3次元コンピュータグラフィックスの基礎 -11

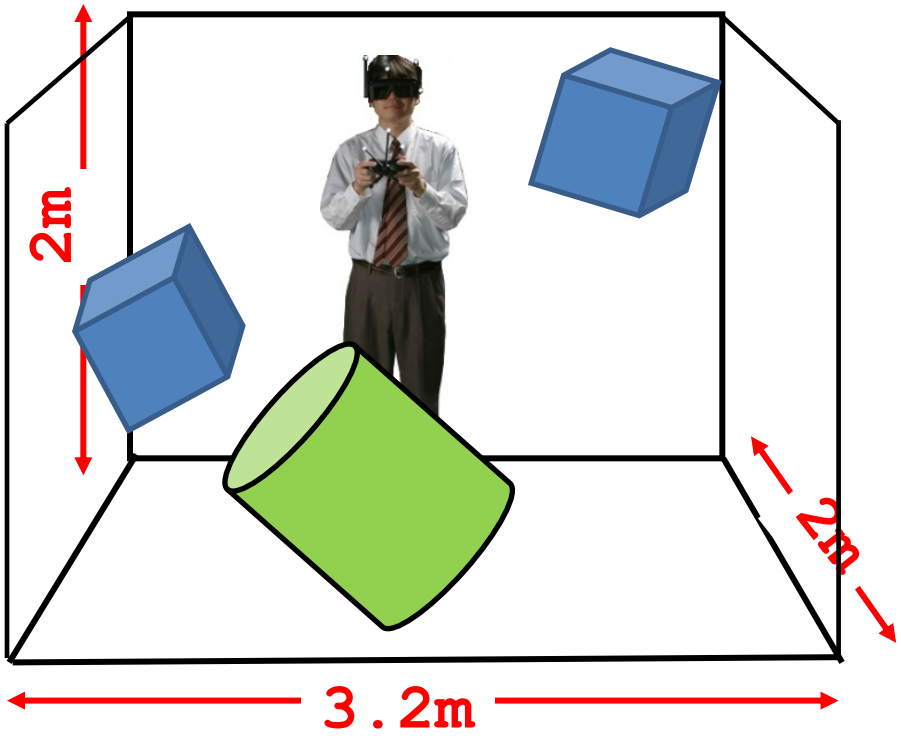
大野

CAVEのハードウェアシステム

- 県立大のCAVEのハードウェアシステム
  - 背面投影型スクリーン(正面スクリーン, 側面スクリーンx2)
  - 床面スクリーン
  - 液晶ステレオプロジェクター x4
  - 日本SGI社製ワークステーション(Xeon E5-2640v3 2.6GHz x2, メモリ:64GB, グラフィックス:Quadro K5200x4)
  - 液晶シャッター眼鏡(立体メガネ)
  - コントローラー(一般のシステムではワンド)
  - トラッキングシステム

3Dディスプレイ

(1) 前方の壁
(2) 床面
(3) 右面
(4) 左面



グラフィックスワークステーション
日本SGI製
CPU: Xeon E5-2640v3 2.6GHz (8コア) x 2
メモリ:64GB
グラフィックス:Quadro K5200 x 4

