概要

ゲーム名:
ジャンル
製作期間: 6 カ月

環境

OS	:Windows10
開発環境	:VisualStudio 2019
言語	:C/C++
使用ライブラリ	:DX ライブラリ、concurrentqueue

作品説明

この作品は技術デモとして作りました。
メモリ管理とマルチスレッドプログラミングに挑戦し、それらを用いたゲームを製作しました。

操作

移動:WASD
射撃:右クリック(マウスカーソルの方向)

製作期間

22 年 9 月～23 年 5 月
ほとんどの時間をメモリアロケータやマルチスレッドなどに費やしました
その後、作ったものからどういったものがいいかを考え、現在のゲームになりました。

プログラムのアピールポイント

メモリアロケータ(Lib¥GameSystem¥Allocator)

exe の起動時にまとめてメモリを確保しゲームで使うメモリのほとんどを
アロケータのメモリとスタックメモリで扱えるようになりました。
またオブジェクトの種類に応じてメモリの配置場所を変えることによって Enemy や Collision などをまとまった場所に配置でき、
現在どの種類のオブジェクトがどの程度メモリを消費しているのかの表示や、
特定の種類やタグを指定して対象オブジェクトを削除することができるようになり、
メモリリークの発生を抑えることができるようになりました。
また、オブジェクトをまとまった場所に配置できるようになったことでキャッシュヒット率の向上を期待できるようにしました。
デバックでの実行時にメモリリークが起きたファイルと行を表示してメモリの開放忘れを検出できるようにしました。

マルチスレッド(Lib¥GameSystem¥Thread)

マルチスレッドプログラミングを用い、ゲーム内の各オブジェクトの処理を並列で実行できるようにしました。
これにより学校の環境で六百程度のオブジェクトなら 60FPS を保った状態でステージなどとの当たり判定などを実行できるようになりました。
関数ポインタを、キューを利用して受け取ってそれを各スレッドで実行していく形式になっています。
今回のプロジェクトでは使っていませんが、バックグラウンドで実行する処理にも対応しているので、オートセーブなどの実装も可能です。

マネージャー関係(Lib¥ManagerBase.h)

マネージャーの基底クラスを定義しリストで管理するようにしました。
同時にシングルトンを継承することも可能にしました。
これによりコードの冗長性を回避しつつ、新しいマネージャークラスを増やしても 1 行の追加だけで対応することができるようにしました。
またマルチスレッドとの兼ね合いもあってマネージャーにインデックスをつけてその順番で競合するのか、や処理の順番を制御するようにしました。

レンダークラス(Lib¥Render)

描画関係をデバック用の文字描画などを除いてまとめました。
3D モデルの描画設定(最前面、影の表示の有無)や
2D 画像の描画順制御などを行っています。
最初はレンダリングのマルチスレッド化のために制作したのですが、
内部での競合の原因特定ができず処理と描画を同じスレッドで行っています。

コリジョン(Src¥Collision)

当たり判定を当たった瞬間、当たっている間、離れた瞬間の 3 種類に分けてとれるようにしました。
前フレームの当たり判定情報と現フレームの当たり判定情報を格納してその二つの情報から該当判定を行っています。

オペレーターのオーバーロード(Lib¥ Math¥ MyMathData.h 等)

ベクトルや行列の計算をオペレーターに定義し、+でベクトルの加算などを行えるようにしました。

行列計算の高速化(Lib¥ Math¥ MyMathData.h)

SIMD 命令を用い、行列の計算をベクトル化しました。
これにより行列の内積の場合、通常の for 文を書く方法の半分程度の実行ステップ数になりました。

プログラムの苦労したところ、失敗したところ

メモリアロケーター

初めて低レイヤーに近いところを触ったため
C/C++や現代のコンピュータのメモリに関する仕様から理解しなおす必要がありました。
メモリの管理方式に関する仕様や、メモリ確保、開放時に生まれる空白を埋める為のメモリの移動処理への対応などを考えるのに時間を使ってしまいました。

マルチスレッドプログラミング

メモリアロケータの後に実装したためその時に得た知見が理解するための足掛かりになりました。
スレッドというものに対する理解や、それらを活用するための方法を理解していくのに苦労しました。
実装していくにあたってライブラリに対する理解が足りていなく、何の関数を呼べば競合が起きるかを手探りで確かめて、
考察していって問題点を洗い出だしていきました。
実装するにあたってできるだけロックフリーにしたかったのですが、リストなどの関係でクリティカルセクションなどを利用するしかなく、
結果としてロックが発生してしまっています。
ロックの待ち時間が多いため現在のプロジェクトはまだ CPU の待機時間が多くなってしまっています。

注意したところ

constexpr でマジックナンバーを減らす。
リテラル演算子で回転値のラジアン変換などの冗長性を排除する
関数ポインタを用いて分岐処理の軽量化、冗長性の排除を行う
競合を起こさないために外部変数にアクセスする処理を別の関数で実行するように

今後のゲーム制作に活かしたいこと

ライブラリに時間をとられすぎて、ゲームに回す時間が少なくなってしまいました。
また、今回のゲームはマルチスレッドで実行するには少しオブジェクト単体の処理負荷が軽いため、そのあたりが課題だと感じました。