

---

# imstk-unity Documentation

*Release 1.0.0*

**Kitware**

**Jun 30, 2022**

## Introduction

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Overview . . . . .	2
1.2	Limitations . . . . .	2
1.3	Project Structure . . . . .	3
<b>2</b>	<b>Setup for Development</b>	<b>3</b>
<b>3</b>	<b>Devices</b>	<b>4</b>
3.1	Debugging . . . . .	4
<b>4</b>	<b>Tutorial</b>	<b>4</b>
<b>5</b>	<b>Example Scenes</b>	<b>10</b>
5.1	PbdClothCollision . . . . .	10
5.2	PbdClothScene . . . . .	10
5.3	PbdThread . . . . .	10
5.4	RbdScene . . . . .	10
5.5	Tutorial . . . . .	10
5.6	Devices . . . . .	11
<b>6</b>	<b>Usage</b>	<b>11</b>
<b>7</b>	<b>Component Structure</b>	<b>11</b>
7.1	Infrastructure . . . . .	11
7.2	Models . . . . .	13
7.3	Interactions . . . . .	13
7.4	Devices . . . . .	13
<b>8</b>	<b>Releases</b>	<b>14</b>
<b>9</b>	<b>Apache License</b>	<b>14</b>
9.1	TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION . . . . .	14
9.2	1. Definitions. . . . .	14
9.3	2. Grant of Copyright License. . . . .	15
9.4	3. Grant of Patent License. . . . .	15
9.5	4. Redistribution. . . . .	15
9.6	5. Submission of Contributions. . . . .	16

9.7	6. Trademarks. . . . .	16
9.8	7. Disclaimer of Warranty. . . . .	16
9.9	8. Limitation of Liability. . . . .	16
9.10	9. Accepting Warranty or Additional Liability. . . . .	16
9.11	APPENDIX: How to apply the Apache License to your work . . . . .	16
<b>10 Other Resources</b>		<b>17</b>

---



iMSTK is a free & open source C++ toolkit for the prototyping of interactive multi-modal surgical simulations. The iMSTK-Unity asset gives you access to its capabilities through the Unity authoring interface. While this asset is still under development you can already exercise a variety of parts of iMSTK inside of Unity. This guide will help you get acquainted with the architecture and workings of the plugin.

The latest version of this documentation can be found on [ReadTheDocs](#)

## iMSTK-Unity

### *User Documentation*

## 1 Introduction

This document describes the iMSTK-Unity interface as well as how to build and develop it.

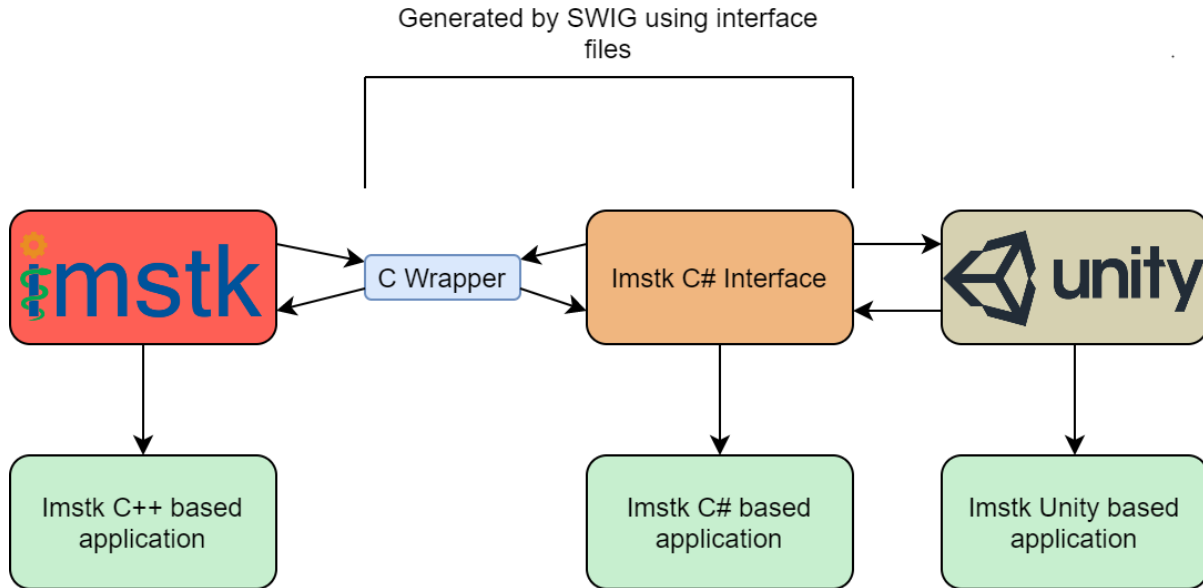
### 1.1 Overview

iMSTK is a free & open source C++ toolkit for prototyping interactive multi-modal surgical simulations. The iMSTK-Unity asset provides classes that allow the use of iMSTK features inside of unity, those are amongst others various rigid and deformable models, collision detection and response, geometric importers and filters.

