目 次

序文 iv
はじめに vi
監修者まえがき ix

第1部　基盤 1

第1章　イントロダクション 3

1.1 典型的なゲーム開発チームの体制 5
1.1.1 エンジニア 5
1.1.2 アーティスト 6
1.1.3 ゲームデザイナ 7
1.1.4 プロデューサー 7
1.1.5 その他のスタッフ 8
1.1.6 パブリッシャーとスタジオ 8
1.2 ゲームとは何か？ 8
1.2.1 ソフトリアルタイムシミュレーションとしてのビデオゲーム 9
1.3 ゲームエンジンとは何か？ 11
1.4 ジャンルごとのエンジンの違い 13
1.4.1 一人称視点シューティングゲーム（FPS） 13
1.4.2 プラットフォーマーおよびその他の三人称視点シューティングゲーム 15
1.4.3 格闘ゲーム 17
1.4.4 レーシングゲーム 19
1.4.5 リアルタイムストラテジー（RTS） 20
1.4.6 多人数同時参加型オンラインゲーム（MMOG） 22
1.4.7 その他のジャンル 23
1.5 ゲームエンジンの調査 23
1.5.1 Quake系エンジン 23
1.5.2 Unreal系エンジン 24
1.5.3 ハーフライフSource Engine 25
1.5.4 マイクロソフトのXNAゲームスタジオ 25
1.5.5 その他の商用エンジン 26
1.5.6 内製エンジン 26
1.5.7 オープンソースのエンジン 26

1.6 ランタイムエンジンアーキテクチャ **27**
1.6.1 ターゲットハードウェア 27
1.6.2 デバイスドライバ 29
1.6.3 オペレーティングシステム 29
1.6.4 サードパーティSDKとミドルウェア 30
1.6.4.1 データ構造とアルゴリズム 30
1.6.4.2 グラフィックス 31
1.6.4.3 コリジョンと物理 31
1.6.4.4 キャラクターアニメーション 31
1.6.4.5 人工知能 32
1.6.4.6 生体力学的モデル 32
1.6.5 プラットフォーム独立層 33
1.6.6 コアシステム 33
1.6.7 リソースマネージャ 34
1.6.8 レンダリングエンジン 35
1.6.8.1 低レベルレンダラ 35
1.6.8.2 シーングラフ／カリング最適化 36
1.6.8.3 視覚効果 37
1.6.8.4 フロントエンド 38
1.6.9 プロファイリングおよびデバッグツール 39
1.6.10 コリジョンと物理 39
1.6.11 アニメーション 40
1.6.12 ヒューマンインターフェイスデバイス（HID） 41
1.6.13 オーディオ 42
1.6.14 オンラインマルチプレイヤー／ネットワーキング 43
1.6.15 ゲームプレイ基盤システム 44
1.6.15.1 ゲームワールドとオブジェクトモデル 44
1.6.15.2 イベントシステム 46
1.6.15.3 スクリプトシステム 46
1.6.15.4 人工知能の基盤 46
1.6.16 ゲーム固有のサブシステム 47
1.7 ツールとアセットパイプライン **48**
1.7.1 デジタルコンテンツ生成ツール 49
1.7.2 アセット調整パイプライン 49
1.7.3 3Dモデル／メッシュデータ 50
1.7.3.1 ブラシジオメトリ 50
1.7.3.2 3Dモデル（メッシュ） 51
1.7.4 スケルトンアニメーションデータ 51
1.7.5 オーディオデータ 52
1.7.6 パーティクルシステムデータ 52
1.7.7 ゲームワールドデータとワールドエディタ 53
1.7.8 ツールアーキテクチャへの取り組み 53

第2章　仕事用ツール　55

2.1 バージョン管理 56
2.1.1 なぜバージョン管理システムを使うのか？ 56
2.1.2 一般的なバージョン管理システム 57
2.1.3 SubversionとTortoiseSVNの概要 58
2.1.4 Googleでコードリポジトリをセットアップする 59
2.1.5 TortoiseSVNのインストール 60
2.1.6 ファイルバージョン、アップデートおよびコミット 61
2.1.7 マルチチェックアウト、ブランチおよびマージ 63
2.1.8 ファイルの削除 65
2.2 Microsoft Visual Studio 65
2.2.1 ソースファイル、ヘッダおよび翻訳単位 66
2.2.2 ライブラリ、実行ファイルおよびダイナミックリンクライブラリ 66
2.2.3 プロジェクトとソリューション 67
2.2.4 ビルド構成 68
2.2.4.1 一般的なビルドオプション 69
2.2.4.2 一般的なビルド構成 71
2.2.4.3 プロジェクト構成のチュートリアル 73
2.2.4.4 新しい.vcprojファイルの生成 76
2.2.5 コードのデバッグ 77
2.2.5.1 スタートアッププロジェクト 78
2.2.5.2 ブレークポイント 78
2.2.5.3 コードをステップ実行する 78
2.2.5.4 コールスタック 79
2.2.5.5 ウォッチウィンドウ 79
2.2.5.6 データブレークポイント 81
2.2.5.7 条件付きブレークポイント 82
2.2.5.8 最適化したビルドのデバッグ 83
2.3 プロファイリングツール 84
2.3.1 プロファイラのリスト 86
2.4 メモリリークと破壊の検出 86
2.5 その他のツール 87