プロジェクター x4
バルコ製
解像度 1920x1200
10,000アンシルーメン

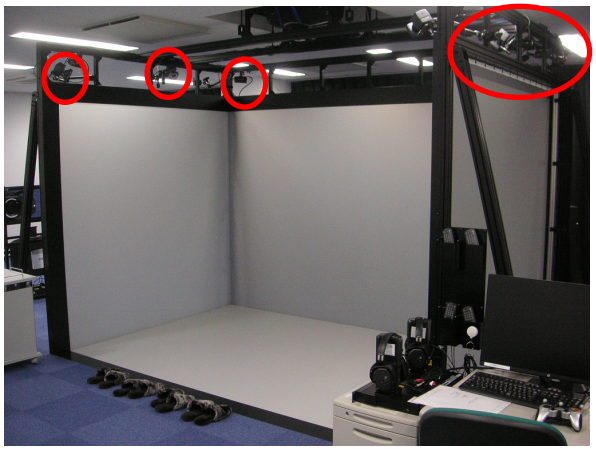
無線ヘッドフォンによる3次元音響

3Dディスプレイ

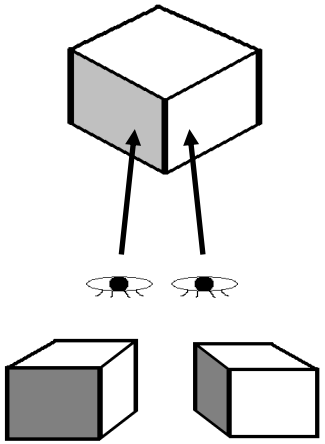
立体メガネ

光学カメラ（トラッキングシステム）

コントローラー

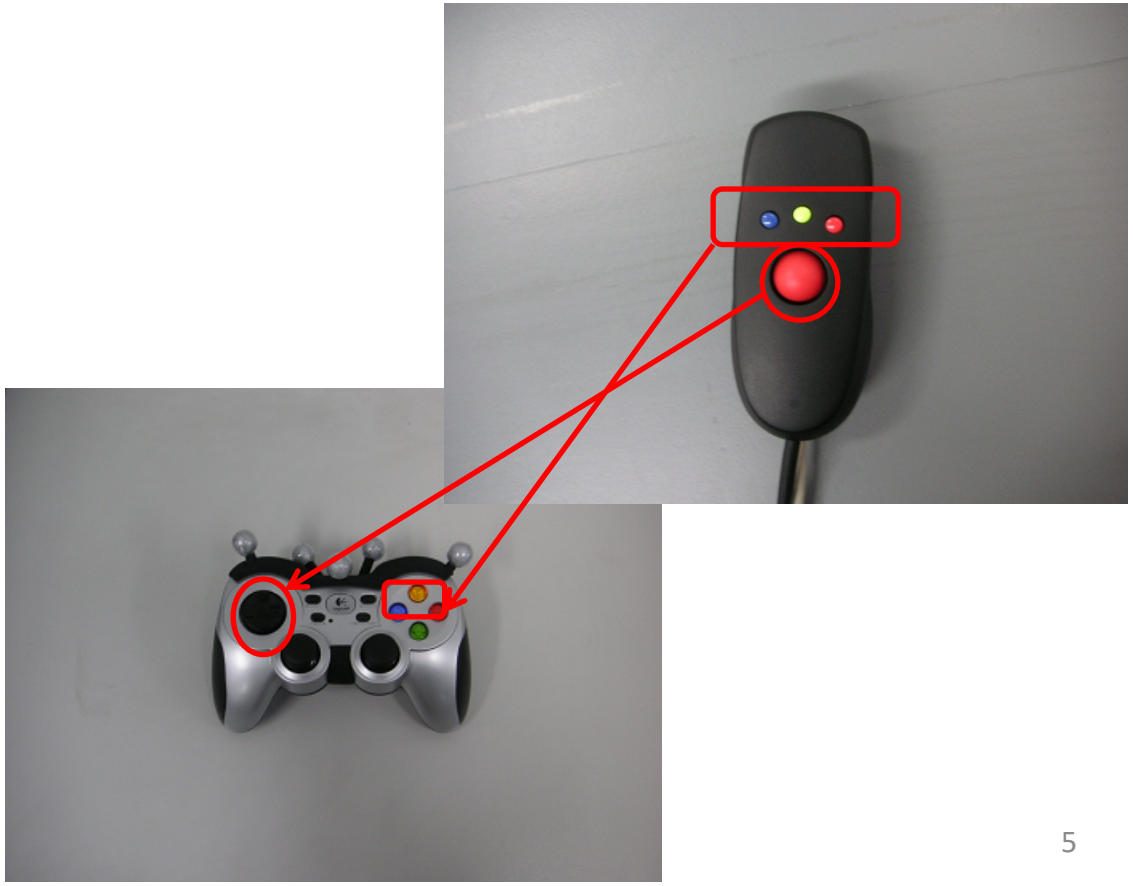


CAVEのハードウェアシステム



左目が見る映像　右目が見る映像

ワンドとコントローラーの対応
(ボタン × 3, ジョイスティック)



