Particle-Based Fluid Simulation

CSE169: Computer Animation

Steve Rotenberg

UCSD, Winter 2020

Del Operations

- Del: $\nabla = \left[\frac{\partial}{\partial x} \quad \frac{\partial}{\partial y} \quad \frac{\partial}{\partial z}\right]$
- Gradient: $\nabla s = \left[\frac{\partial s}{\partial x} \quad \frac{\partial s}{\partial y} \quad \frac{\partial s}{\partial z}\right]$
- Divergence: $\nabla \cdot \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}$
- Curl: $\nabla \times \mathbf{v} = \left[\frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z} \quad \frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x} \quad \frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y}\right]$
- Laplacian: $\nabla^2 s = \frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2} + \frac{\partial^2 s}{\partial z^2}$

Transport Equations

- Advection: $\frac{ds}{dt} = -\mathbf{v} \cdot \nabla s$
- Convection: $\frac{d\mathbf{v}}{dt} = -\mathbf{v} \cdot \nabla \mathbf{v}$
- Diffusion: $\frac{ds}{dt} = k\nabla^2 s$
- Viscosity: $\frac{d\mathbf{v}}{dt} = \mu\nabla^2 \mathbf{v}$
- Pressure: $\frac{d\mathbf{v}}{dt} = -\nabla p$

Navier-Stokes Equation

- The incompressible Navier-Stokes equation describes the forces on a fluid as the sum of convection, viscosity, and pressure terms:

$$\frac{\partial \mathbf{v}}{\partial t} = -\mathbf{v} \cdot \nabla \mathbf{v} + \mu \nabla^2 \mathbf{v} - \nabla p$$

- In addition, we also have the incompressibility constraint:

$$\nabla \cdot \mathbf{v} = 0$$

- As discussed in the previous lecture, these are formulated from the point of view of a fixed point in space (Eulerian frame of reference)

Lagrangian Approach

- In the Lagrangian approach, we formulate the Navier-Stokes equation from the point of view of a particle moving along with the fluid
- The pressure and viscosity terms don't change, but the convection term goes away entirely

$$\frac{d\mathbf{v}}{dt} = \mu\nabla^2\mathbf{v} - \nabla p$$

- The paper also adds a term for externally applied forces as well as a $^1/_\rho$ term to account for the fact that the particle based methods will have some variation in density

$$\frac{d\mathbf{v}_i}{dt} = \mu\nabla^2\mathbf{v}_i - \frac{1}{\rho_i}\nabla p_i + \frac{\mathbf{f}_i}{m_i}$$

Particle-Based Fluids

Grid-Based Methods

- Fluid dynamics has traditionally been simulated on uniform grids or irregular meshes
- These techniques are good for engineering applications due to their potential for high accuracy
- They are effective in computer graphics uses as well, particularly for smoke and simulations of airflows
- However, it is very difficult to handle fluid interfaces (such as the water-air boundary) and complex splashing situations, which are the situations one often wants in entertainment applications

Smooth Particle Hydrodynamics

- Particle based fluid simulation is often referred to as *smooth particle hydrodynamics* or SPH
- Some of the original work was done for simulating galactic gas dynamics by astrophysicists
- The technique was introduced to the computer graphics community around 2003
- In recent years, advances in the techniques as well as increases in GPU computational power have made large-scale SPH simulations possible
- The technique has proven very effective, especially for simulating very dynamic situations with lots of splashing and interaction with complex surfaces

SPH Fluids in Computer Graphics

- We will follow the paper “SPH Fluids in Computer Graphics”
- Authors: Markus Ihmsen, Jens Orthmann, Barbara Solenthaler, Andreas Kolb, Matthias Teschner
- This was a ‘State of The Art Report’ (STAR) published in Eurographics 2014, and it summarizes much of the development on the subject that had been done in the previous 10 years
- This is a great overview of the subject and the important issues, and with lots of other great references in the bibliography
- Many of the formulas in these slides come from the paper as well as a few quoted statements

Field Representation

- We saw that scalar and vector fields can be represented by grids or meshes by sampling the field at the grid points and interpolating the values in between
- Alternately can use a meshless set of irregularly spaced particles to represent a field
- Rather than being a 0 dimensional point, a particle is thought of as being smeared out over some small radius
- The maximum radius of influence for a particle is called the *support radius*
- The particles have to be close enough together so that they effectively cover all of the domain of interest, and often they are arranged so that every point in the domain is overlapped by several particles (maybe around 3-10)

