

Technical Report 2015

Investigation of Mesh to Point Cloud Conversion Approaches for
Applications in SPH-based Fluid-Solid Interaction Simulations

Felipe Gutierrez, Charles Ricchio, Milad Rakhsha, Arman Pazouki

August 21, 2015

Abstract

Contact detection in SPH-based fluid dynamics engines can be efficiently done if a sorted neighbor list of each SPH marker is constructed. In fluid-solid interaction simulations an SPH marker can be one of three different types: fluid, boundary or solid. Solid SPH markers represent a solid body and can be think of as a point cloud representations of a 3D geometry. In real world applications the solid body geometries are often complex and generating a point cloud representation of them is a very challenging task. Computer Aided Design (CAD) tools, however, make it easy for a user to generate any kind of 3D mesh representation. Unfortunately, contact detection of SPH markers and 3D mesh is a highly inefficient process, therefore we want to be able to represent this mesh as a point cloud of solid markers. In this report, we present three different approaches to convert a 3D mesh, generated using a CAD tool, into a point cloud representation of the mesh. Finally, we run the same simulation multiple times with a point cloud generated with each approach and analyze which approach produces the most stable simulations.

Keywords: Graphics, Mesh, Point Cloud, Smoothed Particle Hydrodynamics, SPH

Contents

1	Introduction	3
2	Ray Crossing Approach	3
2.1	Algorithm Overview	3
2.2	Implementation	3
2.3	Tool Usage	4
2.4	Results	5
3	Point-Plane Projection Approach	6
3.1	Algorithm Overview	6
3.2	Implementation	6
3.3	Tool Usage	7
3.4	Results	7
4	Salome Approach	7
4.1	Algorithm Overview	8
4.2	Tool Usage	8
4.3	Results	9

1 Introduction

Background goes here: Possible stuff to introduce here:

2 Ray Crossing Approach

In this section we are going to present and explain the essential code snippets that allow

2.1 Algorithm Overview

The algorithm presented in this section to generate a point cloud from an input mesh relies on answering one simple question repeatedly: Is the current point inside the mesh, or not? In order to determine if a point is inside, an approach known in the literature as: even-odd rule, the crossing number method and ray-crossing method was used [6,8]. The algorithm goes as follow; a ray is cast in any direction from the point to infinity, then the number of intersection of that ray with the mesh is counted, if the number of intersections is odd then the point is inside the mesh, otherwise it is outside the mesh. An visualization of this technique can be seen in Figure 1.

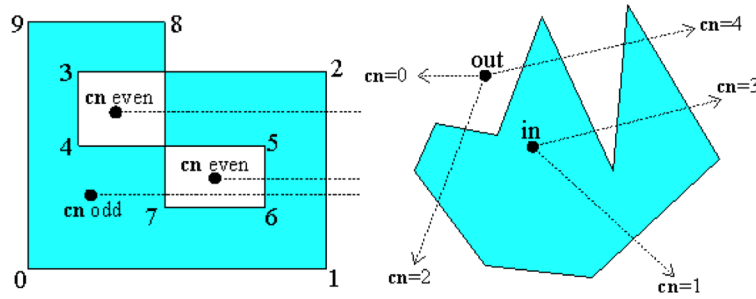


Figure 1: Crossing Number Example. On both the left and the right example we can see that the number of ray intersections from points inside the blue meshes is odd.

2.2 Implementation

The algorithm was implemented as a web application in Javascript using a lightweight 3D library called Three.js [1]. The workflow of the web application goes as follows:

Action 1

Import the mesh. This can be done in two different ways: Use one of the pre-made meshes provided by Three.js or import a custom mesh defined in an .obj file format.

Action 2

Scale the mesh to the needed dimensions of the application.

Action 3

Compute a bounding box surrounding the mesh. This allows us to only iterate through points that are either inside or that are closely surrounding the mesh.

Action 4

Define the spacing between points. In SPH simulations this would be what is usually called h .

Action 5

Iterate through the points inside the bounding box separated from each other at a distance which was specified in step 4. At each point cast 3 rays: one pointing to the center, another one pointing close to the center and a third one pointing to one of the bounding box corners. If the number of intersects with the mesh of all rays is odd, then the point is inside the mesh.

Casting only one ray per point in any direction theoretically should be enough to give us all points inside the mesh. However, the ray casting tool provided by Three.js sometimes makes mistakes and includes a few points outside the mesh. The reason for this might be that: the ray is going exactly through where a point defining a vertex is located, or the point casting the ray is located exactly on a triangle of the mesh, or the casted ray is tangent to a vertex of the mesh. In order to remove these noisy points, 3 different rays are cast in different directions.

2.3 Tool Usage

The mesh to point cloud tool web application is available at <http://euler.wacc.wisc.edu/felipegb94/PointCloudWebUtils/>. This can only be accessed if the user is connected to the UW-Madison network. There is no user interface yet, but all of the functionality is currently there. Therefore to use it you will have to edit a few files in Euler at `/home/felipegb94/public_html/PointCloudWebUtils`.

