ゲーム物理の扱い方

株式会社　ソニー・コンピュータエンタテインメント
松生裕史、高橋律視、櫻井亮介

内容

1. ゲーム物理とは
   櫻井亮介

2. 剛体物理の扱い方
   松生裕史

3. 流体シミュレーションの素を意識する
   高橋律視

ゲーム物理とは

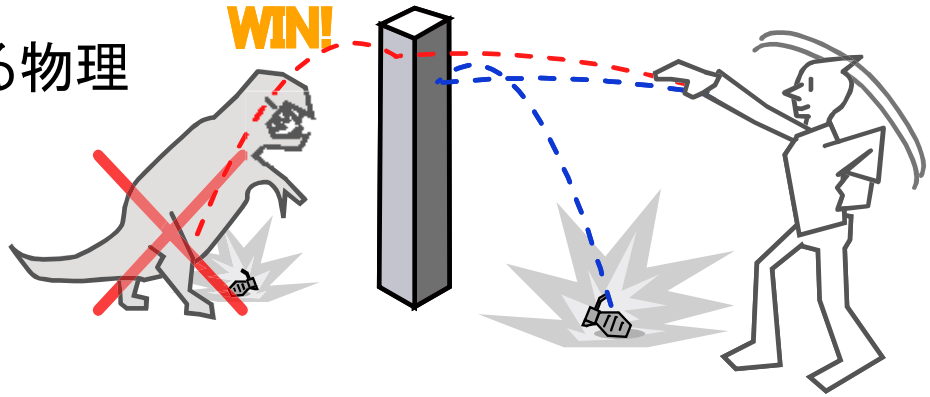
ゲーム物理の使われ方

演出効果としてのゲーム物理

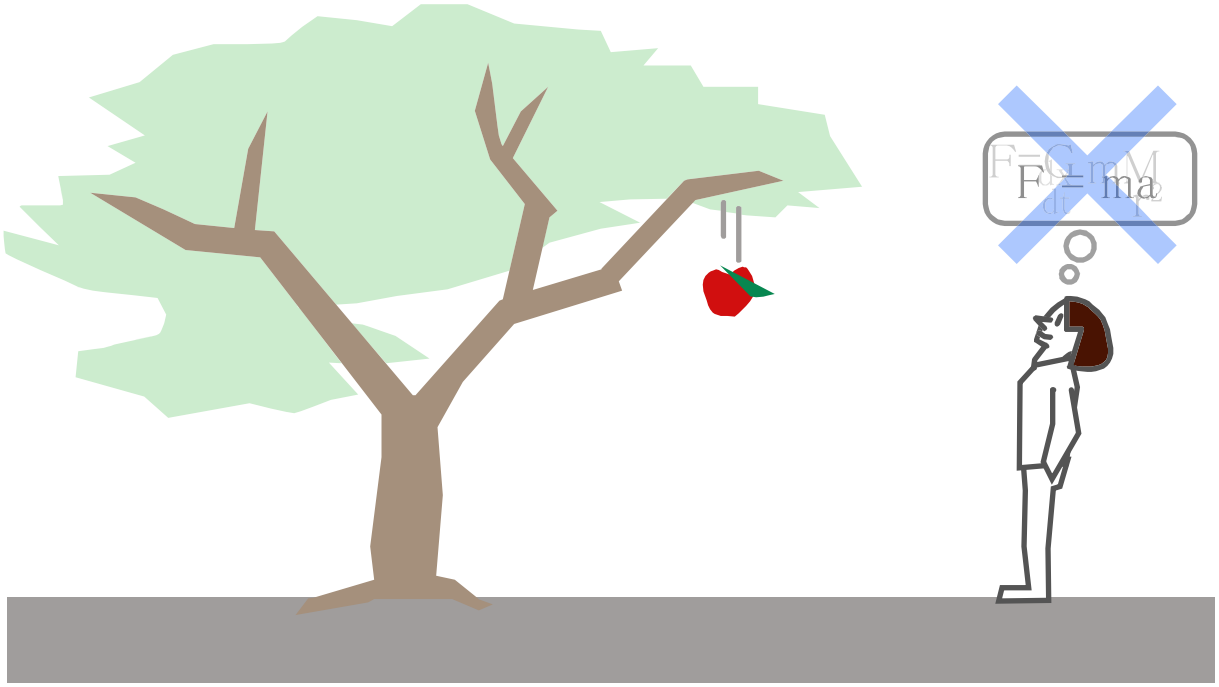
水しぶき、服、髪の揺れの演出
日本のゲームに多い

ゲームロジックに深くかかわるゲーム物理

ゲームの勝敗にかかわる物理
海外のゲームに多い



物理の印象...



りんごは目に見えるが物理法則は見えない・・・

現在のゲーム物理で扱うもの

- 剛体

- 流体（水、炎、花火、煙の動き）、柔軟体、布等
- パーティクル

- ジョイント（スライダ 、ちょうつがい等）
- レイキャスト

- 組み合せ
  ビークル、ラグドール

物理といっても... (1)

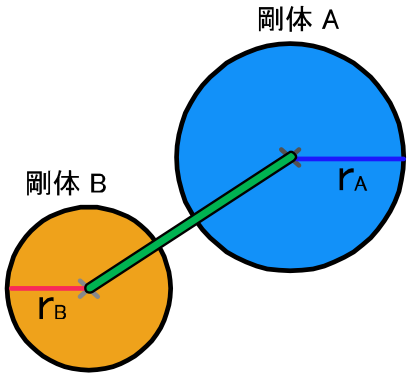
物理エンジンは身近にある物理現象を
表現する為の再利用可能な形で抜き出した
プログラム集

物理といっても古典的な力学が中心で
電磁気学、光学、相対性理論、量子力学などは
ほとんど扱いません

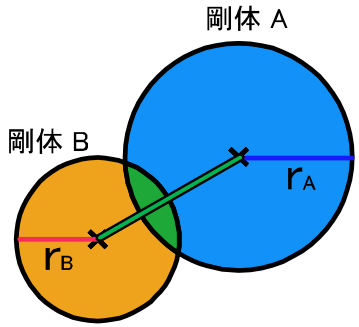
物理といっても... (2)

物理以外の要素も入っている

衝突判定

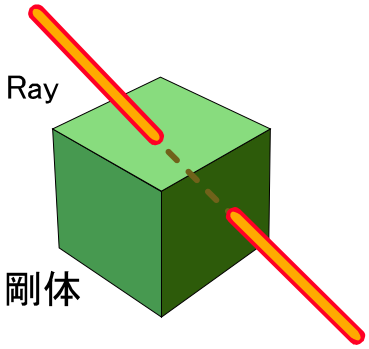


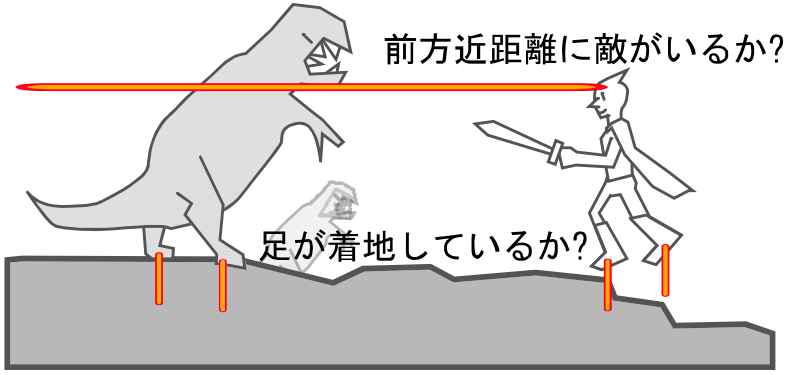
中心間の距離 ＞ 剛体A,Bの半径の和



中心間の距離 ≦ 剛体A,Bの半径の和

RayCast　Ray(=光線) Cast(=投射)

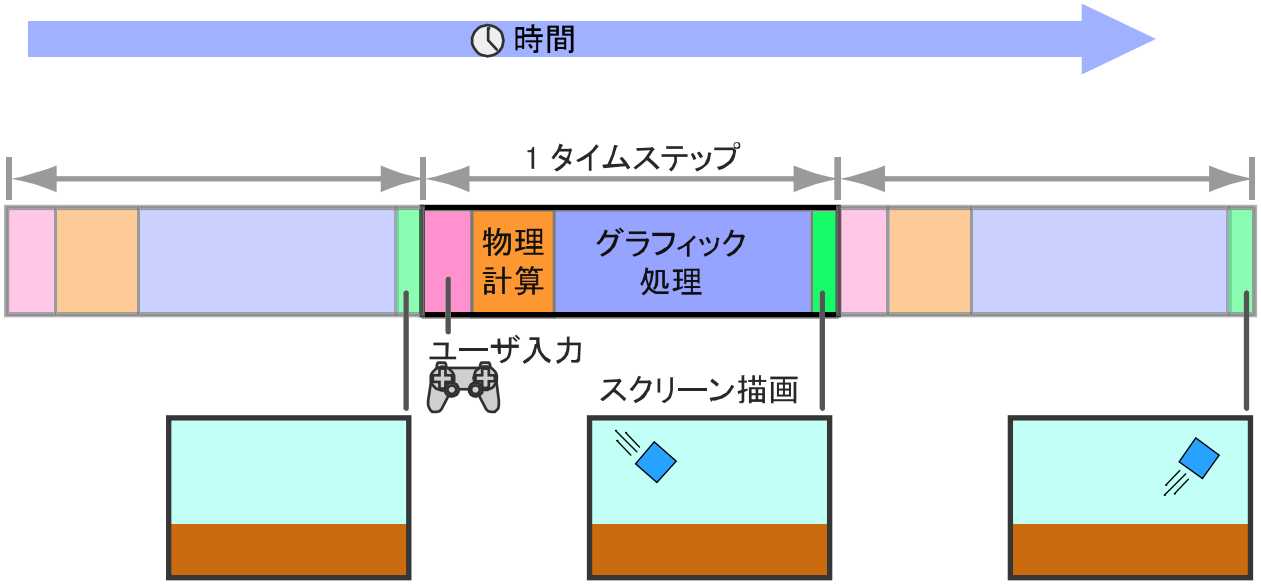




物理といっても... (3)

