分子動力学シミュレーション ソフトウェア mdspass
高速化設計書

2024 年 3 月 22 日
株式会社知能情報システム
担当: 松枝敦夫

[目次]

[変更履歴]

日付	版	担当	変更内容
2024/03/22	1.0.0	松枝敦夫	新規作成。

1 概要

この文書は、分子動力学シミュレーション ソフトウェア mdspass（以下、mdspass プログラム） の描画高速化作業における改修点と高速化において追加された構造体、クラスについて説明しています。

2 高速化手法

ここでは mdspass の描画の高速化手法について説明します。高速化は以下の 2 つの方法にて行いました。

1. OpenGL API の呼び出し回数の抑制
2. 描画されるポリゴンの削減

以下では各方法について説明します。

2.1 OpenGL API の呼び出し回数の抑制

高速化前のコードでは、描画処理の中で原子の数だけループが繰り返され、その中で個別に描画命令（glBegin 関数、glutSolidSphere 関数、glutWireSphere 関数、gluCylinder 関数等） が呼び出されていました。本改修ではバッファー上に作成した全ての原子分のメッシュ、線分の座標データ等一式を、glDrawElements 関数で一括して描画することで描画速度を改善しました。

高速化前のコードにおける上述のループは、原子の数が多いときにその全てのループを処理した場合、GPU への命令の実行回数が膨大な数となります。GPU への命令はオーバーヘッドが大きいため、原子の数の増加とともに描画命令の実行回数が増えると描画速度が低下します。本改修では各ループでは同じパラメーターを用いた描画しか行われない点に着目し、ループ単位であらかじめメッシュ データをメイン メモリ上に作成し、各ループが終了した後にメッシュ データを一括して描画するように改修しました。これにより、描画命令は原子の数の影響を受けることなく、1 フレームあたり十数回程度の実行で済むようになりました。

2.2 描画されるポリゴン数の削減

高速化前のコードでは全ての原子が常に同じ数のポリゴンを用いて描画されていました。本改修では小さく表示されている、あるいは表示されない原子のポリゴン数を減らすことで、描画速度を改善しました。

具体的には高速化前のコードでは 1 原子あたりのポリゴン数は 400 強となっており、36000 原子を表示する場合、フレーム毎に 15 百万以上のポリゴンを描画する必要があり

ました。しかしながら、画面上で数ピクセル程度の面積しかない球体であればほとんどのポリゴンは表示されず、描画速度を低下させるだけとなります。そのため、原子が小さく表示されているときはポリゴン数を減らし、大きく表示するときにはポリゴン数を増やすように改修しました。ただし、大きく表示したときに全ての原子のポリゴン数を均一に増やすと全ポリゴン数が高速化前のコードと同じ程度にまで増加してしまい、描画速度が低下する恐れがあります。そのため、小さく表示されるか、画面上に表示されない可能性が高い原子については、描画に使用するポリゴン数を減らす処理を入れることで、描画対象となるポリゴン数の抑制を図りました。

3 追加したクラス、構造体

ここでは高速化作業にあたって mdspass プログラムに追加したクラスおよび構造体について説明します。なお、ここで説明する構造体は MeshRenderer クラスの中でのみ使用しています。

3.1 MeshRenderer クラス

OpenGL を用いた描画を実行するクラスです。このクラスはあらかじめメンバー変数に登録された描画対象となるメッシュ、線分のデータを一括して描画します。なお、描画する毎にメモリの解放と確保を行うと描画処理全体の速度が低下します。そのため、確保したメモリは可能な限り再利用し、メモリが足りなくなったときのみメモリの再確保を行うようにしています。

ここでは public なメンバー関数の概要のみを説明します。MeshRenderer クラスのメンバー変数は全て private としているため、ここでは説明しません。

A) clear 関数
メンバー変数上に登録されているメッシュ、線分データを全て削除します。メモリの解放は行いません。

B) appendSphere 関数
原子を表す球メッシュを登録します。色情報を持ちます。

C) appendArrowVector、appendArrowPosition 関数
矢印メッシュを登録します。色情報と法線情報は登録されません。

D) appendTriangle 関数
三角形（ポリゴン）を登録します。色情報と法線情報は登録されません。

E) appendLine 関数
線分を登録します。色情報と法線情報は登録されません。

F) appendCylinder 関数

円筒を登録します。法線情報は生成しますが、色情報は登録されません。

G) drawMeshesWithColor 関数
登録済みの色情報をもつメッシュを全て描画します。appendSphere 関数で登録したメッシュのみが対象です。他の関数で登録されたメッシュがある場合、正常な描画はできません。

