知的コミュニティ基盤研究センター研究談話会

物理モデルとGPUを用いたリアルタイムCGアプリケーションの開発

図書館情報メディア研究科
情報メディア創成学類
助教　藤澤 誠

2011-10/13

基本的な3DCG生成の流れ

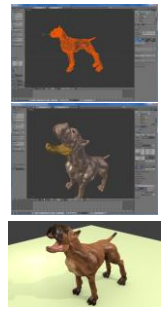
形状(3D)モデルを生成

↓

モデルに動きをつける

↓

結果をレンダリングして画像/動画として出力

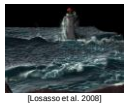


物理法則に基づくコンピュータグラフィックス

- Physics based rendering
  - 光路のトレース
  - レイトレーシング，パストレーシング，フォトンマッピング法など
- Physics based simulation (animation)
  - 物体の動き
  - 剛体，弾性体，流体，人体(骨と筋肉，髪)など


[Narain et al. 2009]

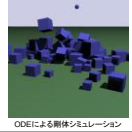

[Goldenthal et al. 2007]

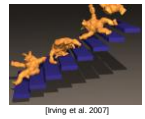

[Losasso et al. 2008]


[Rasmussen et al. 2003]

物理シミュレーション

- 目に見える現象は物理法則に支配されている！
- 物理シミュレーション:物理学の法則に基づきさまざまな物質の挙動をコンピュータで計算する
  1. 支配方程式(ほとんどの場合，非線形の常微分方程式(ODE))が存在
  2. ポリゴン，粒子，グリッドなどで離散化
  3. 数値演算で各フレームの形状，座標などを計算


ODEによる剛体シミュレーション

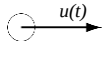

[Irving et al. 2007]


[Enright et al. 2002]


[Fujisawa et al. 2007]

物理シミュレーションの原理

- 簡単な例
  - 支配方程式　$\frac{\partial x}{\partial t} = u(t)$



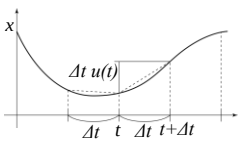
  - グリッドで離散化

微分　$\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{x(t+\Delta t) - x(t)}{\Delta t}$

差分　$\frac{dx}{dt} = \frac{x(t+\Delta t) - x(t)}{\Delta t}$

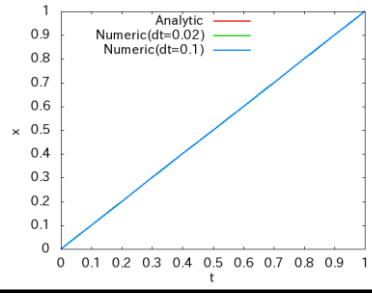
  - 次の[illegible]

$$x(t+\Delta t) = x(t) + \Delta t\, u(t)$$



物理シミュレーションの原理

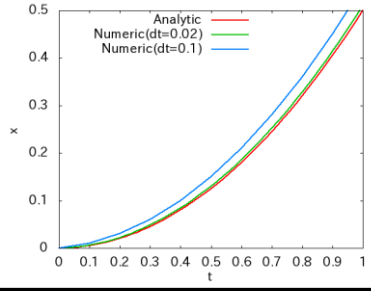
速度 $u(t) = a$　　解析解 $x(t) = at + x_0$
数値解 $x(t+\Delta t) = x(t) + \Delta t\, u(t)$



物理シミュレーションの原理

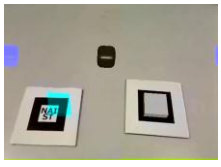
速度 $u(t) = at$　　解析解 $x(t) = \frac{1}{2}at^2 + x_0$

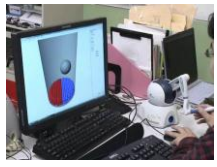
数値解 $x(t + \Delta t) = x(t) + \Delta t\, u(t)$



リアルタイムシミュレーション

- アニメーションの条件である30fpsを実現したシミュレーション
  - インタラクティブアプリケーション
  - AR, ハプティックティックデバイス





アニメーションの原理

仮現運動 : 実際には動いていない物が動いているように見える現象

2 [fps]