Field Representation

- Each particle i represents a small sample of the fluid so it will have a mass m_i, position $\mathbf{x}_i$, and velocity $\mathbf{v}_i$
- We will also compute various properties per particle such as the local density ρ_i, pressure p_i, or volume V_i

Kernel Function

- We define a *kernel function W*() that represents the strength of a particle's influence as a function of the distance from the particle
- At the support radius, we expect *W*() to go to 0 and smoothly increase to 1 as we get closer to the particle
- There have been many kernel functions proposed, and we will look at the one used in the paper

Kernel Function

- "A quantity A_i at arbitrary position $\mathbf{x}_i$ is approximately computed with a set of known quantities A_j at neighboring particle positions $\mathbf{x}_j$"

$$A_i = \sum_j \frac{m_j}{\rho_j} A_j W_{ij}$$

- With W_{ij} being a kernel function of the form

$$W_{ij} = W\left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{h}\right) = W(q) = \frac{1}{h^d} f(q)$$

- A typical choice for the function *f* () would be

$$f(q) = \frac{3}{2\pi} \begin{cases} \frac{2}{3} - q^2 + \frac{1}{2} q^3 & 0 \le q < 1 \\ \frac{1}{6}(2-q)^3 & 1 \le 2 \\ 0 & q \ge 2 \end{cases}$$

Kernel Function

$$W_{ij} = W\left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{h}\right) = W(q) = \frac{1}{h^d} f(q)$$

- The value *d* is the number of dimensions
- The value *h* is the *smoothing radius*, which is not the same as the support radius
- The particle spacing is typically close to *h*
- From examining the equations, one can see that the support radius is 2*h* in the example from the paper, but they mention it may vary from *h* to 3*h* depending on choice of kernel function *f*()

Derivative Computation

- When using uniform grids, we were able to easily compute various spatial derivatives by using finite differencing
- With particles, the irregular spacing makes derivative computation a little more tricky and several approaches have been used in the literature
- The paper uses the following:

$$\nabla A_i = \rho_i \sum_j m_j \left(\frac{A_i}{{\rho_i}^2} + \frac{A_j}{{\rho_j}^2} \right) \nabla W_{ij}$$

$$\nabla \cdot \mathbf{A}_i = -\frac{1}{\rho_i} \sum_j m_j \mathbf{A}_{ij} \cdot \nabla W_{ij}$$

$$\nabla^2 A_i = 2 \sum_j \frac{m_j}{\rho_j} A_{ij} \frac{\mathbf{x}_{ij} \cdot \nabla W_{ij}}{\mathbf{x}_{ij} \cdot \mathbf{x}_{ij} + 0.01h^2}$$

Derivative Computation

- Notice that all of the derivatives involve summing over *j*, which is the set of nearby particles within the support radius
- In order to do the calculations, every particle must determine which particles are nearby
- This can be problematic when you are dealing with millions of particles, and so special data structures are required to make this neighbor search fast
- We will look at those in a little bit

Equations of Motion

- SPH uses the Lagrangian form of the Navier-Stokes equation:

$$\frac{d\mathbf{v}_i}{dt} = \mu\nabla^2\mathbf{v}_i - \frac{1}{\rho_i}\nabla p_i + \frac{\mathbf{f}_i}{m_i}$$

- This is computed for each particle and then integrated using the standard forward Euler method of integration that we've been using for other particle systems
- Note that it requires a density ρ and pressure p to be calculated first and the paper has a couple different methods to do that depending on the specific approach

Algorithm

- The paper presents 4 different algorithms for SPH simulation
- They are all similar and effectively try to calculate the same thing, but they use different computational approaches and don't produce the exact same results
- The main difference between the approaches involves the exact order that things are done and how the incompressibility constraint is handled
- Most methods allow for some compression, but some of the better approaches attempt to enforce incompressibility using iterative approaches or solutions to large systems of equations
- In general, the more sophisticated methods require more time to compute a single time step, but run more stable and can handle larger time steps
- The result is that usually the more sophisticated approaches will outperform the simpler approaches in real world use cases

Integration & Time Steps

- We need small time steps to keep things stable, but ideally we could use larger time steps if things aren't moving as quickly
- We can therefore consider adapting the time step over time based on what is happening in the simulation
- It is common to base this off of the Courant-Friedrich-Levy (CFL) condition that limits the time step so that the fastest particle doesn't move more than some portion of it's smoothing radius:

$$\Delta t \leq \lambda \frac{h}{\|\mathbf{v}^{max}\|} \text{ with } \lambda \approx 0.4$$

