A Multigrid Fluid Pressure Solver Handling Separating Solid Boundary Conditions

Nuttapong Chentanez

Matthias Müller

Monday, August 15, 2011
Main Contributions

• Multigrid method for solving the weighted Poisson equations
  – From the variational framework for fluid simulation in Batty et al. 07 (BBB07), Batty and Bridson 08

• Modifications to solve LCP
  – To enforce separating solid boundary condition
Example

\[(u - u_s) \cdot n_s = 0\]

\[(u - u_s) \cdot n_s \geq 0\]
Example

\[(u - u_s) \cdot n_s = 0\]

No wall separating boundary condition

\[(u - u_s) \cdot n_s \geq 0\]

With wall separating boundary condition (Node Base)
Background

Foster and Fedkiw 01

Houston et al. 03

Rasmussen et al. 04

BBB07
Method

- Inviscid Incompressible Euler Equations

\[
\frac{\partial \mathbf{u}}{\partial t} = - (\mathbf{u} \cdot \nabla) \mathbf{u} + \frac{f}{\rho} - \frac{\nabla p}{\rho}
\]

- Subject to \( \nabla \cdot \mathbf{u} = 0 \)

- Inside region \( \phi < 0 \),

\[
\frac{\partial \phi}{\partial t} = - \mathbf{u} \cdot \nabla \phi
\]
Method

- Discretize to staggered grid as in BBB07

- Cell center
  - Pressure $p$, level set $\phi$, solid fraction $V$

- Face center
  - Components of velocity $u = [u, v, w]^T$
  - Face center solid fraction $V_u, V_v, V_w$
Method

- Time integration
  - Standard grid based sim

- Novelty in pressure projection

- Velocity Extrapolation
- Level Set Reinitialization
- Velocity and Level Set Advection
- Pressure Projection

Monday, August 15, 2011
Pressure Projection

- Let $u^*$ be the velocity field before pressure projection.
- Then

$$u^{n+1} = u^* - \frac{\Delta t}{\rho} \nabla p$$
Pressure Projection

- Let $u^*$ be the velocity field before pressure projection
- Then
  $$u^{n+1} = u^* - \frac{\Delta t}{\rho} \nabla p$$
- Kinematic energy of the liquid
  - Integrated over the liquid domain
    $$\frac{1}{2} \int u^{n+1} \cdot u^{n+1} dV$$
  - Take solid fraction and free surface location into account
Pressure Projection

- Let $u^*$ be the velocity field before pressure projection
- Then
  \[ u^{n+1} = u^* - \frac{\Delta t}{\rho} \nabla p \]
- Kinematic energy of the liquid
  - Integrated over the liquid domain
  \[ \frac{1}{2} \int u^{n+1} \cdot u^{n+1} dV \]
  - Take solid fraction and free surface location into account
- Pressure $p$ found by minimizing the kinetic energy, BBB07
  - Automatically yields divergence free velocity field
Separating Solid Boundary Condition

• Commonly used fluid solver enforces

\[(u - u_s) \cdot n_s = 0\]

  – On the solid boundary
  – Neumann boundary condition
  – Yields linear system that must be solved for \(p\)

• For a static ceiling with \(u_s = 0\),
  – Liquid sticks unnaturally
Separating Solid Boundary Condition

• BBB07 propose to enforce
  \[(u - u_s) \cdot n_s \geq 0\]
  - If liquid separates from the solid then it becomes a free surface \(p = 0\)
  - Otherwise, \(p > 0\) disallowing suction

• Hence
  \[0 \leq p \perp (u - u_s) \cdot n_s\]
  - Linear Complementarity Problem (LCP)
  - Only need to enforce \(p \geq 0\), BBB07
Multigrid LCP solver

- We propose to use a multigrid solver for this
- Important observation
  - Don’t need to enforce $p \geq 0$ exactly on solid surface
  - Just need to enforce at solid nodes next to liquid

With wall separating boundary condition (Edge Base)  With wall separating boundary condition (Node Base)
Multigrid LCP solver

- We propose to use a multigrid solver for this.
- Important observation
  - Don’t need to enforce $p \geq 0$ exactly on solid surface
  - Just need to enforce at solid nodes next to liquid
Multigrid LCP solver

- We propose to use a multigrid solver for this
- Important observation
  - Don’t need to enforce $p \geq 0$ exactly on solid surface
  - Just need to enforce at solid nodes next to liquid
Multigrid LCP Solver

