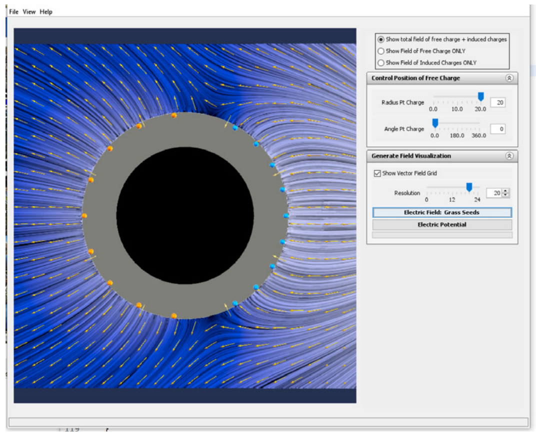
TEALsim: A Guide to the Java 3D Software


File View Help
Control Position of Free Charge
Generate Field Visualization
Electric Field: Grass Seeds
Electric Potential

(● Show total field of free charge + induced charges) ○ Show Field of Free Charge ONLY ○ Show Field of Induced Charges ONLY

Radius Pt Charge: 20

Angle Pt Charge: 0

[✓] Show Vector Field Grid

Resolution: 20

The ConductingSphericalShellShielding App

John Belcher, Andrew McKinney, Philip Bailey, Michael Danziger

The software described herein has a *github* site
https://github.com/jbelcher-mit/TEALsim.git

Comments and questions to jbelcher@mit.edu

Supported by NSF Grant DUE-0618558, the d'Arbeloff Fund for Excellence in MIT Education, the Davis Educational Foundation, iCampus, the MIT School of Science, NSF Grant DUE-9950380, the Helena Foundation, and the MIT Classes of 1951, 1955, and 1960.

Table of Contents

Figure Captions

1 Introduction

1.1 Purpose and Intended Audience

The *TEAL* simulation system, *TEALsim*, is designed as a framework for authoring, presenting, and controlling simulations related to topics in classic electromagnetism. The goal of *TEALsim* is twofold: first, it aims to provide a relatively simple, "API-style" interface, such that non-expert programmers can produce full-featured interactive electromagnetic simulations from the ground up, including all aspects of visualization, user interface and public posting, with minimal exposure to the inner workings of the system. Second, it offers a flexible framework for more experienced programmers to extend and expand the functionality of the system to suit their specific needs.

This document is intended as a general guide to building electromagnetic simulations using the *TEALsim* environment. The *TEALsim* environment was developed by the Technology Enabled Active Learning (*TEAL*) Project at MIT (http://icampus.mit.edu/teal/).

This documentation is aimed at undergraduates, graduate students or postdoctoral fellows in science and engineering who have some knowledge of programming but little if any knowledge of 3D graphics and conventions (e.g. scene graphs, branch groups, view platforms, and so on). We have assembled a set of instructions and tutorials that will enable the user with this background to create and package 3D simulations using the *TEALsim* platform. We also provide enough insight into the workings of *Java3D* for the uninitiated to have a passing knowledge of what goes on, and we provide references to books and online resources for those interested in expanding that knowledge beyond what we present here.

In the text below we use *TEALsim* and *MIT_TEALsim_2024* interchangeably.

1.2 Organization

The organization of this documentation is as follows. We first give instructions for downloading and installing the code from *github* in Section 3, including instructions for running the code in *Eclipse*, an *Integrated Development Environment*. We assume the user is on a *Windows 10* PC with *PowerShell* installed. We then give a general description of the *TEALsim* architecture in Section 4. In Section 5, we give examples of building electromagnetic simulations out of standard components in the *TEALsim* universe. In Section 6 we discuss specific features that can be built into *TEALsim* simulation. In Section 7, we discuss some of the applications in terms of their intended purpose, use, and their mathematical and physical underpinnings.

2 URL'S for Java Downloads

2.1 Java 8/1.8 and 21

We need both JDK's, since we cannot do the *ant build* to generate the jars in Java 8 (see Section 11.2). Note also that we need Java Web Start (JWS) to start our applications. JWS was deprecated in Java 9, and starting with Java 11, Oracle removed JWS from their JDK distributions. But since JWS is still contained in Java 8/1.8, we continue to use that application.

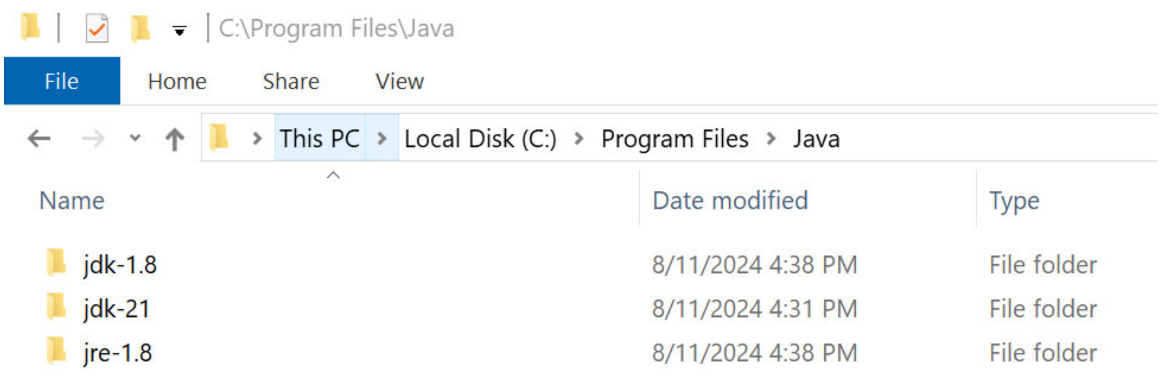
Java 8/1.8:

https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html

Java 21:

https://www.oracle.com/java/technologies/downloads/#java21

After installing both 8/1.8 and 21, my C:\Program Files\Java directory looks like this:



Name	Date modified	Type
jdk-1.8	8/11/2024 4:38 PM	File folder
jdk-21	8/11/2024 4:31 PM	File folder
jre-1.8	8/11/2024 4:38 PM	File folder

2.2 Java3d 1.5.2

We do not need to download the java3d 1.5.2 jars, as I have included them in the *github* repository, and they are available when you download the repository (*C:\Users\[user]\git\repository\MIT_TEALsim_2024\java\lib\j3d\1.5.2*). ?? For reference I give the URL where they exist:

https://web.archive.org/web/20110824143742/http://download.java.net/media/java3d/builds/release/1.5.2/

I downloaded the *j3d-1_5_2-windows.amd64.zip* for the 64-bit architecture.

NOTE: The java 3d jars as downloaded are not *properly signed*. I have re-signed them with my code signing certificate, and the signed jars will expire on August 1, 2025. See Appendix 11.1.

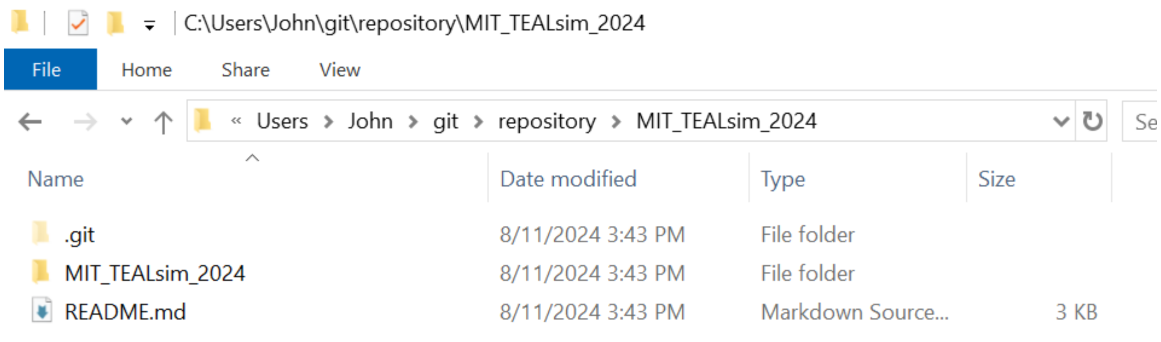
3 Setting up the *TEALsim* Project

3.1 Downloading and installing the *TEALsim* Code Base from *github*

We assume the user is familiar with *github*. On my machine, I bring up *PowerShell*, go to the directory *C:\Users\John\git\repository*, and issue the command:

```
git clone git@github.com:jbelcher-mit/TEALsim.git
```

This will create a git repository in a folder named *TEALsim,* in C:\Users\John\git\repository\MIT_TEALsim:


C:\Users\John\git\repository\MIT_TEALsim_2024

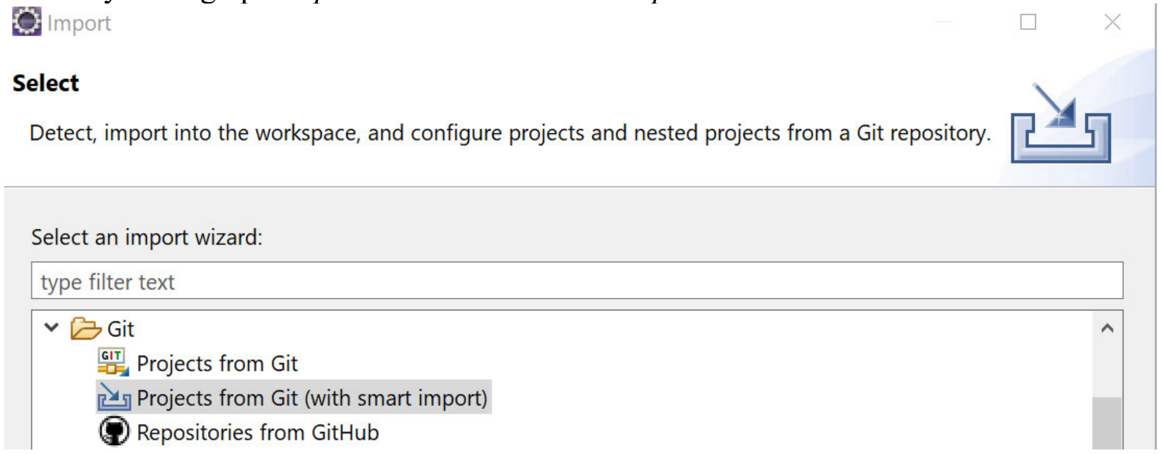
Name	Date modified	Type	Size
.git	8/11/2024 3:43 PM	File folder	
MIT_TEALsim_2024	8/11/2024 3:43 PM	File folder	
README.md	8/11/2024 3:43 PM	Markdown Source...	3 KB

3.2 Running *TEALsim* in *Eclipse* in a Windows Environment

We discuss the *TEALsim* software in the context of *Eclipse* (http://www.eclipse.org/), a free, state-of-the-art Java Integrated Development Environment (IDE). We will assume that you are familiar with the *Eclipse* IDE interface and with developing *Java* applications in this environment. We are using *Eclipse IDE 2024-06*.

3.3 Creating a *TEALsim* Java Project in *Eclipse*

We follow the standard procedure for creating a *Java Project* in Eclipse from a *github* directory. Bring up *Eclipse* and click on *File > Import*. You will see


Import

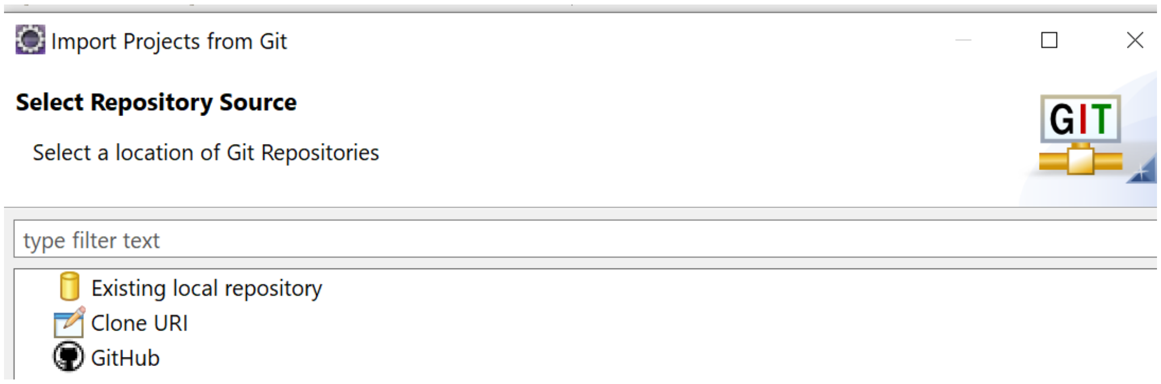
Select

Detect, import into the workspace, and configure projects and nested projects from a Git repository.

