue 作成
  - SwapChain 作成 (ダブルバッファリング)
  - DescriptorHeap 管理クラス
  - Fence による CPU-GPU 同期
  - CommandAllocator / CommandList 管理
- パイプラインステート基盤
  - RootSignature ビルダー
  - PSO ビルダー
  - DXC によるシェーダーコンパイル
- 三角形描画 (Hello Triangle)
- フレームタイミング・FPS制御
- ログシステム

成果物: 色付き三角形がウィンドウに描画される

フェーズ 1: 2D 描画エンジン（目安: 6-8 週）

ゴール: DXLib の 2D 描画機能を全て再現

- テクスチャ管理
  - WIC / stb_image による画像読込 (PNG, JPG, BMP, TGA, DDS)
  - テクスチャリソース管理 (アップロード、キャッシュ)
  - ミップマップ生成
- スプライトバッチ
  - 動的頂点バッファによるバッチ処理
  - DrawGraph 相当 (位置指定描画)
  - DrawRotaGraph 相当 (回転・拡大)
  - DrawRectGraph 相当 (矩形切り出し)
  - DrawExtendGraph 相当 (拡縮)
  - DrawModiGraph 相当 (自由変形)
  - ブレンドモード (None, Alpha, Add, Sub, Mul, Screen...)
  - カラーモジュレーション (SetDrawBright 相当)
- スプライトシート・アトラス
  - LoadDivGraph 相当
  - テクスチャアトラスパッカー
  - スプライトアニメーション再生
- プリミティブバッチ
  - DrawLine / DrawLineAA
  - DrawBox / DrawFillBox
  - DrawCircle / DrawCircleAA / DrawOval
  - DrawTriangle / DrawFillTriangle
  - DrawPixel
  - アンチエイリアス対応 (距離ベース AA シェーダー)
- RenderTarget

    - MakeScreen 相当 (任意サイズ RT 作成)
    - SetDrawScreen 相当 (描画先切替)
    - GetDrawScreenGraph 相当 (スクリーンキャプチャ)
- ソフトイメージ
    - CPU 側ピクセル操作
    - GPU ← → CPU 転送
- カメラ2D (ビュー行列変換)

成果物: DXLib の 2D サンプルが全て再現可能

フェーズ 2: テキスト・入力・サウンド（目安: 4-6 週）

ゴール: ゲームとして最低限遊べるインフラ

テキスト描画

- DirectWrite によるフォントラスタライズ
- SDF フォントアトラス生成
- SDF テキストレンダリングシェーダー
- DrawString / DrawFormatString 相当
- フォントハンドル管理 (CreateFontToHandle 相当)
- 文字列幅・高さ取得
- リッチテキスト (色・サイズ混在)
- ビットマップフォント (BMFont 対応)

入力

- キーボード (Raw Input)
    - CheckHitKey / GetHitKeyStateAll 相当
    - キーリピート制御
- マウス (Raw Input)

    - 位置取得、ボタン状態、ホイール
    - カーソル表示制御、クリッピング
- ゲームパッド
    - XInput 対応 (Xbox コントローラー)
    - DirectInput フォールバック (汎用パッド)
    - アナログスティック デッドゾーン
    - バイブレーション
- アクションマッピングシステム
    - 設定ファイルによるキーバインド定義
    - ランタイム再マップ

サウンド

- XAudio2 初期化
- WAV / OGG / MP3 デコード
- サウンドリソース管理 (LoadSoundMem 相当)
- 再生制御 (Play, Stop, Pause, Resume)
- 音量・パン・再生速度
- BGM ストリーミング再生
- 3D サウンド (X3DAudio)
- オーディオミキサー (マスター / BGM / SE / Voice バス)
- エフェクトチェーン (リバーブ等)

成果物: 音が鳴り、操作可能な 2D ゲームが作れる

フェーズ 3: 3D 描画エンジン（目安: 8-12 週）

ゴール: PBR ベースの 3D レンダリング

モデルローダー

- glTF 2.0 ローダー (cgltf / tinygltf)
  - メッシュ、マテリアル、テクスチャ
  - スケルタルアニメーション
  - モーフターゲット
- FBX ローダー (Assimp オプション)
- OBJ ローダー
- MV1 ローダー (DXLib 互換、オプション)