第3章　ゲームのためのソフトウェアエンジニアリングの基本　89

3.1 C++の復習とベストプラクティス 90
3.1.1 オブジェクト指向プログラミングの簡単な復習 90
3.1.1.1 クラスとオブジェクト 90
3.1.1.2 カプセル化 90
3.1.1.3 継承 90
3.1.1.4 ポリモーフィズム 92
3.1.1.5 コンポジションと集約 94
3.1.1.6 デザインパターン 95
3.1.2 コーディング標準：なぜ、どのくらい？ 95

3.2 C/C++のデータとコードとメモリ **97**
3.2.1 数値表現 97
3.2.1.1 基数 97
3.2.1.2 符号付き整数と符号なし整数 97
3.2.1.3 固定小数点数 98
3.2.1.4 浮動小数点数 99
3.2.1.5 アトミックなデータタイプ 101
3.2.1.6 マルチバイト値とバイトオーダー 104
3.2.2 宣言、定義およびリンケージ 107
3.2.2.1 翻訳単位について、もう一度 107
3.2.2.2 宣言 vs 定義 109
3.2.2.3 リンケージ 111
3.2.3 C/C++のメモリ配置 113
3.2.3.1 実行可能イメージ 113
3.2.3.2 プログラムスタック 115
3.2.3.3 動的割り当てのヒープ 117
3.2.4 メンバ変数 118
3.2.4.1 クラスの静的メンバ 119
3.2.5 メモリ内のオブジェクト配置 120
3.2.5.1 アラインメントとパッキング 121
3.2.5.2 C++クラスのメモリ配置 124
3.3 エラーの捕捉と処理 **127**
3.3.1 エラーの種類 127
3.3.2 エラーの処理 128
3.3.2.1 プレイヤーエラーの処理 128
3.3.2.2 開発者エラーの処理 128
3.3.2.3 プログラマエラーの処理 129
3.3.3 エラー検出と処理の実装 130
3.3.3.1 エラーのリターンコード 130
3.3.3.2 例外 130
3.3.3.3 アサート 131

第4章 ゲームのための3D計算 135

4.1 2Dで3Dの問題を解決する **136**
4.2 ポイントとベクトル **136**
4.2.1 ポイントとデカルト座標 137
4.2.2 左手座標系 vs 右手座標系 138
4.2.3 ベクトル 139
4.2.3.1 デカルト基底ベクトル 139
4.2.4 ベクトル操作 140
4.2.4.1 スカラーによる乗算 140
4.2.4.2 加算と減算 141
4.2.4.3 大きさ 141
4.2.4.4 ベクトル操作の実例 142

4.2.4.5 正規化と単位ベクトル 143
4.2.4.6 法線ベクトル 143
4.2.4.7 ドット積と射影 144
4.2.4.8 クロス積 146
4.2.5 ポイントとベクトルの線形補間 149
4.3 行列 150
4.3.1 行列の乗算 150
4.3.2 行列としての座標とベクトルの表現 151
4.3.3 単位行列 152
4.3.4 逆行列 152
4.3.5 転置行列 153
4.3.6 同次座標 153
4.3.6.1 方向ベクトルの変換 154
4.3.7 アトミック変換行列 155
4.3.7.1 平行移動 156
4.3.7.2 回転 156
4.3.7.3 スケーリング 157
4.3.8 4×3の行列 158
4.3.9 座標空間 158
4.3.9.1 モデル空間 159
4.3.9.2 ワールド空間 160
4.3.9.3 ビュー空間 161
4.3.10 基底の変更 162
4.3.10.1 座標空間の階層構造 162
4.3.10.2 基底変換行列の作成 162
4.3.10.3 行列からの単位基底ベクトルの抽出 164
4.3.10.4 座標系の変換 vs ベクトル 164
4.3.11 法線ベクトルの変換 165
4.3.12 メモリへの行列の格納 166
4.4 クォータニオン 168
4.4.1 3D回転としての単位クォータニオン 169
4.4.2 クォータニオン演算 169
4.4.2.1 クォータニオンの乗算 169
4.4.2.2 共役と逆数 170
4.4.3 クォータニオンでベクトルを回転する 171
4.4.3.1 クォータニオンの連結 172
4.4.4 クォータニオンと行列の同一性 172
4.4.5 回転の線形補間 174
4.4.5.1 球面線形補間 174
4.4.5.2 SLERPすべきか、せざるべきか（それが問題だ） 176
4.5 回転表現の比較 176
4.5.1 オイラー角 176
4.5.2 3×3行列 177
4.5.3 軸＋角度 177

4.5.4 クォータニオン 178
4.5.5 SQT変換 178
4.5.6 デュアルクォータニオン 179
4.5.7 回転と自由度 179
4.6 その他の便利な数学オブジェクト 180
4.6.1 直線、レイ、線分 180
4.6.2 球 181
4.6.3 平面 182
4.6.4 軸平行バウンディングボックス（AABB） 183
4.6.5 有向バウンディングボックス（OBB） 184
4.6.6 錐台 184
4.6.7 凸多面体 185
4.7 ハードウェアアクセラレートされたSIMD計算 185
4.7.1 SSEレジスタ 186
4.7.2 __m128データ型 187
4.7.2.1 __m128変数のアラインメント 187
4.7.3 SSE組み込み命令を使ったコーディング 187
4.7.4 SSEを使ったベクトルと行列の乗算 189
4.8 乱数生成 192
4.8.1 線形合同法を用いた生成器 193
4.8.2 メルセンヌ・ツイスタ 193
4.8.3 Mother-of-AllとXorshift 194

