Launch

後編　Java を用いたゲームアプリケーションの公開

2024 年 7 月 20 日

Virtual Causality

1　背景・目的

前編でも述べたとおり，本プロジェクトは，実務を経験していない筆者がベンチマークとしてゲームアプリケーションを開発するというものである．後編では，まず，未知のライブラリを扱う場合に必要である，サンプルにおける 1 対 1 の関係に対する実用における 1 対多の関係のような，資料に直接記載されない部分の予測を行う能力を確認する．次に，大学のカリキュラムにも含まれないことが多い，実際に開発した成果物を公開するフェーズを問題無く通過できることを確かめる．

2　知識

2.1　Java

Java は，多くの開発現場で用いられているプログラミング言語である．プログラムをコンパイルしたバイトコードは JVM 上で動作する．ゲームアプリケーションにおける使用例は Minecraft Java Edition が存在する他に，あまり多くない．Java には GUI アプリケーションを開発するための標準ライブラリが十分備わっているものの，Windows の設定のひとつである高 DPI 設定によって，意図しない解像度で表示される不具合があり，ゲームアプリケーションの開発には不向きである．こうした事情から，本作では Java におけるゲーム開発向けのライブラリとして，LWJGL（Lightweight Java Game Library）を使用する．LWJGL は，OpenGL や Vulkan 等，GPU を用いる API の Java バインディングや，画像，音声，3D モデル等の入出力に用いるライブラリ，ネイティブライブラリとの連携を実現するための諸機能を備える．

2.2　Java Sound

Java Sound，または Sound sampled は，Java が持つ音声処理のための標準拡張機能である．いくつかのラインを持つことができるので，効果音を止めずに複数鳴らすことができる．

2.3　Vulkan

Vulkan とは，標準化団体である Khronos Group が策定する，GPU を用いたレンダリングおよび並列計算のための API である．先代となる OpenGL と比較して低レイヤーの操作が可能であり，実行時のオーバーヘッドの低減が期待できる一方，実装は難しく，多くの場合デフォルトまたは暗黙的な設定に依存することができない．

本プロジェクトでは，ライブラリと接する時間を長くとることができ，経験の応用が利くことから，他のゲームエンジンや高レベルのライブラリではなく，Vulkan を採用する．

2.4　GLSL

GLSL とは，GPU が並列に実行するプログラムを記述する言語である．あらかじめコンパイルしておき，中間表現である SPIR-V 形式として保存することで，実行時にかかる時間を短縮することができる．

3　経過

3.1　分割方法が不明かつ直接実行できないタスクへの対応

本作 Java 版の描画機能は，複数種の描画命令機能と描画に用いるリソースの管理機能に分かれる．それらは，モックアップの実装における各種描画命令そのものを実装したものと概ね等しい．つまり，少なくとも本作の描画に必要な条件の範囲を満たす程度に抽象的な機能が複数存在することになるが，当初，このような用途における Vulkan の扱いが分からず，条件の範囲と数を確定することができなかった．そのため，現実的な時間内に効果的なインプットを織り交ぜながらタスクの分割を試行する方法がとられた．インプットを織り交ぜるということは，学習，実務に関係なく当たり前のことである一方，その効率を求めることは難しい．なぜなら，インプットのためには質問または検索が必須だが，そのための用語が既知でなければならないうえに，返答または検索結果の妥当性を検証する能力や検索結果から結論を導き出す推論能力を，より高い水準で問われるからである．

3.2　実装

3.2.1　モックアップからの移植

ゲーム本体の処理については，グローバル変数をローカル変数に置き換えたこと以外，大きな変更は無い．一方，状態管理は専用のクラスを作成して行うことになり，各状態は整数型の状態コード，押下時判定の処理，長押し判定の処理，毎フレームの処理を持ち，ゲームループ内部で必要に応じてそれらの処理が呼び出される仕様へ変更された．

