Neutrino Documentation

Release 1.0

Ram

Mar 13, 2018

Contents

Contents:

CHAPTER 1

Neutrino User Guide

Neutrino is a general purpose simulation and visualization environment developed by Neutrino Dynamics Initiative. Neutrino Dynamics is composed of a group of diverse scientific research team distributed around the world. Neutrino currently provides research organizations, academia, various industries a full-featured simulation and visualization environment for the most demanding tasks and problems. Neutrino is currently in use in many diverse sectors, including energy research, risk analysis, flooding & bio fuel catalysis.

1.1 Requirements

1.1.1 Hardware Requirements

Neutrino runs on the following hardware

- PC/Mac with atleast 8 GB Memory, Intel/AMD processors
- HPC Cluster

1.1.2 Operating Systems

Neutrino currently runs on the following platforms more platforms are in the works of being supported

- Microsoft Windows 7/8/10
- Microsoft Windows 2012/2016 Server R1/R2
- CentOS 7.x +
- Ubuntu 13.x +
- SuSE Enterprise 11.x +
- Red Hat Enterprise Linux 7+
- Fedora Linux 16+

- OS X 10.9+

1.1.3 Essential Components

- **OpenGL 4.x Compatibility mode drivers.**
  - If OpenGL Software drivers (Mesa) are used - Mesa 11.x +

1.1.4 GUI Mode Requirements

Recommended Graphics Cards

- NVIDIA Kepler Architecture and above

Optional

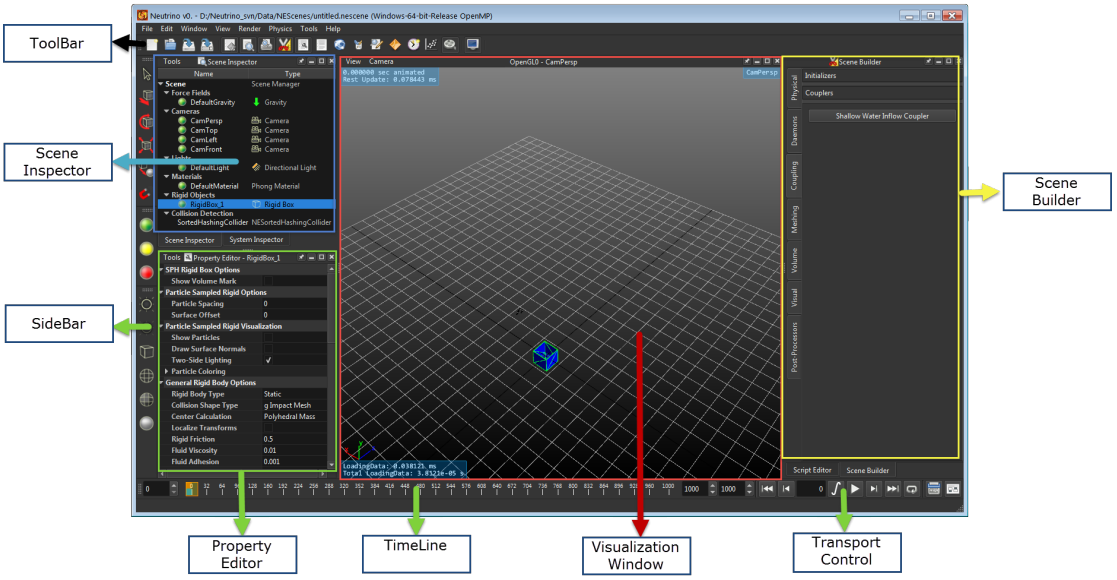
- CUDA
- OpenCL

1.2 Installation

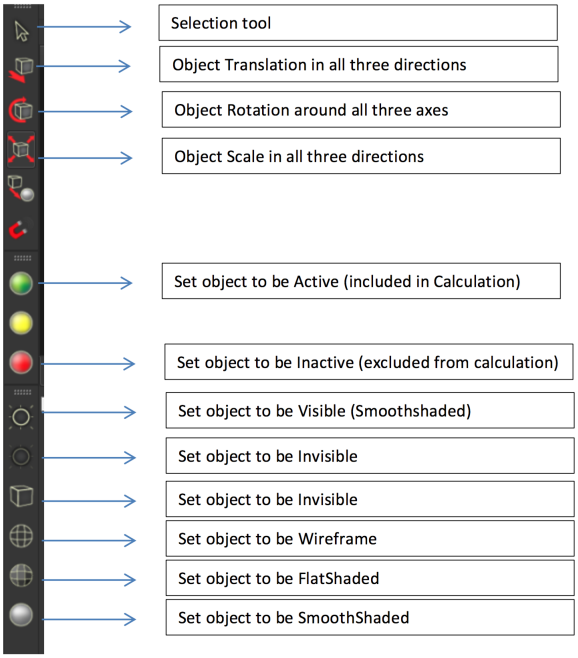
Please run the downloadable binary (.run file for Linux/OS-X platforms , .exe for Windows platforms) and specify the install path for the binaries. The rest of the setup is automatic.

1.3 Interface

Neutrino's user friendly interface enables easy setup of models for simulation. Parameters and operators can be added easily to perform the various operations. Interface items are depicted below.



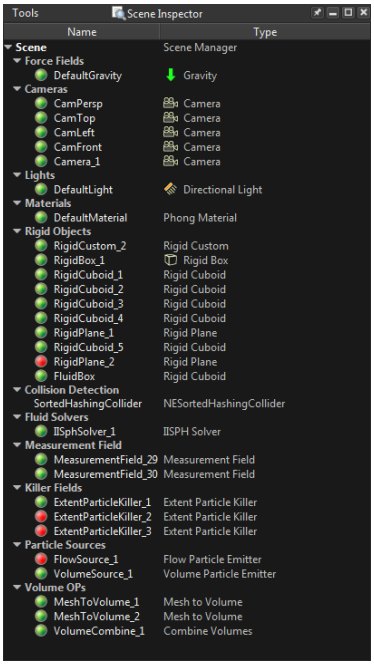
1.3.1 Toolbars



Scene Inspector

Description of Various Tools in Neutrino.

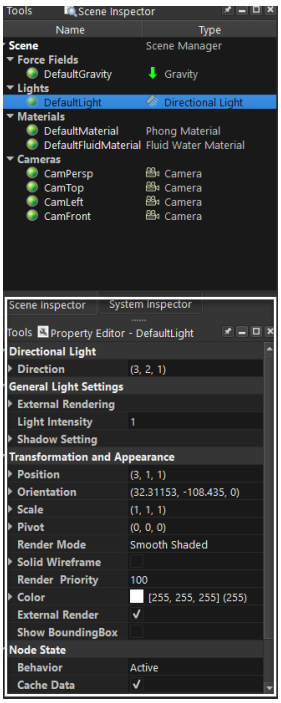
1.3.2 Scene Inspector



Scene Inspector

The Scene Inspector contains all the items in the scene which can be selected. The selected item's properties are displayed in the property editor. Context menu items appear as the right mouse button is pressed with the object being selected.

1.3.3 Property Editor



Property Editor

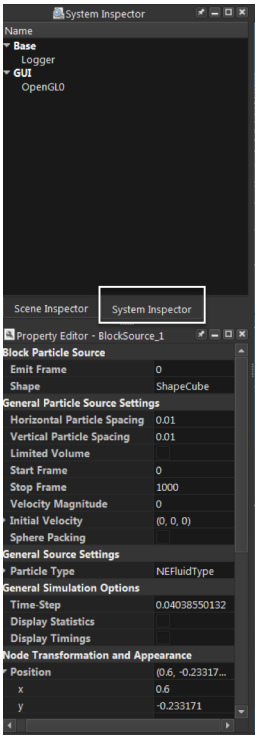
Property Editor is an user editable list of properties for the selected scene item. The properties of the rigid vary according to the type of item selected.

1.3.4 Scene Builder

Scene Builder

Scene Builder is a panel where entries in the scene can be constructed easily without navigating through the menu items. | The various categories of neutrino nodes are on the side shelf panel and upon selection they reveal the nodes. Nodes are created and placed in the scene upon click on that button.

1.3.5 System Inspector



System Inspector

System Inspector is a list of global system entries whose properties can be edited as well.

By default there are four(4) entries in the System Inspector. Selecting each of these entries in the Object Browser will reveal the properties of them. System Inspector are global entities during each session.

- Base

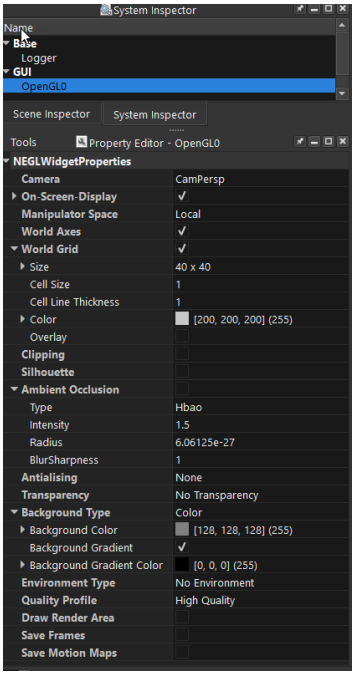
Parameter	Description
OpenMP Threads	Number of Threads Neutrino will use for Simulation (8)
Server Port	If Neutrino is controlled by external communication protocol the socket number (20200)
Scene Path	Specify the information needed to be exported.
Data Read Directory	Specifies the directory where all the caches for entries in the scene reside
Data Write Directory	Specifies where the cache data will be written onto
Optimize Net Caches	In certain cases setting this will optimize block writes onto network drives
Compress Fluid Caches	This will create fluid caches with compression but for large particle counts this might result in performance loss

- GUI

Parameter	Description
Timeline Units	Number of Threads Neutrino will use for Simulation (8)
Save OpenGL Frames	If Neutrino is controlled by external communication protocol the socket number (20200)

GL Properties

A Neutrino session can have several interactive views for visualization and model setup and the main engine to render them is OpenGL. Its properties are as follows



Rendering Properties .

Many of the global rendering parameters can be controlled using these. These include Ambient occlusion, transparency etc.

- OpenGL

There can be many OpenGL objects depending on how many views are present in the scene. By default this is 1. The views can be split according to user specification and the View menu described later controls the splits and usage of multiple visualization/model setup views.

The main properties control the render of each of these views.

Parameter	Description
Camera	The Camera used for rendering this view
On Screen Display	Whether or not to display statistics
Manipulator Space	Local Space operates with respect to the center of the object .
World Axes	The axes ON/OFF in the visualization window
World Grid	Whether a flat grid is drawn in the visualization or not
Clipping	Uses a normal for clipping objects - Deprecated in favor of a Clipping Object
Silhouette	Unused and deprecated
Ambient Occlusion	Turning this on produces self shadows and better looking renders (HBAO)
Antialiasing	The method used for improving quality of lines etc in the gl/visualization window
Transparency	The transparency method used - different modes are a trade-off between quality/time
Environment Type	The background image used in visualization (either cube 6 sides or sphere
Quality Profile	The render quality used for preview/visualization. Some features are turned off/on for
Draw Render Area	Not used
Save Frames	Save the frames displayed in this specific visualization window/GL to file (png)
Save Motion Maps	DEPRECATED

1.3.6 Timeline

Timeline

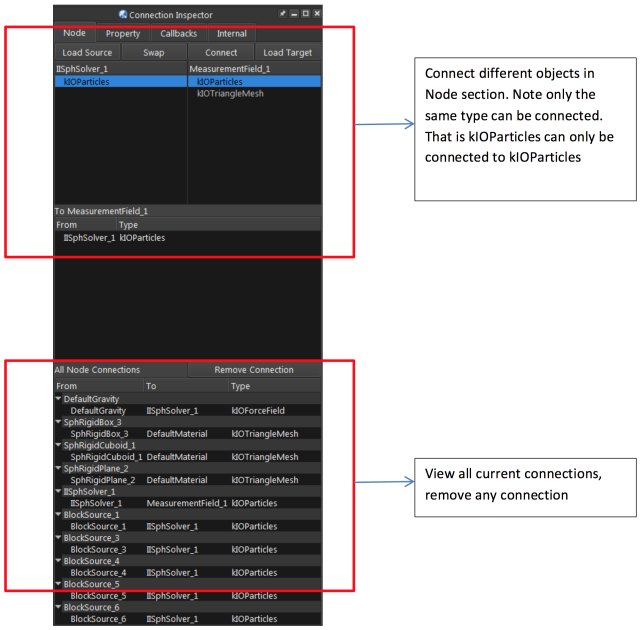
TimeLine, used to record and playback the simulation.

The two modes are Simulate and Playback. Simulate mode is activated by the button. The play button is then used to either simulate (Simulate Mode) or Play (Playback Mode).

Turns Green when armed and ready to simulate Simulation is Armed to proceed. Next Play would actually Simulate the scene.

Run simulation if Armed or just playback the recorded simulation from caches if not armed.

1.3.7 Connection Inspector



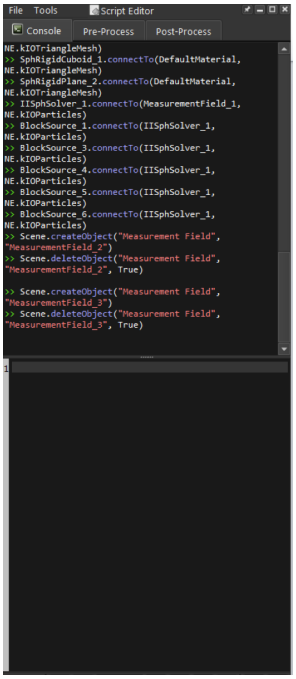
Connections

View and Modify input/output connection to/from nodes. Connection Inspector is used to connect various inputs and outputs from nodes to process during simulate/playback mode. The emitter's creates Particles (kIOParticles)

which is connected to the solver's input. Solver's output (kIOParticles) can be connected to the Measurement Field's input (kIOParticles).

In Neutrino you can consider all the objects displayed in the Scene Inspector to have inputs and outputs and they are evaluated during a simulation. For example the output of a particle emitter would be particles (kIOParticles) and that is automatically connected to the input of a solver (kIOParticles) . Some connections are automatically made for example in this case. But for some other cases the user has to make the connections manually depending on what the scenario is. If the particle emitter's kIOParticles to the solver connection is deleted then there would be no particles input to the solver and hence no solve. Neutrino node's most common inputs/outputs are the following kIOParticles, kIOTriangleMesh kIOVolumes

1.3.8 Script Editor



Script Editor

Python based command window.

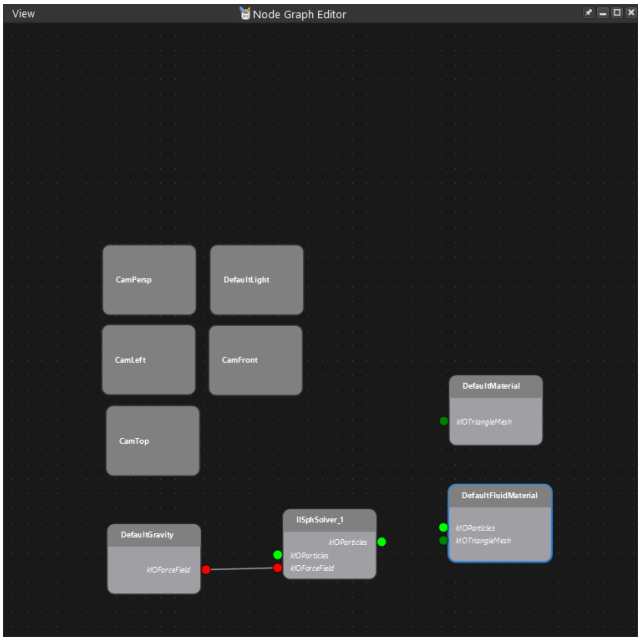
Script Editor Commands

Script Editor could be used for performing any action in Neutrino but through the python interface. The python interface to Neutrino's system is explained in a separate section.

1.3.9 Node Graph Editor

Node Graph Editor, used to visualize various connections. It can be seen clearly that all fluid objects go to IISPHSolver and measurement field takes data from IISPHSolver to determine the parameter we are interested in.

