情報処理学会論文誌 Vol. 50 No. 2 1234–8 (Feb. 2009)

Vol.2009-ARC-183 No.38
Vol.2009-OS-111 No.38
2009/4/24

Cell Task Manager Cerium のSPU内データ管理

多賀野海人[†4] 宮國 渡[†1]
河野真治[†2] 小林佑亮[†4]

PlayStation 3 では、搭載されている Linux を用いてゲームを開発することができるが、GPU の詳細が公開されていないため、Frame Buffer 上に描画する必要がある。Frame Buffer 上の描画は非常に低速である。本研究では Cell 用に開発した Task Manager を用いて SPE 上のデータ管理を行い、Cell プログラミングの並列度の確保を目的とする。

Data management in SPU on Cell Task Manager Cerium

Kaito TAGANO,[†4] Wataru MIYAGUNI,[†1]
Shinji KONO[†2] and Yusuke KOBAYASHI [†4]

In PS3, the game can be developed by using installed Linux. However, because details of GPU are unpublished, it is necessary to draw on Frame Buffer. Drawing on Frame Buffer is very low-speed. In this research, with Task Manager which we developed for Cell, we perform data management in SPE, and It's intended that getting a parallelism of Cell programming.

†1 琉球大学理工学研究科情報工学専攻
Interdisciplinary Infomation Engineering, Graduate School of Engineering and Science, University of the Ryukyus.
†2 琉球大学工学部情報工学科
Infomation Engineering, University of the Ryukyus.
†3 琉球大学理工学研究科情報工学専攻
Interdisciplinary Infomation Engineering, Graduate School of Engineering and Science, University of the Ryukyus.
†4 琉球大学工学部情報工学科
Infomation Engineering, University of the Ryukyus.

1. 研究の目的

PS3 上の Cell[1] に搭載されている SPE は Local Store (256KB) にしかアクセスできず、メインメモリには直接アクセスすることができない。メインメモリにアクセスするには Memory Flow Controller を用いて Direct Memory Access 命令を送らなければならない。またこの DMA には待ち時間が発生する。待ち時間の間 SPE が動作しなければマルチコアプロセッサのパフォーマンスが極端に下がる。

本研究では、SPE 内のデータ管理を行うことによって Cell プログラミングの並列度を確保する手法を提案する。

本稿では、本研究室で開発した Cerium[2] というの Rendering Engine を用いたゲームプログラミングを例題とする。描画するオブジェクトに用いられる Texture データは、SPE の Local Store に収まりきらない場合があるので、データを分割して転送、処理する必要がある。既に SPE 内にデータが転送されている場合、そのデータの管理を SPE 内で行うことによって、DMA 転送待ち時間の間も SPE 内での処理を継続させる。

2. Cell

Cell[1] は、マルチコア CPU の 1 つで、構成は「ヘテロジニアス・マルチコアプロセッサ構成」である。汎用的な用途に対応した 1 種類のコアを用意するのではなく、制御系プロセッサコア (PPE) と演算系プロセッサコア (SPE) という異なるコアを用意している。

2.1 Cell の構成

Cell はメインプロセッサである 1 基の PowerPC Processor Element (PPE) と 8 基のデータ処理プロセッサアーキテクチャー Synergistic Processor Element (SPE) からなる非対称なマルチコアプロセッサであり、高速リングバスで構成されている。[1]

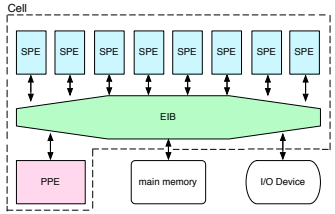

図 1 Cell の構成要素

Vol.2009-ARC-183 No.38
Vol.2009-OS-111 No.38
2009/4/24

2.2　PPE

PPE は 複数の SPU をコアプロセッサとして使用することができる汎用プロセッサである。オペレーティングシステムの役割であるメインメモリや外部でバイスへの入出力制御に加えて、SPE を制御する役割も担っている。PPU (PowerPC Processor Unit) は、PPE の演算処理をおこなう核となるユニットで、PowerPC アーキテクチャをベースとした命令セットを持つ。PPSS (PowerPC Processor Storage Subsystem) は、PPU からメインメモリへのデータアクセスを制御するユニットである。