3.2.2　GPU の利用

GPU を利用するプログラムは，通常のプログラムとは異なる部分が存在する．代表的なものとして，メモリの割り当てと，コマンドの実行が挙げられる．メモリの割り当ては，GPU が利用するデータを置く領域，GPU が計算結果を置く領域，ホストまたは GPU か

らのデータを中継する領域などの用途に合わせて要件を満たすフラグを持つ領域に行う必要がある．コマンドの実行はキューを介して行われる．近年の API（Vulkan や Direct3D 12 等）では，ソースコードの見た目どおりに実行される保証があるのはシェーダの中だけであり，キューに投入されたコマンドは GPU の空き状況によって実行順序が変更される可能性があるため，必要に応じて依存関係を指定することで，守るべき最低限の順番を指定しなければならない．

3.2.3　シェーダプログラム

本作では，大半の描画をコンピュートシェーダで行い，最後にクライアント領域全体を覆うポリゴン 2 枚にテクスチャとして貼り付けることで，拡大を実現する．コンピュートシェーダ（パイプライン）を多用する理由は，目的に合わせた 2D の描画命令を多数用意する工数が，グラフィックスパイプラインを用意するより少ないからである．

3.2.4　ソースコードの解説

Java 版のソースコードは分量が多いので，主要な部分のみ解説する．Vulkan を利用する部分に，Vulkan tutorial by Alexander Overvoorde ported to Java[1]を一部引用している．

3.2.4.1　メモリ管理（主に init が付くメソッド等）

LWJGL を用いた実装では，オフヒープバッファまたは構造体にも使用できるスタックへの割り当てを行い，必要な値を代入して API の関数に与えるか，割り当てたバッファを関数に与え，バッファを経由して目的の値を読み出すという操作が発生する．オフヒープバッファはプログラムが終了するまでに手動で解放しなければならないため，初期化を担うメソッドの中で，寿命が長く容量が大きいリソースのために割り当てられることが多い．スタックに割り当てた領域は try-with-resource 文を用いることで自動解放することができる一方，デフォルトは 256KB とサイズに限りがあるので，頻繁に再割り当てが行われる容量が小さい変数のために割り当てられることが多い．

3.2.4.2　ユーティリティ関数（LaunchRenderer3 内の一部メソッド）

Vulkan を用いたレンダラーを実装する際，特にリソースの確保に関する処理において，チュートリアルにならって再利用性の高いユーティリティ関数を作成する．また，複数の変数からなるリソースはレンダラーの内部クラスにまとめ，解放の処理を持たせることで管理を容易にする．

3.2.4.3　シェーダ用のクラス群（Copy が付く各クラス）

本作では，描画命令ごとにシェーダを使い分けるため，シェーダを呼び出すために必要なディスクリプタやパイプライン，テクスチャや描画先など共通のリソースのアドレス，描画命令の引数をシェーダが読み取れる形に変換する処理等をまとめたクラスをシェーダごとに用意する．レンダラーはこれらのクラスからオブジェクトを作り，ゲームシステムが描画命令を使用できるようにする．本作の場合，任意の背景色または背景とピクセルアルファブレンドしたテクスチャの貼り付け，盤面の一括描画，セグメントの一括描画，テ

クスチャからの ASCII 文字の描画等が利用できる．次の表は作成したクラスと用途の一覧である．

表 1　シェーダ用のクラスと描画命令の一覧

クラス	命令	用途
SingleCopy	background	背景のコピー
	copy	テクスチャの背景色付きピクセルアルファブレンドコピー
ASCII8x20Copy	ascii	ASCII 文字の一括描画（最大 40 文字）
CellCopy	cells	盤面の一括描画
ControlCopy	control	キーバインド表示の一括描画
GuideCopy	guide	操作ガイドの一括描画
SegmentCopy	segment	セグメントの一括描画
ColorCopy	rect	指定色をピクセルアルファブレンド

