VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Data Visualization

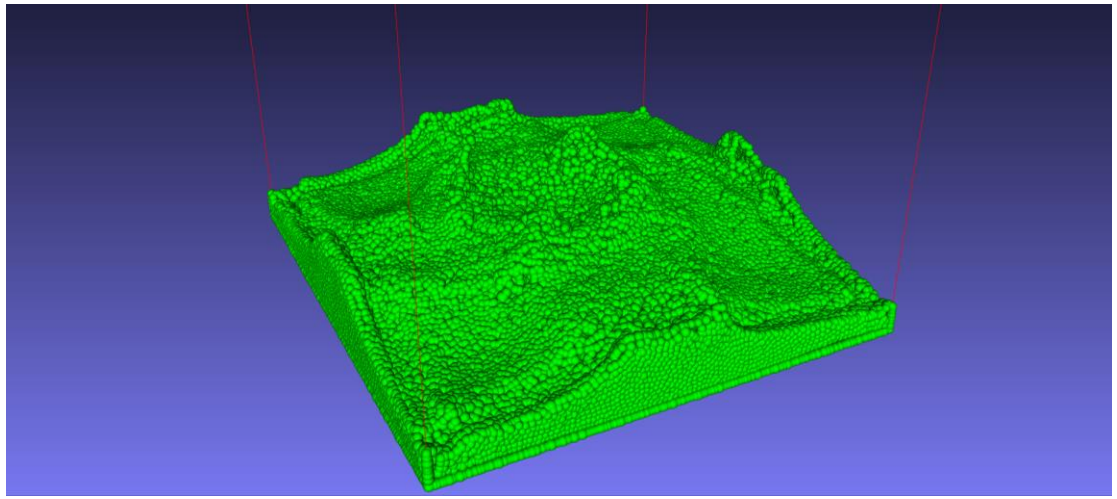
460-4120

Fall 2024

Last update 13. 11. 2024

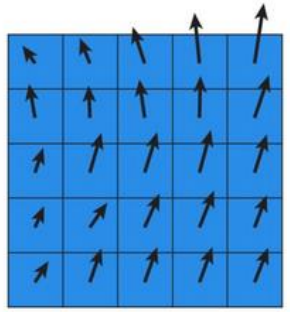
Smoothed Particle Hydrodynamics (SPH)

- Developed by Gingold, Monaghan and Lucy in 1977 for astrophysical problems
- Used (but not only) for simulation of fluid dynamics
- The fluid is represented by a particle system
- The key idea is to determine some particle properties by taking an average over neighboring particles (similarity with grid less interpolation methods)

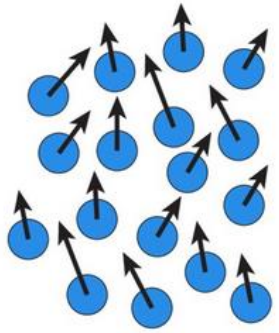


Grid-based Methods vs SPH

- Traditionally, numerical methods for hydrodynamics are based on discretization (spatial grid, inherently Eulerian)
  - Well developed technique
  - Easy to deal with boundary conditions

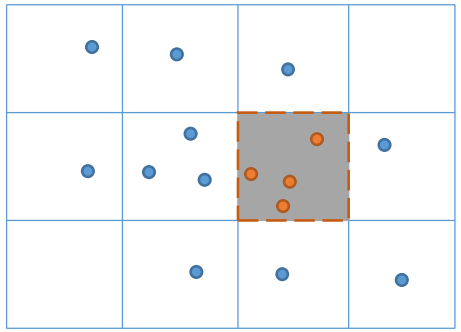


- SPH – discretization on mass (discrete fluid elements, inherently Lagrangian)
  - Resolution follows density
  - Difficult to implement boundary conditions

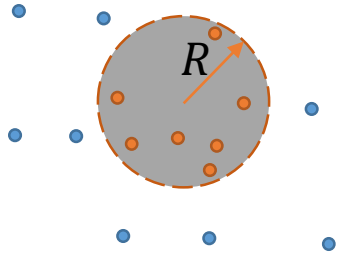


Fundamental Idea Behind SPH

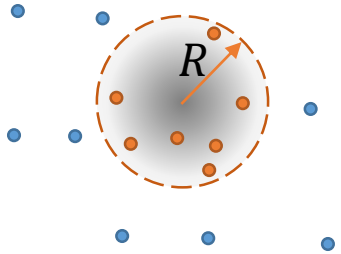
- How to compute density from a collection of point masses?



Method 1: construct a mesh around the points, then sum particles within cell and divide by cell volume



Method 2: construct local sample volumes, then sum particles within volumes and divide by volume



Method 3: weight contributions according to distance from sample point (SPH)

Particle Description

- Each particle is described by a list of its properties
  - Carried by particle: mass m_i (const.), position $\boldsymbol{x}_i$, velocity $\boldsymbol{v}_i$,
  - Computed: force $\boldsymbol{F}_i$, density ρ_i (varies), pressure P_i, color $\boldsymbol{C}_i$

  - These properties forms a particle state vector $(m_i, \boldsymbol{x}_i, \boldsymbol{v}_i, \boldsymbol{F}_i, \rho_i, P_i, \boldsymbol{C}_i)$

Density From Collection of Masses

- Density ρ_i at the point $\boldsymbol{x}_i$ is computed (approximated) as a weighted sum of N particle masses m_j as follows

$$\rho_i \approx \sum_{j=1}^{N} m_j W_{ij}$$

where the smoothing kernel

$$W_{ij} = W(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|, h)$$

and h is smoothing length

- This formulation ensures that the resolution follows density such that

$$\rho h^3 = const.$$

Smoothed Particle Interpolation

- If we know that $\rho_i = \sum_{j \in N_i} m_j W_{ij}$, how we can compute arbitrary (smoothed) physical quantity A_i of i-th particle?
- We can use a volume of i-th particle $V_i = \frac{m_i}{\rho_i}$ from which $m_i = V_i \rho_i$
- Now we get $\rho_i = \sum_{j \in N_i} V_j \rho_j W_{ij}$ and we have the same quantity on both sides of equation provided that $i \in N_i$
- It also holds that $V_i = \frac{m_i}{\rho_i} = \frac{m_i}{\sum_{j \in N_i} m_j W_{ij}}$
- Now, in general, for arbitrary quantity A_i, we can write the smoothed interpolation as follows

$$A_i = \sum_{j \in N_i} V_j A_j W_{ij} = \sum_{j \in N_i} \frac{m_j}{\underbrace{\sum_{k \in N_j} m_k W_{jk}}_{\rho_j}} A_j W_{ij}$$