• Adapt from MG Solver in Chentanez & Müller 11

• Idea
  – Replace Gauss Seidel with Projected Gauss Seidel

\[ Ap = b \]
Multigrid LCP Solver

- Adapt from MG Solver in Chentanez & Müller 11

- Idea
  - Replace Gauss Seidel with Projected Gauss Seidel

\[
A_{i,j,k}^i p_{i,j,k} + A_{i,j,k}^{i+1} p_{i+1,j,k} + A_{i,j,k}^{i-1} p_{i-1,j,k} + \ldots = b_{i,j,k}
\]
Multigrid LCP Solver

- Adapt from MG Solver in Chentanez & Müller 11
- Idea
  - Replace Gauss Seidel with Projected Gauss Seidel

\[
p_{i,j,k} = \frac{1}{A_{i,j,k}} \left( b_{i,j,k} - A_{i,j,k}^{i+1,j,k} p_{i+1,j,k} - A_{i,j,k}^{i+1,j,k} p_{i+1,j,k} - \ldots \right)
\]
Multigrid LCP Solver

- Adapt from MG Solver in Chentanez & Müller 11

Idea
- Replace Gauss Seidel with Projected Gauss Seidel

\[ p_{i,j,k} = \frac{1}{A_{i,j,k}} (b_{i,j,k} - A_{i,j,k}^{i+1,j,k} p_{i+1,j,k} - A_{i,j,k}^{i+1,j,k} p_{i+1,j,k} - \ldots) \]

- Gauss Seidel applies the above equation iteratively
Multigrid LCP Solver

• Adapt from MG Solver in Chentanez & Müller 11

• Idea
  – Replace Gauss Seidel with Projected Gauss Seidel

\[
p_{i,j,k} = \max\left(p_{\min \ i,j,k}, \frac{1}{A_{i,j,k}} (b_{i,j,k} - A_{i+1,j,k} p_{i+1,j,k} - \ldots)\right)
\]

• Projected Gauss Seidel applies the above equation iteratively, where

\[
p_{\min \ i,j,k} = \begin{cases} 0 & \text{if } i,j,k \text{ is inside a solid} \\ -\infty & \text{otherwise} \end{cases}
\]
Multigrid LCP Solver

- Build hierarchy of grids
  - 8-to-1 down sampling (in 3D)
  - Down sampling $\phi$ specially
- Preserving air bubbles in a few finest levels, Chentanez & Müller 11
Algorithm 2 V_Cycle(m)
1: if \( m == 1 \) then
2:   Solve the linear system, \( A^1 p^1 = b^1 \)
3: else
4:   for \( i = 1 \) to num_Pre_Sweep do
5:     Smooth(\( p^m \)) and enforce \( p^m \text{min} \) (PRBGS)
6:   end for
7:   \( r^m = b^1 - A p^m \)
8:   \( b^{m-1} = \text{Restrict}(r^m) \)
9:   \( p^{m-1} = 0 \)
10: if \( m > M - S \) then
11:   \( p^{m-1}_\text{min} = \text{DownsampleSubtract}(p^{m-1}_\text{min}, p^m) \)
12: else
13:   \( p^{m}_\text{min} = -\infty \)
14: end if
15: V_Cycle(m - 1)
16: \( p^m = p^m + \text{Prolong}(p^{m-1}) \)
17: for \( i = 1 \) to num_Post_Sweep do
18:   Smooth (\( p^m \)) and enforce \( p^m \text{min} \) (PRBGS)
19: end for
20: end if
Algorithm 2 V_Cycle(m)

1: if \( m = 1 \) then
2:     \[
3:     \text{Solve the linear system, } A^1 p^1 = b^1
4:     \]
5: else
6:     for \( i = 1 \) to num_Pre_Sweep do
7:         Smooth(\( p^m \)) and enforce \( p^m_{\text{min}} \) (PRBGS)
8:     end for
9:     \( r^m = b^l - A p^m \)
10:    \( b^{m-1} = \text{Restrict}(r^m) \)
11:    \( p^{m-1} = 0 \)
12:    if \( m > M - S \) then
13:        \( p^m_{\text{min}} = \text{DownsampleSubtract}(p^m_{\text{min}}, p^m) \)
14:    else
15:        \( p^m_{\text{min}} = -\infty \)
16:    end if
17: V_Cycle(m - 1)
18: \( p^m = p^m + \text{Prolong}(p^{m-1}) \)
19:     for \( i = 1 \) to num_Post_Sweep do
20:         Smooth(\( p^m \)) and enforce \( p^m_{\text{min}} \) (PRBGS)
21:     end for
22: end if
Multigrid LCP Solver