PBR レンダリング

- Cook-Torrance BRDF 実装
  - GGX 法線分布
  - Smith-GGX ジオメトリ関数
  - Fresnel (Schlick 近似)
- PBR マテリアルシステム
  - Albedo, Normal, Metallic, Roughness, AO マップ
  - Emissive マップ
- ライティング
  - Directional Light
  - Point Light (減衰付き)
  - Spot Light
  - 複数ライト対応 (Forward+ or Deferred)
- IBL (Image Based Lighting)
  - 環境キューブマップ
  - Irradiance Map
  - Pre-filtered Specular Map
  - BRDF LUT

シャドウ

- CSM (Cascaded Shadow Maps) 実装
  - 分割数設定
  - PCF フィルタリング
  - VSM オプション
- ポイントライトシャドウ (キューブマップ)
- スポットライトシャドウ

カメラ

- Perspective / Orthographic
- FPS / TPS / フリーカメラ
- カメラ振動 (スクリーンシェイク)

スカイボックス

- キューブマップスカイボックス
- HDR 環境マップ
- プロシージャル空 (Preetham / Hosek-Wilkie)

その他 3D

- フォグ (線形、指数、ボリューメトリック)
- 3D プリミティブ描画 (球、箱、円柱等)
- インスタンシング描画
- LOD システム
- 地形 (ハイトマップ + テッセレーション)

成果物: PBR lit シーンが描画でき、モデルがアニメーションする

フェーズ 4: ポストエフェクトパイプライン（目安: 6-8 週）

ゴール: 全ポストエフェクトが動作し、設定ファイルで制御可能

- PostEffectPipeline フレームワーク

    - エフェクトチェーン管理
    - 中間 RT の自動管理
    - 設定ファイル (JSON) による有効/無効切替
    - エフェクトパラメータのランタイム変更
- Bloom
    - 輝度抽出
    - Dual Kawase ダウンサンプリング
    - アップサンプリング + 合成
- Tonemapping
    - ACES Filmic
    - Reinhard
    - AgX
    - 自動露出 (Eye Adaptation)
- SSAO (GTAO)
    - 法線 + 深度からの AO 計算
    - ブラー + 適用
- 被写界深度 (DoF)
    - CoC (Circle of Confusion) 計算
    - Bokeh シミュレーション
- モーションブラー
    - Velocity Buffer 生成
    - ブラー適用
- カラーグレーディング
    - 3D LUT 適用
    - LUT ベイク・ブレンド
- アンチエイリアス

    - FXAA 3.11
    - TAA (Temporal Anti-Aliasing)
- その他
    - ビネット
    - 色収差 (Chromatic Aberration)
    - SSR (Screen Space Reflections)
    - ボリューメトリックライト
    - 輪郭線 (トゥーン向け)

成果物: 全エフェクトが ON/OFF でき、ビジュアルが劇的に向上

フェーズ 5: 描画レイヤーシステム（目安: 3-4 週）

ゴール: レイヤー単位での描画制御が完全動作

- RenderLayer クラス
    - 個別 RenderTarget
    - Z-order ソート
    - ブレンドモード設定
    - 不透明度
    - カメラ参照 (パララックス)
- LayerStack 管理
    - レイヤー追加・削除・並び替え
    - レイヤーグループ
- LayerCompositor
    - レイヤー合成シェーダー
    - レイヤー単位 PostFX 適用制御
    - マスクレイヤー
- ステンシルマスクシステム

- DXLib のマスク機能再現
- 図形マスク、テクスチャマスク

成果物: 背景 → ゲーム → UI が独立レイヤーで管理され合成される

フェーズ 6: XML-GUI システム（目安: 8-10 週）

ゴール: XML でゲーム UI が構築・操作できる

パーサー・データモデル

- XML パーサー (pugixml or RapidXML)
- スタイルシートパーサー (CSS-like)
- ウィジェットツリー構築
- プロパティ解決 (インライン > class > デフォルト)

