

情報メディア実験A

# 物理エンジンを使った アプリケーション開発

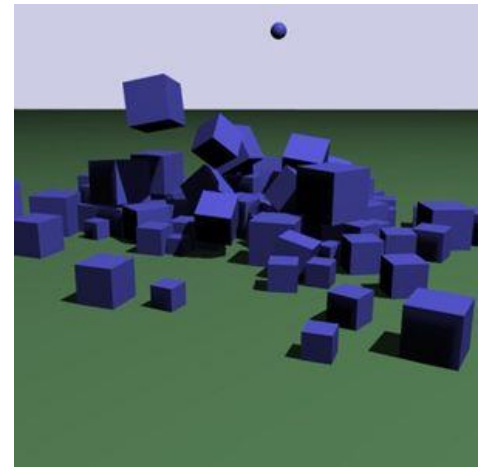
筑波大学情報学群  
情報メディア創成学類  
藤澤誠

# 実験スケジュール

- 毎週 水3,4限 & 金5,6限
- テーマ内スケジュール
  1. ガイダンス & 事前知識(C++)説明 : 4/16
  2. 物理シミュレーションとは? : 4/21, 23, 28
  3. 物理エンジンとは? : 4/30, 5/7, 12
  4. 剛体間の衝突判定, 衝突応答 : 5/14, 21, 26, 28
  5. 剛体間リンク : 6/2, 4, 9, 11
  6. 3Dモデル読み込みと弾性体 : 6/16, 18, 23, 25
  7. アプリケーション開発 : 6/30, 7/2, 7, 9, 14, 16, 21
  8. 成果発表会 : 7/28

赤太字は説明回, 7/22は発表準備 & レポート作成回  
5/7(金)は水曜授業日, 5/19(水)は春A期末試験日

# 物理エンジンとは？



- 物理シミュレーションを簡単に  
使うためのライブラリ
  - 主に剛体を扱い，弾性体や流体を扱える  
エンジンもある
  - 物理エンジンの主機能
    - 剛体の運動(Dynamic and Kinematics Rigid  
Body)の計算
    - 物体同士の衝突検出(Collision Detection)&  
衝突応答(Collision Response)

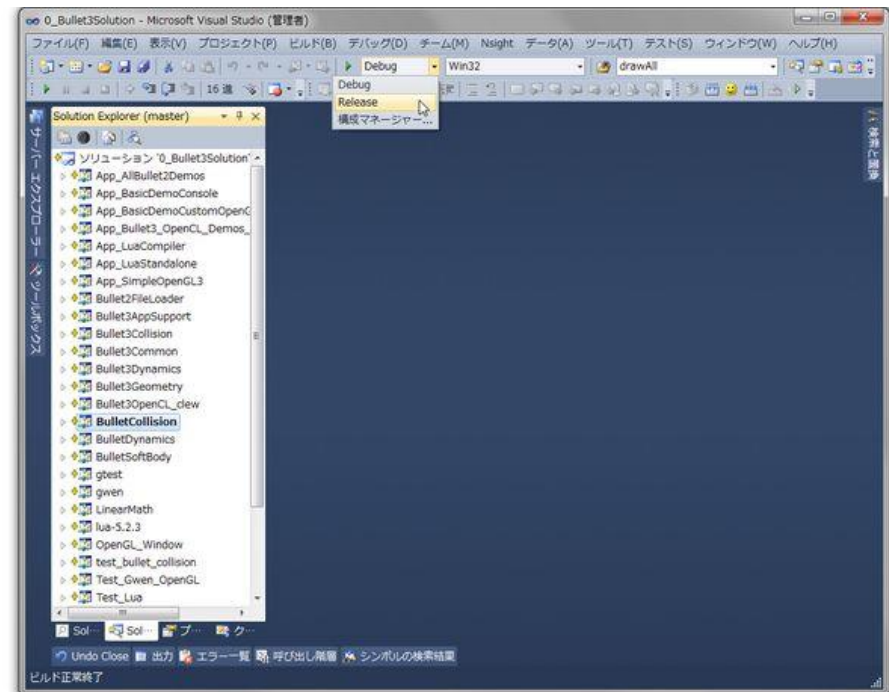
# 実験で使う物理エンジンについて

- Bullet(Bullet Collision Detection and Physics library)
  - 衝突判定, 剛体運動ライブラリ
  - オープンソース(Zlibライセンス,商用利用も可)
  - マルチプラットフォーム(Win, Linux, Mac OS, iOS, Android, PS3, Xbox360, Wii)
  - C++版 + Python版(pybullet)
  - Maya, Blender, Houdiniなどでプラグインとして提供されている