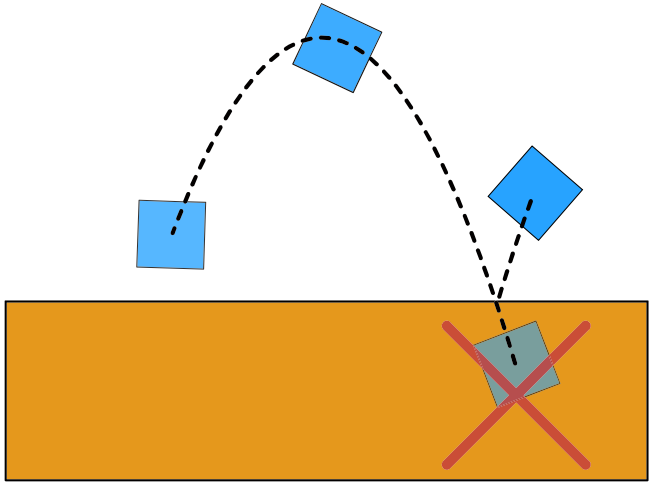
学術分野、映画などオフラインでの映像の
物理ほど厳密なシミュレーションをしなくてもよい

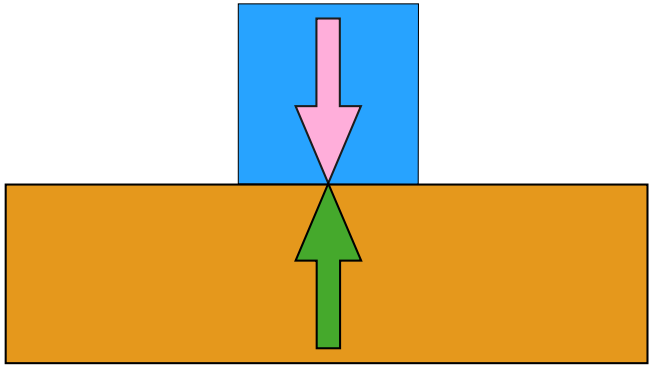
ゲーム物理の場合はサクサクと気持ちよく
動かすことが求められる
(実行時に与えられた時間が少ない)



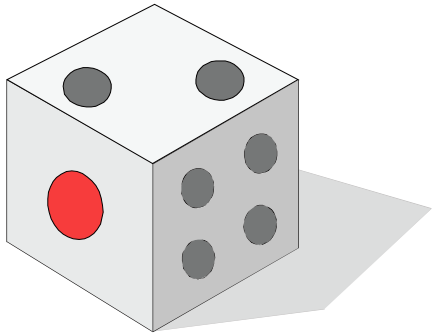
物理計算とは

複数オブジェクトの相互作用を物理現象に基づいて解決している

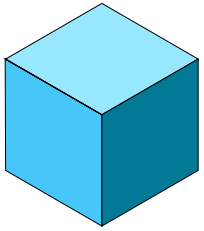




オブジェクトの表現



形状



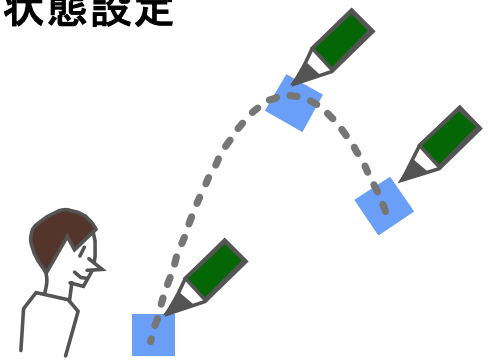
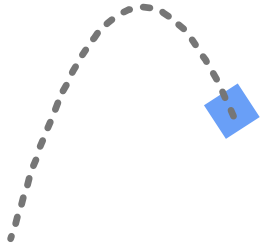
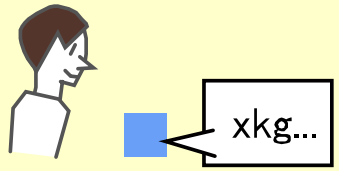
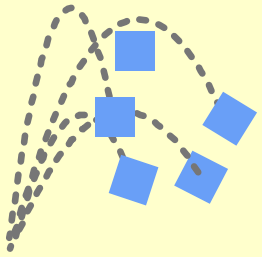
状態

位置
姿勢
速度
・・・

物理属性

質量
摩擦係数
・・・

アニメーションとゲーム物理

分類	ゲーム作成時の設定	実行時の特徴
キーフレーム アニメーション	アーティストによる個々の 状態設定 	作成される挙動パターンの数が 限られる 
ゲーム物理	物理属性の設定 	挙動パターンの数は無限 物理法則に従った挙動 

ゲーム物理の悪い点、良い点

×悪い点

難しい、扱いづらそう
実行時に計算コストがかかる
QAに時間がかかりそう

○良い点

アーティストの作業を軽減できる
リアルな表現
- その時、その場1度限りの体験、臨場感
- 言語、文化、世代を越えた世界中の人の共通体験をゲームに取り入れることができる

剛体物理の扱い方

はじめに

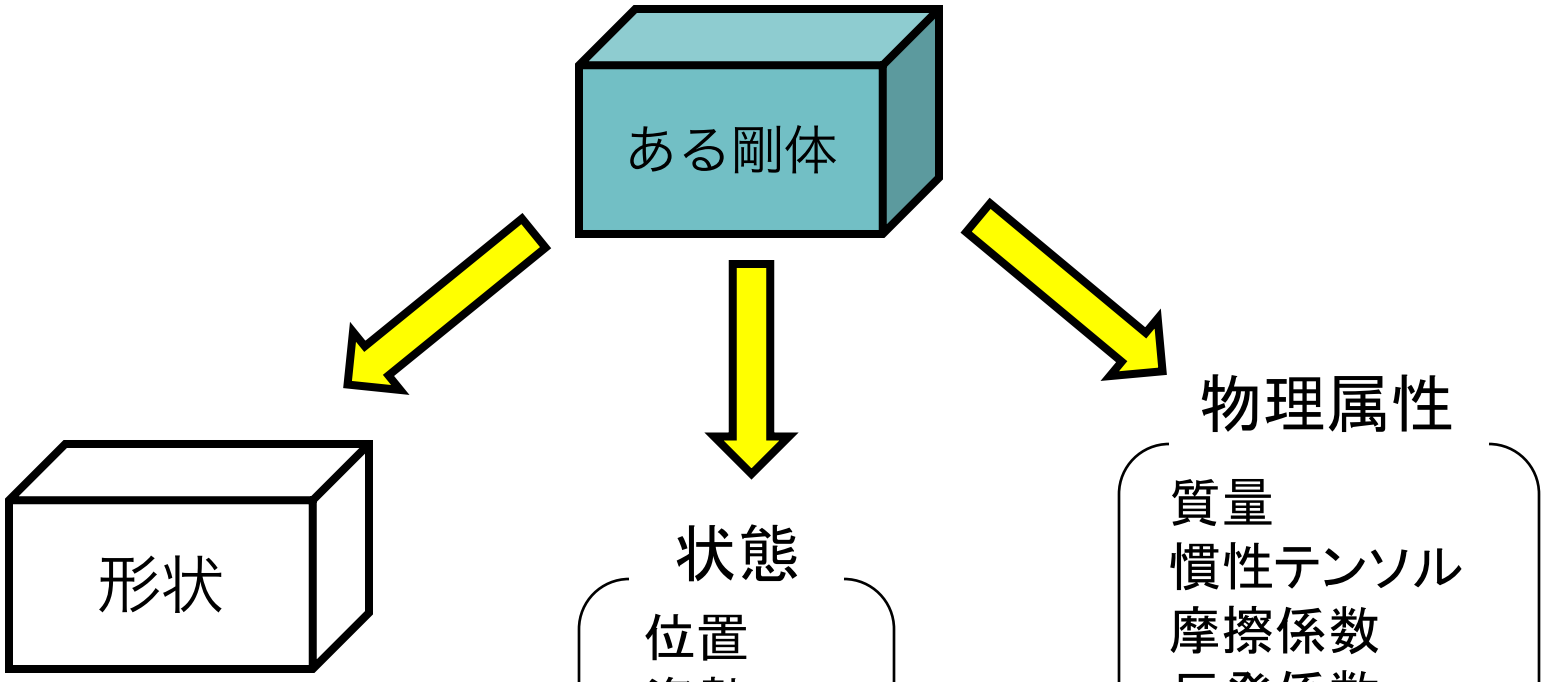
- 基本方針
  - 現実世界を忠実に再現することが目的ではない
  - ゲームに都合のよい挙動を再現することが重要
  - それっぽく動けばOK

- ゲーム物理の現実
  - 現実と同じ物理属性を設定すると不安定になる
  - パラメータと挙動の関連性が実感できない
  - 挙動が安定するパラメータの範囲が非常に狭い
  - ラグドールの関節（コンストレイント）が伸びる

