ポートフォリオ説明資料

3D ゲームにおけるカメラ制御及び画面描画の基礎研究

Basic Research of Movement of Camera and Rendering in 3D Game

2026

目次

0. 抄録

3D のゲームについて、プレイヤー及びカメラをクォータニオンによって回転させ、空間上の物体を座標変換と透視投影により描画した。移動と回転の速度は個別の設定が可能で、負の値にすると方向が反転する。設定した値は外部のファイルに書き込むことで、ゲーム終了後も保持される機構となっている。また空間上の物体はランダム生成となっており、描画範囲から大きく逸脱した際に削除され、反対側かつ描画範囲の若干外側に再生成される。

回転は WASD + QE キーで制御しており、設定の変更やゲームの終了はエスケープキーから行う。またタイトルと設定の画面については十字キーで選択、スペースまたはエンターキーで決定となっている。

作品は GitHub にアップロードした。(https://github.com/TCAPG22503004/Portfolio)

1. はじめに

人類は空を目指す生き物である。その歴史は深く、少なくともギリシャ神話の時代には Icarus が太陽を目指して飛行している。近代においても、1903 年に Wilbur Wright & Orville Wright が空を飛ぶことに成功している他、人類以外の話としても例えば *Coccinellidae*（テントウムシ科）の昆虫は上に向かう習性がある。太陽、あるいは空そのものへ挑戦する意志は、遥か昔から現代に至るまで失われていないのである。

昨今の科学の発展は凄まじく、それに伴い人類の飛行技術は大幅に向上した。総合政策局情報政策課 交通経済統計調査室（2025）は、2024 年の国内定期航空輸送の旅客数は 10,702 万人、貨物重量は 60 万 1,720 トン、国際航空輸送の旅客数は 2,022 万人、貨物重量は 151 万 4,802 トンであることを報じている。今や航空は人や物を運ぶ重要なインフラストラクチャーとなっており、飛行機に搭乗する目的は空を飛ぶことよりも長距離の移動が主であると思われる。

しかしながら、空に対するロマンが無くなったわけではない。パラグライダーやハンググライダー等の空を飛ぶアクティビティが存在していることに加え、スターフォックスやエースコンバット等のフライトシューティングゲームも数多く発売されている。翼を持たない種族である以上、翼が必要な領域への憧れはそう簡単に消滅しないものであると考えられる。

それはそうと、12 月の中頃にふと 3D ゲームの仕組みに興味が湧き、実際に作ってみることにした。本資料はその結果をまとめ、反省点を考察することを目的とする。

2. データと方法

コンパイラとして mingw-w64 の x86_64-12.2.0-release-win32-seh-ucrt-rt_v10-rev2.7z を、また対応するバージョンの DX ライブラリ（山田, 2001）をダウンロードした。コンパイル時の bat ファイルは、Nkmrtkhd (2020）の記事を参考に作成した。ファイルの操作は Windows Subsystem for Linux のバージョン 2.6.3.0 を用いた。OS は Ubuntu 22.04.5 LTS(Jammy Jellyfish)である。テキストエディタは Vim を使用した。 データの公開とバックアップを兼ね、作品は GitHub (https://github.com/TCAPG22503004/Portfolio）にアップロードした。

概要としては、カメラすなわちプレイヤーの回転量を表すクォータニオンを保持し、回転した座標系における直方体の座標を用い、透視投影によってスクリーンに描画するという流れとなっている。3 章ではそれぞれの工程について、コードを含めて解説する。

3. 解説

3.0. 試作: test ディレクトリ

始めに Python 及び Pygame を用いて研究と試作を行った。初期では回転行列を使用していた（Matrix*.py）が、調べた結果と先生からのアドバイスに基づきクォータニオンに切り替えている。また動作研究の過程で、一人称視点と三人称視点は二行の書き換え、あるいは if 文を五行程度追加することで切り替えられることが分かった（Camera1st.py, Camera3rdPerson.py）。ゲームの完成系が確認できた（Final.py）後に、C++での制作を開始した。

3.1 カメラ制御

カメラはクォータニオンにより回転させており、quaternion.cpp の関数で計算している。計算式は矢田部（2007）の記事から引用している。すなわち、クォータニオン p とクォータニオン q の積は

$$\tilde{q}\tilde{p} = \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

位置ベクトル r の回転は

$$r' = \begin{pmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{pmatrix} r$$

座標系 r の回転は

$$r' = \begin{pmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{pmatrix} r$$