第2部 低レベルエンジンシステム 195

第5章 エンジンサポートシステム 197

5.1 サブシステムのスタートアップとシャットダウン 198
5.1.1 C++における静的初期化の順序（またはその欠如） 198
5.1.1.1 オンデマンドのコンストラクタ呼び出し 199
5.1.2 うまくいく単純な方法 201
5.1.3 エンジンの実例 203
5.1.3.1 OGRE 3D 203
5.1.3.2 『アンチャーテッド エルドラドの秘宝』（ノーティドッグ社） 205
5.2 メモリ管理 206
5.2.1 動的メモリ割り当ての最適化 207
5.2.1.1 スタックベースのアロケータ 208
5.2.1.2 プールアロケータ 210
5.2.1.3 アラインメントされた割り当て 211
5.2.1.4 単一フレームと二重バッファ型メモリアロケータ 214
5.2.2 メモリ断片化 216
5.2.2.1 スタックおよびプールアロケータを使った断片化回避 217
5.2.2.2 デフラグと再配置 218

5.2.3 キャッシュの一貫性 220
5.2.3.1 L1およびL2キャッシュ 221
5.2.3.2 命令キャッシュとデータキャッシュ 222
5.2.3.3 キャッシュミスの回避 222
5.3 コンテナ 224
5.3.1 コンテナ処理 225
5.3.2 イテレータ 226
5.3.2.1 プレインクリメント vs ポストインクリメント 227
5.3.3 アルゴリズムの複雑さ 227
5.3.4 独自仕様のコンテナクラスを作成する 229
5.3.4.1 作るべきか、使うべきか、それが問題だ 229
5.3.4.2 動的配列およびチャンクでの割り当て 233
5.3.4.3 連結リスト 233
5.3.4.4 ディクショナリとハッシュテーブル 239
5.4 文字列 242
5.4.1 文字列の問題 242
5.4.2 文字列クラス 243
5.4.3 一意の識別子 244
5.4.3.1 ハッシュ化された文字列ID 245
5.4.3.2 実装アイデア 245
5.4.4 ローカリゼーション 247
5.4.4.1 Unicode 248
5.4.4.2 その他のローカリゼーションの問題 250
5.5 エンジンのコンフィグレーション 252
5.5.1 オプションのロードと保存 252
5.5.2 ユーザーごとのオプション 253
5.5.3 実際のエンジンにおけるコンフィグレーション管理 254
5.5.3.1 例：QuakeのCVAR 254
5.5.3.2 例：OGRE 3D 255
5.5.3.3 例：『アンチャーテッド エルドラドの秘宝』 255

第6章 リソースとファイルシステム 259

6.1 ファイルシステム 261
6.1.1 ファイル名とパス 261
6.1.1.1 オペレーティングシステムの違い 261
6.1.1.2 絶対パスと相対パス 263
6.1.1.3 検索パス 264
6.1.1.4 パスのAPI 264
6.1.2 基本のファイル入出力 265
6.1.2.1 ラップすべきか、せざるべきか 266
6.1.2.2 同期ファイル入出力 266

6.1.3 非同期ファイル入出力 267
6.1.3.1 優先順位 270
6.1.3.2 非同期ファイルの入出力の動作 270
6.2 リソースマネージャ 271
6.2.1 オフラインリソースマネージメントとツールチェーン 271
6.2.1.1 アセットのリビジョン管理 271
6.2.1.2 リソースデータベース 273
6.2.1.3 成功したリソースデータベース設計 274
6.2.1.4 アセット調整パイプライン 279
6.2.2 ランタイムリソースマネージメント 281
6.2.2.1 ランタイムリソースマネージャの責任 281
6.2.2.2 リソースファイルとディレクトリ編成 282
6.2.2.3 リソースファイルフォーマット 284
6.2.2.4 リソースGUID 285
6.2.2.5 リソースレジストリ 285
6.2.2.6 リソースの寿命 286
6.2.2.7 リソースのメモリ管理 288
6.2.2.8 複合リソースと参照整合性 293
6.2.2.9 リソース間の相互参照を処理する 294
6.2.2.10 ロード後の初期化 298

第7章 ゲームループとリアルタイムシミュレーション 301

7.1 レンダリングループ 302
7.2 ゲームループ 303
7.2.1 簡単な例：ポン 303
7.3 ゲームループのアーキテクチャスタイル 305
7.3.1 Windowsメッセージポンプ 305
7.3.2 コールバック駆動フレームワーク 306
7.3.3 イベントを使った更新 308
7.4 抽象的な時間軸 308
7.4.1 リアルタイム 308
7.4.2 ゲーム時間 309
7.4.3 ローカル時間軸とグローバル時間軸 309
7.5 時間の計測と処理 311
7.5.1 フレームレートと時間差分 311
7.5.2 フレームレートから速度へ 311
7.5.2.1 昔のCPU依存のゲーム 312
7.5.2.2 経過時間に基づいた更新 312
7.5.2.3 移動平均を使う 313
7.5.2.4 固定フレームレート 313
7.5.2.5 垂直帰線消去期間 314
7.5.3 高分解能タイマーでリアルタイムを計測する 315
7.5.3.1 高分解能クロックのドリフト 316