Select an import wizard:

type filter text

Choose the option indicated. Then in the subsequent panel choose “Existing local repository”:


Import Projects from Git

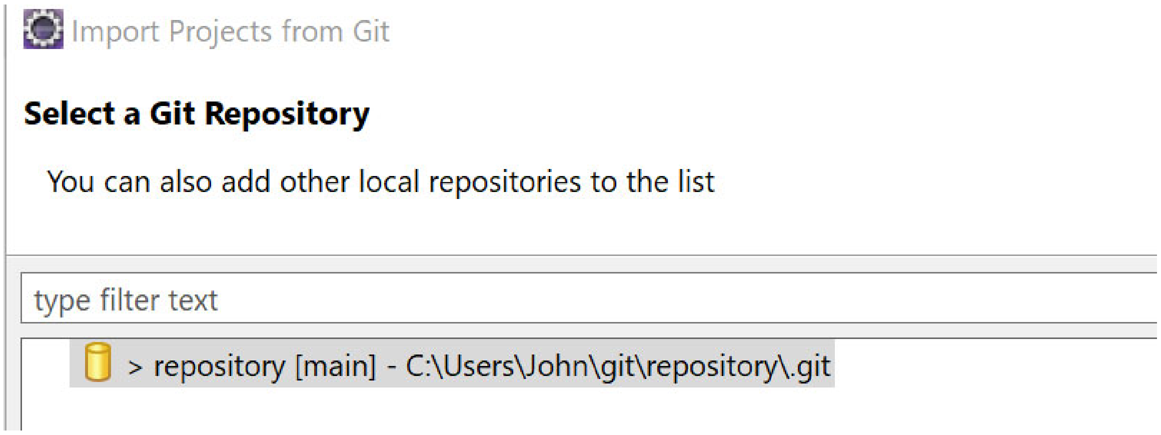
Select Repository Source

Select a location of Git Repositories

type filter text

Existing local repository
Clone URI
GitHub

Then choose *Existing local repository*. You will then see


Import Projects from Git

Select a Git Repository

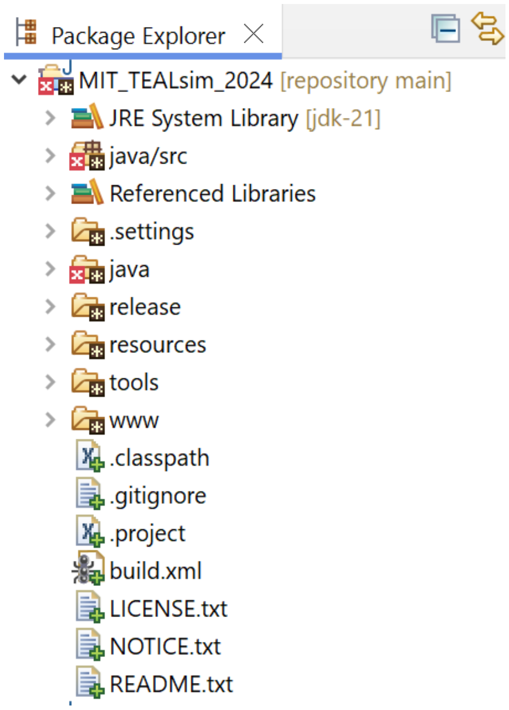
You can also add other local repositories to the list

type filter text

> repository [main] - C:\Users\John\git\repository\.git

Hit "Add" in the upper right corner of the above dialog box. Then in the dialog box following, click on *C:\Users\John\git\repositories\MIT_TEALsim_2024\.git.* Then hit *add* in the lower right bottom. Then click through to the *finish*.

The resulting appearance of the project in Eclipse is as follows:

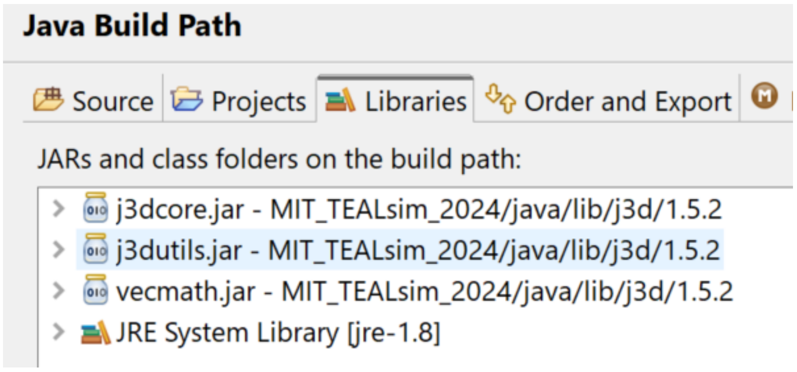


Package Explorer

MIT_TEALsim_2024 [repository main]
- JRE System Library [jdk-21]
- java/src
- Referenced Libraries
- .settings
- java
- release
- resources
- tools
- www
- .classpath
- .gitignore
- .project
- build.xml
- LICENSE.txt
- NOTICE.txt
- README.txt

Figure 3-1: Package Explorer of TEALsim

The project should have the following build path:

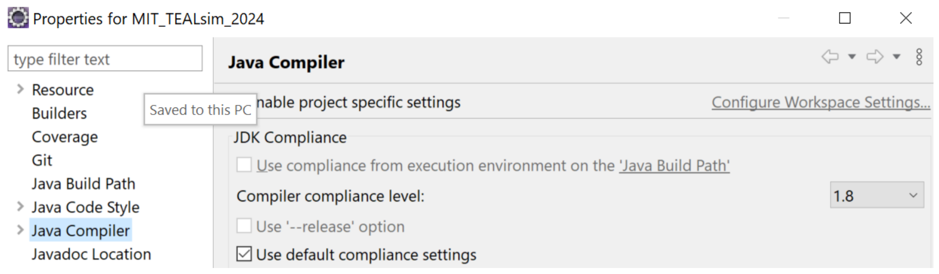


Java Build Path

JARs and class folders on the build path:
- j3dcore.jar - MIT_TEALsim_2024/java/lib/j3d/1.5.2
- j3dutils.jar - MIT_TEALsim_2024/java/lib/j3d/1.5.2
- vecmath.jar - MIT_TEALsim_2024/java/lib/j3d/1.5.2
- JRE System Library [jre-1.8]

Figure 3-2: Java Build Path

If we right click on our project name and choose *Properties* at the bottom of the menu that comes up, and choose *Java Compiler*, we should see:



Properties for MIT_TEALsim_2024

Java Compiler

Enable project specific settings

JDK Compliance

☐ Use compliance from execution environment on the 'Java Build Path'

Compiler compliance level: 1.8

☐ Use '--release' option

☑ Use default compliance settings

Figure 3-3: Java Compiler Option in Properties

3.4 Downloading j3dcore-ogl.dll

Download this file using the following link, and install it in *C:\Windows*

https://web.mit.edu/viz/EM/simulations/java3d/j3d/1.5.2/windows/j3dcore-ogl.dll

3.5 Instructions for Running a Simulation

In *Eclipse*'s *Package Explorer*, choose *SimPlayerApp.java* (see Figure 3-4) by left clicking on it.

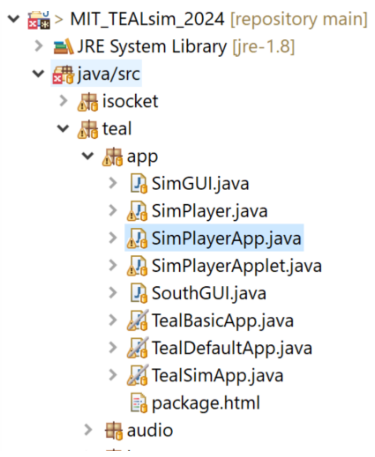
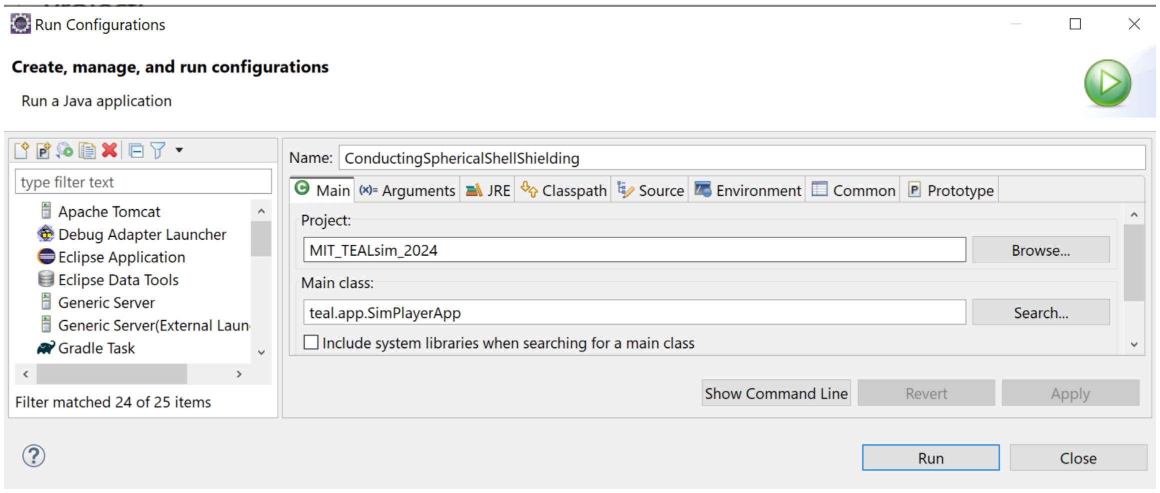


Figure 3-4: Running *SimPlayerApp*

Right click and choose "*Run as > Run Configurations"* from the menu. In the left column, click "*Java Application*". You will see a dialog box (see Figure 3.5). We want to run the application *ConductingSphericalShellShielding*. Enter "*ConductingSphericalShellShielding*" in the *Name* box. Left click on the *Arguments* tab and in the box labeled *Arguments* (see Figure 3.6). Type *"-n tealsim.physics.em.ConductingSphericalShellShielding*" in the box on top (see Figure 3-6).


Run Configurations

Create, manage, and run configurations

Run a Java application

Name: ConductingSphericalShellShielding

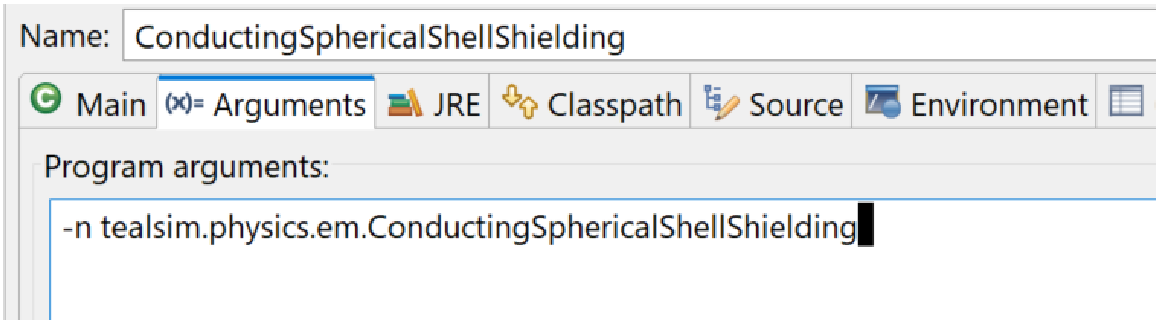
Project: MIT_TEALsim_2024

Main class: teal.app.SimPlayerApp

☐ Include system libraries when searching for a main class

Filter matched 24 of 25 items

Figure 3-5: The *Run* dialog box


Name: ConductingSphericalShellShielding

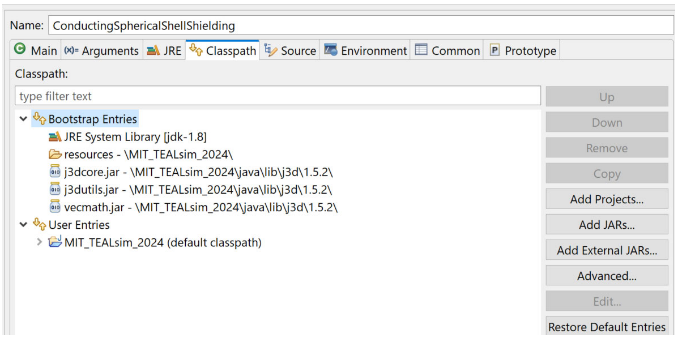
Program arguments:

```
-n tealsim.physics.em.ConductingSphericalShellShielding
```

Figure 3-6: The *Arguments* panel in the *Run* dialog box

If you have a simulation that requests additional memory, set more memory by typing "*-Xmx512m*" (or "*-Xmx256m*") in the "VM arguments:" in the "Arguments" panel.

You then need to make sure your java3d jars are included in the run configuration class path, as well as the resources folder. To add these, select "*Bootstrap Entries*" when in the "*Classpath*" tab, see Figure 3-7. You can add jars by using the "*Add JARS*" button in the right panel, and to add the resources folder, click on "*Advanced*" in the right panel. After you are finished you should see the following:


Name: ConductingSphericalShellShielding

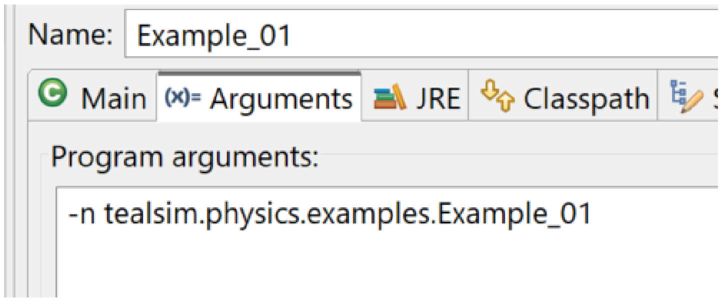
Classpath:

- Bootstrap Entries
  - JRE System Library [jdk-1.8]
  - resources - \MIT_TEALsim_2024\
  - j3dcore.jar - \MIT_TEALsim_2024\java\lib\j3d\1.5.2\
  - j3dutils.jar - \MIT_TEALsim_2024\java\lib\j3d\1.5.2\
  - vecmath.jar - \MIT_TEALsim_2024\java\lib\j3d\1.5.2\
- User Entries
  - MIT_TEALsim_2024 (default classpath)

Figure 3-7: The *Class Path* Panel in the *Run* dialog box

Make sure you are using 1.8 by clicking on the *JRE* tab in Figure 3-5. Now click on *Run* at the bottom of the dialog box in Figure 3-5. The *ConductingSphericalShellShielding* simulation will come up. The image on the title page of this document shows what you will see.