**Algorithm 2 V_Cycle(m)**

1: if $m == 1$ then
2:   Solve the linear system, $A^1 p^1 = b^1$
3: else
4:   for $i = 1$ to num_Pre_Sweep do
5:     Smooth($p^m$) and enforce $p^m_{\min}$ (PRBGS)
6:   end for
7:   $r^m = b^1 - A p^m$
8:   $b^{m-1} = \text{Restrict}(r^m)$
9:   $p^{m-1} = 0$
10:  if $m > M - S$ then
11:     $p^{m-1}_{\min} = \text{DownsampleSubtract}(p^m_{\min}, p^m)$
12:  else
13:     $p^m_{\min} = -\infty$
14:  end if
15:  V_Cycle($m - 1$)
16:  $p^m = p^m + \text{Prolong}(p^{m-1})$
17:  for $i = 1$ to num_Post_Sweep do
18:     Smooth ($p^m$) and enforce $p^m_{\min}$ (PRBGS)
19:  end for
20: end if

**Pre-smoothing**

$p_{i,j,k} = \max(p_{\min i,j,k}, \frac{1}{A_{i,j,k}} (b_{i,j,k} - A_{i+1,j,k}^i p_{i+1,j,k} - \ldots))$

Projected Red Black Gauss Seidel
Multigrid LCP Solver

Algorithm 2 V_Cycle(m)
1: if \( m == 1 \) then
2: \[ \text{Solve the linear system, } A^1 p^1 = b^1 \]
3: else
4: for \( i = 1 \) to \( \text{num\_Pre\_Sweep} \) do
5: \[ \text{Smooth}\left(p^m\right) \text{ and enforce } p^m_{\text{min}} \text{ (PRBGS)} \]
6: end for
7: \( r^m = b^l - A p^m \)
8: \( b^{m-1} = \text{Restrict}(r^m) \)
9: \( p^{m-1} = 0 \)
10: if \( m > M - S \) then
11: \( p^{m-1}_{\text{min}} = \text{DownsampleSubtract}(p^{m\_\text{min}}, p^m) \)
12: else
13: \( p^{m\_\text{min}} = -\infty \)
14: end if
15: \( \text{V\_Cycle}(m - 1) \)
16: \( p^m = p^m + \text{Prolong}(p^{m-1}) \)
17: for \( i = 1 \) to \( \text{num\_Post\_Sweep} \) do
18: \[ \text{Smooth}\left(p^m\right) \text{ and enforce } p^m_{\text{min}} \text{ (PRBGS)} \]
19: end for
20: end if

Pre-smoothing

\[ p_{i,j,k} = \max(p_{\text{min}\_i,j,k}, \frac{1}{A_{i,j,k}^{i+1,j,k}} (b_{i,j,k} - A_{i,j,k}^{i+1,j,k} p_{i+1,j,k} - \ldots)) \]

Projected Red Black Gauss Seidel
Algorithm 2 V_Cycle(m)
1: if $m == 1$ then
2:     Solve the linear system, $A^1 p^1 = b^1$
3: else
4:     for $i = 1$ to num_Pre_Sweep do
5:         Smooth($p^m$) and enforce $p^m_{\min}$ (PRBGS)
6:     end for
7:     $r^m = b^l - A p^m$
8:     $b^{m-1} = \text{Restrict}(r^m)$
9:     $p^{m-1} = 0$
10:    if $m > M - S$ then
11:        $p_{\min}^{m-1} = \text{DownsampleSubtract}(p_{\min}^m, p^m)$
12:    else
13:        $p_{\min}^m = -\infty$
14:    end if
15:    V_Cycle($m - 1$)
16:    $p^m = p^m + \text{Prolong}(p^{m-1})$
17:    for $i = 1$ to num_Post_Sweep do
18:        Smooth ($p^m$) and enforce $p^m_{\min}$ (PRBGS)
19:    end for
20: end if

Compute Residual
Multigrid LCP Solver

Algorithm 2 V_Cycle(m)

