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

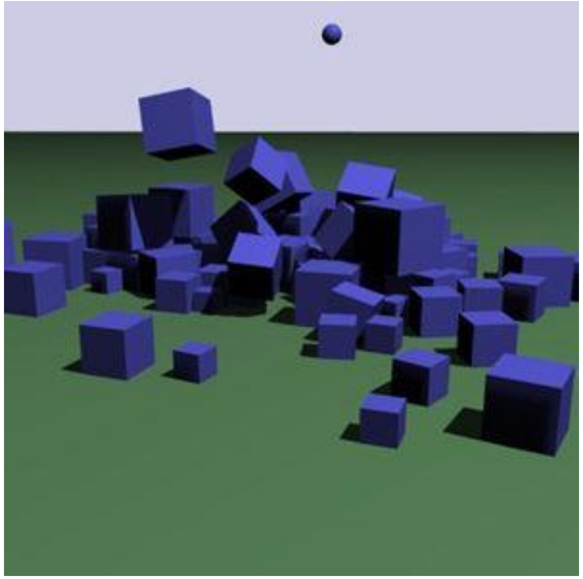
実験の目的

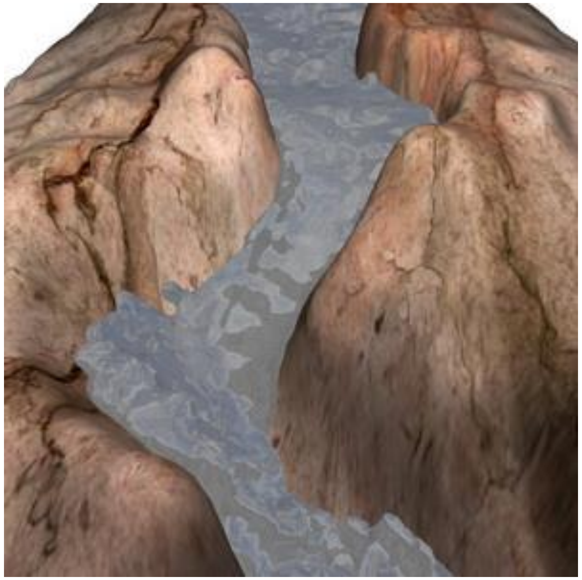
物理シミュレーションエンジンを利用したアプリケーションが作れるようになる

- 教員による説明回+講義ページの説明を理解して練習問題を解く(実装する)
- 物理エンジンを用いた簡単なアプリケーションを作成

物理シミュレーションとは?

- 物体の物理運動をリアルに再現するための手法 → CG作成によく使われる
  - 質量・速度・摩擦・風といったものを物理法則に基づきシミュレーション





物理エンジンで使う数学

- ニュートンの運動方程式

$$\boldsymbol{F} = \boldsymbol{ma}$$

力[N] =質量[kg] × 加速度[m/s^2]

⇩

- 物体に力を加えたときにどれぐらい加速するかが分かる
- 物体を描画するときに必要なのは位置$\boldsymbol{x}$

物理エンジンで使う数学

- ニュートンの運動方程式

$$\boldsymbol{F} = m\ddot{\boldsymbol{x}}$$

$\ddot{\boldsymbol{x}}$: $\boldsymbol{x}$の時間2階微分($\frac{d^2\boldsymbol{x}}{dt^2}$)

⇩ 1階微分×2に分解

$$\ddot{\boldsymbol{x}} = \frac{d\dot{\boldsymbol{x}}}{dt} = \frac{d\boldsymbol{v}}{dt}, \qquad \boxed{\dot{\boldsymbol{x}} = \boldsymbol{v} = \frac{d\boldsymbol{x}}{dt}}$$

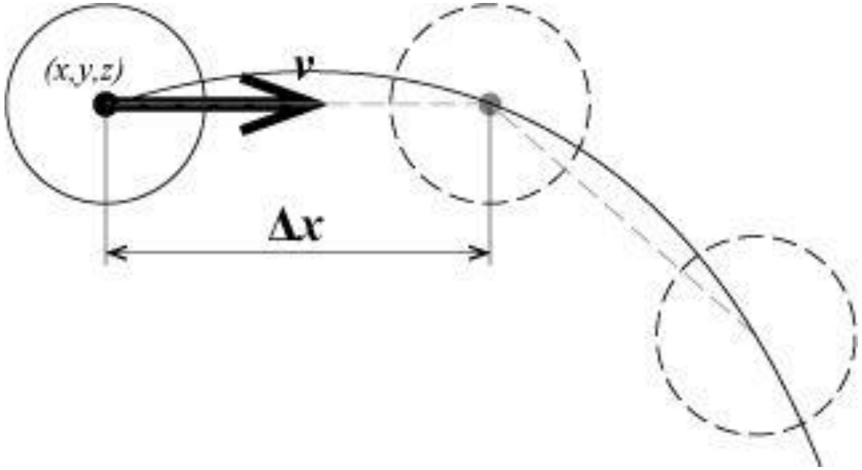
物理エンジンで使う数学

- 時間積分の離散化

$$\boldsymbol{v}(t) = \frac{d\boldsymbol{x}}{dt} = \left(\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt}\right)$$

