Real-Time Simulation and Control of Physics

Jos Stam
Research
Alias Systems, Toronto
Physics in this Talk

- Fluids: Smoke, Fire, Water
- Cloth
- Hair
- Caustics
- Diffraction
Goals

• Has to Look Good

• Be Fast – Real-Time (Stable)

• Simple to Code

• Control
Fluid Mechanics

- Natural Framework
- Lots of Previous Work
- Very Hard Problem
- We want Visual Accuracy…
Fluid Mechanics
Main Application

Moving Densities

2D

3D
Main Application

While ( Simulating )

Get Forces from UI
Get Source Densities from UI
Update Velocity Field
Update Density Field
Display Density
Navier-Stokes Equations

\[ \frac{\partial u}{\partial t} = -(u \cdot \nabla)u + \nu \nabla^2 u + f \]

Velocity

\[ \frac{\partial \rho}{\partial t} = -(u \cdot \nabla)\rho + \kappa \nabla^2 \rho + S \]

Density

Equations Very Similar
What Does it Mean?

\[
\frac{\partial \rho}{\partial t} = -(u \cdot \nabla) \rho + \kappa \nabla^2 \rho + S
\]

Over Time…
What Does it Mean?

\[
\frac{\partial \rho}{\partial t} = - (u \cdot \nabla) \rho + \kappa \nabla^2 \rho + S
\]

Density Follows Velocity...
What Does it Mean?

\[
\frac{\partial \rho}{\partial t} = -(u \cdot \nabla)\rho + \kappa \nabla^2 \rho + S
\]

Density Diffuses at a Rate ... $\kappa$
What Does it Mean?

\[ \frac{\partial \rho}{\partial t} = - (u \cdot \nabla) \rho + \kappa \nabla^2 \rho + S \]

Density Increases Due to Sources...
Fluid in a Box

float dens[i,j], u[i,j], v[i,j]

Density + velocity constant in each cell
Simulation

Initial State → Add Sources → Diffuse → Follow Velocity
Diffusion Step

dens0[i,j] → dens[i,j]

dt
Diffusion Step

Exchanges Between Neighbors
Diffusion Step

Exchanges Between Neighbors
Diffusion Step

Change: Density Flux IN – Density Flux OUT

\[ \text{diff}\times\text{dt}\times\left(\text{dens0}\left[i-1,j\right] - \text{dens0}\left[i,j\right]\right)/\left(h\times h\right) \]
Diffusion Step

dens[i,j] - dens0[i,j] = a*(dens0[i-1,j]+dens0[i+1,j]+dens0[i,j-1]+dens0[i,j+1]-4*dens0[i,j]);

a = diff*dt/(h*h)
Diffusion Step

\[\text{dens}[i,j] = \text{dens0}[i,j] + a \times (\text{dens0}[i-1,j]+\text{dens0}[i+1,j]+\text{dens0}[i,j-1]+\text{dens0}[i,j+1]-4\times\text{dens0}[i,j]);\]

\[a = \text{diff} \times \text{dt}/(\text{h} \times \text{h})\]
void diffuse_bad ( float * dens, float * dens0 )
{
    int i, j;
    float a = diff*dt/(h*h);

    for ( i=1 ; i<=N ; i++ ) {
        for ( j=1 ; j<=N ; j++ ) {
            dens[i,j] = dens0[i,j] + a*(dens0[i-1,j]+dens0[i+1,j]+dens0[i,j-1]+dens0[i,j+1]-4*dens0[i,j]);
        }
    }
}

Simple But Doesn’t Work: Unstable

\[ a = \frac{\text{diff} \times dt}{(h \times h)} > 0.5 \]
Diffusion Step

\[ \text{dens0}[i,j] \quad \text{dens}[i,j] \]

Diffuse Backwards: Stable
Diffusion Step

dens0[i,j] = dens[i,j] -
a*(dens[i-1,j]+dens[i+1,j]+ dens[i,j-1]+dens[i,j+1]-4*dens[i,j]);