1: if \( m == 1 \) then
2:   Solve the linear system, \( \mathbf{A}^1 p^1 = b^1 \)
3: else
4:   for \( i = 1 \) to num_Pre_Sweep do
5:     Smooth\((p^m)\) and enforce \( p_{min}^m \) (PRBGS)
6:   end for
7:   \( r^m = b^1 - \mathbf{A} p^m \)
8:   \( b^{m-1} = \text{Restrict}(r^m) \)
9:   \( p^{m-1} = 0 \)
10: if \( m > M - S \) then
11:   \( p_{min}^{m-1} = \text{DownsampleSubtract}(p_{min}^m, p^m) \)
12: else
13:   \( p_{min}^m = -\infty \)
14: end if
15: V_Cycle\((m - 1)\)
16: \( p^m = p^m + \text{Prolong}(p^{m-1}) \)
17: for \( i = 1 \) to num_Post_Sweep do
18:   Smooth \((p^m)\) and enforce \( p_{min}^m \) (PRBGS)
19: end for
20: end if

Restrict
- Down sampling
- Tri-linear interpolation
Multigrid LCP Solver

**Algorithm 2 V_Cycle(m)**

1. if $m == 1$ then
2. Solve the linear system, $A^1 p^1 = b^1$
3. else
4. for $i = 1$ to num_Pre_Sweep do
5. Smooth($p^m$) and enforce $p^m_{\min}$ (PRBGS)
6. end for
7. $r^m = b^l - A p^m$
8. $b^{m-1} = \text{Restrict}(r^m)$
9. $p^{m-1} = 0$
10. if $m > M - S$ then
11. $p^{m-1}_{\min} = \text{DownsampleSubtract}(p^{m-1}_{\min}, p^m)$
12. else
13. $p^m_{\min} = -\infty$
14. end if
15. V_Cycle($m - 1$)
16. $p^m = p^m + \text{Prolong}(p^{m-1})$
17. for $i = 1$ to num_Post_Sweep do
18. Smooth ($p^m$) and enforce $p^m_{\min}$ (PRBGS)
19. end for
20. end if

Want to make sure

$$p^m_{i,j,k} + \text{Prolong}(p^{m-1}_{i,j,k}) \geq p^m_{\min i,j,k}$$

Guaranteed by

$$p^{m-1}_{\min i,j,k} = \max_{a,b,c \in \{0,1\}} (p^{m}_{\min 2i+a,2j+b,2k+c} - p^{m}_{2i+a,2j+b,2k+c})$$

Only needed for the finest $S$ levels.
Algorithm 2 V_Cycle(m)

1: if \( m == 1 \) then
2:     Solve the linear system, \( A^1 p^1 = b^1 \)
3: else
4:     for \( i = 1 \) to num_Pre_Sweep do
5:         Smooth\( (p^m) \) and enforce \( p^m_{\text{min}} \) (PRBGS)
6:     end for
7:     \( r^m = b^l - A p^m \)
8:     \( b^{m-1} = \text{Restrict}(r^m) \)
9:     \( p^{m-1} = 0 \)
10: if \( m > M - S \) then
11:     \( p^m_{\text{min}} = \text{DownsampleSubtract}(p^m_{\text{min}}, p^m) \)
12: else
13:     \( p^m_{\text{min}} = -\infty \)
14: end if
15: V_Cycle(m - 1)
16: \( p^m = p^m + \text{Prolong}(p^{m-1}) \)
17: for \( i = 1 \) to num_Post_Sweep do
18:     Smooth\( (p^m) \) and enforce \( p^m_{\text{min}} \) (PRBGS)
19: end for
20: end if

Recursive to solve for \( p \)
Multigrid LCP Solver

Algorithm 2 V_Cycle(m)
1. if \( m == 1 \) then
2. Solve the linear system, \( A^1 p^1 = b^1 \)
3. else
4. for \( i = 1 \) to num_Pre_Sweep do
5. Smooth(\( p^m \)) and enforce \( p^m_{min} \) (PRBGS)
6. end for
7. \( r^m = b^l - A p^m \)
8. \( b^{m-1} = \text{Restrict}(r^m) \)
9. \( p^{m-1} = 0 \)
10. if \( m > M - S \) then
11. \( p^{m-1}_{min} = \text{DownsampleSubtract}(p^{m}_{min}, p^m) \)
12. else
13. \( p^m_{min} = -\infty \)
14. end if
15. \( V\_Cycle(m - 1) \)
16. \( p^m = p^m + \text{Prolong}(p^{m-1}) \)
17. for \( i = 1 \) to num_Post_Sweep do
18. Smooth (\( p^m \)) and enforce \( p^m_{min} \) (PRBGS)
19. end for
20. end if

Prolong
- Up sampling
- Tri-linear interpolation
Algorithm 2 V_Cycle(m)