20 [fps]

連続する1枚1枚の絵を少しずつ変化させる

アニメーションの原理

- 仮現運動
  - 時差をおいて異なる映像が提示されたときに動きを知覚する現象
    → パラパラ漫画

- ビデオ映像
  - コマ撮りされた画像を連続表示
  - 1秒間に表示する画像枚数(フレームレート)
    : 24枚(映画), 25枚(PAL), 30枚(NTSC)
    ビデオゲームなどでは60fps

リアルタイムシミュレーション

- アニメーションの条件である30fpsを実現したシミュレーション
  - インタラクティブアプリケーション
  - AR, ハプティックティックデバイス
- 物理シミュレーションをリアルタイムで実行する際の問題
  - タイムステップ幅 : 大きいほどよい(最大33ms)
    - 安定した手法は一般的に計算時間がかかる(陰解法など)
    - 安定性と計算時間のトレードオフ
  - 離散化の解像度 : 高いほどよい(ピクセル解像度)
    - 計算時間と見た目の美しさのトレードオフ

高速化

- **計算時間を短縮したい(主に数値計算)**

取り扱う問題
ほとんどの問題は並列化可能
: 線形システム(疎行列), 近傍探索

＋

要求
・リアルタイム計算
・PC上で実行できる

安価な並列計算機はないのか？

GPU

GPUとは

- GPU(Graphics Processing Unit)
  - グラフィックス処理(データ並列)に特化したプロセッサ
  - 一般のPCに搭載されている
  - 並列処理によるCPUを上回る高速演算が可能
    (コア数512，約1.5TFLOPS)
  - 処理内容をプログラミング可能(シェーダ言語，NVIDIA CUDA)

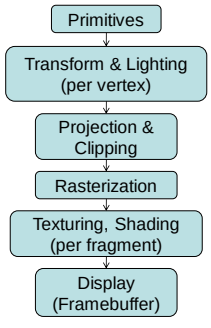
13

GPGPU

- GPUの構成 (NVIDIA Fermi)
  - **SM**(Streaming Multi-Processor)：32個のSP,L1/共有キャッシュ,テクスチャユニット,ジオメトリエンジン
  - **GPC**(Graphics Processing Cluster)：SM×4で構成されるミニGPU．FermiはGPC×4で構成される．
  - ROP(Rendering Output Pipeline)ユニットx8のパーティションx6で48ROP．(ピクセルシェーダからの出力をビデオメモリに書き込む)
  - メモリバス帯域幅は177.4GB/s (GDDR5，384bit)

Programmable Shader

- GPUの歴史
  - **1980s** - 最初の3Dグラフィックス ワークステーション：SGI
    - 2Dグラフィックスハードウェアのみ
  - **1990s** - 最初のPC用GPU(3dfx, Matrox, NVIDIA, ATI)
    - Triangle rasterizationのみ
    - 1999 - ハードウェアT&L対応ビデオカード
    - Fixed Function Pipeline(FFP)
  - **2000s** - プログラマブルGPU (NVIDIA, ATI)
    - Vertex Shader と Fragment Shader

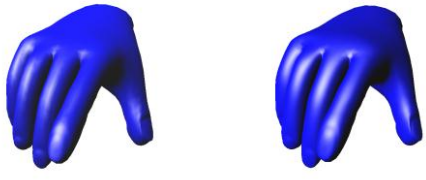


GPU Language

- For graphics
  - Assembly
  - NVIDIA Cg (C for Graphics)
  - HLSL(DirectX High Level Shading Language)
  - GLSL(OpenGL Shading Language)

Example: Phong shader

- Phong illumination model $I_s = k_s(\boldsymbol{R} \cdot \boldsymbol{V})^m l_s$



