2017

IMPETUS-VR Documentation

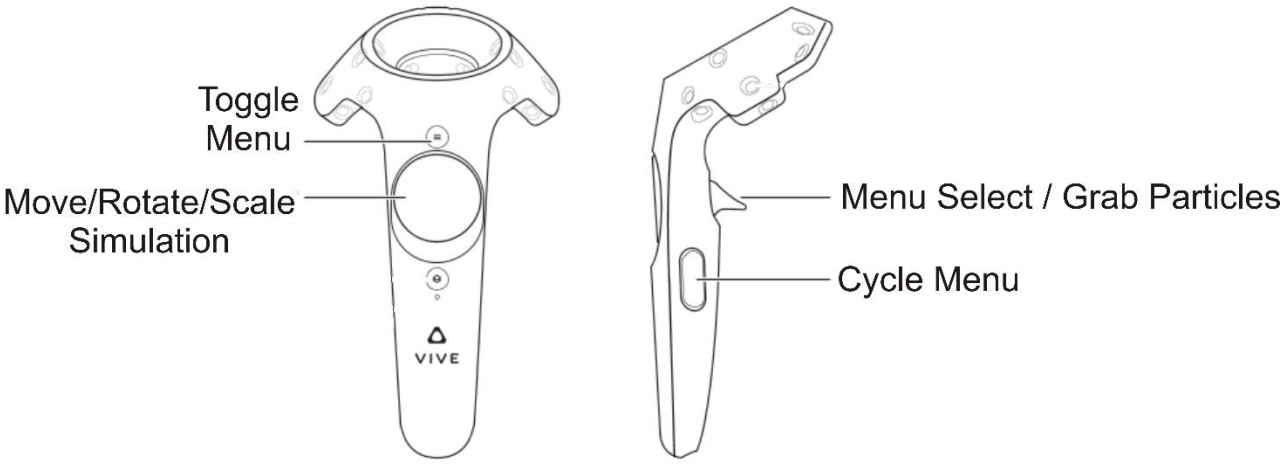
CELLULAR MECANICS LABORATORY
UNIVERSITY OF CONNECTICUT

Table of Contents

Introduction

This is the documentation and user guide for IMPETUS-VR. This software enables users to not only view but to manually interact with their simulations in virtual reality. For a brief overview of the program, please see the accompanying demonstration video.

Controls



Toggle Menu

The user can hide and show the IMPETUS-VR settings and action menus by pressing the Toggle Menu button. Menus appear in front and slightly above the left controller. In order to select a menu option, the user must overlap the red sphere floating slightly in front of the right controller with the desired menu button and pull the trigger on the back of the right controller. While the sphere overlaps a menu button it is highlighted. While a menu is active, the controllers cannot be used to grab or pull particles.

Move, Rotate, and Scale Simulate

These controller buttons allow the user to translate the simulation in virtual reality with the right controller, rotate it relative to the user with the left controller, or rescale the simulation using both at once in the virtual reality space. These changes do not affect the simulation itself, rather how it is displayed in virtual reality.

The simulations are translated, rotated, or scaled only while the controller button is held down. While this button is held down on just the right controller the Simulation is translated in the same directions as the controller itself. While this button is held down on just the left controller, the simulation is rotated about its center by the translation of the left controller. While this button is held down on both controllers, the simulation is scaled based on the change in distance between the two controllers, that is, bringing the two controller closer together shrinks the simulation and brining the two controllers apart expands the simulation.

Menu Select / Grab Particles

The trigger controller button serves three distinct purposes. When a menu is active, this trigger button can only be used to select a menu option by hovering the menu selection sphere over the desired menu button and pulling the trigger on the right controller. While a menu is active all other functions of the trigger button and all function of the left trigger button are disabled.

If the menus are hidden, the trigger buttons can be used to interact with the simulation itself. The user can “grab” particles inside of the simulation. All particles relatively close to the controllers (defined as a sphere around the controllers) will be “grabbed” when the trigger button is pressed. The particles will continue to be “grabbed” by each individual controller while the triggers are held down, and are released when the trigger is released. The functionality of a “grab” changes depending on whether the simulation is paused.