7.5.4 時間単位とクロック変数 316
7.5.4.1 64ビット整数クロック 316
7.5.4.2 32ビット整数クロック 317
7.5.4.3 32ビット浮動小数点数クロック 317
7.5.4.4 浮動小数点数クロックの限界 318
7.5.4.5 その他の時間単位 318
7.5.5 ブレークポイントの処理 319
7.5.6 シンプルなクロッククラス 320
7.6 マルチプロセッサのゲームループ 323
7.6.1 マルチプロセッサゲーム機のアーキテクチャ 324
7.6.1.1 Xbox 360 324
7.6.1.2 PlayStation 3 324
7.6.2 SIMD 325
7.6.3 フォーク - ジョイン 326
7.6.4 サブシステムごとに1つのスレッド 327
7.6.5 ジョブ 328
7.6.6 非同期プログラム設計 330
7.7 ネットワークマルチプレイヤーのゲームループ 332
7.7.1 クライアントサーバ 332
7.7.2 ピアツーピア 334
7.7.3 ケーススタディ:『Quake II』 334

第8章 ヒューマンインターフェイスデバイス(HID) 337

8.1 ヒューマンインターフェイスデバイスの種類 338
8.2 HIDとのインターフェイス 340
8.2.1 ポーリング 340
8.2.2 割り込み 340
8.2.3 ワイヤレスデバイス 341
8.3 入力の種類 341
8.3.1 デジタルボタン 341
8.3.2 アナログ軸とボタン 343
8.3.3 相対軸 344
8.3.4 加速度計 345
8.3.5 WiiリモコンまたはSIXAXISを使った3次元定位 345
8.3.6 カメラ 346
8.4 出力の種類 347
8.4.1 振動 347
8.4.2 力覚フィードバック 348
8.4.3 オーディオ 348
8.4.4 その他の入出力 348

8.5 ゲームエンジンのHIDシステム 349
8.5.1 一般的な要件 349
8.5.2 デッドゾーン 349
8.5.3 アナログ信号のフィルタリング 350
8.5.4 入力イベントの検出 352
8.5.4.1 ボタンアップとボタンダウン 352
8.5.4.2 ボタン同時押し判定 353
8.5.4.3 シーケンスとジェスチャーの検出 354
8.5.5 マルチプレイヤーのための複数HIDの管理 359
8.5.6 クロスプラットフォームのHIDシステム 360
8.5.7 入力の再マッピング 362
8.5.8 状況に応じたコントロール 363
8.5.9 入力を無効にする 364
8.6 ヒューマンインターフェイスデバイスの実際 364

第9章 デバッグおよび開発のツール 365

9.1 ロギングとトレース 366
9.1.1 OutputDebugString()を使ったフォーマット付き出力 367
9.1.2 冗長性 368
9.1.3 チャンネル 369
9.1.4 出力をファイルにミラーする 369
9.1.5 クラッシュレポート 370
9.2 デバッグ描画機能 371
9.2.1 デバッグ描画API 374
9.3 ゲーム内メニュー 378
9.4 ゲーム内コンソール 381
9.5 カメラのデバッグとゲームの一時停止 382
9.6 チート 382
9.7 スクリーンショットと動画キャプチャー 383
9.8 ゲーム内プロファイリング 384
9.8.1 階層プロファイリング 386
9.8.1.1 実行時間を階層的に計測する 387
9.8.2 Excelにエクスポートする 391
9.9 ゲーム内メモリ統計とリーク検出 391

第3部　グラフィックスとモーション　395

第10章　レンダリングエンジン　397

10.1 深度バッファを利用した三角形のラスタライズの原理 **398**
10.1.1 シーンの記述 400
10.1.1.1 ハイエンドのレンダリングパッケージで使われる描写 400
10.1.1.2 三角形メッシュ 401
10.1.1.3 三角形メッシュを構築する 403
10.1.1.4 モデル空間 407
10.1.1.5 ワールド空間とメッシュのインスタンス化 408
10.1.2 表面の視覚特性の記述 409
10.1.2.1 光と色の入門 409
10.1.2.2 頂点属性 411
10.1.2.3 頂点フォーマット 412
10.1.2.4 属性の補間 414
10.1.2.5 テクスチャ 415
10.1.2.6 マテリアル 421
10.1.3 ライティングの基礎 422
10.1.3.1 ローカルおよびグローバルイルミネーションモデル 424
10.1.3.2 フォンライティングモデル 425
10.1.3.3 光源のモデル化 428
10.1.4 仮想カメラ 431
10.1.4.1 ビュー空間 431
10.1.4.2 投影 433
10.1.4.3 ビューボリュームと視錐台 433
10.1.4.4 投影法と同次クリップ空間 434
10.1.4.5 スクリーン空間とアスペクト比 438
10.1.4.6 フレームバッファ 439
10.1.4.7 三角形のラスタライズとフラグメント 439
10.1.4.8 オクルージョンと深度バッファ 441

10.2 レンダリングパイプライン **443**
10.2.1 レンダリングパイプラインの概要 444
10.2.1.1 レンダリングパイプラインのデータ変換方法 445
10.2.1.2 パイプラインの実装 445
10.2.2 ツールステージ 446
10.2.3 アセット調整ステージ 448
10.2.4 GPUの歴史の概要 448
10.2.5 GPUパイプライン 450
10.2.5.1 頂点シェーダ 450
10.2.5.2 ジオメトリシェーダ 450
10.2.5.3 ストリーム出力 451
10.2.5.4 クリッピング 451
10.2.5.5 スクリーンマッピング 451
10.2.5.6 三角形のセットアップ 451