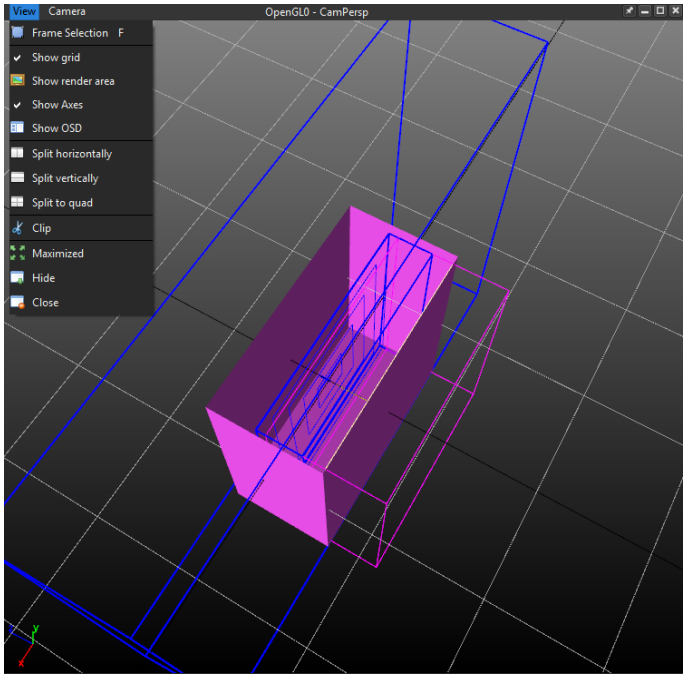
Currently its still under works and not recommended for use.



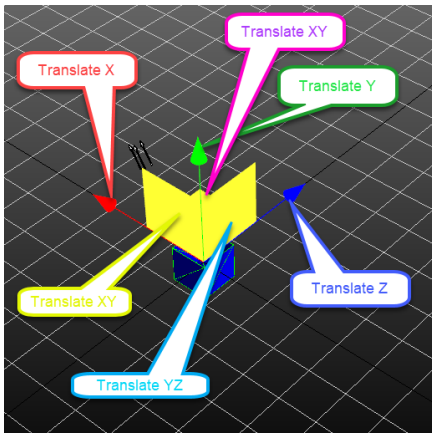
1.3.10 Interactive View Window

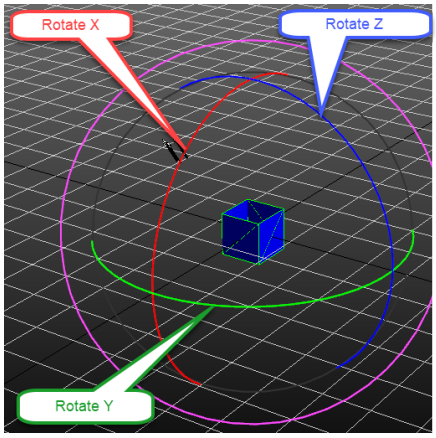
Objects can be selected, transformed in the world space visible throught the selected camera in the GL Widget View. There can be many such views which can be created and deleted in the scene and they are synchronized with each other.

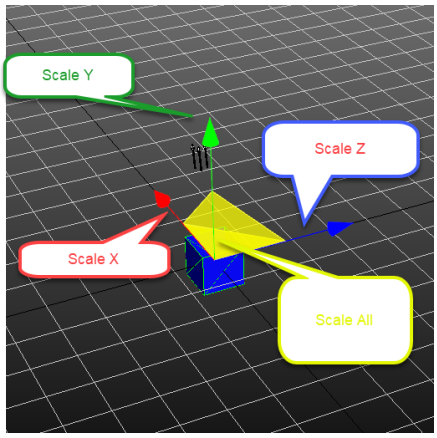
Each view can be split into 2 duplicated and assigned different cameras on the View Menu on top of this window



If you select an object and select the translate (toolbar) on the left side a translate handle appears in the gl preview window. you can select each of the arrows in the axes and move them around and that moves the object. Same goes for scale and rotate, their tools are in the left toolbar.







The camera for the view is selected from the Camera Menu item at the top of the View Window

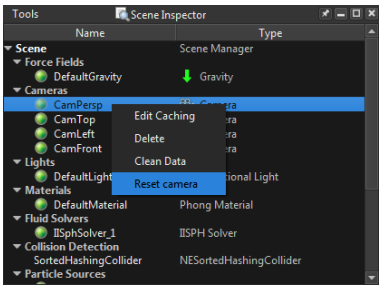
Camera

Alt and mouse will turn the camera around

Alt + middle will pan the camera

Alt + Right will zoom in.out

If a camera gets stuck somewhere you can reset it by selecting the camera in the scene builder and third mouse button and reset camera.



1.3.11 Neutrino Nodes

Neutrino nodes are entities which can be created in Neutrino which exist in the scene and their properties could be modified by the user. They can be connected/created/deleted etc.

Common to all Neutrino nodes are the following properties

Property	Description
Behavior	Active: Participate, Cache: Just Load Data, Inactive: Dont participate
CacheData	Whether data for this node should be cached/not

1.3.12 Spatial Nodes

Spatial nodes are nodes in Neutrino which can be positioned and modified in Space and exist in 3D/2D space in the view. Examples of Spatial Nodes include Rigid Nodes, Cameras, Lights Fields etc. For some of these nodes positions matter and some others they dont.

The common properties to spatial nodes are the following

1.3.13 Rigid Nodes

For Model setup and 3D Geometry various rigids can be imported into Neutrino which can then serve as active boundary conditions for the simulation.

Edit $\rightarrow$ *Add* $\rightarrow$ *Physical* $\rightarrow$ *Rigid*

performs this task.

Or the scene builder could be used to accomplish this.

Neutrino accepts various types of implicit and explicit geometry. The basic ones are 1. Box 1. Cuboid 1. Sphere 1. Cylinder

However external geometry modeled in another package such as strata, Sketchup could be imported by

Edit → Add → Physical → Rigid Custom

or through

Scene Builder → Physical → Rigid → Rigid Custom

Neutrino currently supports the following subset of file formats

COMMON INTERCHANGE FORMATS

1. Autodesk (.fbx) 1. Collada (.dae) 1. Blender 3D (.blend) 1. 3ds Max 3DS (.3ds) 1. 3ds Max ASE (.ase) 1. Wavefront Object (.obj) 1. Industry Foundation Classes (IFC/Step) (.ifc) 1. XGL (.xgl,.zgl) 1. Stanford Polygon Library (.ply) 1. AutoCAD DXF (.dxf) 1. LightWave (.lwo) 1. LightWave Scene (.lws) 1. Modo (.lxo) 1. Stereolithography (.stl)

Future extensions are planned as to import various architectural and engineering formats such as DWG and STEP

The Property editor of the new rigid object added has a file name property to import the rigid geometry.

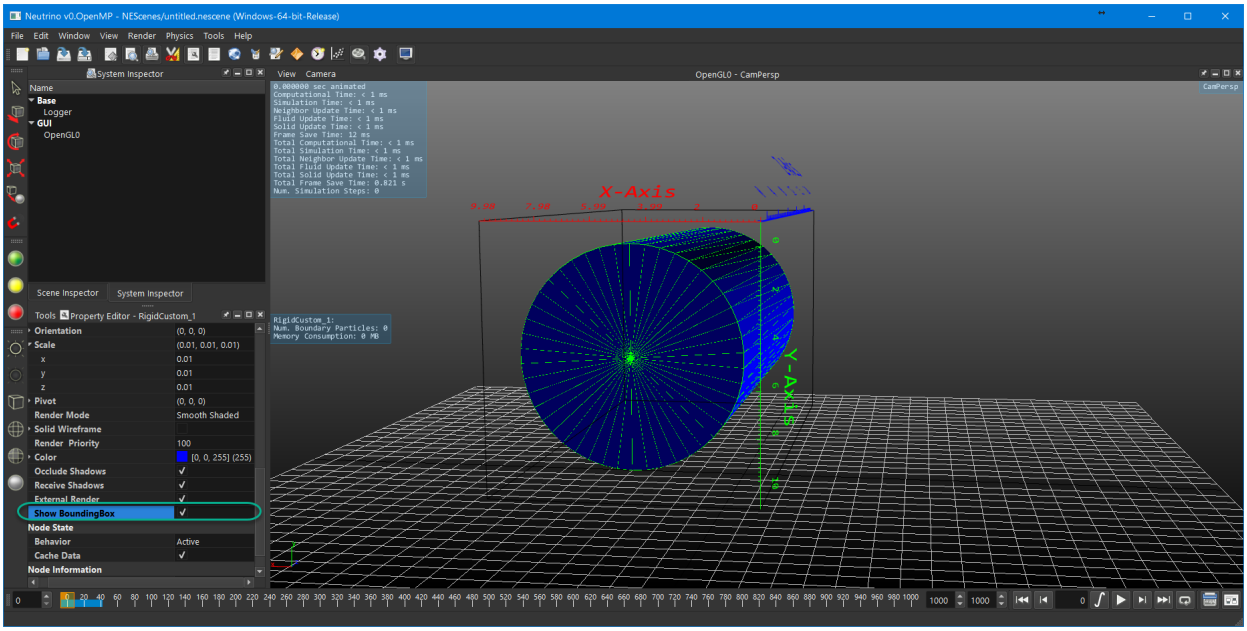
When Rigid Custom is used with STL file, there might be a difference in scale in which the imported geometry might have to be scaled by 0.01 on all axes to match the units or appropriate unit conversions need to be performed from the STL modeling units since Neutrino just assumes the units in the input STL file as meters.

Import of Rigids creates a rigid body seeded with rigid SPH particles.

Some of the properties, like viscosity, adhesion, density and so on, can also be modified easily in the scene inspector window. The rigid body can either move with the fluid (dynamic) or stay at constant position (static).

Bounding Box

The bounding box display can be activated by



1.3.14 Emitter Nodes

Various particle emitters that emits particles can be created using

Edit → Add → Daemons → Emitter

Please note For emitters to emit particles a solver needs to be present.

Planar Emitters

Flow Particle emitter

Flow emitter is a particle emitter rectangular in shape which will emit particles at a constant flow as specified by the flow rate in properties or by a time varying input file.

The rate is specified as part of the properties. If there is a time dependent file specifying various flow rates at times that can be loaded as well from the file parameters.

The flow input file of Flow Emitter has the following format: time, flow rate, width, height; within a line, all are separated by a comma. The file name extension must be ".csv". An example of such a csv file can be Downloaded at `FlowRate.csv`

The velocity is calculated from the flow rate, width and height. Only flow in the normal direction to the emitter plane can be created.

Edit → Add → Daemons → Emitter → Flow Particle Emitter

The properties of the Flow Particle Emitter are as follows

Property	Description	
Shape	Number of Threads Neutrino will use for Simulation (8)	
Flow Setting	Flow Rate Mode	
	Velocity is controlled by rate	
	Flow Rate	Flow Rate in m^3/s of the fluid injected into the system
	Speed Mode	
	Magnitude of velocity Setting, Flow rate is adjusted	
	Flow Rate & Speed Mode	
	Area is adjusted appropriately	
	Flow Speed	Magnitude of the velocity in m/s of the flow
	Pipe Pressure Mode	
	Flow based on static pressure	
	Upstream Pressure	
	Downstream Pressure	
	Particle Data Mode	
	Input can be time dependent number of particles injected	
	Flow Data Mode	
	Time dependent input of flowrates vs time as described	
	Data File Mode	
	Flow rates in a data file (csv)	
Connected Inputs	The Input connections coming into this node	
Advanced Properties		

Torricelli Particle Emitter

Torricelli Particle Emitter is an emitter which models a hydrostatic cuboid source such as a container of fluid enclosing it and a hole punched in through it.

Torricelli particle emitters can be coupled by laying several in a row and by connecting output kIOParticles of one to another to serve as input.

The Torricelli Particle Emitters can be created from

Edit → Add → Daemons → Emitter → Torricelli Particle Emitter

1.3.15 Volume Fluid Emitters

The following section describes volume emitters. This operates a bit differently than the other emitters and is usually used when needing to fill containers with fluid. Rather than use dynamic emitters which take a certain amount of time to fill a container with fluid and settle down, the following emitters in Neutrino could be used as a starting point.

1.3.16 Block Emitter

Block Emitter fills a closed rigid object like a cuboid with fluid. To create a block emitter

Edit → Add → Daemons → Emitter → Block Particle Emitter

1.3.17 Volume Emitter

We can also build fluid bulk with imported closed geometry by converting the rigid body into fluid particles. The interior of the geometry could be filled with particles.

This is useful for example, if a container of an arbitrary shape needs to be filled with fluid.

Several boolean operations could be performed and daisy chained together using the volume tools in Neutrino to create a desired container of fluid to be filled with. The emission can be started/stopped at a given time/frame of simulation by the node parameters.

The various volume nodes which can be used are described in the next section.

First the rigid object needs to be converted into a signed distance field (volume) by adding Mesh to Volume

Edit → Add → Volume → MeshtoVolume

Several cases can be considered after this step

1. Basic Container fluid emitter

Volume Particle Emitter

Add → Daemon → Volume Particle Emitter

and finally making the appropriate connections.

Rigid Objects kIOTriangleMesh (output) is connected to MeshToVolume's kIOTriangleMesh (input) then MeshToVolume kIOVolume (output) is connected to Volume Particle Emitter's kIOVolume (input)

Firstly a mesh can be converted to a volume . When you import a rigid object or create one in Neutrino - one of the output of that is kIOTriangleMesh and when you create a MeshToVolume node in Neutrino - one of its inputs is kIOTriangleMesh. So a Rigid Objects output kIOTriangleMesh could be connected to input of MeshToVolumeNode kIOTriangleMesh. The output of MeshToVolume is kIOVolume . This output would be connected to a Volume Emitter daemon (kIOVolume). and the output of that is kIOParticles which is automatically connected to the solver.

So when a simulation is run in this case the mesh is sampled into a volume filed on the inside then particles are instantiated from the volume and then sent to the solver.

The volume nodes are explained below.

1.3.18 Volume Nodes

Volume nodes in Neutrino are a set of nodes which have an input or an output as kIOVolume. The volume output from Neutrino are OpenVDB volumes. OpenVDB is a sparse volume representation. These nodes can be created by

Edit → *Add* → *Volume*

MeshToVolume

This node accepts kIOTriangleMesh as input and sends kIOVolume as output. The input mesh can be an open or closed mesh. In case of an open mesh, only the boundaries are converted into a volume.

PartToVolume

This node accepts kIOParticles as input and sends kIOVolume as output. The particles are converted into an implicit surface by this node.

VolumeFilter

This node accepts kIOVolume as input and outputs kIOVolume. Several topological operations are availabe under this node. Erode, Dilate, Open, Close etc.

VolumeCombine

This node accepts one or two input kIOVolume inputs and outputs a combined kIOVolume. Some of the operations available under this node are Union, Intersection, Sum, Difference etc. This is a node in which the connection order matters as the operation is performed on the first node by the second node.

1.3.19 Particle Killer Nodes

There are different kinds of killers . Killers can be used to save computational time by removing particles outside the domain or for other purposes like teleporting and/or transporting particles through obstacles etc. The particles killers can be added by

Edit → *Add* → *Daemons*

Extent Particle Killer

Particle Killer daemon eliminates particles inside or outside a specified bounding box. and all particle entering the region of particle killer will be eliminated.

Flow Particle Killer

Just a brief guide on the Flow Particle Killer, its differences compared to the Teleport Particle Killer, and how to use it with the Flow Particle Emitter altogether.

The Flow Particle Killer is a killer that kills the particles crossing its finite plane section; compared to the Teleport Particle Killer, it does not kill all the particles within the other side of the infinite plane. An outflow boundary is simulated using ghost particles when the permeable mode is activated. There is no need of a Ghost Volume. It also includes a flow rate computation, and the corresponding value can be read.

You can connect a Flow Particle Killer to a Flow Particle Emitter. The connection comes in two modes: flow rate mode and teleport mode. The former is realised if kIOData is the input-output type, and the latter in the case of kIOParticles. You can check the connection mode of the emitter in its Property Editor window.

In the flow rate mode, the emitter generates particles according to the flow rate computed by the killer. The advantage of this mode is that the killer and emitter scales do not need to be the same. It currently tends to be unstable and does not conserve mass that well; use with caution. I might test a few things to have it working properly in the future.

In the teleport mode, the emitter generates the particles as if they teleported from the killer. This mode requires the dimensions of the killer and emitter to be the same; prefer it to the flow rate mode when applicable.

Finally, you can specify a delay between the killing and emission through the Property Editor of the emitter.

Coupler Nodes

Inflow Coupler

Inflow shallow water (SW) coupler is now functional.

Description

Given the topology of the inflow boundary and SW data at gauge positions, the inflow SW coupler generates particles accordingly, enforcing inflow boundary conditions (Dirichlet for the velocity and Neumann for the pressure) and mass conservation, and also avoiding the particle deficiency issue of SPH near boundaries.