とした。

3.0 章で述べた試作は（x, y, z) = (右，上，奥）向き正の左手系であるが、z 座標を高さ方向に取る方が一般的であると考え、本制作では（x, y, z) = (右，奥，上）向きを正とする右手系を採用している。カメラの回転はワールド座標に基づいており（void ProductWorld）、ローリングが可能である。対して後述する物体の生成ではローカル座標で回転させているため、積の順番を逆にした関数を別途用意した(void ProductLocal)。

ゲーム内では、(SW, EQ, AD）キーを入力することで（x, y, z）軸を中心に回転が可能である。具体的には、0 番目が $\cos\theta$ で、1 から 3 番目は対応する成分が $\sin\theta$、他の二成分は 0 となるクォータニオンと、現在のクォータニオンとの積を取ることで回転させている。またカメラの移動は、速度 v について（0, v, 0）で表される 3 次元ベクトルを、クォータニオンで回転させることで表現している。キー入力と移動の処理は player.cpp に記載されている。

3.2. 透視投影

空間上の物体について、3.1 章の式で回転させた座標を用い、透視投影でスクリーン上の座標を求めた。三谷（2015）の資料を参考に、perspective.cpp に記載している。

Remi (2014）は透視図法の画角による歪みについて、45 度程度であれば問題が無いことを報告している。そのため本作品においても、左右の視野角は $\pi / 4$ とした。スクリーンの大きさを sx, sy とすると、スクリーンとカメラ間の距離 d、ないし空間上の座標（x, y, z）をスクリーンに投影した座標（X, Y）は、相似比より

$$d = \frac{sx}{\tan\left(\frac{\pi}{4}\cdot\frac{1}{2}\right)}$$

$$X = d\,\frac{x}{y} + sx$$

$$Y = -d\,\frac{z}{y} + sy$$

と表される(図 1)。ただし、空間上の座標は（x, z) = (右，上）向きが正、スクリーン座標は左上原点の（x, y) = (右，下）向きが正であることに注意が必要である。

蛇足であるが、三人称視点はプレイヤーがスクリーンまで移動した状態であると解釈でき、X, Y 共に分母を（y + d）と置き換えることで計算が可能である。

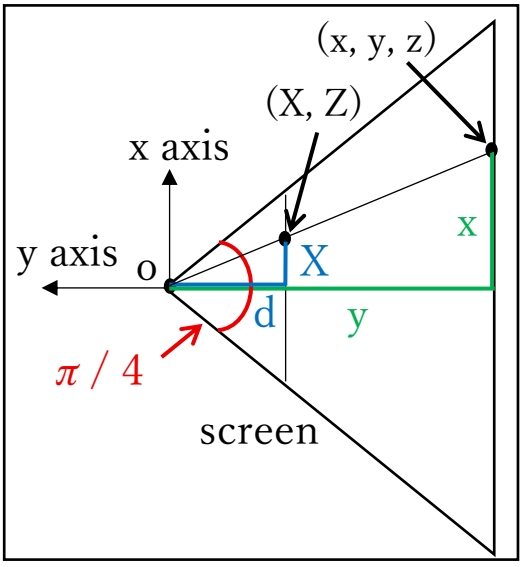

図 1. 透視投影の模式図。
三谷(2015）を参考に作成。

3.3. 空間上の物体

多くのゲームにおいて、マップは事前に作成されたものが用意されている。しかしながら今回は時間と性能の都合上、直方体をランダムに 10 個配置する仕様とした。

直方体は、15 の直線を順に結ぶことで表現した(図 2)。線が重複する分余計な処理が増えているが、3.0 章の試作の際にこちらの方が都合が良く、そのまま受け継いだ形となっている。

ゲーム開始時の配置は乱数で無作為に決定しており、以降は見切れた方向の反対側に新しく作成している（図 3)。例えば図形の相対位置が余白部分のさらに右側（図 3, 赤）となった場合に削除し、左側の余白内（図 3, 青)に新しく生成している。この際、z 座標と図形の大きさはランダムな値となっている。

生成の方向は（0, distance, 0）で表される三次元ベクトルを、プレイヤーのクォータニオンを用いて回転させることで得た。ただし、実際に回転しているプレイヤーに対し、物体は相対的な見かけ上の回転であるため、クォータニオンの角度を負の値とする必要がある。$\cos(-\theta) = \cos\theta$, $\sin(-\theta) = -\sin\theta$ であるため、使用したクォータニオンは虚部、すなわち 1 から 3 番目の要素に -1 を掛けたものである。