10.2.5.7 三角形トラバーサル 452
10.2.5.8 早期zテスト 452
10.2.5.9 ピクセルシェーダ 452
10.2.5.10 マージステージ／ラスターオペレーションステージ 452
10.2.6 プログラマブルシェーダ 454
10.2.6.1 メモリアクセス 454
10.2.6.2 高レベルシェーダ言語の文法入門 456
10.2.6.3 エフェクトファイル 458
10.2.6.4 参考文献 459
10.2.7 アプリケーションステージ 459
10.2.7.1 可視化の決定 459
10.2.7.2 プリミティブの送出 462
10.2.7.3 ジオメトリソーティング 464
10.2.7.4 シーングラフ 465
10.2.7.5 シーングラフを選択する 468
10.3 高度なライティングとグローバルイルミネーション 469
10.3.1 イメージベースドライティング 469
10.3.1.1 法線マッピング 469
10.3.1.2 ハイトマップ：パララックスマッピングとレリーフマッピング 470
10.3.1.3 スペキュラーマップ／グロスマップ 471
10.3.1.4 環境マッピング 471
10.3.1.5 3次元テクスチャ 472
10.3.2 高ダイナミックレンジライティング 472
10.3.3 グローバルイルミネーション 473
10.3.3.1 影のレンダリング 473
10.3.3.2 アンビエントオクルージョン 476
10.3.3.3 映り込み 476
10.3.3.4 コースティクス 477
10.3.3.5 表面下散乱 478
10.3.3.6 事前計算済み放射輝度伝播（PRT） 478
10.3.4 ディファードレンダリング 479
10.4 視覚効果とオーバーレイ 481
10.4.1 パーティクル効果 481
10.4.2 デカール 482
10.4.3 環境効果 483
10.4.3.1 空 483
10.4.3.2 地形 484
10.4.3.3 水 485
10.4.4 オーバーレイ 486
10.4.4.1 正規化スクリーン座標 486
10.4.4.2 相対スクリーン座標 487
10.4.4.3 テキストとフォント 487
10.4.5 ガンマ補正 487
10.4.6 フルスクリーンのポストエフェクト 489
10.5 参考文献 490

第11章　アニメーションシステム　491

11.1 キャラクターアニメーションの種類 492
- 11.1.1 セルアニメーション 492
- 11.1.2 階層型剛体アニメーション 493
- 11.1.3 頂点単位のアニメーションとモーフターゲット 494
- 11.1.4 スキンアニメーション 495
- 11.1.5 データ圧縮手法としてのアニメーション 496

11.2 スケルトン 497
- 11.2.1 スケルトンの階層 498
- 11.2.2 スケルトンのメモリ上での表現 499

11.3 ポーズ 500
- 11.3.1 バインドポーズ 500
- 11.3.2 ローカルポーズ 500
  - 11.3.2.1 関節のスケーリング 502
  - 11.3.2.2 関節のポーズのメモリ上の表現 502
  - 11.3.2.3 基底の変換としての関節のポーズ 503
- 11.3.3 グローバルポーズ 504
  - 11.3.3.1 グローバルポーズのメモリ上の表現 504

11.4 クリップ 505
- 11.4.1 ローカル時間軸 506
  - 11.4.1.1 ポーズの補間と連続時間 506
  - 11.4.1.2 時間単位 507
  - 11.4.1.3 フレーム vs サンプル 508
  - 11.4.1.4 フレーム、サンプル、ループクリップ 508
  - 11.4.1.5 正規化時間（位相） 509
- 11.4.2 グローバルの時間軸 510
- 11.4.3 ローカルクロックとグローバルクロックの比較 512
  - 11.4.3.1 ローカルクロックでアニメーションを同期する 513
  - 11.4.3.2 アニメーションとグローバルクロックを同期する 514
- 11.4.4 単純なアニメーションのデータフォーマット 515
  - 11.4.4.1 アニメーションのリターゲティング 516
- 11.4.5 連続チャンネル関数 517
- 11.4.6 メタチャンネル 517

11.5 スキニングと行列パレットの生成 519
- 11.5.1 頂点単位のスキニングの情報 519
- 11.5.2 スキニングの数学 520
  - 11.5.2.1 単純な例：関節が1つのスケルトン 520
  - 11.5.2.2 関節が複数あるスケルトンへの拡張 522
  - 11.5.2.3 モデル - ワールド変換の導入 523
  - 11.5.2.4 1つの頂点を複数の関節にスキニングする 523