Method

The boundary topology is defined by the positions of the gauges and extremity points and the connections between gauges/extremity point. If the boundary is assumed to be along a rectangle, the connections are automatically computed. The boundary is partitioned into several “sources”, one for each pair of gauges or gauge/extremity point. A source generates particles with a given inflow velocity and height at a time. Interpolations in space and time are performed for Enforcement of the boundary conditions is realised through the use of ghost particles, with velocity prescribed according to the SW data and pressure extrapolated from the neighbouring fluid particles. The ghost particles are generated as far from the boundary as needed in order to avoid the particle deficiency issue. Minimal deviation from mass conservation is ensured with a particle generation minimising the deviation from a rest density, Cartesian grid configuration within a volume defined by the source width, inflow height and velocity, provided by the SW data, and time step.

Assumptions

1. The gravity is in the y-direction, while the fluid displacement is in the x- and z- directions.
2. The flow direction is perpendicular to the boundary.
3. The boundary topology can be defined as a set of simple, connected, undirected graphs with vertex degrees not higher than two.
4. The ground floor is flat at the boundary location.

1.3.20 Particle Fluid Solver Nodes

Setup

Fluid solver is key to SPH simulation, particles movements and key properties are determined the solver. The Solver can be added by

Edit → *Add* → *Physics* → *Particle Fluid* →

or through the scene builder by

Physical → *Particle Fluid* →

Implicit Incompressible Solver Nodes

Neutrino IISPH Solver

There are many key parameters of simulation that can be easily changed in Neutrino Solver.

Table : Key factors of IISPH Solver and how they can be changed in specific simulation conditions

Factors | Choices & Comments	
Integration Scheme	Verlet (second order integration)/Euler Cromer
Viscosity Model	Laminar (a more physical model)/ Artificial
Solver tolerance	Max. compressibility allowed as a percentage
Max. Iteration	Max number iterations steps to calculate pressure Usually set to 100 but convergence is reached in usually < 20 more than 20 iterations is usually a sign of slowness/prob
Max. Time Step	Max. unit time step allowed defaults to 0.02 but can be increased to more if blowups and high velocity projectiles/splashes are not encountered
Viscosity	Fluid-fluid viscosity
Cohesion/Adhesion Coefficient	Surface tension/wettability coefficient
Particle Interaction Radius/Particle Rest Distance	Any change here will be automatically synchronized elsewhere Interaction Radius is usually 2 times interparticle distance if cubic spline kernel is used Interaction Radius is modifiable and rest distance will be set automatically based on the kernel and other factors This is the main parameter indicating resolution of simulations
Kernel Model	Cubic Spline (Quintic and Weindland kernel support planned
Particle Visualization	Different parameter-base coloring; Max./Min. coloring limit

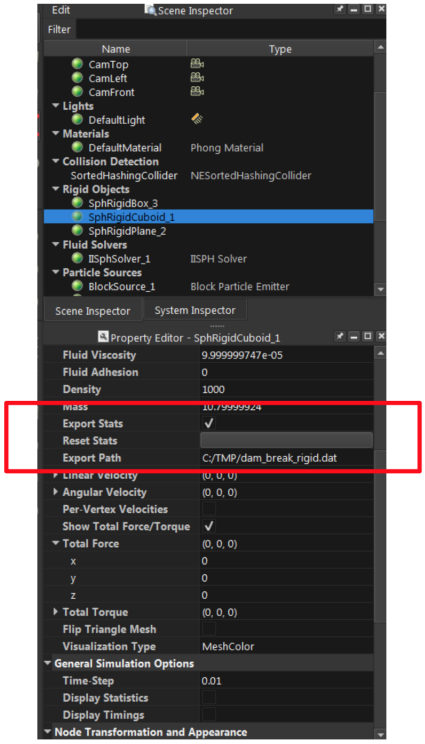
1.3.21 Neutrino WCSPH Solver Node

Neutrino's Weakly Compressible SPH Solver can be created by NWCSPH Solver. In certain cases of gas flows/wind fields and non-isothermal flows

1.3.22 Measurement Field Nodes

Sometimes, we would like to measure some parameters in certain area of interests, in Neutrino additional measurement can be set up onto any position by Add -> Daemons -> MeasurementField and moving the field to desired position. The volume enclosed by the measurement field will be taken into consideration for measurement.

Rigid Force Measurement and Data Export



Measurement

Measures Various Quantities of fluid and rigids in the simulation.

To export the force applied by the fluid onto a rigid, do the following:

1) Before simulating, turn on Show Total Force/Torque, to force the solver performing the fluid->solid force computations. This step is not necessary if the rigid object is dynamic.

2. When your simulation is done, first turn on Export Force and enter a path to Export File Path, then playback the simulation.

3) The force data can be found under the repository Measurements. Let me know if this works for you. You could also export the pressure and friction components of the force by turning on Export Force Decomposition.

Also in the rigid body Compute Forces/Compute Torque has to be checked to be able to check the forces on the rigid.

The output file contains a header that specifies what are the units. Something like:

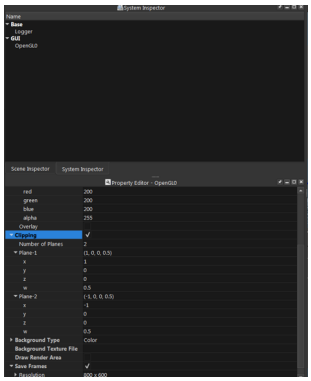
Time (s), Position X,Y,Z (m), Linear Velocity X,Y,Z (m/s), Angular Velocity X,Y,Z (rad/s), Force X,Y,Z (N), Torque X,Y,Z (N.m) ##

Fluid Properties Measurement and Data Export

After we put the measurement field in desired position, we can export the measured parameters by checking the "Export Stats" box in Scene Inspector. The Measurement field's input kIOParticles has to be connected to Solver's output (kIOParticles) by means of the Connection editor in order for measurement field to start measuring particle quantities.

Also we can visualize the velocity field inside the measurement field. Available options of measured data are listed as following:

Options	Meaning
Export Per Particle	If checked, information for all particles will be exported at each time step. Or an average will be taken and exported with time If not checked then averaged information for the measurement field will be written for all timesteps in one file.
Export Path	No need to add extension or the full path name. The measurements will be stored under the Measurements subfolder under the current scene folder.
Export Type	Specify the kind of information to be exported . (Not for Per Particle Output)
Subdivide	Divide the measurement field with specified mesh division. If checked, resolution of grid in each direction can be designated by entering the number you want. Then information will be averaged across each grid to the center of the mesh.
Velocity arrow scale	Specify the length of arrow in velocity field. Larger value will give a longer arrow.



1.3.23 Visualization Nodes

Edit → *Add* → *Visual*

Any node specically used for Visualization and Rendering only are created from this section.

Camera Node

Edit → *Add* → *Visual* → *Cameras* → *Camera*

This creates a new perspective camera in the scene. The cameras view can be altered interactively or through the properties and the camera can be assigned to a specific view.

Light Nodes

Directional Light

Edit → Add → Visual → Lights → Directional Light

The default light in the scene is a directional light and it shades objects based on direction of the light .

Spot Light

Edit → Add → Visual → Lights → Spot Light

Spot lights with cone and varying intensity based on distance to light can be created using this light. Creates a more realistic lighting scenario

Area Light

Edit → Add → Visual → Lights → Area Light

Not functional yet

Image Based Light

Edit → Add → Visual → Cameras → Camera

Not functional yet

Sky Light

Edit → Add → Visual → Cameras → Camera

Not functional yet

Material Nodes

PBR

Edit → Add → Visual → Materials → PBR

Physically based shading/rendering of rigid objects. Can be used for metals and other diffuse surfaces

Phong

Edit → Add → Visual → Materials → Phong

Structure

Edit → Add → Visual → Materials → Structure

Not functional

Toon

Edit → Add → Visual → Materials → Toon

Shades rigid objects

Tone

Edit → Add → Visual → Materials → Tone

Shades rigid objects with tone varying based on a primary color.

Ocean

Edit → Add → Visual → Materials → Structure

Not functional yet.

Clear Water

Edit → Add → Visual → Materials → Clear Water

Assigned only to fluid surfaces. Not functional yet.

Fluid Water

Edit → Add → Visual → Materials → Fluid Water

This is the material which is assigned to particles for fluid to emulate fluid/water like shading with transparency, reflection and refraction.

Gizmos

Null Gizmo

:menuselection:*'Edit –> Add –> Visual –> Gizmo –> Null Gizmo'*

Null Objects could be placed in the scene.

Interactive Measurement Gizmo

Edit → Add → Visual → Gizmo → Interactive Measurement Gizmo

Measurement can be made in the scene in default scene units (m) using this gizmo. Place the first null at the start point and the second one at the end point and measurement will be shown.

Clip Plane Node

Edit → Add → Visual → Clip Plane

Objects in the scene can be clipped according to where the clipping plane is placed.

1.3.24 Measurement Data Output

The output can be a text file or a csv file and the output can be either one file / frame of simulation or output for all frames for all particles. To make it work. Make sure that the kIOParticles of the solver is connected to kIOParticles of the MeasurementField in the connection editor which it looks like you are already doing since the property editor is displaying those values.

Then check the Export Stats to ON Then set the Export Path to a some name, by default the name would be the name of the measurement field. For example MeasurementField_0

TXT files are stored with the extention .txt and CSV files are stored with an extension .csv

If particle data for all frames is desired then check the box Export Per Particle should be set to ON.

Then Rewind the simulation to the beginning.

Then either run the simulation or playback.

The per particle data is stored per frame under Measurements folder with the prefix Measurement File Name and frame number with .txt

For Example MeasurementField_0.000000.txt etc

1.3.25 Visualization/Playback and Recording

Visualization results are in the Visualization directory.

1. OpenGL visualizations are under GLFrames/<Camera Name> folder and stored as a png sequence.
2. Recording of these images are described in the following sections.

Viewing Results

As Neutrino proceeds with the simulation, the results are displayed in the interactive GL Visualization Window. There are several sections to the Visualization window

On Screen Displays

The On Screen Displays displays (OSD), display various properties of the nodes in the simulation. There's a integrated OSD from the Neutrino Dynamic System and there are various user created OSDs.

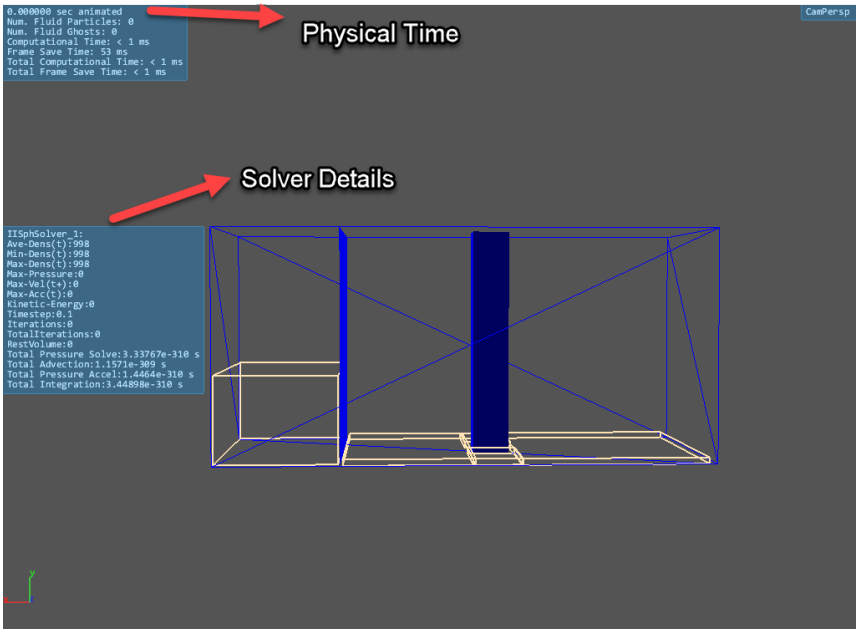
Integrated OSD

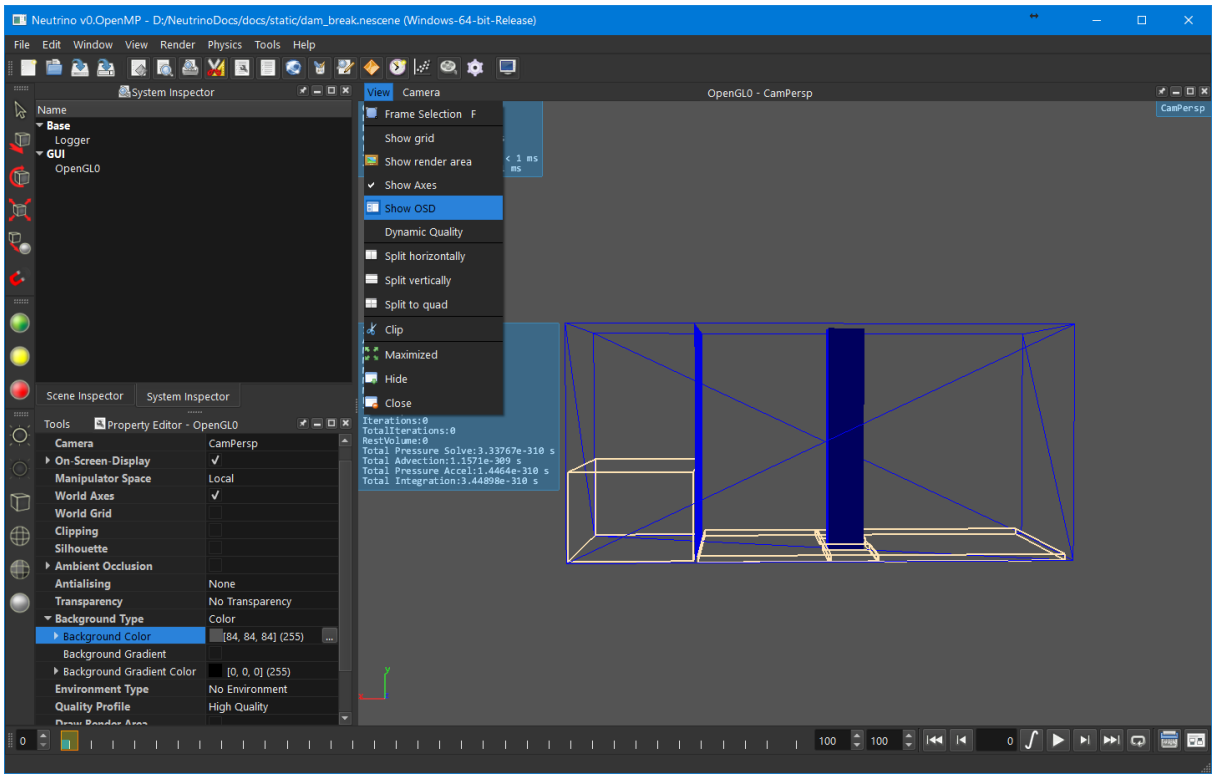
The integrated OSD displays information about the various nodes in the Neutrino Scene and Global Nodes as well. Each Neutrino node has an OSD property indicating the location of the OSD and some nodes have a toggle to Display Statistics ON/OFF which will turn ON/OFF the display of that node's OSD. The integrated OSD is depicted in the figure Fig. 1.3.25

The integrated OSD is toggled ON/OFF for display as depicted in the figure Fig. 1.3.25

User Defined OSD

There's another OSD which could be user defined and placed in the scene interactively. Its display is depicted by figure Fig. 1.3.25. The properties of this OSD could be configured to display user defined custom text labels and Option to turn on/off global time display.