Spatial Hash Tables

Neighbor Search

- The physics equations per particle are relatively fast to compute, as they mainly involve multiplication and addition with very little division, no trig functions, and no real exponentials
- One of the most expensive parts of the physics simulation in SPH is actually the search for the nearest neighbors for each particle
- To calculate the derivatives, every particle needs to know which particles are within its support radius
- If there are *N* particles, and every particle could potentially be a neighbor to every other particle, the search is inherently $O(N^2)$ which gets extremely expensive as *N* gets high (which it will!)
- With the right choice of data structures however, we can reduce the neighbor search for a single particle to constant time, making the overall search linear or $O(N)$

Uniform Grids

- We could use a grid structure to optimize our neighbor search by storing a list of particles in each grid cell
- The spacing of the grid cells is set to the support radius of the particles so that all of a particles neighbors will be within one grid cell away
- To search for neighbors, each particle would loop through the 3x3x3 set of nearby cells
- The physics will limit the maximum number of particles per cell with the pressure force
- Therefore, the neighbor search for a single particle is constant, as it requires checking 27 cells that will each contain roughly the same number of particles
- The two big problems with grids is the fact that they have a limited size, putting a range on how far our particles can move, and that they take up a lot of memory to account for all of the unoccupied cells. If we try to increase the range our particles can move, we need to increase the grid size, causing a very large memory penalty

Spatial Hash Table

- A *spatial hash table* is a wonderful data structure that has all of the advantages of uniform grids and doesn't suffer from the memory or range limitations
- It is effectively a virtual grid that extends infinitely in every direction, but we only store data for occupied cells, making unoccupied cells totally free

Spatial Hash Table

- A spatial hash table operates very much like a standard hash table, where a hashing function maps some key (like a string) to an integer, which is then mod'ed into an array of slots. Items can be added, removed, or accessed through the table in constant time
- The spatial hash table is essentially the same thing, but it uses a 3D position to map to a grid cell which is then hashed into the table
- The table stores occupied cells, each of which may contain several particles, but will be limited to some maximum number due to the physics
- If more than one occupied cell maps to the same table entry, then the table entry can simply contain a linked list of cells. In practice, if the table size is anywhere near the number of particles, then this will happen very rarely
- The paper refers to a 'compact hashing' scheme that uses some additional tricks to keep the memory size manageable

Hash Function

- A point in infinite space is mapped into a finite list of cells using a hash function such as:

$$c = \left[\left(\left\lfloor\frac{x}{d}\right\rfloor \cdot p_1\right) \text{xor} \left(\left\lfloor\frac{y}{d}\right\rfloor \cdot p_2\right) \text{xor} \left(\left\lfloor\frac{z}{d}\right\rfloor \cdot p_3\right)\right] \%m$$

- With *d* being the cell spacing, *m* the hash table size, and p_1, p_2, and p_3 being large prime numbers such as 73856093, 19349663, and 83492791

Rendering

SPH Rendering

- Now that we can simulate water with SPH and can do the calculations efficiently, we turn to the subject of rendering
- Rendering a smooth surface from a set of particles is challenging, and one of the more common criticisms of the technique is its inability to handle calm smooth water surfaces
- But who wants to simulate calm smooth water surfaces anyway?
- Actually, modern surface extraction techniques do a pretty good job at this, but one can still often spot bumpy visual artefacts in SPH renderings
- The paper describes 4 overall approaches to rendering SPH simulations:
  - Extract a polygonal surface and render that through standard techniques
  - Render each surface particle individually as some kind of splat
  - Use a ‘screen-space’ technique to render the pixels the contain the water surface
  - Use a volumetric rendering technique
- Surface extraction is more common for high quality renderings, and splat based and screen-space techniques tend to be more appropriate for real time rendering

Marching Cubes

- Most surface extraction techniques are based on the *marching cubes* algorithm or some variation of it
- With this approach, we first create a virtual grid around our particles where the grid size is a little smaller than the particle radius
- We then evaluate a ‘distance’ or ‘density’ type function on this grid, where each point computes some density value based on the particle nearby
- The surface is implicitly defined as being the set of points where the density value is some constant (i.e., an *isosurface*). An isosurface is a 3D version of the 2D isocurves one finds on a topographic map
- To find the surface, we loop through each cell and examine the 8 values on the corners of the cell. If some of the values of the cell are above the isosurface value and some are below, then we know that the surface passes through the cell. We then triangulate that small section of the surface and repeat for all cells