以上の処理は、object.cpp と game.cpp で行われている。

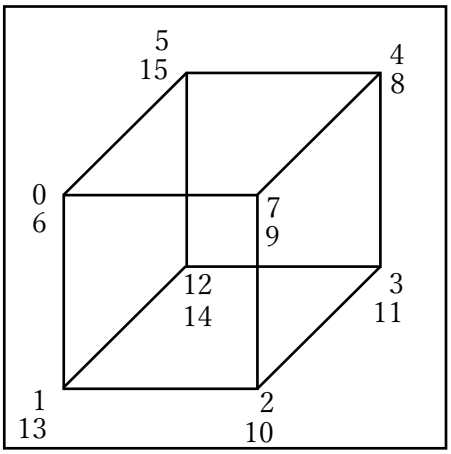

図 2. 直方体の描画順。

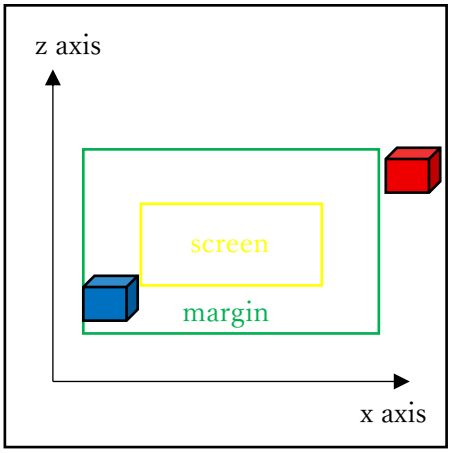

図 3. 生成位置の模式図。
赤は削除、青は生成位置を示す。

3.4. 設定の保存

カメラ操作の仕様として、速度と反転の双方を考慮する必要がある。そこで速度の値をゲーム内で設定し、外部のファイルに書き込むことでゲーム終了後も値が保持されるように設計した。

まず、設定画面を開いた際に、ファイルからそれぞれの値を読み込み表示している。設定画面から離れる際にファイルに書き込み、ゲーム開始時と設定画面からゲームに戻る際にファイルの値を変数に代入している。以上の処理は config.cpp で完結させる予定であったが、ファイルの書き込みが出来なかったため、急遽 write.cpp を後付けで作成した。不具合の原因は不明であるが、解決でき次第 config.cpp に統一し、write.cpp は削除する予定であるため、write のヘッダーファイルは作成していない。

4. 解説

4.1. マジックナンバー

本作品には主に 3 つのマジックナンバーが含まれている。座標、クォータニオン、そして頂点の数である。

座標は xyz の 3 成分、クォータニオンは 4 成分で表現され、恐らく不変である。そのため、頂点の数が 16 であることを直接入力することの是非を考察する。

まず当然のデメリットとして、16 という数字が直感的でないことが挙げられる。通常の直方体では頂点の数は 8 であり、かつその場合でも一目見ただけで理解できるような一般性は無い。まして 16 という数値がいきなり出現しても、何を示しているのかは実に不明瞭である。加えて、値の変更の際に、調整すべき箇所が膨れ上がるというマジックナンバー全体の問題点も含まれている。

以上を踏まえてもなおマジックナンバーを使用したことの理由は二つある。一つはそもそもの普遍化の話であり、もう一つは C++の特性に由来する。

前者に関して、今回は全ての頂点の座標を愚直に変換しており、それなりに痛い計算量となっている。近年のゲームは一つのオブジェクトですらポリゴン数が数千から数万となっており、恐らく私の知らない最適化術が存在しているのではないかと踏んでいる。とどのつまり、この作品では直方体以外を作らないため数値が変わることが無いこと、そして他のプロジェクトでは未知の高速化技術が取り入れられ、このコードは流用されないことが予測されるという二点を以て、この作品のみで使用される値としてマジックナンバーを使用している。

そして後者について、C++の配列はコンパイル時にメモリを確保しなければならないという制約が影響している。変数で動的に配列を作成するためには new を使用する必要があるが、この手法で作成した配列は関数を返す前に delete で削除しなければメモリがリークしてしまうため、関数を引き継いで使用することができない。また n 番目の要素に直接アクセスする場合と new の相性が悪いことも気がかりである。static const int を使用する場合も、結局のところ該当する全てのヘッダーファイルで定義が必要であり、手間がかかることに変わりはない。尤も、私は C++に明るくないため、もしもこれらを解決する画期的な手法が存在しているのであれば、是非ともご教示いただきたい。