Linear System: Ax=b

Use a Fast Sparse Solver
Linear Solvers

- Gaussian Elimination: $N^3$
- Gauss-Seidel Relaxation: $N^2$
- Conjugate Gradient: $N^{1.5}$
- Cyclical Reduction: $N \log N$
- Multi-Grid: $N$
void lin_solve ( float * x, float * b, float a, float c )
{
    int i, j, n;

    for ( n=0 ; n<20 ; n++ ) {
        for ( i=1 ; i<=N ; i++ ) {
            for ( j=1 ; j<=N ; j++ ) {
                x[i,j] = (b[i,j] + a*(x[i-1,j]+x[i+1,j]+x[i,j-1]+x[i,j+1]))/c;
            }
        }
    }
}

void diffuse ( float * dens, float * dens0 )
{
    float a = diff*dt/(h*h);
    lin_solve ( dens, dens0, a, 1+4*a );
}
Simulation

Initial State  Add Sources  Diffuse  Follow Velocity
Follow Velocity

dens0[i,j], dens[i,j], u[i,j], v[i,j]
Follow Velocity

Fluxes Depend on Velocity
Follow Velocity

Better Idea:

Step Easy If Density Were Particles
Follow Velocity
Follow Velocity
For Each Cell…

dens[i,j]
Follow Velocity

Trace BackWard

\[ x = i - dt \cdot u[i, j]; \]
\[ y = j - dt \cdot v[i, j]; \]
Follow Velocity

Find Four Neighbors

\[ i_0 = \text{(int)}x; \quad i_1 = i_0 + 1; \quad s = x - i_0; \]
\[ j_0 = \text{(int)}y; \quad j_1 = j_0 + 1; \quad t = y - j_0; \]
Follow Velocity
Interpolate From Neighbors

\[ d = (1-s)*((1-t)*dens0[i0,j0]+t*dens0[i0,j1]) + 
    s *((1-t)*dens0[i1,j0]+t*dens0[i1,j1]); \]
Follow Velocity
Set Interpolated Value in Cell

dens[i,j] = d;
Follow Velocity

```c
void advect ( float * dens, float * dens0, float * u, float * v )
{
    int i, j, i0, j0, i1, j1;
    float x, y, s0, t0, s1, t1;

    for ( i=1 ; i<=N ; i++ ) {
        for ( j=1 ; j<=N ; j++ ) {
            x = i-dt*u[i,j]; y = j-dt*v[i,j];
            if (x<0.5) x=0.5; if (x>N+0.5) x=N+0.5;
            if (y<0.5) y=0.5; if (y>N+0.5) y=N+0.5;
            i0=(int)x; i1=i0+1; j0=(int)y; j1=j0+1;
            s1 = x-i0; s0 = 1-s1; t1 = y-j0; t0 = 1-t1;
            dens[i,j] = t0*(s0*dens0[i0,j0]+s1*dens0[i0,j1])+
                         t1*(s0*dens0[i1,j0]+s1*dens0[i1,j1]);
        }
    }
}
```
Simulation

```c
void dens_step ()
{
    add_sources(dens);
    SWAP(dens,dens0); diffuse(dens,dens0);
    SWAP(dens,dens0); advect(dens,dens0,u,v);
}
```
Navier-Stokes Equations

\[
\frac{\partial u}{\partial t} = -(u \cdot \nabla)u + \nu \nabla^2 u + f
\]

Velocity

\[
\frac{\partial \rho}{\partial t} = -(u \cdot \nabla)\rho + \kappa \nabla^2 \rho + S
\]

Density
Navier-Stokes Equations

\[ \frac{\partial u}{\partial t} = -(u \cdot \nabla)u + \nu \nabla^2 u + f \]

Velocity moved by itself: self-advection
Velocity Solver

```c
void velocity_step ()
{
   add_sources(u);
   add_sources(v);
   SWAP(u,u0); SWAP(v,v0);
   diffuse(u,u0);
   diffuse(v,v0);
   SWAP(u,u0); SWAP(v,v0);
   advect(u,u0,u0,v0);
   advect(v,v0,u0,v0);
   project(u,v,u0,v0);
}
```

```c
void dens_step ()
{
   add_sources(dens);
   SWAP(dens,dens0); diffuse(dens,dens0);
   SWAP(dens0); diffuse(dens0);
   advect(dens,dens0,u,v);
}
```