本研究で用いた PS3Linux では、6 個の SPU を制御することができる。

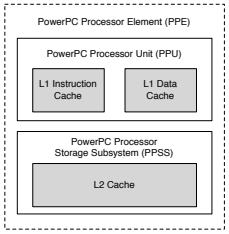

図 2　PowerPC Processor Element (PPE) の構成要素

2.3　SPE

SPE は、計算を単純に繰り返すマルチメディア系の処理を得意とする演算系プロセッサである。SPU (Synergistic Processor Unit) は、SPE の演算処理をおこなう核となるユニットで、各 SPE 上に搭載されている。SPU は、PPU とは異なる独自の命令セットを持つ。また、LS (Local Store) という 256KB のメモリを持ち、メインメモリへのアクセスは MFC (Memory Flow Controller) へチャンネルを介して DMA (Direct Memory Access) 命令を送ることで行われる。SPU は直接メインメモリへアクセスすることはできない。

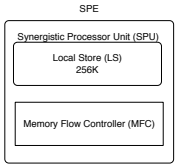

図 3　Synergistic Processor Element (SPE) の構成要素

2.4　DMA (Directi Memory Access)

DMA (Direct Memory Access) 転送とは、CPU を介さずに周辺装置とメモリとの間で高速にデータ転送する手段である。Cell では、DMA 転送を利用してメインメモリと LS 間のデータ転送をおこなう。DMA 転送の実行は、基本的に SPE プログラムで制御されます。SPE プログラムが、メインメモリと LS との間で DMA 転送を実行する手順は以下のとおりである。

(1)　DMA 転送の命令発行
(2)　メインメモリと LS 間 の DMA 転送の実行
(3)　DMA 転送の完了待ち

DMA 転送には待ち時間が存在する。

2.4.1　SPE 内の処理の様子

(1)　Task のデータを読み込む
(2)　読み込んだデータの処理を行っている間に次の Task のデータを読み込む
(3)　処理したデータの転送の間に、2 で読み込んだデータの処理、次の Task のデータの読み込みを行う。

＊　Task とはプログラムも含む、処理に必要なデータのことを指す。

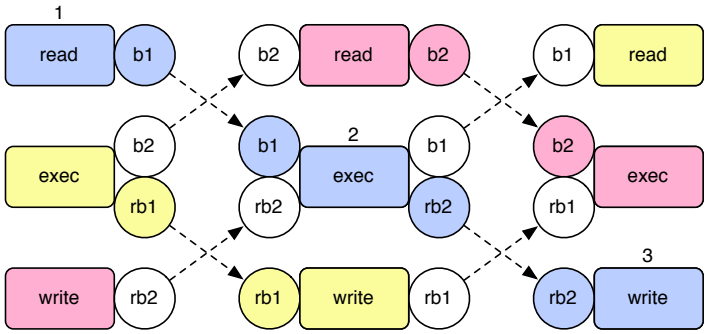

図 4　SPE 内のパイプライン

Vol.2009-ARC-183 No.38
Vol.2009-OS-111 No.38
2009/4/24

3. Cerium

本論文で示しているゲームフレームワークは学生実験での使用を目的としている。現在、学生実験では Playstation 3 を用いたゲームプログラミングを行っている。

我々は独自の Rendering Engine を開発した。その Rendering Engine を Cerium[2] としている。Cerium は以下の 3 つの要素から構成されている。

- SceneGraph
- Rendering Engine
- Task Manager[4]

3.1 SceneGraph

ゲームの中の一つの場面 (Scene) を構成するオブジェクトやその振る舞い、ゲームのルールの集合を SceneGraph とする。SceneGraph の各ノードがゲームの一部であるオブジェクトの振る舞いやゲームのルールとなり、ノードをたどり実行することでゲームの中の一つの場面となる。SceneGraph はゲームプログラムとしての条件を満たす物なので、一つの SceneGraph で小さなゲームと言える。

3.2 Rendering Engine

Rendering Engine は Polygon から Span を生成する部分と Span に RGB をマッピングし描画する部分と二つに分けることが出来る。Span とは、Polygon 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。

3.2.1 Cerium 内の Rendering

Polygon の Span から、描画に必要な Texture データを計算して SPE に転送し、Rendering を行う。

一度に SPE に転送できるデータ量は Local Store の容量が 256KB なので制限される。処理に必要な Texture データが SPE 内に無い場合があるので、その都度転送する必要がある。このため、図 4 で示したパイプラインがうまく働かないという問題がある。