Marching Cubes

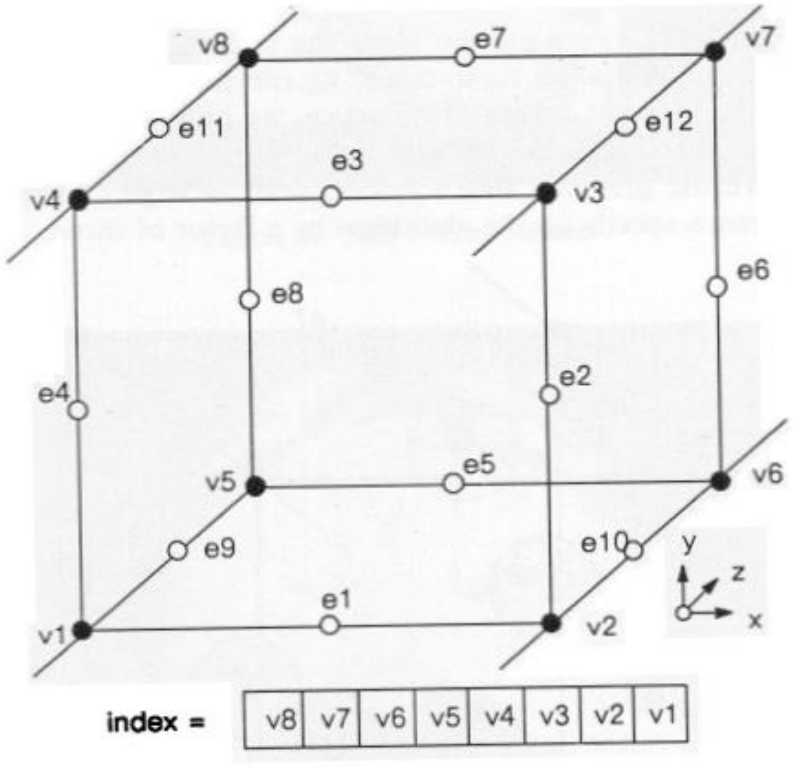


Figure 4. Cube Numbering.