Reuse Density Solver Code
Except for one Routine...
Projection Step

Flow should conserve mass

Flow IN = Flow OUT
Projection Step

Hodge Decomposition:

\[ \text{Any Field} = \text{Mass Conserving} + \text{Gradient} \]
Projection Step

Subtract Gradient Field

Mass Conserving = Any Field - Gradient
Projection Step

Gradient: Direction of steepest Descent of a height field.

\[ G_{x}[i,j] = 0.5* (p[i+1,j] - p[i-1,j]) / h \]
\[ G_{y}[i,j] = 0.5* (p[i,j+1] - p[i,j-1]) / h \]
Projection Step

Height Field satisfies a Poisson Equation


\[ \text{div}[i,j] = -0.5h*(u[i+1,j] - u[i-1,j] + v[i,j+1] - v[i,j-1]) \]

Ideally div is zero: Flow IN = Flow OUT

Reuse linear solver of the diffusion step
void project ( float * u, float * v, float * div, float * p )
{
    int i, j;
    // compute divergence
    for ( i=1 ; i<=N ; i++ ) {
        for ( j=1 ; j<=N ; j++ ) {
            div[i,j] = -0.5*h*(u[i+1,j]-u[i-1,j]+v[i,j+1]-v[i,j-1]);
            p[i,j] = 0.0;
        }
    }
    set_bnd ( 0, div ); set_bnd ( 0, p );
    // solve Poisson equation
    lin_solve ( p, div, 1, 4 );
    // subtract gradient field
    for ( i=1 ; i<=N ; i++ ) {
        for ( j=1 ; j<=N ; j++ ) {
            u[i,j] -= 0.5*(p[i+1,j]-p[i-1,j])/h;
            v[i,j] -= 0.5*(p[i,j+1]-p[i,j-1])/h;
        }
    }
    set_bnd ( 1, u ); set_bnd ( 2, v );
}
Boundaries

Fixed Walls

Periodic

Inflow + internal
Boundaries

Add another layer around the grid
**Boundaries**

<table>
<thead>
<tr>
<th></th>
<th>0.5</th>
<th>0.1</th>
<th>0.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.2</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>0.1</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

Densities: simply copy over values
### Boundaries

<table>
<thead>
<tr>
<th></th>
<th>0.0</th>
<th>0.5</th>
<th>0.1</th>
<th>0.2</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.5</td>
<td>-0.5</td>
<td>0.5</td>
<td>0.1</td>
<td>0.2</td>
<td>-0.2</td>
</tr>
<tr>
<td>-0.4</td>
<td>-0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>0.0</td>
<td>-0.0</td>
</tr>
<tr>
<td>-0.2</td>
<td>-0.2</td>
<td>0.2</td>
<td>0.1</td>
<td>0.0</td>
<td>-0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**U-velocity**: zero on vertical boundaries
**Boundaries**

<table>
<thead>
<tr>
<th></th>
<th>0.0</th>
<th>-0.5</th>
<th>-0.1</th>
<th>-0.2</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>0.1</td>
<td>0.2</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>0.4</td>
<td>0.2</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>0.2</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>-0.2</td>
<td>-0.1</td>
<td>-0.0</td>
<td>0.0</td>
<td></td>
</tr>
</tbody>
</table>

**V-velocity:** zero on horizontal boundaries
void set_bnd ( int b, float * x )
{
    int i;

    for ( i=1 ; i<=N ; i++ ) {
        x[0,i] = b==1 ? -x[1,i] : x[1,i];
        x[N+1,i] = b==1 ? -x[N,i] : x[N,i];
        x[i,0] = b==2 ? -x[i,1] : x[i,1];
        x[i,N+1] = b==2 ? -x[i,N] : x[i,N];
    }

    x[0,0] = 0.5*(x[1,0]+x[0,1]);
    x[0,N+1] = 0.5*(x[1,N+1]+x[0,N]);
    x[N+1,0] = 0.5*(x[N,0]+x[N+1,1]);
    x[N+1,N+1] = 0.5*(x[N,N+1]+x[N+1,N]);
}