Kernel Function

- Kernel function should approximate a delta function, i.e. particles which are closer should contribute more to the local evaluation of fluid properties
- First choice – Gaussian kernel

$$W(r,h) = \frac{1}{\pi^{1/3}h^3} e^{-(r/h)^2}$$

- One issue with this function is that the support is not compact and summation must be done over all particles
- One can choose a kernel with compact support (weight vanishes beyond a given distance)
- Better choice – Cubic spline kernel (neighborhood contains only particles lying within $2h$ distance)

Kernel Function

- In general, smoothing kernel can be any function which satisfies the following to properties
  - Normalization

$$\int_V W(r,h)\mathrm{d}V = 1$$

  - Dirac delta function approximation

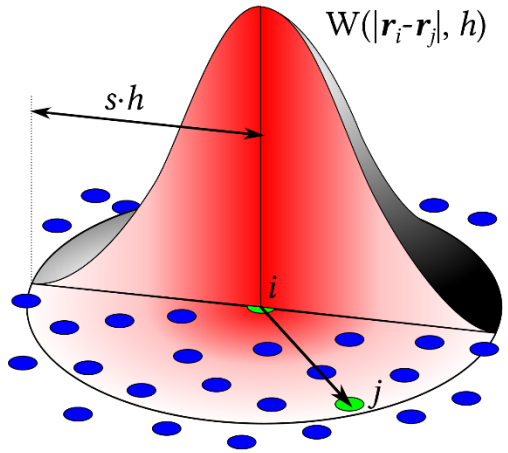
$$\lim_{h\to 0} W(r,h) = \delta(r)$$

Cubic Spline Kernel Function

- The smoothing kernel function is defined as follows

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

where $q = \frac{r}{h} = \frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{h}$



Arbitrary Quantity Derivatives

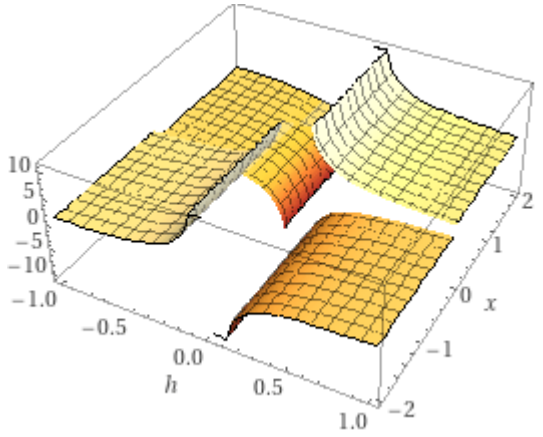
- Repeat that $A_i = \sum_{j \in N_i} V_j A_j W_{ij}$
- Gradient

$$\nabla A_i = \sum_{j \in N_i} V_j A_j \nabla_i W_{ij}$$

- Laplacian

$$\Delta A_i = \sum_{j \in N_i} V_j A_j \Delta_i W_{ij}$$

Kernel Gradient



- $\nabla_i W_{ij} = \begin{bmatrix} \frac{\partial W_{ij}}{\partial x_i} \\ \frac{\partial W_{ij}}{\partial y_i} \\ \frac{\partial W_{ij}}{\partial z_i} \end{bmatrix} = \frac{\partial W_{ij}}{\partial q} \nabla_i q = \frac{\partial W_{ij}}{\partial q} \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\| h}$

Note that $\frac{\partial}{\partial x}\left(\frac{|x|}{h}\right) = \frac{x}{|x|h} = \frac{x}{\sqrt{x^2}h} = \frac{\operatorname{sgn} x}{h}$

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

Kernel Laplacian

- $\Delta_i W_{ij} = \frac{\partial^2 W_{ij}}{\partial x_i^2} + \frac{\partial^2 W_{ij}}{\partial y_i^2} + \frac{\partial^2 W_{ij}}{\partial z_i^2} = \frac{\partial^2 W_{ij}}{\partial q^2}\frac{1}{h^2} + \frac{\partial W_{ij}}{\partial q}\frac{2}{h}$

- $\frac{\partial^2 W_{ij}}{\partial q^2} = \frac{3}{2\pi h^3}\begin{cases} -2+3q & 0 \le q < 1 \\ 2-q & 1 \le q < 2 \\ 0 & q \ge 2 \end{cases}$

Fluid Dynamics

- Govern by Navier-Stokes equation (momentum equation)

$$\boldsymbol{a} = \frac{\partial \boldsymbol{v}}{\partial t} = \underbrace{-\frac{1}{\rho}\nabla p}_{\boldsymbol{a}^P} + \underbrace{\frac{\mu}{\rho}\Delta \boldsymbol{v}}_{\boldsymbol{a}^V} + \underbrace{\cancel{\frac{1}{\rho}\boldsymbol{F}_{ext}}}_{\cancel{\boldsymbol{a}^E}} + \underbrace{\boldsymbol{g}}_{\boldsymbol{a}^G}$$

- It stems from motion equation in elasticity
- $\boldsymbol{v}$ is a velocity field, p is a pressure field, ρ is a density field
- Note: dynamic (absolute) viscosity μ and kinematic viscosity (momentum diffusivity) $\nu = \frac{\mu}{\rho}$

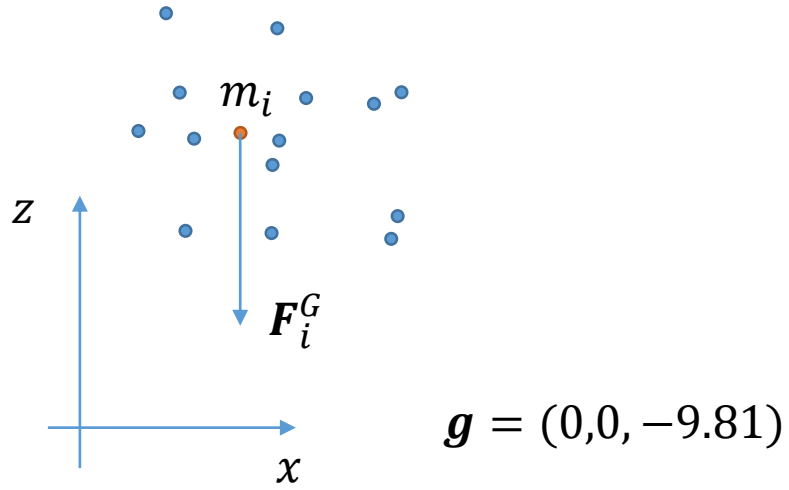
Fluid Dynamics

- Velocities and positions of particles are calculated from acting forces
- Three forces are applied on each particle (~~external force is excluded here~~)
  - Gravity force $\boldsymbol{F}_i^G = m_i \boldsymbol{g}$
  - Fluid pressure force $\boldsymbol{F}_i^P = -V_i \sum_{j \in N_i} V_j p_j \nabla_i W_{ij}$
  - Fluid viscosity force $\boldsymbol{F}_i^V = m_i \nu \sum_{j \in N_i} V_j \boldsymbol{v}_j \Delta_i W_{ij}$

