

**Implicit Material Point Method
For
Cell Biomechanics**

USER'S MANUAL

Edited by Jim Guilkey, Ph.D. & Ben Ellis

Last updated : May 24, 2007

TABLE OF CONTENTS

TABLE OF CONTENTS	1
1. Introduction	2
2. Building An Implicit MPM Code	2
2.1. STEP 1: Download Source Code and 3 rd Party Software	2
2.2. STEP 2: Unpack the Installation Files	4
2.3. STEP 3: Run “Configure”	4
2.4. STEP 4: Set Environment Variables	4
2.5. STEP 5: “Make” the Code	4
2.6. STEP 6: Set up Soft Directory Links	5
3. Running Implicit MPM	6
3.1. Getting Started	6
3.2. Example Problem #1 – Compression of an Elastic Billet	7
3.3. Example Problem #2 – Cell Surrogate Embedded in a Collagen Substrate	10

1. Introduction

The implicit Material Point Method (MPM) code used in the angiogenesis study is one component within the Uintah Computational Framework (UCF) which is, in turn, one of the software packages within the SCIRun Problem Solving Environment (PSE). The UCF contains a number of other components, including an explicit MPM implementation, a compressible CFD component known as ICE, as well as a fluid structure interaction component known as MPMICE. The SCIRun PSE also contains visualization tools for interacting with the data created by the Uintah components.

2. Building An Implicit MPM Code

The following are instructions for building our implicit MPM code. While getting an executable will take something of an investment in time, once complete, the user will have a massively parallel, scalable simulation tool, as well as visualization tools for interrogating the data. Furthermore, these instructions are for building on a generic Linux platform. The code will also build on the Macintosh platform. As this is high performance software, Windows is not supported.

2.1. STEP 1: Download Source Code and 3rd Party Software

To begin, one needs to download the SCIRun source code, and some third party software that supports it. A .zip file containing the SCIRun source code, the third party software, and this manual can be downloaded from the MRL website at:

http://mrl.sci.utah.edu/software.php?title_id=3

The latest version of the SCIRun source can be obtained by:

```
svn checkout https://code.sci.utah.edu/svn/SCIRun/trunk/ SCIRun
```

This will take approximately 15 minutes to download the entire source.

Most of the necessary Thirdparty software has been bundled and can be obtained by:

```
svn co https://code.sci.utah.edu/svn/Thirdparty
```

This will create a directory called Thirdparty, which contains several subdirectories:

```
1.25.4  3.0.0  3.0.1  3.0.2  3.1.0
```

You may remove all but the 1.25.4 directory:

```
rm -rf 3.*
```

To install the Thirdparty package, do:

```
./install.sh <desired-thirdparty-location> <desired-bits>
```

Note that you will have to specify if you want to build for a 32bit or 64 bit system. Just specify the number '32' or '64' into the install command above. For example, if you

wanted to install Thirdparty in /usr/local/SCIRun/Thirdparty on a 32 bit machine, you would run:

```
./install.sh /usr/local/scirun/Thirdparty 32
```

Do not try and add any nested directories for different compilers, versions, bitness, etc. The Thirdparty install script will sort these out for you automatically.

After the install script is finished, you should see the following output on your screen:

```
VERIFYING THE INSTALLATION...
```

```
Final verification results:
```

```
-----
png:             installed
freetype:        installed
Teem:            installed
mpeg:            installed
libxml2:         installed
glew:            installed
tcl:             installed
tk:             installed
itcl:            installed
blt:            installed
```

```
Updating permissions on thirdparty files (go+rX).
```

```
  bin...
  include...
  lib...
  man...
  share...
  src...
```

```
-----
Please use the following as an argument when
configuring SCIRun:
```

```
  --with-thirdparty=<path-to-thirdparty>
```

The last bit gives the user part of the configure line that they will use in configuring SCIRun, which will be covered shortly.

One additional third party software package is required, namely PETSC, the Portable Extensible Toolkit for Scientific Computing. The implicit MPM code makes use of the iterative linear solvers available in PETSC.