CAVEプログラミング

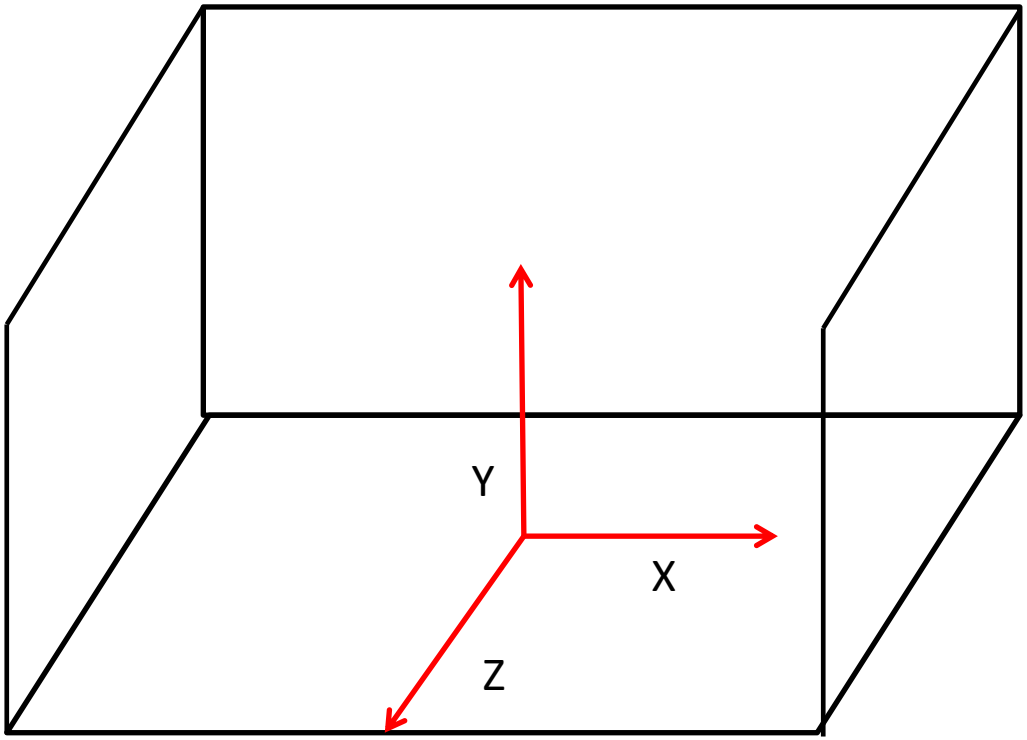
- CAVELibを使用することで、
  - 射影計算
  - 立体映像のための視差
  - スクリーン間の映像同期
  - トラッキング

等を考えなくとも、自動的に行ってくれる

CAVEプログラミング

- CAVELibを使用することで、
  - OpenGLで映像を作れる(DirectXには非対応)
  - トラッキングのデータも簡単に使える
    - 立体メガネ、コントローラーの位置、向いている方向
    - ボタンの状態（押された、離された、押されている）
    - ジョイスティックの状態（どの向きに倒されているか？）
  - 描画、計算は並列実行してくれる
- CAVELibのプログラムは、GLUTのプログラムと、非常に似ている

CAVEプログラミング(座標)



CAVEシミュレータ

- 可視化装置室に行かなくとも、リモートログインでプログラミングをすることが可能である。
- CAVEシミュレータという、CAVEに表示する画面を、2次元のWindowに表示する便利な機能がある
- .caverc というファイルをプログラムファイルと同じディレクトリに置くことで、いろいろな表示モードに変えることができる

```
Simulator␣y⏎
DisplayMode␣mono⏎
```