If the simulation has been paused, particles will follow the grabbing controller by rotating and translating to match the controller. If the simulation is not paused, the grabbed particles will experience a steering force towards the center of the controller. The amplitude of this steering for is directly proportional to the distance of the particles to the center of the controller in the simulation space. This acts like a spring between the particles and the controller.

Cycle Menu

This button switches which menu is active when the menus are visible.

Menus

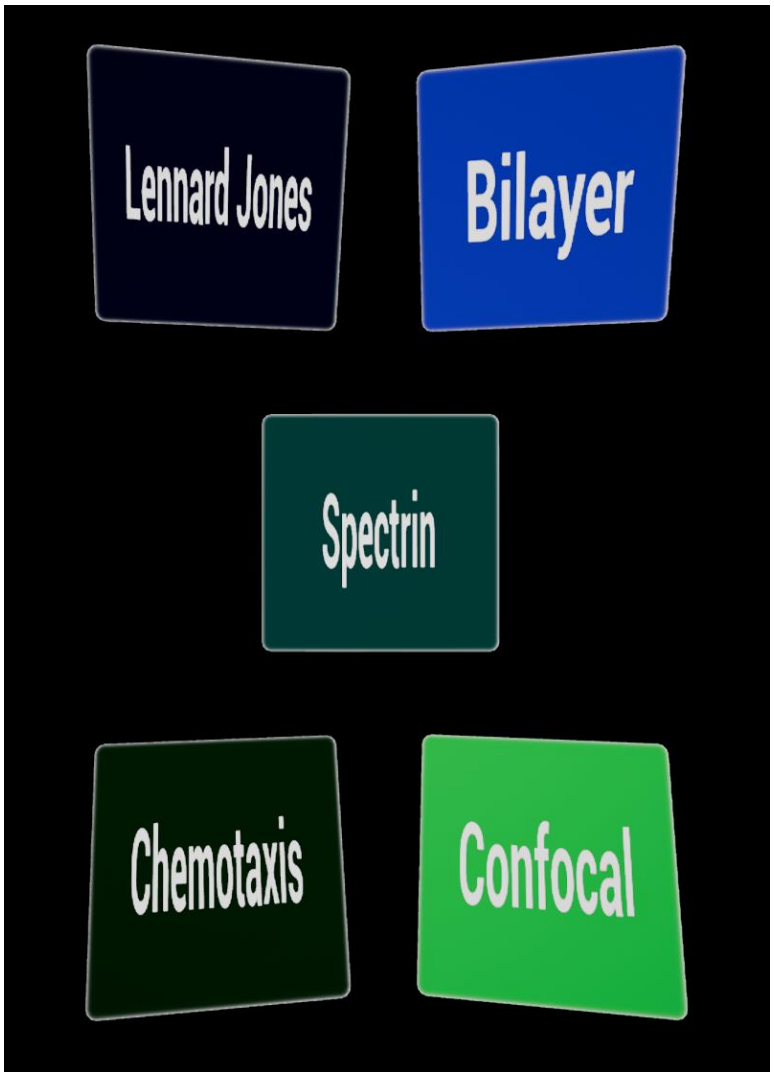
The following menus are available to the users

Main Menu

From the main menu, users can

1. **Play/Pause:** pause and unpause the simulation
2. **Clear:** clear the simulation
3. **Toggle Wands:** hide the controllers
4. **Toggle Hints:** hide the controller hints
5. **Exit:** exit the program