Physical Time
Solver Details


Neutrino v0.OpenMP - D:/NeutrinoDocs/docs/static/dam_break.nescene (Windows-64-bit-Release)
File Edit Window View Render Physics Tools Help
System Inspector
Name
Base
Logger
GUI
OpenGL0
Scene Inspector
System Inspector
Tools
Property Editor - OpenGL0
Camera
CamPersp
On-Screen-Display
Manipulator Space
Local
World Axes
World Grid
Clipping
Silhouette
Ambient Occlusion
Antialising
None
Transparency
No Transparency
Background Type
Color
Background Color
[84, 84, 84] (255)
Background Gradient
Background Gradient Color
[0, 0, 0] (255)
Environment Type
No Environment
Quality Profile
High Quality
View
Camera
OpenGL0 - CamPersp
Frame Selection F
Show grid
Show render area
Show Axes
Show OSD
Dynamic Quality
Split horizontally
Split vertically
Split to quad
Clip
Maximized
Hide
Close

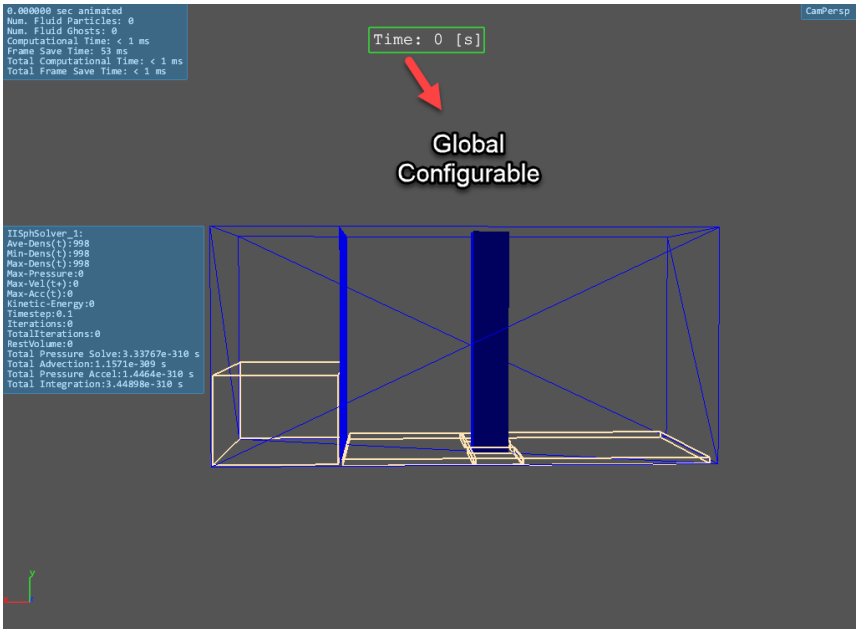
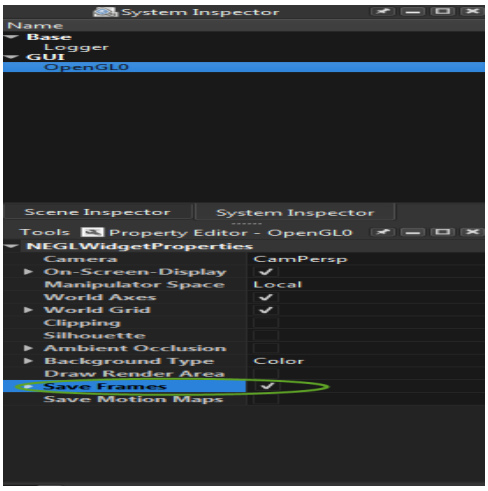


Image recording

Current OpenGL Window used for viewing can be exported as a sequence of frames for playback later.

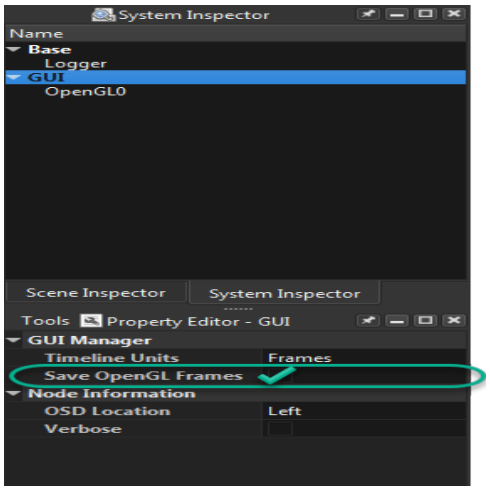
First check “saveGLFrames” under GUI system inspector and then check “Save Frames” under the Camera system inspector.

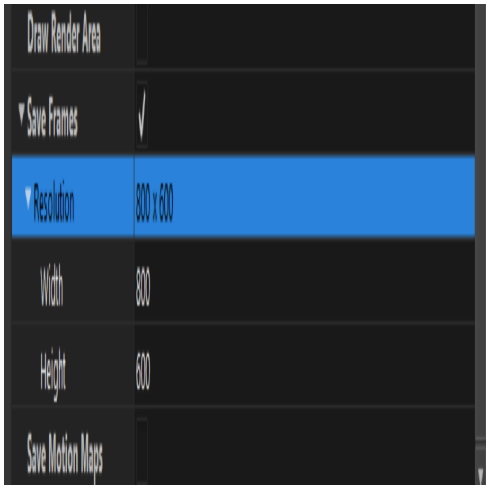


Their resolution also can be set as follows

After this is set when either Simulation is either “Run” or “PlayBack” the frames are stored as png files under the appropriate Camera Name folder under Visualization sub folder under the Scene Folder. The images are prefixed by the padded frame number and its stored as an image sequence. To convert it into movies for playback, ffmpeg could be used.

For example to store the image sequence as a movie file (MPEG4) for playback in VLC Player or web sites or presentations the following command could be used.


System Inspector
Name
Base
Logger
GUI
OpenGL0
Scene Inspector
System Inspector
Tools
Property Editor - GUI
GUI Manager
Timeline Units
Frames
Save OpenGL Frames
Node Information
OSD Location
Left
Verbose


Draw Render Area
Save Frames
Resolution
800 x 600
Width
800
Height
600
Save Motion Maps

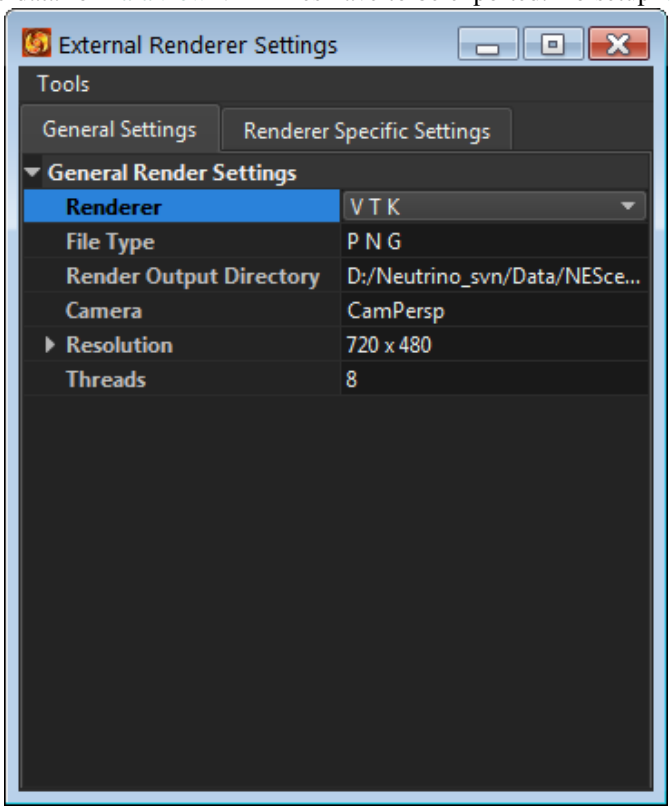
```
ffmpeg -i %06d.png -c:v libx264 -r 15 -pix_fmt yuv420p out.mp4
```

1.3.26 Rendering/Export to VTK/Renderers

Neutrino data can also be exported to VTK or other third party visualization packages like ParaView .

VTK/ParaView Export

To export the scene data for ParaView VTK files have to be exported. To setup VTK export open Render->External Renderer Settings.



The Renderer is set to VTK . Then playback or simulate the simulation in Neutrino. By default the VTK files go under RenderData subfolder under the Scene folder.

Static Rigid Geometry are exported with .vtp extension Measurement Fields if subdivided are exported as volume files (structured grid) .vts Particles are exported as vtp as well. Particle data is exported once per frame to be loaded into ParaView easily.

1.3.27 Neutrino Scene Data

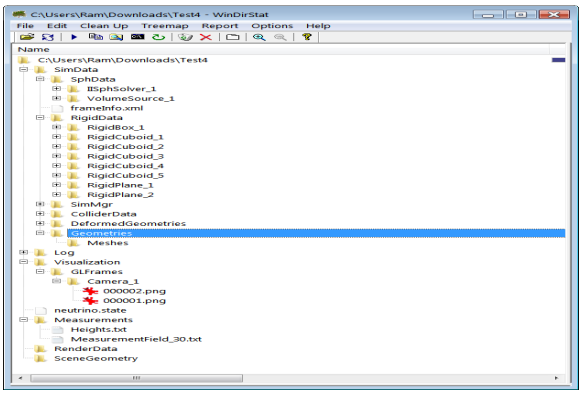
Results

> The results of the simulation are all placed under a directory under the scene directory with the scene name

1. Log 1. Measurements 1. RenderData 1. SceneGeometry 1. SimData 1. Visualization

Scene Directory Structure

The following is an example of the directory structure of a sample scene.



All of Neutrino data pertaining to a scene reside in a directory with the same scene name as the .nescene file currently open.

- **Logs**
  - Messages - Where the log of simulation runs are stored by data
- **Measurements**
  - MeasurementFieldName - Directory under which measurement data are cached
- **RenderData**
  - Where data exported to a third party renderer like VTK etc are stored
- **SceneGeometry**
  - All the local geometry obj, stl files etc stored for use in the scene are here. Rigid pathnames without a path refer to a file stored here.
- **SimData**
  - **SphData**
    - **SolverName**
      - **The particle caches are stored here** ######.neparticles - Particle data is cached here per frame. Particle data format is documented in the following section ######.sinf - information pertaining to a frame of particles
    - **EmitterName**
      - Various data pertaining to particle emitters in the scene are stored here.

1.4 Video Tutorials

1.4.1 General Capabilties

1.4.2 Pipe Flow Problem Setup

You can Download `CylinderFlowPipe.nescene` - the Neutrino scene file in this setup to compare the setups.

1.4.3 Internal Flooding Problem Setup

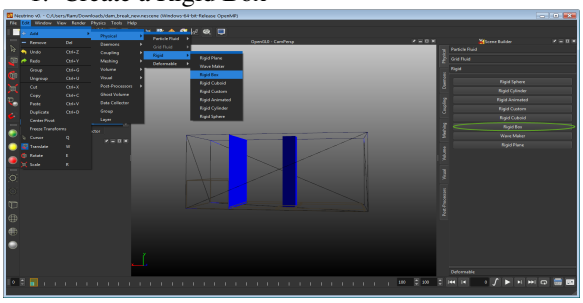
The Assets for this tutorial can be downloaded from :download *Torricelli_Tutorial_Models.zip <static/Internal_Flooding/Torricelli_Tutorial_Models.zip>* . The Neutrino scene file used to construct this is at

1.5 Tutorials

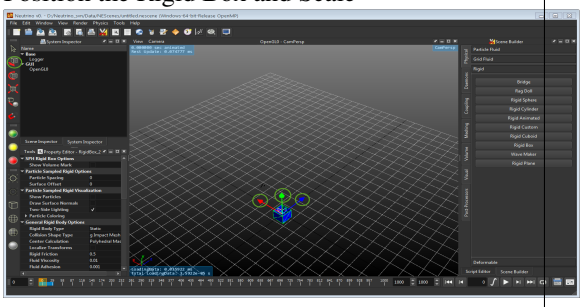
1.5.1 Basic Scene

This is a very simple scenario containing a box and a fluid emitter with a solver to emulate a very basic flow.

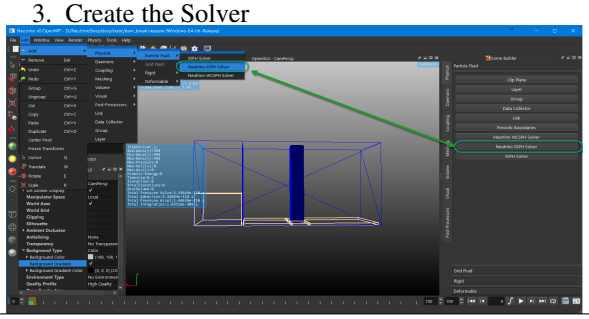
1. Create a Rigid Box



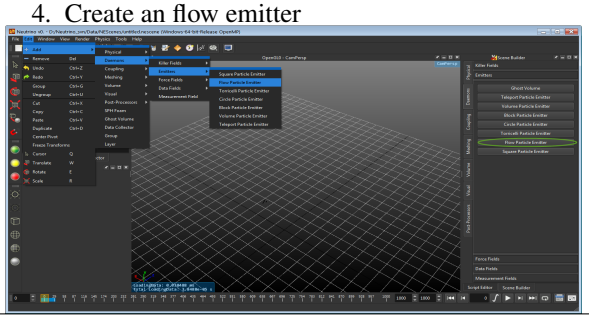
2. Position the Rigid Box and Scale



3. Create the Solver



4. Create an flow emitter



- **Position the emitter** Use the scale, rotate and position toolbar to position the emitter at the appropriate place.
- Adjust the flow rate

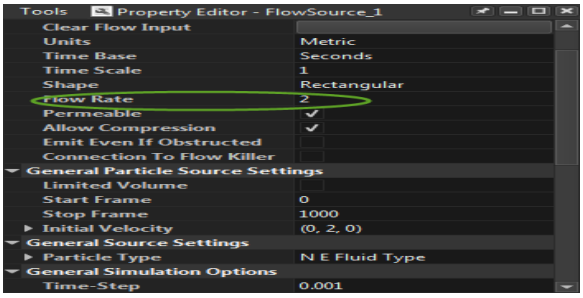


Fig. 1.1: The flow rate units is cc/m^3.

- Simulation

Arm the simulation and Simulate

- Results

  Further analysis can be done using a measurement field or visualized using sections described above.

1.5.2 Dam Break Scene

Dam Break Simulation and force measurement

Download `dam_break.nescene` - the Neutrino scene file in this setup.

The simulation is set up based on work from Cummins et al. 2012 and the configuration is shown as followed:

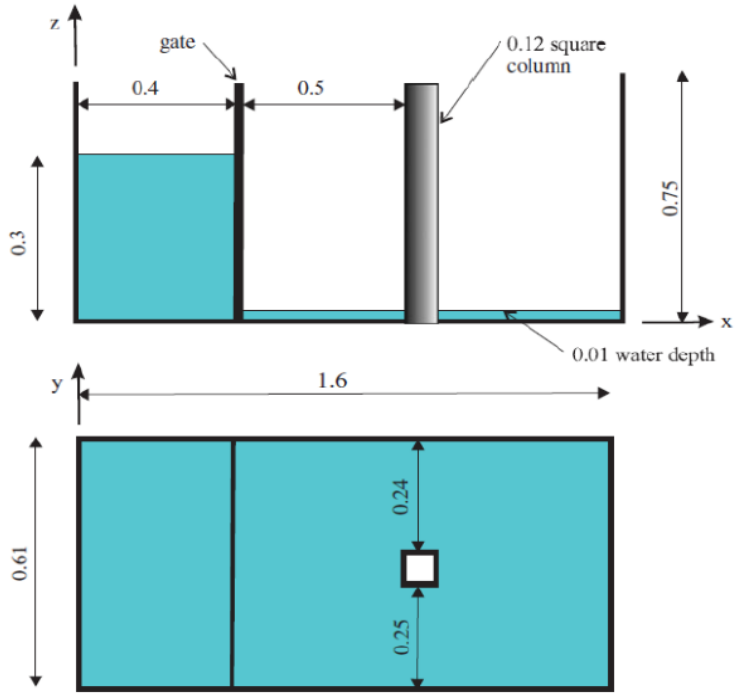
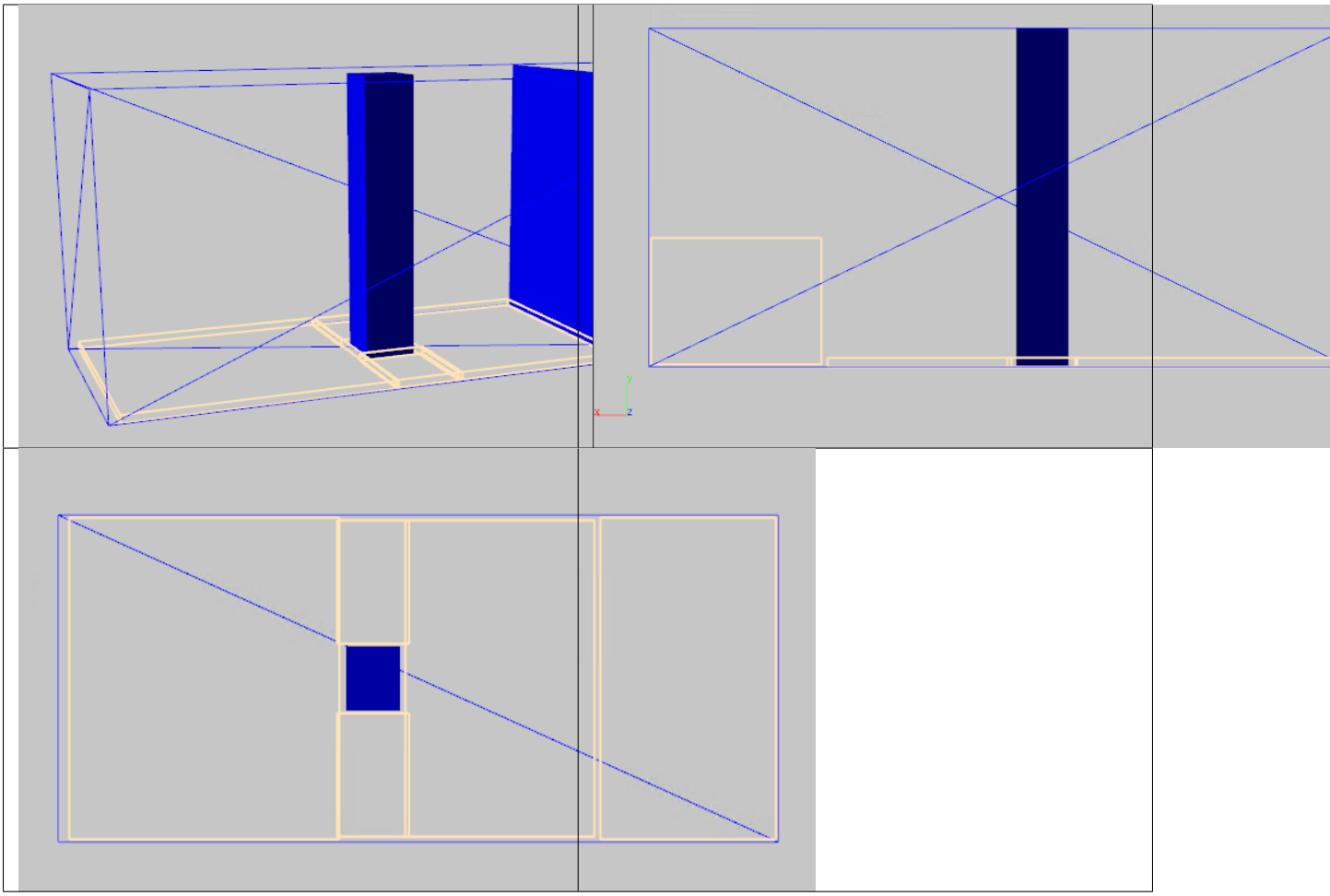