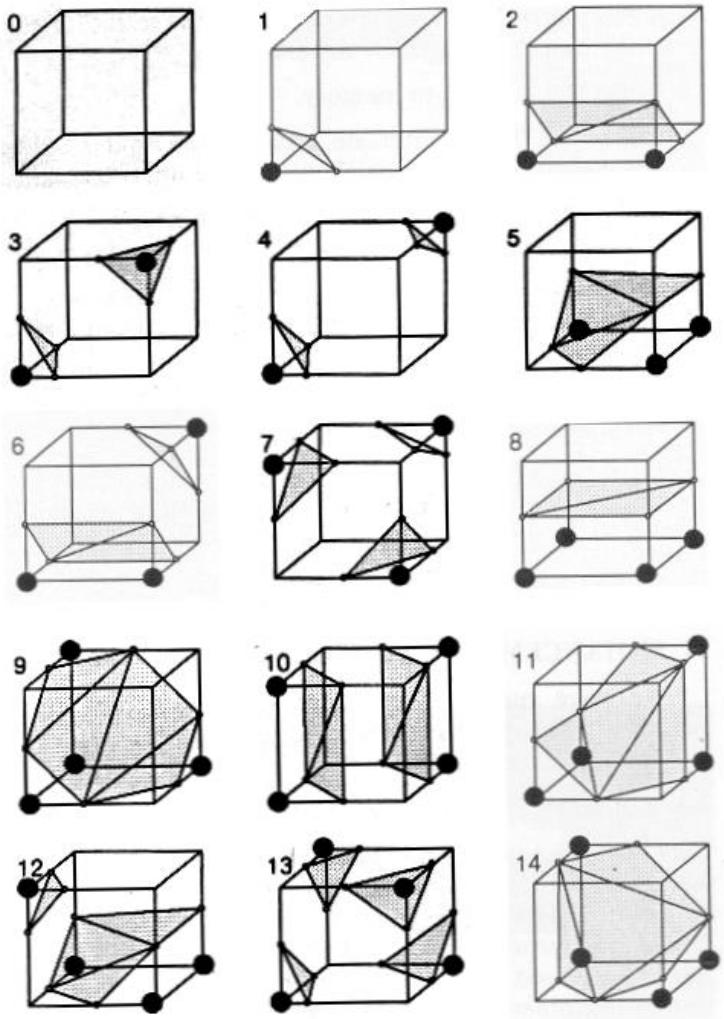
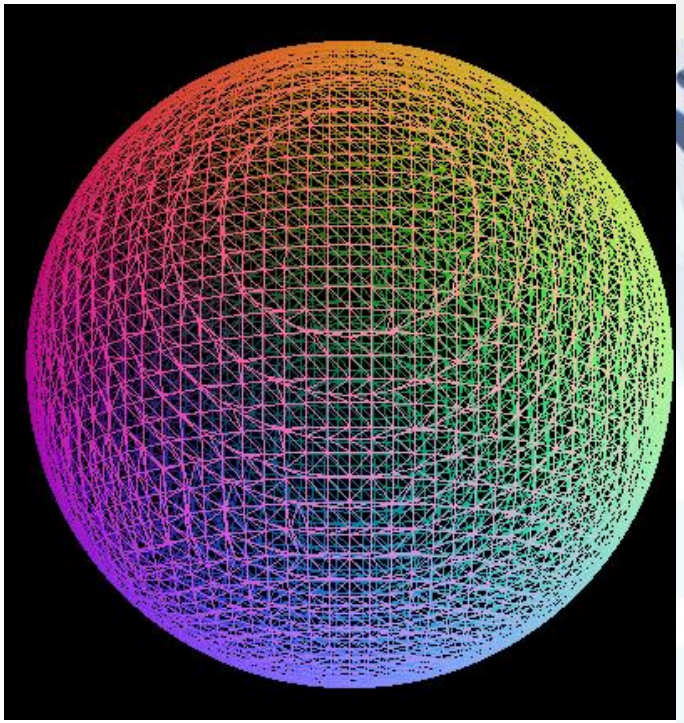