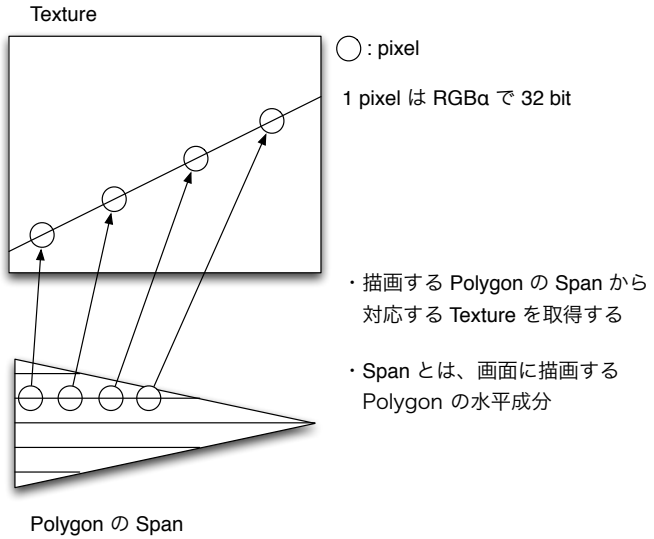

図 5　Rendering

3.3 Task Manger

Task Manager は、Task と呼ばれる分割された各プログラムを管理するライブラリである。Task 同士の依存関係を考慮しながらメモリ上にマッピングし、SPE 上ではそのプログラムを DMA 転送によりロードする。SPE は 256KB という小さなデータ量しか持てず、大量のプログラムを全て SPE 上に置いておくことはできない可能性がある。そのため、必要な時に必要な実行プログラムだけが SPE 上にロードされていることが望ましい。

4. Rendering 部分の高速化

4.1 Texture の分割

SPE の Local Store は 256KB しかないので、Texture データを一度に転送すると容量を超えてしまう可能性がある。そこで、描画に必要な Texture データを分割し、転送するという手法を用いる。

Vol.2009-ARC-183 No.38
Vol.2009-OS-111 No.38
2009/4/24

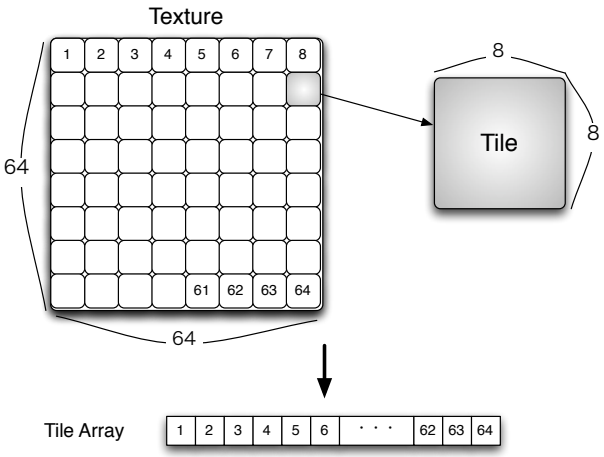

図 6　Texture の分割

Texture を 8 x 8 pixel 単位に分割するという手法を用いる。また、分割した 8 x 8 pixel の Texture を Tile とし、Tile Array に格納する（図 6）

4.2　Tile Array を用いた Rendering

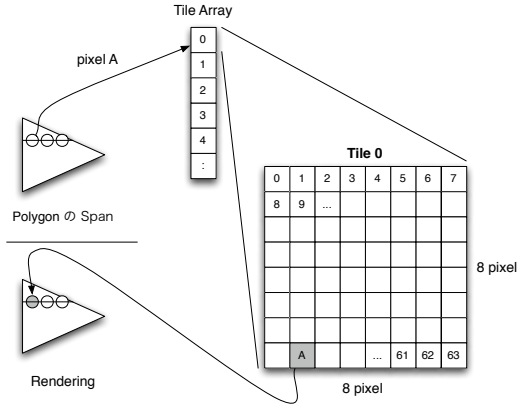

図 7　Span から描画に必要な Tile の計算

（1）　Polygon の Span から、描画に必要な pixel A を含む Tile を計算する。
（2）　pixel A を含む Tile 0 を DMA で取得する。
（3）　Tile 0 の pixel A の 情報 (RGB α 値) を取得して描画を行う。