4.2. 反省点 1: ヘッダーファイルの煩雑化

今回の作品では、複数の関数にまたがって使用されている変数はグローバルに定義している。これが災いして、ヘッダーファイルの変数の数がかなり多く、視覚的に望ましくない形となっている。

解決策として、ループの度に値が変動する変数はループの中で定義し、他の関数の引数として渡す方法が挙げられる。現時点での関数の引数はほとんどが 0 であり、多少増やしても見栄えの悪化はあまり深刻化しないことが予測される。

加えて、初期化をヘッダーファイルで行わない手段もある。一部のクラスでは、DX ライブラリの関数を使用するため、Init 関数を定義している。これを全てのクラスに適用し、パラメータは cpp ファイルで調整するというルールに統一する手段も有効であると考えられる。

4.3. 反省点 2: 作業の順序

3.0 章で試用品を作成したこともあり、C++の本制作では各部品を独立して作成し、後から組み合わせる手法を採った。しかしながら、全体の進捗が分かりにくく、モチベーションへの悪影響があった。加えて、全てを組み合わせてからでないと関数の動きを見ることが出来ず、不具合発生の由来を特定する時間が増加したという問題もあった。試作の時と同様に、仕様を一つずつ順に確認しながら制作を進めた方が、より効率良く進めることが出来たのではないかと評価する。

4.4. 当資料について

一般に履歴書のアピールポイントは、簡潔で分かりやすく、一目見るだけで面白さが伝わる形式が望ましいとされている。これは企業の採用担当は一日に数十から数百の選考を進めなければならず、一人ひとりに充てられる時間がごく僅かであることが原因である。風の便りによると、人事が履歴書に目を通す時間は、一人当たり二秒であるという。もしもそうであるのならば、細かい文字で長々と説明を続けるよりも、一言二言を強調してアプローチをかける戦略が採られることは、合理的で正しいといえる。

ポートフォリオの説明資料についても同様だろう。PowerPoint を用い、適宜図を入れながら解説文を見栄え良く配置することが、無難な選択肢としてよく採用されているように思える。あるいは A4 一枚の大きさに全てを詰め込んでも良いかもしれない。間違っても word なんてソフトウェアはまず使用されないし、論文のフォーマットに則った資料など言語道断であろう。

ところで、私は”やるべきことをやる”をモットーとする人間で、それが面白そうであれば猶更である。就職活動にあたり、プログラムと解説資料を提出する必要がある。説明という言葉から真っ先に連想される単語といえば、当然卒業論文である。こうして出来上がってしまったのが当資料である。

5. まとめ

何となくの思い付きから、3D ゲームの基礎部分を研究し、ゲームの形に落とし込んだ。クォータニオンや透視投影、及び関連する複数の概念については知らない分野であったため、1 月の終わりごろまではこれらの理解に勤しんだ。その後に得られた知見を活かし、

C++で動くモノを創り上げた。保守性や可読性の問題が浮き彫りとなったため、今後の課題として改善したい。

6. 謝辞

当ファイルは GitHub にアップロードする都合上、本名は伏せさせていただきます。
本研究を進めるにあたり、回転系や数学等の専門的な相談に快く乗って下さった Y 先生に深く感謝申し上げます。また担任の T 先生には、普段の学校生活を陰ながらサポートいただいております。記して感謝申し上げます。最後に、常日頃からお世話になっている全ての方々へ感謝の意を表します。

7. 参考

三谷 純（2015）. コンピュータグラフィックス基礎 第 3 回 3 次元の座標変換. https://mitani.cs.tsukuba.ac.jp/lecture/old2015/cg/03/03_slides.pdf ,（参照 2026-02-13）.

Nkmrtkhd（2020）. Windows のバッチのループ内で変数を置換する. *Qiita.* https://qiita.com/nkmrtkhd/items/100949962a3b216fe524 ,（参照 2026-02-13）.

Remi（2014）. 歪みの原因 – 画角. *パースフリークス.* http://www.persfreaks.jp/main/aov/dist2/ ,（参照 2026-02-13）.

総合政策局情報政策課 交通経済統計調査室（2025）. 航空輸送統計速報(令和 6 年(2024 年)分). 国土交通省. https://www.mlit.go.jp/report/press/content/001876790.pdf ,（参照 2026-02-12）.

山田 巧（2001）. DX ライブラリ置き場. https://dxlib.xsrv.jp/ ,（参照 2026-02-12）.

矢田部 学（2007）. クォータニオン計算便利ノート. *MSS 技報・Vol. 18.* https://www.mesw.co.jp/business/report/pdf/mss_18_07.pdf ,（参照 2026-02-13）.