Gravity Force

- The steady gain in speed of mass particle caused exclusively by the force of gravitational attraction is give by

$$\boldsymbol{F}_i^G = m_i \boldsymbol{g}$$



$$\boldsymbol{g} = (0,0,-9.81)$$

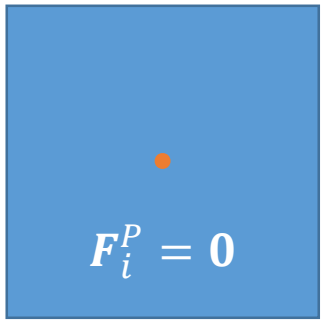
Fluid Pressure Force

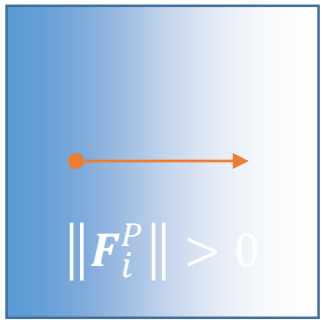
- We compute fluid pressure force induced by pressure term from NSE as follows

$$\boldsymbol{F}_i^P = m_i \boldsymbol{a}_i^P = m_i \left(-\frac{1}{\rho_i} \nabla p_i \right) = -V_i \nabla p_i = -V_i \sum_{j \in N_i} V_j \underbrace{p_j}_{\frac{p_i + p_j}{2}} \nabla_i W_{ij}$$

action ≠ reaction
symetrization needed

- Pressure force depends on the difference (i.e. gradient) of pressure
- There is no pressure force (i.e. no acceleration) in areas with constant pressure





Fluid Pressure Force

- Pressure at $\boldsymbol{x}_i$ can be computed via the ideal gas state equation

$$p_i = k(\rho_i - \rho)$$

where k is a gas constant that depends on the temperature and ρ is a rest (reference) density

- Here, we use Cole equation

$$p_i = B\left(\left(\frac{\rho_i}{\rho}\right)^{\gamma} - 1\right)$$

where B is tuneable gas constant with presure units and adiabatic index $\gamma \cong 7$

Fluid Viscosity Force

- We compute fluid viscosity force induced by viscosity term from NSE as follows

$$\boldsymbol{F}_i^V = m_i \boldsymbol{a}_i^V = m_i \frac{\mu}{\rho_i} \Delta \boldsymbol{v}_i = m_i \nu \Delta \boldsymbol{v}_i = m_i \nu \sum_{j \in N_i} V_j \boldsymbol{v}_j \Delta_i W_{ij}$$

Material density is constant in case of incompressible flow (i.e. resist volume change) but it is not absolutely true here

$\boldsymbol{v}_j - \boldsymbol{v}_i$ asymmetric again force depends on velocity differences only

- Viscosity causes loss of energy due to internal friction
- In viscous flow, particles should move together with the same velocity
- Resulting force is minimizing velocity difference between neighboring particles

Particles Position Update

- For every i-th particle compute force $\boldsymbol{F}_i = \boldsymbol{F}_i^P + \boldsymbol{F}_i^V + \boldsymbol{F}_i^G$ using its neighborhood set N_i
- Update the velocity $\boldsymbol{v}_i \mathrel{+}= \frac{\boldsymbol{F}_i}{m_i}\Delta t$
- Update the position $\boldsymbol{x}_i \mathrel{+}= \boldsymbol{v}_i \Delta t$

Kernel Variants

- Other various kernels were developed to improve numerical stability
- $W_{ij}(r,h) = \begin{cases} \frac{315}{64\pi h^9}(h^2 - r^2)^3 & 0 \le r < h \\ 0 & r \ge h \end{cases}$
- $\nabla W_{ij}(r,h) = \begin{cases} -\frac{45}{\pi h^6}(h-r)^2 & \begin{cases} \frac{(1,1,1)}{\|(1,1,1)\|} & 0 \le r < \varepsilon \\ \frac{\boldsymbol{x_i} - \boldsymbol{x_j}}{\|\boldsymbol{x_i} - \boldsymbol{x_j}\|} & \varepsilon \le r < h \end{cases} \\ \mathbf{0} & r \ge h \end{cases}$

Kernel Variants

- Other various kernels were developed to improve numerical stability
- $\Delta W_{ij}(r,h) = \begin{cases} \frac{45}{\pi h^6}(h-r) & 0 \le r < h \\ 0 & r \ge h \end{cases}$
- These kernels can be used in the following simplified formulas for computing accelerations from the original NSE
- $\boldsymbol{a}_i^G = \boldsymbol{g}$
- $\boldsymbol{a}_i^P = -\sum_{j \in N_i, j \neq i} \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) m_j \nabla W_{ij}$
- $\boldsymbol{a}_i^V = \frac{\mu}{\rho_i} \sum_{j \in N_i, j \neq i} (\boldsymbol{v}_j - \boldsymbol{v}_i) \frac{m_j}{\rho_j} \Delta W_{ij}$

Algorithm

Init particles ($m_i = m/n,\ \boldsymbol{v}_i = \mathbf{0},\ \boldsymbol{x}_i \in \text{cube}$)

For each time step:

  Init neighborhoods
  Compute densities
  Compute pressures from densities

  For each i-th particle:

    Compute acceleration $\boldsymbol{a}_i = \boldsymbol{a}_i^G + \boldsymbol{a}_i^P + \boldsymbol{a}_i^V$
    Update velocity $\boldsymbol{v}_i \mathrel{+}= \boldsymbol{a}_i \Delta t$
    Update position $\boldsymbol{x}_i \mathrel{+}= \boldsymbol{v}_i \Delta t$
    Check boundaries (prevent particle from leaving simulation domain)

kNN Radius Search

```
typedef std::vector<int> Neighbourhood;

Neighborhood SPHSolver::GetNeighborhood( const int i, const float r ) {
  Neighborhood indices;
  std::vector<float> distances;

  Vector3f x_i = particles_[i].position;
  std::vector<float> query{ x_i.x, x_i.y, x_i.z };
  cvflann::SearchParams search_params;

  int n = search_index_->radiusSearch(query,indices,distances,r*r,search_params);

  if ( n > 0 ) return Neighborhood( &indices[0], &indices[n] );
  return Neighborhood( 0 );
}
```