The PETSC source code can be downloaded from:

<http://www-unix.mcs.anl.gov/petsc/petsc-as/download/index.html>

petsc-2.2.1 is the version that should be most straightforward to use. Newer version (e.g. 2.3.2) will also work but will likely require small hacks to the SCIRun configure script. Build instructions can be found on the PETSC website, but briefly, for version 2.2.1:

2.2. STEP 2: Unpack the Installation Files

Unzip and untar the PETSc installation:

```
gunzip petsc-2.2.1.tar.gz
tar -xf petsc-2.2.1.tar
```

2.3. STEP 3: Run “Configure”

Change directory to the PETSc directory you just untarred and run it's configure command.

```
cd petsc-2.2.1
./config/configure.py --with-x=false --with-matlab=false --download-f-
blas-lapack=ifneeded --with-mpi-dir=<path-to-mpi>
```

If it fails with something like 'unable to use mpicc', then pass it the compilers you are using:

```
--with-cc=<C-compiler> --with-cxx=<CXX-compiler> --with-f77=<fortran-
compiler> --with-mpi-compilers=false
```

2.4. STEP 4: Set Environment Variables

After configuring, there are two environment variables, PETSC_DIR and PETSC_ARCH that must be set before proceeding. At the end of the PETSc configure output, PETSc will tell you to what these two variables need to be set.

For csh-type shells type:

```
setenv PETSC_DIR <petsc-dir>
setenv PETSC_ARCH <petsc-arch>
```

For sh-type shells type:

```
PETSC_DIR=<petsc-dir>
PETSC_ARCH=<petsc-arch>
export PETSC_DIR
export PETSC_ARCH
```

2.5. STEP 5: “Make” the Code

Now you can issue the make command from the petsc directory. Note that the following command contains the letter "O" and NOT the number "0":

```
cd ${PETSC_DIR}
make BOPT=O
```

2.6. STEP 6: Set up Soft Directory Links

Currently, the SCIRun configure script might not agree with PETSc as to what the architecture is called so you will have to make a soft link that is named whatever SCIRun expects that points to what PETSc created. For example:

```
cd ${PETSC_DIR}/bmake
ln -s linux-gnu linux
cd ${PETSC_DIR}/lib/libO
ln -s linux-gnu linux
```

Once Thirparty and PETSC have been built, you are ready to configure and build SCIRun. Start by going into the SCIRun directory:

```
cd SCIRun
mkdir opt
cd opt
../src/configure --enable-package=Uintah --with-thirdparty=<path-to-
thirdparty> --enable-optimize='-march=pentium4 -msse -msse2 -O3' --
enable-assertion-level=0 --with-petsc=<path-to-petsc>
```

For the <path-to-thirdparty> you will pass the command that thirdparty configure reported (see above). For the <path-to-petsc> it expects the top level petsc directory, e.g. /usr/local/petsc-2.2.1 The optimization options given above are for a pentium4 machine, these will be somewhat platform dependent. If in doubt, just using -O3 is a good start. If you are on a 64bit machine, you will want to have:

```
--enable-64bit in your configure line as well.
```

Assuming that the petsc and thirdparty installations worked, configure should also work. Note that this is assuming that the user has an MPI version (lam or mpich preferably) already installed. If not, you may want to visit:

```
http://www.csafe.utah.edu/Information/Thirdparty/mpi.html
```

to learn more.

Assuming that configure has completed successfully, you are now ready to build the code. From within the opt directory you can just do:

```
make
```

or if you have a multiprocessor machine, you can speed this up by doing:

```
make -j 4
```

Getting a full optimized build will likely take 30-45 minutes.

3. Running Implicit MPM

3.1. Getting Started

To get started, the executable, called "sus" lives in:

```
SCIRun/opt/Packages/Uintah/StandAlone/
```

to go there:

```
cd SCIRun/opt/Packages/Uintah/StandAlone/
```

if you just type "sus" at the command line, you should get usage information:

```
No input file specified
Usage: sus [options] <input_file_name>

Valid options are:
-h[elp]           : This usage information.
-mpm             : option for explicit MPM
-impm           : option for implicit MPM
-mpmf           : option for Fracture
-rmpm           : option for rigid MPM
-smpm           : option for shell MPM
-smpmice        : option for shell MPM with ICE
-rmpmice        : option for rigid MPM with ICE
-fmpmice        : option for Fracture MPM with ICE
-ice            :
-arches         :
-AMR            : use AMR simulation controller
-nthreads <#>   : Only good with MixedScheduler
-layout NxMxO   : Eg: 2x1x1. MxNxO must equal number
                  of boxes you are using.
-emit_taskgraphs : Output taskgraph information
-restart        : Give the checkpointed uda directory as the input
file
-combine_patches : Give a uda directory as the input file
-reduce_uda     : Reads <uda-dir>/input.xml file and removes
unwanted labels (see FAQ).
-uda_suffix <number> : Make a new uda dir with <number> as the default
suffix
-t <timestep>     : Restart timestep (last checkpoint is default,
                  you can use -t 0 for the first checkpoint)
-svnDiff         : runs svn diff <src/...../Packages/Uintah
-copy           : Copy from old uda when restarting
-move          : Move from old uda when restarting
-nocopy        : Default: Don't copy or move old uda timestep
when
                  restarting
```