ほとんどの問題は基本原理を理解していれば回避できる

ゲーム物理の基本原理

剛体の表現



当たりの判定に使われる

世界のどこにどんな姿勢で
存在しているのか？
時間が経つとどこへ進むか

剛体の持つ物理的な性質。
反発等の物理演算に使われる

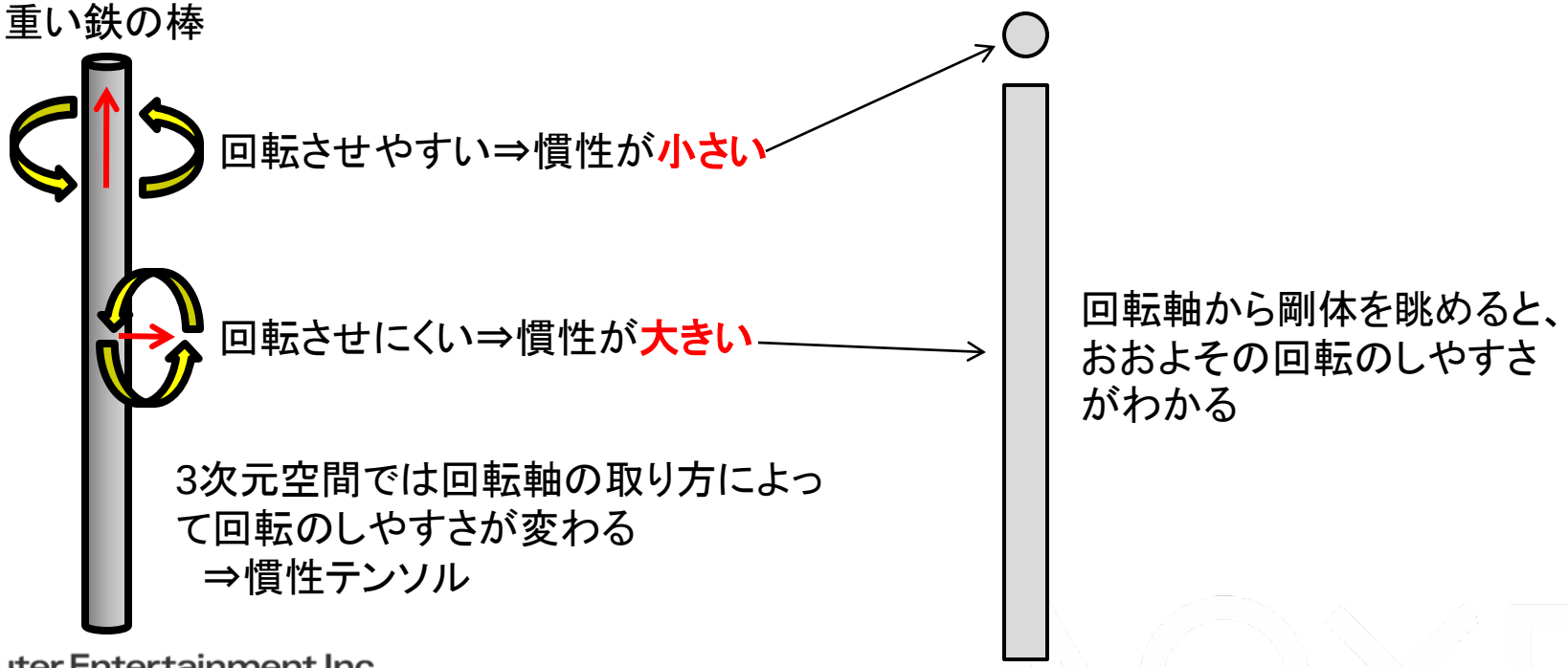
剛体の運動を直線と回転に分ける

- **質量は直線運動のしにくさを表現**
- **慣性テンソルは回転運動のしにくさを表現**

別々に指定することで直線運動と回転運動を個別にコントロールできる

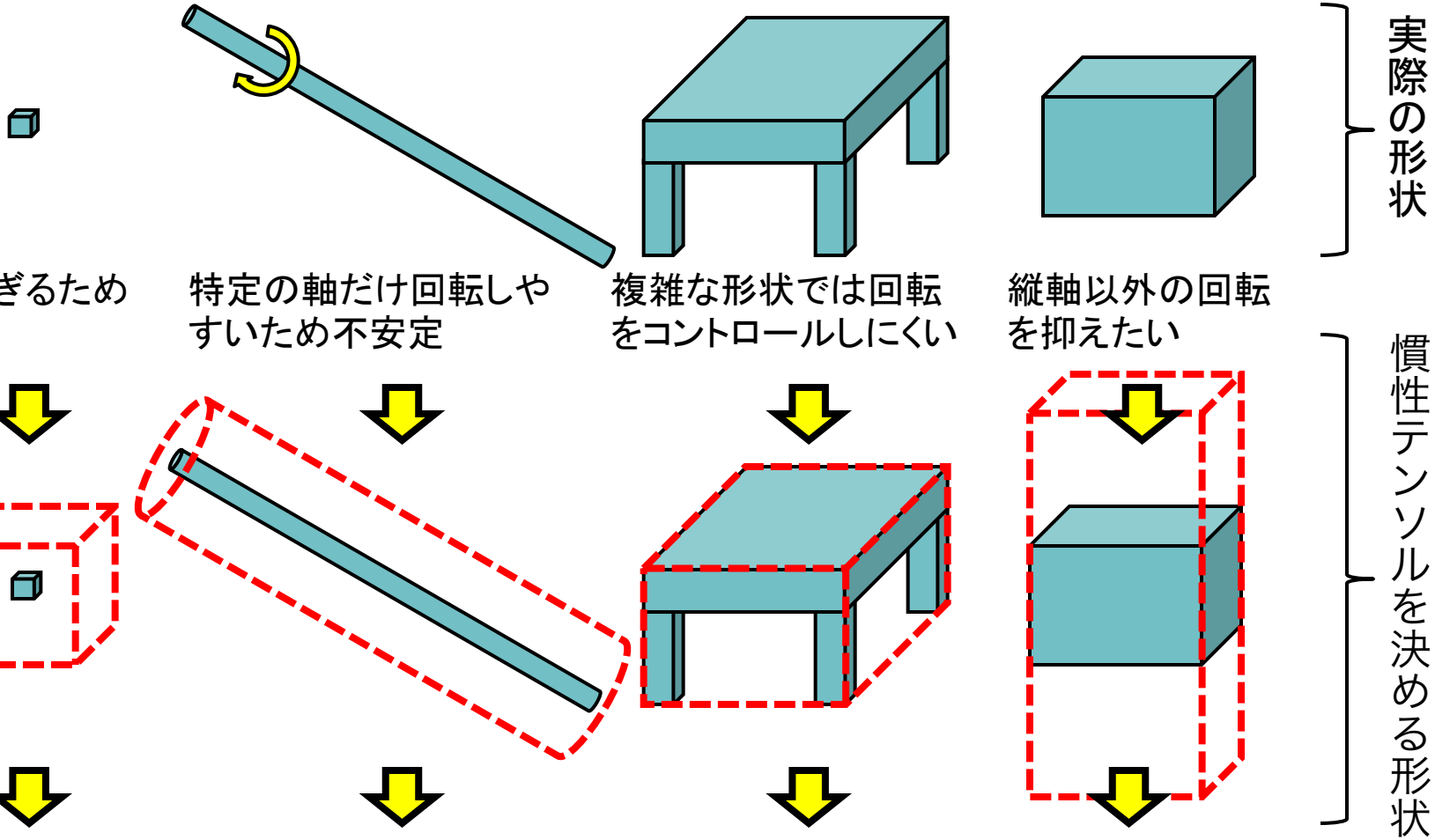
慣性テンソルが小さいと微振動が収まりにくい⇒不安定

慣性テンソルのイメージ



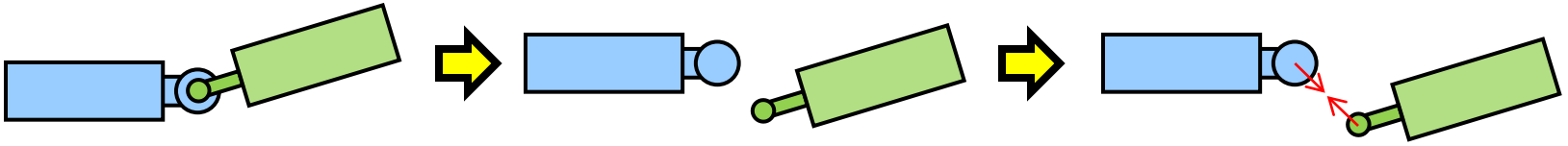
慣性テンソルの調整

- 慣性テンソルは質量と形状で決まる

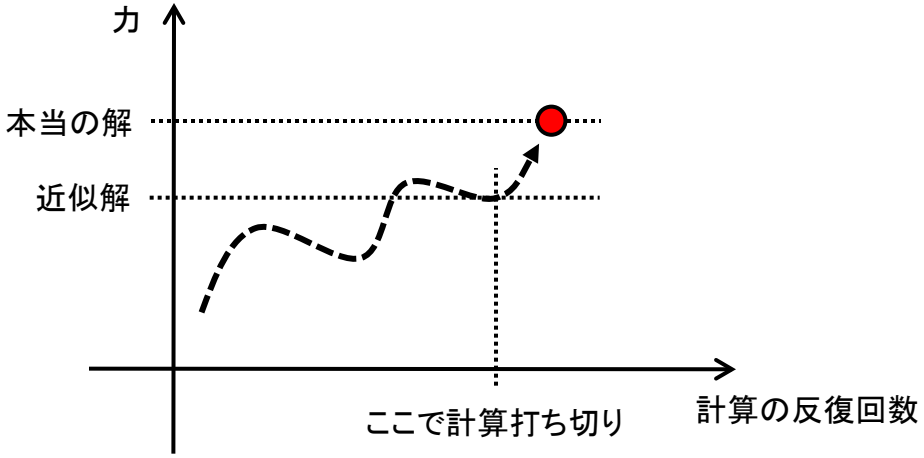


なぜ関節が伸びるのか

- ほっとくと接合部はずれる
- 物理エンジンは接合部を元に戻すための「力」を計算する
- 計算を繰り返して最適解へ近づける



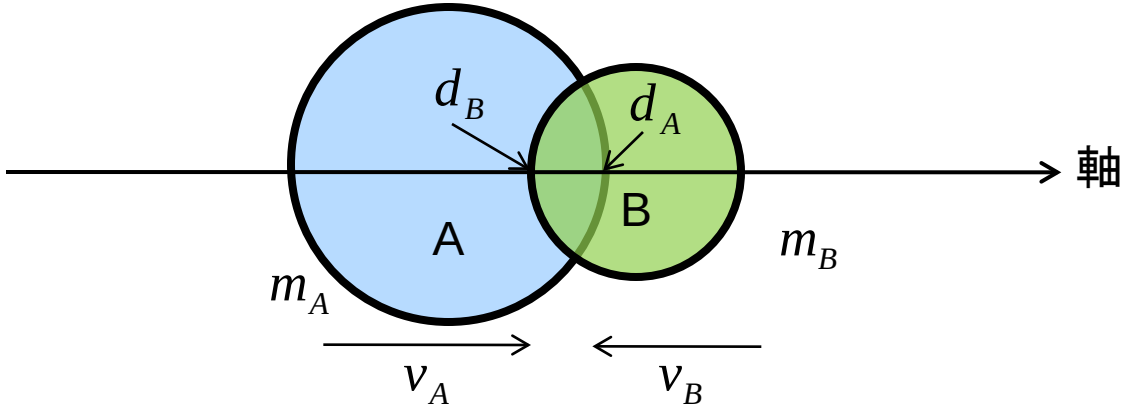
通常5～10回程度の反復回数で計算を打ち切るため結果は近似解になる



結果、ラグドールの関節が伸びるという現象につながる

コンストレイントの直感的な理解

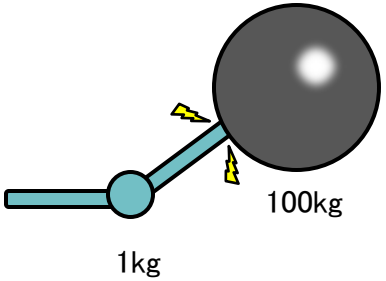
- コンストレイントとは剛体の動きに制約をかける仕組み
  - 衝突も関節もコンストレイントで表現する



コンストレイント簡易式⇒質量比×(調整値α×速度エラー ＋調整値β×位置エラー)

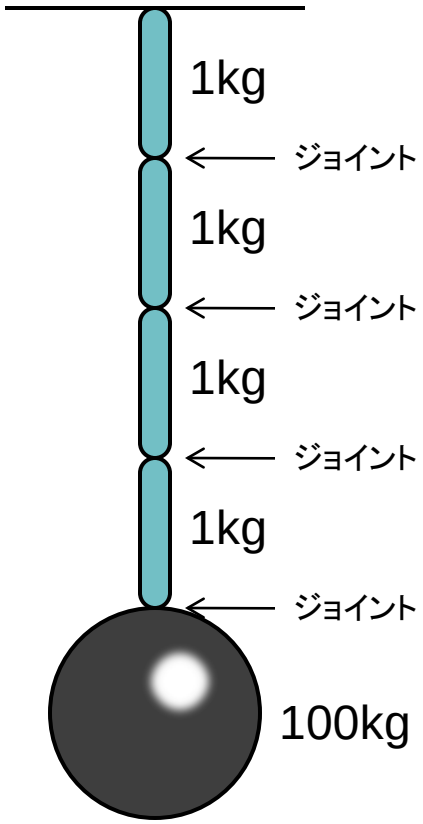
$$v_A' = v_A - \frac{m_B}{m_A + m_B}\left(\alpha\left(v_A - v_B\right) + \beta\left(d_A - d_B\right)\right) \qquad v_B' = v_B + \frac{m_A}{m_A + m_B}\left(\alpha\left(v_A - v_B\right) + \beta\left(d_A - d_B\right)\right)$$

2つの剛体の質量(＋慣性テンソル)の違いが大きく影響する



腕の速度変化の大きさ＝0.99×(エラー補正値)
錘の速度変化の大きさ＝0.01×(エラー補正値)