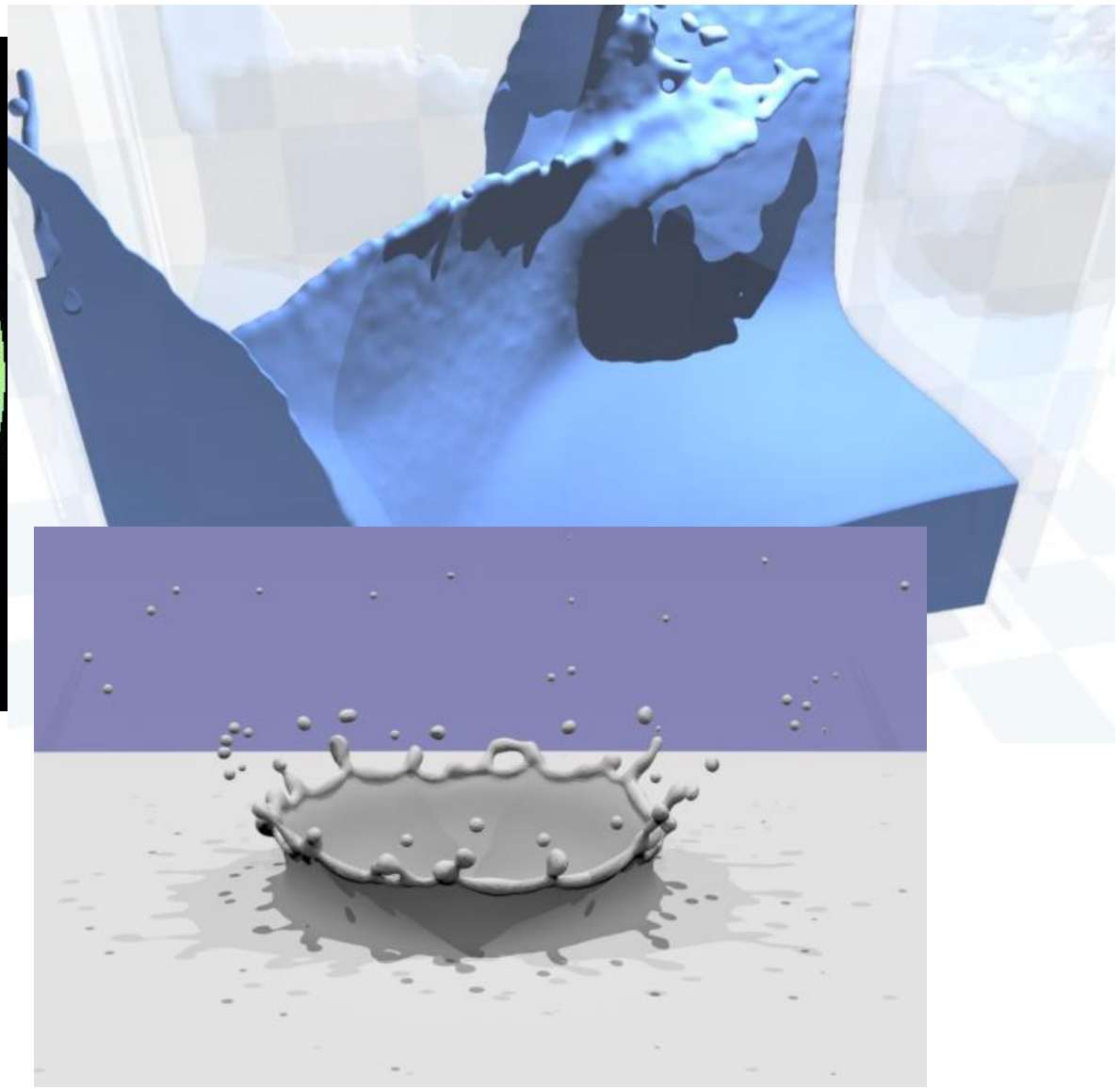