3.2. Example Problem #1 – Compression of an Elastic Billet

For starters, we'll run a 2D implicit MPM simulation of the compression of an elastic billet by rigid platens (Figure 1). An input file describing this simulation is at:

```
inputs/UintahRelease/IMPM/billet.ups
```

(Note that all input files end with the ups extension, which indicates Uintah Problem Specification.)

A full description of this problem can be found in:

Guilkey JE, Weiss JA, Implicit time integration with the Material Point Method. *International Journal for Numerical Methods in Engineering*, 57:1323-1338, 2003.

To run this simulation, do:

```
sus -impm
inputs/UintahRelease/IMPM/billet.ups
```

sus will generate output describing its progress. The entire simulation will take 1-2 minutes. Data from the simulation is stored in `billet.static.uda.000` (Here, `uda` stands for Uintah Data Archive, and is the standard extension for data sets generated by Uintah codes. Also, note the `000` extension. If you run the simulation again, there will be a new data set with a `001` extension. This is to prevent data from being clobbered by subsequent runs.)

To visualize the data that you've just created, you'll use the `scirun` executable, which sits at `SCIRun/opt/scirun`

To get started, you may want to open a new shell, and:

```
cd SCIRun/opt
```

and then do:

```
scirun
../src/Packages/Uintah/StandAlone/inputs/UintahRelease/IMPM/billet.srn
```

In this case, the `.srn` extension refers to a `scirun` "network", which is a file that described the dataflow throughout the various `scirun` modules being used to render the data. Executing the above command will begin to make this clear. When you do, the upper left

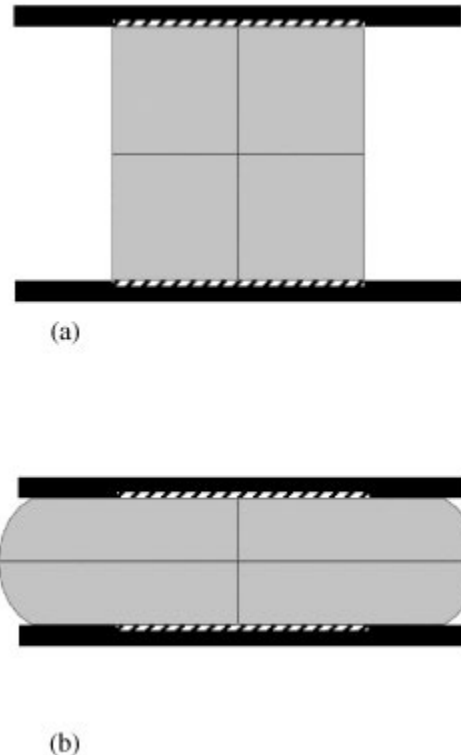


Figure 1. Compression of a 2D elastic billet showing (a) the uncompressed and (b) the compressed configurations.

module will be an ArchiveReader. Click on the UI (User Interface) button, click "Select UDA" and navigate to the `billet.static.uda.000` directory (note, you want to be inside of this directory) and hit OK. When you do this the first time, you may see bars inside some of the boxes turn yellow. This indicates that those modules are dynamically compiling. When everything turns green, the dataflow is complete, and you can click on the UI in the Viewer module. This will bring up a window with a rendering of your data (Figure 2). You may need to hit the "Autoview" button to get a correct view. Within that window, you can translate, rotate and zoom using your mouse buttons (play with this to get a feel for it). Click on the UI for the ParticleFieldExtractor and make sure that the radio buttons for both material 1 and 2 are selected. Clicking on the UI for the TimestepSelector will allow you to move through different timesteps of the simulation. Describing the full capabilities of these tools is difficult, playing around the different modules is the best way to become familiar, and remember, it's software, it may crash, but you can't really break anything.