CAVEシミュレータを使い、表示はモノ（立体視不可）

```
Simulator␣n⏎
DisplayMode␣stereo⏎
```

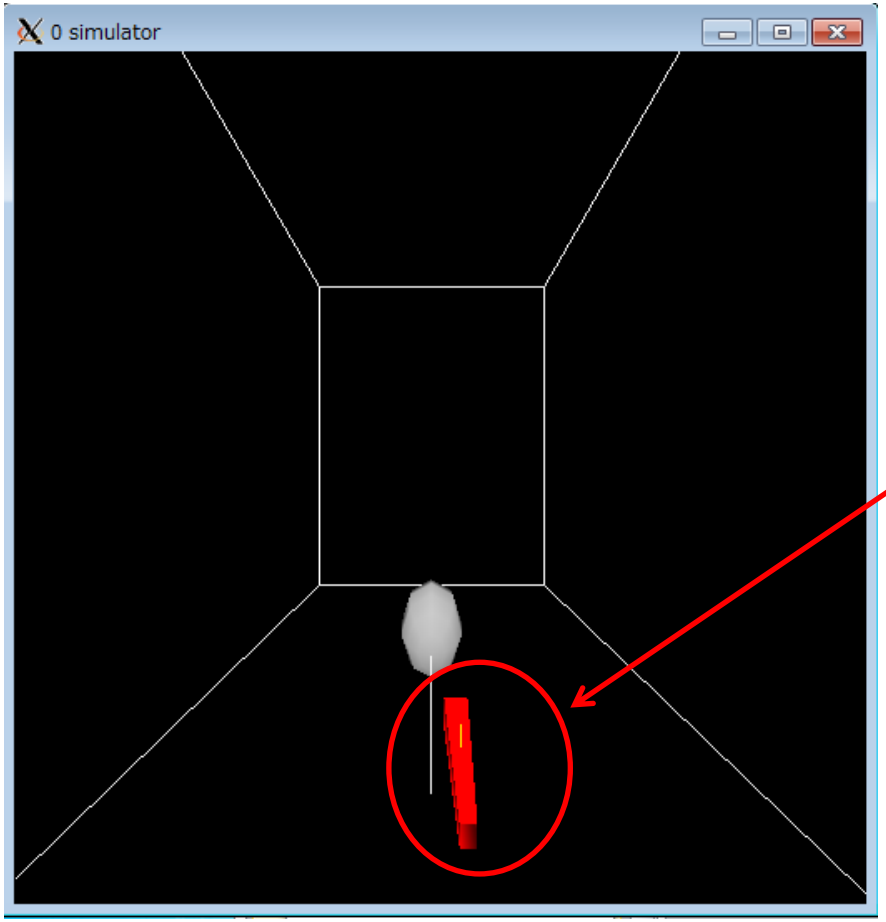
CAVE本体を使い、表示はステレオ（立体視可）

```
DisplayMode␣mono⏎
```

CAVE本体を使い、表示はモノ（立体視不可）
(デフォルトで本体を使う設定なので)

ドットファイル(ファイル名が . で始まる)は`ls`では表示されない。-aオプションが必要

CAVEシミュレータ



このような窓が開く

スクリーンは6面

これはワンド（コントローラー）を表す

CAVEシミュレータ

視点の動きのコントロール方法	
←	move left
→	move right
↑	move forward
↓	move backward
Shift + ↑	move up
Shift + ↓	move down
Alt + ←	rotate left
Alt + →	rotate right
Alt + ↑	rotate up
Alt + ↓	rotate down
p	Reset

CAVEシミュレータ

ワンドの操作	
Ctrl + mouse movement	move wand left/right/forward/back
Shift + mouse movement	move wand left/right/up/down
Alt + mouse movement	rotate wand left/right/up/down
mouse left button	wand left button
mouse middle button	wand middle button
mouse right button	wand right button
Space + mouse movement	Joystick shifting

CAVELibとglutプログラムの比較

ball.cとball_glut.cを比較してみる。非常に似ていることがわかる

```
int main(int argc,char **argv) {
 height = 0.0;
 angle = 0.0;
 glutInit(&argc, argv);
 CAVEConfigure(&argc,argv,NULL);
 CAVEInitApplication(init_gl,0);
 CAVEDisplay(draw,0);
 CAVEInit();

  while(!CAVEgetbutton(CAVE_ESCKEY)){
    compute();
    CAVEUSleep(10000);
  }
 CAVEExit();
 return 0;
}
```