3.2.4.4　デバイスの取得（LaunchRenderer3#createDevice）

Vulkan では，API を経由してコンピュータに接続された物理デバイスを取得し，次に必要な機能を有効化した論理デバイスを作成し，使用する．レンダリング結果を画面に表示するための機能は拡張機能なので，使用できるか確認したうえで有効化しなければならない．当該メソッドでは，デバイスの作成に加え，最後まで変更されないリソースの一部の初期化も行っている．

3.2.4.5　スワップチェーンの作成（LaunchRenderer3#createSwapchain 他）

画面に表示する画像は，ディスプレイの更新周期と同期しないで表示した場合，ティアリングが発生する．本作はスワップチェーンを作り直すことで垂直同期とウィンドウの大きさを切り替えられる設計としている．次の表は関係する変数と定数である．

表 2　スワップチェーンの作り直しを伴う設定に用いる定数

変数	定数	備考
presentMode	VK_PRESENT_MODE_FIFO_KHR	垂直同期オン
	VK_PRESENT_MODE_IMMEDIATE_KHR	垂直同期オフ
extent	width=800	1 倍
	height=900	
	width=1600	2 倍
	height=1800	

3.2.4.6　リソースの読み込み・確保

Vulkan におけるリソースは，多くの場合バッファまたはイメージとよばれるものであ

り，用途や大きさを指定した論理オブジェクトを作成し，論理オブジェクトを利用して物理的に領域を確保する．画像の読み込みには LWJGL に同梱されている stb single-file public domain libraries を用いる．まず，画像データを CPU 側から見える領域に読み込む．次に，CPU，GPU 双方から見える領域に，Vulkan の関数を用いて一時的に領域を確保し，コピーする．その後，GPU 専用の領域に転送し，最後に CPU 側のメモリ，一時的に確保した領域を解放することで，読み込みが完了する．

3.2.4.7　パイプラインの作成（LaunchRenderer3#createGraphicsPipeline 他）

グラフィックスパイプラインは，描画に必要なシェーダをまとめてひとつのパイプラインとして作ることができる．上記のメソッドでは，作成に必要な構造体にデータを格納し，Vulkan の関数を用いてパイプラインを作成している．本作のように頂点シェーダとフラグメントシェーダの 2 つからなる単純なパイプラインでさえ，ソースコードでは 100 行ほどと大きく見えるが，実際には大半が定数を代入する行で構成されていることが分かる．

描画命令の実装に用いるコンピュートシェーダは，シェーダとパイプラインが 1 対 1 で対応しており，一般的なプログラムにおけるリフレクションに近い実装となっている．

3.2.4.8　状態管理用クラス（Launch.Stat）

クラス Stat には状態を表す整数型の値のほかに，然るべきタイミングで呼び出される関数型オブジェクトを 3 つ持つ．これらは抽象クラスのメソッドのオーバーライドによって実現することも可能だが，頻繁に処理を使いまわす本作のような場合は，関数型オブジェクトを用いる方が可視性に優れていると考えられる．

3.2.4.9　効果音再生（initSound 以下）

効果音の再生には，Java の標準拡張である Sound sampled を用いる．

クラス AudioSystem を用いて音声ファイルを読み込む処理は，ユーティリティ関数として readSoundFile メソッドに分離する．音声に関する他の処理に必要な AudioFormat オブジェクトは AudioInputStream から取得しなければならないが，AudioFormat オブジェクトがすべての効果音で共通しているのに対し，AudioInputStream はファイルごとに異なるものを使用する．そこで，メソッドでは AudioFormat の参照に見せかけた長さ 1 以上の配列が渡されたときのみオブジェクトを取得するようにした．

本プロジェクトの制限によって音声素材が不足したことから，いくつかの効果音は既存の音声の逆再生である．reverseSound メソッドにおいて，渡された素材となる音声データを，フレームと呼ばれる単位で分割し，フレームを逆順に入れ替えることで実現している．

読み込んだデータの再生は，SourceDataLine，以下ラインにデータを流すことで行う．ラインはいくつか作成することができ，それぞれに音量などのコントロールを設置することもできる．本作ではラインを 2 本確保し，短い間隔で音声が再生される際に音声が途切れにくくなるよう工夫した．