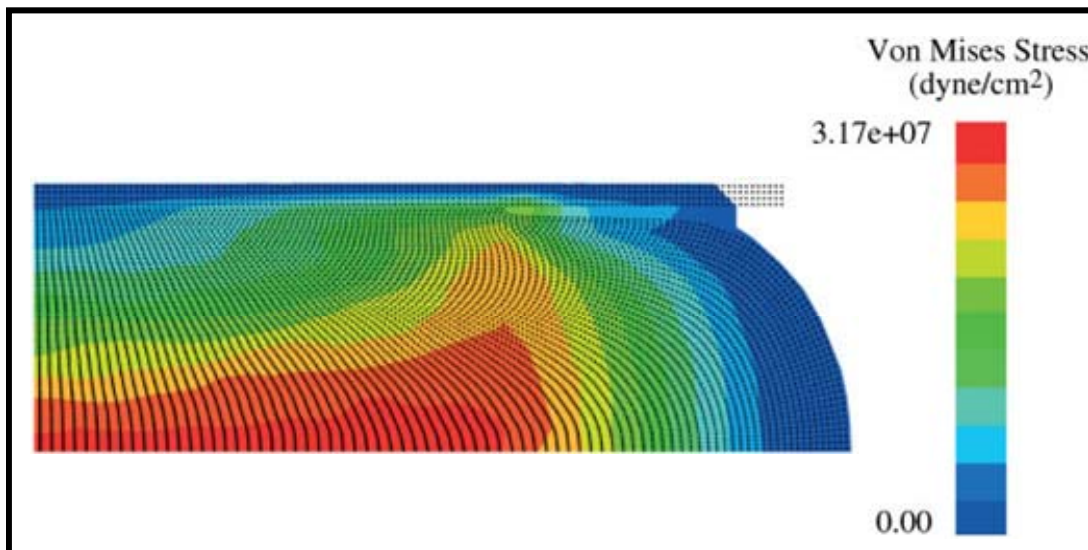


Figure 2. Rendering of the Von Mises Stress distribution within the $\frac{1}{4}$ symmetry model of the billet during compression.

Backing up a little bit, the user may wish to know a bit about the input files. Open up `inputs/UintahRelease/IMPM/billet.ups`. All `.ups` files are in xml format, which is convenient for a number of reasons. The first interesting section, `<Time>` describes the relevant timestepping data. The `<Grid>` section describes the extents of the grid (`<lower>` and `<upper>`) and the resolution (`<spacing>`). Note that the patch distribution is currently set to `[1,1,1]`. To run this simulation with two processors, one would simply change this to, say, `[2,1,1]` and run using:

```
mpirun -np 2 sus -impd inputs/UintahRelease/IMPM/billet.ups
```

Still in the grid section, is <BoundaryCondition> information. Here we have set planes of symmetry on all 6 faces. This first achieves the 1/4 symmetry that we can use for this problem, and also results in plane strain in the out of plane direction (Figure 3).

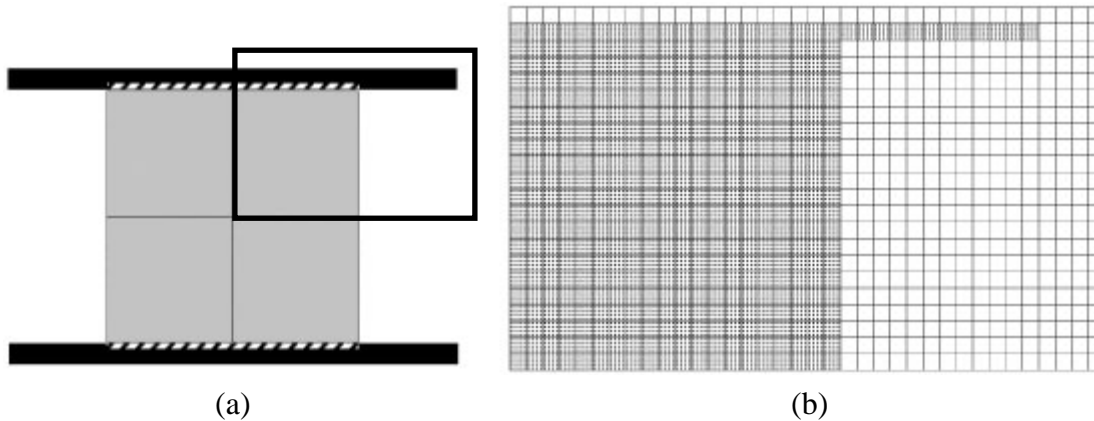


Figure 3. Billet $\frac{1}{4}$ symmetry model showing (a) the portion being discretized and (b) the resulting background grid and material point distribution.