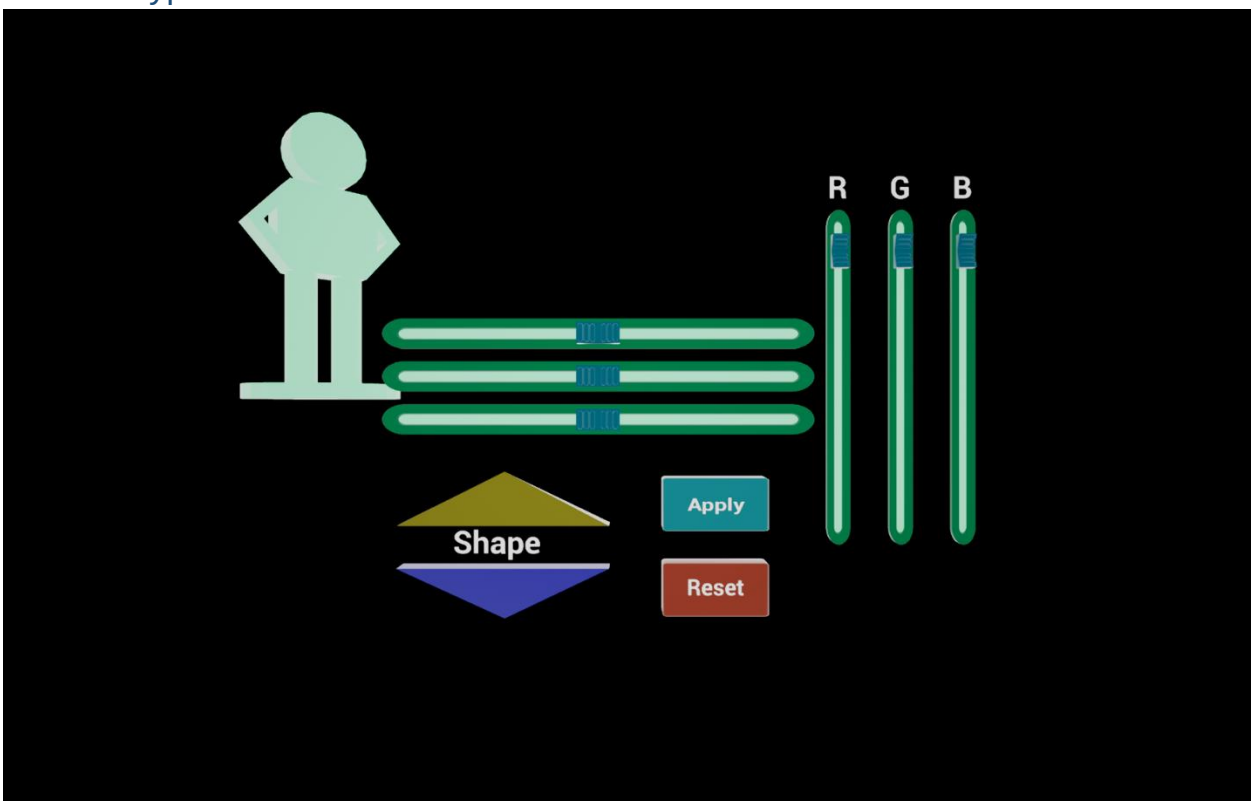
Simulation Menu



A number of example simulations are available “out of the box.” These include the follow:

1. **Leonard-Jones**: a particle simulation based on the Lennard-Jones pair potential in a canonical ensemble
2. **Bilayer**: a solvent-free model of a phospholipid bilayer
3. **Spectrin**: a simulation of the red blood cell plasma membrane skeleton
4. **Chemotaxis:** a model of the interaction between cranial neural crest and cranial placode cells during collective cell migration
5. **Confocal**: a representation of a $102\ \mu m$ x $102\ \mu m$ x $19\ \mu m$ confocal microscopy image of a kidney mouse section

Particle Type Menus



Each simulation dynamically creates menus to edit the visualization of each kind of particle. Each individual particle type has a corresponding particle type menu. A sample of each particle is displayed as part of the menu. Any desired changes only change the sample mesh until the “Apply” button is pressed, at which point the sample mesh is applied to all particles of the corresponding type in the simulation. The “reset” button can be used to bring the sample mesh back to what it represents in the simulation. Using the shape buttons, users can cycle through different meshes that can be applied. Be aware that in particularly large simulation a mesh with a high amount of detail (polygons) may cause the program to stutter. The RBG sliders allow the user to adjust the color of the particle type in terms of a red, green, and blue color selection. The size sliders adjust the size of each particle type mesh in the 3 dimensions.

