ソフトウェアRenderingEngineの並列処理による高速化

065725F 金城裕　指導教員：河野真治

1　概要

近年、CPU の消費電力増加や、クロック周波数対効果の停滞により、 CPU コア数は増加傾向にある。コア数の増加にともない処理速度の向上が見込める。しかし、 CPU のコア数が増加すると、コア数に見合った並列プログラミングを行う必要がある。

並列プログラミングを行う際、Amdahl 則より、並列化を意識してプログラミングしなければならない。しかし、並列化を問題毎に毎回考えるのは面倒だ。ならば並列プログラミングを裏でサポートしてくれるソフトウェアを作ればよい。

研究では、Cell の並列プログラミングサポートフレームワークを開発する。当研究室で開発した Cerium は現在十分な並列化、また実行速度をサポートできていない。Cerium、特に RenderingEngine 部分を改良し、信頼できる並列化サポートフレームワークを目指す

2　Cell

Cell は「ヘテロジニアス・マルチコアプロセッサ構成」を採用し、1 基の制御系プロセッサコア (PPE:PowerPc Processor ELement) と 8 基の演算系プロセッサコア (SPE:Synergistic Processor Element) で構成される。各プロセッサコアは、EIB (Element Interconnect Bus) と呼ばれる高速なバスで接続されている。また、EIB はメインメモリや外部入出力デバイスとも接続されていて、各プロセッサコアは EIB を経由してデータアクセスをおこなう。

この 2 種類の CPU をプログラマ自身が用途に合わせて適切に使い分けるように考慮する必要がある。

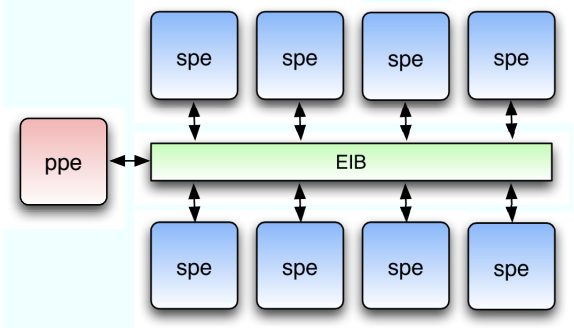

図 1: Cell プロセッサの構成

3　Cerium

当研究室では Cerium と呼ばれるゲーム開発フレームワークがあり、以下の 3 つの要素から構成されている。

- SceneGraph
- Rendering Engine
- Task Manager

Cerium は独自に RenderingEngine を持つ。ゲーム中のオブジャクトの振る舞いやルールは SceneGraph で管理し、それらの動きやレンダリングの処理を動的に SPE に割り振るカーネルとして、TaskMnager が用いられる。TaskManager は、Task と呼ばれる、分割された各プログラムを管理する。Task の単位はサブルーチンまたは関数とし、Task 同士の依存関係を考慮しながら実行していく。現在 Cerium は linux,macosx 上で動作し、コンパイル方法によって Cell の spe を使うかどうかを選択できる。

4　RenderingEngine

RenderingEngine では、SceneGraph から、実際に表示するポリゴンの抽出、ポリゴンから Span の生成、Span に RGB をマッピングし描画する部分と 3 つに分ける事ができる。ここでいう Span とは、ポリゴンに対するある特定の Y 座標に関するデータを抜き出したものである。

4.1　光源

RenderingEngine で、未実装だった光源の計算を実装した。各オブジェクトには自身の座標や親子関係などの情報を持っており、その中に法線がある。法線と光のベクトルとの内積を rgb にかけ算することにより光の計算を行っている。以下に抜粋したコードと、光源の計算をした画像を示す。

```
unsigned char rgb[4];
int light_rgb;
int flag;
float normal_vector[4] = {normal_x,normal_y,
                          normal_z,0};

float light_vector[4] = {0,0,-1,0};
float inner_product;

// 引数で受け取った color の rgb 情報の抜き出し
rgb[0] = (color & 0xff000000) >> 24;
rgb[1] = (color & 0x00ff0000) >> 16;
rgb[2] = (color & 0x0000ff00) >> 8;
rgb[3] = (color & 0x000000ff);

// 法線ベクトルと光源ベクトルとの内積をとる
inner_product = innerProduct(normal_vector,
                             light_vector);
// 内積がマイナスの場合は色がない。
flag = (inner_product > 0);

// 内積を rgb にかけていく
rgb[0] = (unsigned char)(rgb[0]*inner_product*flag);
rgb[1] = (unsigned char)(rgb[1]*inner_product*flag);
rgb[2] = (unsigned char)(rgb[2]*inner_product*flag);

//計算した rgb を light_rgb にまとめる。
light_rgb =   (rgb[0] << 24)
            + (rgb[1] << 16)
            + (rgb[2] << 8)
            + (rgb[3]);
```

図 2: 光源計算のコード

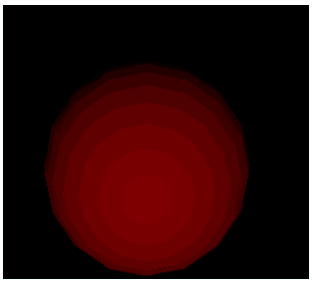

図 3: 光源計算をした描画画像

現在、光源は ppe だけを使った fifo 形式のみに実装されている。

4.2 高速化

Cerium の RenderingEngine は描画する対象がない部分も毎フレーム描画計算をしている。それでは計算する必要のない部分も計算してしまい、無駄な時間が生じる。そこで、描画する対象がない部分の計算は行わないようにすれば高速化が望める。

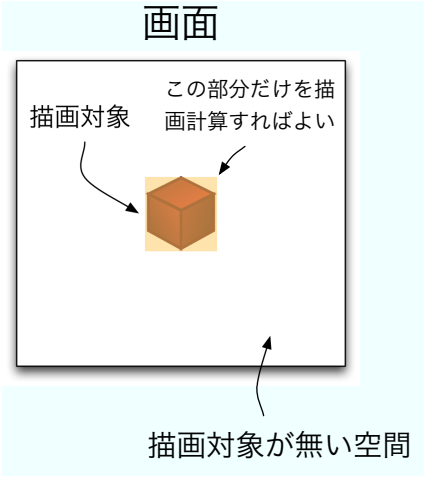

図 4: 描画計算の範囲

5 今後の課題

光源が fifo 形式でのみ有効になるので、Cell を使った並列処理をするときにも動作するように実装する。RenderingEngine での描画計算の際に、描画対象がない部分は計算しない実装に改良する。

参考文献

[1] 宮國渡 ”Implementation of Fine-grain Task Manager for Cell” 平成 20 年度 学位論文 (修士)

[2] 多賀野海人 ”並列プログラミングを用いたゲームフレームワークの設計と実装” 2008 年 卒業研究中間報告資料

[3] fixstars: http://cell.fixstars.com/ps3linux/index.php/メインページ