コンストレイントの調整例



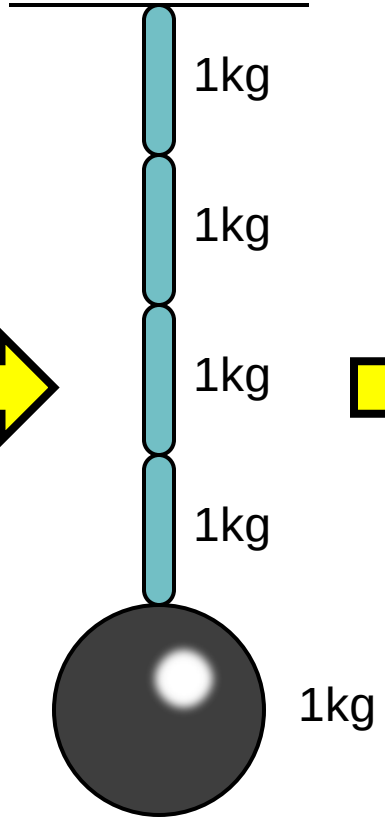
錘に引っ張られ、ひもは接続された状態を維持できない

接続状態を維持するためにはどうすればよいか？

各接続位置の関節にかかる力を均等にする

調整案その1

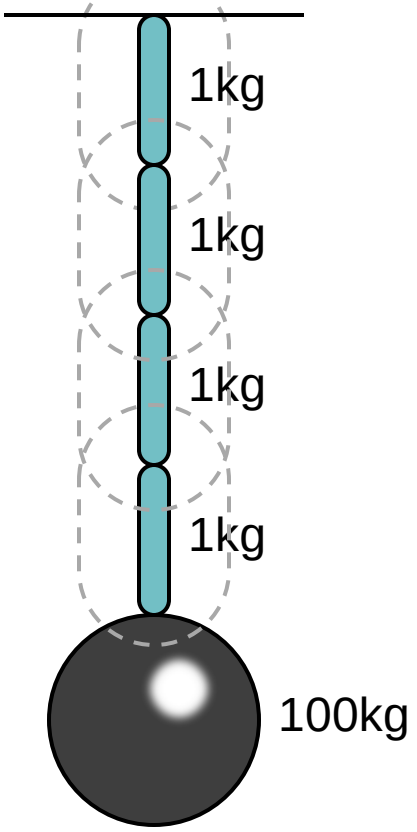
全部同じ重さにする



確かに関節は伸びなくなるが、錘の挙動が軽く不自然に見える。

調整案その2

ひもの慣性テンソルだけ増加

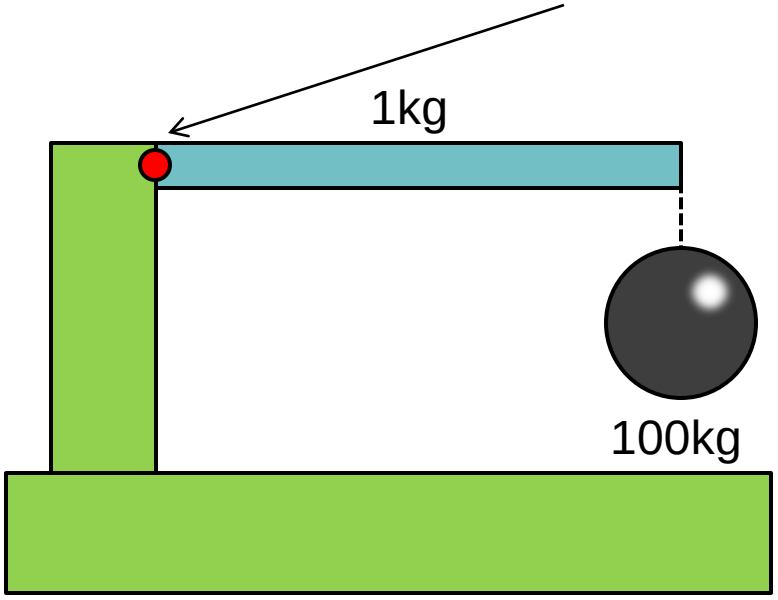


接続状態はほぼ維持され、かつ挙動も自然

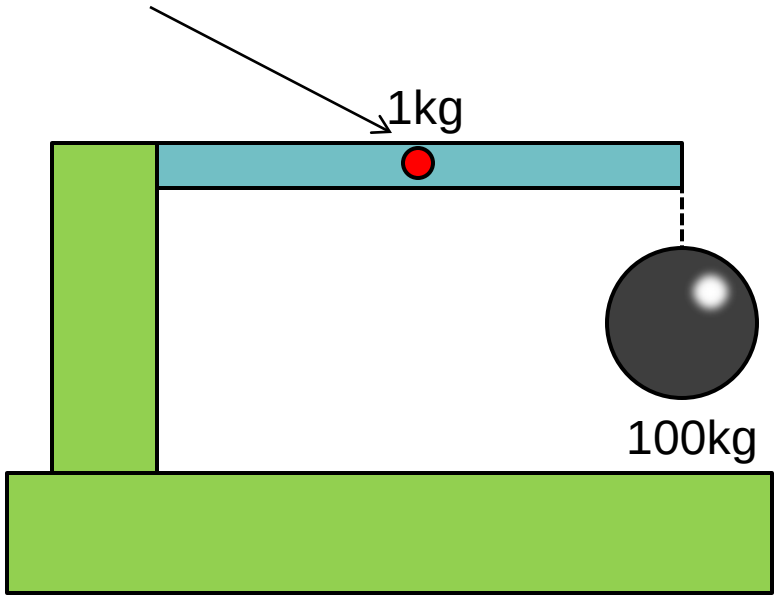
物理調整テクニック

コンストレイントの接続点

壁に棒を固定するコンストレイントの接続点



現実の世界と同じ接続点の設定

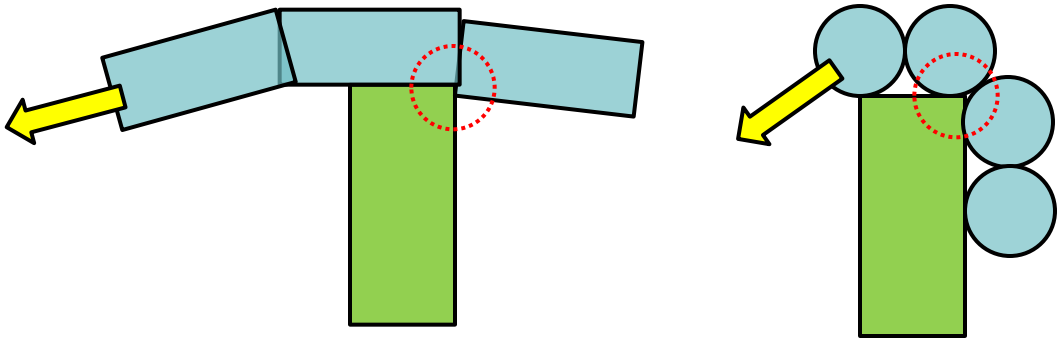


現実ではありえないが、安定した接続点