If we want to run one of the examples discussed below in Section 5, we use the following path in the *Arguments* section:



Name: Example_01

Program arguments:

```
-n tealsim.physics.examples.Example_01
```

Figure 3-8: The *Arguments* panel for running Example_01

4 ***TEALsim*** Overview

The TEAL simulation system, *TEALsim*, is designed as a framework for authoring, presenting, and controlling simulations in a variety of domains, beginning with physics. Architecturally, *TEALsim* follows a "*Model-View-Control*" design pattern, with several major modules representing the three components: the simulation engines make up the "model"; the renderer and viewer make up the "view"; and the user interface makes up the "control." These three components are combined into a simulation. Each component is largely defined by a set of interfaces that suggest the required functionality of that piece, so that the actual implementation details can be customized as necessary (for example, leaving the specific rendering implementation up to the developer). All major components of the system are *JavaBeans*. Most classes implement well defined interfaces, the most basic being *TElement*.

What follows is a brief description of the basic components.

4.1 The Simulation (TSimulation, SimEM)

The *TSimulation* interface (*java.src.teal.sim.simulation.TSimulation*) defines the requirements for a complete interactive simulation, collecting together all of the components that make up the entire user experience. Typically this includes: a simulation engine (*java.src.teal.sim.engine.TSimEngine*), a Viewer (*java.src.teal.render.viewer.Viewer*), the UI elements, and all of the objects being simulated or otherwise displayed in the Viewer. *SimEM* (*java.src.teal.physics.em.SimEM)* is an example of a class that implements *TSimulation*. It incorporates the *EMEngine* (*java.src.teal.physics.em.EMEngine*) and a Java 3D viewer and provides the basis for all electromagnetic simulations.

Any simulation object added to a simulation must implement the *TElement* (*java.src.teal.core.TElement*) interface, which provides a standard set of functionalities for all objects in the world. That is, *any* object which is specified, included or defined within the simulation must implement the *TElement* interface. This includes physical objects (e.g. a point charge), graphical elements, control objects and simulation viewers. In particular, the functionalities inherent in the *TElement* interface include support for *Routes* and *PropertyChangeEvents*. This allows any simulation elements in the world to exchange information with any other element in the world, and with User Interface (UI) components. For example, a UI slider can be wired to a property of a simulation object (for example, the mass of an electric point charge) to directly manipulate that quantity. Or, the simulation itself can be wired to a property of a simulation object in order to monitor that property of the object and/or take some action depending on its value.

A completely realized *TSimulation* class defines the properties of the simulation as a whole, such as the initial spatial configuration of simulation objects, initial conditions of any variables, and any special "wiring" between objects and/or interface components. Essentially, it contains the entire logic specific to a particular simulation.

A *TSimulation* object is then presented to the user using a *SimPlayer*, as discussed below.

4.2 SimPlayer (*TFramework*) and SimPlayerApp

A *SimPlayer* (*java.src.teal.app.SimPlayer*) "plays" a *TSimulation* instance—that is it takes as input a *TSimulation* object and presents that *TSimulation* to the viewer. It implements the *TFramework* (*java.src.teal.framework.TFramework*) interface, which is the glue that holds all of the components in the *TSimulation* together. *SimPlayer* is the application inside which all of the components of the *TSimulation* are running. The *TFramework* interface contains methods that allow loaded simulations to customize the *SimPlayer* application window, including modification of the UI and menu bars.

SimPlayerApp (*java.src.teal.app.SimPlayerApp*) contains the main routine of the *TEALsim* package. When we run *SimPlayerApp* as specified above in Section 3.4 above, it instantiates a *SimPlayer* and uses the argument

-n tealsim.physics.em.ConductingSphericalShellShielding

to create an instance of the class *ConductingSphericalShellShielding* and tells the *SimPlayer* to play *ConductingSphericalShellShielding*, which is a simulation object.

4.3 SimEngine – the Simulation Engine

The physics in a *TEALsim* simulation is contained in a *SimEngine* object (*java.src.teal.sim.engine.SimEngine)*, which represents the simulation engine itself. The simulation engine is responsible for all of the computation involved in the system being simulated. This includes dynamically processing and updating *TSimElement* simulation objects (*java.src.teal.sim.TSimElement*) according to the rules of the simulation engine (including adding and removing them from the "world" when necessary), and performing numerical integration of simulation variables. The exact type of processing and integration will depend on the specific type of simulation being implemented; for example, the "electromagnetism" extension of *SimEngine*, *EMEngine* (*java.src.teal.physics.em.EMEngine*), computes the total electromagnetic fields created by a set of field-generating simulation objects, each of which is an *EMObject* (*java.src.physics.em.EMObject*), as well as the resulting dynamics of the set of electromagnetic objects (velocity, position, rotation, etc.). It also handles related tasks, such as collision detection and resolution.

In general, the *SimEngine* runs a continuous loop that performs the following actions:

1) Computes values of dependent simulation variables for the current time step.
2) Updates simulation objects to reflect new values.
3) Informs the renderer of any visual changes to the simulation.

This loop in the *SimEngine* represents the main application thread for a *TEALsim* simulation.

4.4 Viewer and Viewer3D – the Rendering Engine

The *Viewer* (*java.src.teal.render.viewer.Viewer*) is the window into the simulation space, representing the rendering engine and its output. It is responsible for rendering the visual elements of a simulation to the screen in real-time 3D and managing user interaction with the rendered image. As such, it is tightly coupled to the *SimEngine*. The *SimEngine* must inform the *Viewer* of (visual) changes to simulation elements, and each visual simulation element must have an associated visual representation that can be drawn by the *Viewer*. Conversely, the *Viewer* must report back to the simulation when a user manipulates the visual representation of a simulation object (for example, by clicking and dragging on an object in the *Viewer*).

While the actual implementation of the *Viewer* can vary, the interface *TViewer* (*java.src.teal.render.viewer.TViewer*) defines a set of functionality that any *Viewer* implementation should support. This includes general rendering properties and tasks, such as camera controls, visual effects, maintaining lists of rendered objects, and handling mouse-based "picking" and manipulation of objects in the *Viewer*. In addition, it should also handle the explicit rendering of the scene.

The current default *Viewer* implementation is based on *Java3d 1.5.2*, which is a scene-graph based renderer layered on top of *OpenGL* or *DirectX*. In this case, the *Viewer* sets properties on the scene graph, which is then rendered implicitly through *Java3d*'s rendering thread. A more direct (or "immediate") rendering implementation (such as rendering directly through *OpenGL*) should obviously include explicit rendering instructions for all visual elements.

4.5 The User Interface

The user interface (*java.src.teal.ui*) is the means by which a user interacts with the application (in addition to user interaction through the *Viewer*), and through which the user receives feedback about a simulation's properties. It is responsible for producing the types of controls and read-outs necessary to manipulate a simulation. These can include buttons, sliders, checkboxes, combo-boxes, graphs, text fields, and numerical displays.

4.6 Units

A unit length in *TEALsim* is one meter, a unit mass is one kilogram, a unit time is one second, and a unit force is one Newton.

A unit charge in *TEALsim* is $\sqrt{\varepsilon_0}$ Coulombs, or 2.975 micro Coulombs. ($\sqrt{\varepsilon_0} = \sqrt{8.85\times10^{-12}} = 2.975\times10^{-6}$). This unit charge arises because we set ε_0 to 1 in *teal.config.Teal*. Since the force between two point charges is given by $qQ/(4\pi\varepsilon_0 r^2)$, for this force to be in Newtons if *r* is in meters, we must take the unit charge to have this value.

A current unit in *TEALsim* is $\frac{1}{\sqrt{\mu_0}}$ amps, or 892.0620581 amps. This unit current arises because we set μ_0 to 1 in *teal.config.Teal*. Since the force per unit length between two current elements separated by a distance *d* is given by $\mu_0 iI/(2\pi d)$, for this force/length to be in Newtons/meter if *d* is in meters, we must take the unit current to have this value.

5 Examples in tealsim.examples

We present several examples below to illustrate various features of *TEALsim* and how to use them to construct simulations. These examples are contained in the package *java.src.tealsim.physics.examples*. As we present the examples, we will point out how they implement the overall scheme described above in Section 4. For more detail about specific features, Section 6 looks at individual features (for example, the "grass seeds" representation).

5.1 Example_01: 3D objects (native and imported)

We begin with an example which demonstrates how much of the usual infrastructure for setting up a Java3D scene is hidden from the user at this level. The example *java.src.tealsim.physics.examples.Example_01* extends *SimEM*, which has already created the lights, cameras, etc. We create two native Java3D objects, a flat red disk and a green sphere, and add them to the scene. We also import two *.3DS* files, an orange hemisphere and a tapered cone with a texture applied.

A screen capture from this example is shown in Figure 5-1. The coordinate system in *Java3d* is as follows: the *x* axis is horizontal and positive to the right; the *y*-axis is vertical and positive upward; the *z*-axis completes the right-handed system and is out of the page. The lights and the camera in this example are all set in the viewer for this world, *teal.render.j3d.ViewerJ3D*, which extends

teal.render.viewer.AbstractViewer3D

The camera is placed at *(0.,0.,5.)* (see Figure 5-2). There are four lights in the scene, one omni light and three directional lights. The directions of the directional lights are: (1) *(4., -7., -12.)*; (2) *(-6., -2., -1.)*; (3) *(-6., 2., 1.)*.

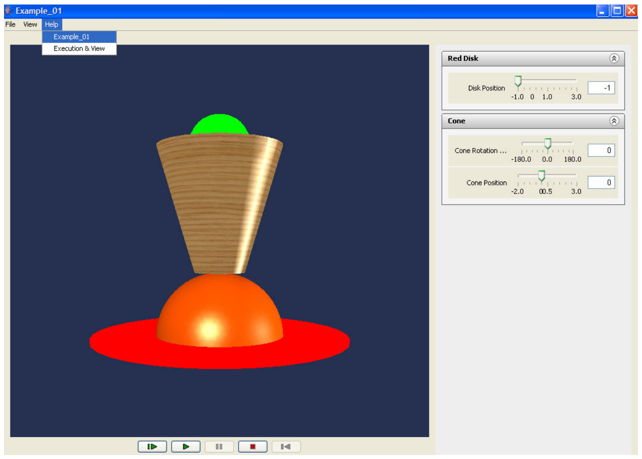


Figure 5-1: Screen capture from Example_01

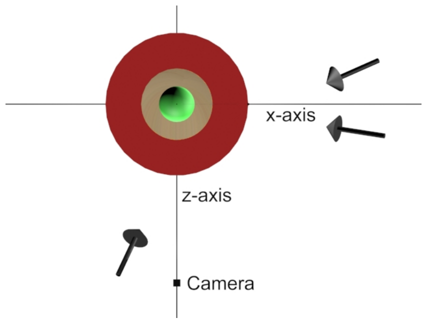


Figure 5-2: Top view of scene: axes and the directions of the 3 directional lights.

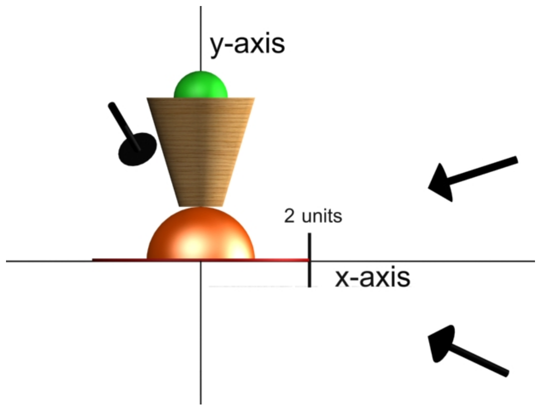


Figure 5-3: Front view of scene in Example_01.

We add to the upper tool bar in the simulation (see upper left of Figure 5-1) links to two help files, one which explains what the simulation does (*resources.help.example_01.html*) , and one which explains the view and execution controls (r*esources.help.executionView.html*) for the 3D window and simulation engine, respectively. These files are opened using the browser included in the TEAL project, *teal.browser.Browser*.

In terms of the overview provided in Section 4 above, all of our examples extend *teal.physics.em.SimEM* which in turn extends *teal.sim.simulation.Simulation3D*. *Simulations3D* sets the viewer to *teal.render.j3d.ViewerJ3D*. *ViewerJ3D* extends *teal.render.viewer.AbstractViewer3D* which in turn extends *teal.render.viewer.Viewer*.

The sliders are built out of components in *teal.ui.control.PropertyDouble*, which extends *teal.ui.control.PropertySlider*. The vertical (*y*) position of the red disk is controlled by a one slider. The vertical position of the cone is also controlled by a slider, as is its orientation in the *xy* plane (the plane of the screen as seen from the initial camera position).