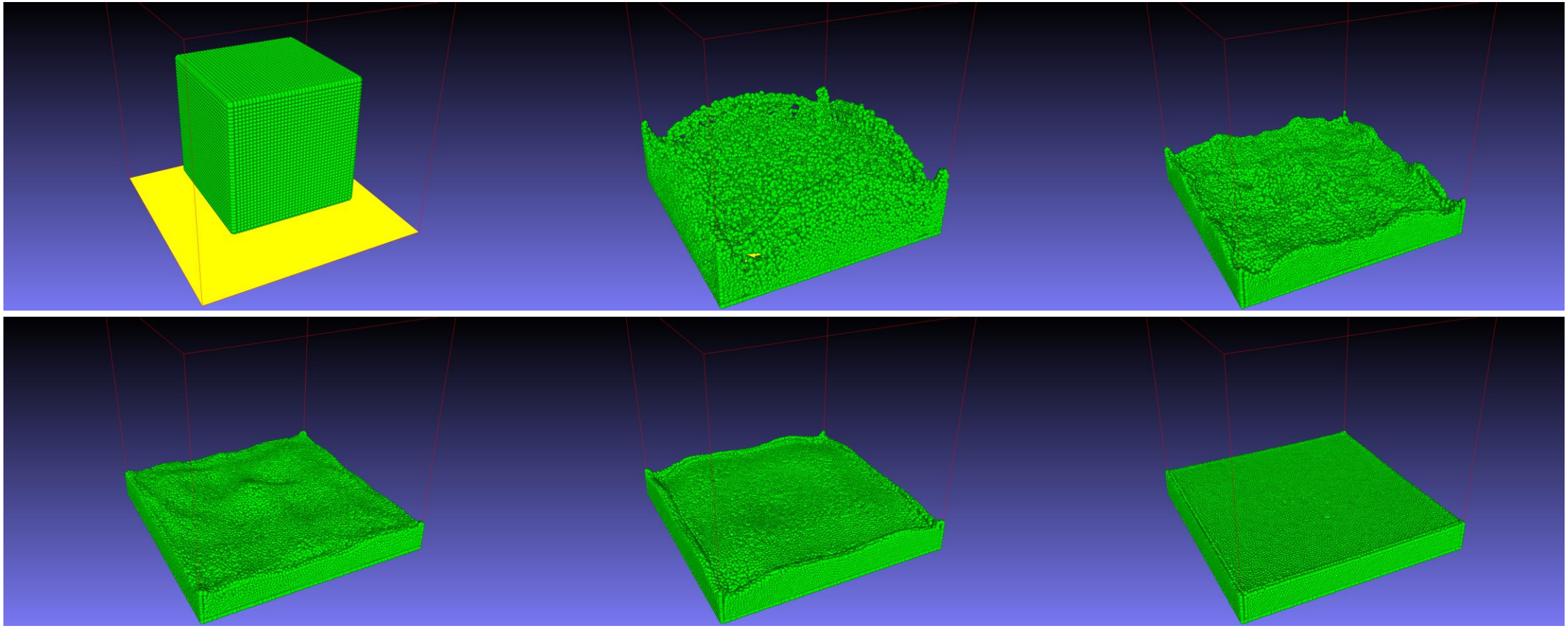
Search Index

```
typedef cv::flann::GenericIndex<cv::flann::L2<float>> SearchIndex;
int SPHSolver::InitSearchIndex() {
  cv::Mat features( n_, 3, CV_32FC1 );
  for ( int i = 0; i < n_; ++i ) {
    const Vector3f & position = particles_[i].position;
    features.at<float>( i, 0 ) = position.x;
    features.at<float>( i, 1 ) = position.y;
    features.at<float>( i, 2 ) = position.z;
  }
  search_index_ = std::make_unique<SearchIndex>( features,
    cvflann::KDTreeSingleIndexParams( 10, false ) );
}
```

