ODE で物理シミュレーション

- 物理シミュレーションとは？
  - 運動方程式の数値計算をしてくれる
  - ODEの場合は剛体の運動
  - 物が落ちたり、衝突したり

ODEの座標系

- 直交座標系（右手系）
  - 右手系
  - 原点：9個あるピラミッドの中央
  - X軸：原点から赤ピラミッド
  - Y軸：原点から青ピラミッド
  - Z軸：原点から上空
- 単位
  - SI単位系 (P59)
    - 長さ m
    - 質量 kg
    - 時間 s
    - 力 N

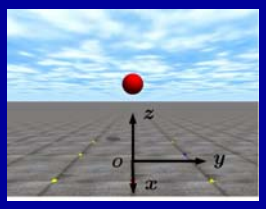

図1.2 ODEの座標系（教科書P9から転載）

ODE を使うときの main 関数

重力、地面、物体を設定する

```
int main(int argc, char **argv)
{setDrawStuff();                    // 描画関数を設定
 dInitODE(); // ODEの初期化
 world = dWorldCreate();            // シミュレーション世界を作る
 dWorldSetGravity(world,0,0,-0.2);  // 重力を設定
 space = dHashSpaceCreate(0);       // 衝突用空間を作る
 ground = dCreatePlane(space,0,0,1,0); // 平面ジオメトリ(地面)の生成
 contactgroup = dJointGroupCreate(0);  // ジョイントグループを作る
 SetParameters();                   // パラメータを設定する
 CreateObjects(world);              // 物体をつくる
 dsSimulationLoop(argc,argv,320, 240,&fn);// シミュレーションループ
 dSpaceDestroy(space);              // 衝突用空間を消す
 dWorldDestroy(world);              // シミュレーション世界を消す
 dCloseODE(); // ODEの終了
 return 0;}
```

CreateObjects 関数

球を作って
初期位置を設定

```
void
CreateObjects(dWorldID w)
{
 CreateSphere(&apple, w, 0.2, 1.0, 1.0, 1.0, 0.0, 0.0);
                                                  // 球の生成
 dBodySetPosition(apple.body, 0.0, 0.0, 2.0);
                                  // 球の位置(x,y,z)を設定
 CreateSphere(&ball, w, 0.1, 1.0, 0.5, 0.0, 0.0, 1.0);
                                                  // 球の生成
 dBodySetPosition(ball.body, 0.5, 0.0, ball.r);
                                  // 球の位置(x,y,z)を設定
}
```

直方体などを作るときは、この関数を基に

CreateSphere 関数

```
void    CreateSphere(struct sphere *s,
   dWorldID world, dReal r, dReal m,
   dReal bounce,
   dReal R, dReal G, dReal B);
```

ポインタ s に、世界 world の,
半径 r, 質量 m, 地面との反発係数 bounce
色 R, G, B
の球を作る

dBodySetPosition 関数

```
void dBodySetPosition(BodyID id,
   dReal x, dReal y, dReal z)
```

id 番の剛体の位置を (x, y, z) にする

command 関数

```
void command(int cmd)
{  const dReal *apos = dBodyGetPosition(apple.body);
 switch(cmd) {
 case ' ':
  if (dBodyIsEnabled(
    dBodyDisable(appl
  else
    dBodyEnable(apple.body);
  break;
 case 'j':
  dBodySetPosition(apple.body, apos[0]-.1, apos[1],
  apos[2]);
  break;
```

キーボードからの入力に関係
スペースで赤い球が運動開始/停止
j, k で赤い玉の x 座標を減/増
r でリセット

演習 1

- hello.cpp をダウンロードして、ビルド・実行してみる
- hello-ode.cpp をダウンロードして、ビルド・実行してみる
- hello-ode.cpp の command 関数を変えて
  - k で赤い球の y 座標を減らし
  - l で赤い球の y 座標を増やす

  ようにする

演習 2

- hello-ode.cpp の CreateObjects 関数で球の数を増やしてみる

物体を作って表示、捨てる

物体に必要な構造体の変数を用意する
例）

```
struct sphere apple, ball;
```

すべての物体について、3つの関数内に書く:

- CreateObjects()　作る
- DrawObjects()　表示
- DestroyObjects()　捨てる

CreateObjects 関数

球を作って
初期位置を設定

```
void
CreateObjects(dWorldID w)
{
 CreateSphere(&apple, w, 0.2, 1.0, 1.0, 1.0, 0.0, 0.0);
                                          // 球の生成
 dBodySetPosition(apple.body, 0.0, 0.0, 2.0);
                                  // 球の位置(x,y,z)を設定
 CreateSphere(&ball, w, 0.1, 1.0, 0.5, 0.0, 0.0, 1.0);
                                          // 球の生成
 dBodySetPosition(ball.body, 0.5, 0.0, ball.r);
                                  // 球の位置(x,y,z)を設定
}
```

DrawObjects関数

```
void
DrawObjects()
{
 DrawSphere(&apple);
 DrawSphere(&ball);
}
```

DestroyObjects関数

```
void
DestroyObjects()
{
  DestroyObject(apple.body, apple.geom);
  DestroyObject(ball.body, ball.geom);
}
```

引数に渡すメンバ変数に注目!

```
// 反発係数を求める
dReal
getgBounce(dGeomID id)
{
  if (id == apple.geom)
    return apple.gBounce;
  else if (id == ball.geom)
    return ball.gBounce;

  return 1.0;
}
```

実行を速く

- テキスチャを切る
- 影を切る

- 重力を大きくする
- シミュレーションステップを大きくする
  - ときどき結果がおかしくなるので注意