5.2 Example_02: Simple simulation using *theEngine*

In *Example_01* we only illustrated properties of the Java3D viewer, with no simulation dynamics (that is, nothing happens as world time progresses). In *Example_02* which again extends *SimEM* (as does all of our examples), we introduce a simple dynamical situation to illustrate how *theEngine* and engine controls work. The simulation *SimEM* sets the simulation engine to *teal.physics.em.EMEngine*, which extends *teal.sim.engine.SimEngine*. In *Example_02*, we set the dynamical time step to 0.02 seconds using the method *theEngine.setDeltaTime*, and add to the scene a *teal.physics.em.EMObject* object, the *teal.physics.em.PointCharge* object. This object knows how to interact with the world, as we explain in this example and others to come.

In Example_02, the *PointCharge* object *floatingCharge* has mass, which means it will interact with the general gravitation field of the world, which is set by default to (0.,-9.8,0.) in *EMEngine*. To show how we change the default gravity in the world, we add a statement which reduces the default value of *g* to 4.9 meters per second squared in the negative *y*-direction in this example. We also create a "wall" 1.25 meters beneath the initial position of the charge, which appears in the scene as a transparent rectangle. If we step the dynamics using the step control (see Figure 5-4), it takes about 35 steps at the 0.02 second step size in this world for the point charge to fall the 1.25 meters to the position of the wall, which is correct for a gravitational acceleration of 4.9 meters per second squared.

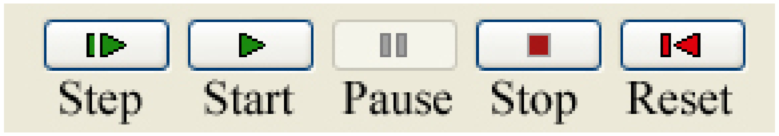


Figure 5-4: The simulation control icons

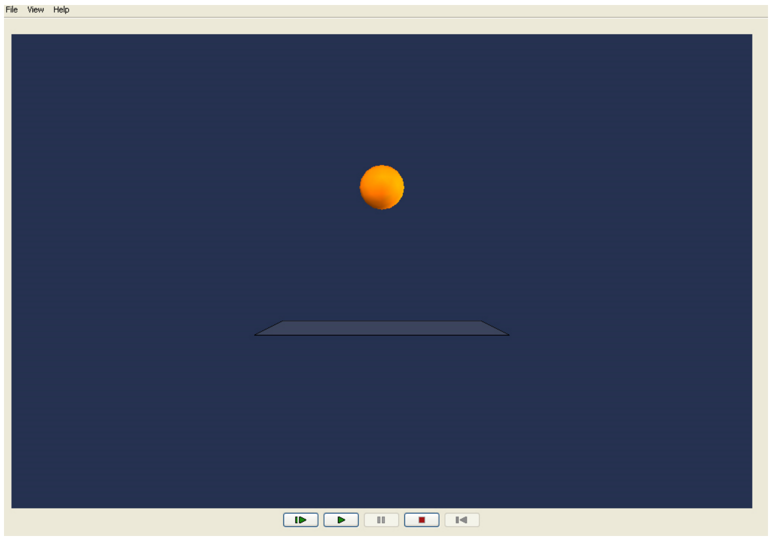


Figure 5-5: A charge falling toward the floor.

We added a collision controller to the *floatingCharge* so that it will be recognized as a colliding object when it touches the "Wall". The collision controller *teal.sim.collision.SphereCollisionController* is a sphere whose radius is the radius of the *floatingCharge*, placed at the center of the *floatingCharge*, This has the effect that when the *floatingCharge* center position comes within its radius of the "Wall" it "collides" with the "Wall" and the *floatingCharge* bounces. We set the elasticity of the wall to 1.0, which means the collision is perfectly elastic.

The dynamics of this falling charge is computed as follows. First, the six equations of motion for the position **x** and velocity **v** of the point charge that we are trying to solve are

$$\frac{d}{dt}\mathbf{x} = \mathbf{v} \qquad \frac{d}{dt}\mathbf{v} = \mathbf{F}/m \tag{5.1}$$

(we ignore the rotational dynamics of the object, although this can be included). Thus, the dependent variables that we are integrating are the position and velocity of the floating charge. Our point charge object extends *EMObject*, which extends *teal.physics.physical.PhysicalObject*, and therefore has a method *getDependentValues()*. This method simply returns the current values of the six dependent values of **x** and **v**. It inherits from *PhysicalObject* a method *getDependentDerivatives()* which calculates the derivatives of these dependent values using equation (5.1). This method accomplishes this using a metho*d getExternalForce* in *PhysicalObject* which adds up the total force on the point charge, including gravity, an over all frictional force proportional to the velocity (-*friction****v**), any electromagnetic forces *q*(**E**+**v**x**B**) acting on the point charge due to other electromagnetic objects in the world (zero in this example), and a Pauli force which is always repulsive at close distances (we discuss this further in *Example_04* below).

Once we have the current values of the dependent variables and their derivatives, we integrate the equations of motion using a fourth order Runge-Kutta scheme *teal.math.RungeKutta4*, using routines in *teal.sim.engine.SimEngine*. In particular, the method *nextStep* in *SimEngine* advances the world one time step using the integration routine, among other methods.

5.3 Example_03: Simple simulation with a graph and a damping slider

Example_03 is the same as *Example_02* except that we add a graph of the vertical position of the charge and a slider that controls the amount of damping in the world. To accomplish this we must import user interface classes for the panel display, for the damping control slider, and for plotting.

Note that we can change the friction value dynamically between time steps in the world. That is, in the integration for the position of the point charge, before each time step *nextStep* checks to see if the friction in the world has changed and computes the next dynamical step using the current value of the friction.

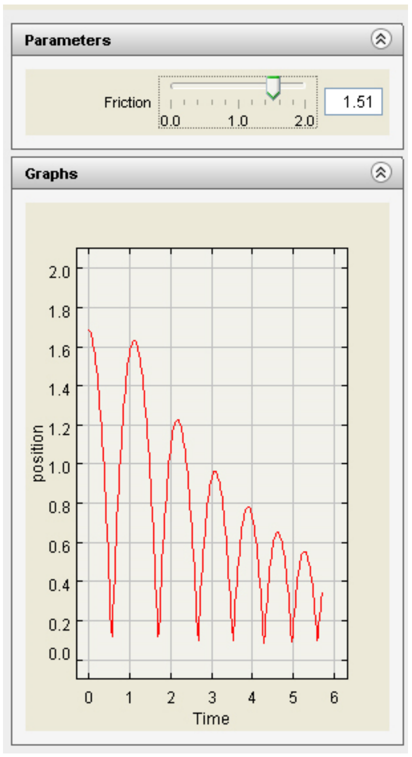


Figure 5-6: The graph and slider control for the falling charge

5.4 Example_04: Two Interacting Electromagnetic Objects (Point Charges)

In Example_04 we create two point charges which interact via their electric fields. The two objects are our point charge from *Example_02*, free to move along the vertical axis, and a new point charge which is fixed in space 0.8 meters below the wall in *Example_02*, again on the vertical axis. We set the friction in the world to a high value, so that the floating charge will quickly settle down quickly to its equilibrium position.

The charge on the fixed-in-space charge can be varied between -10 and 50 charge units, where a charge unit is 2.975 micro-Coulombs (see Section 3.6.2). The charge on the floating charge is constant at 1 charge unit, its radius is 0.2 meters and its mass is 0.035 kg. The force of gravity on the floating charge, m**g**, is 0.343 Newtons. For a fixed charge of about 4.4 charge units the upward electrostatic repulsion when the floating charge is resting on the wall (center 1 meter above the fixed-in-space charge) will just about balance the downward force of gravity, and the floating charge will levitate above the wall for values of the charge on the fixed-in-space charge above this value.

Note that if we change *fixedCharge.setMoveable(false)* to *fixedCharge.setMoveable(true)* in *Example_04*, the position and velocity of our fixed charge will be added to the dependent variable list of our world. When we start the simulation, we will be integrating 12 dependent variables, 6 for each charge, and the "fixed" charge will begin to fall under gravity etc.

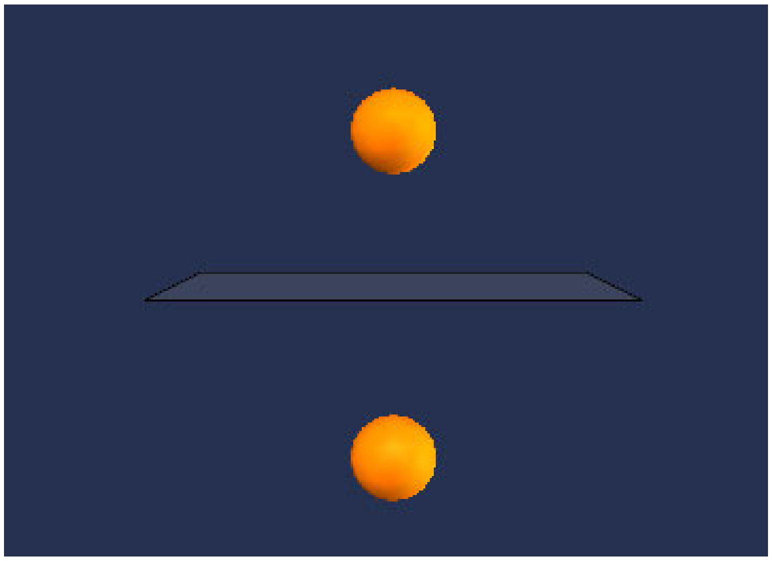


Figure 5-7: Two Electromagnetic Objects Interacting

5.5 Example_05: Vector Field Grid

In *Example_05*, we add a vector field grid to *Example_04* to demonstrate one of three methods we use to visualizing electromagnetic fields.

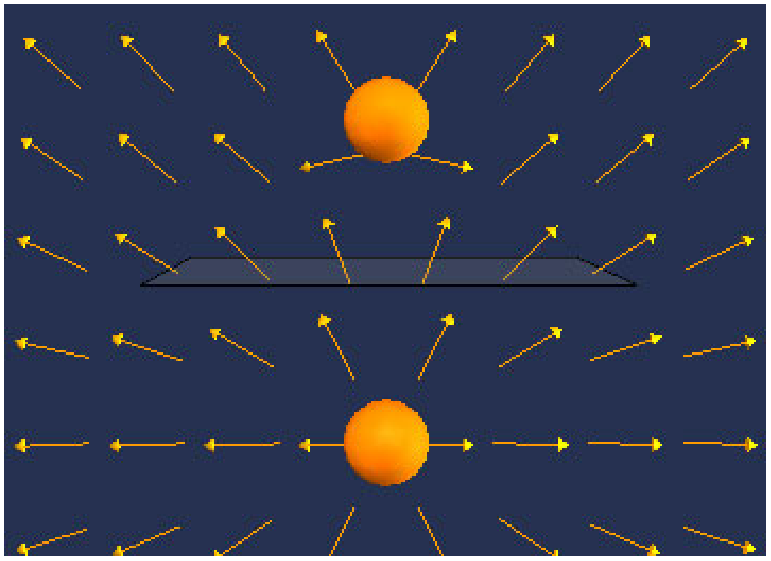


Figure 5-8: An example of a vector field grid.

5.6 Example_06: Integrate and Draw (IDRAW)

In *Example_06*, we add a line integral convolution example to *Example_04* to demonstrate one of three methods we use to visualize electromagnetic fields.

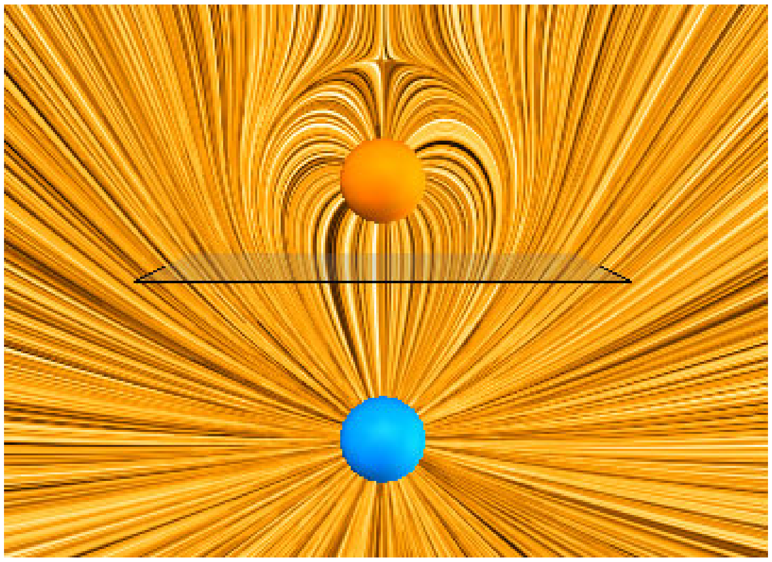


Figure 5-9: An example of a line integral convolution.

5.7 Example_07: Traditional Field Lines

In *Example_07* we add field lines to *Example_04* to demonstrate one of three methods we use to visualize electromagnetic fields. In this case we add two kinds of field lines.

The first kind (the two red field lines to the right of the vertical center line in Figure 5-10) are field lines that always go through the same spatial point, in this case (1,0,0) and (1,2,0). The point at which the field line starts for both field lines is shown by a lighter segment along the field line. Note that for the red field lines we construct the field in both directions from the starting point; that is both along the field and opposite the field.