CAVELib

```
int main(int argc, char** argv) {
  height = 0.0;
  angle = 0.0;
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE |
GLUT_RGB | GLUT_DEPTH);
  glutInitWindowPosition(0, 0);
  glutInitWindowSize(500, 500);
  glutCreateWindow("ball_glut.cpp");
  init_gl();
  glutDisplayFunc(draw);
  glutIdleFunc(compute);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

GLUT

CAVELibとglutプログラムの比較

- glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
- glutInitWindowPosition(0, 0);
- glutInitWindowSize(500, 500);
- glutCreateWindow("ball_glut.cpp");
- glutReshapeFunc(reshape);

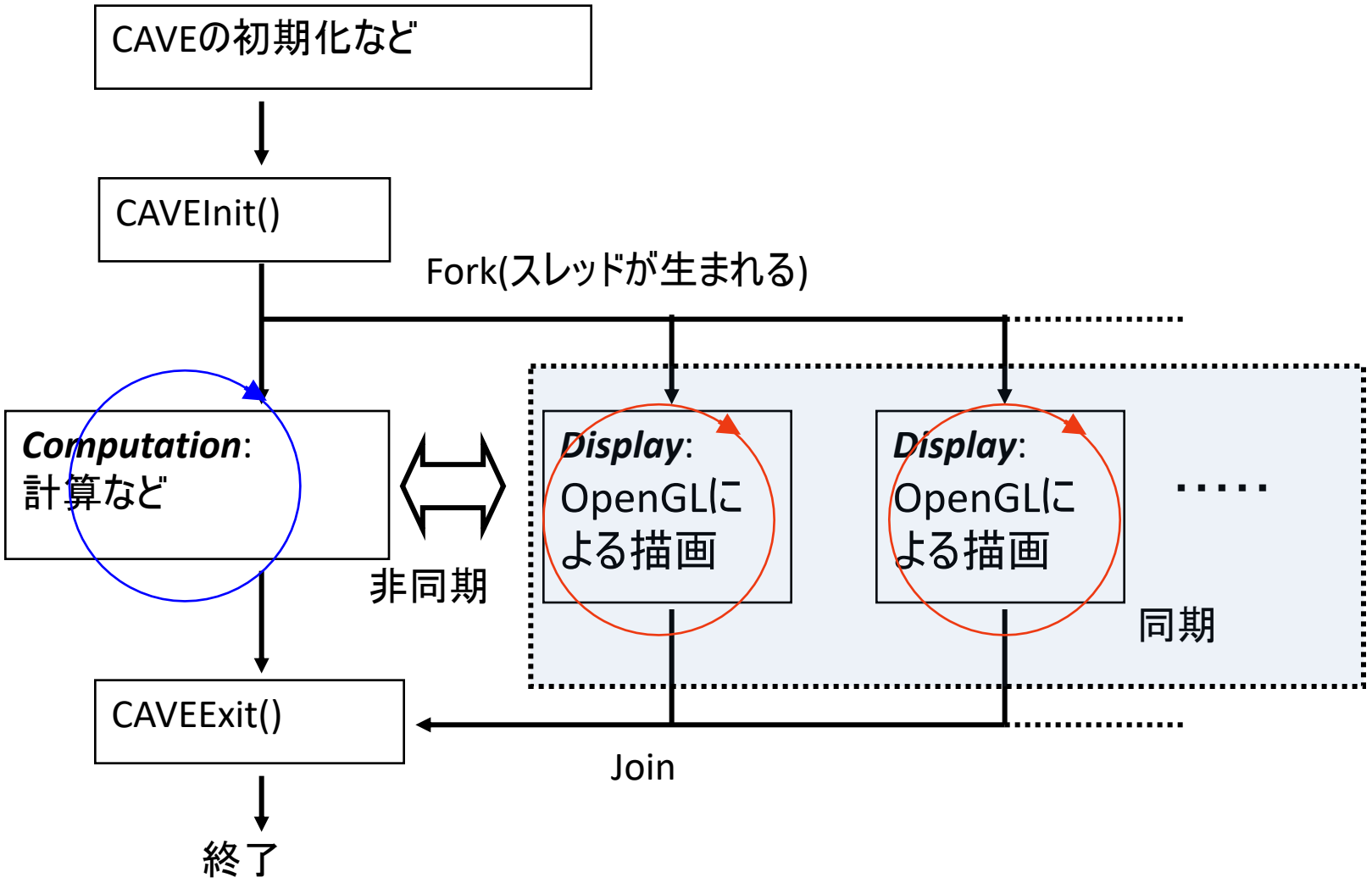
ウインドウ関係の関数は必要なし。