Visualizing Custom Volumetric Renders

In order to visualize custom volumetric images, such as those produce in confocal microscopy, the cross-sectional slices of the image need to placed in the following folder

"…\[IMPETUS-VR directory]\Source\ImpetusVive\UIModels\Confocal\input\ConfocalInputs"

This folder initially contains the mouse kidney volumetric images. These mouse kidney images should first be removed from this location. The software assumes that you are attempting to visualize square images thus if your source images are not square add an equally sized black border on either "short" side to create square images.

You will also need to adjust the input parameter file found one directory above in "\input" called "param.input" with any basic text editor such as notepad. Be sure not to alter any of the input keywords, all of which start with [IN] and are case sensitive. If there is a typo in the key words, a default value will be chosen instead of your input value. Input the desired value on the line directly following the keyword. Any other inputs within this file besides the keywords and the lines directly following the keywords are ignored. The following keywords can be set

[IN] slice gap

This value controls the distance between each cross-sectional slice in UE4 units. Setting this value properly will require you to do some unit conversations. For reference, every cross-section is displayed as a 100 UE4 unit by 100 UE4 unit plane.

[IN] alpha contrast

This value controls the contrast of the images in terms of only the transparency. The lower the value below 1, the greater the contrast is increased. Values of around 0.8 are recommended.

[IN] brightness

This value controls the brightness of each slice, where 1 represents its original brightness.

[IN] red amplitude

This value controls the brightness of all red color in each slice **before** the transparency of each pixel is calculated. Here 1 represents the original brightness. Setting this value at 0 removes red from the final render. This value should be set between 0 and 1.

[IN] green amplitude

This value controls the brightness of all green color in each slice **before** the transparency of each pixel is calculated. Here 1 represents the original brightness. Setting this value at 0 removes green from the final render. This value should be set between 0 and 1.

[IN] blue amplitude

This value controls the brightness of all blue color in each slice **before** the transparency of each pixel is calculated. Here 1 represents the original brightness. Setting this value at 0 removes blue from the final render. This value should be set between 0 and 1.

Code Structure

The vast majority of operating IMPETUS-VR involves using IMPETUS. For instructions on IMPETUS and the associated input files please see the previously published instructions.

In order to customize IMPETUS-VR, we have included with this manual a brief overview of the organizational structure of the IMPETUS-VR code. As mentioned previously, the visualization of IMPETUS-VR is powered by UNREAL Engine 4 which uses both C++ and a unique visual programming language named "Blueprints." All control inputs and outputs, including the IMPETUS-VR menus are written in Blueprints, while IMPETUS and the rendering are written in C++. Specifically, the class **FImpetus** runs the simulation in a thread parallel to the rendering, while the class **Simulator** and its Blueprints child class **BPSimulator** read the data generated and manipulated by **FImpetus** and visualize it. When the user interacts and manipulates the simulation or its visualization, the user inputs are interpreted by the **MotionControllerPawn** and forwarded to the **Simulator**. If the user interaction was with the simulation itself, such as movie particles or applying a steering force, the **Simulator** class passes the user inputs on to the **FImpetus** class.

FImpetus

The call to begin the static **FImpetus** thread is made by the **Simulator** class when a new simulation is loaded. When the call is made a new instance of **FImpetus** first checks that the system is capable of multithreading and whether an instance of **FImpetus** already exists. Once the new thread confirms this, the simulation is initialized by matching the input simulation ID integer with the appropriate **UIModel** and running the appropriate initialization file. Once the simulation is ready, **FImpetus** enters a paused state and waits for the user to unpause the simulation and the **Simulator** reads the data generated by **FImpetus** and the **UIModel** in order to visualize the simulation.