位置$\boldsymbol{x}$の時間変化

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t) + \Delta\boldsymbol{x}$$



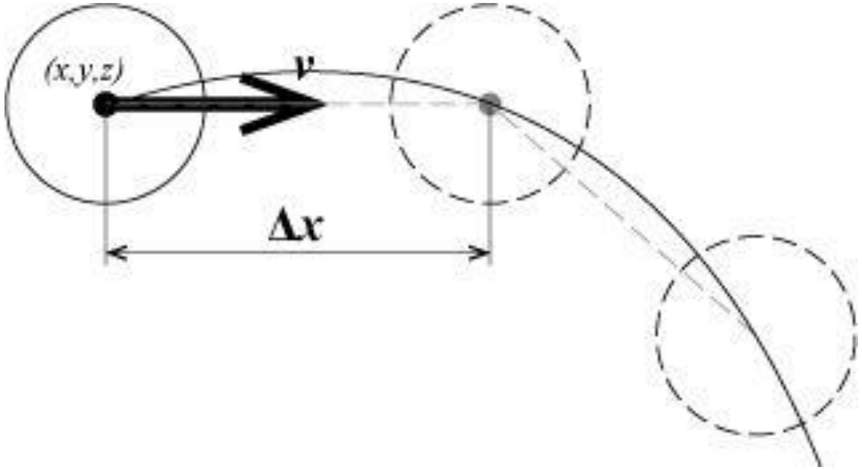
物理エンジンで使う数学

- 時間積分の離散化

$$\boldsymbol{v}(t) = \frac{d\boldsymbol{x}}{dt} = \left(\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt}\right)$$

位置$\boldsymbol{x}$の時間変化

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t) + \boxed{\Delta \boldsymbol{x}}$$



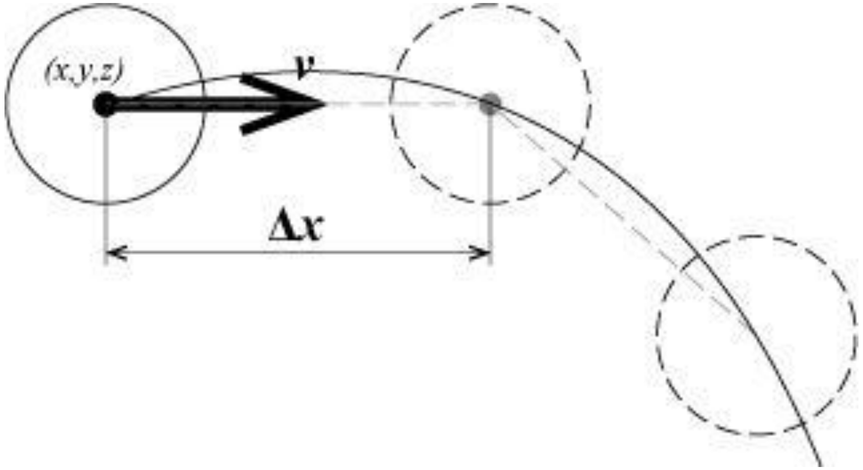
物理エンジンで使う数学

- 時間積分の離散化

$$\Delta \boldsymbol{x} = \int_{\mathrm{t}}^{t+\Delta t} \boldsymbol{v}(t) dt$$

速度がtから$t+\Delta t$の間一定の値だと仮定

$$\boldsymbol{x}(t+\Delta t) = \boldsymbol{x}(t) + \boldsymbol{v}(t)\Delta t$$ **:前進オイラー法**



物理エンジンで使う数学

- ここまでのまとめ
  - 物体を動かすためには力Fを与える
  - Fから運動方程式で加速度，加速度から前進オイラー法で速度⇒位置と計算
    - 時間ステップ幅Δtを与える必要がある
    - 精度はΔtが小さいほど良くなる(計算時間は大きくなる)