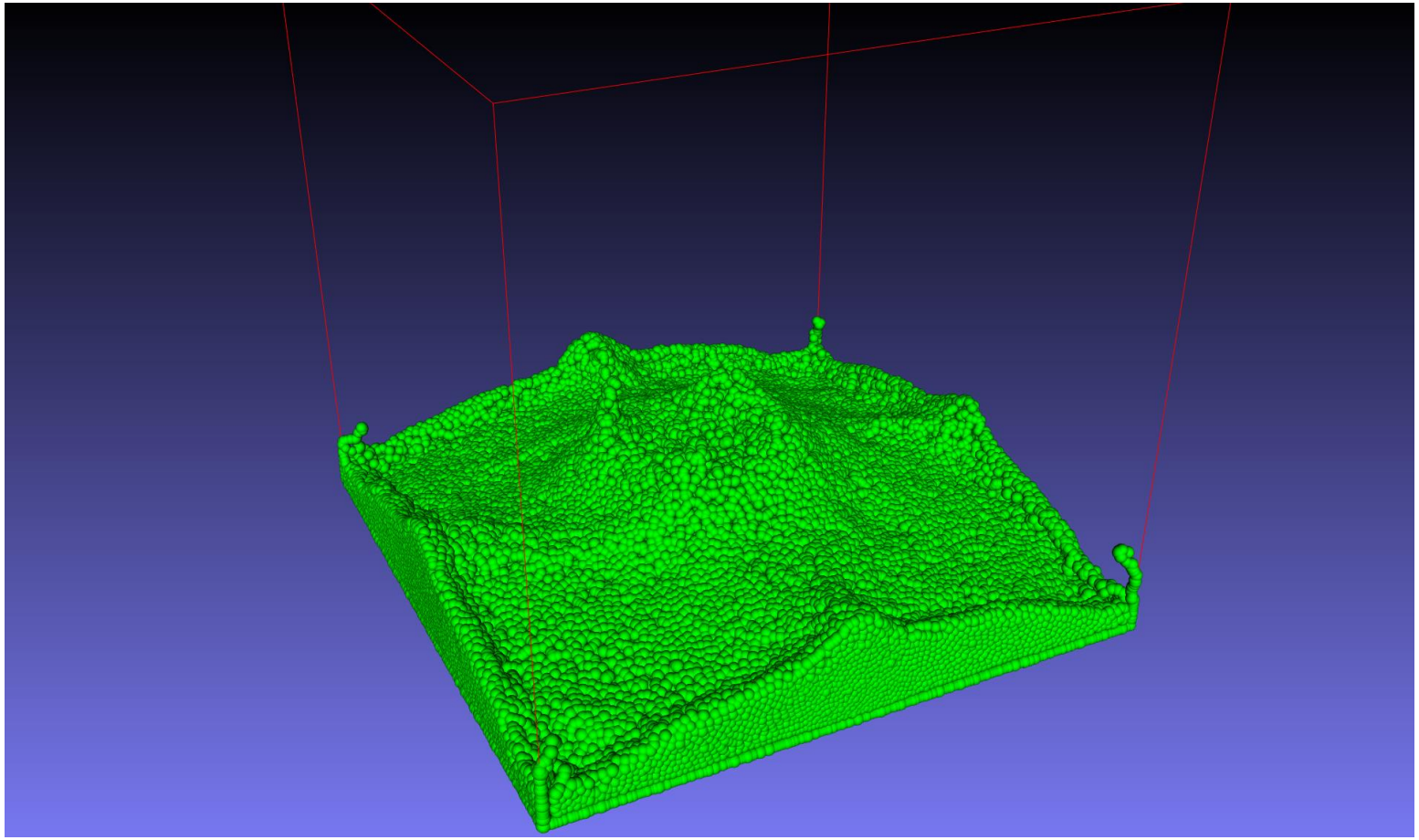
Parameters Settings

- $n = 40 \times 40 \times 40$ particles
- $\rho = 1000$ kg·m^{-3} (rest density)
- $\boldsymbol{g} = (0,0,-9.81)$ m·s^{-2}
- $B = 3.0$ (fluid stiffness)
- $h = 0.055$ m (smoothing length)
- Simulation domain size $1 \times 1 \times 1$ m
- Initial particle separation 0.018 m
- $\mu = 3.5$ N·s·m^{-2} (dynamic or absolute viscosity)
- $\Delta t = 0.001$ s

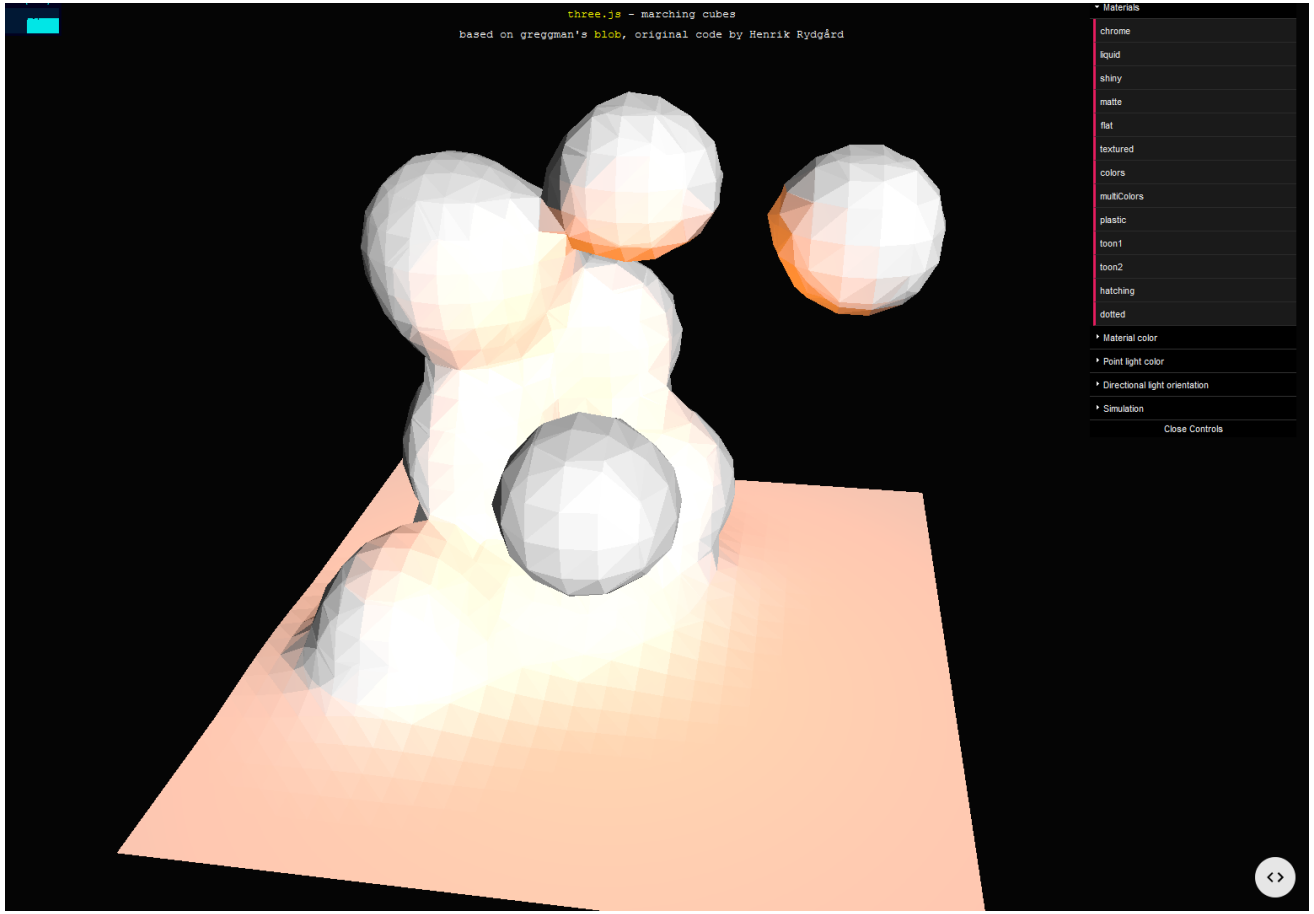
SPH Results



SPH Results



Iso Surface Reconstruction

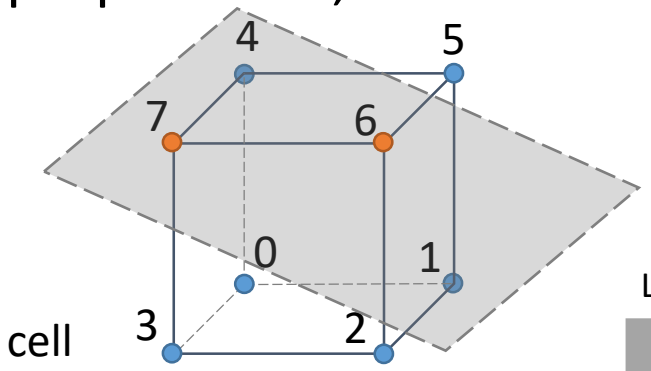


Marching Cubes

- Originally published in Marching cubes: A high resolution 3D surface construction algorithm". ACM SIGGRAPH Computer Graphics (1987)
- Topological issues fixes and further improvements were presented later
- The goal is to extract a polygonal mesh of an isosurface from the 3D discrete scalar field
- Elements of such 3D scalar field are called voxels (CT and MRI scans)
- In general, the algorithm determines the polygons needed to represent the part of isosurface passing by the give voxel (cube)