To create a point cloud from a pre-made Three.js meshes you have to:

Action 1

Open *MeshToPointCloud.html*

Action 2

Go to line 114

Action 3

Edit the line that says: `RenderThreeMesh(Cylinder);`

Action 4

Change Cylinder for any of the following pre-made meshes available: Cube, Sphere, Cylinder, Cone, Torus, SemiTorus, TorusKnot.

Action 5

Go to <http://euler.wacc.wisc.edu/~felipegb94/PointCloudWebUtils/> and click on GeneratePC and then on Threejs mesh to PC.

To create a point cloud from a custom mesh defined in an .obj file you have to:

Action 1

Copy the .obj file to Euler inside the folder: `/home/felipegb94/public_html/PointCloudWebUtils/`

Action 2

Open *MeshFileToPointCloud.html*

Action 3

Go to line 114

Action 4

Edit the line that says: *RenderMesh("meshes/humvee.obj");*

Action 5

Change *meshes/humvee.obj* to *meshes/nameOfYourMesh.obj*

Action 6

Go to <http://euler.wacc.wisc.edu/~felipegb94/PointCloudWebUtils/> and click on GeneratePC and then on Mesh File to PC.

To edit the point spacing you have to:

Action 1

Open *js/ThreeUtils.js*

Action 2

In the function *MeshToPC(MeshGeometry, MeshObject)*, look for the definition of *dr*, set it to the spacing you want.

To edit the scale of the point cloud of a custom mesh you have to:

Action 1

Open *js/RenderMesh.js*

Action 2

In the function *RenderMesh(filename)*, look for *object.scale = new THREE.Vector(xDimension, yDimension, zDimension)* and change to the preferred dimensions.

2.4 Results

Figure 2 contains

3 Point-Plane Projection Approach

3.1 Algorithm Overview

In the algorithm used, the positions of a predefined volume of evenly spaced points are checked against the planes defined by the triangles in a triangle mesh imported from a Wavefront OBJ file. Points are projected onto these planes, which are then checked to see if they lie within the points that define the triangle itself [7]. If the projected points are within the triangle, the distance between the original point and projected point is checked to verify that the distance is within a tolerance defined by the spacing of the original set of points. If the distance falls within the tolerance, then the position of the point is kept to be exported.

3.2 Implementation

This approach was implemented in C++, using the TinyOBJLoader library for mesh importing, and the GLM library for vector types and math [3, 4]. The program's procedure goes as follow:

Action 1

Generate a volume of points of defined dimensions and density from which to subtract points.

Action 2

Import the mesh and save the vertex list and index list.

Action 3

Iterate through each point in the volume, and then through each triangle in the mesh, moving on to the next point when the current one is verified to either be within or outside of the tolerable range. To determine if a point is within the tolerable range the following calculations are made:

- (a) Calculate the normal of the triangle in question
- (b) Find the central point of the triangle
- (c) Find the distance of this point from the origin of the coordinate grid
- (d) Project the point in question onto the plane defined by this information
- (e) Verify that the projected point lies within the legs of the triangle
- (f) Verify that the distance from the projected point to the original point is within tolerance

Action 4

Finally, if a point meets all of these conditions as it is being checked, it is pushed into a vector which is returned to the caller at the end of the iteration. This list of points is then written to a CSV file.

3.3 Tool Usage

The source for this tool can be found at <https://github.com/uwsbel/Mesh-to-pointcloud-tool>. TinyOBJLoader is already included in the source, meaning GLM is the only library you need to acquire. The only source files that need to be built are *main.cpp* and *tiny_obj_loader.cc*. These should be compiled with threading and at least SSE2 enabled. To create a point cloud of an OBJ, run the executable and answer the prompts

Action 1

Enter the path to the OBJ file

Action 2

Enter the spacing desired between each point

Action 3

Enter the desired side length of the volume of points to check against

Action 4

Enter the amount of threads you want this to run on

3.4 Results

Figure ?? contains different point clouds that were created with this tool.

4 Salome Approach

Another approach to solve the problem is to use the existing tools/applications whose goal can be modified in order for us to create a point cloud. One of these techniques is to use CFD pre-processor applications which can be used to create structured/unstructured mesh. Assuming that a CFD mesh of a geometry is available, one can use the information of cells which have been created in the application and use the information of nodes data of the mesh and use the resulting points as a point cloud. In this technical report, Salome [2] is used in order to show this technique. It is worth mentioning that other pre-processor open source applications such as Gmesh [5] can be used in a similar manner.