Fig. 1.2: Simulation setup by Cummins et al 2012

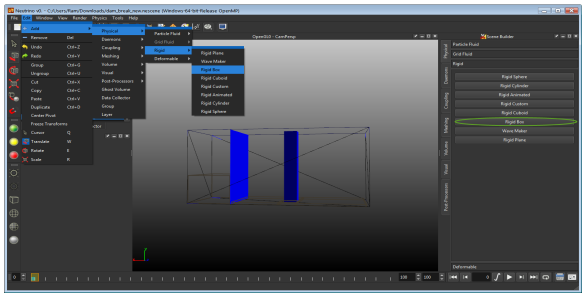
Setup of this problem is described in steps. The model is depicted as a box container to hold the fluid



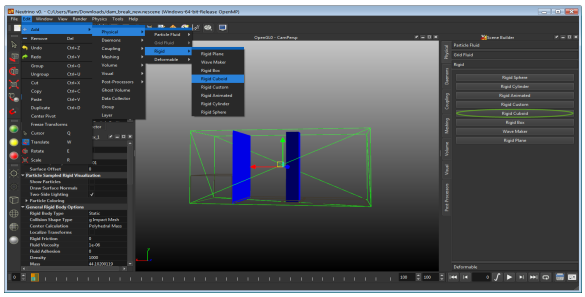
Setup in Neutrino

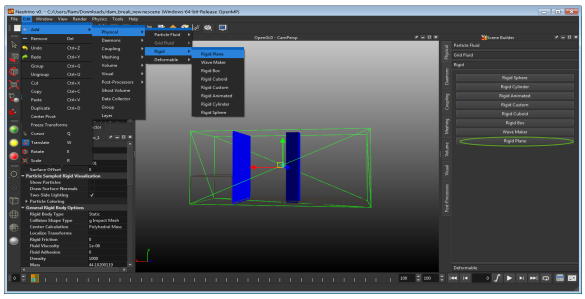
1. Create Geometry

- A Rigid box is set to be the simulation domain. Its created and positioned as follows



- Then a rigid cuboid is set as the dam structure. Rigid cuboid is created and positioned as follows.
- A rigid plane is also set as the gate. Rigid Plane is created and positioned as follows.





2. Fluid Containers

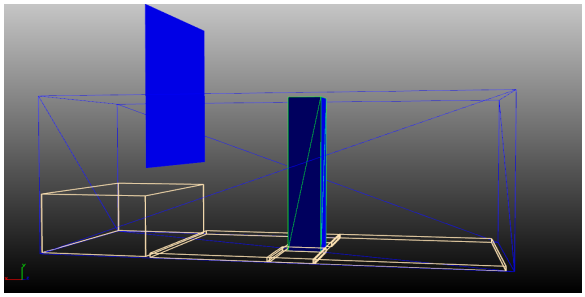
Five blocks of water are put inside the domain. The position and dimension of these objects are shown in Table 1. Block Particle emitters are created as follows

3. Positioning

The positions and scales are set to the following values. For positioning objects

- Select the object in the GL Window or in the Scene Inspector

- Look at its properties in the property editor
- Adjust the parameters of Position and Scale as indicated by the table



	Position	Scale
Rigid Box	(0, 0, 0)	1.6 * 0.75 * 0.61
Rigid Cuboid	(- 0.1, 0, 0)	0.12 * 0.75 * 0.12
Rigid Plane	(0.4, 0, 0)	0.75 * 1 * 0.61
Fluid Source #1	(0.6, - 0.23, 0)	0.39 * 0.28 * 0.6
Fluid Source #2	(- 0.48, - 0.365, 0)	0.6 * 0.02 * 0.6
Fluid Source #3	(0.18, - 0.365, 0)	0.42 * 0.02 * 0.59
Fluid Source #4	(- 0.1, - 0.365, - 0.18)	0.16 * 0.02 * 0.23
Fluid Source #5	(- 0.1, - 0.365, 0.18)	0.16 * 0.02 * 0.23

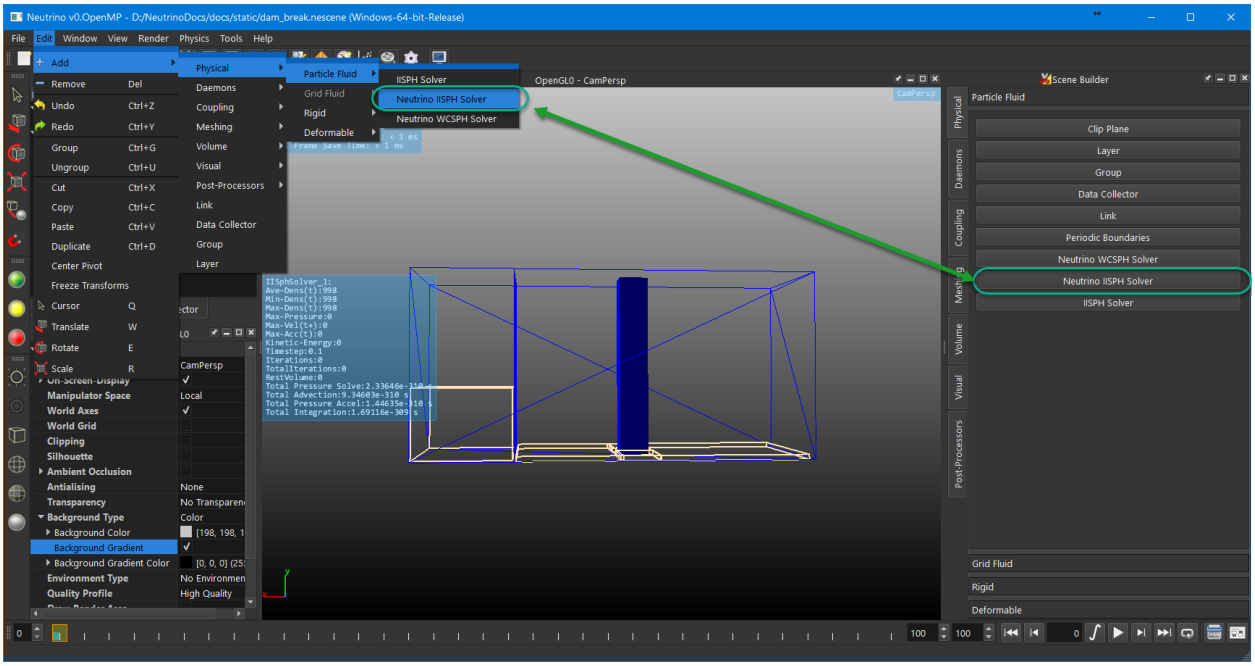
widths auto

align center

4. Solvers

Create a Solver (NIISPH Solver) as follows. The following figure indicates two different ways of creating the solver. The solver used in this scene is IISPHSolver but its better to use the Neutrino IISPH Solver as indicated in the picture if the scene is created from scratch.

Table : Value of coefficient in simulation setup



Coefficient	Value
Rigid Body	
Viscosity	1 * 10^{- 6}
Fluid	
Initial Velocity	0
IISPH Solver	
Viscosity type	Laminar
Viscosity	1 * 10^{- 6}
Cohesion Model	None
Adhesion Model	None
Particle Interaction Radius	0.01
Maximum Timestep	0.002

Note that the simulation is very sensitive to the Timestep, large time will cause fluid to be very unstable. Please make sure to check Maximum Timestep (effective when Adaptive Timestep is checked) and Timestep settings every time before the simulation starts.

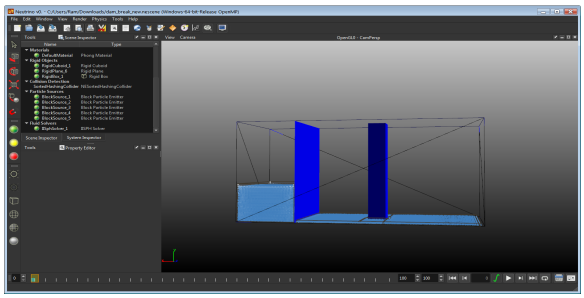
Simulation:

After finishing the setup, simulation is run for 1sec (50 frames in default) until all particles are settled down.

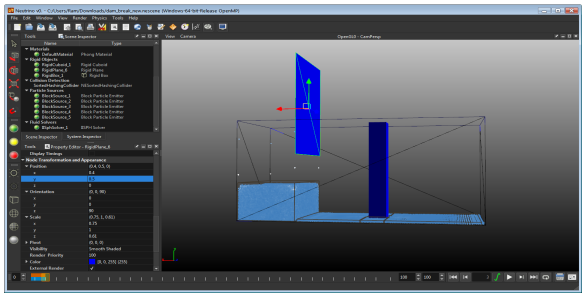
5. Run Simulation To Run the simulation please take a look at the following guide

* First Arm the simulation (Simulate)
* Then Run the Simulation (Proceed)

If the simulation is run for 1 second, the status would be looking like the following.



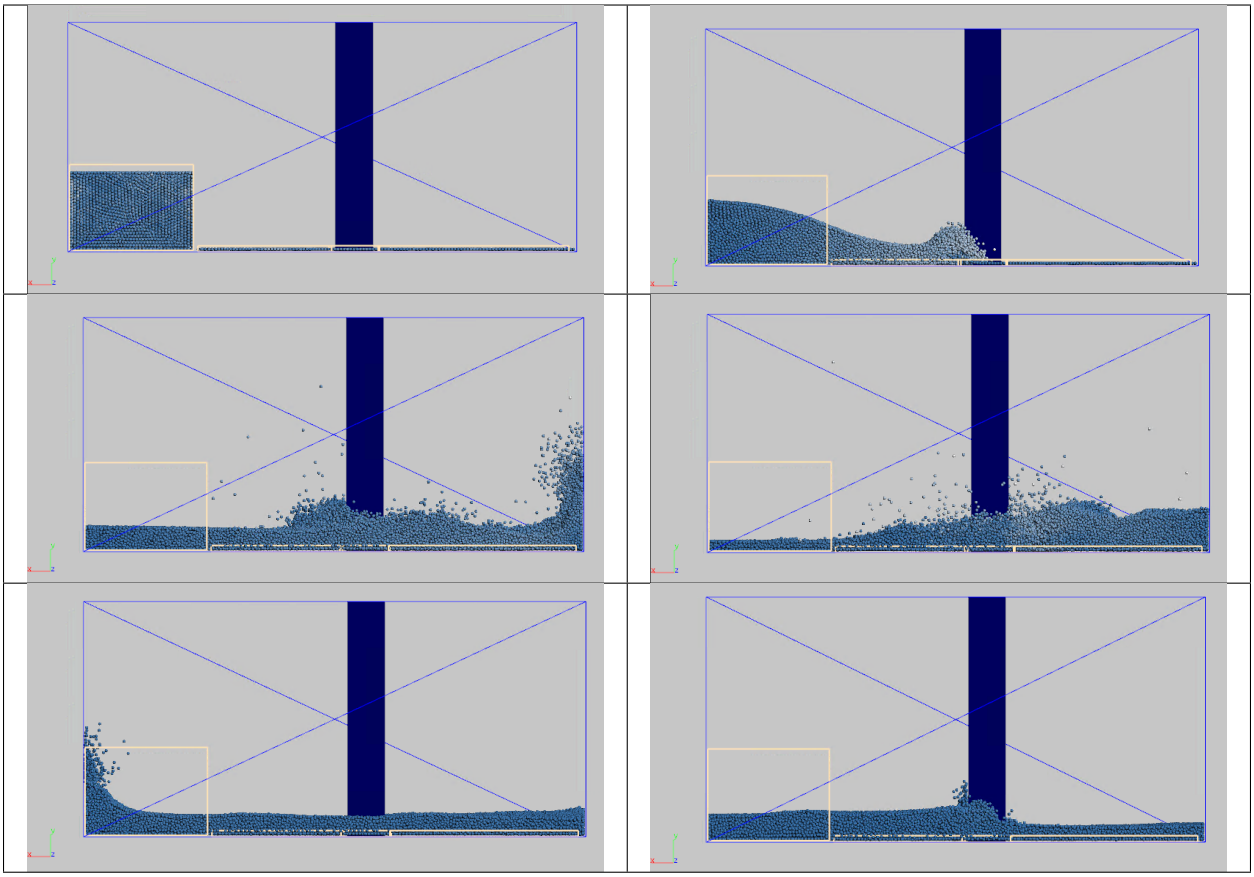
Then the gate (rigid plane) is open by translate the rigid plane up.



- Then continue the simulation.

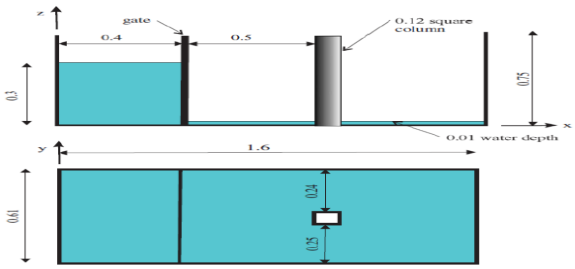
Fluid will flow under the effect of gravity. The force acting on the column at each frame can be found by checking the "Export Stats" of Rigid Cuboid inspector. A file will be created in designated path with magnitude of force, torque and angular velocity at each frame.

Simulation Progress



6. Estimating Rigid Forces and Output to file

The rigid force data exerted by the fluid on the solid can be exported by enabling Export in the Rigid Properties as follows



1.5.3 Periodic Boundary Conditions Scene

Download `PoiseuilleFlow5m.nescene` - the Neutrino scene file in this setup.

The scene is an example of a poiseuille flow

1. Setup

The following image shows a sample periodic boundary setup

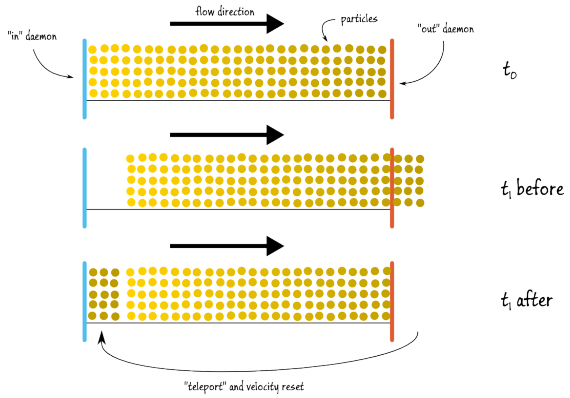


Fig. 1.3: Periodic Boundary Depiction

Periodic Boundary is enforced by ghost particles and in this scene there is a force constantly applied in the flow direction and eventually due to non slip boundary conditions a steady state is reached. Higher reynolds number flows are simulated by increasing the force or reducing viscosity. In this case laminar viscosity is used with carefully tweaked dynamic viscosity and kinematic visosity settings

- Create Four Planes enclosing the volume
- Create a Block Source emitter
- Create a Periodic Boundary Node * Edit -> Add -> Periodic Boundaries*
- Position the periodic boundary node to closely enclose the
- volume of the Block Source Emitter

Set it up, so it surrounds the particles nicely.