Marching Cubes

- 1. Assign a scalar value to each vertex of a cube
- If the scalar field value at the given vertex is bellow a certain threshold (iso-value), assign 0 to the appropriate bit, otherwise set this bit to 1

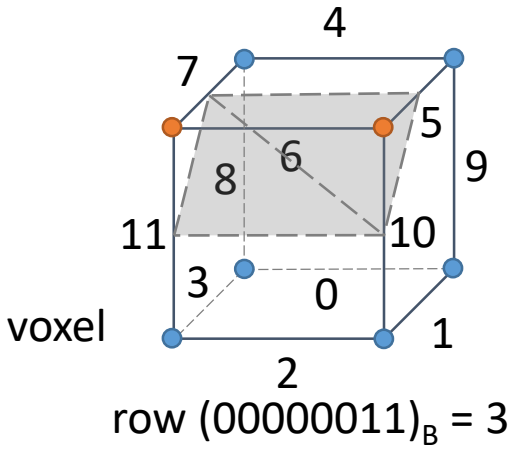


LSB							MSB
0	1	2	3	4	5	6	7
0	0	0	0	0	0	1	1

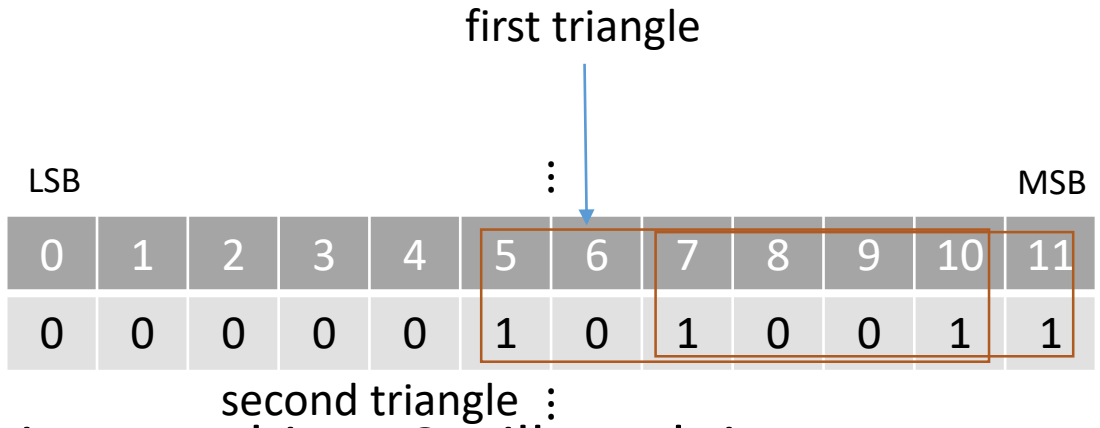
- In total, we get $2^8 = 256$ possible assignments (two states {inside, outside} in 8 vertices of a cube)

Marching Cubes

- 2. Based on step 1, we connect points on 12 line segments of the cube
- Some lookup table contains 256 entries of 12 bits representing connected mid-points on line segments



row $(00000011)_B = 3$



- Of the 256 different combinations 2 will not give any object, 8 will result in a triangle placed in every corner of the cube and so on

Marching Cubes

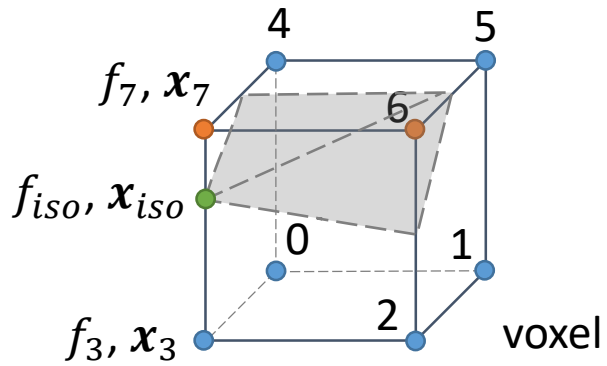
- 3. Set proper positions of all vertices $\boldsymbol{x}_{iso}$ on segments $\boldsymbol{x}_i$, $\boldsymbol{x}_j$ via interpolation
- Interpolation weights are derived from known functional values at selected vertices of given cube

$$\Delta f = \frac{f_{iso} - f_i}{f_j - f_i}$$

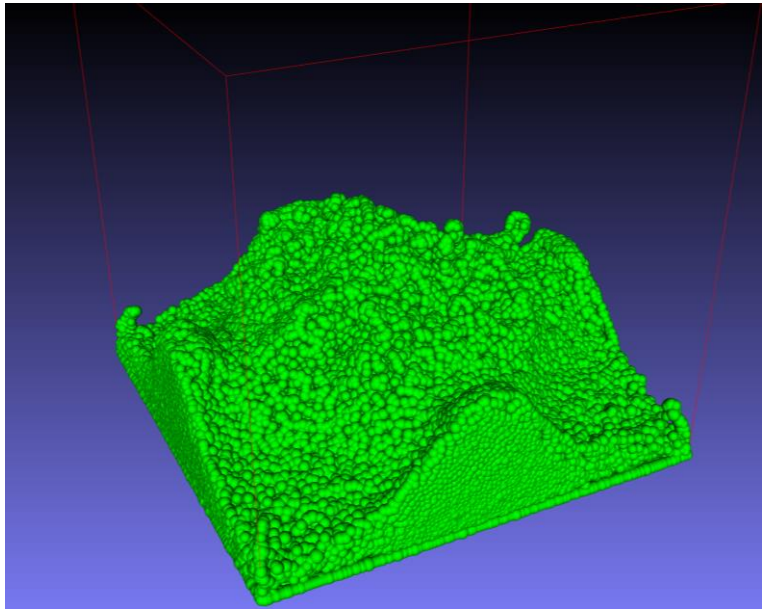
- Final point for each vertex of the triangle is computed as follows

$$\boldsymbol{x}_{iso} = \boldsymbol{x}_i + (\boldsymbol{x}_j - \boldsymbol{x}_i)\Delta f$$

