

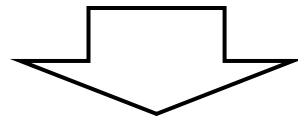
情報メディア実験A

# 物理エンジンを使った アプリケーション開発

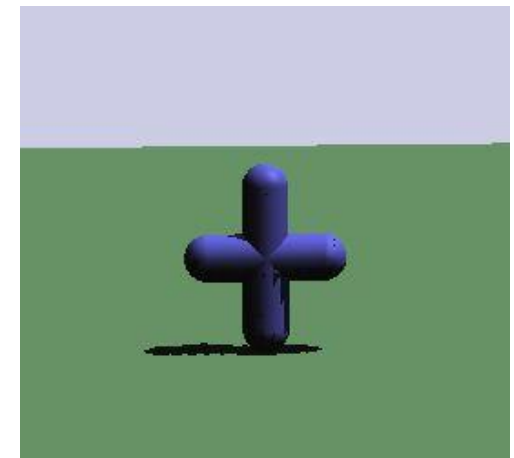
筑波大学情報学群  
情報メディア創成学類  
藤澤誠

# 複合形状

- 複数の単純な形状をつなげることで複雑な形状ができないのか？
  - btCompoundShape  
複数の形状(プリミティブ)を組み合わせて、新しい形状を作る



プリミティブ間の位置関係は固定なので、動きのあるオブジェクトは作れない！



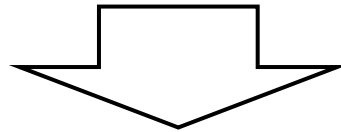
カプセル形状2つの組み合わせでできる形状

# Constraint

- プリミティブ間で動きのある複合形状はできないのか？

例) ヒンジで壁とつながったドア, 関節でつながった腕, 車軸で車体とつながったタイヤ, etc.

⇒ 剛体間の動きに一定の制約をかければよい



拘束条件  
(Constraint)

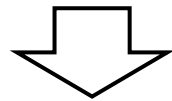
# Constraint-based method

- 拘束条件に合った動きを求めるために, 拘束条件を数式化

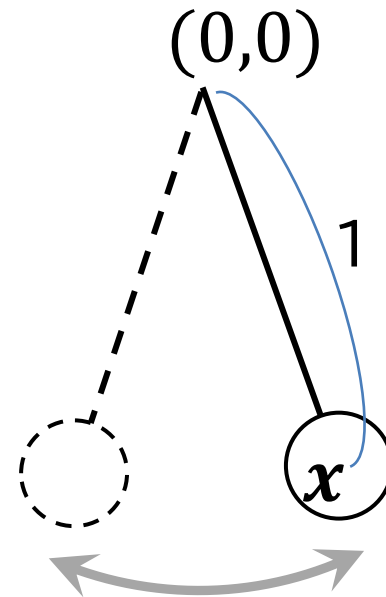
$$\text{拘束条件} : C(\boldsymbol{x}) = 0$$

- 例) 振り子(原点中心, 長さ1)

$$C(\boldsymbol{x}) = |\boldsymbol{x}| - 1 = 0$$



$$C(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{x} - 1 = 0$$



# Constraint-based method

- 拘束条件を満たすためにかけるべき力  $f_c$

$$\underline{f_c} = \underline{J^T} \underline{\lambda}$$

拘束条件  $c$  の空間微分 方向に  $\lambda$  倍 (スカラー倍)

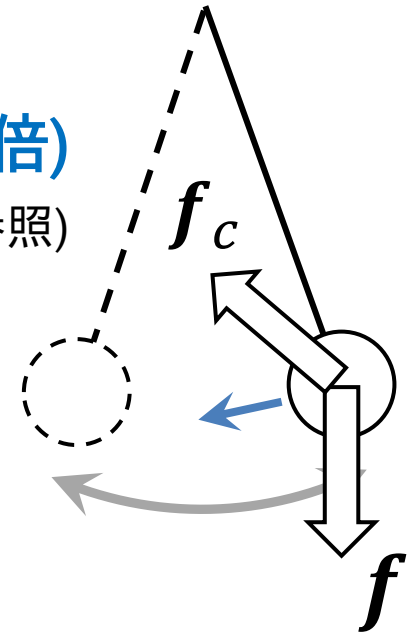
(導出は実験Webページ参照)

- 単位時間あたりの仕事量(力  $f$  と速度  $v$  をかけたもの)を考えて式変形

$$\underline{J} \underline{v} = 0$$

拘束方程式

(この式を満たす  $f_c$  を求めて使うのが Constraint-based method)