The iMSTK-Unity classes utilizes a C# interface of iMSTK that is generated by using the [SWIG](#) tool. This interface is wrapped around most C++ functions and forwards its calls to the original C++ code. This allows you to use iMSTK in C# almost the same way as you would in C++.

### 1.2 Limitations

Not all iMSTK classes are currently accessible through the Unity layer. Additionally there may be some combinations that have not been tested. Some errors *may* cause Unity to crash, save early and save often. Please report issues you encounter in the iMSTK-Unity [issue tracker](#)



While there is an OpenHaptics asset available on the [Unity Asset Store](#). This is currently unsupported by the iMSTK Unity Asset. If you need support for the 3D-Touch series of devices you will have to build iMSTK with the appropriate settings.

### 1.3 Project Structure

The files of the asset are split in the following directories:

- **Scripts:** This directory is for the runtime scripts of the plugin.
- **Scripts/Editor:** This directory is for the editor scripts of the plugin. These scripts implement custom editor functionality. Editor scripts may use runtime scripts but won't be included in the player, which means you should not use functions from these in your own scripts
- **Resources:** This directory contains resources required during runtime.
- **Editor/Resources:** This directory contains resources required in the editor (such as style sheets or UI markup).
- **Plugins:** This directory is for libraries (dlls/so) that Unity needs to load.

Directories needed for the project but not relevant to the plugin:

- **Models, Materials, & Textures:** These directories are for various assets used by the Demo. They are not part of the plugin.

The asset on the store may not contain the latest version of iMSTK-Unity, if you want to be up to date you can check out the sources [from gitlab](#). As the binary files for iMSTK aren't included in the repository, you will also have to build iMSTK yourself when you go this route.

## 2 Setup for Development

When checking out iMSTK-Unity from gitlab you will first need to build iMSTK. iMSTK uses CMake for its build system. iMSTK is a superbuild meaning it builds, includes, and links to all of its dependencies. There is no need to go find them. To build iMSTK clone [the iMSTK repository](#) and build it using cmake. Turn on the `IMSTK_BUILD_FOR_UNITY` cmake switch. This will reduce the number of dependencies that are being built. It will

also enable the generation of “.cs” files as well as “.dlls”/”.so” files in your install directory that are needed by the “.cs” code. These will be used in Unity. Optionally enable `iMSTK_USE_VRPN` and/or `iMSTK_USE_OpenHaptics` for device support.

You can install the binaries two different ways, for both methods you need to know the directory in which iMSTK was installed. If you didn’t change anything that directory will be called *install* and resides inside the directory that you defined in cmake to build imstk, e.g. `C:\Projects\imstk\build\install`:

- Using **`InstallImstk.bat`** from the asset directory the format is `InstallImstk.bat <path to imstk/install>`
- Alternatively you can enable *Developer Mode* for iMSTK inside of Unity. This option can be found in “Edit->Project Settings->iMSTK Settings”. When turned on, you will be prompted to install iMSTK whenever you start Unity. After turning on this option, you may need to restart Unity. When it prompts you, select yes and provide your iMSTK install directory, located in `<iMSTK build directory>/install`. All the necessary files will be copied. If you don’t want to be asked to reinstall iMSTK toggle this option off again.

If you make changes in iMSTK you must reinstall it to Unity.

Note: There is a ‘Force Install’ button to install iMSTK into Unity at the instant it is clicked. This may not always work depending on if an iMSTK script (that utilizes a given dll) has run in the editor. In that case those dll’s cannot be unloaded from Unity.

Note: Make sure Unity is not running, even though closed it may be running in the background and prevent any installation.

## 3 Devices

The asset available from the asset store does not have [OpenHaptics](#) or [VRPN](#) enabled. To use external devices you will have to build iMSTK. When building, enable `iMSTK_USE_VRPN` or `iMSTK_USE_OpenHaptics` respectively. This will create a version of iMSTK with device support enabled. When the build is done install iMSTK into the Unity Asset as described above.

### 3.1 Debugging

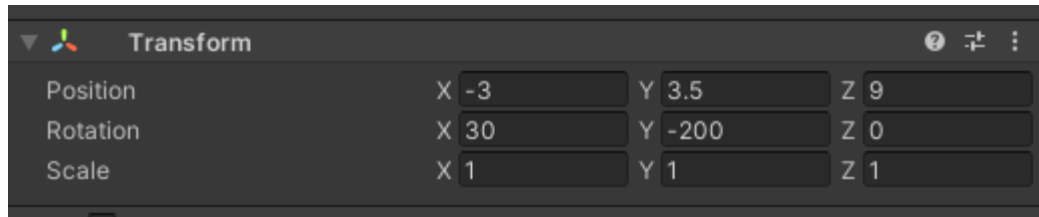
Debugging may be done in visual studio, even on the native side. You must attach the debugger to Unity as you would to any external process. With iMSTK open in visual studio, click debug->attach to process. Select the Unity executable and make sure you’re set on native debugging.