音量の調整は volume メソッドから行う．コンフィグでは 0〜100 までのリニアな整数値を用いた設定が可能だが，音量がリニアに変化しているように見せるため，常用対数を用いて値を変換したものを設定する．

3.2.4.10　設定の読み込みと保存

コンフィグはすべての値を半角スペース区切りで 1 行に並べ，テキストファイルに保存する．読み込みはゲームシステムの初期化の直前に，保存はゲームシステムの終了処理の最初に行う．

3.3　素材の制作

3.3.1　画像素材

本作は 1 ブロックに使用できる画素数が少なく，ブロックの出現パターンが多岐にわたることから，盤面の背景は黒，ブロックは白，爆弾は灰色，貴重なブロックは彩度が高い色をベースとする，視認性に優れた色使いとした．

ソフトウェアはフリーソフトである AzPainter2，Windows 付属のペイントを使用した．

3.3.2　音声素材

本プロジェクトの制約により，収録に使用できる機材やソフトウェアには限りがあったので，ウェブカメラのマイクを用いて飲み終わった栄養ドリンクの瓶，ウエットティッシュの袋，エアダスターから出た音を収録し，カットしたものを使用した．

収録には Windows 付属のサウンドレコーダー，カット編集にはフリーソフトである AviUtl を使用した．

3.4　テスト

本作の制作において，テストツールは使用していない．そのため，import 以外の外部への依存関係が発生しない範囲でソースコードを切り出し，テスト用のクラスの main メソッドを実行する方法がとられた．

テスト環境は次のとおりである．

表 3　開発・テスト環境

項目	型番等	備考
CPU	Intel Core i7-12700K	x86_64
RAM	DDR5-4800 32GB (16GBx2)	
GPU	Intel UHD Graphics 770	Windows の設定から切り替え
	Intel Arc A770 Graphics (16GB)	
	NVIDIA GeForce GT 1030	
OS	Windows 11 Home (23H2)	
Java	OpenJDK 17	Eclipse 2021-12

3.5　同封物の制作

本作は配布を前提としているので，HTML 形式のマニュアルを同梱する．他に，昨今の情勢を鑑みて，アプリケーションの利用規約を明記する．Web ページは解像度，アスペクト比ともに多くのパターンが考えられるので，解像度に応じて画像の大きさや背景の装飾を切り替える，レスポンシブデザインを一部採用している．

バージョン 1.0.2 より，Java を同梱することとなり，起動用の exe ファイル（ランチャー）を追加した．ランチャーは C 言語で書かれており，execl 関数を用いて起動に必要なコマンドの実行を試み，失敗したときダイアログを表示する．

3.6　公開

本作の公開は，フリーゲーム配信サイトである，Freem において行う予定である．

4　評価

ソースコードの一部にハードコーディングされた箇所や単純な繰り返しが残っていることからも分かるように，より便利なユーティリティ関数の作成や最適化の余地が見られる．

5　まとめ

本プロジェクトでは，実務を想定した訓練としてゲームアプリケーションの開発を行った．今後の課題として，レンダラーを依存関係から切り離し，より高度な GUI を開発するための橋頭堡として整備するタスクが挙げられる．