4.3　Texture の縮小画像の作成（ Scale ）

描画されるオブジェクトが小さい場合、そのオブジェクトに使用される Texture はそのままの大きさである必要はない。そこで、読み込んだ Texture を 2 分の 1 ずつ縮小させた Scale を用意する (最小 8 x 8 pixel)。描画に必要な Scale の Texture は、Span の長さから選択する。

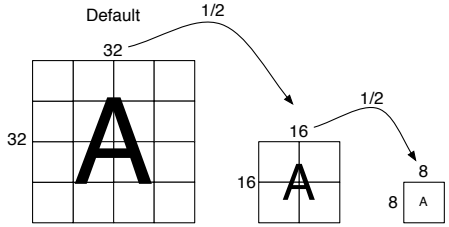

図 8　Texture の縮小画像の作成

以下に Scale の作成と選択のプログラムを示す。

- Scale の作成

```
static uint32*
makeTapestry(int tex_w, int tex_h, uint32 *tex_src,
             int all_pixel_num, int scale_cnt)
{
   uint32 *tex_dest;

   int t = 0;
   int diff = TEXTURE_SPLIT_PIXEL;
   int p_diff = 1;

   tex_dest = (uint32*)manager->allocate(sizeof(int)*all_pixel_num);

   while (scale_cnt) {
     for (int y = 0; y < tex_h; y += diff) {
       for (int x = 0; x < tex_w; x += diff) {
         for (int j = 0; j < diff; j += p_diff) {
           for (int i = 0; i < diff; i += p_diff) {
```

```
          tex_dest[t++] = tex_src[(x+i) + tex_w*(y+j)];
        }
      }
    }
  }

  diff <<= 1;
  p_diff <<= 1;
  scale_cnt >>= 1;
 }

  return tex_dest;
}
```

- 最適 Scale の選択

```
static int
getScale(int width, int height, int tex_width, int tex_height, int scale_max)
{
  int base, tex_base;
  int scale = 1;

 /**
  * width と height で、長い方を基準に、
  * texture の scale を決める
  */
  if (width > height) {
    base = width;
    tex_base = tex_width;
  } else {
    base = height;
    tex_base = tex_height;
  }

  if (tex_base > base) {
    int t_scale = tex_base/base;
    while (t_scale >>= 1) {
      scale <<= 1;
   }
  }

  return (scale > scale_max) ? scale_max : scale;
}
```

Vol.2009-ARC-183 No.38
Vol.2009-OS-111 No.38
2009/4/24

5. 評価と考察

5.1 Scale を用いた描画処理の効果検証

- 検証に用いたサンプル (1 0個のオブジェクト)
  - Polygon 総数 : 19860
  - Texture 総数 : 10
    (8x8(3)、512x384(2)、616x123(4)、1024x768(1)) pixel
- 1つのオブジェクトが親、その他のオブジェクトがその子になり回転しながら移動する。

5.1.1 実 行 結 果

表 1　Scale を用いることによる実行速度の比較

	Scale なし (FPS)	Scale あり (FPS)	速度の向上 (%)
Mac OSX	7.0	8.5	21
PS3Linux(SPE 1)	4.3	5.6	30
PS3Linux(SPE 6)	10.8	13.5	25

∗ FPS とは Frame Per Second の略で、1秒間に何回画面を書き換えたかを表している。

- 実行速度の比較を行った結果、20 〜 30% の速度向上が見られる。
- Mac OSX は SDL[7] 経由で出力、PlayStation 3 は Frame Buffer へ直接出力している。

5.1.2 考　　察

Scale を適用した結果、実行速度の向上が見られた。しかし、SPE を1つ用いたときと、6つ用いたときの速度が 2.6 倍ほどしか向上していないことがわかる。

台数効果が出ていない原因として、DMA 転送の待ち時間の無駄を十分に利用できてことが考えられる (Amdahl 則[8])

Vol.2009-ARC-183 No.38
Vol.2009-OS-111 No.38
2009/4/24

5.2　キャッシュ

SPE 上の Texture データのキャッシュの有効性を検証する。
キャッシュを用いた場合と用いてない場合を SPE の数を変更して比較する。

表 2　キャッシュの有無による処理速度の比較

キャッシュなし (SPE 1 個)	0.08 FPS
キャッシュなし (SPE 6 個)	0.42 FPS
キャッシュあり (SPE 1 個)	0.59 FPS
キャッシュあり (SPE 6 個)	2.54 FPS