## 4 Tutorial

In this tutorial we are going to assemble a simple scene containing a deformable object colliding with static scenery. The completed tutorial is available as the *Tutorial* scene in the scene folder.

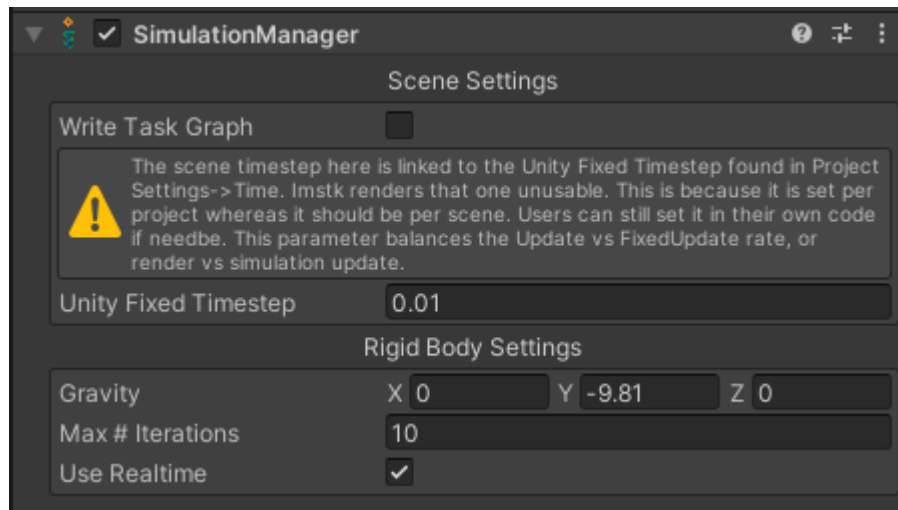
For the purpose of this guides component names will be written in a mono-spaced font e.g. `SimulationManager`, whereas the names of GameObjects or EditorFields will be in italics e.g. *HeartModel*

- Create a new scene in Unity
- Set the *Main Camera* transform to the values as seen in the image below



- Create a new gameobject and add a `SimulationManager` component to it

The `SimulationManager` is a required component for all simulations using `iMSTK-Unity`, it controls initialization and updates and also lets you set a number of global variables. Set up the manager parameters as shown in the picture by checking the *Use Realtime* checkbox.



- Create an empty `GameObject`, add a `PBDModel` component and name it *HeartModel*.

`PBDModel` is one of the classes that `iMSTK-Unity` offers, it implements the `PBD`<sup>1</sup> and `XPBD`<sup>2</sup> methods for simulating deformable objects.

We will add geometry separately but you can also use the items under the menu `iMSTK\GameObject` to add very simple objects with algorithmically generated meshes.

The `PBDModel` has a large number of parameters but we will focus only on a few. This will eventually become a deformable object but there is still a lot of work to do. For more information look at the [iMSTK Documentation](#)

We will come back to this later.

- From the *Models* folder add model named *heart* under the *HeartModel* `GameObject`

This will serve as the visual representation of what the simulation is doing, to make it look better, get the *flesh* material from the *Materials* folder and assign it as well.

With the *heart* `GameObject` selected add a `GeometryFilter` component to it and drag the *Mesh-Filter* component, the name will be *Heart\_mesh\_surface (Mesh Filter)* into the *Mesh* field of the

<sup>1</sup>

M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff, "Position based dynamics," *Journal of Visual Communication and Image Representation*, vol. 18, no. 2, pp. 109–118, Apr. 2007, doi: 10.1016/j.jvcir.2007.01.005.

