Physically Based Simulations  
(on the GPU)

(some material from slides of Mark Harris)

CS535  
Fall 2014

Daniel G. Aliaga  
Department of Computer Science  
Purdue University
Simulating the world

- Floating point arithmetic on GPUs and their speed enable us to simulate a wide variety of phenomena using PDEs
Some Basics

• Operators (on images/lattices)
• Diffusion
• Bouyancy
Operators

• Given an image:
  – Gradient (vector)
    \[ \nabla f(x, y) = \frac{\partial f}{\partial x} \hat{x} + \frac{\partial f}{\partial y} \hat{y} \n\]
  – Laplacian (scalar)
    \[ \nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \]
Discrete Laplacian

\[ \nabla^2 f(x, y) = \\
\quad f(x - 1, y) + f(x + 1, y) + \\
\quad f(x, y - 1) + f(x, y + 1) - \\
\quad 4f(x, y) \]

• Matrix form = ??
Discrete Laplacian

• $\nabla^2 f(x, y) =$

$$f(x - 1, y) + f(x + 1, y) + f(x, y - 1) + f(x, y + 1) - 4f(x, y)$$

• Matrix form =

$$\begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{bmatrix}$$
Heat Equation

$$\frac{\partial f}{\partial t} = \nabla^2 f$$
Diffusion Equation

[Weisstein 1999]

\[ f'(x, y) = f(x, y) + \frac{c_d}{4} \nabla^2 f(x, y) \]

where \( c_d \) is the coefficient of diffusion...
(Anisotropic) Diffusion

(a) Original Image

(b) Time = 5

(c) Time = 10

(d) Time = 30
Buoyancy

• Used in convection, cloud formations, etc.

• Given a temperature state \( T \):
  – a vertical buoyancy velocity is ‘upwards’ if a node is hotter than its neighbors’ and
  – a vertical buoyancy velocity is ‘downwards’ if a node is cooler than its neighbors
Buoyancy

\[ \nu(x, y)' = \nu(x, y) + \frac{c_b}{2} (2f(x, y) - f(x + 1, y) - f(x - 1, y)) \]

where \( c_b \) is the buoyancy strength
Bouyancy
(considering neighbors)

\[
\begin{align*}
f'(x, y) &= f(x, y) - \frac{\sigma}{2} f(x, y) \\
&= \left[ \rho(f(x, y + 1)) - \rho(f(x, y - 1)) \right]
\end{align*}
\]

where \( \rho(f) = \tanh(\alpha(f - f_c)) \) (an approx. of density relative to temperature \( f \)) and \( \sigma \) is buoyancy strength and \( \alpha \) and \( f_c \) are constants.
Euler Method (for ODE)

• Given:
  \[ y'(t) = f(t, y(t)) \] with \( y(t_0) = y_0 \)

• Do:
  \[ y_{n+1} = y_n + hf(t_n, y_n) \]
Classical Runge Kutta Method

- Given:
  \[ y'(t) = f(t, y(t)) \text{ with } y(t_0) = y_0 \]

- Do:
  \[ y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \]
  \[ t_{n+1} = t_n + h \]

where

\[ k_1 = f(t_n, y_n), \]
\[ k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1), \]
\[ k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2), \]
\[ k_4 = f(t_n + h, y_n + hk_3). \]
Example: (Water) Boiling

• Based on [Harris et al. 2002]
• State = Temperature
• Three operations:
  – Diffusion, buoyancy, & latent heat
• 3D Simulation
  – Stack of 2D texture slices
“Alan Turing in 1952 describing the way in which non-uniformity (stripes, spots, spirals, etc.) may arise naturally out of a homogeneous, uniform state. The theory (which can be called a reaction–diffusion theory of morphogenesis), has served as a basic model in theoretical biology, and is seen by some as the very beginning of chaos theory.”

\[
\frac{\partial U}{\partial t} = D_U \nabla^2 U - k(UV - 16)
\]

\[
\frac{\partial V}{\partial t} = D_V \nabla^2 V + k(UV - 12 - V)
\]
Gray-Scott Reaction-Diffusion

- State = two scalar chemical concentrations
- Simple:
  - just **Diffusion** and **Reaction** ops

\[
\begin{align*}
\frac{\partial U}{\partial t} &= D_u \nabla^2 U - UV^2 + F(1 - U), \\
\frac{\partial V}{\partial t} &= D_v \nabla^2 V + UV^2 - (F + k)V
\end{align*}
\]

*U, V are chemical concentrations,*

*F, k, D_u, D_v are constants*
Some research...