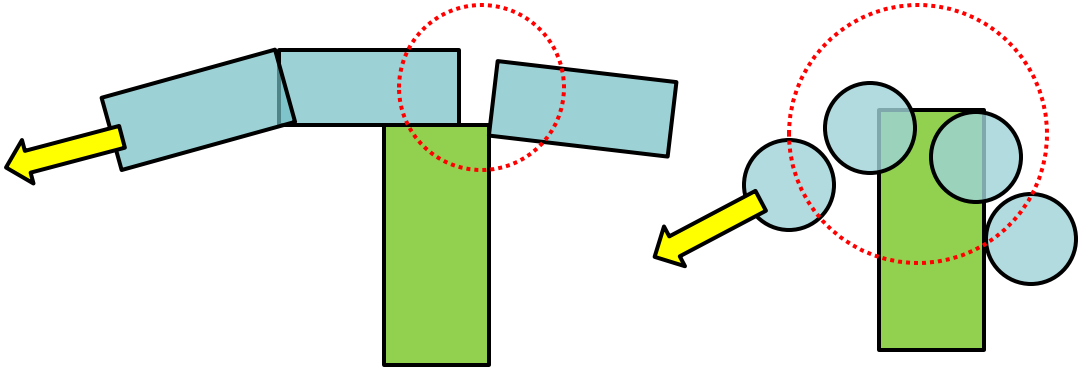
接続点はどこでも良い⇒最も安定する位置を選ぶ

形状の選択による挙動の違い

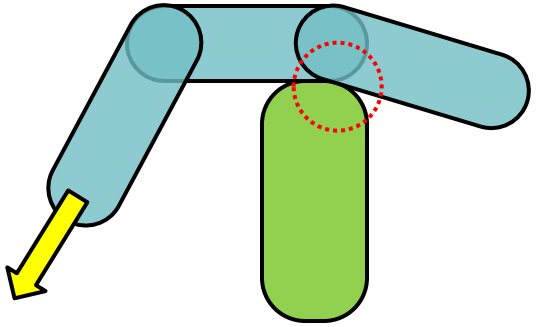
- 剛体を繋げて「ひも」を作るケース



角ばった形状や隙間のできる形状は引っ掛かりやすい



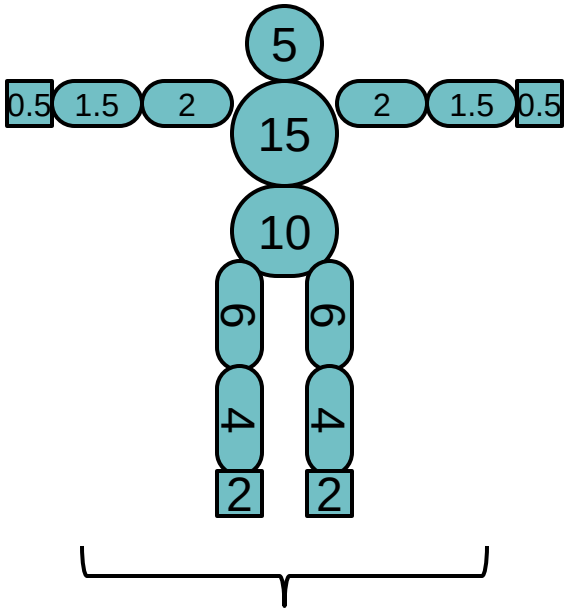
無理に引っ張ると伸びたり、最悪の場合コリジョン抜けを起こす



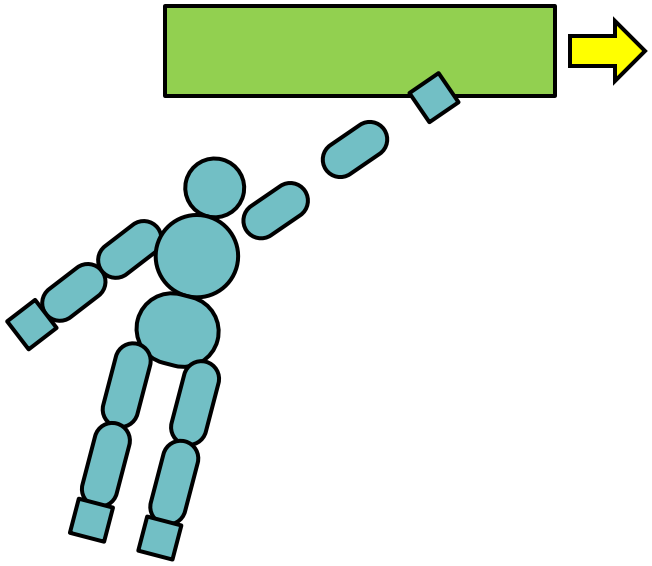
適切な形状を選択すると挙動がスムーズになり引っ掛からない

質量バランスの調整 1

- キーフレームアニメーションで動く物体にラグドールを接続



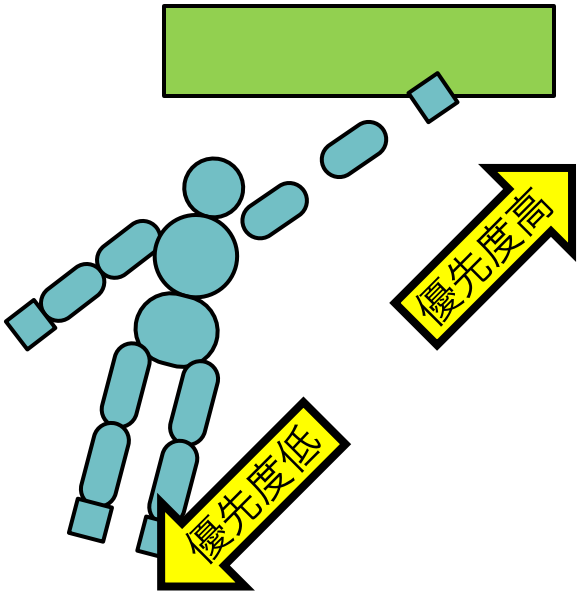
一般的な質量設定



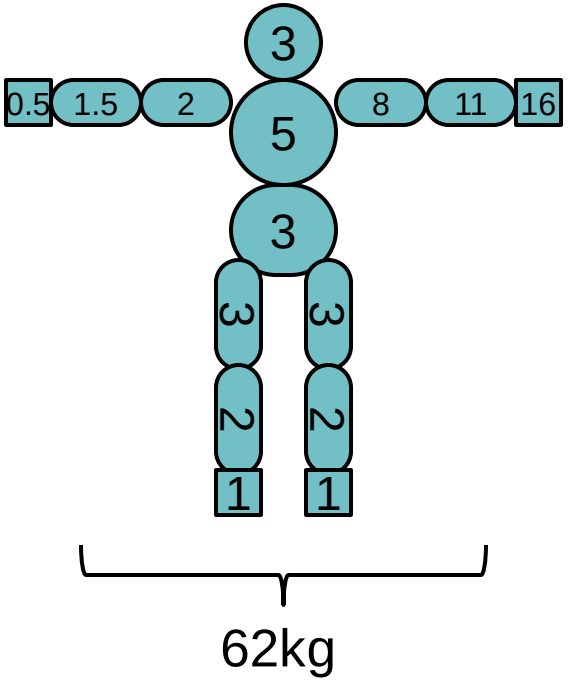
そのままアニメーション駆動の物体にラグドールを接続すると、引っ張られたときに手足が伸びる