- Run the Simulation

The flow profile displays an arc where the boundary conditions are enforced by a non-slip condition and eventually reaches steady state

1.5.4 Sloshing Box Scene

Download `SloshingBox.nescene` - the Neutrino scene file in this setup.

The scene is an example of a sloshing box setup to do some validations.

1.5.5 Wave Tank Scene

Download `wavetank.nescene` - the Neutrino scene file in this setup.

This scene depicts a movement of a wavetank affecting a simulation domain. The wave tank motion is prescribed using keyframes as noted in the orientation property of the Wave Machine which is a rigid plane whose rotation would be in the direction of the tank.

This is a 3D setup which depicts the 2D setup from SPHERIC.

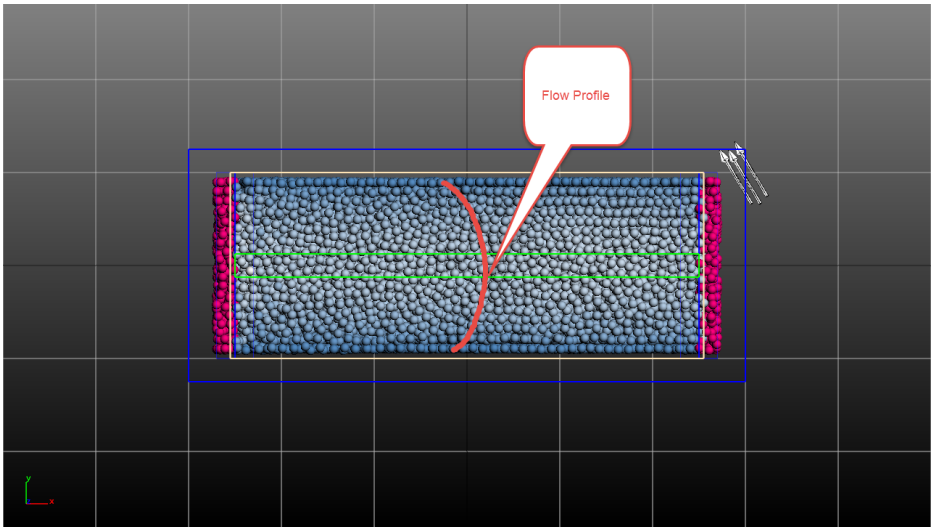


Fig. 1.4: Poiseulle Flow

The setup of the keyframes is done by invoking the python script `waveflap.py` . Also `Flap.dat` which is a description of the motion of the flap as a data file. The python script loads this data file and sets the various keyframes in Neutrino to mimic the motion according to the data file.

However the scene file already has this loaded so its not necessary to invoke these scripts and data . The python and the data file is just given as a description on how to setup something similar.

1.5.6 Flow Under Door Scenario Scene

In cases of simulating flow of fluid under doors, the particle sizes have to be sufficiently small in order for the flow to seep through the openings of the doors. This might result in added computation time. However there are some shortcuts to modeling this scenario and still preserving the accuracy by adding flow killers and flow emitters as depicted in the following figure.

Download `differentialFlow_v1.nescene` - the Neutrino scene file in this setup.

This setup consists of a description on how to setup the various Emitters/Killers to couple flow on one room to be transported into another when the gap under the door is small or when flow is needed through drains and/or when the drain size is smaller with respect to the particle size.

This file also is an example setup of how killers and emitters can be connected to each other thereby providing a transport mechanism of particles.

The procedure can be broadly described as follows

1. Create a Flow Killer, a Torricelli Source, and two Measurement Fields.
2. Adjust their positions, orientations, and scales, with the Flow Killer and one of the Measurement Field upstream, and the other Measurement Field downstream.
3. Set the connections: Measurement Field upstream -> Torricelli Source (kIOFluidHeight), Measurement Field downstream -> Torricelli Source (kIOHydrostaticPressure), and TorricelliSource -> Flow Killer (kIONumParticles).
4. Set the properties Acc. Floor and Discharge Coefficient under Torricelli Input Parameters.

1.5.7 Coupling Simulations Scene (Coupling with Shallow Water Solvers)

Download `coupling.nescene` - the Neutrino scene file in this setup.

Neutrino allows simulation to be performed on a narrow 3D domain with inputs from another Solver providing boundary inflow conditions.

Setup

You can load the .obj file of the terrain in Neutrino by: 1. creating a Rigid Custom node; then 1. entering the path to that file for the property "File Path".

In order to use coupling of Neurino with external Depth Averaged Solvers, you have to convert your gauge data and shallow-water data outputs to the data format readable by the Inflow Coupler of Neutrino. A Python script could do that.

The gauge data file is simply a set of lines "x-coordinate,z-coordinate", with as many lines as gauges. x- and z-coordinates are Cartesian coordinates in the local coordinate system of the Inflow Coupler. If you want to define a linear or rectangular boundary, these have to be normalized in the range [-0.5,0.5].

The shallow-water data file is a set of lines "time,x-velocity,z-velocity,height", each corresponding to a gauge and with a chronologically ascending order.

There also are two optional files used by the Inflow Coupler. One is for defining "extremity points" of the boundary; an other, used in the case of an arbitrary boundary-shape, is for explicitly specifying connections between the gauges. Depending on what to achieve, you may or may not need these two files.

The file format for the connecions is very simple. Here's an example connection file. Download: :download: *connexions.csv <static/connexions.csv>* Note that the indexing of the gauges starts with 0, not 1.

In the property editor of the inflow coupler in Neutrino, do the following: 1) Choose "Arbitrary" for Boundary Shape; 2) Set the path of the corresponding files for Gauge Input File, Connexion Input File, and Shallow Water Data Input File; 3) Adjust the values of Scale (x and z) to recover the size in meters of the coupling boundary; 4) Choose between "Interior Velocity" and "Exterior Velocity" for Flow Direction, depending on if you expect the flow to go inwards or outwards. That is it, basically.

Also, note that if you right-click on the inflow coupler in the scene inspector there are options that show up for reloading or clearing the files.

1.6 Common Questions

1.6.1 Core

1. What are Neutrino's Units ?

Unless specified otherwise, all parameter values in Neutrino are to be expressed according to the international unit system (i.e., in meters for distance, seconds for time, meters per second for speed, and so on). However, in the specific case of the property "Uniform Scale", it is a pure scaling factor, hence it does not have any actual unit. Basically, if your rigid object has been modeled in a modeling software with a size of, let us say, about 1 cm, you could set "Uniform Scale" to 100 in order to get a size of about 1 m for your rigid object. This is somewhat equivalent to set the three values of "Scale" to 100. In the future this property will be removed, as there is redundancy with "Scale".

1.6.2 Rigid Properties

1. What is remesh sampling of a rigid custom ?

This feature is deprecated and not working.

2. What is Surface offset ?

The property "Surface Offset" offsets the position of each solid particle towards either the interior (if the value set is negative) or exterior (if the value set is positive) of the rigid object. It is used for rigid of analytical shapes (Rigid Cuboid, Rigid Box, etc.), but does only have an effect for Rigid Custom when "Volume Sampling" is turned on. This is important for fine-tuning the location of the fluid-solid interface.

3. What's the rigid body? What's the difference by selecting dynamic/static/kinematic in rigid body type?

"Static" means the rigid body is completely fixed and won't "feel" the fluid (e.g., a wall). "Kinematic" means the rigid body can move (according to user input) but still won't "feel" the fluid (e.g., a door that can slide). "Dynamic" means the rigid body will "feel" the fluid and move accordingly (e.g., a ship floating on water).

4. What's Collision shape type and its options?

These are some advanced stuff about algorithms for collision. Just defaults are good . There is no need to play with this.

5. What's localize transforms?

Usually when rigid CAD geometry is imported they might not be placed at the origin. They might be modeled at an arbitrary lo

1. Polyhedral mass properties (Takes into account the concavity of the object)
2. Average of vertices

Usually option b is ok.

6. What is rigid friction used for? In what case can I use more than one rigid?

That property represents a coefficient of rigid-rigid friction. Obviously, that won't have any effect unless you have at least two rigid objects among which at least one is dynamic and collide with others.

7. Is the rigid not an idealized domain but a real thing that has mass and density?

It becomes a "real thing" when you set "Rigid Body Type" to "Dynamic". In the other cases, you do not have to specify a density.

8. Why cannot I make changes in Linear and Angular velocity?

If "Rigid Body Type" is set either to "Kinematic" or "Dynamic", these values are computed from their motion unless its at the beginning of the simulation where you can set it.

9. What is per-vertex velocities?

When turned on, the velocities of each vertex is calculated. This is used mainly for advanced rendering/visualization with motion blur. You can keep it turned off.

Solver Properties

1. Does the particle spacing refer to the size of the particle? What does boundary particle mean?

Yes, the property "Particle Spacing" is automatically adjusted to the "size" of the fluid particles. You do not have to manually change its value. The boundary particles refer here to the solid particles of the rigid object. These are used to represent the wall/solid boundaries and enforce boundary conditions with respect to the fluid.

2. Where can I set the density of the fluid?

Among the properties of Neutrino IISPH solver, you normally have "Fluid Preset" and "Rest Density". The property "Fluid Preset" set pre-defined values for the density and viscosity, such as those for water at 293 K. If you want to

freely choose the density (and viscosity) of the fluid, first choose "No Preset" for "Fluid Preset", then set the value you want to "Rest Density".

3. I found time step settings in both NIIsphsolver and inflowcoupler and terrain. Are they independent?

The inflow coupler and terrain should not have any property "Time Step" visible. I guess you have a relatively old version of Neutrino.

4. What assumptions did you use, is there any introduction about them?

The inflow coupler assumes that the vertical axis is the y-axis and that the gravity acceleration is along the minus y-direction. Neutrino IISPH solver solves the Navier-Stokes equations with the following assumptions on the fluid and flow: isothermal, incompressible, Newtonian, and viscosity constant in space.

5. Right now my computing is too slow, even on a 100 x 100 m area. Is there any way that can speed it up?

Possibly. There might be several ways to speed it up. What are your machine and its specifications? What is the "size" of the particles (i.e., the value of the property "Particle Rest Distance")? How many fluid particles do you have? These are some of the factors which should be taken into consideration

1.7 Some Simulation Guidelines

Expert knowledge and experience are required for properly setting-up a simulation, so that the best compromise between stability, accuracy, computational efficiency and memory usage is achieved. This section provides some pieces of advice and warning on setting-up the simulation and what can be done in the case the simulation is running unstably, too slowly or inaccurately.

1.7.1 Setting-up the simulation

- Be sure the solid particles are correctly positioned with regards to the solid boundaries. Have the solid particles displayed (property "Show Particles" of the solids) and adjust their positions (properties "Position" or "Surface Offset" of the solids) so that they are in contact with their corresponding solid mesh but not overlapping with it. E.g., in the case of a rigid box: if the simulation domain is contained inside, offset the surface by half the rest inter-particle distance (property "Particle Rest Distance" of the fluid solver"); if the simulation domain is on the outside, offset the surface by minus half the rest inter-particle distance; if both the inside and outside are part of the simulation domain, set the surface offset to zero.
- Be sure there is no extreme oversampling of rigid customs with solid particles. Have the solid particles displayed (property "Show Particles" of the rigid customs) to check it. Use the volume-based sampling (property "Volume Sampling" of the rigid customs) to remedy it.
- Be sure there is no extreme overlapping of solid particles due to solids too close to each others. Put the solids further apart, or use a more appropriate combination of solid shapes.
- Be sure there is no overlapping at all between fluid and solid particles. Adjust the positions of the fluid emitters and solids.
- Be sure that a correct volume/mass of fluid is generated. The number of fluid particles should be close enough to the ratio of the expected volume of fluid by the cube of the rest inter-particle distance (property "Rest Distance" of the fluid solver).
- Let the fluid rest down for a certain duration before starting the actual simulation. The process can be accelerated by temporarily increasing the fluid viscosity (property "Kinematic Viscosity" of the fluid solver) and the fluid-solid friction coefficients (property "Fluid Viscosity" of the solids). Be aware that too strong viscosity and friction may yield instabilities.

- Be sure there is no significant up-and-down oscillation of the fluid surface. If it is the case, increase the level of incompressibility enforcement by reducing the allowed density deviation (property "Stop Threshold" of the fluid solver). Note that it might be necessary to start with a relatively large compressibility before progressively makes the fluid more incompressible, in order to avoid strong instabilities.
- Use the cubic B-spline kernel (property "SPH Kernel" of the fluid solver) until the fluid is approximately at rest; starting directly with the quintic Wendland kernel tends to result in unstable behaviors. Later on, both choices should be reasonable.

1.7.2 Stabilizing the simulation

- Decrease the CFL coefficient of the adaptive time-stepping scheme (property "Time-Step CFL Coeff." of Neutrino IISPH solver). Be aware that the simulation will normally be slowed down.
- Decrease the viscosity coefficient of the adaptive time-stepping scheme (property "Time-Step Visc. Coeff." of Neutrino IISPH solver). This is relevant only for flows at low Reynolds number. Be aware that the simulation will normally be slowed down.
- Increase the maximum allowed number of pressure iterations (property "Max. Iter. Num." of Neutrino IISPH solver). This is relevant only when the pressure solver has trouble to converge sufficiently; check that the stats "Num. Pressure Iter." is always below the maximum allowed number of iterations. Be aware that the simulation might be slowed down.
- Increase the level of incompressibility enforcement, by decreasing the allowed density deviation (property "Stop Threshold" of Neutrino IISPH solver). This is relevant mostly in the case the fluid surface oscillates up-and-down too significantly. Be aware that the simulation might be slowed down.
- Add some amount of artificial viscosity (property "Add Artificial Viscosity" of Neutrino IISPH solver). This is relevant only for nearly inviscid fluids. The amount of artificial viscosity can be controlled (property "Artificial Viscosity Const." of Neutrino IISPH solver). Be aware that the simulation accuracy might be affected, positively or negatively.
- Use the particle shifting technique (property "Particle Shifting" of Neutrino IISPH solver) to regularize the particle distribution. Be aware that the simulation will be slowed down, and that the simulation accuracy will normally be impacted, positively or negatively, with a more regularized particle distribution but the loos or worsening of some desirable properties of the solver, such as conservation of mechanical energy. Getting the best from this feature requires fine tuning of the numerical parameters calibrating its strength (properties "Diffusion Constant" and "Max. Displ. Length Coeff." of Neutrino IISPH solver).

1.7.3 Accelerating the simulation

- Adjust the number of threads (property "OpenMP Threads" of "Base" under the system inspector) in accordance with the hardware and programs concurrently running.
- Be sure the simulation has not severely blown up. Check that the following stats does not have the following: for "Max. Velocity", an unexpectedly high value, infinity or NaN; for "Time-Step", an unexpectedly low value, zero or NaN; for "Num. Pressure Iter.", the value set to property "Max. Num. of Iter." of Neutrino IISPH solver.
- Be sure there is no particle leaving the simulation domain and free-falling outside by gravity. Check that the stats "Max. Velocity" does not continuously increase linearly. Use an extent particle killer to kill the particles exiting the simulation domain.
- Increase the size of the hash table of the collider (property "Hash Table Size" of the sorted hashing collider). A value above twice the number of fluid particles plus once the number of solid particles is advised. Be aware that the memory usage will increase.

- Adjust the pressure iteration coefficient of the adaptive time-stepping scheme (property "Time-Step Pressure Iter. Coeff." of Neutrino IISPH solver). The number of iterations of the pressure solver (stats "Num. Pressure Iter.") should ideally be around 10 to 15 per time step in average. If setting a very large value to the pressure iteration coefficient leads to an average number lower than 10, using such a large value, equivalent to deactivating the pressure iteration constraint on the time step, is optimal; conversely, if it gives an average number above 15, reduce the pressure iteration coefficient to be in the range 10 to 15. Be aware that the simulation accuracy might be slightly affected.
- Increase the CFL coefficient of the adaptive time-stepping scheme (property "Time-Step CFL Coeff." of Neutrino IISPH solver). Be aware that both the simulation stability and accuracy might become reduced.
- Decrease the level of incompressibility enforcement, by increasing the allowed density deviation (property "Stop Threshold" of Neutrino IISPH solver). Be aware that the simulation stability and accuracy might become reduced.
- Increase the rest inter-particle distance (property "Particle Rest Distance" of Neutrino IISPH solver). Be aware that the simulation accuracy will become reduced, potentially severely.