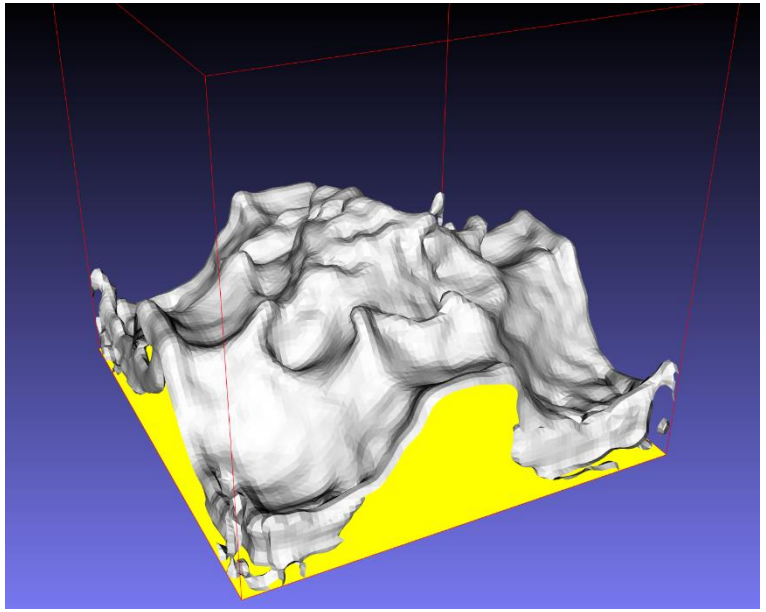
Used indexing: Note that vertex and edge indices may differ in a particular implementation, e.g. https://paulbourke.net/geometry/polygonise/

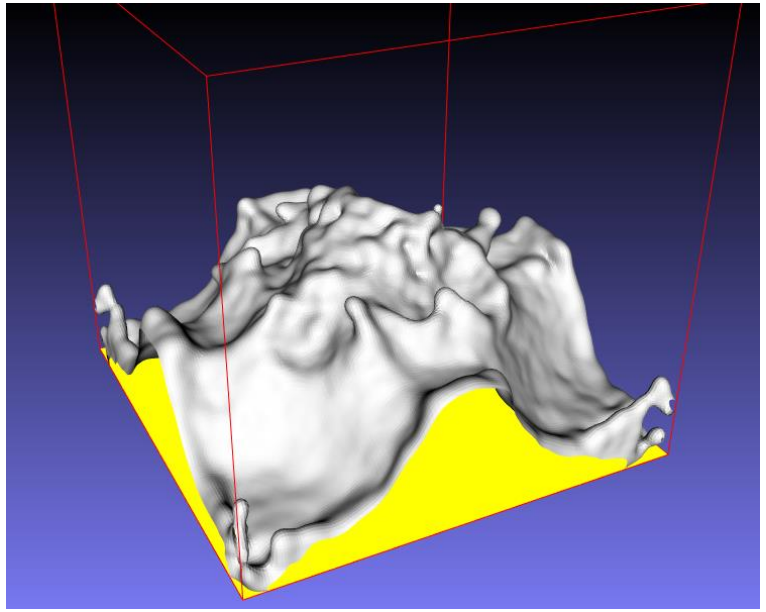


Marching Cubes Results



Original SPH particles





Marching Cubes iso-surface reconstructions with two different sizes of sampling steps producing different amount of details

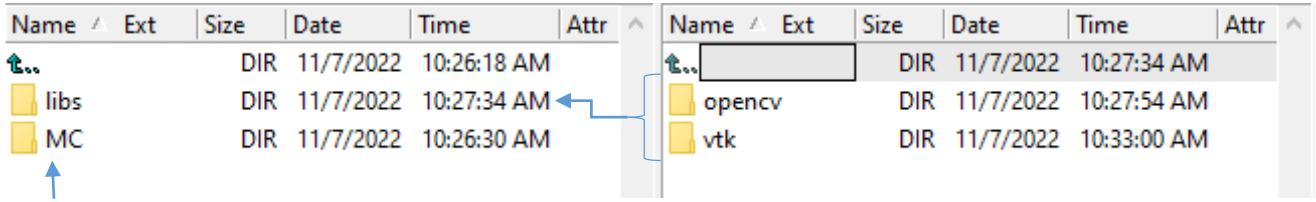
Note that presented images are captured at different times

References and Further Readings

- MÜLLER, Matthias; CHARYPAR, David; GROSS, Markus H. Particle-based fluid simulation for interactive applications. In: *Symposium on Computer animation*. 2003.
- IHMSEN, Markus, et al. SPH fluids in computer graphics. 2014.
- KOSCHIER, Dan, et al. Smoothed particle hydrodynamics techniques for the physics based simulation of fluids and solids. 2024.
- VIOLEAU, Damien; ISSA, Reza. Numerical modelling of complex turbulent free-surface flows with the SPH method: an overview. *International Journal for Numerical Methods in Fluids*, 2007.
- SUTTI, Marco. *SPH treatment of boundaries and application to moving objects*. École polytechnique fédérale de Lausanne. 2014.