質量バランスの調整２

- 優先順位を決め、質量を設定する



優先度の高い剛体に質量を集める

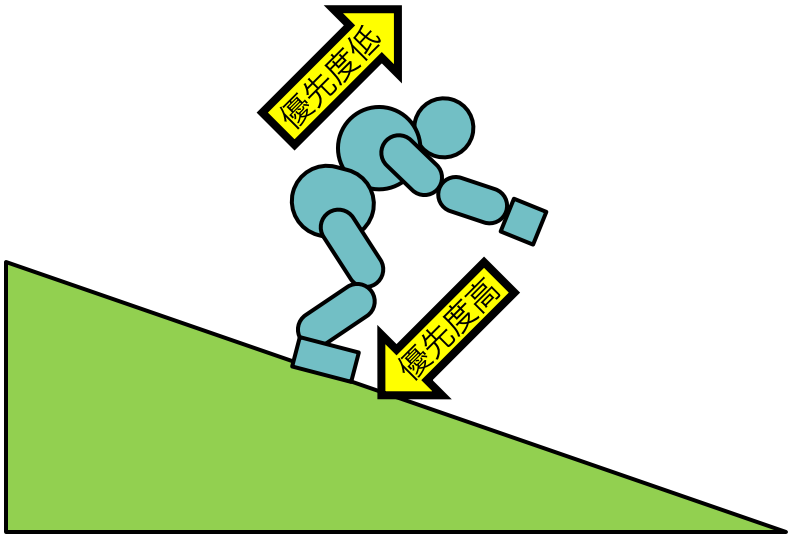


優先度を考慮した質量設定

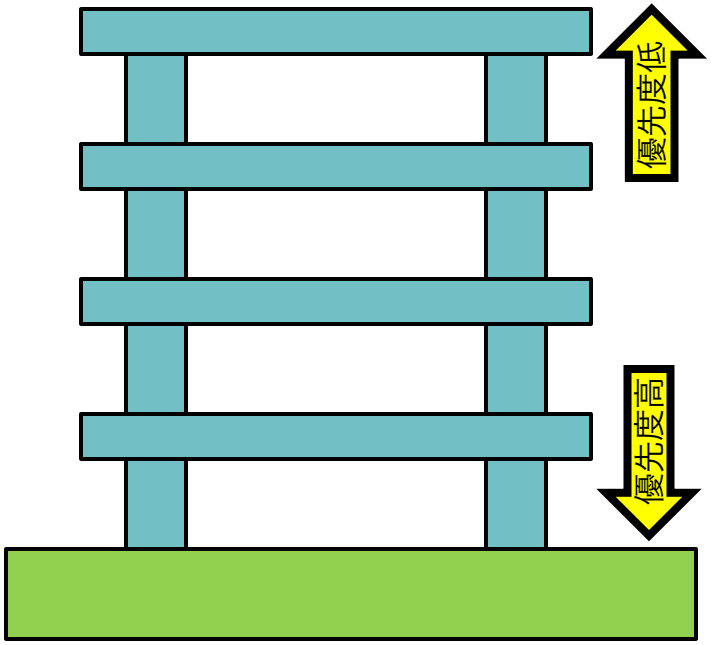
重さが軽く感じられる場合はダンピング（空気摩擦）で調整できる

質量バランスの調整３

- 質量バランスの調整は応用範囲が広い



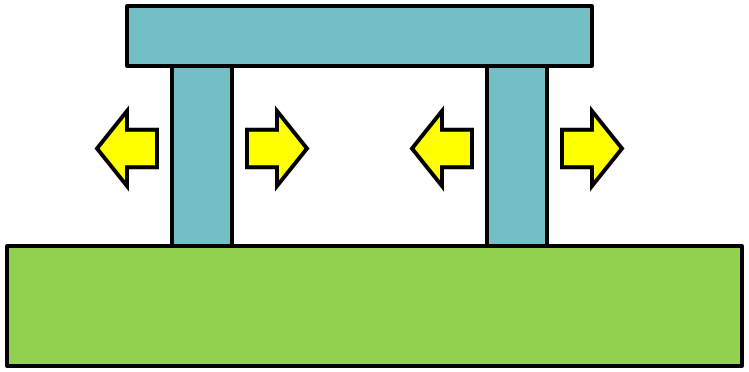
坂の上でラグドールを安定して止めたい場合は足元に質量を集める



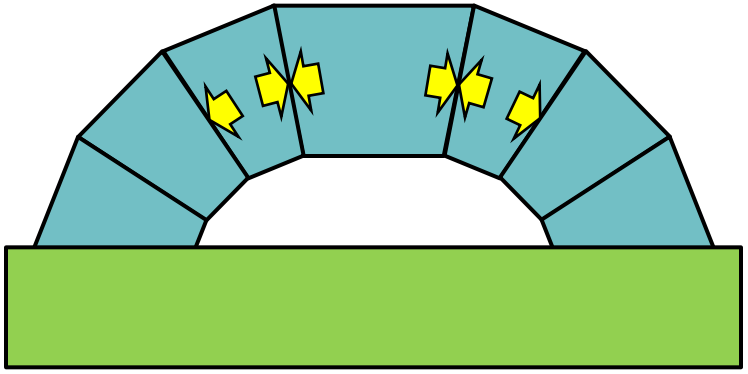
建造物を組み上げる場合も下から上へと質量を軽くしていくと安定する

実在の構造をヒントにする

- アーチ状に構築された建造物は強固

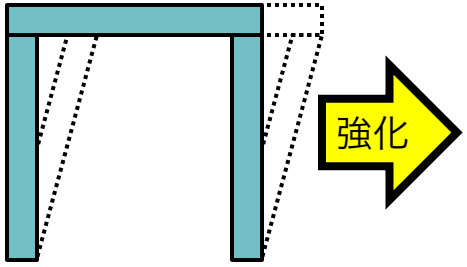


横方向に力が加わると簡単に崩壊

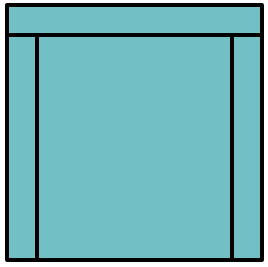


圧縮方向に力がかかるためアーチ構造は強い

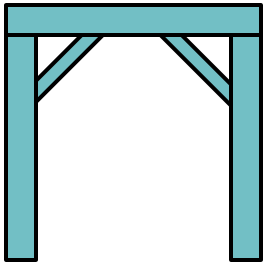
こういった構造上の利点はそのままゲーム物理でも表現できる



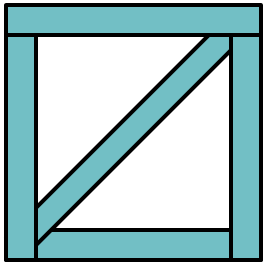
不安定な構造



面を張る



弱い角を補強

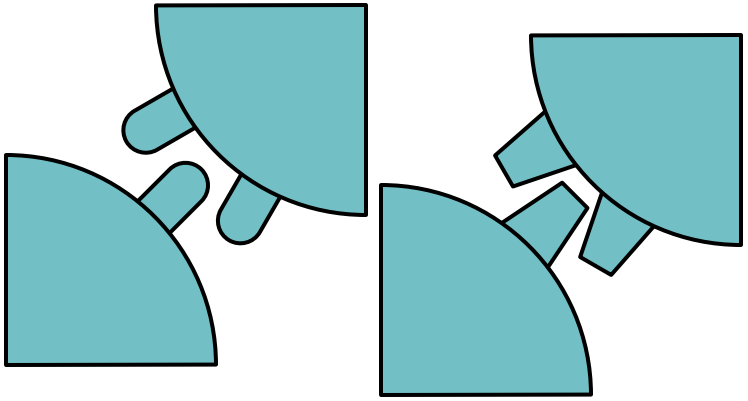


三角形で強化

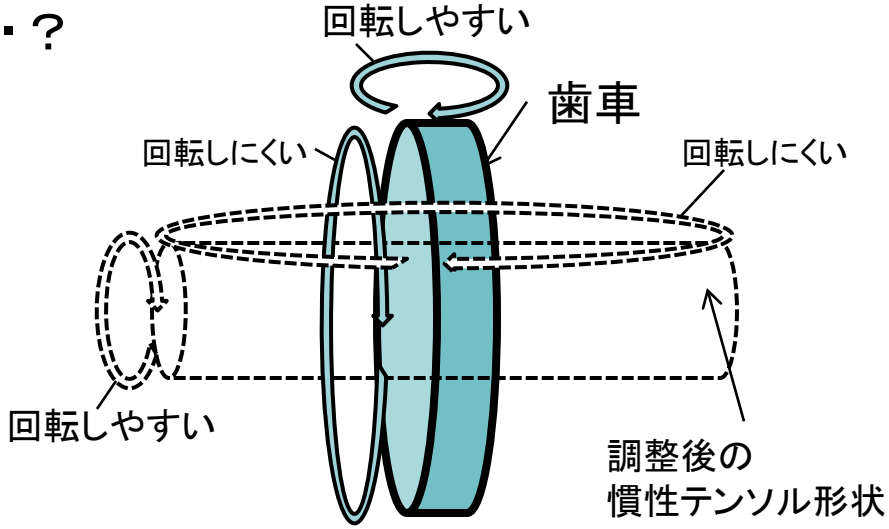
複雑な仕掛け

- 歯車
  - 運動の種類・方向を変えることができる
  - 複雑な仕掛けを作り出せる

滑らかに歯車を連動させるには・・・？



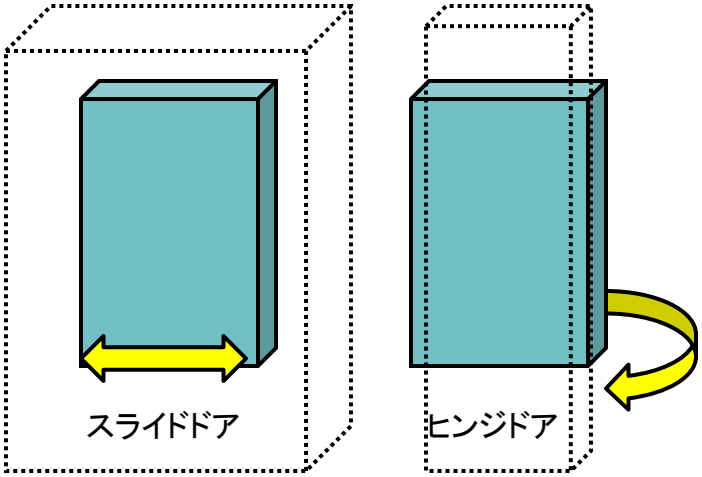
歯を引っ掛かりにくい形状にする



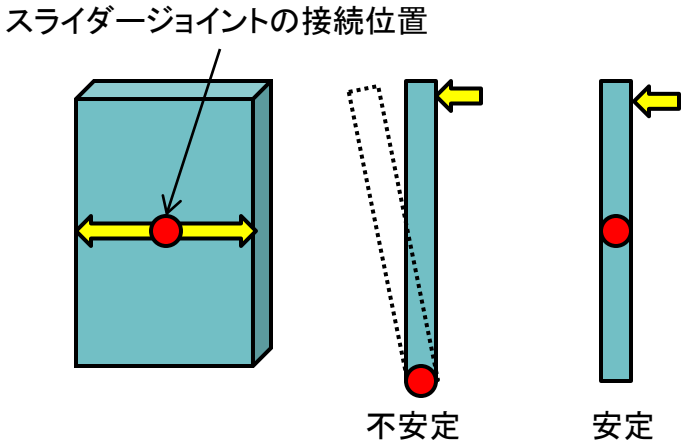
回転する必要のない方向の慣性テンソルを大きめに設定する⇒挙動の安定化

応用：逆回転を防ぐ機構なども可

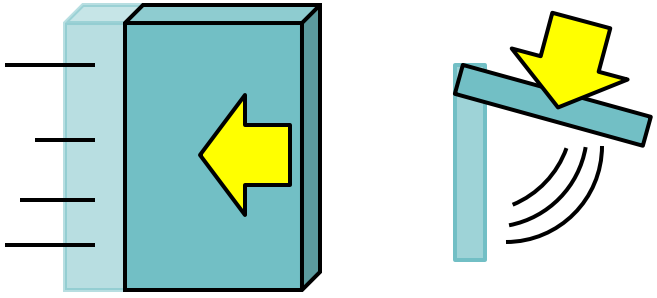
様々な扉を再現



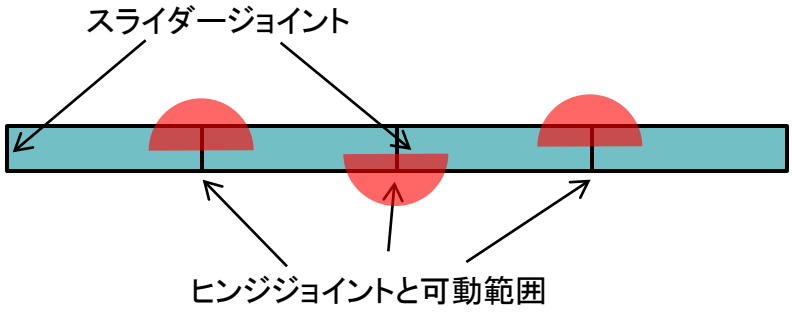