11.6 アニメーションブレンディング 524
- 11.6.1 LERPブレンディング 524

11.6.2 LERPブレンディングの応用 526
11.6.2.1 時間的な補間 526
11.6.2.2 動きの連続性：クロスフェード 526
11.6.2.3 方向を持った移動 529
11.6.3 複雑なLERPブレンディング 531
11.6.3.1 一般化1次元LERPブレンディング 531
11.6.3.2 単純な2次元LERPブレンディング 532
11.6.3.3 三角形2次元LERPブレンディング 533
11.6.3.4 一般化2次元LERPブレンディング 535
11.6.4 部分的なスケルトンのブレンディング 536
11.6.5 加算ブレンディング 537
11.6.5.1 数学的な定式化 538
11.6.5.2 加算ブレンディング vs 部分的なブレンディング 539
11.6.5.3 加算ブレンディングの限界 539
11.6.6 加算ブレンディングの応用 540
11.6.6.1 姿勢のバリエーション 540
11.6.6.2 運動のノイズ 541
11.6.6.3 エイミングと注視 541
11.6.6.4 時間軸のオーバーロード 542
11.7 ポスト処理 543
11.7.1 プロシージャルアニメーション 543
11.7.2 逆運動学 544
11.7.3 ラグドール 545
11.8 圧縮のテクニック 546
11.8.1 チャンネルの省略 546
11.8.2 量子化 547
11.8.3 サンプリング周波数とキーフレームの省略 550
11.8.4 曲線に基づく圧縮 551
11.8.5 選択的ロードとストリーミング 551
11.9 アニメーションシステムのアーキテクチャ 552
11.10 アニメーションパイプライン 553
11.10.1 データ構造 555
11.10.1.1 共有リソースデータ 555
11.10.1.2 インスタンス単位のデータ 556
11.10.2 均一な重み付き平均ブレンディングの表現 557
11.10.2.1 例：OGRE 3D 558
11.10.2.2 例：Granny 560
11.10.3 ブレンドツリー 560
11.10.3.1 2入力のLERPブレンディング 561
11.10.3.2 一般化された1次元LERPブレンディング 561
11.10.3.3 単純な2次元LERPブレンディング 562
11.10.3.4 三角形LERPブレンディング 562
11.10.3.5 一般化された三角形LERPブレンディング 562
11.10.3.6 加算ブレンディング 563

11.10.4 クロスフェードのアーキテクチャ 564
11.10.4.1 均一な重み付き平均を使ったクロスフェード 564
11.10.4.2 式木を使ったクロスフェード 566
11.10.5 アニメーションパイプラインの最適化 566
11.10.5.1 PlayStation 2での最適化 567
11.10.5.2 PlayStation 3での最適化 567
11.10.5.3 XboxとXbox 360での最適化 568
11.11 アクションステートマシン 569
11.11.1 アニメーションのステート（状態） 569
11.11.1.1 ステートとブレンドツリーの仕様 570
11.11.1.2 カスタムのブレンドツリーノードの種類 570
11.11.1.3 例：ノーティドッグ社の『アンチャーテッド』エンジン 571
11.11.1.4 例：Unreal Engine 3 574
11.11.2 遷移 576
11.11.2.1 遷移の種類 576
11.11.2.2 遷移のパラメータ 577
11.11.2.3 遷移行列 577
11.11.2.4 遷移行列の実装 577
11.11.3 状態のレイヤー 580
11.11.4 制御パラメータ 582
11.11.5 制約（コンストレイン） 583
11.11.5.1 アタッチメント 583
11.11.5.2 オブジェクト間の登録 584
11.11.5.3 把持とハンドIK 587
11.11.5.4 運動抽出とフットIK 588
11.11.5.5 その他の制約 591
11.12 アニメーションコントローラ 592

第12章 コリジョンと剛体力学 593

12.1 あなたのゲームに物理は必要ですか？ 595
12.1.1 物理計算システムでできること 595
12.1.2 物理は楽しい？ 596
12.1.2.1 シミュレータ 596
12.1.2.2 物理パズルゲーム 596
12.1.2.3 サンドボックスゲーム 596
12.1.2.4 ゴールを目指すストーリー主導のゲーム 597
12.1.3 ゲームにおける物理の影響 597
12.1.3.1 ゲームデザインへの影響 597
12.1.3.2 開発への影響 598
12.1.3.3 アートへの影響 599
12.1.3.4 その他の影響 599
12.2 コリジョン／物理のミドルウェア 600
12.2.1 I-Collide、SWIFT、V-Collide、RAPID 600

12.2.2 ODE 600
12.2.3 Bullet 601
12.2.4 TrueAxis 601
12.2.5 PhysX 601
12.2.6 Havok 601
12.2.7 Physics Abstraction Layer (PAL) 602
12.2.8 Digital Molecular Matter (DMM) 602
12.3 コリジョン検出システム 602
12.3.1 衝突可能物 603
12.3.2 コリジョン／物理ワールド 604
13.3.2.1 物理ワールド 605
12.3.3 シェイプの概念 605
12.3.3.1 交差 606
12.3.3.2 接触 606
12.3.3.3 凸形状 606
12.3.4 コリジョンプリミティブ 607
12.3.4.1 球 607
12.3.4.2 カプセル 607
12.3.4.3 軸平行バウンディングボックス 608
12.3.4.4 有向バウンディングボックス 608
12.3.4.5 離散方向ポリトープ 609
12.3.4.6 任意の凸多面体 609
12.3.4.7 ポリゴンスープ 610
12.3.4.8 複合シェイプ 611
12.3.5 コリジョン検出と解析幾何学 612
12.3.5.1 点 vs 球 612
12.3.5.2 球 vs 球 613
12.3.5.3 分離軸定理 613
12.3.5.4 AABB vs AABB 615
12.3.5.5 凸形状のコリジョン検出 : GJKアルゴリズム 616
12.3.5.6 その他の形状の組み合わせ 618
12.3.5.7 動いている物体間のコリジョン検出 619
12.3.6 パフォーマンス最適化 621
12.3.6.1 時間的コヒーレンス 622
12.3.6.2 空間分割 622
12.3.6.3 ブロードフェーズ、ミッドフェーズ、ナローフェーズ 622
12.3.7 コリジョンのクエリー 623
12.3.7.1 レイキャスト 624
12.3.7.2 シェイプキャスト 625
12.3.7.3 ファントム 627
12.3.7.4 他のタイプのクエリー 628
12.3.8 コリジョンのフィルタリング 628
12.3.8.1 コリジョンのマスキングとレイヤー 628
12.3.8.2 コリジョンのコールバック 629
12.3.8.3 ゲームに固有のコリジョンのマテリアル 629