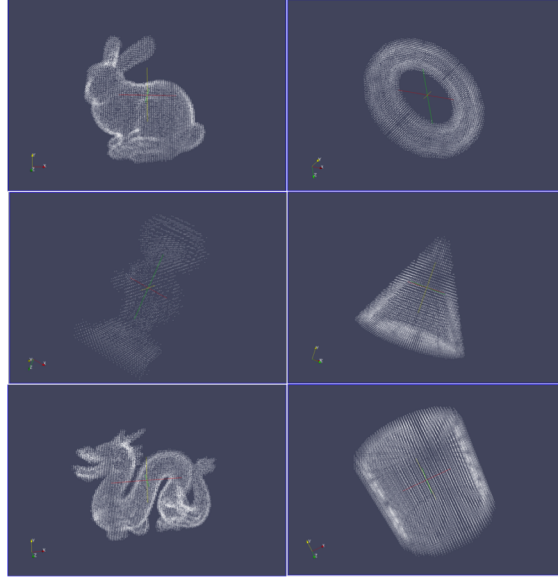


Figure 2: Resulting point clouds from different mesh inputs using the Point-Plane Projection Approach.

4.1 Algorithm Overview

In order for Salome to create a mesh, either a geometry of interest should be imported in the program or the geometry should be created inside the geometry module of the Salome. For simple geometries, using the native geometry manipulation of Salome should solve the problem, although for more complex geometries one might need to consider creating the geometry in a CAD generator software such as SolidWorks [?] and save the file as a .step and import the resulting file in Salome.

Either way when the geometry is created, the mesh module inside Salome can be used in order to create a mesh. Different approaches can be chosen for the choice of meshing technique depending on the requirements of the point cloud. Finally, when the meshing is done, the file can be exported in a DAT format.

4.2 Tool Usage

In this tutorial a simple geometry is created in Salome in the first step. For this purpose, geometry modules is selected as shown in the following figure.

Action 1

Select the geometry module as shown in figure 3.

Action 2

A new entity can be created by selecting: New Entity→Primitives→Torus.

Action 3

Select the mesh module as shown in figure 4.

Action 4

Create a new mesh by selecting: Mesh→Create Mesh.

Action 5

In the tree menu right clicking on the Mesh.1 and selecting the Edit Mesh/Sub-mesh the spacing of the point cloud can be changed. Figures 5 show the simplest algorithm to create the point cloud.

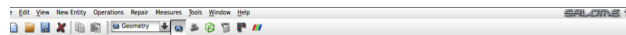


Figure 3: Salome's toolbar when selecting the geometry module



Figure 4: Salome's toolbar when selecting the mesh module

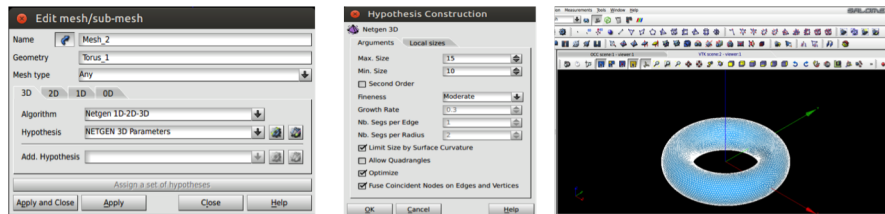


Figure 5: Salome's toolbar when selecting the mesh module

4.3 Results

Figure 6 shows how a torus point cloud was created using Salome.

References

- [1] R. Cabello. Three.js. <https://github.com/mrdoob/three.js>.
- [2] Open Cascade. Salome: The open source integration platform for numerical simulation. <http://www.salome-platform.org/>, 2015.
- [3] G-Truc Creation. OpenGL mathematics (glm). <http://glm.g-truc.net/0.9.7/index.html>, 2015.

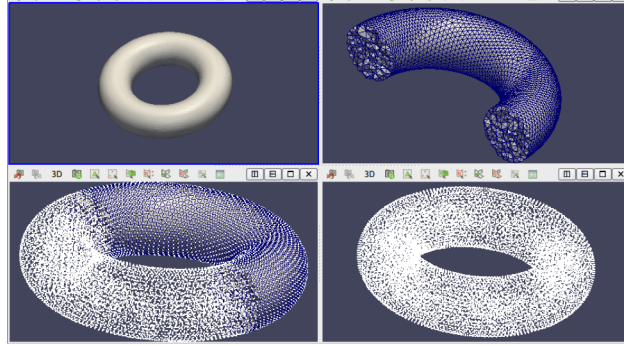


Figure 6: Mesh to Point-Cloud stages.

- [4] S. Fujita. Tinyobjloader. <https://github.com/syoyo/tinyobjloader>.
- [5] C. Geuzaine and J.F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, (79):1309–1331, 2009.
- [6] Point in Polygon. Point in polygon — Wikipedia, the free encyclopedia, 2015. [Online; accessed 1-August-2015].
- [7] Point-Plane Distance. Point-plane distance — wolfram mathworld. [Online; accessed 1-August-2015].
- [8] D. Sunday. Inclusion of a point in a polygon. http://geomalgorithms.com/a03-_inclusion.html.