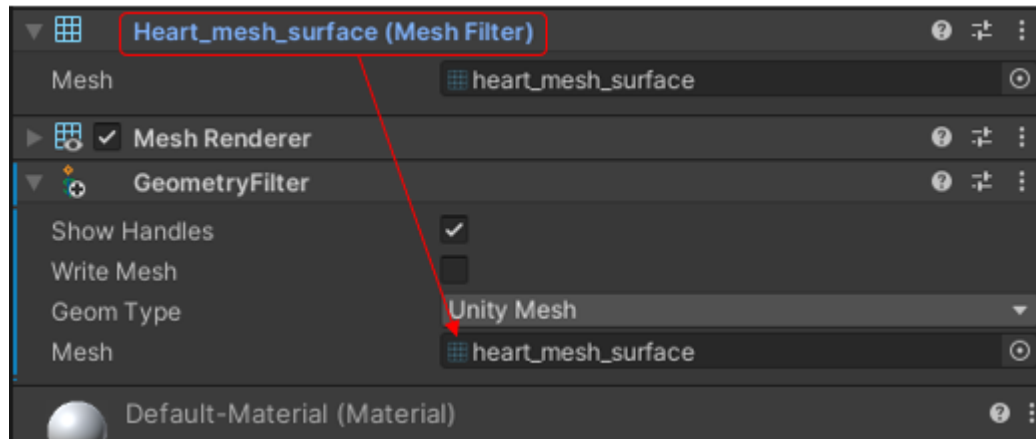
<sup>2</sup>

M. Macklin, M. Müller, and N. Chentanez, "XPBD: position-based simulation of compliant constrained dynamics," in *Proceedings of the 9th International Conference on Motion in Games, Burlingame California, Oct. 2016*, pp. 49–54. doi: 10.1145/2994258.2994272.

GeometryFilter. The drop down menu should say *Unity Mesh*. This makes this mesh available to iMSTK-Unity.

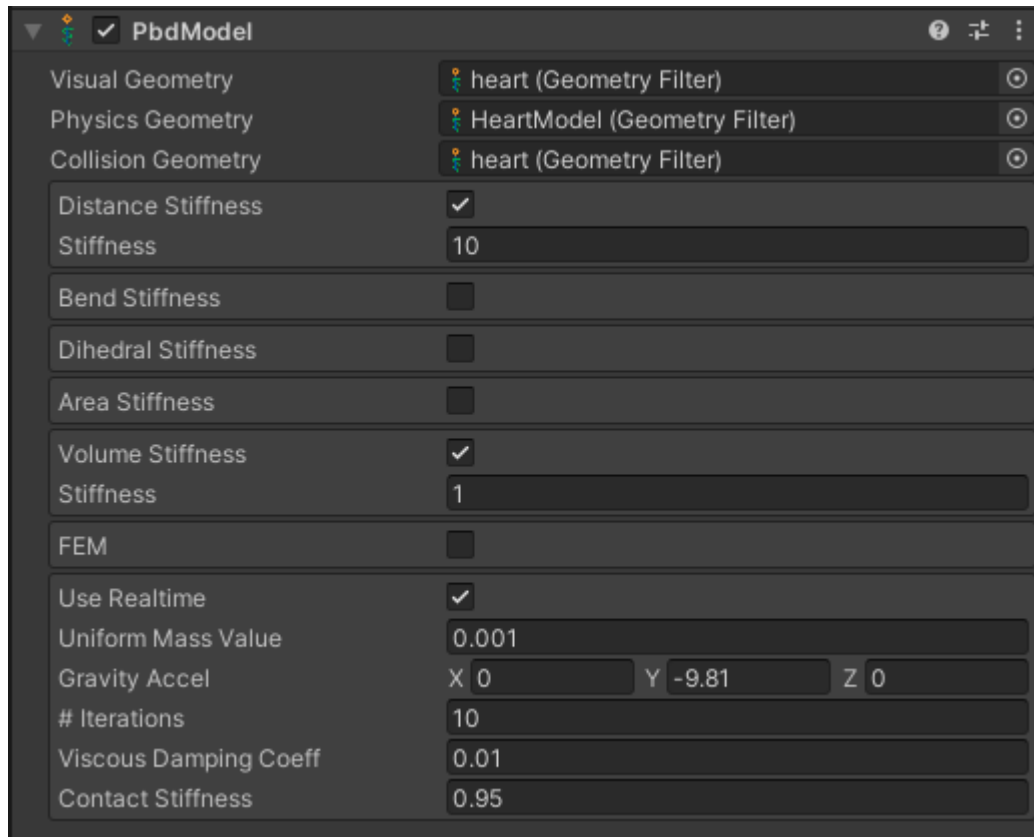
For every kind of geometry you want to use in the PBD model, iMSTK-Unity needs a GeometryFilter. This maps the Unity type to something that iMSTK can understand. It can also be used to define fixed shapes like Plane or Capsule. For meshes the source of a Geometry filter can either be a Unity mesh in the scene, a mesh asset, or (especially for tetrahedral meshes) an asset imported by the geometry importer of iMSTK-Unity.