1.7.4 Improving the simulation accuracy

- Decrease the rest inter-particle distance (property "Particle Rest Distance" of the fluid solver). Be aware that the simulation will be slowed down, potentially severely.
- Decrease the CFL coefficient of the adaptive time-stepping scheme (property "Time-Step CFL Coeff." of Neutrino IISPH solver). Be aware that the simulation will normally be slowed down.
- Increase the level of incompressibility enforcement, by decreasing the allowed density deviation (property "Stop Threshold" of Neutrino IISPH solver). Be aware that the simulation might be slowed down.
- Adjust the amount of artificial viscosity (properties "Add Artificial Viscosity" and "Artificial Viscosity Const." of Neutrino IISPH solver). This is relevant only for nearly inviscid fluids. Too less viscosity might result in instabilities, while too much will yield excessive damping.
- Use the particle shifting technique to regularize the particle distribution (property "Particle Shifting" of Neutrino IISPH solver). Be aware that the simulation will be slowed down, and that there is the possibility of a loos of accuracy instead of a gain, due to the loos or worsening of some desirable properties of the solver, such as conservation of mechanical energy. Getting the best from this feature requires fine tuning of the numerical parameters calibrating its strength (properties "Diffusion Constant" and "Max. Displ. Length Coeff." of Neutrino IISPH solver).

CHAPTER 2

Neutrino API Guide

2.1 Introduction

Neutrino supports dynamic loading of various kinds of plugins which are recognized as they are placed in the appropriate folder under Neutrino Data.

2.2 Plugin Types

1. **Solvers**
    (a) Particle Solvers
2. **Emitters**
    (a) Particle Emitters
3. Force Fields
4. Surface Generation
5. Materials

2.2.1 Particle Solvers

Its best to explain plugins in Neutrino with the help of an example plugin.

In this case lets take an example of an implementation of a sample solver plugin Look at the example in Data/Plugins/SampleSPHSolver

This implements a solver based on predictive corrective SPH.

Building the plugin

A sample qmake pro file is included in the plugin. Please update the NE_BINARY_PATH to where the binary Neutrino is installed. Typically this is the directory containing the file neutrino.lib or neutrino.so depending on the Operating System.

Execute qmake and build/make.

The dlls or so of the plugin is installed in Neutrino_Install_Directory/Data/Plugins

Next time neutrino is started it loads the plugins from this directory structure. The various kinds of plugins supported by neutrino are indicated by the subfolders under Plugins.

For example a solver plugin would be under

Neutrino Data Folder (Neutrino Install Dir/Data) | Plugins -> release -> Spatial -> Physical -> SphFluid (For Release build)

Neutrino finds the plugin type from the directory structure and the type of plugin written and loads it in the Plugin Manager

Simulation Init

```
* NESceneManager::startUpdateTimer()
        * NESceneManager::prepareSceneForSimulation()
                * NESimulationManager::preparePhysicalsForSimulation
                        * NERigidManager::prepareRigidsForSimulation()
                                * Ex: NESphBox::generateBoundaryParticles
                        * NECollisionManager::update()
                                * NECollider::update()
                                * NEUniformGridCollider::update()
                                        *       Ex:␣
↪NESortedHashingCollider::constructGrid()
                                        *       Ex:␣
↪NESortedHashingCollider::insertRigidsIntoGrid()
                                        *       Ex:␣
↪NESortedHashingCollider::insertStaticRigidIntoGrid()
                        * NEGeometryManager::update()
                        * NEPostProcessManager::update()
```

Simulation Update

```
* NESceneManager::update()
        * NESimulationManger::update()
                * NELiquidManager::updateForces()
                        * NESphLiquidSolver::update()
                * NESphLiquidManager::integrate()
                        * NESphLiquidSolver::integrate()
                * NECollisionManager::update()
                        * NECollider::update()
```

Classes

class NESampleSimpleSolver
Implementation of a simple particle solver.

A Simple Solver which moves particles

Inherits from NESphLiquidSolver

Public Functions

uint *NESampleSimpleSolver*`init` (uint *initMode* = NE::kInitOnCreate)
Node initialization function, should be called from all inheriting classes.

`virtual` void *NESampleSimpleSolver*`deleteParticles` ()
Deletes all marked particles out of the vector.

`virtual` void *NESampleSimpleSolver*`deleteParticle` (NEParticle **particle*)
Deletes the particle.

`virtual` bool *NESampleSimpleSolver*`explicitBoundaryHandling` () `const`
Returns true if explicit boundary handling is required used when integrating particles.

`virtual` uint *NESampleSimpleSolver*`memoryConsumption` ()
Returns the memory consumption of the particles.

Public Slots

`virtual` void *NESampleSimpleSolver*`addParticles` (NEParticleSource **source*, std::vector<NEParticle> *&newParticles*, `const` vec3f *vel*)
For adding particles to the fluid, for example by a source that is connected to the fluid.

void *NESampleSimpleSolver*`update` ()
The main update routine for single-phase fluids, called by Liquid Manager.

NESphUpdatePart *NESampleSimpleSolver*`updateMultiPhase` (NESphUpdatePart *currentStep*)
The main update routine for multiphase fluids.

`virtual` qreal *NESampleSimpleSolver*`updateTimeStep` ()
Computes the required time step and sets it.

`virtual` uint *NESampleSimpleSolver*`saveFrameData` (QString *path*, `const` int *frameNumber*)
Overloaded function, to additionally save the timestep.

`virtual` uint *NESampleSimpleSolver*`loadFrameData` (QString *path*, `const` int *frameNumber*)
Overloaded function, to additionally load the timestep.

void *NESampleSimpleSolver*`setRadius` (qreal *r*)
Overloaded function for modifying fluid radius.

void *NESampleSimpleSolver*`setRestDensity` (qreal *density*)
Overloaded function for modifying rest density of the fluid.

2.3 Neutrino Classes

The full class documentation can be found under classes

2.4 Particle Cache Format

Download `NEParticleLoader.cpp` - sample code which loads an NEParticle cache file and displays information.

The particle cache file is divided into the following sections

1. Header

```
    struct NEBinaryDataSection
    {
            short Id;
            int contentFlag;
            int nParticles;
            long long Offset;
    };
```

2. Section Data

Particle as stored sequentially with their corresponding data described as follows.

```
    enum NEParticleDataType
    {
            IdPosition = 1,
            Color = 2,
            Velocity = 4,
            Density = 8,
            Pressure = 16,
            Vorticity = 32,
            Normal = 64,
            Neighbors = 128,
            Type = 256
    };
```

```
struct Color32
{
        inline Color32(){}
        inline Color32(uchar r, uchar g, uchar b, uchar a)
        {
                red = r; green = g; blue = b; alpha = a;
        }
        uchar red;
        uchar green;
        uchar blue;
        uchar alpha;
        inline void Set(uchar r, uchar g, uchar b, uchar a)
        {
                red = r; green = g; blue = b; alpha = a;
        }

        // Some preset colors
        static Color32 kRed;
        static Color32 kGreen;
        static Color32 kBlue;
        static Color32 kMagenta;
        static Color32 kYellow;
        static Color32 kCyan;
        static Color32 kOrange;
        static Color32 kWhite;
```

(continues on next page)

(continued from previous page)

```
        static Color32 kBlack;
};
```

Data structure describing per particle data.

```
class NEParticle
{
public:
        NEParticle() :  m_Id(0), m_Color(200,200,255,255), m_Position(0,0,0), m_
↪Velocity(0,0,0),
                m_Density(0), m_Pressure(0), m_Normal(0,0,0){}

        virtual ~NEParticle()  {}

        inline const vec3f& position() const { return m_Position; }
        vec3f& position() { return m_Position; }
        inline void setPosition(const vec3f& position) { m_Position = position; }

        inline const vec3f& velocity() const { return m_Velocity; }
        inline vec3f& velocity() { return m_Velocity; }
        inline void setVelocity(const vec3f& velocity) { m_Velocity = velocity; }

        inline uint id() const {return m_Id; }
        inline void setId(const uint id){m_Id = id; }

        inline const Color32& color() const { return m_Color; }
        inline void setColor(Color32 color) { m_Color = color; }


        inline float density() const { return m_Density; }
        inline float& density() { return m_Density; }
        inline void setDensity(const float density){m_Density = density; }

        inline float pressure() const {return m_Pressure; }
        inline float& pressure() { return m_Pressure; }
        inline void setPressure(float pressure){m_Pressure = pressure; }

        vec3f normal() const { return m_Normal; }
        void setNormal(vec3f val) { m_Normal = val; }

        uint type() const { return m_Type; }
        void setType(const uint &Type) { m_Type = Type; }


protected:

        uint m_Id;
        //      Modifying the relative order of color and position will break the␣
↪renderer
        // NEW: keep these variables at the end of the member variables list
        Color32 m_Color;
        vec3f m_Position;
        vec3f m_Velocity;
        float m_Density;
        float m_Pressure;
        vec3f m_Normal;
        uint m_Type;
```

(continues on next page)

(continued from previous page)

```
};
```

CHAPTER 3

Neutrino Physics

This document is a rough guide for the internal physics implementation of the Neutrino SPH solver. More details of the implementation could be found in *[SAMPATH2016]*, *[AKINCI2012]* and *[IHMSEN2014]* and *[MKSIISPH]*.

3.1 Introduction

Smoothed Particle hydrodynamics (SPH) is a common mesh-free Lagrangian method. It was originally introduced for dealing with astrophysical, compressible fluid flows *[MM1994]*, and later, its range of application domain extended to a wider variety of flows, such as non-Newtonian fluids , granular flows , and even solid deformation and fracture . SPH has been applied to incompressible, free-surface flows for more than two decades along with the closely related moving particle semi-implicit (MPS) method. Mainly, two approaches have been used in SPH for enforcing incompressibility: (1) state equation-based SPH (SESPH), in which a compressible approach is employed with an equation of state, relating density, pressure and speed of sound in the fluid, and (2) in incompressible SPH (ISPH), in which pressure forces are implicitly computed and iteratively refined by solving a pressure Poisson equation until the desired compressibility is reached.

SPH requires the computation of sums over dynamically changing sets of neighboring particles. Spatial data structures are employed to accelerate the neighborhood search, and efficient construction and query are thus essential. While hierarchical data structures are used in simulations with a variable particle interaction radius , uniform grids are more efficient when the radius is fixed. The cell size is usually set to the interaction radius. Index sort-based grids are often preferred, because the basic grid suffers from low-cache-hit rate and race conditions during parallel construction.

3.2 SPH Theory

SPH is a numerical method for both interpolating quantities and approximating spatial differential operators from a set of known quantities at sampled positions. Among the different ways to interpolate from scattered data, SPH relies on the use of radial basis functions. The value of a field at any point of space is estimated based on the value of this field at neighbor sampling points and its distance to them.

Usually a particle in physics is a minute quantity of matter and could be one of the following

- Smallest constituents of matter (Standard Model)
- Nanoparticles, colloidal particles
- Dust, powder, ashes
- Sediment grains, water droplets etc.

However the particles in SPH have the following properties

- They are material points
- They have volume, mass, pressure, density, etc.

3.3 SPH Approximation

Particle a has position ra, mass ma, volume Va, etc. * Particle interaction are computed using the 'kernel' w(r) * Cubic Spline Kernel is used in Neutrino

3.4 SPH Kernels

The kernel function used for interpolation in SPH must satisfy a certain number of properties. * It must be defined on a compact support * It must also be radial, feature a Gaussian-like shape, be sufficiently smooth * It is even and normalized leads to a first-order consistent continuous SPH interpolation. * Note that the particle distribution has an impact on the accuracy. * A regular and isotropic distribution is desired.

In Neutrino, the cubic B-spline kernel like the following is employed.

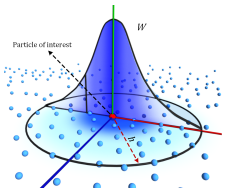


Fig. 3.1: Cubic Spline Kernel

3.5 SPH Navier-Stokes Equations

The single-phase, isothermal, incompressible, Newtonian fluid flows are the considered physical model. In addition, the viscosity is assumed constant in space, and the surface tension forces are ignored. The Navier-Stokes equations in their Lagrangian, velocity-pressure formulation then reads:

$$\frac{dv}{dt} = -\frac{1}{\rho}\nabla p + \nu \boldsymbol{\nabla^2} \mathrm{v} + \mathrm{g} \tag{3.1}$$

$$\frac{d\rho}{dt} = -\rho \boldsymbol{\nabla} . \boldsymbol{v} \tag{3.2}$$

$$\mathbf{v} = velocity$$

$$\mathbf{p} = pressure$$

$$\nu = viscosity$$

and

$$\mathbf{g} = gravity$$

Equation (3.1) comes from the momentum conservation, and involves pressure, viscosity, and gravity forces. Equation (3.2) is derived from the mass conservation. The incompressibility condition is satisfied either by

$$\frac{d\rho}{dt} = 0$$

or

$$\boldsymbol{\nabla}.\boldsymbol{v} = 0;$$

these two ways are equivalent in the continuous representation.

3.6 SPH Solvers

3.6.1 Implicit Incompressible SPH (IISPH)

Neutrino base solver is named as IISPH. IISPH solver computes pressure by iteratively solving a linear system to meet the incompressibility criterion. IISPH is based on semi-implicit form of the density prediction using the time rate of change of the density. By discretizing the continuity equation

$$\frac{D\rho}{Dt} = -\rho\nabla$$

$$\frac{\rho_i(t+t)}{t} = \sum_j m_j \mathbf{v}_{\mathrm{ij}}\,(t+t) \bullet \nabla W_{\mathrm{ij}}(t)$$

The unknown velocities $\mathbf{v}_{\mathrm{ij}}\,(t+t)$ depend on unknown pressure values at time , it's firstly predicted using any known non-pressure forces such as gravity, surface tension and viscosity. Then the predicted velocity is used to determine a predicted density.

$$\mathbf{v}_{\mathrm{ij}}^{\mathrm{adv}} = \mathbf{v_i}\,(t) + t\frac{\mathbf{F}_i^{\mathrm{adv}}(i)}{m_i}$$

$$\rho_i^{\mathrm{adv}} = \rho_i\,(t) + t\sum_j m_j \mathbf{v}_{\mathrm{ij}}^{\mathrm{adv}} \bullet \nabla W_{\mathrm{ij}}(t)$$

Now the pressure forces to resolve the compression is searched based on:

$$t^2 \sum_j m_j \left(\frac{\mathbf{F}_i^p(t)}{m_i} - \frac{\mathbf{F}_j^p(t)}{m_j}\right) \bullet \nabla W_{\mathrm{ij}}\,(t) = \rho_0 - \rho_i^{\mathrm{adv}}$$

Where $\mathbf{F}_i^p(t)$ is calculated by[1]:

$$\mathbf{F}_i^p\,(t) = -m_i \sum_j m_j \left(\frac{p_i\,(t)}{\rho_i^2\,(t)} + \frac{p_j\,(t)}{\rho_j^2\,(t)}\right) \nabla W_{\mathrm{ij}}(t)$$

From this point, a linear system can be got with unknown pressure value for each particle with the form:

$$\sum_j a_{\mathrm{ij}} p_j = b_i = \rho_o - \rho_i^{\mathrm{adv}}$$

With the calculated pressure on each particle, the unknown velocity $\mathbf{v}_{\mathrm{ij}}\,(t+t)$ can be calculated as:

$$\mathbf{v_i}\,(t+t) = \mathbf{v}_{\mathrm{ij}}^{\mathrm{adv}} + t\mathbf{F}_{\mathbf{i}}^{p}(t)/m_i$$

Then we will need this velocity to calculate the position of each particle. In Neutrino, different time integration schemes are applied.

3.6.2 Time Integration

a. Euler Cromer Integration

In Euler Cromer integration scheme, the new position of particle is calculated as:

$$\mathbf{x_i}\,(t+t) = \mathbf{x_i}(t) + t\mathbf{v}_i(t+t)$$

Verlet Integration

Verlet scheme calculate the new position as:

$$\mathbf{x_i}\,(t+t) = \mathbf{x_i}\,(t) + t\mathbf{v}_i\,(t+t) + 0.5t^2\mathbf{F}_i$$

The numerical tests show Verlet scheme has less numerical dissipation than Euler Cromer, this is because Verlet has a higher order calculation of position.