The second kind of field line (the three blue field lines to the left of the vertical center line in the figure) starts out at the radius of the sphere at the same angles (in this case -45, -90, and -135 degrees) measured from the object direction of the object to which it is attached, the floating charge. As the floating charge moves up and down, the position at which the blue field lines start is always at the same angle from the axis of the charge direction, moving with the charge.

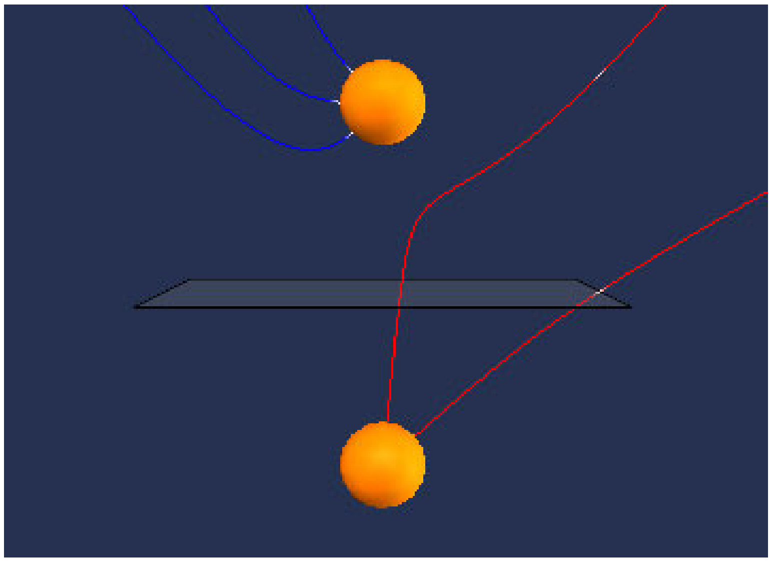


Figure 5-10: Examples of two fixed and three relative field lines.

5.8 Example_08: Flux Preserving Field Lines Electric

In *Example_08*, we also add field lines to *Example_04* to demonstrate one of three methods we use to visualize electromagnetic field, but a very different kind of field line. In this case we add field lines which preserve electric flux.

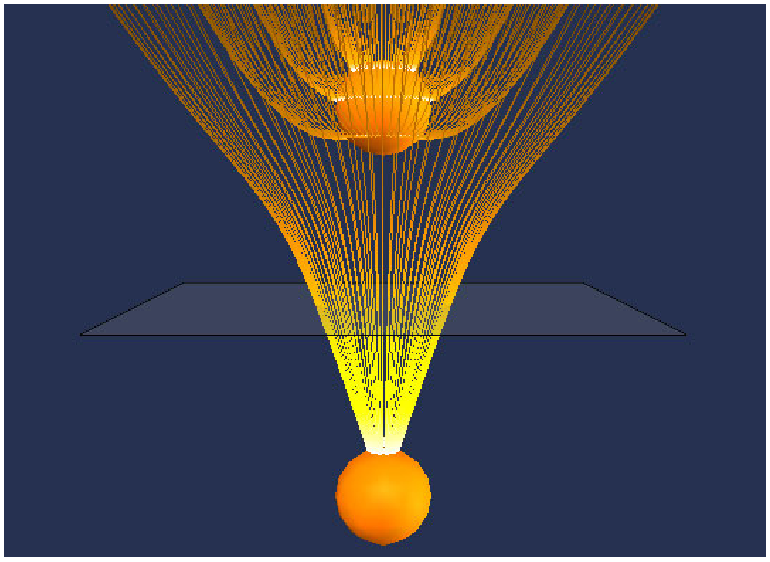
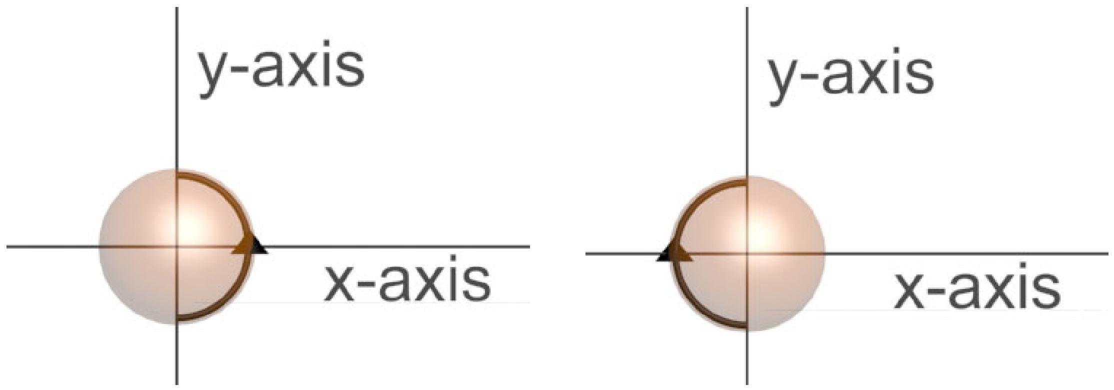
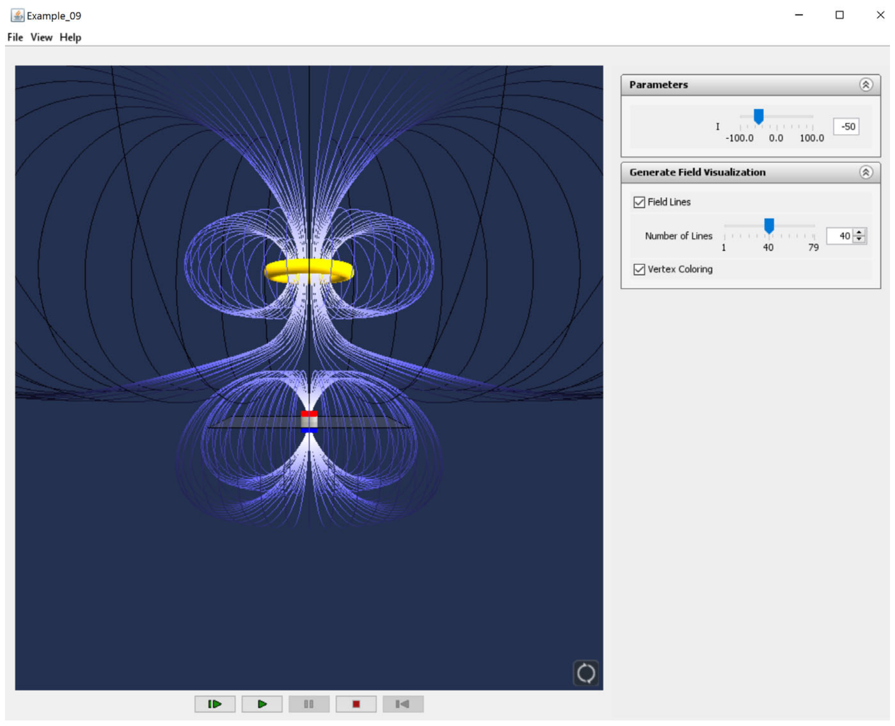


Figure 5-11: Flux Preserving Electric Field Lines



5.9 Example_09: Flux Preserving Field Lines Magnetic

In *Example_09*, we add field lines to a new magnetostatic simulation. These field lines preserve magnetic flux. The simulation consists of a floating ring levitating against gravity above a permanent magnetic dipole (see Figure 5-12). The current in the ring can be varied using a slider, with positive current representing current flow counterclockwise as seen from above.



Parameters
I -100.0 0.0 100.0 -50

Generate Field Visualization
☑ Field Lines
Number of Lines 1 40 79 40
☑ Vertex Coloring

Figure 5-12: Flux Preserving Magnetic Field Lines

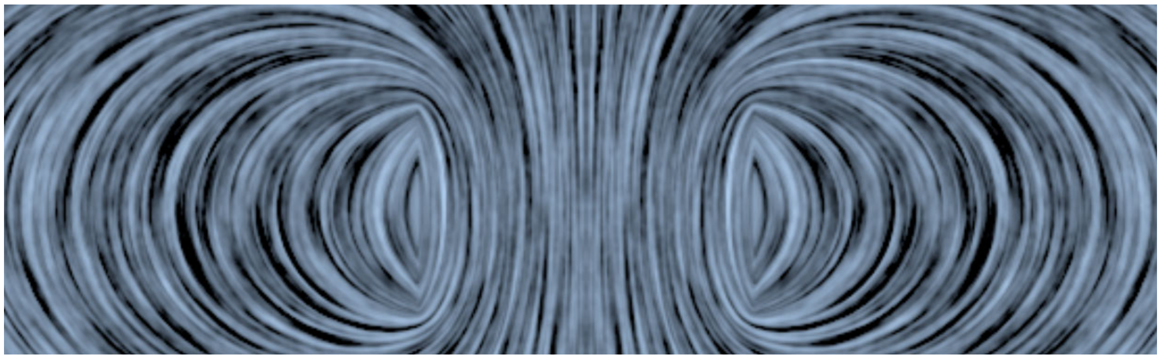
5.10 Example_10:

5.11 Example_11:

5.12 Example_12:

6 EMObjects in teal.physics.em

6.1 BField.java
6.2 Circuit.java
6.3 CircuitParticleSystem.java
6.4 Coil.java
6.5 CoilBeanInfo.java
6.6 ConductingSphericalShell.java
6.7 ConstantField.java
6.8 CurrentSlab.java
6.9 CylindricalBarMagnet.java


I think I created this out of the class RingOfCurrent although I am not sure.

6.10 CylindricalField.java
6.11 CylindricalMagnet.java
6.12 Dipole.java
6.13 DipoleBeanInfo.java
6.14 EField.java
6.15 ElectricDipole.java
6.16 ElectricDipoleBeanInfo.java
6.17 EMEngine.java
6.18 EMObject.java
6.19 EMRadiator.java
6.20 EMRadiatorBeanInfo.java
6.21 FiniteWire.java
6.22 GeneratesB.java
6.23 GeneratesE.java
6.24 GeneratesP.java
6.25 GenericBField.java
6.26 GenericEField.java
6.27 GenericPField.java

6.28 HasCharge.java
6.29 HasCurrent.java
6.30 HasDipoleMoment.java
6.31 HasInductance.java
6.32 HasResistance.java
6.33 HollowCurrentShell.java
6.34 InfiniteConductingPlane.java
6.35 InfiniteLineCharge.java
6.36 InfiniteWire.java
6.37 InfiniteWireBeanInfo.java
6.38 LineMagneticDipole.java
6.39 MagneticDipole.java

6.40 MagneticDipoleBeanInfo.java
6.41 PField.java
6.42 PointCharge.java
6.43 PointChargeBeanInfo.java
6.44 PointChargeConductingSphere.java
6.45 RingOfCurrent.java
6.46 RingOfCurrentBeanInfo.java
6.47 SimEM.java

7 Applications in tealsim.physics.em

7.1 AmperesLaw.java
7.2 AmperesLawCurrentDensity.java
7.3 boxInduction.java

Add many field lines to each charge in this application.

7.4 Capacitor.java

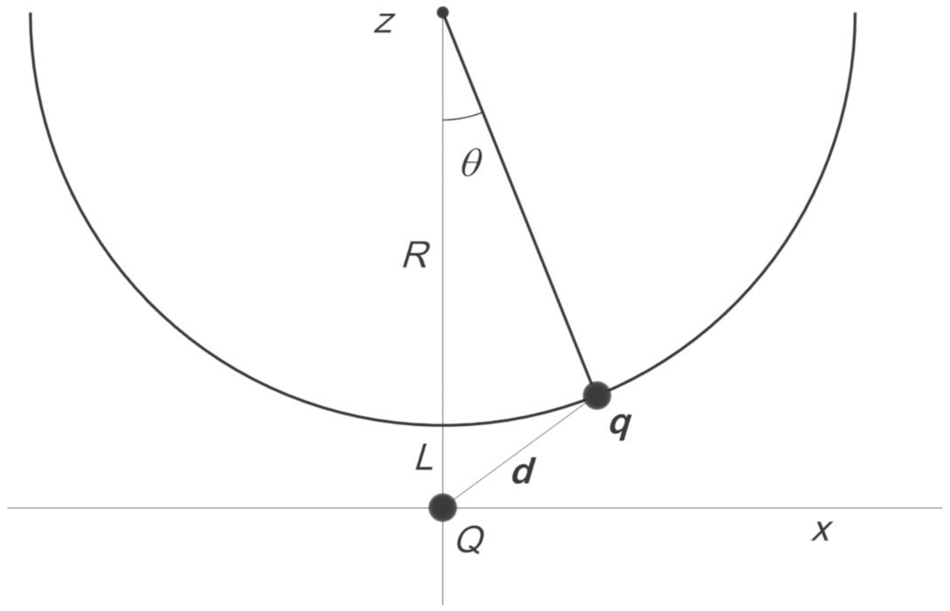
Add many field lines to each charge in this application.

7.5 ChargeByInduction.java

Add many field lines to each charge in this application.