$J$  は横ベクトル,  $f$  や  $v$  は縦ベクトルということに注意しよう

# Soft Constraint

- 拘束方程式を満たす力  $f_c$  は必ず求まるのか？

答え) 多くのオブジェクトがつながっていると  
解が求まらないことがある(or 計算時間がかかる)

- ペナルティ法と同じく多少のずれは許してしまっても  
どうか？

⇒ Soft Constraint

$$Jv + \underbrace{\frac{\beta}{\Delta t} C(x)}_{\text{ERP}} + \underbrace{\gamma\lambda}_{\text{CFM}} = 0$$

# Soft Constraint

$$Jv + \underbrace{\frac{\beta}{\Delta t} C(x)}_{\text{ERP}} + \underbrace{\gamma\lambda}_{\text{CFM}} = 0$$

- CFM(Constraint Force Mixing)  
柔らかい拘束を実現するための項  
拘束を満たすための力の大きさ $\lambda$ に係数をかける  
ことでずれを許容する
- ERP(Error Reduction Parameter)  
CFMでできたずれを修正する項

# Constraintの設定方法

- bulletでのConstraint設定
  - btRigidBodyのようにbtDynamicWorldにbt\*Constraintを追加すればよい

```
btHingeConstraint *c = new btHingeConstraint(body1,body2,...);  
g_dynamicsworld->addConstraint(c, true);
```

- CleanBullet関数での破棄も忘れずに

```
for(int i = g_dynamicsworld->getNumConstraints()-1; i>=0 ;i--){  
    btTypedConstraint* constraint = g_dynamicsworld->getConstraint(i);  
    g_dynamicsworld->removeConstraint(constraint);  
    delete constraint;  
}
```

world中のConstraint数を取得

Constraintをワールドから削除

Constraintオブジェクトを取得



# Constraintの設定方法

- bulletでのConstraint設定
  - Soft Constraintの設定方法

```
btHingeConstraint *c = new btHingeConstraint(body1,body2,...);  
c->setParam(BT_CONSTRAINT_ERP, 0.5);  
c->setParam(BT_CONSTRAINT_CFM, 0.5);
```

\*基本的にはbullet側で適切な値が設定されるのでSoft Constraintを使いたくないなどの特殊なケース以外はいらない

# Bulletで扱えるConstraintの種類

## ■ Point2Point Constraint

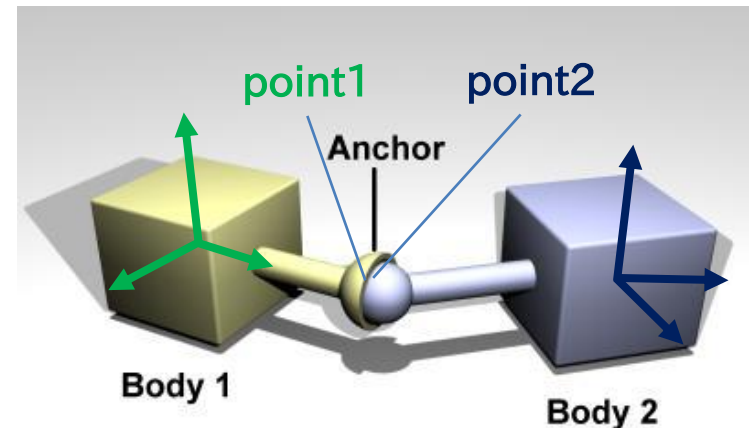
2つの物体を1点でつなぐ制約(位置固定で回転は自由)

```
btPoint2PointConstraint *c = new btPoint2PointConstraint(body1, body2,  
                                                         point1, point2);  
g_dynamicsworld->addConstraint(c, true);
```

point1とpoint2は同じ位置を示しているが、それぞれのbtRigidBodyから見た相対座標値(ローカル座標)なので値は異なることに注意!

1つの物体を空間上に固定する場合

```
btPoint2PointConstraint *c  
= new btPoint2PointConstraint(body1, point1);
```