UIModels

Unreal-IMPETUS model (**UIModel**), is a base IMPETUS-VR class that is used to create simulation models interfacing Unreal engine and IMPETUS. The purpose of this class is to allow users to have an easy way to generate simulation models without needing to modify the source code. Upon generation of the **FImpetus** object, a pointer of **UIModel** is also created, this pointer contains no simulation models. In order to convert it into a simulation, users need to write or download a child model class. The child model class is the C++ class that contains the whole simulations and is created as a child class of **UIModel**. In order to convert **UIModel** to a full functioning simulation, the child model class needs to override only 3 functions from the original **UIModel**:

1. Initiation: Initial configurations and parameters of the simulations
2. Iteration: IMPETUS MD functions (e.g., the leapfrog integration algorithm, the Lennard-Jones potential)
3. Post-Iteration: System adjusting functions such as thermostats and barostats, and particle and system property recorders (e.g., positions, velocities, mean squared displacement)

For more information on the **UIModel** class and an example of implementation, please refer to the generic UIModel file, **GenericUIModel.h,** as well as the UIModels bundled "out of the box" with IMPETUS-VR.

Rendering

The visualization scene consists of the user tools, the simulation bounds, and the contents of the simulation itself. The user's controller representations match the physical size and location of the real world Vive controllers. These are coded in the Blueprints **MotionControllerPawn** and **BPMotionController**. The **MotionControllerPawn** also spawns and handles all menus, which inherit from the base class **MenuBase**. The visualization itself is handled by the instance of the **BPSimulator** class that is in the scene. The **BPSimulator** class inherits from the **Simulator**. All visualization is done in the C++ code of **Simulator**, and the child class **BPSimulator** itself is only responsible for aligning the Simulation bounds whenever a new simulation is loaded.

Simulator

The **Simulator** class is responsible for launching an instance of **FImpetus**, generating the visualization of each simulation, updating the visualization whenever the simulation is updated, and shutting down the **FImpetus** thread the user loads a new simulation, clears the simulation, or exits the program. This class initializes and retains the pointers to the Instanced Static Mesh Components (ISMCs) that are actively rendered by Unreal. The ISMCs are an efficient way to visualize many objects that have the same shape and color, but are different sizes, at different locations, and have different orientations. The **Simulator** class attempts to visualize three simulation components of IMPETUS, the individual particles, the "edges" that can connect some particles, and any three dimensional vector fields present in the simulation. The **Simulator** will visualize any number of different types of these components. All types of particles are stored in an array of a simple class called UParticleGroup. This array is called ArrayofParticleGroups. Each ParticleGroup instance contains the pointer to the appropriate Unreal ISMC, and some basic information about the particles, such as their world size, color, and mesh. The UParticleGroup class also contains an array of the class UParticle, called ParticleList. Each UParticle has a pointer to the corresponding IMPETUS particle as well as some basic (though in certain cases incomplete) methods to convert between the simulation coordinates and the visualization coordinates. In the same way, **Simulator** also contains the arrays ArrayofMDEdges and ArrayofVectorFields of the classes UMDEdgeGroup and UMDVectorFieldGroup respectively. These are organized in much the same way. Every rendered frame, **Simulator** checks to make sure that **FImpetus** is still a thread that is running as well as checks whether **FImpetus** has raised the "RenderFlag" Boolean signaling **Simulator** to update the visualization. If the flag is raised, **Simulator** updates the visualization and resets the flag. At the time of writing, **FImpetus** raises this flag (sets the Boolean to true) every time it computes a time step. For more information on the individual functions and components of the **Simulator** class, refer to **Simulator.cpp** and **Simulator.h** and the comments therein.

Motion Controller Pawn

All inputs and outputs, including the menus are written in the Blueprints language and are part of the **MotionControllerPawn** instance in the scene. This class is responsible for spawning the representations of the controllers and updating their positions and orientations, managing the player's camera and perspective, and handling all user button presses. New menus must be introduced in the Blueprints such that they are added to the array of menus and inherit from the **MenuBase** Blueprints class.