Next is the DataArchiver which indicates what data will be saved and how frequently it will be saved. It also indicates checkpoint saving every .05 seconds. This allows a restart capability, in case a job dies, or one wishes to extend the run time. To restart, one would do:

```
sus -impm -restart billet.static.uda.000
```

Next, the <MPM> section includes information about the solver. The <MaterialProperties> section actually includes physical properties for the materials of interest, as well as geometry description, as well as initial conditions for that piece of geometry. Uintah makes use of a number of geometric primitives (box, cylinder, sphere, etc.), as well as boolean operations (union, intersection, difference) in order to describe geometry. (Other ways will be described below.) Here, the first material is the rectangular billet, with dimensions 20 X 20. You also see that the material is a compressible Neo-Hookean material, there are a large number of material models available in Uintah-MPM. The <res> section describes how many particles are in each cell in each dimension. The next material is the platen which will be compressing the billet. Note that it has an initial velocity of [0.0,-50.0,0.0]. The subsequent <contact> section indicates that rigid contact will be employed. The second material is denoted as being rigid, which means that it will continue to move with its initial velocity immune to dynamics. This is how displacement boundary conditions are usually achieved.

3.3. Example Problem #2 – Cell Surrogate Embedded in a Collagen Substrate

For the user who is specifically interested in simulations of multicellular constructs, an example is provided to help them get started. The example that we'll consider is actually a surrogate for a cellular construct in which rods of Tecoflex fiber are embedded in a collagen substrate (Figure 4).

Image data was collected via micro-CT and is stored in a raw file called `teco_288_896_1.raw`. This raw file consists of 8 bit intensity (0-255) stored at each of the 288 X 896 pixels. This file can be viewed in, for instance, NIH ImageJ, but for the user, a jpeg version is also supplied `teco_288_896_1.jpg`. (Note that this two-dimensional image is one slice from a three-dimensional image.) Based on settings on the micro-CT equipment, each pixel represents an area $2.06e-3$ cm squared.

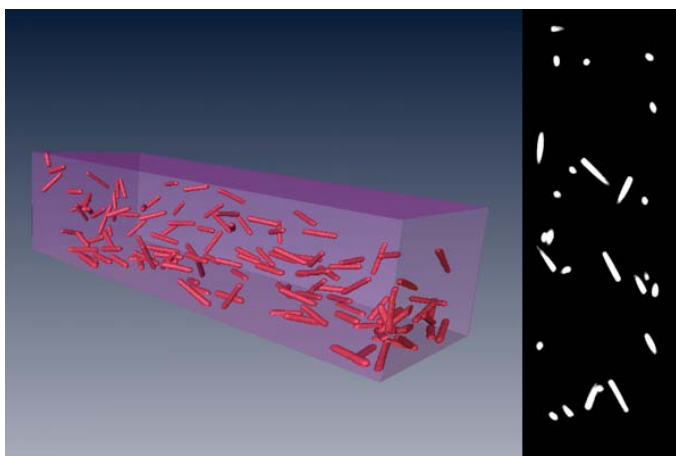


Figure 4. Volume rendering from CT data of surrogate vessels in collagen gel (left). Single CT slice (right).

An input file is provided, `Teco_collagen_2D.ups`, that simulates 6% extension in the long dimension of the sample. This is achieved by placing a rigid plate at the top of the sample which is bonded in the direction of motion to the sample, but allows lateral contraction. The plate is given an initial velocity, it then pulls the sample in the direction of motion for the specified time. The sample is specified in two locations in the input file, first to specify the collagen matrix:

```
<image>
  <name>teco_288_896_1.raw</name>
  <res>[288,896,1]</res>
  <threshold>[0,119]</threshold>
</image>
<file>
  <name>syn_col.pts</name>
  <format>bin</format>
</file>
<res>[8,8,1]</res>
```

and next to specify the Tecoflex fibers:

```
<image>
  <name>teco_288_896_1.raw</name>
  <res>[288,896,1]</res>
  <threshold>[120,255]</threshold>
</image>
<file>
  <name>syn_ves.pts</name>
  <format>bin</format>
</file>
<res>[8,8,1]</res>
```