You can use the check box named *Show Handles* to verify that the mesh is in the correct location



- Set up the parameters for our PBDModel

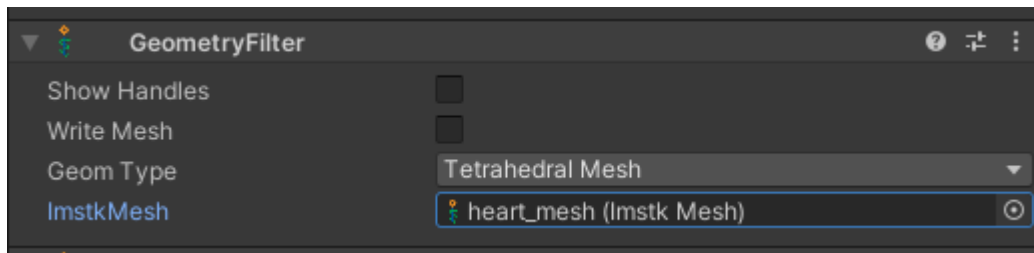
First check the *Distance Stiffness*, *Volume Stiffness* and *Use Realtime* checkboxes and fill in the properties as you see in the image below. This sets the material properties of this object. Uncheck all other stiffness options



- Add a GeometryFilter to use in the simulation

We will need a mesh to use as the geometry for calculating the physical behavior of our object. Add a GeometryFilter to the *HeartModel* object. As we will use a tetrahedral mesh, the method to assign the mesh is slightly different than what we used before. First Select *Tetrahedral Mesh* in the drop down menu. Then click on the *o* icon to the right of the *Mesh* input field. This will bring up an input dialog. Select the “Assets” tab, and double click the item name *heart\_mesh*.

As you can see the GeometryFilter component can be used for meshes in the scene or just assets of the project.



- Now we will set the shapes that are being used for simulation and visualization.

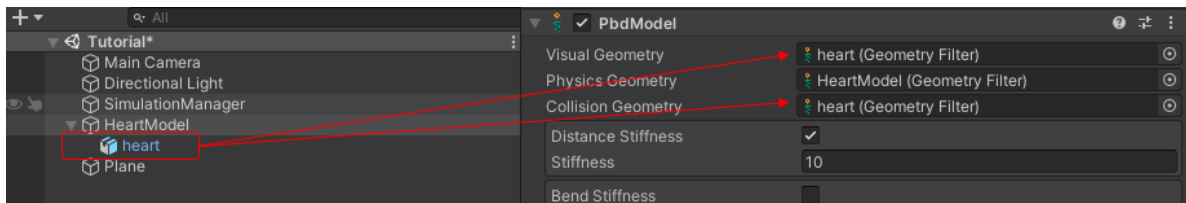
The PBDModel uses three different geometries

- *Visual Geometry* is the geometry that is being shown on the screen, this is usually some textured mesh
- *Physics Geometry* is the geometry that is being used in the simulation
- *Collision Geometry* is the geometry that is being used to determine collisions with other objects.

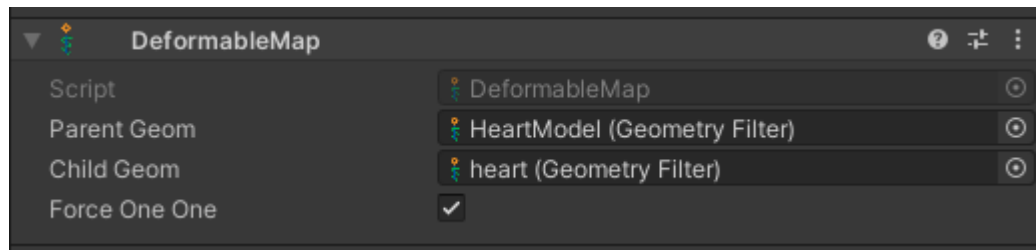
The tetrahedral mesh that was set up in the previous step will be used for the *Physics Geometry*; the other mesh

from earlier will be used for the two other geometries.

First Drag the `Geometry Filter` that you just created into the *Physics Geometry* of the `PbdModel` component. Then drag the *heart* GameObject from the hierarchy view to both the *Visual Geometry* and the *Collision Geometry* fields.



As we used different geometries for visualization and simulation we need a way to keep those in the same state, this is the responsibility of the `GeometryMap` component. Add one and drag the *HeartModel* Gameobject into the *Parent Geom* slot. Then drag the *heart* GameObject into the *Child Geom* slot. Additionally make sure the *Force One One* is checked.



This concludes the setup for the `PbdModel` object.