7.6 ChargedMetalSlab.java
7.7 ChargeInMagneticFieldGame.java
7.8 ChargesInBox.java
7.9 CompassAndMagnet.java
7.10 ConductingSphericalShellShielding.java

7.11 CoulombsLaw.java
7.12 ElectrostaticForce.java
7.13 ElectrostaticPendulum.java



$$\frac{1}{2}mv^2 + mgz + \frac{qQ}{4\pi\varepsilon_o d} = \text{constant} \tag{1.1}$$

The charge q is located at (x,z). Because it is located on a circle of radius R whose center is at $(0,R+L)$, we must have

$$x^2 + \left(R+L-z\right)^2 = R^2 \tag{1.2}$$

$$d^2 = x^2 + z^2 = R^2 - \left(R+L-z\right)^2 + z^2 \tag{1.3}$$

$$\frac{1}{2}mv^2 + mgz + \frac{qQ}{4\pi\varepsilon_o\sqrt{R^2 - \left(R+L-z\right)^2 + z^2}} = \text{constant} \tag{1.4}$$

We evaluate the constant at *t = 0*, when the charge is at $(R, R+L)$ with zero velocity:

$$0 + mg(R+L) + \frac{qQ}{4\pi\varepsilon_o\sqrt{R^2 + \left(R+L\right)^2}} = \text{constant} \tag{1.5}$$

So

$$\frac{1}{2}mv^2+mg\left(z-R-L\right)+\left(\frac{qQ}{4\pi\varepsilon_o}\right)\left(\frac{1}{\sqrt{R^2-\left(R+L-z\right)^2+z^2}}-\frac{1}{\sqrt{R^2+\left(R+L\right)^2}}\right)=0 \quad (1.6)$$

For the velocity to be zero at $z=L$, we must have therefore

$$-mgR+\left(\frac{qQ}{4\pi\varepsilon_o}\right)\left(\frac{1}{L}-\frac{1}{\sqrt{R^2+\left(R+L\right)^2}}\right)=0 \quad (1.7)$$

$$+\frac{q}{Q}=\frac{4\pi\varepsilon_o mgR}{Q^2\left(\frac{1}{L}-\frac{1}{\sqrt{R^2+\left(R+L\right)^2}}\right)}=\frac{mgR}{\left(Q^2/4\pi\varepsilon_o L\right)\left(1-\frac{1}{\sqrt{\left(R/L\right)^2+\left(R/L+1\right)^2}}\right)} \quad (1.8)$$

So the above charge will bring it to rest at θ equal to zero. What if we want to bring it at rest at an angle not equal to zero. In this case,

$$z=R+L-R\cos\theta=R(1-\cos\theta)+L \quad (1.9)$$

$$\frac{1}{2}mv^2-mgR\cos\theta+\left(\frac{qQ}{4\pi\varepsilon_o}\right)\left(\frac{1}{\sqrt{R^2-\left(R\cos\theta\right)^2+\left(R(1-\cos\theta)+L\right)^2}}-\frac{1}{\sqrt{R^2+\left(R+L\right)^2}}\right)=0$$

(1.10)

$$mgR\cos\theta=\left(\frac{qQ}{4\pi\varepsilon_o}\right)\left(\frac{1}{\sqrt{R^2-\left(R\cos\theta\right)^2+\left(R(1-\cos\theta)+L\right)^2}}-\frac{1}{\sqrt{R^2+\left(R+L\right)^2}}\right) \quad (1.11)$$

$$\frac{q}{Q}=\frac{mgR\cos\theta}{\frac{Q^2}{4\pi\varepsilon_o L}\left(\frac{1}{\sqrt{\left(R/L\right)^2-\left(R\cos\theta/L\right)^2+\left(R(1-\cos\theta)/L+1\right)^2}}-\frac{1}{\sqrt{\left(R/L\right)^2+\left(1+R/L\right)^2}}\right)}$$

(1.12)

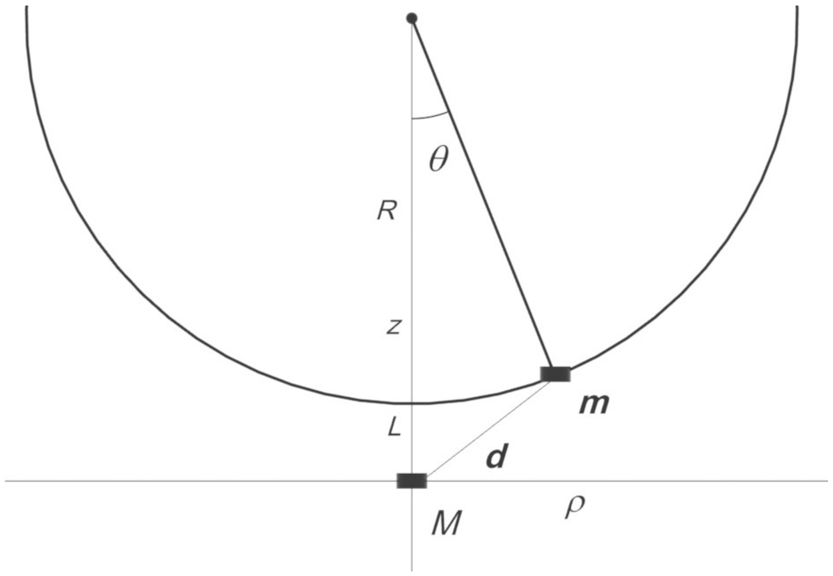
7.14 ElectrostaticPendulumSpherical.java

7.15 EMRadiatorApp.java

7.16 EMVideogame.java
7.17 EMVideogame2.java
7.18 EMZoo.java
7.19 EMZooTestQuestions.java
7.20 ExBDrift.java
7.21 ExploringElectricField.java
7.22 ExploringPotential.java
7.23 FallingCoil.java
7.24 FallingMagnet.java
7.25 FaradayIcePail.java
7.26 FaradayIcePailShield.java
7.27 FaradaysLaw.java
7.28 FaradaysLawRotation.java
7.29 filledCylinderShell.java
7.30 FloatingCoil.java
7.31 GaussLawFlux.java
7.32 GaussLawSymmetry.java
7.33 ImageChargeInfinitePlane.java
7.34 InverterMagnet.java

7.35 InverterMagnet1D
7.36 Landscape.java
7.37 MagDipoleFluxThroughDisk.java
7.38 MagneticPendulumSpherical.java
7.39 MagneticLevitationByRotation.java

7.40 MagnetostaticPendulum.java



$$\frac{1}{2}mv^2 + mgz + \mathbf{m}_d \cdot \mathbf{B}_M = \text{constant} \tag{1.13}$$

$$\mathbf{B}_M = -\hat{\mathbf{z}}\frac{\partial}{\partial z}\frac{\mu_o}{4\pi}\frac{Mz}{\left(z^2+\rho^2\right)^{3/2}} - \hat{\boldsymbol{\rho}}\frac{\partial}{\partial \rho}\frac{\mu_o}{4\pi}\frac{Mz}{\left(z^2+\rho^2\right)^{3/2}} \tag{1.14}$$

$$\mathbf{B}_M = \hat{\mathbf{z}}\frac{\mu_o}{4\pi}\frac{M\left(2z^2-\rho^2\right)}{\left(z^2+\rho^2\right)^{5/2}} + \hat{\boldsymbol{\rho}}\frac{\mu_o}{4\pi}\frac{2Mz\rho}{\left(z^2+\rho^2\right)^{5/2}} \tag{1.15}$$

$$\mathbf{m}_d \cdot \mathbf{B}_M = \frac{\mu_o}{4\pi}\frac{m_d M\left(2z^2-\rho^2\right)}{\left(z^2+\rho^2\right)^{5/2}} \tag{1.16}$$

The magnetic dipole m_d is located at (x,z). Because it is located on a circle of radius R whose center is at $(0,R+L)$, we must have

$$\rho^2 + \left(R+L-z\right)^2 = R^2 \tag{1.17}$$

$$d^2 = \rho^2 + z^2 = R^2 - \left(R+L-z\right)^2 + z^2 \tag{1.18}$$

$$\frac{1}{2}mv^2 + mgz + \frac{\mu_o}{4\pi}\frac{m_d M\left(2z^2 - R^2 + \left(R+L-z\right)^2\right)}{\left(R^2 - \left(R+L-z\right)^2 + z^2\right)^{5/2}} = \text{constant} \tag{1.19}$$

We evaluate the constant at *t = 0*, when the charge is at $(R,R+L)$ with zero velocity:

$$0 + mg\left(R+L\right) + \frac{\mu_o}{4\pi}\frac{m_d M\left(2\left(R+L\right)^2 - R^2\right)}{\left(R^2 + \left(R+L\right)^2\right)^{5/2}} = \text{constant} \tag{1.20}$$

So, we have

$$\frac{1}{2}mv^2 + mg\left(z-R-L\right) + \frac{\mu_o}{4\pi}\frac{m_d M\left(2z^2 - R^2 + \left(R+L-z\right)^2\right)}{\left(R^2 - \left(R+L-z\right)^2 + z^2\right)^{5/2}} = \frac{\mu_o}{4\pi}\frac{m_d M\left(R^2+4RL+2L^2\right)}{\left(2R^2+2RL+L^2\right)^{5/2}} \tag{1.21}$$

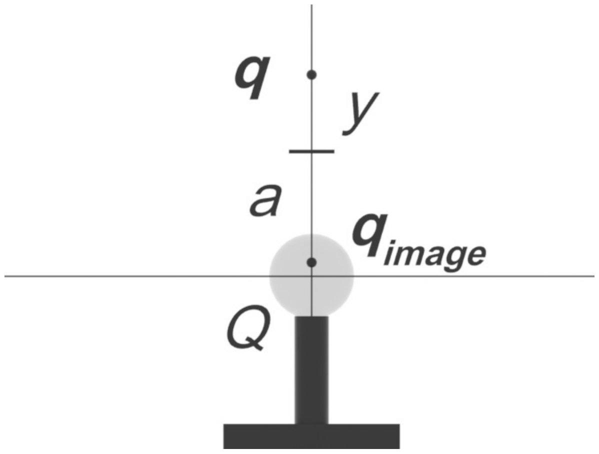
For the velocity to be zero at $z = L$, we must have therefore

$$-mgR + \frac{\mu_o}{4\pi}\frac{2m_d M}{L^3} - \frac{\mu_o}{4\pi}\frac{m_d M\left(R^2+4RL+2L^2\right)}{\left(2R^2+2RL+L^2\right)^{5/2}} = 0 \tag{1.22}$$

$$\frac{m_d}{M} = \frac{mgR}{\frac{\mu_o}{4\pi}\frac{M^2}{L^3}\left[\frac{\left((R/L)^2+4R/L+2\right)}{\left(2(R/L)^2+2R/L+1\right)^{5/2}}-2\right]} \tag{1.23}$$

So, the above relationship between *M* and m_d will bring it to rest at θ equal to zero.

7.41 MappingFields.java
7.42 MoveableConductingSphericalShellShielding.java
7.43 PCharges.java
7.44 Pentagon.java
7.45 Pentagon2.java
7.46 RadiationCharge.java
7.47 TorqueOnDipoleB.java
7.48 TorqueOnDipoleE.java
7.49 TwoRings.java
7.50 VandeGraff.java



The file generating this image is in

F:\My Documents\max\MaxRoutinesScripts\electrostatics\VandeGraff

A point charge *q* is at rest on a platform that a distance a above a Van de Graff with radius *R* and charge *Q*. The center of the Van de Graff sphere is at the origin, and initally the Van de Graff is uncharged.

The potential outside the sphere is the potential due to the point charge plus the potential due to an image charge with charge $-qb/r'$ located inside the shell at a distance b^2/r'. Section 2.3 of *TEAL_Physics_Math*.

```
   double vdgRadius = 1.;
```

```
imageCharge.setPosition(new Vector3d(0., vd

imageCharge.setCharge(-pcCharge * vdgRadius / pcPosition.y);gRadius * vdgRadius / pcPosition.y, 0.));
```

The free charge is 1, we run the code with the Van de Graff charge of 50, the maximum allowed. The free charge starts out a distance of 6 units above the center of the Van de Graff. The gravitational potential energy is relative to this position,

```
gpEnergy = m1 * (0.04) * (pos1.y - 6.) *1.;  line 120 of TwoBodyEnergyPlot
```

where 0.04 is the acceleration of gravity. See line 122 in main routine.

The electric energy is defined by

```
eEnergy = q1 * q2 * (1/(pos1.y)); // * a constant 8.897e8 *

eEnergy = eEnergy / 456.4;    line 112 of TwoBodyEnergyPlot
```

The kinetic energy is defined by

```
kEnergy = 0.5 * m1 * vel1.lengthSquared() * 1.;   line 119  of TBEP

kgpEnergy = kEnergy + gpEnergy+eEnergy;
kgpEnergy = kgpEnergy/34.2775;
```

```
time: 86.0  eEnergy: 0.011070008286533815 y position: 9.89638136014041 speed: 0.4567990033608627
  q1:  1.0 q2: 50.0 mass: 1.0
kEnergy: 0.10433266473573873 gpEnergy 0.1558552544056164 kgpEnergy: 0.007913585513175959
```

$$mgh + \frac{1}{2}mv^2 + \frac{1}{4\pi\varepsilon_o}\frac{q_1 q_2}{y} = const$$

7.51 WireAndMagnet.java

8 Building Specific Features into the *TEALsim* Simulations

In addition to the examples above, we here consider in detail the properties of various aspects of visualization features that can be added to a *TEALsim* simulation, starting with how to set the basic visual features of the world (e.g. background color, and so on).