12.4 剛体力学 **630**
12.4.1 原理 631
12.4.1.1 単位 632
12.4.1.2 並進運動と回転運動の力学の可分性 632
12.4.1.3 重心 632
12.4.2 並進運動の力学 633
12.4.2.1 並進速度と加速度 633
12.4.2.2 力と運動量 634
12.4.3 運動方程式を解く 635
12.4.3.1 関数としての力 635
12.4.3.2 常微分方程式 636
12.4.3.3 解析的解法 636
12.4.4 数値積分 637
12.4.4.1 陽的オイラー法 637
12.4.4.2 数値解析手法の特性 638
12.4.4.3 陽的オイラー法の代替手法 639
12.4.4.4 ベルレ積分 640
12.4.4.5 速度ベルレ法 640
12.4.5 2次元の回転運動の力学 641
12.4.5.1 向きと角速度 641
12.4.5.2 角速度と角加速度 641
12.4.5.3 慣性モーメント 642
12.4.5.4 トルク 642
12.4.5.5 2次元の角運動方程式を解く 643
12.4.6 3次元の回転運動の力学 644
12.4.6.1 慣性テンソル 644
12.4.6.2 3次元での向き 645
12.4.6.3 3次元の角速度と角運動量 645
12.4.6.4 3次元のトルク 647
12.4.6.5 3次元の角運動方程式を解く 647
12.4.7 衝突応答 649
12.4.7.1 エネルギー 649
12.4.7.2 衝撃の衝突応答 649
12.4.7.3 ペナルティー力 652
12.4.7.4 コンストレインを使ってコリジョンを解く 653
12.4.7.5 摩擦 653
12.4.7.6 接合 654
12.4.7.7 停止、アイランド、スリープ 655
12.4.8 コンストレイン 656
12.4.8.1 点と点のコンストレイン 657
12.4.8.2 硬いばねのコンストレイン 657
12.4.8.3 ヒンジコンストレイン 658
12.4.8.4 スライダコンストレイン 658
12.4.8.5 その他によく使われるコンストレインの種類 659
12.4.8.6 コンストレインチェイン 659

12.4.8.7 ラグドール 659
12.4.8.8 パワーコンストレイン 660
12.4.9 剛体の動きを制御する 661
12.4.9.1 重力 661
12.4.9.2 力を作用させる 662
12.4.9.3 トルクを作用させる 662
12.4.9.4 衝撃を作用させる 662
12.4.10 コリジョン／物理のステップ 662
12.4.10.1 コンストレインソルバー 663
12.4.10.2 エンジン間の差異 664
12.5 物理エンジンをゲームに組み込む 665
12.5.1 ゲームのオブジェクトと剛体の関連付け 665
12.5.1.1 物理駆動の剛体 667
12.5.1.2 ゲーム駆動の剛体 667
12.5.1.3 固定体 669
12.5.1.4 Havokのモーションタイプ 669
12.5.2 シミュレーションの更新 669
12.5.2.1 コリジョンのクエリーのタイミング 671
12.5.2.2 シングルスレッドによる更新 672
12.5.2.3 マルチスレッドによる更新 673
12.5.3 ゲームにおけるコリジョン検出と物理システムの利用例 675
12.5.3.1 単純な剛体のゲームオブジェクト 675
12.5.3.2 弾丸の軌跡 675
12.5.3.3 手榴弾 676
12.5.3.4 爆発 677
12.5.3.5 壊れるオブジェクト 677
12.5.3.6 キャラクターの制御機構 678
12.5.3.7 カメラのコリジョン 680
12.5.3.8 ラグドールの組み込み 681
12.6 将来に目を向ける：高度な物理的要素 683

第4部 ゲームプレイ 685

第13章 ゲームプレイシステムの概要 687

13.1 ゲームワールドの構造 689
13.1.1 ワールド要素 689
13.1.1.1 静的なジオメトリ 691
13.1.2 ワールドチャンク 691
13.1.3 高レベルのゲームフロー 693
13.2 動的要素の実装：ゲームオブジェクト 694
13.2.1 ゲームオブジェクトモデル 695
13.2.2 ツールでの設計とランタイムの設計 696

13.3 データ駆動型ゲームエンジン **697**
13.4 ゲームワールドエディタ **698**
13.4.1 ゲームワールドエディタの典型的な機能 700
13.4.1.1 ワールドのチャンクの生成と管理 700
13.4.1.2 ゲームワールドの視覚化 701
13.4.1.3 ナビゲーション 701
13.4.1.4 選択 701
13.4.1.5 レイヤー 702
13.4.1.6 プロパティグリッド 702
13.4.1.7 オブジェクトの配置と整列のサポート 704
13.4.1.8 特別なオブジェクトのタイプ 704
13.4.1.9 ワールドチャンクの保存とロード 705
13.4.1.10 迅速な反復開発 706
13.4.2 統合されたアセット管理ツール 706
13.4.2.1 データ処理のコスト 708