回転する必要のない方向の慣性テンソルを大きめに設定する



スライドドアのジョイント接続点は中心に設定する



ドアが高速に動きすぎないようにダンピングを入れる



折れ戸を上から見た図

複数のドアを連結して折れ戸を再現

まとめ

- 安定化の基本方針
  - 慣性テンソルをコントロール
  - 適切な形状を使う
  - ラグドールの質量バランスを再検討
  - コンストレイントは最も安定する位置で接続する
  - 実在の構造を観察する

TIPS

TIPS 1　全般

- 精度を早めに決めておく
  - 精度を上げるほど挙動が安定するがパフォーマンスは低下
  - 開発の後半で変えると、設定値を全て再調整する羽目になる
  - ソルバー反復回数、タイムステップ

- タイムステップを固定する
  - 内部処理の一貫性を保つことができる
  - デバッグ時に同じ挙動を再現できる

TIPS 2　安定化

- 適切な単位系を使う
  - グラフィックスのスケールをそのまま物理で使うと調整に苦労
  - グラフィックスと物理ではスケールを分ける
- 質量差の大きい剛体同士が干渉しないように気をつける
  - 10倍以内が理想
- 回避できない状況に陥らないように注意する
  - 初期状態で剛体が交差している
  - アニメーション駆動キャラクターと背景に剛体が挟まれた状況
- 剛体の最大速度を制限する
  - 移動量を制限することで、コリジョン抜けを減らせる

TIPS 3　パフォーマンス

- ブロードフェーズをより効果的に効かせる
  - 地形などの広大なオブジェクトは分割したほうが効率が良い

- レイキャストの工夫
  - 多く投射しなくてよい方法を考える
  - まとめて投射する
  - レイを投射する区域を小さくする

- 処理を軽くする工夫
  - 用の済んだ物理オブジェクトは退避させる
  - 強制的に動作を止めてシミュレーションからはずす

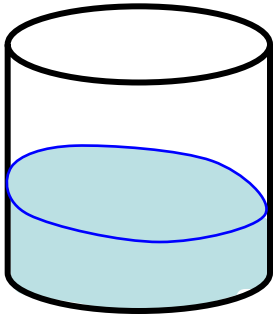
シミュレーションの素を意識する
～　流体シミュレーションを例に　～

流体シミュレーション　（水、煙、炎　etc)

- 流体シミュレーションでは計算の方法はいくつか存在する
  - HeightFieldベース手法
  - Particleベース手法
  - Gridベース手法
- それぞれの計算方法には得意、不得意がある
- レンダリングの仕方によっても、受け手の感覚は変わる