Navier-Stokes Equations

- Describe flow of an incompressible fluid

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

\[ \nabla \cdot \mathbf{u} = 0 \]

- Advection
- Pressure Gradient
- Diffusion (viscosity)
- External Force

Velocity is divergence-free
Fluid Dynamics

• Solution of Navier-Stokes flow eqs.
  – Stable for arbitrary time steps (=fast!)
  – [Stam 1999], [Fedkiw et al. 2001]

• Can be implemented on latest GPUs
  – Quite a bit more complex than R-D or boiling

• See “Fast Fluid Dynamics Simulation on the GPU” (Harris, GPU Gems, 2004)
Fluid Simulations
Thermodynamics

- Temperature affected by
  - Heat sources
  - Advection
  - Latent heat released / absorbed during condensation / evaporation

- $\Delta$ temperature = advection + latent heat release + temperature input
Cloud Dynamics

- 3 components
  - 7 unknowns

- Fluid dynamics
  - Motion of the air

- Thermodynamics
  - Temperature changes

- Water continuity
  - Evaporation, condensation

Velocity: \( \mathbf{u} = (u, v, w) \)
Pressure: \( p \)

Potential temperature: \( \theta \)
(see dissertation)

Water vapor mixing ratio: \( q_v \)
Liquid water mixing ratio: \( q_c \)
Cloud Dynamics
Wave Equation

• Remember heat equation:
  – Rate of change of value proportional to Laplacian

• Wave equation:
  – Rate of change of the rate of change is also proportional to the Laplacian
Wave Equation

\[ \frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u \]

where \( u \) models the displacement and \( c \) is the propagation speed.
Water Simulation: Wave Equation

\[ U = \text{value}, \ V = \text{rate of change} \]

\[ \frac{\partial U}{\partial t} = \frac{b}{k} + d \nabla^2 U \]

\[ \frac{\partial V}{\partial t} = k \nabla^2 U \]
Water Simulation:
Wave Equation

• Demo...
Water Simulation: Sine Waves

\[ Asin(\omega x + t) \]
Water Simulation: Sine Waves

\[ A_1 \sin(\omega_1 x + t_1) + A_2 \sin(\omega_2 x + t_2) + \cdots \]
Water Simulation: Sine Waves

• Using sine-wave summations:

\[ H(x, y, t) = \sum A_i \sin(D_i \cdot (x, y)\omega_i + t\phi_i) \]

[use H as height or a pixel intensity]

• Pixel values over time are:

\[ P(x, y, t) = (x, y, H(x, y, t)) \]
Water Simulation: Sine Waves

(here, pixel normals are computed as well for reflections)
Water: Surface Normals

- Use binormal and tangent:

\[
B(x, y, t) = \left( \frac{dx}{dx}, \frac{dy}{dx}, \frac{dH(x,y,t)}{dx} \right) = (1, 0, \frac{dH(x,y,t)}{dx})
\]

\[
T(x, y, t) = \cdots = \left( 0, 1, \frac{dH(x, y, t)}{dy} \right)
\]

- Normal is:

\[
N(x, y, t) = B \times T
\]

\[
N(x, y, t) = \left( -\frac{dH(x, y, t)}{dx}, -\frac{dH(x, y, t)}{dy}, 1 \right)
\]
Water Simulation: Gerstner Waves

• These waves also change the $x, y$ of the wave imitating how points at top of wave are squished together and points at bottom are separated
Water Simulation: Gerstner Waves

\[ P(x, y, t) = \left[ x + \sum Q_i A_i D_i \cdot x \cos(\omega_i D_i \cdot (x, y) + \phi_i t) \right] \]

\[ = \left[ y + \sum Q_i A_i D_i \cdot y \cos(\omega_i D_i \cdot (x, y) + \phi_i t) \right] \]

\[ \sum A_i \sin(\omega_i D_i \cdot (x, y) + \phi_i t) \]

where \( Q_i = \text{sharpness} \)
Water Simulation: Gerstner Waves
Video

- https://www.youtube.com/watch?v=lqPa389v_i4s
Simulation Algorithm

- Advect quantities
  - Similar to [Stam, 1999]

- Compute and apply accelerations
  - Buoyancy

- Compute condensation, evaporation, and temperature changes

- Enforce momentum conservation
  - Otherwise velocity dissipates, loses "swirls"
  - Projection step of "Stable Fluids" [Stam, 1999]
Simulation Algorithm

• Most steps are simple
  – Most use one fragment program, one pass
  – Programs come directly from equations

• Tricky parts:
  – Staggered grid discretization
  – Stable Fluids projection step
  – Boundary conditions
  – 3D Simulation
Flat 3D Textures

3D Texture

Corresponding Flat 3D Texture

quad primitives

line primitives

N

N
Flat 3D Textures

• Advantages
  – One texture update per operation
  – Better use of GPU parallelism
  – Non-power-of-two Textures
  – Quick simulation preview

• Disadvantage
  – Must compute texture offsets