6.　Gallium

Gallium3D[?)] とは Tungsten Graphics 社が開発している、オープンソースの 3D グラフィックドライバ である。現在、Gallium には Intel GPU (i915) と Cell のドライバが実装されており、OSMesa[6)] ではこの Cell ドライバを用いている。現在の Cell Driver のバージョンは 0.2 で、Cell SDK (Software Development Kit) の version 2.1 もしくは 3.0 が必要となる。

Gallium Cell Driver では、Triangle を SPE に送り、そこから得られた Span に対してテクスチャを貼りレンダリングする。この時、Tile と呼ばれる、分割した描画領域毎に FrameBuffer へ書き込む。SPE は 6 個使用しており、SPE 上では SIMD 演算を積極的に使用している。

我々がレンダリングエンジンとして OSMesa を採用しなかった理由は先行研究[?)] で述べられているように、OSMesa の実装自体が非常に複雑で、SPE を使用するための拡張が困難であると判断したからである。その OSMesa の Cell 実装として OpenGL のドライバである Gallium が登場したことにより、レンダリングエンジンとして Gallium を採用することを考えた。しかし、Cell 上でのゲーム開発において、レンダリングエンジンのみを SPE 上に実装しても、Amdahl 則を考慮すると並列度の確保が困難であると考えられる。よって、我々はレンダリングエンジンとして Gallium を採用せず、レンダリングだけではなく SceneGraph の動作も SPE 上で行わせる Cerium の開発を行っている。

6.1　実行速度比較

Cerium は OSMesa のレンダリング実装を参考にしているため、Gallium と同様のレンダリングを行っている。ここでは、Cerium と Gallium の実行速度を計測し比較した。例題プログラムとして、図 9 地球のオブジェクトを表示して、SDL から得られる実行時間を元に FPS (Frame Per Second) を出力して比較する。解像度は 1920x1080 とする。Gallium、Cerium それぞれで以下のような実行環境を構築している。

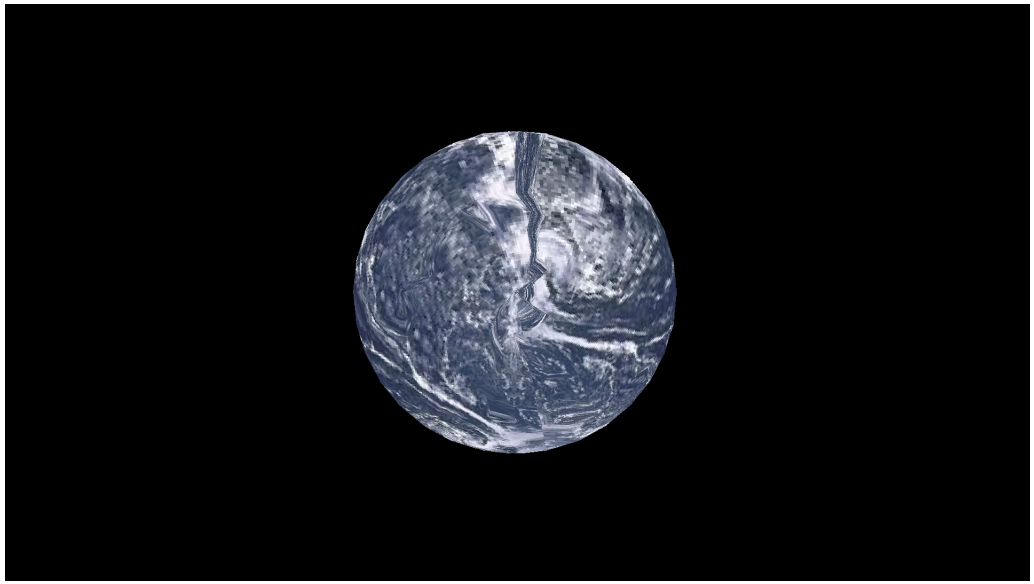

図 9　Gallium、Cerium 例題

6.1.1　実行環境構築 Gallium

Gallium の例題プログラムでは、Cell Driver が組み込まれている OpenGL library (libGL) をロードして実行する。また、xml ファイル を用いて OpenGL の API で表示するルーチンは同研究室の杉山が実装した。以下がそのコードとなる。

```
void
SceneGraph::gl_draw(void)
{
  glEnable(GL_TEXTURE_2D);
  glBindTexture(GL_TEXTURE_2D, (GLuint)texture);
```