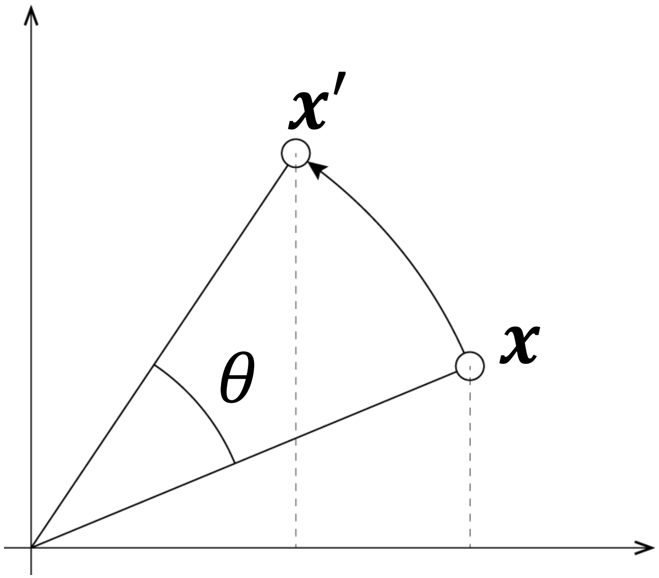
物理エンジンで使う数学

- 行列を使った回転の表現
  2次元の回転⇒角度θで表現可能

$$\boldsymbol{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

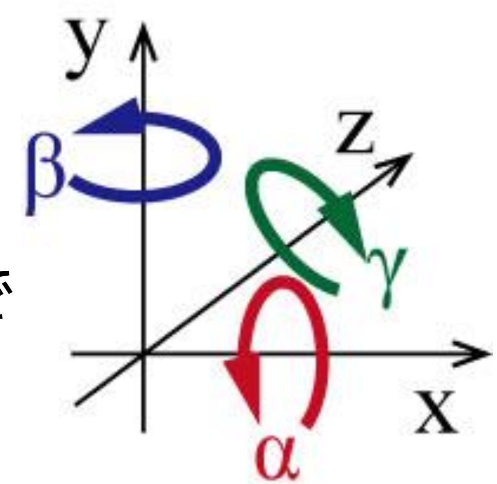
⇩

$$\boldsymbol{x}' = \boldsymbol{R}\boldsymbol{x}$$



物理エンジンで使う数学

- 行列を使った回転の表現
  3次元の回転⇒？
  - オイラー角
    x, y, z軸周りの回転角(α, β, γ)
    で回転を表現
    - それぞれに回転行列R_x, R_y, R_z
      があり，それを合成することで
      1つの行列で回転を表現



計算が大変, 1つの回転に9つの要素が必要

(ジンバルロックの問題もある)

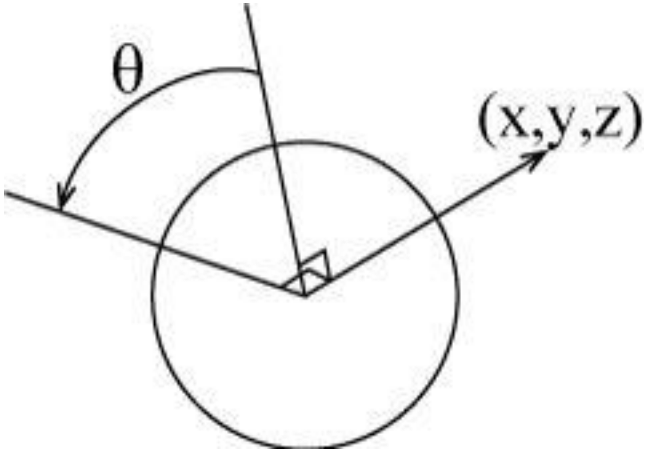
物理エンジンで使う数学

- 四元数
 実数部(スカラー部)1つ,虚数部(ベクトル部)3つを持つ4次元数

$$\boldsymbol{q} = (\boldsymbol{v}, s) = iv_x + jv_y + kv_z + s$$

ベクトル部を回転軸，スカラー部を回転角と考えると任意の回転を表現可能

$$\boldsymbol{q} = \left(\underbrace{x \sin\frac{\theta}{2}, y \sin\frac{\theta}{2}, z \sin\frac{\theta}{2}}_{\text{ベクトル部}}, \underbrace{\cos\frac{\theta}{2}}_{\text{スカラー部}} \right)$$



物理エンジンで使う数学

- 四元数を使った回転の表現

$$\boldsymbol{x}' = \boldsymbol{q}\,(\boldsymbol{x}, 0)\,\boldsymbol{q}^{-1}$$

- 四元数で覚えておくこと
  - 単位四元数
    $$\boldsymbol{q} = (0, 0, 0, 1)$$
  - 逆回転を表す四元数(共役四元数)
    $$\boldsymbol{q}^{-1} = (-\boldsymbol{v}, s)$$
  - 回転行列と相互変換可能