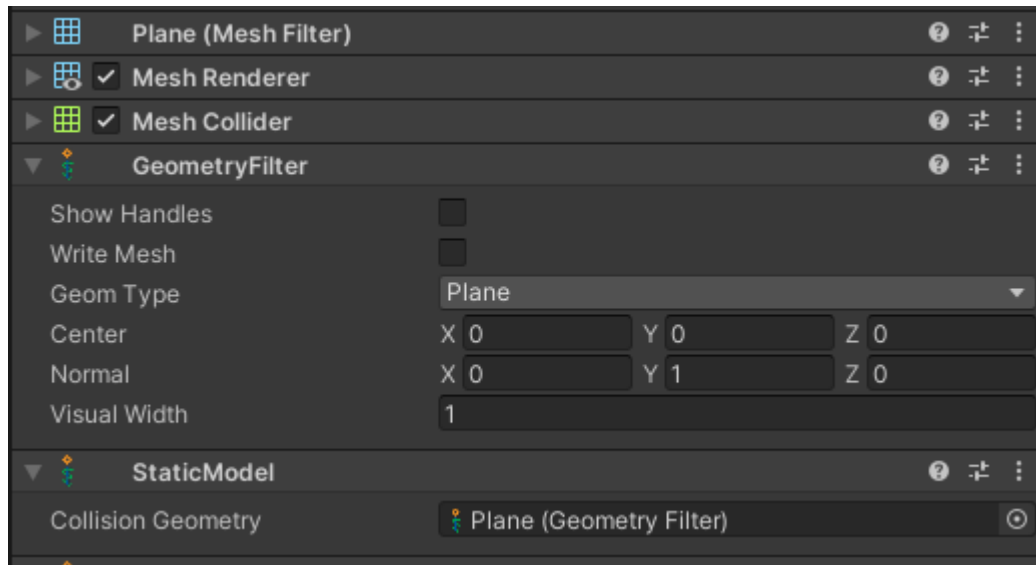
You should be able to run the scene now but as there are no other objects to interact with the heart will just succumb to gravity and drop on the ground.

- Let's add a plane for collisions

Instead of meshes we will use a fixed shape for the other side of the collision. In the hierarchy view add a `Plane` and move it to a position of 0.0, -2.5, and 0.0. Add a `Geometry Filter` to the plane object and select *Plane* in the drop down menu. The default settings for the plane will work, its position and normal will be calculated from the transform. Even though the visual mesh of the plan is finite in the editor, with regards to iMSTK this plane is infinite.

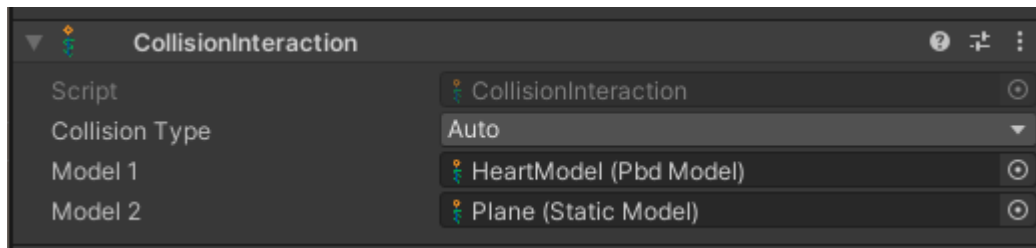
To enable the plane to interact with other iMSTK objects we need to set up a model for it as well. Add a `StaticModel` component to the *Plane* object and drag the `GeometryFilter` component into the *Collision Geometry* field. A `StaticModel` represents an object that participates in collision but doesn't react.