1: if m == 1 then
2:   Solve the linear system, $A^1 p^1 = b^1$
3: else
4:   for $i = 1$ to num_Pre_Sweep do
5:     Smooth($p^m$) and enforce $p^m_{\text{min}}$ (PRBGS)
6:   end for
7:   $r^m = b^l - A p^m$
8:   $b^{m-1} = \text{Restrict}(r^m)$
9:   $p^{m-1} = 0$
10: if $m > M - S$ then
11:   $p^{m-1}_{\text{min}} = \text{DownsampleSubtract}(p^{m}_{\text{min}}, p^m)$
12: else
13:   $p^m_{\text{min}} = -\infty$
14: end if
15: V_Cycle(m - 1)
16: $p^m = p^m + \text{Prolong}(p^{m-1})$
17: for $i = 1$ to num_Post_Sweep do
18:   Smooth ($p^m$) and enforce $p^m_{\text{min}}$ (PRBGS)
19: end for
20: end if
Multigrid LCP Solver

Algorithm 2 V_Cycle(m)
1: if $m == 1$ then
2:   Solve the linear system, $A^1 p^1 = b^1$
3: else
4:   for $i = 1$ to num_Pre_Sweep do
5:     Smooth($p^m$) and enforce $p_{min}^m$ (PRBGS)
6:   end for
7: $r^m = b^l - A p^m$
8: $b^{m-1} = \text{Restrict}(r^m)$
9: $p^{m-1} = 0$
10: if $m > M - S$ then
11:   $p_{min}^{m-1} = \text{DownsampleSubtract}(p_{min}^m, p^m)$
12: else
13:   $p_{min}^m = -\infty$
14: end if
15: V_Cycle($m - 1$)
16: $p^m = p^m + \text{Prolong}(p^{m-1})$
17: for $i = 1$ to num_Post_Sweep do
18:   Smooth ($p^m$) and enforce $p_{min}^m$ (PRBGS)
19: end for
20: end if

Differences from traditional Multigrid
Algorithm 3 Full_Cycle()

1: \( p^{\text{tmp}} = p^M \)
2: Compute \( p^M_{\text{min}} \)
3: \( p^M_{\text{min}} = p^M \)
4: \( r^M = b^M - Ap^M \)
5: for \( m = M - 1 \) down to 1 do
6: \( r^m = \text{Restrict}(r^{m+1}) \)
7: if \( m \geq M - S \) then
8: \( p^m_{\text{min}} = \text{DownsampleSubtract}(p^{m+1}_{\text{min}}, 0) \)
9: else
10: \( p^m_{\text{min}} = -\infty \)
11: end if
12: end for
13: \( b^1 = r^1 \)
14: Solve the linear system, \( A^1 p^1 = b^1 \)
15: for \( m = 2 \) to \( M \) do
16: \( p^m = \text{Prolong}(p^{m-1}) \)
17: \( b^m = r^m \)
18: \( \text{V_Cycle}(m) \)
19: end for
20: \( p^M = p^{\text{tmp}} + p^M \)
Results

3D Dam Break in a Box

64x64x64 Grid
Results

- Timing in ms, done in GTX480

<table>
<thead>
<tr>
<th>Case</th>
<th>Res</th>
<th>No LCP</th>
<th>LCP</th>
<th>% Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>BallBox</td>
<td>$64^3$</td>
<td>19.00</td>
<td>21.26</td>
<td>11.89</td>
</tr>
<tr>
<td>DambreakBox</td>
<td>$64^3$</td>
<td>18.89</td>
<td>21.17</td>
<td>12.07</td>
</tr>
<tr>
<td>RotatedBox</td>
<td>$128^3$</td>
<td>109.78</td>
<td>122.97</td>
<td>12.01</td>
</tr>
<tr>
<td>DambreakSphere</td>
<td>$128^3$</td>
<td>109.67</td>
<td>122.58</td>
<td>11.77</td>
</tr>
</tbody>
</table>

- No more than 12% slower than multigrid w/o LCP
  - MG faster than CG about 13X, CM11
  - Expected to be much faster than BBB07

- Because the solver used was much slower than CG
Discussions

- Only one way solid-liquid coupling is currently supported

- Two-way solid-liquid such as by incorporating Robinson-Mosher et al. 08
  - Will be challenging and interesting future work
Thank you for your attention!
Solving LCP

• BBB07 formulate as quadratic programming (QP)
  – Used PATH solver for it
  – Slow, feasible only for small 2D domain

• Narian et al. 10
  – Solve LCP resulting from sand simulation
  – Use conjugate gradient - liked QP solver