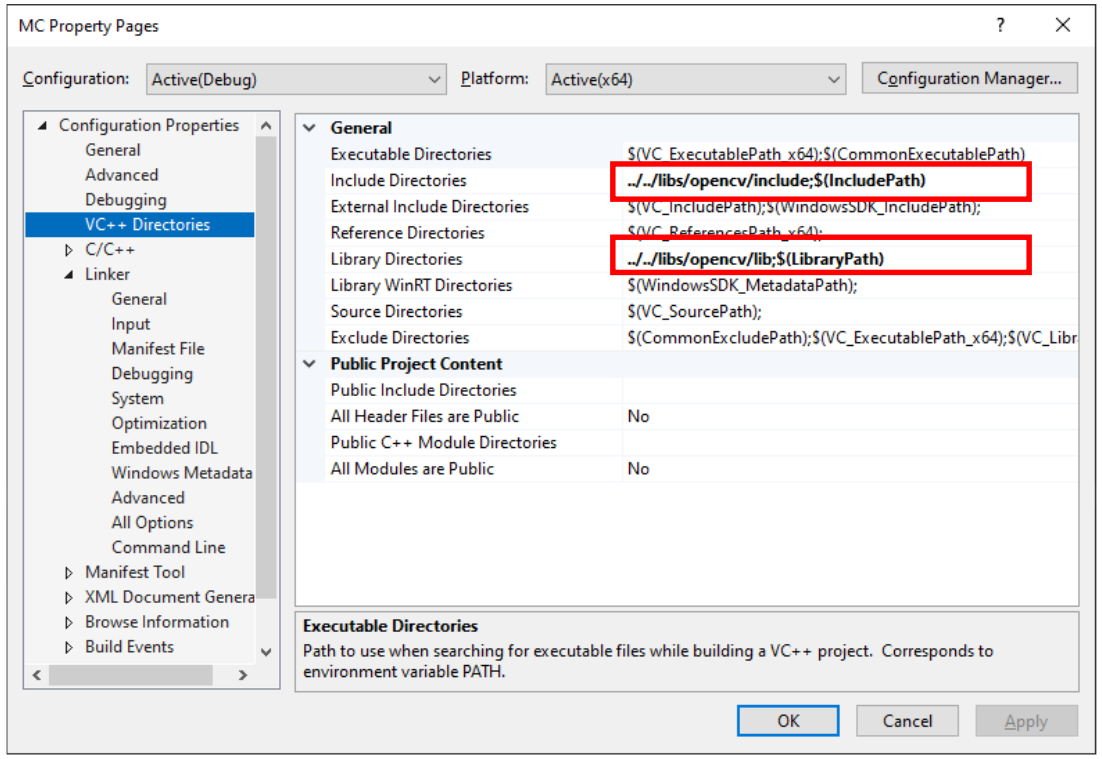
Technical Notes

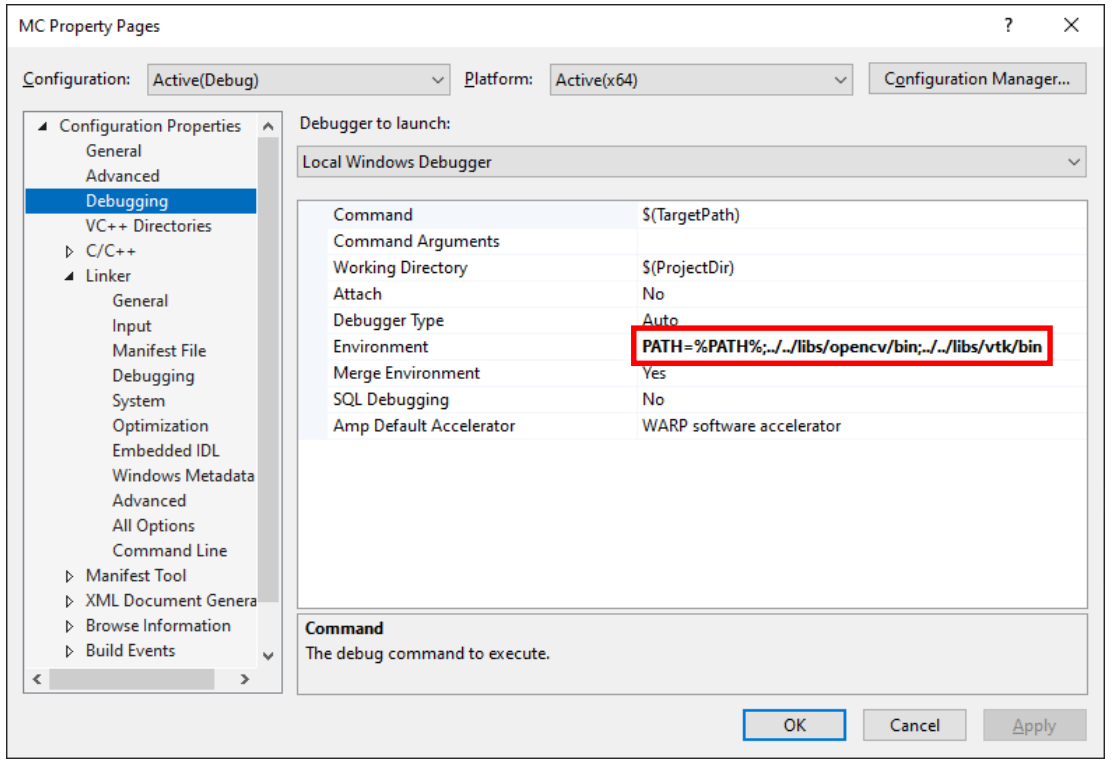
Expected directory structure:



Created VS solution

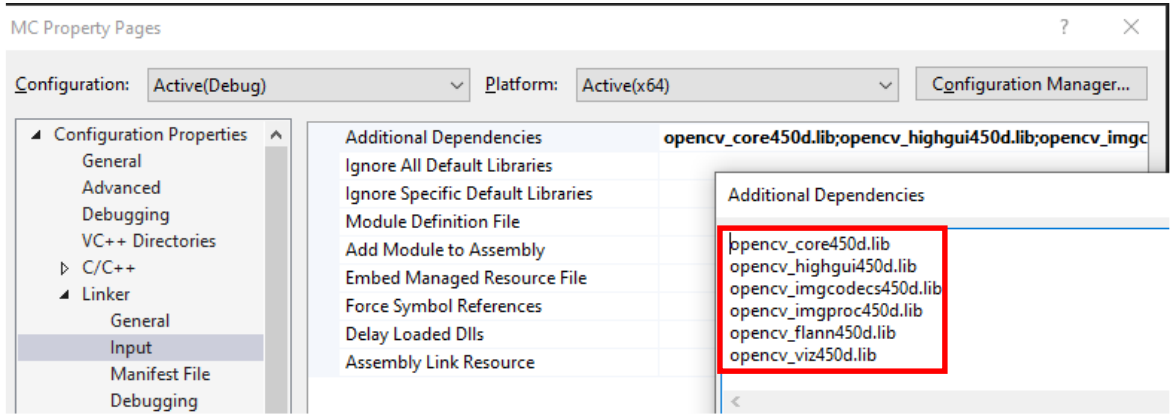
- How to set paths for OpenCV and VTK libraries…





Technical Notes

- How to set paths for OpenCV and VTK libraries…



Technical Notes

```cpp
#include <opencv2/opencv.hpp>
#include <opencv2/viz.hpp>

#include "half.hpp"
using half_float::half;

int test()
{
    cv::viz::Viz3d window("VD-SPH");
    window.spinOnce(1, true);

    cv::Vec3f cam_pos(1.0f, 2.0f, 1.5f), cam_focal_point(0.0f, 0.0f, 0.25f), cam_y_dir(0.0f, 0.0f, -1.0f);
    cv::Affine3f cam_pose = cv::viz::makeCameraPose(cam_pos, cam_focal_point, cam_y_dir);
    window.setViewerPose(cam_pose);

    cv::viz::WCube domain(cv::Point3d(-0.5, -0.5, 0.0), cv::Point3d(0.5, 0.5, 1.0), true, cv::viz::Color::red());
    window.showWidget("domain", domain);

    cv::viz::WPlane plane(cv::Size2d(1.0, 1.0), cv::viz::Color::yellow());
    window.showWidget("plane", plane);

    window.spinOnce(10000, true);

    half half_float_value = half(0.0f);

    return 0;
}

int main()
{
    return test();
}
```

Add half.hpp into your project from
http://half.sourceforge.net/index.html