デフォルトでダブルバッファモード。

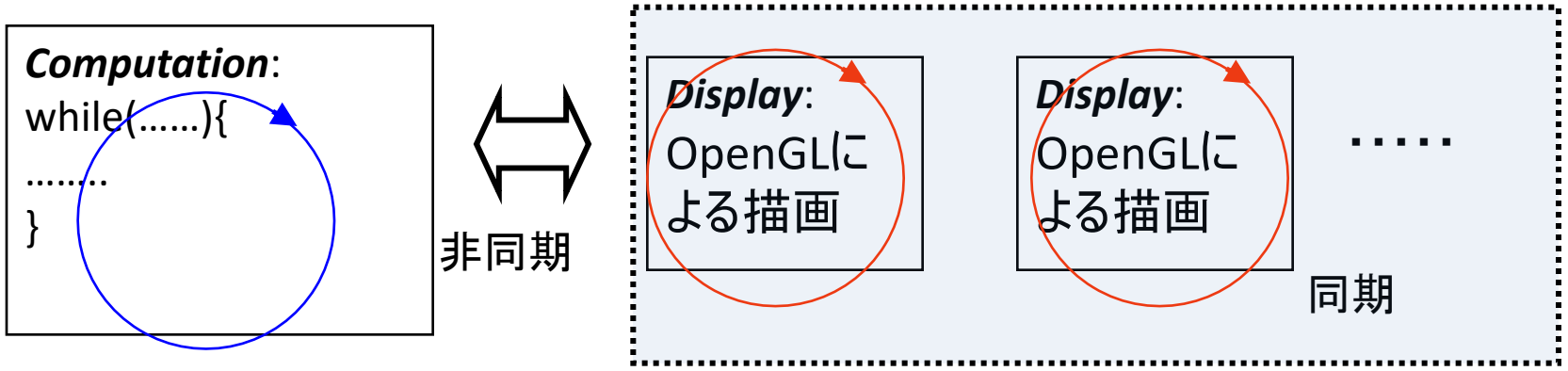
CAVELibとglutプログラムの比較

- glEnable(GL_DEPTH_TEST);
  - デフォルトでenableになっている
- glutSwapBuffers();
- glutPostRedisplay();
  - 自動的に、常にダブルバッファモードで画像を描くので、必要ない
- gluLookAt, glViewPort, glMatrixMode, glLoadIdentity, gluPerspective
  - 視点関係、射影関係はCAVElibが自動的に面倒を見てくれる

CAVELibプログラムの大まかな流れ



CAVELibプログラムの性質



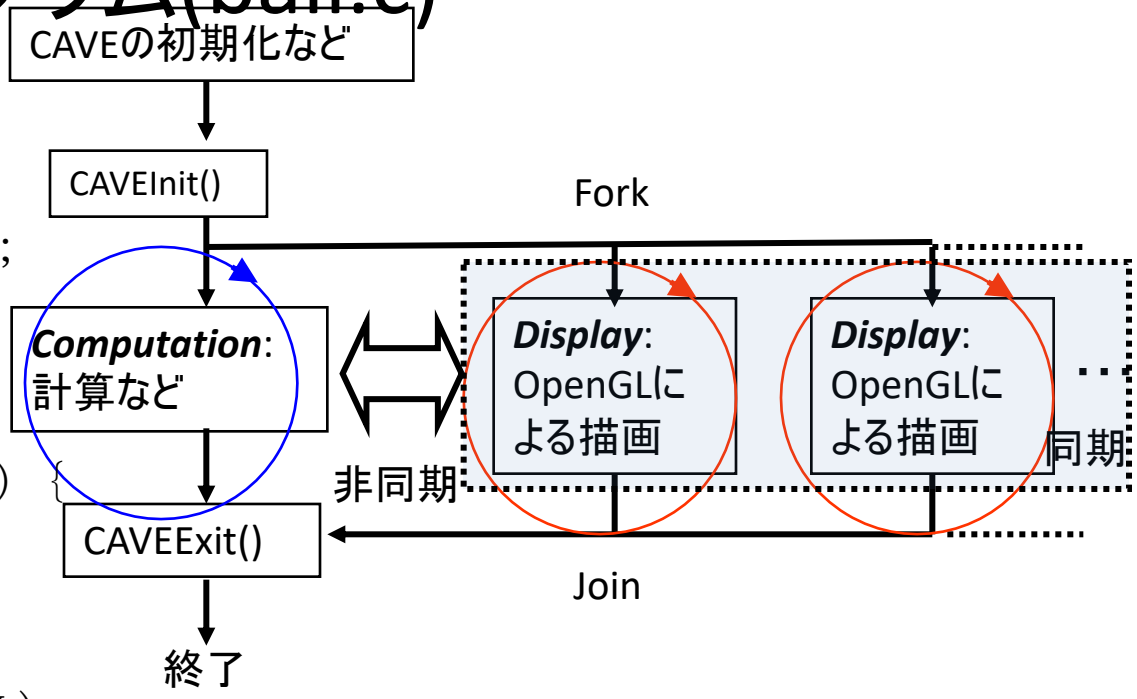
- 常に立体メガネ、ワンドの位置や角度の情報を取得している
- 立体メガネの情報を元に、表示画像を常に描きかえる (Display)
- Displayスレッド同士は、同期をとっている。
- DisplayスレッドとComputationスレッドは同期を取っていない
  - ➢ Computationスレッドのループは、ユーザーがwhileなどを使って自分で作る。
  - ➢(普通は、)Computationスレッドのループのほうが速くまわる

サンプルプログラム(ball.c)

～略