# Bulletで扱えるConstraintの種類

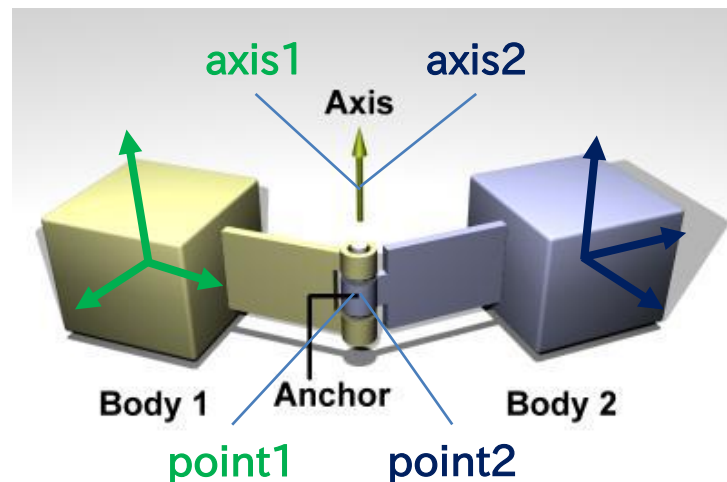
## ■ Hinge Constraint

1軸の回転のみを許す制約(ドアの蝶番,車のタイヤなど)

```
btHingeConstraint *c = new btHingeConstraint(body1,body2,  
                                              point1,point2,  
                                              axis1,axis2);  
g_dynamicsworld->addConstraint(c, true);
```

point1,point2,axis1,axis2はそれぞれのbtRigidBodyのローカル座標系での位置と方向. 特に方向は初期配置で決まるので注意

enableAngularMotorメンバ関数で一定速度の回転を加えることも可能



# Bulletで扱えるConstraintの種類

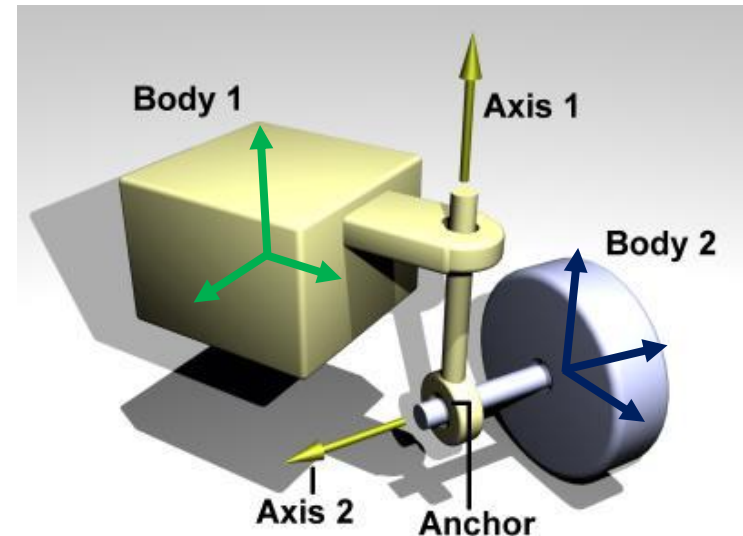
## ■ Hinge2 Constraint

直交した2軸の回転が可能な制約(車の前輪など)

```
btHinge2Constraint *c = new btHinge2Constraint(body1,body2,  
                                                anchor,  
                                                axis1,axis2);  
g_dynamicsworld->addConstraint(c, true);
```

anchor,axis1,axis2はHingeと違ってグローバル座標(ワールド座標)であることに注意

axis1とaxis2は直交するように設定



# Bulletで扱えるConstraintの種類

## ■ Slider Constraint

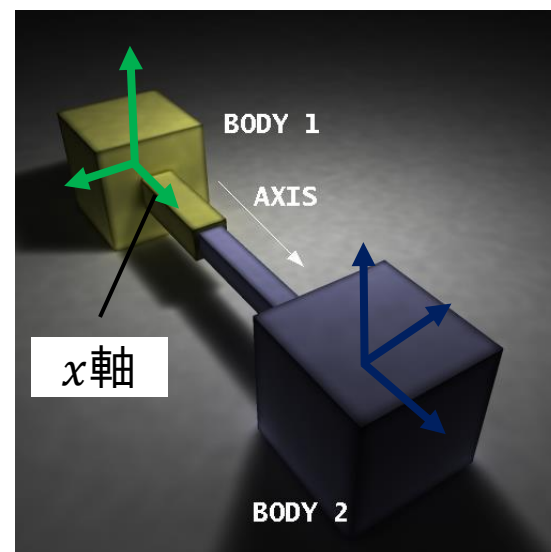
設定した1方向しか動けない制約(カーテンレールなど)

```
btSliderConstraint *c = new btSliderConstraint(body1,body2,  
                                                frame1,frame2,  
                                                ref1);  
g_dynamicsworld->addConstraint(c, true);
```

frame1とframe2はそれぞれbody1,body2の何も力がかかっていない状態での相対位置/姿勢(btTransform)