Vol.2009-ARC-183 No.38
Vol.2009-OS-111 No.38
2009/4/24

```
  glTranslatef(xyz[0], xyz[1], xyz[2]);

  glTranslatef(c_xyz[0], c_xyz[1], c_xyz[2]);
  glRotatef(angle[0], 1, 0, 0);
  glRotatef(angle[1], 0, 1, 0);
  glRotatef(angle[2], 0, 0, 1);
  glTranslatef(-c_xyz[0], -c_xyz[1], -c_xyz[2]);

  glBegin( GL_TRIANGLES);
  for(n = 0; n < size*3; n += 3) {
      glTexCoord2f(tex_pos[n], 1-data[n+1]);
      glVertex3f(coord[n], coord[n+1], coord[n+2]);
      glNormal3f(normal[n], normal[n+1], normal[n+2]);
  }
  glEnd();
  glDisable(GL_TEXTURE_2D);
}
```

6.1.2　実行環境構築 Cerium

Cerium の描画出力は SDL Window と FrameBuffer の二種類ある。FrameBuffer では、SPE から直接 FrameBuffer へ DMA 転送により書き込むが、SDL Window 経由では、一度 PPE の バッファ (SDL_Surface) へ書き込んでから FrameBuffer へ書き込む。以下がそのコードとなる。通常、SDL を用いる場合はこの様なコードになる。

```
// bitmap : RGB 値を持つ バッファ (SPE から書き込まれる)
// screen : FrameBuffer のアドレスを指す
SDL_BlitSurface(bitmap, NULL, screen, NULL);
SDL_UpdateRect(screen, 0, 0, 0, 0);
```

6.2　実行結果と考察

各環境の実行結果を **表??** に示す。コンパイラの最適化レベルは -O9 を用いている。

結果から、SPE の数が同じ場合は Cerium の方が Gallium よりも実行速度が速いと確認できた。また、直接 FrameBuffer に書き込む方が速いことも確認できる。

表 3　Gallium、Cerium 実行速度比較結果

Gallium (SPE 6 個)	**5.4 FPS**
Cerium (SDL 出力、SPE 1 個)	2.2 FPS
Cerium (FrameBuffer 出力、SPE 1 個)	2.5 FPS
Cerium (SDL 出力、SPE 6 個)	6.7 FPS
Cerium (FrameBuffer 出力、SPE 6 個)	**9.5 FPS**

Cerium では光源やシェーディング、アルファブレンディングなどの処理は行っておらず、現在は最低限のレンダリング機能しか実装していないため速度に差が出たのではないかと考えられる。しかし、Gallium と違い Cerium は SIMD 演算をまだ組み込んでいないため、さらに速度向上が期待できる。

7. ま と め

Scale による DMA 待ち時間の有効活用と、SPE 上の Texture データのキャッシュの有効性を証明できた。

DMA 転送の待ち時間の無駄を十分に利用できていない可能性が出たので、さらに SPE 上で行う処理を増やし、待ち時間の有効活用を目指す。

以下に実装予定の処理を示す。

- SceneGraph から Polygon の生成
  現在 PPE を用いて行っている作業を SPE 上へ移行
- SPE 上のコードの入れ替え
  SPE を用いて行う処理の入れ替え

FPS を検証の指標として利用すると、描画処理などの他の処理の誤差に隠れてしまう可能性がある。DMA 転送の待ち時間を利用した SPE のデータ管理における実行速度の検証を、FPS だけでなく特定の命令数、命令の実行時間などの別の指標を用いて検証を行うことが必要である。

Vol.2009-ARC-183 No.38
Vol.2009-OS-111 No.38
2009/4/24

参 考 文 献

1) Sony Corporation: Cell broadband engine architecture (2005).
2) : SourceForge.JP: Project Info - Cerium Rendering Engine, https://sourceforge.jp/projects/cerium/.
3) 宮國渡 : Cell 用の Fine-Grain Task Manager の実装: 琉球大学理工学研究科情報工学専攻 平成 20 年度学位論文 (2009).
4) 神里晃 : Cell を用いたゲームフレームワークの実装: 琉球大学理工学研究科情報工学専攻 平成 19 年度学位論文 (2009).
5) : Galium3D , http://www.tungstengraphics.com/wiki/index.php/Gallium3D
6) : The Mesa 3D Graphics Library , http://www.mesa3d.org/
7) : Simple DirectMedia Layer, http://www.libsdl.org/.
8) Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, and Doug Lea: *Java Concurrency in Prac-tice*, Addison-Wesley Professional (2005).