3.7 Momentum Equation

The conservation of momentum equation can be described as:

$$\frac{D\mathbf{v}}{\mathrm{Dt}} = -\frac{1}{\rho}\nabla P + \mathbf{g} + v_0\nabla^2\mathbf{v}_{\mathrm{ab}}$$

τ represents the viscosity term. In Neutrino, two different viscosity models are applied: a. Artificial viscosity; b. Laminar Viscosity.

3.7.1 Artificial Viscosity

The particle acceleration due to artificial viscosity proposed by Monaghan*[MM1994]* is defined as:

With

$$\mu_{\mathrm{ab}} = \frac{h\mathbf{v}_{\mathrm{ab}}\mathbf{r}_{\mathrm{ab}}}{\mathbf{r}_{\mathrm{ab}}^2 + \eta}$$

Distance between particles a and b is

$$\mathbf{r}_{\mathrm{ab}}$$

Average sound speed of particle pair a & b

$$c_{\text{ab}} = \frac{c_a + c_b}{2}$$

Average density of particle pair a & b

$$\rho_{\text{ab}} = \frac{\rho_a + \rho_b}{2}$$

$\eta = 0.01h^2,\ \alpha$ is the artificial viscosity parameter and it can be understand in term of kinematic viscosity as*[LIU03]*:

$$v_0 = \frac{\text{hc}}{8}$$

3.7.2 Laminar Viscosity

The laminar viscosity proposed by Morris*[MORRIS97]* is defined as:

$$\frac{dv_a}{dt} = -\sum_b m_b \left(\frac{4v_0 \mathbf{r}_{\text{ab}} \bullet \nabla W_{ab}}{(\rho_a + \rho_b)\mathbf{r}_{\text{ab}}^2} \right) \mathbf{v}_{ab}$$

v_0 is the kinematic viscosity of fluid simulated

3.8 Boundary Handling

The fluid-rigid boundaries require special attention. First, discontinuities of physical quantities that occur at boundaries are problematic for the usual forms of SPH. Proper boundary handling is necessary to avoid underestimated densities and non-physical pressure forces. Furthermore, pressure and friction forces between the fluid and the rigid bodies must be accounted for, and non-penetration must be ensured. Handling of thin objects or with complex geometries is particularly challenging, as well as when multiple dynamic objects are involved. Different strategies have been proposed through use of distance-based penalty forces, frozen particles , mirror particles *[LIU03]* or a wall renormalisation factor.

In Neutrino an efficient technique based on frozen particles and able to cope with complex boundaries, multiple bodies and the discontinuity issue is employed. Moreover, the technique uses fluid-rigid pressure and friction force models conserving linear and angular momentum.

3.8.1 Single Layer Boundary Mode

Appropriate adjustments are made to the boundary conditions to ensure the particle deficiency in using a single layer of particles.

The boundary coefficient is impacting the magnitude of the pressure forces, and it was empirically calibrated.

3.8.2 Multi Layer Boundary Mode

Currently Under Development

3.9 Fluid Structure Interactions

Using a standard SPH method in the sense that the time-integrated scheme works as classically done, with the following chronological steps (at each time step) :

1. compute the accelerations using the positions and velocities;
2. update the velocities using the accelerations;
3. update the positions using the velocities;

Typically, the steps 2) and 3) consists in a variant of Euler, Runge-Kutta, or velocity Verlet method.

3.9.1 IISPH vs Position Based Dynamics (PBD)

Conversely, PBD reformulates the force computations as a set of constraints on the positions and updates the positions directly without using the velocities. 1) compute the constraints using the positions and velocities; 2) update the positions by solving for the constraints; 3) update the velocities using the positions. Typically, the Stormer-Verlet method, or a higher-order equivalent one, is used.

3.10 Sample Cases & Validations

3.10.1 Dam Break

Dam break is simulated by lots of SPH program or software to demonstrate the SPH's power of dealing free-surface slamming phenomena. In this case, we measure the force exerted by fluid particles onto the dam structure and compare the results with experimental data. The simulation is one-to-one scale to real experiments and set up as in Figure 1 according to Cummins work*[CUMMINS12]*. In neutrino, we firstly put the gate in position and wait for 1 sec until all fluid particles are settled down. Then we open the gate and let the fluid flow under the effects of gravity.

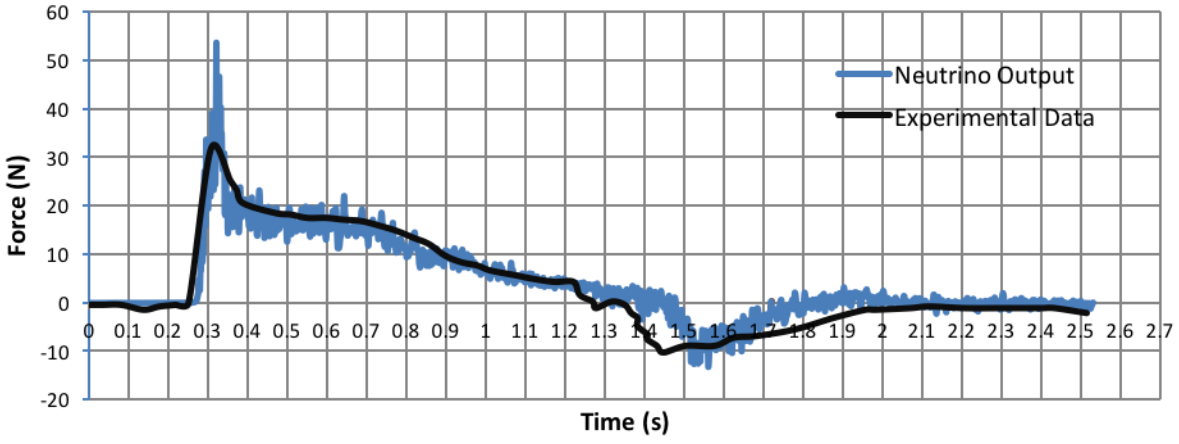


Figure : Comparison of the forces (Simulated Vs Experiment)

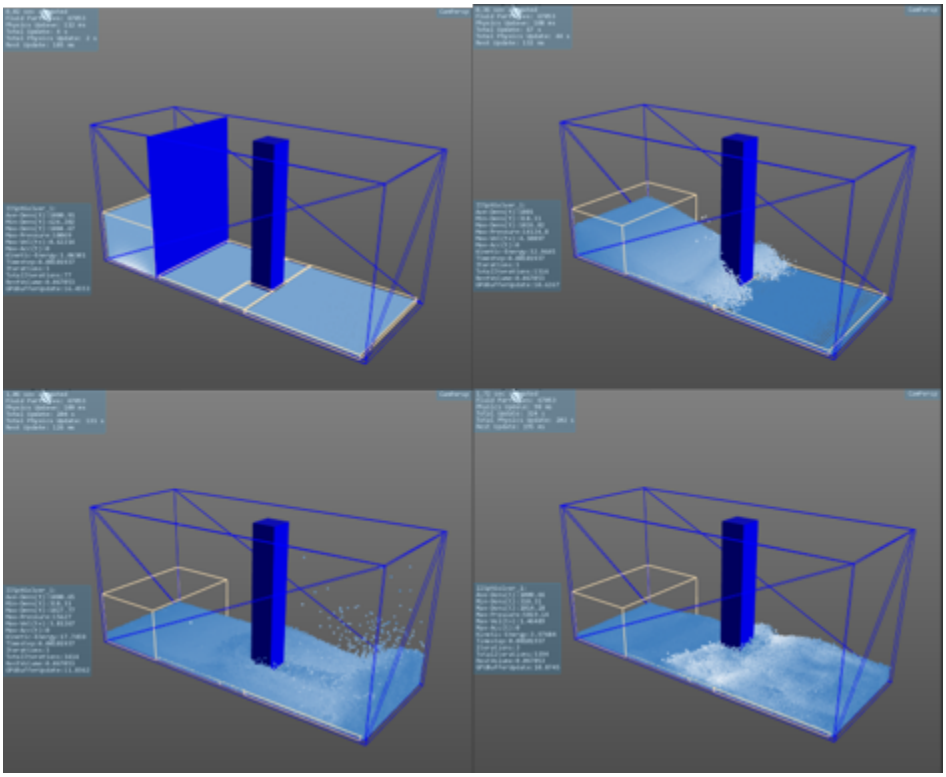


Figure : schematic diagram of the dam geometry*[CUMMINS12]*

Dual-Sphysics, Neutrino and LAMMPS-SPH can all fulfill dam-break case, which make it possible to compare their accuracy and time consuming. Besides, the force exerted from fluid particles to rigid body is an important factor to our purpose, by comparing the simulation data to the real experimental data, the feature of each software can be clearly seen. Table 2 shows the comparison of key parameters for each program. A comparison of simulated force to experimental data is shown in Figure 2 and a good agreement is found.

Table : comparison of experimental measurement, neutrino, LAMMPS-SPH and Dual-SPHysics

	Force Measurement	# Particles	Time Consumption (sec/step)	Avg. unit Time step (sec)
Experimental Data	Yes			
Neutrino	Yes	67053	0.057 (4 cores)	0.00102
LAMMPS-SPH	Need post-processing	64906	0.136 (8 cores)	0.00025
Dual-SPHysics	Need post-processing	116795	1.454 (4 cores)	0.00004

Figure : comparison of Neutrino output to experimental data

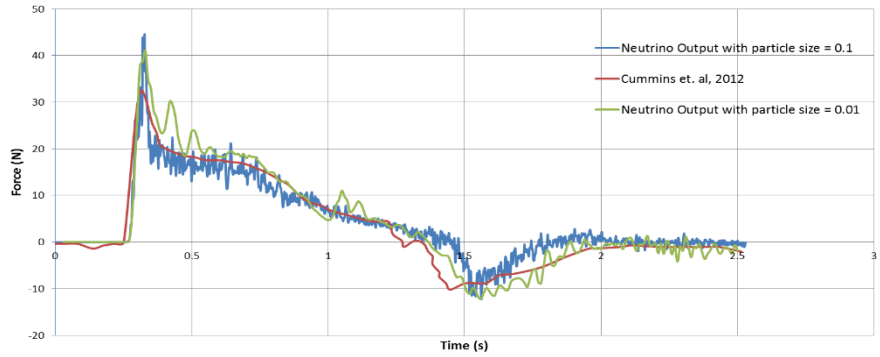


Figure: Comparison of forces using several particle sizes in Neutrino.

The first peak, representing the first slamming from fluid to the dam structure, is higher than the experimental data. This is mainly caused by the repulsive boundary treatment at fluid-solid interface. This also shows SPH is capable of determining fluid-solid interaction accurately.

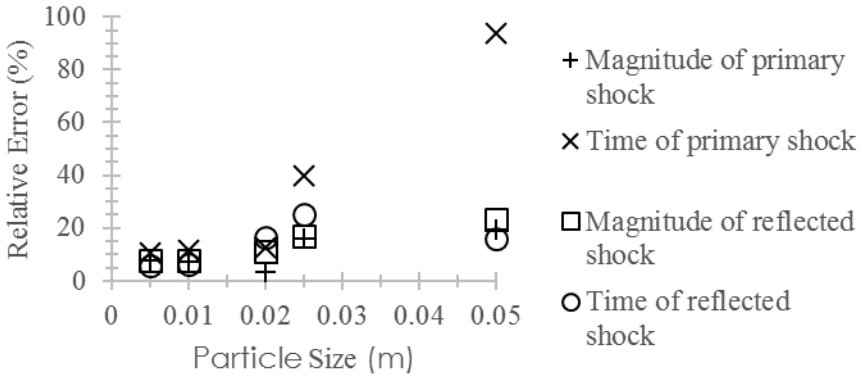
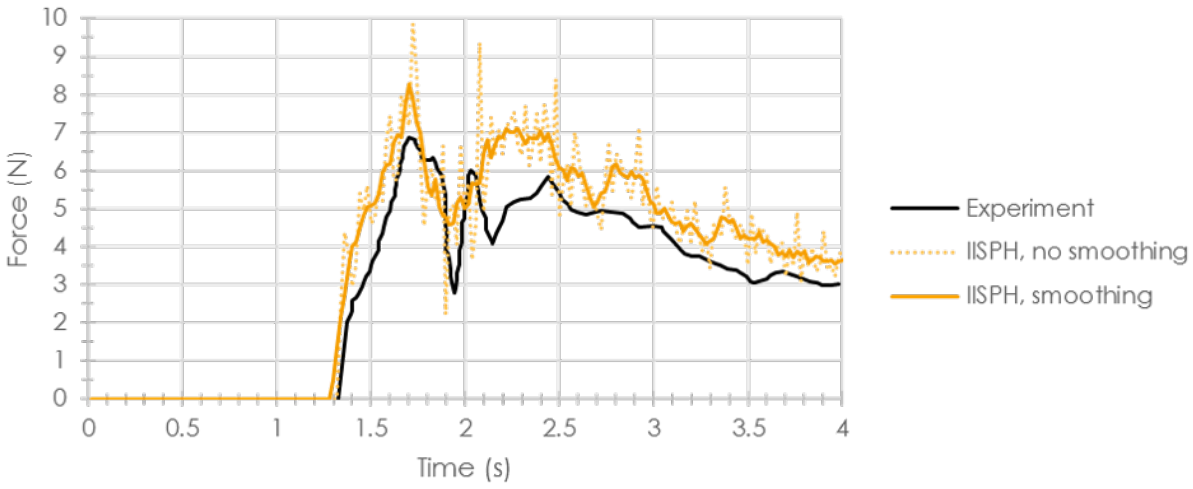


Figure: Estimate on the error based on the particle resolution

3.10.2 Aureli Dam Break

The Aureli dam break experiment is based the experiment from *[Aureli2015]*. The impact forces are about 5x lesser than the previous damn break experiment and there's a significant amount of air entrapment.

The measurement of the impact forces is done by smoothing out the highest frequency noise . It seems like the forces are overestimated by about 1N and the secondary shock not captured very well.



3.10.3 Poisuelle Flow

3.10.4 Faltisen - Wave Sloshing Experiment

3.10.5 Falling Body in Water

3.10.6 Solitary Wave Past Shore

3.11 References

Bibliography

[SAMPATH2016] Sampath, R. Montanari, N. Akinci, N. Prescott, S. Smith, C., Large-scale solitary wave simulation with implicit incompressible SPH, Journal of Ocean Engineering and Marine Energy, 2016, 1-17, 10.1007/s40722-016-0060-8, http://dx.doi.org/10.1007/s40722-016-0060-8

[AKINCI2012] N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler, M. Teschner, "Versatile Rigid-Fluid Coupling for Incompressible SPH", ACM Transactions on Graphics (Proc. SIGGRAPH 2012), vol. 31, no. 4, pp. 62:1-62:8, July 2012

[IHMSEN2014] Ihmsen, M. and Cornelis, J. Solenthaler, B. Horvath, C. Teschner, M, Implicit Incompressible {SPH}, IEEE Transactions on Visualization and Computer Graphics, 2014

[MM1994] Monaghan, J. J., "Smoothed particle hydrodynamics", *Annual Rev. Astron. Appl.*, 30: 543- 574., 1992

[MKSIISPH] Markus, I, etc., Implicit Incompressible SPH,

[MORRIS97] Morris, J. P, etc., "Modeling Low Reynolds Number Incompressible Flows Using SPH", Journal of Computational Physics 136, 214-226., 1997

[LIU03] Liu. G. R., etc., "Smoothed Particl Hydrodynamics: a meshfree particle method", World Scientific, 2003.

[CUMMINS12] S. J. Cummins., etc., "Three-dimensional wave impact on a rigid structure using smoothed particle hydrodynamics", International Journal for Numerical Methods in Fluids (2012); 68: 1471-1496, http://onlinelibrary.wiley.com/doi/10.1002/fld.2539/epdf

[CHERN] Chern, M.J., Borthwick, A.G.L. and Eatock Taylor, R. "Pseudospectral element model for free surface viscous flows", Int. J. Num. Meth. For Heat & Fluid Flow (2005), 15(6), 517 – 554, http://www.researchgate.net/publication/235263308_Pseudospectral_element_model_for_free_surface_viscous_flows

[GHIA82] Ghia, U., etc., "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method", Journal of Computational Physics, v. 5, n. 48, p.387-411, 1982

[Aureli2015] Aureli, Francesca and Dazzi, Susanna and Maranzoni, Andrea and Mignosa, Paolo and Vacondio, Renato, 10.1016/j.advwatres.2014.11.009}, Adv. Water Resources, Volume 76, January, 2015

Index

N