Call after every update of the grids
Internal Boundaries
Internal Boundaries
Internal Boundaries

For density
# Internal Boundaries

For density

<table>
<thead>
<tr>
<th></th>
<th>0.0</th>
<th>0.0</th>
<th>0.5</th>
<th>0.1</th>
<th>0.2</th>
<th>0.1</th>
<th>0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>0.2</td>
<td>0.5</td>
<td>0.2</td>
<td>0.3</td>
<td>0.2</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>0.6</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>0.5</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>0.2</td>
<td>0.1</td>
<td>0.5</td>
<td>0.8</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>0.7</td>
<td>0.9</td>
<td></td>
</tr>
</tbody>
</table>
The Code

Entire solver in 100 lines of (readable) C-code...
#define IX(i,j) ((i)+(N+2)*(j))
#define SWAP(x0,x) {float * tmp=x0;x=tmp;tmp=x0;}
#define FOR_EACH_CELL for ( i=1 ; i<=N ; i++ ) {
    for ( j=1 ; j<=N ; j++ ) {
        for (...
    }
    END_FOR
}

void add_source(int N, float *x, float *s, float dt)
{
    int i, size=(N+2)*(N+2);
    for ( i=0 ; i<size ; i++ ) x[i] += dt*s[i];
}

void set_bnd(int N, int b, float *x)
{
    int i;
    for ( i=1 ; i<=N ; i++ ) {
        x[IX(0  ,i)] = b==1 ? -x[IX(1,i)] : x[IX(1,i)];
        x[IX(N+1,i)] = b==1 ? -x[IX(N,i)] : x[IX(N,i)];
        x[IX(i,0  )] = b==2 ? -x[IX(i,1)] : x[IX(i,1)];
        x[IX(i,N+1)] = b==2 ? -x[IX(i,N)] : x[IX(i,N)];
    }
    x[IX(0  ,0  )] = 0.5f*(x[IX(1,0  )]+x[IX(0  ,1)]);
    x[IX(0  ,N+1)] = 0.5f*(x[IX(0  ,N+1)]+x[IX(0  ,N)]);
    x[IX(N+1,0  )] = 0.5f*(x[IX(N,0  )]+x[IX(N+1,1)]);
    x[IX(N+1,N+1)] = 0.5f*(x[IX(N,N+1)]+x[IX(N+1,N)]);
}

void lin_solve(int N, int b, float *x, float *x0, float a, float c)
{
    int i, j, n;
    for ( n=0 ; n<20 ; n++ ) {
        FOR_EACH_CELL
            x[IX(i,j)] = (x0[IX(i,j)]+a*(x[IX(i-1,j)]+x[IX(i+1,j)]+x[IX(i,j-1)]+x[IX(i,j+1)]))/c;
        END_FOR
    }
}

void diffuse(int N, int b, float *x, float *x0, float diff, float dt)
{
    float a=dt*diff*N*N;
    lin_solve ( N, b, x, x0, a, 1+4*a );
}

void advect(int N, int b, float *d, float *d0, float *u, float *v, float dt)
{
    int i, j, i0, j0, i1, j1;
    float x, y, s0, t0, s1, t1;
    dt0 = dt*N;
    FOR_EACH_CELL
        x = i*dt0*u[IX(i,j)]; y = j*dt0*v[IX(i,j)];
      if (x<0.5f) x=0.5f; if (x>N+0.5f) x=N+0.5f; i0=(int)x; i1=i0+1;
      if (y<0.5f) y=0.5f; if (y>N+0.5f) y=N+0.5f; j0=(int)y; j1=j0+1;
      s0 = x-i0; s1 = 1-s0; t0 = y-j0; t1 = 1-t0;
        d[IX(i,j)] = s0*(t0*d0[IX(i0,j0)]+t1*d0[IX(i0,j1)])+
                    s1*(t0*d0[IX(i1,j0)]+t1*d0[IX(i1,j1)]);
    END_FOR
    set_bnd ( N, b, d );
}