These two sections are nearly identical, but differ primarily in that the first section, which specifies the collagen material, indicates that pixels with an intensity between 0 and 119 will be considered to be collagen. Similarly, the "vessel" section indicates that those pixels with an intensity between 120 and 255 will be assumed to be Tecoflex. The <name> section indicates the root name for files that will be generated containing the respective points for each, following a preprocessing step described shortly. The <format>bin</format> section indicates that those files are expected to be binary. Also in these sections, are the <res>[8,8,1]</res> tags. These indicate that there will be 8 particles per cell in the x and y direction, and only 1 particle in the z (due to the 2D nature of this case). For simulations involving image data, there is an inverse relationship between the res tag specified here, and the Grid resolution specified at the bottom of the file. In reality, this problem is probably under resolved, and would give better results with 4X4X1 particles. In this case, one would simply change the <res> variable accordingly, and double the number of cells specified in the Grid section.

In order to proceed with the simulation, one must first use a utility called pfs2. The executable is at:

```
SCIRun/opt/Packages/Uintah/StandAlone/tools/pfs/pfs2
```

To use it, do:

```
SCIRun/opt/Packages/Uintah/StandAlone/tools/pfs/pfs2 -b
Teco_collagen_2D.ups
```

pfs stands for "particle file splitter". Its purpose is two-fold, first, to split the image files out according to intensity, as described above, and also, for multi-processor runs, to divide the particle files up according to the patch distribution. The latter function allows each processor to only read in those particles that belong to it, and this prevents possibly 10s or 100s of processors all trying to access the same file from which they only need a small amount of information.

The result of the above operation is, in this case:

```
syn_col.pts.0          syn_ves.pts.0
```

These are the file names specified in the input file, and the .0 extension means that they go with patch and processor 0 (if there were multiple patches specified, there would be a series of each of these files).

At this point, one can do:

```
SCIRun/opt/Packages/Uintah/StandAlone/sus -impm Teco_collagen_2D.ups
```

which will begin the simulation. The full simulation on a single 2.4 GHZ P4 takes about 2 hours.

Since Uintah is part of the SCIRun problem solving environment, one can use scirun visualization tools to look at the newly created data. To do this:

```
SCIRun/opt/scirun Teco.srn
```

The Teco.srn file is a so-called scirun network file. It contains a series of modules that constitute a data flow, where data is piped from one module to another, and finally into a viewer window. To load your data, click on the UI button in the ArchiveReader module, and navigate to your dataset. Note that you want to go inside the directory, so that you see a series of directories like t00001 and so on. Click OK, and if the Viewer window is not already open, click on the UI in the Viewer module. Next, click on the UI button for the ParticleFieldExtractor, and within it, click on the radio buttons for material 1 and 2 (in addition to the already selected material 0). This should result in a view such as that shown in equiv_stress.jpg (Figure 4). The full range of features of scirun is exhaustive, interested users are encouraged to contact the MRL for more information.

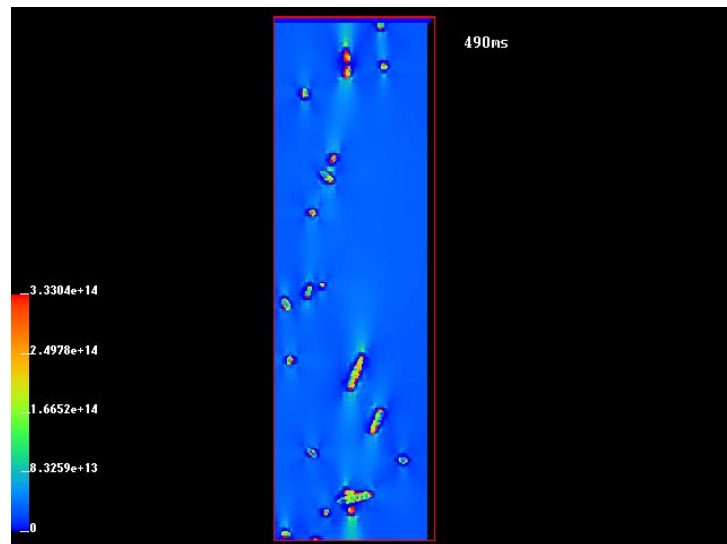


Figure 5. Rendering of the stress distribution within the cell surrogate embedded collagen gel.