Gouraud Shading　　Phong Shading

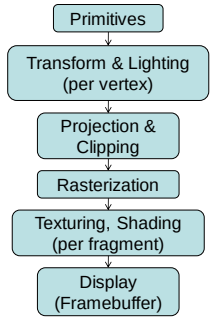
Example: Fresnel shader

- Fresnel reflection and refraction

Reflection　　Phong Shading

Programmable Shader

- GPUの歴史
  - **1980s** - 最初の3Dグラフィックス ワークステーション：SGI
    - 2Dグラフィックスハードウェアのみ
  - **1990s** - 最初のPC用GPU(3dfx, Matrox, NVIDIA, ATI)
    - Triangle rasterizationのみ
    - 1999 - ハードウェアT&L対応ビデオカード
    - Fixed Function Pipeline(FFP)
  - **2000s** - プログラマブルGPU (NVIDIA, ATI)
    - Vertex Shader と Fragment Shader
    - General Purpose GPU (GPGPU)



GPGPU

シェーダー言語を用いたGPGPUの例

- リアルタイムビデオ安定化
  ビデオ映像のみを入力として，ぶれを除去した動画を生成

使用動画 [pixel]	CPU [sec/frame]	GPU [sec/frame]
200×200	1.32	0.012
500×500	8.89	0.038
700×700	17.1	0.081
1000×1000	36.6	0.141

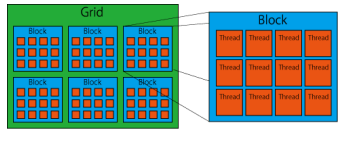
CPU : Core2Duo 3.00GHz, GPU : GeForce8800GTX

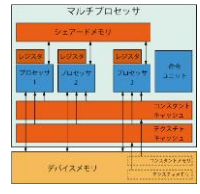
GPU Language

- For graphics
  - Assembly
  - NVIDIA Cg (C for Graphics)
  - HLSL(DirectX High Level Shading Language)
  - GLSL(OpenGL Shading Language)
- For general purpose
  - BrookGPU
  - CUDA (C for CUDA)
  - OpenCL

CUDA

- CUDA(Compute Unified Device Architecture)の特徴
  - 統合シェーダ
  - 数千~数万のスレッドの同時実行
  - スレッド，ブロック，グリッド，カーネル
  - 高速なシェアードメモリの利用
  - 基本的には単精度(倍精度もあり)



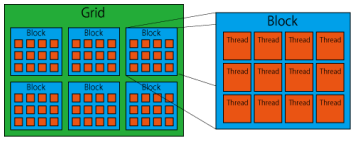


GPGPU

- 多数の並列スレッドによるメモリレイテンシの隠蔽

  演算の実行単位：数サイクル
  グローバルメモリ転送：数百サイクル

あるスレッドグループ(warp)がデータ転送している間に，他のwarpが実行される．

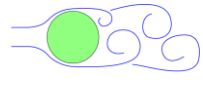



```
max threads / block : 1024
max size of each dim. of block : (1024, 1024, 64)
max size of each dim. of grid : (65535, 65535, 65535)
```

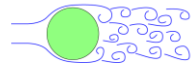
粒子法とウェーブレット解析によるリアルタイム乱流シミュレーション

乱流とは？

挙動の予測が困難な非常に乱れた流れ
⇔層流



乱流はなぜ解析が難しい？

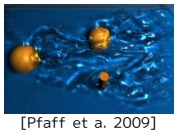


- 支配方程式を完璧に解けば再現できる
  - 膨大な量の計算時間が必要
- 平均流と細かな渦に流れを分割
- 渦エネルギーに基づく方法

関連研究

- **High order advection schemes**
  BFECC [Kim et al. 2005], MacCormack [Selle et al. 2008], CIP [Kim et al. 2008], adaptive grid [Losasso et al. 2004]
- **Vortex particle method**
  - Vortex particle method [Selle et al. 2005]
  - 壁面乱流境界層 [Pfaff et a. 2009]
  - WCSPHと固体周りのグリッド [Zhu et al. 2010]


[Selle et al. 2005]


[Pfaff et a. 2009]


[Zhu et al. 2010]

関連研究

- **Procedural turbulence synthesis**
  - Energy transport model [Pfaff et al. 2010]
  - Random Forcing [Zhao et al. 2010]
  - 疎／密グリッドの組み合わせ [Chang et al. 2010]
  - ウェーブレット解析 [Kim et al. 2008]

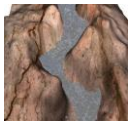

[Pfaff et al. 2010]