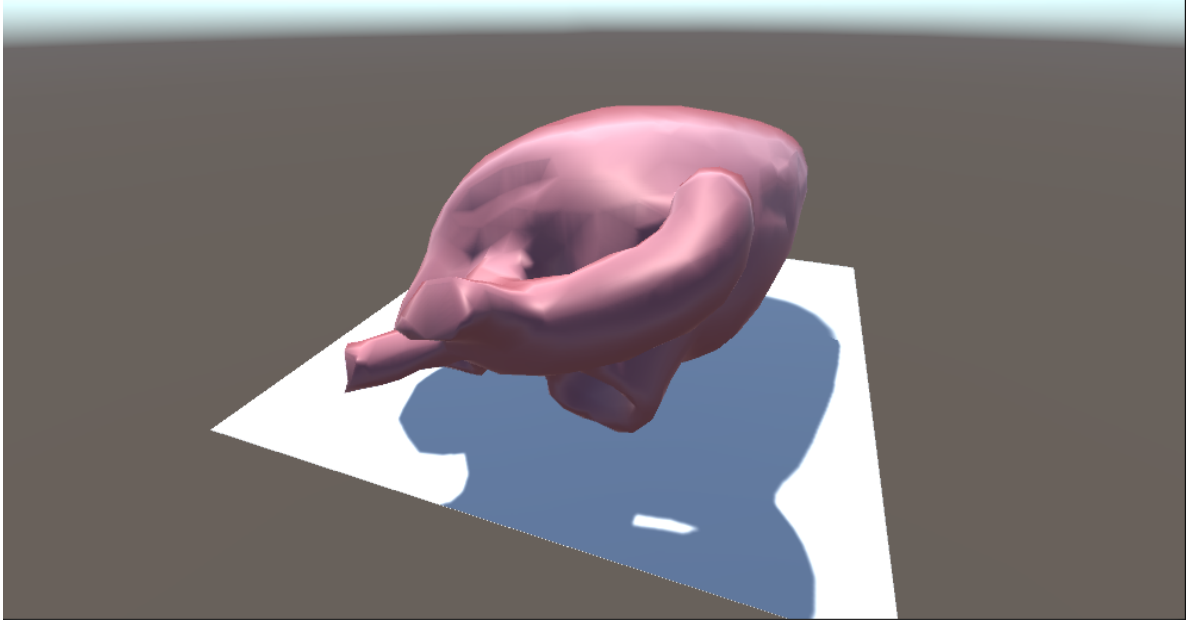


- Add the interaction between the Heart and the Plane

iMSTK needs to know about which objects can interact with each other, in this case we want the heart and the plane to collide with each other. Add a `CollisionInteraction` to the plane object. Drag the *HeartModel* into the slot named *Model 1* and drag the plane itself into the slot named *Model 2*. The drop down menu named *Collision Type* can be left in the setting *Auto*. At the moment there is most likely no need to select the *Collision Type* yourself, the automatic mechanism should be sufficient.



- This concludes the tutorial scene setup, press play you should see the deformable object hit the plane and bounce slightly on it.



Please note that at the moment iMSTK-Unity is very sensitive to misconfigurations and may cause Unity to crash, we are working to improve the error handling and stability both on the Unity asset side and inside the iMSTK sources.

## 5 Example Scenes

### 5.1 PbdClothCollision

A scene with a freely moving deformable item (Cloth) that demonstrates how to set up a deformable (PBDCloth) with and various static obstacles.

### 5.2 PbdClothScene

A cloth constrained on the top that demonstrates how to set up boundary conditions on a PbdObject

### 5.3 PbdThread

This demonstrates a line mesh that can be used as a thread

### 5.4 RbdScene

This is a simple example of two rigid spheres colliding with each other and the scenery. Uses geometric shapes rather than meshes, static colliders and collisions between dynamic objects

### 5.5 Tutorial

A scene that is used in the tutorial. It uses a deformable model colliding with a plane

## 5.6 Devices

The scenes in the devices folder can only be used with VRPN or OpenHaptics built. You will have to build iMSTK and install it into the asset as described in *Setup for Development* to support these features

### RbdController

This sets up a `OpenHapticsDevice` with a `RbdModel` and a `RigidController` to show how these pieces are interactive

### RbdControllerVRPN

Uses the `VRPNDevice` in the same scene as the *RBDController*

### TissueContact and PbdThinTissueContact

Demonstrates the use of a rigid with haptics interacting with two kinds of simulated tissue

## 6 Usage

For minimal usage of iMSTK-Unity, two things must be added to a Unity Scene.

- A `GameObject` with a `SimulationManager` attached to it
- A `GameObject` with a `Model` attached to it.

Commonly a `PhysicsGeometry` is also needed on the `Model` `GameObject`.

## 7 Component Structure

While the iMSTK C# wrapper supports almost all iMSTK classes. There is a subset that is made available as Unity components. These can be assembled in the editor to create simulations using iMSTK inside of Unity. The following section describes the roles and responsibilities of the available iMSTK-Unity classes

### 7.1 Infrastructure

#### SimulationManager

This is a component responsible for controlling the simulation. There may only exist one. It also controls the construction, initialization, and destruction of `iMSTKBehaviour` to ensure execution ordering:

- Simulation Manager created
- iMSTK objects created and internally initialized
- iMSTK objects externally initialized
- `SimulationManager` Start
- Updates
- iMSTK objects cleaned up

- `SimulationManager` cleaned up.

This component is required to be in the scene for simulations to run. It is created before any other iMSTK components on any `GameObject`. It implements the start, stop, pause, and other global scene related tasks.

### **iMSTKBehaviour**

An extension of `MonoBehaviour` to provide different callbacks for special construction, initialization, and destruction ordering.

### **GeometryFilter**

Similar to a `MeshFilter` in Unity. It provides an input and output geometry. It may take in any iMSTK geometry, as well as a Unity Mesh (one can also drag/drop a `MeshFilter` to it). These are instances of geometries used in all of iMSTK unity scripts.

### **GeometryMap**

Allows the use of separate meshes for the deformable, visual and collision representation. Will move the vertices of the target mesh according to matching points on the source mesh. The points do not have to completely coincide.

### **RigidController**

Object used between a device handled by the user and a `RbdModel`. It utilizes a mass spring system to correct for latency in the system. It corrects for problems with haptics in simulation systems. By manipulating the spring parameters the haptic response can be tuned to the behavior of the computer and the simulated system.

### **BoundaryCondition**

Use this behavior to mark vertices on a deformable object as *fixed*. This means they won't move but are still part of the overall system. In general this will mean that the object will be attached to the points selected. As the shape assigned can be any mesh this is an easy way to fix an object in space.

