# 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** 

### 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
(https://pybullet.org/wordpress/)
からダウンロード・解凍してVisual Studioなどで
ビルドする。

⑥ → ③ 🎁 → 😝 🖺 🚇 🥠 → 🤍 → Debug 🕶 x64 ▶ ローカル Windows デバッガー ▼ ▷ □ □ □ □ Live Share № リューション エクスプローラ □ ソリューション '0\_Bullet3Solution' (82/82 プロジェ ▶ ♠ App\_BasicExample ▶ ♣ App\_BasicExampleGui ▶ ■ App\_BasicExampleGuiWithSoftwareRenderer ▶ ♣ App\_BasicExampleTinyRenderer App\_BasicExampleVR ▶ ♣ App\_BulletExampleBrowser ▶ ♣ App\_HelloWorld ▶ ♣ App\_InverseDynamicsExample ▶ App InverseDynamicsExampleGui ▶ ♠ App\_InverseDynamicsExampleGuiWithSoftware ▶ Mapp InverseDvnamicsExampleTinvRenderer ▶ App\_PhysicsServer\_SharedMemory ▶ App\_PhysicsServer\_SharedMemory\_GUI ▶ Mapp PhysicsServer SharedMemory VR ▶ App\_PhysicsServerSharedMemoryBridgeTCP ▶ ♣ App\_PhysicsServerSharedMemoryBridgeUDP プロジェクト 'LinearMath' をアップグレードしています. ▶ ♠ App\_PhysicsServerTCP 構成 'Debug|Win32': プラットフォーム ツールセットを 'V143' に変更しています (以前は 'V100')。 ▶ ♣ App\_PhysicsServerUDP 構成 'Debug|x64': プラットフォーム ツールセットを 'v143' に変更しています (以前は 'v100')。 ▶ ♣ App\_RigidBodyFromObjExample 構成 'Release|Win32': プラットフォーム ツールセットを 'v143' に変更しています (以前は 'v100')。 ▶ ♠ App\_RigidBodyFromObjExampleGui 構成 'Release x64': プラットフォーム ツールセットを 'v143' に変更しています (以前は 'v100')。 再ターゲットの終了: 完了 82、失敗 θ、スキップ θ App. RigidBodyFromOhiExampleGuiWithSoftwa

ソリューション エクスプローラー Git 変更
↑ ソース管理に追加 ◆ ◆ リポジトリの選択 ◆

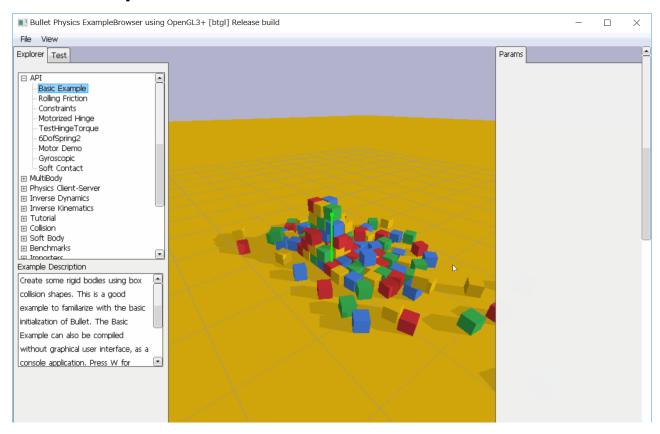
### Bulletのビルド

#### ■ 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
- sIn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。実際にはそれを実行するためのbatファイルが各種用意されているのでそれを実行する.
- プロジェクトの設定がWin32になっていたらx64に変えること!
- 環境によってはいくつかのプロジェクトのビルドに失敗するかもしれないが、BulletCollision、BulletDynamics、BulletSoftBody、LinearMathの4つが成功していれば実験上は問題ない

## 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_dynamicsworld; //!< 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_dynamicsworld = new btDiscreteDynamicsWorld(dispatcher, broadphase, solver, config);
g_dynamicsworld->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 ballrad);
// 球体の初期位置・姿勢
btQuaternion qrot(0, 0, 0, 1);
btDefaultMotionState* motion state = new btDefaultMotionState(btTransform(grot, 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 dynamicsworld->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); body1->setRestitution(0.5); body2->setRestitution(0.5); は両者を掛けた値(0.9×0.5=0.45)となる
```

摩擦力:摩擦係数で設定

```
body1->setFriction(0.8); body1-body2が接触している場合の摩擦係数body2->setFriction(0.8); は両者を掛けた値(0.8×0.8=0.64)となる
```

10

### 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 Timer関数内で

```
// bulletのステップを進める
if(g_dynamicsworld){
    g_dynamicsworld->stepSimulation(g_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_dynamicsworld->getBroadphase();
delete g_dynamicsworld;
```

### 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!)

14