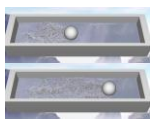

[Kim et al. 2008]

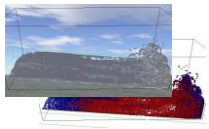
粒子法とウェーブレット解析によるリアルタイム乱流シミュレーション

粒子法を用いた**リアルタイム流体シミュレーション**に**乱流**を付加することでよりリアリティの高い映像を生成する手法

- ウェーブレット解析による乱流生成域の特定
- 複数の周波数を含むノイズ関数による乱流渦の生成
- サブパーティクル分割による非常に細かな渦の考慮
- **GPUを用いた超並列計算**



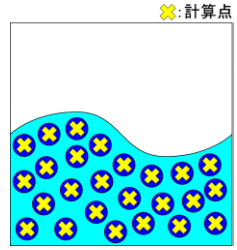




シミュレーション空間の離散化

離散化手法

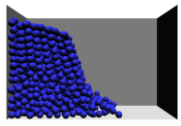
- オイラー的手法
  空間を格子状に離散化
  → 一般に高計算コストだが，滑らかな表面が得られる
- ラグランジュ的手法(粒子法)
  粒子で流体を離散化
  → 一般に低計算コストだが，液体表面が凸凹



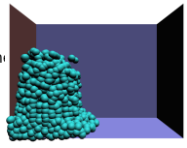
30

粒子法

- Moving Particle Semi-implicit (MPS)
  - 粒子数密度の導入により非圧縮性を厳密に表現できる
  - 圧力のポアソン方程式を解く必要があり，計算コストが高い
  - 計算例 ： 計算点6,000 - 0.94 sec/frame



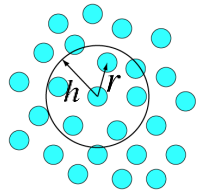
- Smoothed Particle Hydrodynamics (SPH)
  - 単純な関数の重ね合わせで計算できるため，計算コストが低い
  - 非圧縮性を厳密に定めているわけではない
  - 計算速度 ： 計算点6,000 - 0.056 sec/fram



SPHの理論

- SPHの離散式

$$A(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h)$$



h ： 有効半径

W: カーネル関数

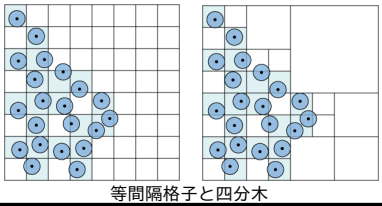
$0 \le r \le h$ のときのみ値を持つ

r が大きくなるにつれて W は小さくなる

近傍探索の高速化

空間分割法による近傍探索の高速化

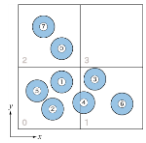
1. 物体の登録
   i. 空間を分割
   ii. 各物体が所属する分割領域を計算
2. 近傍探索
   i. 探索中心座標が所属する領域を計算
   ii. 自身と隣接する領域の物体をリストアップ
   iii. リストアップされた物体との距離を算出



等間隔格子と四分木

近傍探索の高速化

空間分割法のGPU実装



1. 全パーティクルのハッシュ値を計算
   hash = $i+j*n_x$

ハッシュ値	2	0	0	1	1	0	1	2
粒子番号	0	1	2	3	4	5	6	7

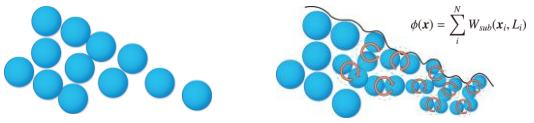
2. ハッシュ値をキーとしてソート
   radix sort を使用[Satish et al. IPDPS' 09]