## **Importers**

iMSTK-Unity provides a custom Unity importer to import geometry using iMSTK. This can import point, line, surface, tetrahedral, & hexahedral meshes (vtk, vtu, stl, ply, veg, ...). If the mesh imported is a point, line, or surface mesh then it will be imported as a Unity Mesh object. Anything else not supported by Unity, is loaded as an iMSTK-Unity Geometry Object. When a volumetric mesh (such as a tetrahedral mesh) is imported the accompanying surface is extracted and provided as an additional asset.

## **Editor Scripts**

iMSTK-Unity provides extensions to the Unity editor. These extensions include:

- Custom inspectors for the models and geometry components.
- A global settings menu.
- Menu Items for quick creation of `GameObject` with iMSTK items already setup.
- Editors/windows for various operations

## 7.2 Models

### **PbdModel**

Use this to represent deformable objects. Position Based Dynamics (PBD), is used to model the deformation. This model supports Lines (1D), Surface Meshes (2D) and, Tetrahedral Meshes (3D) dynamical models see the [iMSTK Documentation](#) for more information on constraints and models. Visual, physics and collision geometry can be assigned separately. If you do, a separate map will be necessary to update the various meshes.

The physics geometry determines the type of constraint that can be used, an invalid constraint may cause problems.

Table 1: Valid Constraint Combinations

Physics Geometry Type	Valid Constraints
Line Mesh (Threads)	Distance Stiffness, Bend Stiffness
Surface Mesh (Membranes, Bags)	Distance Stiffness, Dihedral Stiffness, Area Stiffness
Volumetric Mesh (Tissue)	Distance Stiffness, Volume Stiffness, Fem (all models)

### **RbdModel**

Use this to represent moveable rigid objects like forceps or scalpels. Visual and collision geometry can be assigned separately, this behavior will update the transform of the GameObject with each update. For more information see the [iMSTK Documentation](#)

### **StaticModel**

Use this to represent un-moveable rigid objects like the ground plane or other obstacles.

## 7.3 Interactions

### **CollisionInteraction**

Use this behavior to set up collisions between two objects, in general this behavior can detect what the type of the two objects is that are interacting (mode *Auto*). But you can also select the algorithm that should be used.

### **PbdObjectInteraction**

A collision interaction specific to PBD models, it allows setting of values specific to interactions with PBDs.

### **PbdRigidGraspingInteraction**

An interaction that allows the grasping of objects, when activated it will attach a point on the rigid to a point on the deformable object and will attempt to keep the two in the same place.

## 7.4 Devices

### **OpenHapticsDevice**

This device is only available with a custom build of iMSTK. It enables the use of the [3DSystems](#) haptic device family.

## VrpnDevice

This device is only available with a custom build of iMSTK. It enables interactions with devices run by a [VRPN](#) server.

## 8 Releases

### 2022-jul v1.0 First release (amongst others):

- iMSTK Version 6.0
- Support to import VTK and other mesh formats (e.g. vtk, vtu, stl, ply, veg, ...)
- Deformable & Rigid Body Models to be used in Unity
- Custom Editors for iMSTK Behaviors
- Helper Scripts to create simple dynamic objects

## 9 Apache License

**Version** 2.0

**Date** January 2004

**URL** <http://www.apache.org/licenses/>

### 9.1 TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

#### 9.2 1. Definitions.

**“License”** shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

**“Licensor”** shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

**“Legal Entity”** shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, “control” means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

**“You”** (or **“Your”**) shall mean an individual or Legal Entity exercising permissions granted by this License.

**“Source”** form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

**“Object”** form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

**“Work”** shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

**“Derivative Works”** shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

**“Contribution”** shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, “submitted” means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as “Not a Contribution.”

**“Contributor”** shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

## 9.3 2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

## 9.4 3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

## 9.5 4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

- You must give any other recipients of the Work or Derivative Works a copy of this License; and
- You must cause any modified files to carry prominent notices stating that You changed the files; and
- You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
- If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License. You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

## 9.6 5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

## 9.7 6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

## 9.8 7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an **“AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND**, either express or implied, including, without limitation, any warranties or conditions of **TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE**. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

## 9.9 8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

## 9.10 9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

## END OF TERMS AND CONDITIONS

## 9.11 APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets “[ ]” replaced with your own identifying information. (Don’t include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same “printed page” as the copyright notice for easier identification within third-party archives.



Copyright 2020 iMSTK-Unity

Licensed under the Apache License, Version 2.0 (the "License");  
you may **not** use this file **except in** compliance **with** the License.  
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law **or** agreed to **in** writing, software  
distributed under the License **is** distributed on an "AS IS" BASIS,  
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express **or** implied.  
See the License **for** the specific language governing permissions **and**  
limitations under the License.

## 10 Other Resources

- iMSTK Discourse Forum
- iMSTK-Unity Gitlab
- iMSTK Documentation
- iMSTK Gitlab