void project(int N, float * u, float * v, float * p, float * div)
{
    int i, j;
    FOR_EACH_CELL
        div[IX(i,j)] = -0.5f*(u[IX(i+1,j)]-u[IX(i-1,j)]+v[IX(i,j+1)]-v[IX(i,j-1)])/N;
        p[IX(i,j)] = 0;
    END_FOR
    set_bnd ( N, 0, div );
    set_bnd ( N, 0, p );
    lin_solve ( N, 0, p, div, 1, 4 );
    FOR_EACH_CELL
        u[IX(i,j)] = 0.5f*N*(p[IX(i+1,j)]-p[IX(i-1,j)]);
        v[IX(i,j)] = 0.5f*N*(p[IX(i,j+1)]-p[IX(i,j-1)]);
    END_FOR
    set_bnd ( N, 1, u ); set_bnd ( N, 2, v );
}

void dens_step(int N, float *x, float *x0, float *u, float *v, float diff, float dt)
{
    add_source ( N, x, x0, dt );
    SWAP ( x0, x );
    diffuse ( N, 0, x, x0, diff, dt );
    SWAP ( x0, x );
    advect ( N, 0, x, x0, u, v, dt );
}

void vel_step(int N, float *u, float *v, float *u0, float *v0, float visc, float dt)
{
    add_source ( N, u, u0, dt );
    SWAP ( u0, u );
    diffuse ( N, 1, u, u0, visc, dt );
    SWAP ( v0, v );
    diffuse ( N, 2, v, v0, visc, dt );
    project ( N, u, u0, v0 );
    SWAP ( u0, u );
    SWAP ( v0, v );
    advect ( N, 1, u, u0, u0, v0, dt );
    advect ( N, 2, v, v0, u0, v0, dt );
    project ( N, u, u0, v0 );
}
Demo

Show 2D Demos

First Demo

Paint Demo
Demo

Liquid Textures Demo

Liquid Textures Demo

Liquid Textures Demo
3D Solver

for ( i=1 ; i<=N ; i++ ) {
    for ( j=1 ; j<=N ; j++ ) {
        density[i,j] = ...
    }
}

Becomes...

for ( i=1 ; i<=N ; i++ ) {
    for ( j=1 ; j<=N ; j++ ) {
        for ( k=1 ; k<=N ; k++ ) {
            density[i,j,k] = ...
        }
    }
}

3D Solver

Volume render density
Demo

Shroom 3D Demo

3D Falling Ball Demo

3D Rolling Ball Demo

3D Explosion Demo
An Album Of Fluid Motion
An Album Of Fluid Motion

Von Karmann

Kelvin-Helmholtz

Rayleigh-Benard
Flows on Surfaces

Catmull-Clark Fluid Demo
Flows on Surfaces

+ reaction - diffusion
MAYA Fluid Effects

Fluid Solver now available in Maya 4.5+ Unlimited

Download the screensaver

http://www.aliaswavefront.com
MAYA Fluid Effects
Fluids on PDAs

Palm

PocketPC
Cloth Simulation (Old)

System of Springs:

Problem: Cloth is Stiff $\rightarrow$ Slow
Cloth Simulation (New)

See Jakobsen’s GDC 2001 paper
Cloth Simulation (New)
Hair Simulation

1D Cloth:
Hair Simulation

Hair Demo
Caustics
Caustics

Caustics Demo
Diffraction

Vertex Program Version: GPU GEMS
Controlling Simulations

Make simulations do what we want!
Canon Ball

Demo
Framework

simulator

... controls

output state

desired state

optimizer
Rigid Bodies
Control of Fluids

Work with

Adrien Treuille
Antoine McNamara
Zoran Popovic

University of Washington
Seattle
Control of Fluids

Demo

Result
The End

Thanks