第 14 章　ランタイムのゲームプレイ基本システム　709

14.1 ゲームプレイ基本システムの構成要素 **710**
14.2 ランタイムオブジェクトモデルのアーキテクチャ **713**
14.2.1 オブジェクト主体のアーキテクチャ 714
14.2.1.1 Cによる単純なオブジェクトベースのモデル：『Hydro Thunder』 714
14.2.1.2 モノリシックなクラスの階層 716
14.2.1.3 深く広い階層の問題 718
14.2.1.4 コンポジションを使って階層を単純化する 721
14.2.1.5 一般的なコンポーネント 725
14.2.1.6 純粋なコンポーネントモデル 726
14.2.2 プロパティ主体のアーキテクチャ 728
14.2.2.1 プロパティクラスを使って振る舞いを実装する 729
14.2.2.2 スクリプトを使って振る舞いを実装する 729
14.2.2.3 プロパティ vs コンポーネント 730
14.2.2.4 プロパティ主体の設計の長所と短所 730
14.2.2.5 参考文献 731
14.3 ワールドチャンクデータのフォーマット **732**
14.3.1 バイナリのオブジェクトイメージ 733
14.3.2 シリアライズされたゲームオブジェクトの記述 733
14.3.3 スポーナーとタイプスキーマ 735
14.3.3.1 オブジェクトタイプスキーマ 736
14.3.3.2 属性のデフォルト値 738
14.3.3.3 スポーナーとタイプスキーマの利点 738
14.4 ゲームワールドのロードとストリーミング **739**
14.4.1 単純なレベルのロード 740
14.2.2 シームレスなロードを目指して：エアロック 741

14.4.3 ゲームワールドのストリーミング 742
14.4.3.1 ロードすべきリソースを決定する 743
14.4.4 オブジェクト発生のためのメモリ管理 743
14.4.4.1 オブジェクト発生のためのオフラインのメモリ割り当て 744
14.4.4.2 オブジェクト発生のための動的なメモリ管理 745
14.4.5 保存されたゲーム 746
14.4.5.1 チェックポイント 747
14.4.5.2 どこでもセーブ 747
14.5 オブジェクト参照とワールドクエリー 748
14.5.1 ポインタ 748
14.5.2 スマートポインタ 750
14.5.3 ハンドル 751
14.5.4 ゲームオブジェクトのクエリー 754
14.6 ゲームオブジェクトのリアルタイムアップデート 755
14.6.1 単純な（使い物にならない）アプローチ 757
14.6.1.1 アクティブなゲームオブジェクトの集合を保持する 757
14.6.1.2 Update()関数の役割 758
14.6.2 パフォーマンスの制限とバッチ処理によるアップデート 759
14.6.3 オブジェクトとサブシステムの相互依存性 762
14.6.3.1 段階的なアップデート 762
14.6.3.2 バケツ方式アップデート 764
14.6.3.3 オブジェクトの状態の矛盾と1フレームの遅延 766
14.6.3.4 オブジェクトの状態のキャッシュ 768
14.6.3.5 タイムスタンプ 769
14.6.4 並列化の設計 769
14.6.4.1 ゲームオブジェクトモデル自体を並列化する 769
14.6.4.2 並行エンジンサブシステムと接続する 770
14.7 イベントとメッセージ通信 771
14.7.1 静的型付け関数バインディングの問題 772
14.7.2 イベントをオブジェクトにカプセル化する 773
14.7.3 イベントのタイプ 774
14.7.4 イベントの引数 775
14.7.4.1 キーと値をペアとしたイベントの引数 777
14.7.5 イベントハンドラ 777
14.7.6 イベントの引数のアンパック 778
14.7.7 チェイン・オブ・レスポンシビリティ 779
14.7.8 イベントに関心を登録する 781
14.7.9 キューを使うべきか、使わざるべきか 781
14.7.9.1 イベントキューの利点 782
14.7.9.2 イベントキューの問題点 783
14.7.10 即時にイベントを送信することの問題点 786
14.7.11 データ駆動型のイベント／メッセージ通信システム 787
14.7.11.1 データ経路通信システム 789
14.7.11.2 GUIベースのプログラミングの長所と短所 791

14.8 スクリプトによる記述 **791**
14.8.1 ランタイム vs データ定義 791
14.8.2 プログラミング言語の特徴 792
14.8.2.1 ゲームスクリプト言語の一般的な特徴 793
14.8.3 一般的なゲームスクリプト言語 794
14.8.3.1 QuakeC 794
14.8.3.2 UnrealScript 795
14.8.3.3 Lua 796
14.8.3.4 Python 797
14.8.3.5 Pawn／Small／Small-C 799
14.8.4 スクリプト記述のためのアーキテクチャ 799
14.8.5 ランタイムのゲームスクリプト言語の特徴 801
14.8.5.1 ネイティブなプログラミング言語とのインターフェイス 801
14.8.5.2 ゲームオブジェクトハンドル 806
14.8.5.3 イベントの受け取りと処理 808
14.8.5.4 イベントの送信 808
14.8.5.5 オブジェクト指向のスクリプト言語 809
14.8.5.6 スクリプトで書かれた有限状態マシン 810
14.8.5.7 マルチスレッドのスクリプト 811
14.9 高レベルのゲームフロー **814**

第5部 まとめ **815**

第15章 まだやることがあるってこと? **817**

15.1 これまでに取り上げた以外のエンジンシステム **818**
15.1.1 オーディオ 818
15.1.2 ムービープレイヤー 819
15.1.3 マルチプレイヤーネットワーキング 819
15.2 ゲームプレイシステム **819**
15.2.1 プレイヤー制御機構 819
15.2.2 カメラ 820
15.2.3 人工知能 821
15.2.4 その他のゲームプレイシステム 822

参考文献 **823**
索引 **827**