8.1 Setting Background Color

The default background color for a viewer window is set in

8.2 *Grass Seeds (Iron Filings)* representations

In discussing this type of representation, you may want to look at the code for Example_06 above, to see how this is implemented in a working application. Typically when using one of these representations we create a *Field Convolution* object

```
/** The line integral convolution. */
   private FieldConvolution mDLIC;
```

We then set the properties of this object as follows:

```
Vector3d Position, Edge1, Edge2;
RectangularPlane rec;
rec = new RectangularPlane(Position, Edge1, Edge2);
mDLIC = new FieldConvolution();
mDLIC.setSize(new Dimension(1024, 1024));
mDLIC.setComputePlane(rec);
vis.setConvolutionModes(DLIC.DLIC_FLAG_E | DLIC.DLIC_FLAG_EP);
```

8.3 Viewer Navigation Modes

8.3.1 Fixed view, zoom:

mViewer.setNavigationMode(TViewer.ORBIT | TViewer.VP_ZOOM)

8.3.2 Rotatable view, zoom:

This is the default setting, you do not need to set the Navigation Mode of the viewer.

8.4 Removing Run Controls from Bottom of View Window

If you want to hide run controls at the bottom of the view window (for example the application is only showing geometry and has no “dynamics” to run), issue the following command to the EngineControl *mSEC*

```
mSEC.rebuildPanel(0);
```

8.5 Transparency and/or Visibility

The code snippet below creates a Rendered object and sets it transparency to 0.8f. This is very transparent (a transparency of 0.0f would be not be transparent at all).

```
Rendered ring = new Rendered();
TShapeNode node = (TShapeNode) new ShapeNode();
node.setGeometry(Pipe.makeGeometry(50, rad, thick, height));
node.setPickable(false);
node.setColor(new Color3f(Color.ORANGE));
node.setTransparency(0.8f);
ring.setNode3D(node);
addElement(ring);
```

If you want to make the object totally invisible you can either set the transparency to 1.0f or add the instruction

```
ring.setDrawn(false);
```

9 JNLP files in tealsim.java.jnlp

9.1 ampereslaw.jnlp
9.2 ampereslawcurrentdensity.jnlp
9.3 boxinduction.jnlp
9.4 capacitor.jnlp
9.5 celestial.jnlp
9.6 chargebyinduction.jnlp
9.7 chargedmetalslab.jnlp
9.8 chargeinmagneticfield.jnlp
9.9 chargesinbox.jnlp
9.10 circularmotion.jnlp
9.11 coulombslaw.jnlp
9.12 emradiator.jnlp
9.13 esforce.jnlp
9.14 exbdrift.jnlp
9.15 exploringelectricfield.jnlp
9.16 exploringpotential.jnlp
9.17 fallingcoil.jnlp
9.18 fallingmagnet.jnlp
9.19 faradayicepailshield.jnlp
9.20 faradayslaw.jnlp
9.21 faradayslaw02.jnlp
9.22 faradayslawrotation.jnlp
9.23 filledcylindricalshell.jnlp

9.24 floatingcoil.jnlp
9.25 ForceOnDipoleLV.jnlp
9.26 galieosinclinedplane.jnlp
9.27 gausslaw.jnlp
9.28 gausslawflux.jnlp
9.29 gausslawsymmetry.jnlp
9.30 inclinedplane.jnlp
9.31 landscape.jnlp
9.32 mappingfields.jnlp
9.33 newtonscradle.jnlp
9.34 pcharges.jnlp
9.35 PDBViewer.jnlp
9.36 pendulum.jnlp
9.37 pentagon.jnlp
9.38 pentagon2.jnlp
9.39 planets.jnlp
9.40 planewaveapp.jnlp
9.41 projectile.jnlp
9.42 radiationcharge.jnlp
9.43 rigidbar.jnlp
9.44 shielding.jnlp
9.45 simplayer.jnlp
9.46 simplependulum.jnlp
9.47 simplependulumimpulse.jnlp
9.48 teachspin.jnlp
9.49 teachspinLV.jnlp
9.50 tealsimapp.jnl
9.51 torqueondipb.jnlp
9.52 torqueondipe.jnlp
9.53 tworings.jnlp
9.54 vandegraff.jnlp
9.55 videogame.jnlp
9.56 weightedpendulum.jnlp
9.57 wireandmagnet.jnlp
9.58 zoo.jnlp

9.59 Vector Fields

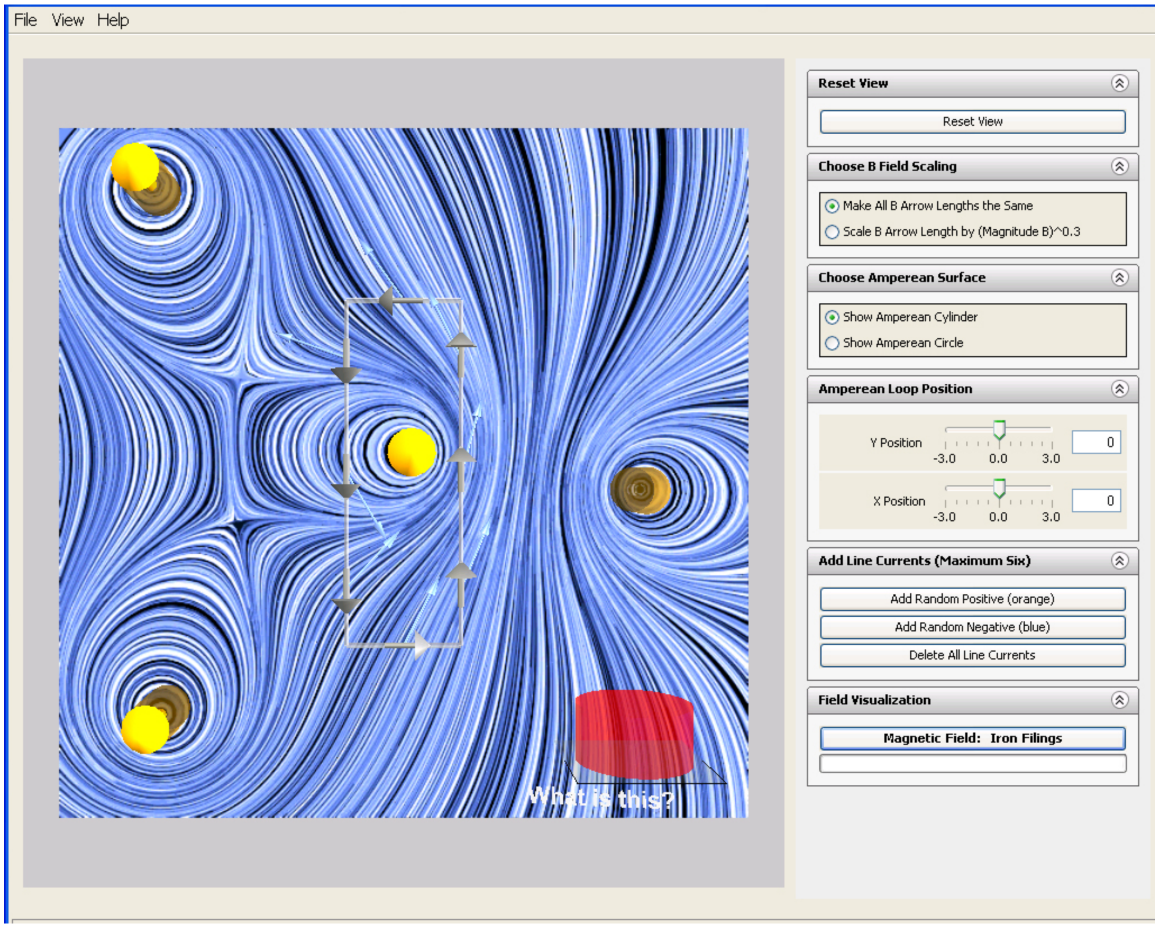
9.59.1 Field Mapping Application

9.60 Electrostatics

9.60.1 Shielding by a Conducting Spherical Shell

9.61 Magnetostatics

9.61.1 Ampere's Law



[✓ Make All B Arrow Lengths the Same] Scale B Arrow Length by (Magnitude B)^0.3

[✓ Show Amperean Cylinder] Show Amperean Circle

Y Position 0

X Position 0

Figure 9-1: Ampere's Law Screen Capture

9.61.1.1 Help Description

This is a simulation illustrating Ampere's Law for a circular or rectangular imaginary Amperean open surface, in the presence of unit line currents either out of the page or into the page. You begin with one line current out of the page and one line current into the page. You can add additional line currents using the Control Panel, or delete all line currents present and start again.

The line currents can be moved around by left clicking and dragging on the line current. You can select multiple line currents to move about by "ctrl left clicking" on successive line currents. You can use the radio buttons on the Control Panel to choose whether your imaginary open Amperean surface is a circle or rectangle. You can move your Amperean surface around and by using the sliders in the Control Panel. You will see tangents to the Amperean contour (gray arrows) at many points on the contour defining the open surface. At those same points you will see the local magnetic field on the contour due to all the line currents in the scene.

If you left click and drag in the view, your perspective will change so that you can see the field vector and tangent orientation better. If you want to return to the original view hit the "Reset View" button in the Control Panel.

10 Java Web Start

Java WebStart can be found in the *C:\Program Files\Java\jdk-1.8\bin* folder (or the corresponding folder for the version of Java that the user is running). In that folder there is an application *javaws.exe*. If started the user will find all the applications that have been downloaded cached there. That is, the user does not have to rely on a live web connection to run an application once it has been downloaded it. This is useful in a classroom situation where the user does not want to rely on a web connection.

11 *TEALsim* Overview in the Context of Example_09

11.1 The Simulation (*SimWorld, TSimulation*) :

We now examine each of the broad categories discussed above in the context of Example_09. We have a ring of current (the yellow torus in Figure 5-12) interacting with the magnetic field of a fixed magnetic dipole (the red/white/blue cylinder in Figure 5-12). The ring of current is assumed to have negligible inductance and is acted upon by gravity in addition to the field of the dipole.

11.2 SimPlayer (*TFramework*) and SimPlayerApp

11.3 SimEngine – the Simulation Engine:

11.4 Viewer and Viewer3D – the Rendering Engine:

11.5 The User Interface:

The current in the ring is set by the user using the slider on the upper right as shown in Figure 5-12. Figure 5-12 shows the controls at the bottom of the view window that allow the user to start, stop, pause, and reset the simulation, as well as step through the simulation one time step at a time. Once the user presses run, the simulation runs until the user presses pause, stop, or reset. The user can also simply press step, which will advance the simulation one time step. The time step is a basic property of the *SimEngine*.

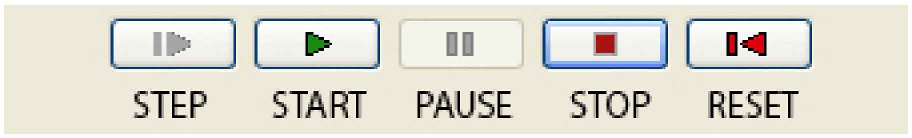


Figure 11-1: Controls for Running Example_09

When the simulation begins, the ring of current is about 1.7 meters above the magnetic dipole, which is located at zero, and has a current of -50 amps (positive current is counterclockwise when viewed from above). As the simulation runs, the ring of current oscillates up and down with an average height of around 1.3 meters. The position of the ring as a function of time is plotted in the graph on the right. When the user hits reset, the ring is returned to its initial position, but the current remains at the last value it was set to using the slider by the user.

At any time as the simulation runs, the user can change the current in the ring by using the slider or by entering the desired current in the box next to the slider **and hitting enter.** The various parts of the UI control appear in separate panels on the right. Controls for the various visualization methods for the magnetic field are contained in the panel entitled "*Field Visualization*". The user can hide the field lines if desired, change their number and color weighting, and change the number of nodes in the vector field grid. The user can also have the simulation stop at any point and calculate a line integral convolution representation of the field by left-clicking on the "*Iron Filings*" button; left-clicking on the "*Magnetic Potential*" button displays a field that is everywhere perpendicular to the magnetic field. The simulation is paused as this calculation is being carried out. To restart the simulation, the user simply left clicks on run again.