H) drawMeshesWithNormal 関数
法線データを持つメッシュを全て描画します。appendSphere 関数と appendCylinder 関数で登録されたメッシュのみが対象です。他の関数で登録されたメッシュがある場合、正常な描画はできません。

I) drawMeshses 関数
メッシュを全て描画します。色情報と法線データは使用しないため、どの関数で登録されたメッシュであっても描画可能です。

J) drawWireFrame 関数
メッシュのエッジのみを描画します。色情報と法線データを使用するため、appendSphere 関数で登録したメッシュのみが対象です。他の関数で登録されたメッシュがある場合、正常な描画はできません。

K) drawLines 関数
登録済みの線分を全て描画します。

3.2 Vertex 構造体

メッシュ、線分を構成する頂点の座標データを格納する構造体です。

メンバー変数	型	説明
x	GLfloat	X 座標
y	GLfloat	Y 座標
z	GLfloat	Z 座標

3.3 Face 構造体

メッシュを構成する三角形（ポリゴン）の 3 頂点を格納する構造体です。頂点の指定は Vertex 構造体の配列のインデックスを用います。

メンバー変数	型	説明
v1	GLuint	三角形の 1 つめの頂点のインデックス
v2	GLuint	三角形の 2 つめの頂点のインデックス
v3	GLuint	三角形の 3 つめの頂点のインデックス

3.4 Line 構造体

線分を構成する 2 頂点を格納する構造体です。頂点の指定は Vertex 構造体の配列のインデックスで行います。

メンバー変数	型	説明
v1	GLuint	線分の 1 つめの頂点のインデックス
v2	GLuint	線分の 2 つめの頂点のインデックス

3.5 Color 構造体

頂点の色情報を保持する構造体です。

メンバー変数	型	説明
r	GLfloat	R 成分の輝度 (0.0 - 1.0)
g	GLfloat	G 成分の輝度 (0.0 - 1.0)
b	GLfloat	B 成分の輝度 (0.0 - 1.0)
a	GLfloat	不透明度 (0.0 - 1.0)

3.6 Normal 構造体

頂点の法線ベクトルを保持する構造体です。

メンバー変数	型	説明
x	GLfloat	ベクトルの X 成分
y	GLfloat	ベクトルの Y 成分
z	GLfloat	ベクトルの Z 成分

4 高速化の結果

ここでは mdspass プログラムの改修前後の描画速度について説明します。

4.1 計測方法

OpenGL を用いた描画を行う最上位の関数である myGlutDisplay 関数全体の処理時間を描画速度として、改修前後で比較しました。処理時間の計測は標準ライブラリの std:chrono を用いて ms 単位で行いました。表示対象としたデータは "CONFIG_A136000" ファイルとしました。

処理時間の計測に用いた環境は以下の 2 種類です。

A) Windows 10 Pro (22H2)

CPU: Intel(R) Core(TM) i7-11700 @ 2.50GHz
メモリ容量: 16GB
GPU: Intel(R) UHD Graphics 750 (CPU 内蔵 GPU です。)

B) Ubuntu 22.04
上記 Windows 環境をホストとする VirtualBox 上にインストールしたものを用いました。プロセッサー数は 2、メモリ容量は 6 GB としました。

ウィンドウ サイズ、表示倍率ともに描画速度に影響を与えるため、計測にあたってはそれぞれの初期値（ウィンドウサイズは幅、高さともに 600 ピクセル、表示倍率は 1.0）を用いました。また、UI ウィンドウが原子描画を行うウィンドウを隠していた場合、データを読み込む前にウィンドウの位置を調整し、原子描画を行うウィンドウが全て画面に表示されるようにしました。

計測はフレーム単位で行いました。条件毎に描画時間を 10 フレーム分取得し、その平均値を 1 フレームの描画時間としました。ただし、Ubuntu 22.04 上で高速化前のコードを計測した場合のみ、7 フレーム分の計測時間の平均値となっています。

4.2 計測結果

計測結果を平均 ± 標準偏差で示します。単位は ms です。

	Windows 10 Pro	Ubuntu 22.04
高速化前のコード	578.9 ± 16.2	2302.7 ± 96.6
改修後コード	40.3 ± 3.8	153.3 ± 7.4

なお、ウィンドウ サイズを大きくすると同時に表示倍率を上げると、描画対象のポリゴン数及びピクセルが増加するため、描画速度が 100 ms 前後まで低下しました。これは Windows でのみ確認しましたが、Ubuntu においても同様の速度低下は生じるものと思われます。また、倍率変更時等、描画対象のポリゴン数が急激に増加する操作を行うと、ポリゴンを格納するためのメモリを確保するため、一時的な速度の低下が発生しました。