# Bulletのビルド

## ■ Bulletのビルド

ソースコードをBullet PhysicsのWebページ  
(<http://bulletphysics.org/wordpress/>)  
からダウンロード・解凍してVisual Studioなどで  
ビルドする。

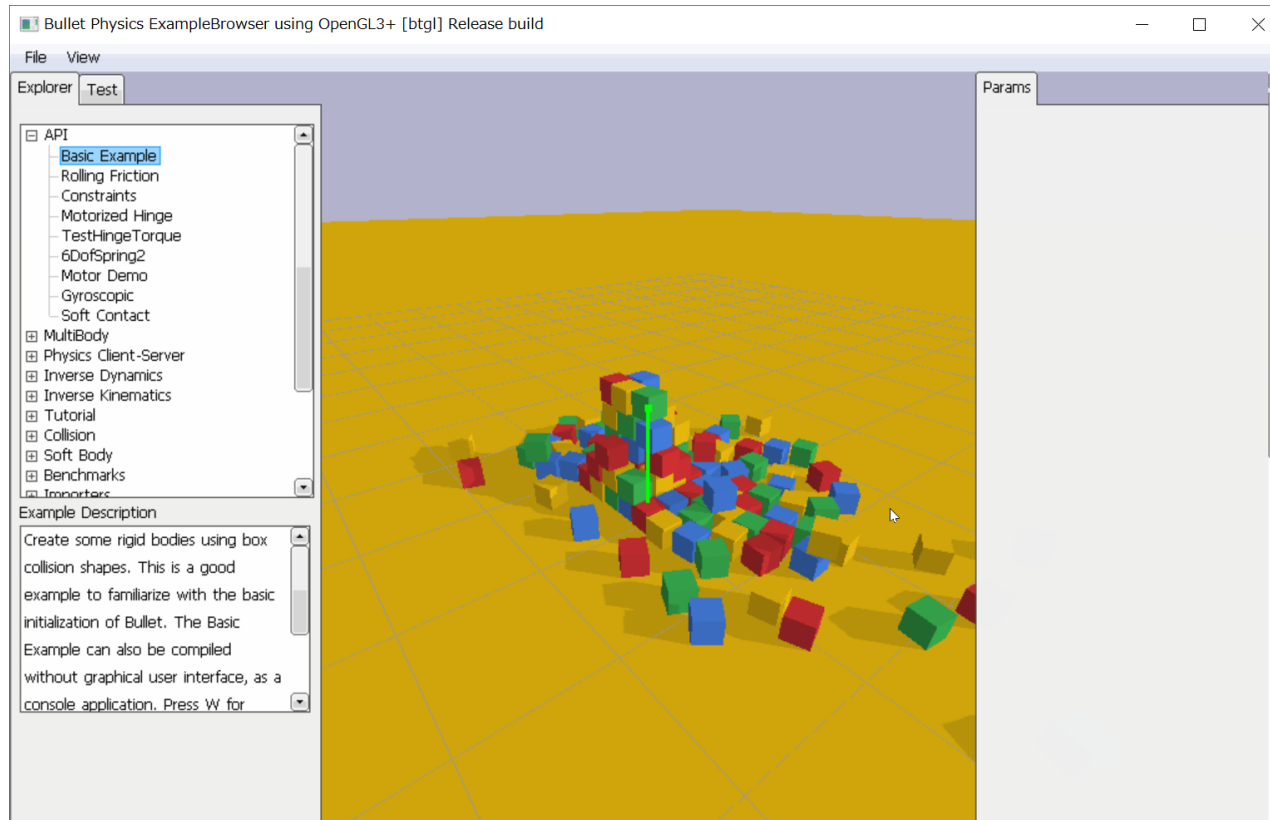


# Bulletのビルド

- ビルドの注意点
  - Visual Studioではプロジェクトをソリューションファイル(\*.sln)を使って管理
  - slnファイル作成のためにpremakeというソフトがBulletには同梱されているが, 最新版v3.09に同梱のpremake5だと上手くいかない可能性があるので, 本実験ではv2.87を使用