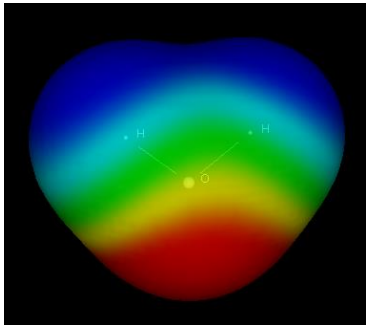
Figure 3. Triangulated Cubes.

Marching Cubes

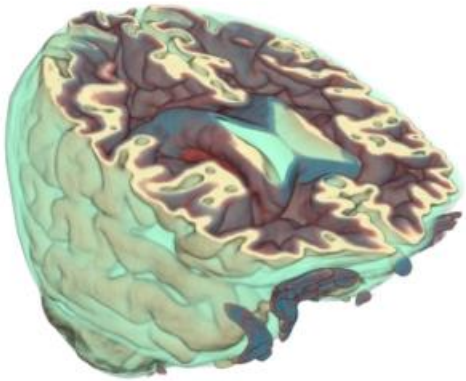


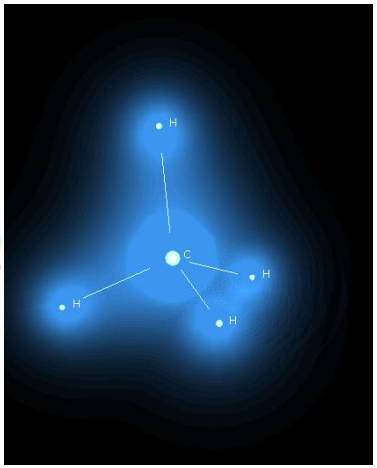


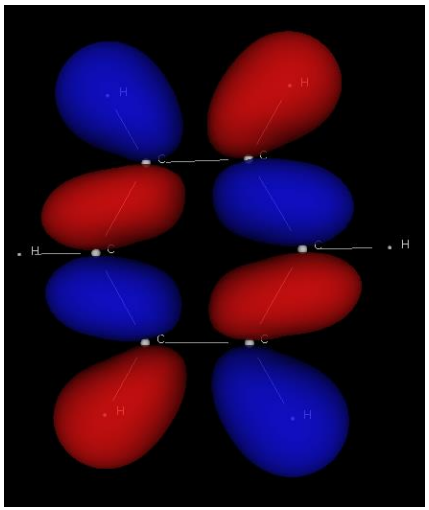
Marching Cubes

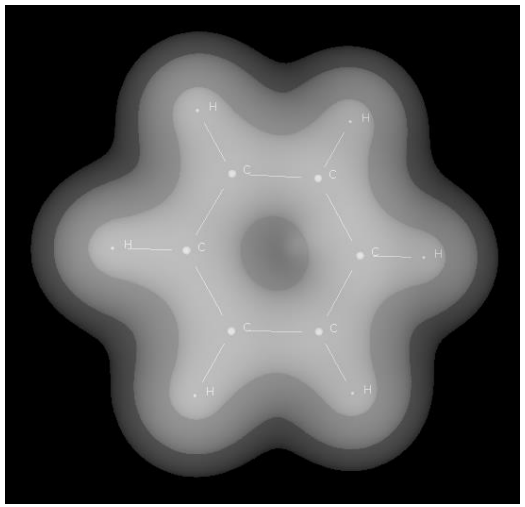


- In this example, we use marching cubes to triangulate the water-air interface. One can get good results by using a sufficiently high resolution and using a very carefully designed distance/density function, referred to in the paper
- However, marching cubes is a very powerful technique for visualizing 3D fields of all sorts. It is used throughout scientific visualization to view all things such as electromagnetic fields, quantum wave functions, gravitational fields, stress fields, MRI scan data, and more









SPH Extensions

SPH Extensions

- **Surface tension**: There are some good models for surface tension that can be added to the force computation. These will improve the behavior of small scale phenomena such as sheet breakup, and spherical drop formation

- **Two-scale simulation**: A common extension to SPH is to do a two level simulation where the first (course) level has does a full physics simulation on a smaller number of particles (say 500,000) and then the second (fine) level simply advects a large number of surface particles (maybe 5 million) through the velocity field calculated from the course scale. This allows efficient physics computation plus high visual detail at the surface where it is needed

- **Adaptive particles**: It's nice to have lots of small particles in areas where we need them such as around splashes. However, for slower moving areas of the fluid, it is inefficient to use tons of tiny particles. Adaptive schemes use particles of varying sizes that adapt automatically to the flow complexity. In other words, they will use large particles where the flow is smooth and the large particles will automatically subdivide into smaller particles as things get more dynamic. Likewise, small particles will coalesce into bigger particles where the flow slows down. Note that this applies to purely the physical simulation of the particles, and so this method is different, yet compatible with the two-scale method described above

SPH Extensions

- **Foam, spray, & bubbles**: One can also support special particles to represent foam on the water surface, spray of fine scale droplets in the air, and air bubbles under the surface. These particles are created in places where there are violent interactions between water particles near the air-water surface. This method dramatically improves the visual quality of crashing waves and large splashing scenes

- **Interaction with solids, cloth**: water particles can interact with solid objects like dynamic rigid bodies, or deformable elastic bodies such as cloth. A common approach is to create a bunch of virtual particles along the surface of the solid objects that interact with the water particles just like any other particle. The forces applied to the virtual particles are passed through to the rigid or elastic bodies to allow for two-way interaction between fluids and solids

- **High viscosity**: when we increase the fluid viscosity substantially, the fluid starts to behave more and more like a solid. When combined with non-Newtonian viscosity, we can get fluids that can hold a solid shape such as clay or toothpaste

SPH Extensions

- **Solid mechanics**: if we take high viscosity to an extreme, we can actually model deformable solid objects as well, such as rubber, metals, and more. Behaviors such as elasticity, plasticity, and fracture can be included

- **Phase changes**: if we can model both liquids and solids with SPH, then we can model phase changes such as melting, freezing, boiling, and condensation

- **Granular materials**: SPH has been extended to handle physics of granular materials like sand. Sand moves in a very fluid way, but requires some additional static friction behaviors that allow it to form piles. Also, as with water, one can use two-scale techniques where sand it modeled at a course level and then rendered at a finer scale

Project 5

Project 5

Choose one of the following:

- **Inverse Kinematics**: implement a single chain of 5 or more links that can interactively track a goal position that the user can move around in x, y, and z
- **SPH**: implement the basic SPH simulation algorithm with a small number of particles (say 1000 or less). No hash tables or surface extraction is required, but the particles should render as individual spheres or sprites at least
- **Rigid Body Dynamics**: implement the basic rigid body motion equations (Newton-Euler) for a box and implement collisions with the ground plane
- **Locomotion**: implement a 4 or 6 legged walking creature that uses analytical IK for the legs and supports at least two different gaits
- **Quaternion Interpolation**: implement a tool that allows the user to manually rotate at least 5 boxes and then displays a moving box that uses smooth quaternion interpolation to pass through the different boxes
- **Choose Your Own**: if you have an idea you'd like to do for the project, that would be great, but please come talk to me about it first