```
void compute(){
 angle += 0.1;
 if(angle > 6.28) angle = 0.0;
 height = 5.0*cos(angle);
}

int main(int argc,char **argv) {
 height = 0.0;
 angle = 0.0;
 glutInit(&argc, argv);
 CAVEConfigure(&argc,argv,NULL);
 CAVEInitApplication(init_gl,0);
 CAVEDisplay(draw,0);
 CAVEInit();
  while (!CAVEgetbutton(CAVE_ESCKEY)) {
    compute();
    CAVEUSleep(10000);
  }
 CAVEExit();
 return 0;
}
```



OpenGLの初期用関数の指定
一度だけ実行される

描画用関数の指定

サンプルプログラム(ball.c)

```
～略
 glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,spec);

  glClearColor(0.0,0.0,0.0,0.0);

}

void draw()
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
 glEnable(GL_LIGHTING);
  glPushMatrix();
   glTranslatef(0,height,-3);
   glutSolidSphere(1.0, 18, 18);
  glPopMatrix();
 glDisable(GL_LIGHTING);
}
```

いろいろな変数

- 大域変数（グローバル変数）
  - すべてのスレッドで共有される
  - 複数のスレッドで値を書き換える場合は注意

  (CAVELockなどで安全性確保：：ちょっと高度)

  例: Aをグローバル変数として、

  複数のスレッドで、A=A+1を実行する場合など
- 局所変数（関数内で宣言された変数。ローカル変数）
  - 同じ関数を実行したとしても、スレッドごとに別々の値
- 静的変数（関数内で宣言時にstaticをつけたもの）
  - スレッド間で共有されるので注意

CAVEプログラム演習

- Mayaへログインして、ball.cをコンパイルしてみよう
  - ssh␣–Y␣IPアドレス⏎
- Windowsを使っている場合は、Xmingを起動後に、PuttyでX-forwardを有効にしてログインする
- export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/CAVELIB/3.3/lib64を実行
- ball.cのコンパイルには、makeコマンドを使う(用意したMakefileもコピーすること)
  - make⏎
- chmod␣764␣ball.sh⏎でball.shに実行権限をあたえ、./ball.shでballを実行する

1. ball.cをCAVEシミュレータで実行し、いろいろ操作してみよう
2. 描画のところ（OpenGLのところ）を書き換えてみよう
3. ball.cとball_glut.cの中身を比較してみよう(宿題)
4. make, makefileについて調べよう(宿題)