# デモプログラムの実行

## ■ BulletExampleBrowser



BULLET/bin/App\_BulletExampleBrowser.exe

# 物理エンジンでの物体の扱い

## **btDynamicsWorld**

### **btRigidBody**

- 質量(0ならstatic),反発率,摩擦係数等のパラメータ
- *btMotionState* : 動きの状態
  - *btTransform*
    - 姿勢(*btQuaternion*/*btMatrix3x3*)
    - 位置(*btVector3*)
- *btCollisionShape* : 物体の形

### **btRigidBody**

⋮



# ワールドの定義

## ■ btDynamicWorldの定義

グローバル変数としてbtDynamicWorld型ポインタ変数を定義

```
btDynamicsWorld* g_pDynamicsWorld;          //!< Bulletワールド
```

## 初期化 (適当な関数内で)

```
// ディスパッチャ, ブロードフェーズ法, 拘束ソルバーの設定(詳しくは今後の講義回で)
...

// Bulletのワールド作成
g_pDynamicsWorld = new btDiscreteDynamicsWorld(dispatcher, broadphase,
                                                solver, config);
g_pDynamicsWorld->setGravity(btVector3(0, -9.8, 0));
```

重力設定, btVector3はBulletで使える3次元ベクトル型

# 剛体オブジェクトの定義

## ■ btRigidBodyの定義

形(btCollisionShape), 位置・姿勢(btMotionState), 質量・慣性モーメントが定義に必要. 定義後にメンバ関数でパラメータ変更可能

```
// 球体形状を設定
btCollisionShape *sphere_shape = new btSphereShape(g_fRad);

// 球体の初期位置・姿勢
btQuaternion qrot(0, 0, 0, 1);
btDefaultMotionState* motion_state = new btDefaultMotionState(btTransform(qrot, pos));

// 慣性モーメントの計算
btVector3 inertia(0, 0, 0);
sphere_shape->calculateLocalInertia(mass, inertia);

// 剛体オブジェクト生成(質量, 位置姿勢, 形状, 慣性モーメントを設定)
btRigidBody *body = new btRigidBody(mass, motion_state, sphere_shape, inertia);

// ワールドに剛体オブジェクトを追加
g_pDynamicsWorld->addRigidBody(body);
```

# 剛体パラメータ設定

- btRigidBodyのパラメータ設定例

- 反発力：反発係数で設定

```
body1->setRestitution(0.9);  
body2->setRestitution(0.5);
```

} body1とbody2が当たった場合の反発係数  
は両者を掛けた値( $0.9 \times 0.5 = 0.45$ )となる

- 摩擦力：摩擦係数で設定

```
body1->setFriction(0.8);  
body2->setFriction(0.8);
```

} body1とbody2が接触している場合の摩擦係数  
は両者を掛けた値( $0.8 \times 0.8 = 0.64$ )となる

# 剛体オブジェクトの定義

## ■ btRigidBodyの種類

- 動力学(動作)剛体(Dynamics (moving) rigidbodies)
  - 質量が正の値
  - 毎シミュレーションステップで動力学に基づきその位置と姿勢が更新される
- 静的剛体(Static rigidbodies)
  - 質量が0
  - 衝突が起こっても動かない(動力学剛体と衝突はする)
  - 地面や壁など
- 運動学剛体(Kinematic rigidbodies)
  - 質量が0
  - ユーザ操作に従って動くという点以外は静的剛体と同じ
  - 動く障害物など

# ワールドの時間を進める

- ある時間ステップ幅でシミュレーションを進める  
GLUTのTimer関数 or Idle関数内で

```
// bulletのステップを進める  
if(g_pDynamicsWorld){  
    g_pDynamicsWorld->stepSimulation(DT, 1);  
}
```

時間ステップ幅とサブステップ数

# ワールドの破棄

- btDynamicWorldの破棄  
newしたらdeleteと同じく, ワールド定義もメモリ確保なので破棄して, メモリ解放する必要がある

```
// ワールド破棄  
delete g_pDynamicsWorld->getBroadphase();  
delete g_pDynamicsWorld;
```

# Bulletのビルドとサンプル実行

ここからは各自の環境で  
実際に作業

実験ページの「2. 物理エンジンとは？」の  
練習問題を実際にやってみよう  
(余裕のある人はoption課題もやってみよう!)