物理エンジンで使う数学

- 四元数$\boldsymbol{q} = (v_x, v_y, v_z, s)$から回転行列への変換

$$R = \begin{pmatrix} 1 - 2v_y^2 - 2v_z^2 & 2v_xv_y - 2sv_z & 2v_xv_z + 2sv_y \\ 2v_xv_y + 2sv_z & 1 - 2v_x^2 - 2v_z^2 & 2v_yv_z - 2sv_x \\ 2v_xv_z - 2sv_y & 2v_yv_z + 2sv_x & 1 - 2v_x^2 - 2v_y^2 \end{pmatrix}$$

  - ほとんどの物理エンジンは四元数を扱うクラス(bulletだとbtQuaternion)で姿勢等を指定できるが，描画側(OpenGL)は回転行列を使っている

    ⇒ 回転行列への変換が必要となる

物理エンジンで使う数学

- ニュートンの運動方程式(回転版)

$$\boldsymbol{T} = \boldsymbol{I}\dot{\boldsymbol{\omega}}$$

トルク[N･m] = 慣性テンソル×角加速度[rad/s^2]

- 角速度$\boldsymbol{\omega}$は角度θor四元数$\boldsymbol{q}$の時間微分
- 慣性テンソルは回転版質量のようなもの
  - 回転のしにくさを表す
  - 同じ質量でも形状やまわす軸でまわしやすさは変わる
  - 物理エンジンにはこれを計算するための関数が用意されている

物理エンジンで使う数学

- 回転版の前進オイラー法(四元数使用)

$$\boldsymbol{\omega} = \frac{\partial \boldsymbol{\theta}}{\partial t}, \dot{\boldsymbol{\omega}} = \frac{\partial \boldsymbol{\omega}}{\partial t}$$

角度$\boldsymbol{\theta}$を四元数$\boldsymbol{q}$にして離散化

$$\boldsymbol{q}(t + \Delta \mathrm{t}) = \boldsymbol{q}_w \boldsymbol{q}(t)$$

$$\boldsymbol{\omega}(t + \Delta t) = \boldsymbol{\omega}(t) + \dot{\boldsymbol{\omega}}(t)\Delta \mathrm{t}$$

並進運動版： $\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t) + \boldsymbol{v}(t)\mathit{\Delta} t$

物理エンジンで使う数学

- 1ステップあたりの姿勢変化$\boldsymbol{q}_w$について
  - 1ステップあたりの姿勢変化量：$\boldsymbol{\omega}\Delta t$
    ⇒ これを四元数変化量に変換すれば良い
    - ベクトル$\boldsymbol{\omega}\Delta t$ をオイラー角 (α, β, γ) と考えて,

$$\boldsymbol{q}_w = \boldsymbol{q}_x\boldsymbol{q}_y\boldsymbol{q}_z = \begin{pmatrix} \sin\frac{\alpha}{2}\cos\frac{\beta}{2}\cos\frac{\gamma}{2} - \cos\frac{\alpha}{2}\sin\frac{\beta}{2}\sin\frac{\gamma}{2} \\ \cos\frac{\alpha}{2}\sin\frac{\beta}{2}\cos\frac{\gamma}{2} + \sin\frac{\alpha}{2}\cos\frac{\beta}{2}\sin\frac{\gamma}{2} \\ \cos\frac{\alpha}{2}\cos\frac{\beta}{2}\sin\frac{\gamma}{2} - \sin\frac{\alpha}{2}\sin\frac{\beta}{2}\cos\frac{\gamma}{2} \\ \cos\frac{\alpha}{2}\cos\frac{\beta}{2}\cos\frac{\gamma}{2} + \sin\frac{\alpha}{2}\sin\frac{\beta}{2}\sin\frac{\gamma}{2} \end{pmatrix}^T$$

と変換できる.

GLFWによるOpenGLアプリケーション

- 本実験テーマではGLFWライブラリによってアプリケーションを構築しています.
  - OpenGLはグラフィック用APIなので
    GUI関係は別に実装する必要がある
    ⇒ GLUT,GLFWなど
  - GLFWはOpenGLのための最低限のGUI環境しか提供しないので，本実験は別途ImGUIというものを用いている.
  - 実験サンプルはすべてGLFWベースですが,
    別のを使っても構わない(ただし自己責任で)

実際のコードを見せながら構造を解説します.