スライダーの移動方向はref1で指定したオブジェクトのx軸方向

(ref1がtrueでbody1,falseでbody2が基準になる)



# Bulletで扱えるConstraintの種類

## ■ Cone Twist Constraint

2つの物体を1点でつなぐ制約(位置固定で回転範囲設定)

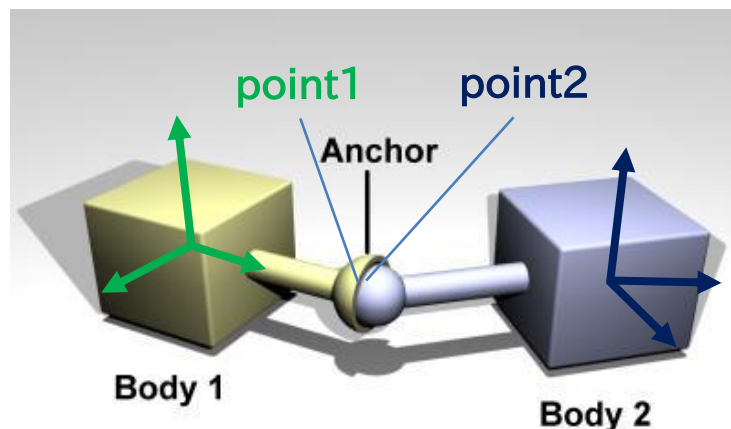
```
btConeTwistConstraint *c = new btConeTwistConstraint(body1, body2,  
                                                       frame1, frame2);  
g_dynamicsworld->addConstraint(c, true);
```

Point2Pointと基本的には同じだけど, 回転角に制限を掛けられるので, キャラクターの間接などに用いられる(ragdollジョイント)

frame1とframe2はそれぞれbody1, body2の相対位置/姿勢 (btTransform)

回転範囲の設定

```
C->setLimit(ang_z, ang_y, ang_x);
```



# Bulletで扱えるConstraintの種類

## ■ Generic 6 DOF Constraint

x,y,z軸方向の平行移動と回転に対する制約を自由に設定

```
btGeneric6DofConstraint *c = new btGeneric6DofConstraint(body1, body2,  
                                                         frame1, frame2  
                                                         ref1);  
g_dynamicsworld->addConstraint(c, true);
```

引数はSlider Constraintと同じなのでそちらを参照  
ここまで説明したConstraintはすべてこれで再現可能

平行移動制約の設定

```
C->setLinearLowerLimit(btVector3(x1,y1,z1));  
C->setLinearUpperLimit(btVector3(x2,y2,z2));
```

回転制約の設定

```
C->setAngularLowerLimit(btVector3(ang_x1,ang_y1,ang_z1));  
C->setAngularUpperLimit(btVector3(ang_x2,ang_y2,ang_z2));
```

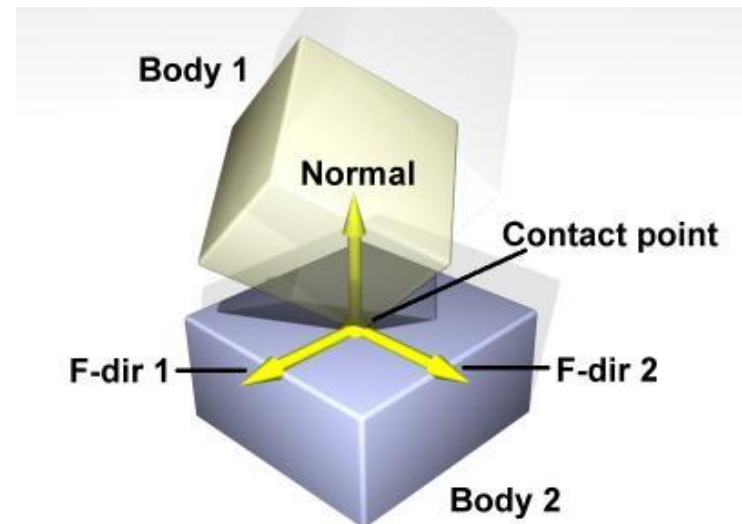
# Bulletで扱えるConstraintの種類

## ■ Contact Constraint

通常の衝突を処理するための制約

```
btContactConstraint *c = new btContactConstraint(manifold,body1,body2);
```

bullet側が自動的に生成してくれるので、わざわざ指定する必要なし  
(bulletがConstraintを使って衝突応答を処理しているということだけ  
理解していればOK)





# BulletでのConstraint

ここからは各自の環境で  
実際に作業

実験ページの「4. 剛体間リンク」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!)