ハッシュ値	0	0	0	1	1	1	2	2
粒子番号	1	2	5	3	4	6	0	7

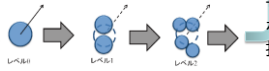
3. ハッシュ値が連続する領域の始点，終点を格納

サブパーティクルスケール乱流

粒子法による平均流の計算
+サブパーティクル分割による乱流渦



各パーティクルの乱流エネルギーeを算出



サブパーティクルを使って表面を抽出

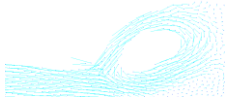
乱流エネルギーに基づき分割

$$e_{cri} = \hat{e}(\frac{1}{2r_L}, \boldsymbol{x})(2^{-\frac{5}{9}})^{L_i}$$

Simulation of Turbulent Flow

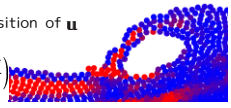
1. SPH法でベースとなる速度場を計算

$$\nabla \cdot \mathbf{u} = 0,$$
$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = \nu \nabla^2 \mathbf{u} - \frac{1}{\rho}\nabla p + \mathbf{f}$$



2. ウェーブレット解析により乱流エネルギーを算出

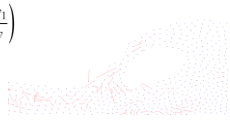
$$\hat{e}(k) = \frac{1}{2}|\hat{\mathbf{u}}(k)|^2 \quad \hat{\mathbf{u}}: \text{wavelet decomposition of } \mathbf{u}$$
$$\hat{u}_i = \frac{1}{\sqrt{5}\psi_{sum}} \sum_j u_j \psi\left(\frac{x_i - x_j}{s}, \frac{y_i - y_j}{s}, \frac{z_i - z_j}{s}\right)$$



Simulation of Turbulent Flow

3. 多解像度の非圧縮渦場(乱流速度場)を生成

$$\boldsymbol{w}(\boldsymbol{x}) = \left(\frac{\partial w_3}{\partial y} - \frac{\partial w_2}{\partial z}, \frac{\partial w_1}{\partial z} - \frac{\partial w_3}{\partial x}, \frac{\partial w_2}{\partial x} - \frac{\partial w_1}{\partial y}\right)$$
$$\boldsymbol{y}(\boldsymbol{x}) = \sum_{i=i_{min}}^{i_{max}} \boldsymbol{w}(2^i \boldsymbol{x}) 2^{-\frac{5}{6}(i - i_{min})}$$



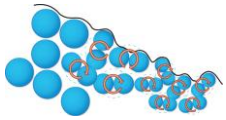
4. 乱流の影響を外力として追加し，粒子の速度・位置を更新

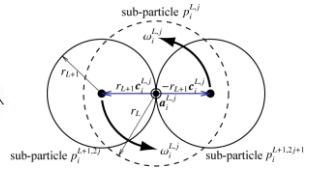
$$\boldsymbol{f}_i^{turb} = A \frac{\rho_i}{\Delta t} \hat{e}_i(k) \boldsymbol{y}(\boldsymbol{x}_i)$$

Sub-Particle-Scale Turbulence

5. サブパーティクルの速度場と位置を更新

GPUでの並列計算のために，全てのサブパーティクルは最大レベルまで分割され，その全ての速度と位置が更新される





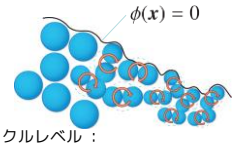
Sub-Particle-Scale Turbulence

6. 乱流エネルギーに基づき液体の表面を抽出

流体の表面を表すレベルセット関数：

$$\phi(\boldsymbol{x}) = \sum_i^N W_{sub}(\boldsymbol{x}_i, L_i)$$

W_{sub}：カーネル関数



乱流エネルギーに基づくサブパーティクルレベル：

$$e_{cri} = \hat{e}(\frac{1}{2r_L}, \boldsymbol{x})(2^{-\frac{5}{9}})^{L_i}$$

結果

パーティクル数：20,000 - 40,000

処理時間：6-10 fps on Intel Core i7 2.93GHz
and GeForce GTX580 (SP数512)

レンダリング：Marching cubes method
(about 80,000 triangles), GLSL

結果

まとめ

- 物理モデルとGPUを用いたリアルタイムCGアプリケーション
  - 物理シミュレーション
  - 数値計算をGPUを用いて並列化
  - 乱流シミュレーションをリアルタイムに近い速度で実現
- 今後の発展
  - シミュレーションの高速化，高精度化
  - AR(拡張現実感)インタフェースの利用
  - デザインや解析への応用