レイアウトエンジン

- ボックスモデル (margin, border, padding)
- Flexbox ライクレイアウト (horizontal, vertical)
- Grid レイアウト
- Absolute positioning
- パーセント / ピクセル / auto サイジング
- アンカー / 整列

描画

- ウィジェット描画 (角丸矩形、ボーダー、影)
- テキスト描画統合
- 画像表示
- グラデーション背景
- 9-slice / 9-patch

インタラクション

- ヒットテスト
- イベント伝搬 (バブリング / キャプチャ)
- フォーカス管理 (Tab ナビゲーション)
- ゲームパッド UI ナビゲーション
- ドラッグ & ドロップ

アニメーション

- プロパティアニメーション (position, color, opacity, scale)
- Easing 関数
- CSS transition 相当
- ページ遷移トランジション

ウィジェット実装

- 基底 Widget クラス
- Panel, Button, Text, Image
- TextInput (IME 対応)
- Slider, CheckBox, RadioButton
- DropDown, ListView, ScrollView
- ProgressBar, TabView, Dialog
- Canvas (カスタム描画)

C++ バインディング

- イベントバインド API
- 動的ウィジェット操作 (Find, Show, Hide, SetProperty)
- データバインディング (値の双方向同期)

成果物: XML + CSS でメニュー・HUD が構築でき、C++ でイベント処理可能

フェーズ 7: ファイル・ネットワーク・動画（目安: 4-6 週）

ゴール: 残りの DXLib 機能を網羅

ファイル・アーカイブ

- ファイルシステム抽象化
  - 物理ファイル / アーカイブ の透過アクセス
  - マウントポイント
- カスタムアーカイブ
  - パッキングツール
  - AES-256 暗号化
  - 圧縮 (LZ4 / zstd)
- 非同期アセットローダー
  - バックグラウンドスレッドでのロード
  - プログレスコールバック
  - ロード完了イベント
- アセットホットリロード
  - ファイル監視 (ReadDirectoryChangesW)
  - テクスチャ / シェーダー / GUI のリロード

ネットワーク

- TCP ソケット (非同期 IOCP)
- UDP ソケット
- HTTP クライアント (WinHTTP)
- WebSocket クライアント

動画

- Media Foundation ベース動画デコード
- テクスチャへのフレーム出力
- 再生制御 (Play, Stop, Seek, Pause)

成果物: 暗号化アーカイブからのアセット読込、HTTP通信、動画再生が動作

フェーズ 8: 数学・物理・衝突判定（目安: 3-4 週）

ゴール: ゲームロジックに必要な計算基盤

- 数学ライブラリ (DirectXMath ラッパー)
  - Vector2, Vector3, Vector4
  - Matrix4x4
  - Quaternion
  - Color (RGBA, HSV 変換)
  - MathUtil (Lerp, SmoothStep, Remap, Clamp)
  - 乱数 (メルセンヌ・ツイスタ / PCG)
- 衝突判定
  - 2D: AABB, Circle, Polygon, Line
  - 3D: AABB, Sphere, Ray, Frustum, OBB
  - Sweep / Continuous Detection
- 空間分割
  - Quadtree (2D)
  - Octree (3D)
  - BVH
- 簡易 2D 物理
  - リジッドボディ
  - コリジョンレスポンス
  - トリガー判定
- Jolt Physics 統合 (3D、オプション)

成果物: DXLib の全数学・衝突判定関数 + 空間分割 + 簡易物理

フェーズ 9: DXLib 互換レイヤー + シェーダーホットリロード（目安: 3-4 週）

ゴール: 既存 DXLib コードの移植を容易にする

- DxLibCompat.h
  - DXLib の全パブリック関数に対応するラッパー
  - 型変換 (COLOR_U8 → GX::Color 等)
  - 定数マッピング (DX_BLENDMODE_ALPHA → GX::BlendMode::Alpha 等)
- 移行ガイドドキュメント
- シェーダーホットリロード
  - ファイル監視 → DXC 再コンパイル → PSO 再生成
  - エラー表示 (コンパイルエラーをオーバーレイ表示)
- シェーダーバリアント管理
  - #define による条件分岐
  - バリアントキャッシュ