ゲームデザインでは、物理方程式を理解する必要は必ずしもないが、
シミュレーションの素（種類や得意、不得意）を理解する必要あり

今回は水に
話の焦点を絞る



水

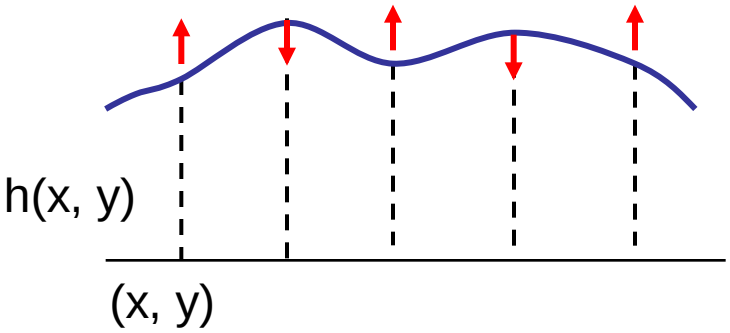
炎

煙

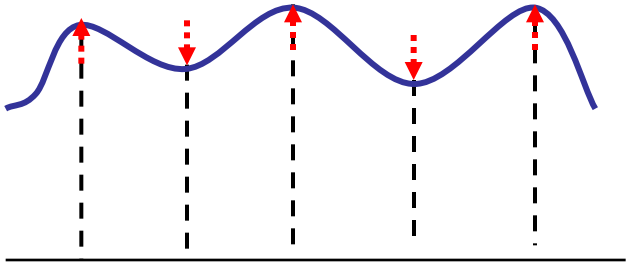
HeightFieldをベースとした手法

プール水面の揺らぎの表現などに用いられる

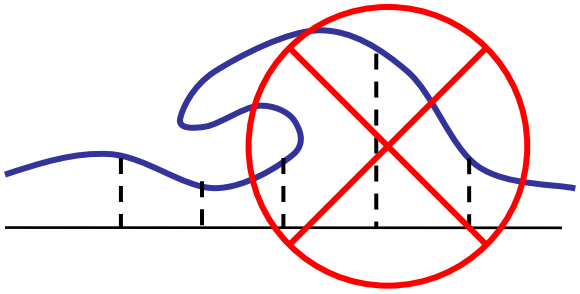
水面高さの変化量



ΔT秒後



水面高さの変化を
追跡する



複雑な水面の変形

＜利点＞

計算コストはかなり軽い

水面生成の計算が必要ない

水量の保存が簡単
→計算による誤差の発生を抑え易い

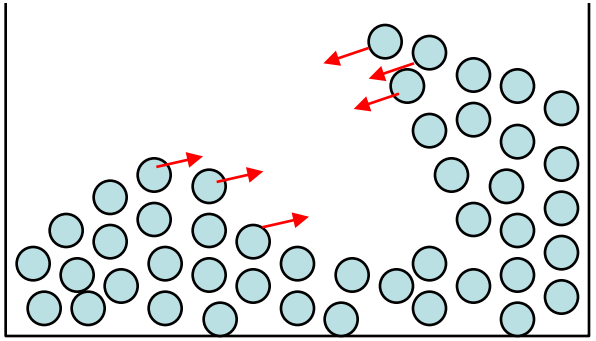
＜弱点＞

複雑な水面の変形を扱えない

Particleをベースとした手法

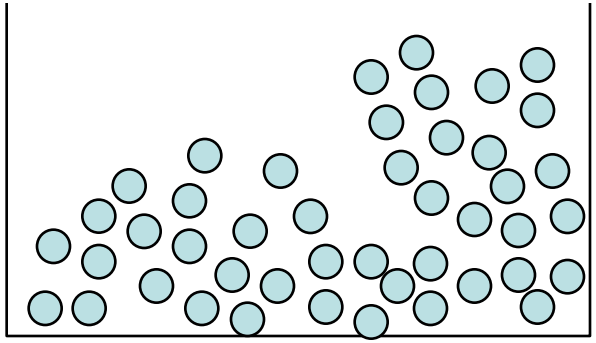
津波やコップ内の水の動きの計算などに用いられる

パーティクルが流体方程式に従って動く

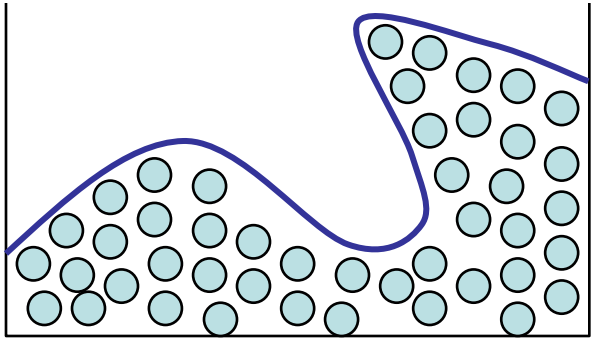


パーティクルの動き
を追跡する

ΔT秒後



Rendering



水としてレンダリングするには
水面の生成計算も必要（水と空気の境界）

＜利点＞

複雑な水面の変形を扱える

水量の保存が簡単
→パーティクルの数を一定に保つ

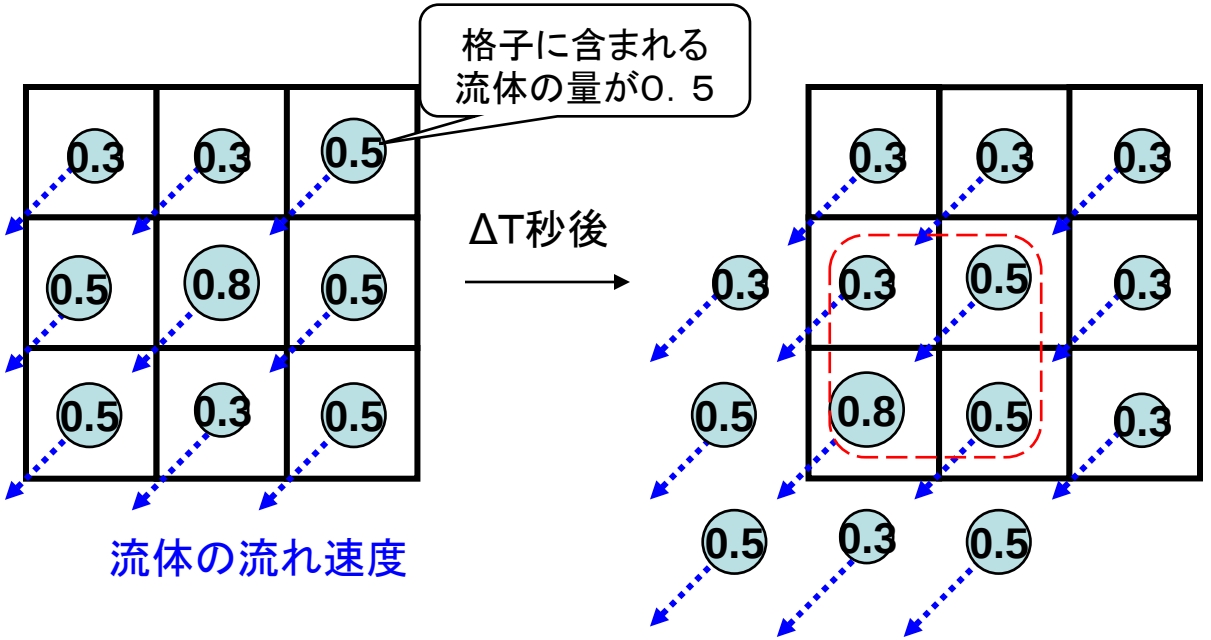
＜弱点＞

水面生成の計算が必要
→シミュレーションに対し、3倍以上の計算コストが生じる場合もある

計算コストは大きい

Gridをベースとした手法

水槽内の水の動きの計算などに用いられる



シミュレーション領域をGridで区分けする

＜利点＞

２次元なら計算コストは比較的軽い

複雑な水面の変形を扱える

＜弱点＞

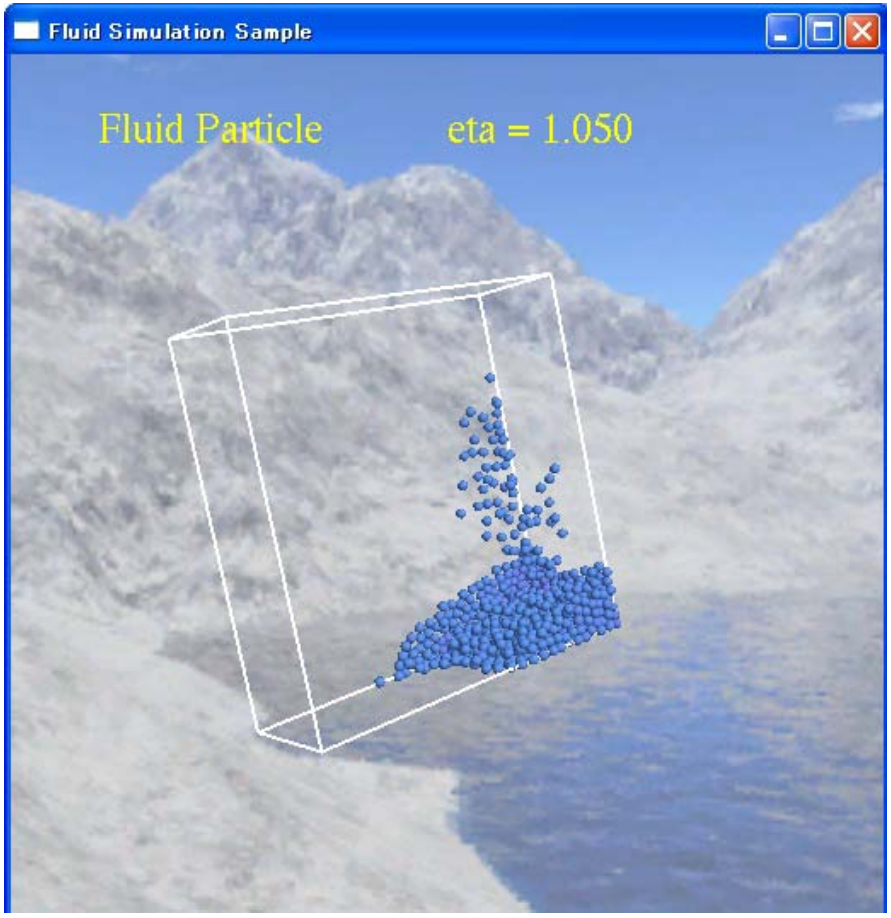
水面生成の計算が必要

水量の保存に工夫が必要
→計算による誤差が生じて水量が減っていきやすい

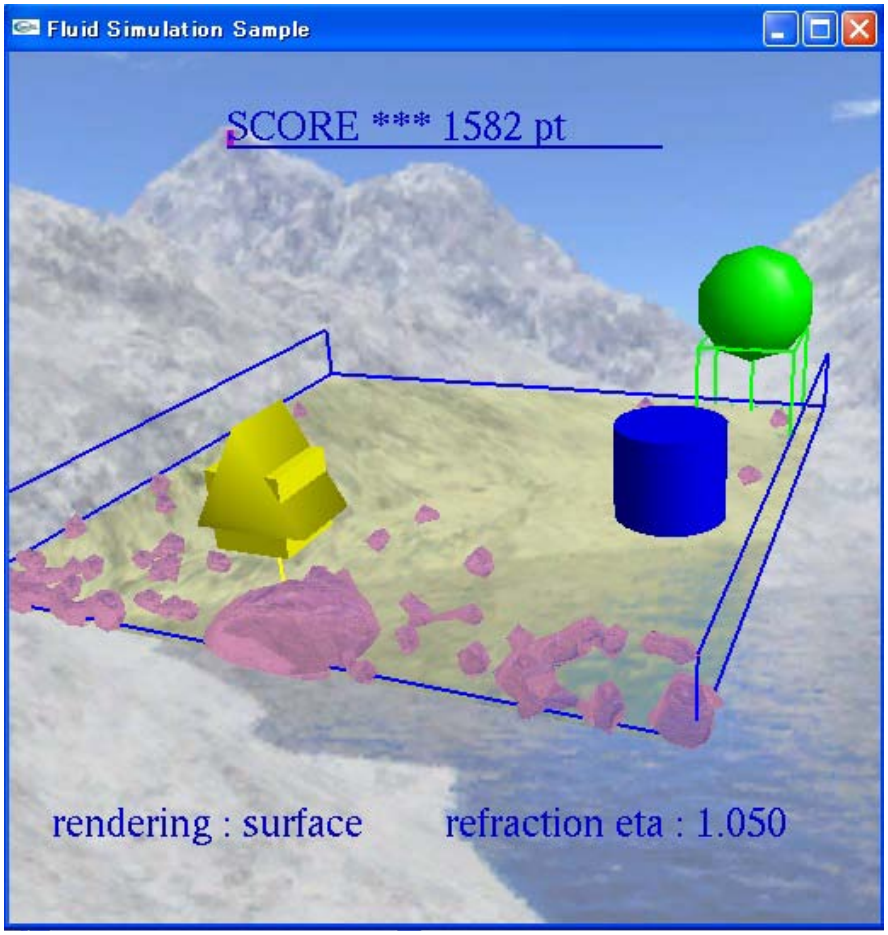
シミュレーション境界の設定自由度に難点

３次元では領域の広さ次第で計算コストがかなり重い

サンプルプログラム



サンプルゲーム


Fluid Simulation Sample
SCORE *** 1582 pt
rendering : surface
refraction eta : 1.050