付録 A： ギャラリー

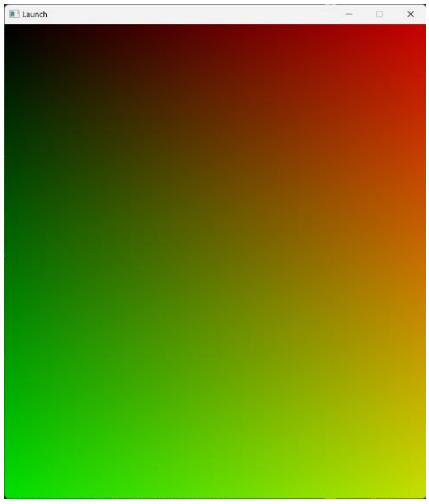

図 1　Hello, Window

図 2　VRAM の中身

図 3　スクリーンショット機能のバグ

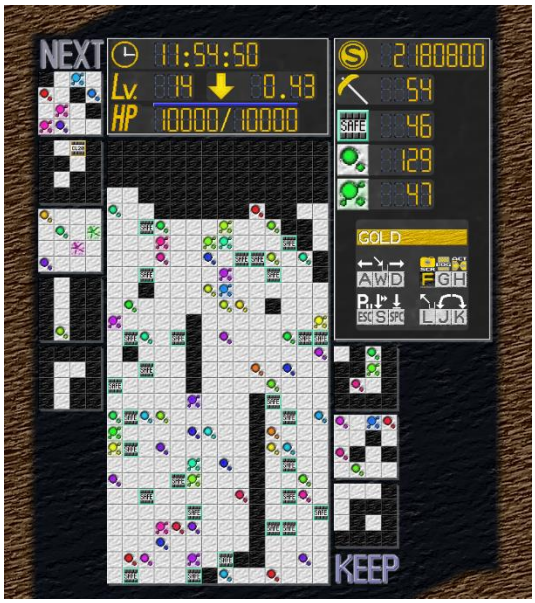

図 4　完成版のスクリーンショット

図 1 は，2 枚のポリゴンを表示したウィンドウである．本作では，テクスチャ用のイメージにコンピュートシェーダを用いて描画し，それをポリゴンに貼り付けることで拡大と表示を行う．各頂点に色を付けることで，正常に表示されていることを確かめている．

図 2 は，テスト画像を読み込んだあとにテクスチャ用の領域を確保したまま初期化しないことで，VRAM の中身を適当に表示したものである．元の画像は左上が 1，右下が 16 だが，正しく表示されていない．

図 3 は，スクリーンショット機能の実装中に発生したバグである．バッファの領域を誤って指定したことが原因とみられる．

図 4 は，完成版をプレイして撮影したスクリーンショットである．

付録 B： アップデートの記録

表 4　バージョン履歴

バージョン	日付	修正された不具合
1.0.0	2024/06/26	-
1.0.1	2024/07/02	● 特定の条件においてチェック漏れが生じ，落下すべきブロックが落下しないもの ● 猶予が-0.01 秒となり，当該条件でテクスチャが正常に参照されないもの
1.0.2	2024/07/10	● コンフィグにおいて体力バーの色設定が正常に反映されないもの ● Java の同梱とランチャーの追加

表 4 は，本作のバージョン履歴と修正の記録である．

付録 C： 使用ソフトウェア一覧

次の表は，本プロジェクトにおいて利用したソフトウェアの一覧である．

表 5　使用ソフトウェア一覧

名称	用途	備考
Eclipse	コーディング，ビルド	Java 版の実装
メモ帳	コーディング	Windows 付属，シェーダの実装
HSP スクリプトエディタ	コーディング，ビルド	モックアップの実装
Google Chrome	テスト	
MSI Afterburner	ベンチマーク	
タスク　マネージャー	ベンチマーク，テスト	Windows 付属
glslangValidator	コンパイル	Vulkan SDK 付属
Vulkan Configurator	テスト	Vulkan SDK 付属
Mingw-w64（GCC）	コンパイル	
コマンドプロンプト	コンパイル	Windows 付属
AzPainter2	素材制作	
ペイント	素材制作	Windows 付属
サウンドレコーダー	素材制作	Windows 付属
AviUtl	素材制作	
メディア　プレイヤー	素材管理	Windows 付属
フォト	素材管理	Windows 付属
OBS Studio	テスト	
Word	ドキュメント作成	Microsoft 365 Personal
javaw.exe	実行環境	JDK 付属
エクスプローラー	ファイル管理	Windows 付属
7-zip	ファイル管理，圧縮	

参考文献

[1] Vulkan tutorial by Alexander Overvoorde ported to Java, Cristian Herrera, SWinxy, https://github.com/Naitsirc98/Vulkan-Tutorial-Java, Jul 2024.