成果物: `#include "DxLibCompat.h"` で既存コードが動作する目処が立つ

フェーズ 10: 最適化・品質・ドキュメント（目安: 4-6 週）

ゴール: プロダクション品質

パフォーマンス最適化

- メモリアロケータ (プール、スタック、リニア)
- リソースバリア最適化
- マルチスレッドコマンドリスト記録
- GPU タイムスタンプ プロファイラ
- 描画コールバッチング最適化
- テクスチャストリーミング

品質

- バリデーションレイヤー (D3D12 Debug Layer 統合)
- GPU ベース回帰テスト (スクリーンショット比較)
- ユニットテスト (Google Test)
- メモリリーク検出

ドキュメント

- API リファレンス (Doxygen)
- チュートリアル (Getting Started → 2Dゲーム → 3Dゲーム → GUI)
- サンプルプロジェクト群
  - 2D シューティング
  - 2D プラットフォーマー
  - 3D ウォークスルー
  - GUI メニューデモ
  - ポストエフェクトショーケース
- DXLib 移行ガイド

成果物: ドキュメント完備、サンプル動作、プロファイラ動作

---

5. 全体スケジュール概算

フェーズ	内容	目安期間	累計
Phase 0	D3D12 基盤 + Hello Triangle	4-6 週	~6 週
Phase 1	2D 描画エンジン	6-8 週	~14 週
Phase 2	テキスト・入力・サウンド	4-6 週	~20 週
Phase 3	3D 描画 (PBR)	8-12 週	~30 週
Phase 4	ポストエフェクト	6-8 週	~38 週
Phase 5	描画レイヤー	3-4 週	~42 週
Phase 6	XML-GUI	8-10 週	~50 週

Phase 7	ファイル・ネット・動画	4-6 週	~56 週
Phase 8	数学・物理	3-4 週	~60 週
Phase 9	互換レイヤー + ホットリロード	3-4 週	~64 週
Phase 10	最適化・品質・ドキュメント	4-6 週	~68 週

総見積: 約 60-70 週（1年〜1年3ヶ月）

※ 個人開発・フルタイム相当の場合。パートタイムなら 1.5〜2 倍。

---

6. 技術的リスクと対策

リスク	影響	対策
D3D12 の複雑さ (リソースバリア、ヒープ管理)	開発遅延、バグ多発	Phase 0 で徹底的に基盤を固める。D3D12MA (メモリアロケータ) 活用
シェーダーコンパイル時間	開発効率低下	DXC + キャッシュ + ホットリロードを早期実装
glTF/FBX のエッジケース	モデル読込バグ	cgltf (軽量) を基本にし、Assimp はフォールバック
GUI のレイアウト複雑化	実装量膨大	Flexbox のみ先行実装。Grid は後回し
パフォーマンスボトルネック	フレームレート低下	GPU プロファイラを Phase 0 から組み込み
スコープクリープ	完成しない	フェーズ毎に「最小限動くもの」を作る。機能追加は後から

---

7. 開発ルール

1. **フェーズ毎にビルド可能**: 各フェーズ完了時にコンパイル・動作するサンプルがある
2. **テスト駆動**: 数学・衝突判定は必ずユニットテスト
3. **ドキュメント先行**: 各クラスの public API は実装前にヘッダーで設計
4. **Git ブランチ戦略**: `main` (安定) / `develop` (統合) / `feature/*` (機能別)

5. **命名規則**:
    - 名前空間: `GX::` (Graphics eXtended)
    - クラス: PascalCase (`SpriteBatch`)
    - メソッド: PascalCase (`DrawSprite`)
    - メンバ変数: `m_camelCase`
    - 定数: `k_PascalCase`
6. **エラーハンドリング**: HRESULT チェック + 構造化ログ、致命的エラーはアサート

---

8. 次のアクション

Phase 0 から着手する。具体的な最初のステップ:

1. CMake プロジェクトを作成 (`GXLib` ライブラリ + `Sandbox` テストアプリ)
2. Win32 ウィンドウを表示
3. D3D12 デバイスを初期化し、画面をクリアカラーで塗りつぶす
4. 三角形を描画

> 準備ができたら Phase 0 の実装に入ろう。