12 MIT TEALsim Software License

13 Appendices

13.1 JNLP files

13.1.1 shielding.jnlp

This file is a typical simulation file, and is located as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<!-- JNLP File for TEALsim -->
<jnlp spec="1.0+" codebase="http://web.mit.edu/viz/EM/simulations"
href="shielding.jnlp">
  <information>
    <title>Shielding Simulation</title>
    <vendor>Studio Physics</vendor>
    <homepage href="http://icampus.mit.edu/teal/content/?TEALsim"/>
    <description>Shielding Simulation</description>
    <description kind="short">Shielding Simulation</description>
    <offline-allowed/>
  </information>
  <security>
     <all-permissions/>
  </security>
  <resources>
    <j2se version="1.4+" initial-heap-size="64m" max-heap-size="512m"/>
    <jar href="lib/TEALsim-core.jar" main="true" download="eager"/>
    <jar href="lib/TEALsim-simulations.jar" download="eager"/>
        <extension name="Java3D" href="java3d/java3d.jnlp"/>
  </resources>
  <application-desc main-class="teal.app.SimPlayerApp">
    <argument>-n</argument>
    <argument>tealsim.physics.em.ConductingSphericalShellShielding</argument>
  </application-desc>
</jnlp>
```

13.1.2 java3d.jnlp

This file is

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://web.mit.edu/viz/EM/simulations/java3d/"
 href="http://web.mit.edu/viz/EM/simulations/java3d/java3d.jnlp">
  <information>
    <title>Java 3D 1.5.2 ( OpenGL )</title>
    <vendor>Sun Microsystems Inc.</vendor>
    <homepage href="http://java3d.dev.java.net"/>
    <description>Java 3D library</description>
    <offline-allowed/>
  </information>
  <security>
   <all-permissions/>
  </security>
  <resources os="Windows">
   <jar href="j3d/1.5.2/windows/j3dcore.jar" download="eager"/>
   <jar href="j3d/1.5.2/windows/j3dutils.jar" download="eager"/>
   <jar href="j3d/1.5.2/vecmath.jar" download="eager"/>
   <nativelib href="j3d/1.5.2/windows/j3dcore-ogl_dll.jar" download="eager"/>
  </resources>
  <component-desc />
</jnlp>
```

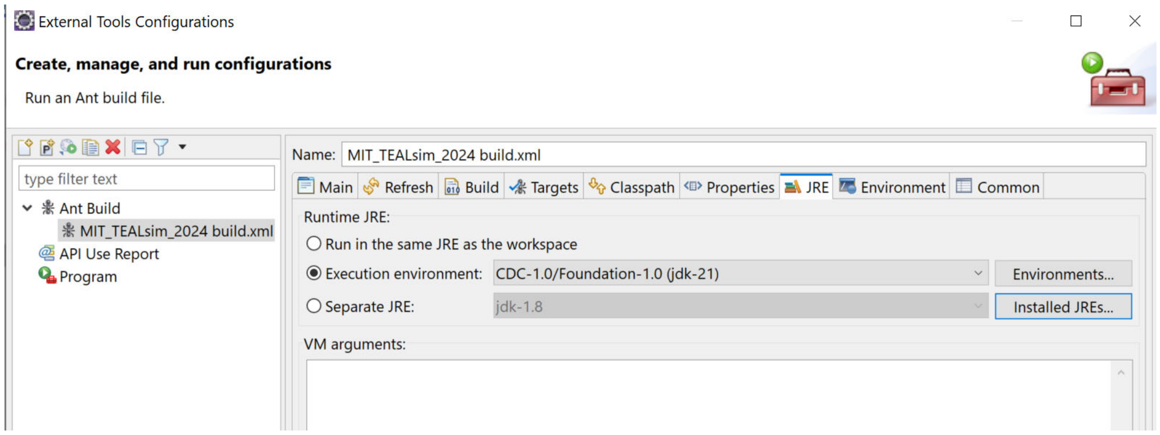
The 2012 version is

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://starapp.mit.edu/viz/EM/simulations/java3d/"
 href="http://web.mit.edu/viz/EM/simulations/java3d/java3d.jnlp">
  <information>
    <title>Java 3D 1.4.0_01 ( OpenGL )</title>
    <vendor>Sun Microsystems Inc.</vendor>
    <homepage href="http://java3d.dev.java.net"/>
    <description>Java 3D library</description>
    <offline-allowed/>
  </information>
  <security>
    <all-permissions/>
  </security>
  <resources os="Windows">
    <jar href="j3d/1.4.0_01/windows-i586/j3dcore.jar" download="eager"/>
    <jar href="j3d/1.4.0_01/windows-i586/j3dutils.jar" download="eager"/>
    <jar href="j3d/1.4.0_01/vecmath.jar" download="eager"/>
    <nativelib href="j3d/1.4.0_01/windows-i586/j3dcore-ogl_dll.jar"
download="eager"/>
    <nativelib href="j3d/1.4.0_01/windows-i586/j3dcore-d3d_dll.jar"
download="eager"/>
    <nativelib href="j3d/1.4.0_01/windows-i586/j3dutils_dll.jar" download="eager"/>
  </resources>
  <resources os="Linux" arch="i386">
    <jar href="j3d/1.4.0_01/linux-i586/j3dcore.jar" download="eager"/>
    <jar href="j3d/1.4.0_01/linux-i586/j3dutils.jar" download="eager"/>
    <jar href="j3d/1.4.0_01/vecmath.jar" download="eager"/>
    <nativelib href="j3d/1.4.0_01/linux-i586/lib_j3dcore-ogl_so.jar"
download="eager"/>
    <nativelib href="j3d/1.4.0_01/linux-i586/lib_j3dutils_so.jar" download="eager"/>
  </resources>
  <resources os="SunOS" arch="sparc">
    <jar href="j3d/1.4.0_01/solaris-sparc/j3dcore.jar" download="eager"/>
    <jar href="j3d/1.4.0_01/solaris-sparc/j3dutils.jar" download="eager"/>
    <jar href="vecmath/1.4.0_01/vecmath.jar" download="eager"/>
    <nativelib href="j3d/1.4.0_01/solaris-sparc/lib_j3dcore-ogl_so.jar"
download="eager"/>
    <nativelib href="j3d/1.4.0_01/solaris-sparc/lib_j3dutils_so.jar" download="eager"/>
  </resources>
  <resources os="SunOS" arch="x86">
    <jar href="j3d/1.4.0_01/solaris-x86/j3dcore.jar" download="eager"/>
    <jar href="j3d/1.4.0_01/solaris-x86/j3dutils.jar" download="eager"/>
    <jar href="vecmath/1.4.0_01/vecmath.jar" download="eager"/>
    <nativelib href="j3d/1.4.0_01/solaris-x86/lib_j3dcore-ogl_so.jar"
download="eager"/>
    <nativelib href="j3d/1.4.0_01/solaris-x86/lib_j3dutils_so.jar" download="eager"/>
  </resources>
  <component-desc />
</jnlp>
```

13.2 Generating, signing, and resigning jars

13.2.1 Creating the TEALsim jars

To generate the *TEALsim-core.jar* and *TEALsim-simulations.jar* jars, we use *Ant* and the *build.xml* file in *eclipse*. We must have Java 21 installed to generate the jars; we cannot do it using Java 8/1.8 (https://bugs.eclipse.org/bugs/show_bug.cgi?id=579317). To do this, select the build.xml file in Package Explorer (Figure 3-1) and *Run as > External Tools Configuration*. Change the JRE to jdk-21.


External Tools Configurations

Create, manage, and run configurations

Run an Ant build file.

Name: MIT_TEALsim_2024 build.xml

Runtime JRE:

○ Run in the same JRE as the workspace

● Execution environment: CDC-1.0/Foundation-1.0 (jdk-21)

○ Separate JRE: jdk-1.8

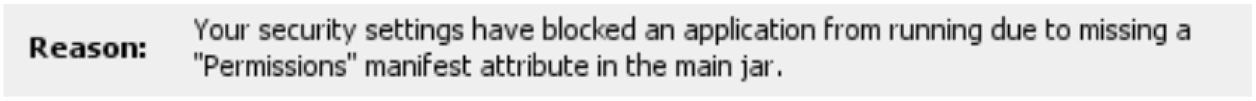
VM arguments:

Figure 13-1: Running an Ant build fille

Running this ***will delete*** and recreate the release directory (see Figure 3-1), creating the new jars. In the *javac* commands in the *build.xml* file, we specify Java 8/1.8. We do this so that that the compilation will be done in 8/1.8 and not 21.

```
<javac srcdir="${build.dir}/src" destdir="${build.dir}/classes"
excludes="tealsim.physics.ilab" includes="tealsim/**" debug="on"
optimize="off" target="1.8" compiler="javac1.8" source="1.8"
executable="C:\Program Files\Java\jdk-1.8\bin\javac" fork="true">
```

This will create our jar files *TEALsim-core.jar* and *TEALsim-simulations.jar*, unsigned. Before we sign them, we must insert a change in the “Permissions” manifest attribute in the main jar *TEALsim-core.jar;* otherwise, when we try to download and run them from the web we will get the following error message:


Reason: Your security settings have blocked an application from running due to missing a "Permissions" manifest attribute in the main jar.

(https://stackoverflow.com/questions/17338711/jarsigner-manifest-permissions)

To change the Permissions, create a file *manifest_adder.mf,* a text file consisting of a single line: “Permissions: all-permissions”, and give the following command in PowerShell:

jar ufm TEALsim-core.jar manifest_adder.mf

13.2.2 Signing the TEALsim jars

I obtained an *Open Source Code Signing in the Cloud* certificate from https://shop.certum.eu/code-signing.html , currently priced at 49 euros. I chose this source because you do not have to be associated with a company or institution, and the certificate comes at a relatively cheap price. Once you have this certificate set up the way they instruct, the command

jarsigner -keystore NONE -certchain "bundle.pem" -sigalg SHA256withRSA -tsa "http://time.certum.pl" -storetype PKCS11 -providerClass sun.security.pkcs11.SunPKCS11 -providerArg "provider.cfg" -storepass "12341234" "TEALsim-core.jar" "1597966324D7039A2AEAE45242E08B6A"

will sign the jar, after you have signed into their *SimplySign* desktop app. You will get a warning when you sign, ignore that.

To sign into their *SimplySign* app, go to search box and search for *SimplySign.* Start this, you must on your phone get a verification code from the *SimplySign* app on your phone. Once you are properly logged in on your desktop app, give the above command in power shell and the signing will take place.

Also do the following:

jarsigner -keystore NONE -certchain "bundle.pem" -sigalg SHA256withRSA -tsa "http://time.certum.pl" -storetype PKCS11 -providerClass sun.security.pkcs11.SunPKCS11 -providerArg "provider.cfg" -storepass "12341234" "TEALsim-simulations.jar" "1597966324D7039A2AEAE45242E08B6A"

The commands
jarsigner -verify .\TEALsim-simulations.jar

jarsigner -verify -verbose .\TEALsim-core.jar

will verify the signing.

The files *bundle.pem* and *provider.cfg* are files you must create as per their instructions, and they must be present in the directory that contains the jar files. Once this is done move the jar files into

/afs/athena.mit.edu/course/other/viz/EM/simulations/lib

13.2.3 Re-signing the Java3d 1.5.2 jars

The vecmath.jar is at
afs/athena.mit.edu/course/other/viz/EM/simulations/java3d/j3d/1.5.2

j3dcore-ogl_dll.jar j3dcore-ogl.dll j3dutils.jar j3dcore.jar

/afs/athena.mit.edu/course/other/viz/EM/simulations/java3d/j3d/1.5.2/windows

13.3 Creating 3D Objects for Import into Java3D

13.3.1 Creating Native *Java3D* Objects

13.3.2 Creating and Importing *Autodesk 3ds Max 8 .3DS* Scene Files

The scale conversion from 3ds max to java 3D is as follows. The height of the cone imported below in the max 3ds file is 200 inches. This is for "Customize/Units Setup" US Standard Decimal Inches and for "Customize/Units Setup/System Unit Setup" 1 unit = 1 inch in Autodesk 3ds Max 8. The conversion between max units and Java3D units is under these circumstances 1 Java3D unit = 1 Max inch. Thus when we scale the cone by a factor of 0.01 it has a height of 2 Java units

13.3.3 Importing *Wavefront .obj* Scene Files

The scale conversion from the .obj file to java 3D is as follows: The height of the box imported below in the .obj file is 100 obj units. We have scaled it by a factor of two, and it then appears in Java 3D as 1 unit high. Thus the conversion is 200 obj units = 1 Java 3D unit, or 1 obj unit = .005 Java 3D unit.

13.3.4 Importing *VRML .wrl* Scene Files

13.3.5 Importing *Lightwave* 3D Scene Files

More memory VM -Xmx512m

13.4 A Guide to Programming Resources for Java 3D

13.4.1 The Java Language

13.4.2 Java 3D

TEALsim is based on Java 3D (http://java.sun.com/products/java-media/3D/) .

This document is intended to be used in conjunction with the browser-accessible, javadoc-generated reference for the code base.

14 Loading 3DS Models

14.1 axes

mapping y axis in max goes into –z in java3d
x axis in max goes into –x axis in java3d
z axis in max goes into y axis in max

14.2 textures

15 Classes

15.1 teal

15.1.1 app

15.1.2 audio

15.1.3 browser

15.1.4 config

15.1.5 core

15.1.6 field

15.1.7 framework

15.1.8 math

15.1.9 physics

15.1.10 plot

15.1.11 render

15.1.12 Sim

15.1.12.1 behavior

15.1.12.2 collision

15.1.12.3 constraint

15.1.12.4 control

15.1.12.5 engine

15.1.12.6 function

15.1.12.7 properties

15.1.12.8 simulation

15.1.12.9 spatial

15.1.12.9.1 AccelerationVector.java

15.1.12.9.2 ComponentForceVector.java

15.1.12.9.3 FieldConvolution.java

15.1.12.9.4 FieldDirectionGrid.java

15.1.12.9.5 FieldLine.java

15.1.12.9.6 FieldLineManager.java

15.1.12.9.7 FieldValue.java

15.1.12.9.8 FieldVector.java

15.1.12.9.9 FluxFieldLine.java

15.1.12.9.10 ForceVector.java

15.1.12.9.11 GeneralVector.java

15.1.12.9.12 RadiationFieldLine.java

15.1.12.9.13 RelativeFLine.java

15.1.12.9.14 Spatial.java

15.1.12.9.15 SpatialField.java

15.1.12.9.16 SpatialPhysicalObject.java

15.1.12.9.17 SpatialTextLabel.java

15.1.12.9.18 SpatialVector.java

15.1.12.9.19 SpatialVisualization.java

15.1.12.9.20 TrailVisualization.java

15.1.12.9.21 VelocityVector.java

15.1.13 ui

15.1.14 util

15.1.15 visualization

16 References

No table of authorities entries found.

17 Index