MODELING OF TRANSPORT IN LITHIUM ION BATTERY ELECTRODES

A Thesis

by

MICHAEL ADAM MARTIN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2012

Major Subject: Mechanical Engineering

Modeling of Transport in Lithium Ion Battery Electrodes

MODELING OF TRANSPORT IN LITHIUM ION BATTERY ELECTRODES

A Thesis

by

MICHAEL ADAM MARTIN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Co-Chairs of Committee,	Devesh Ranjan
	Partha P. Mukherjee
Committee Members,	Kalyan Annamalai
	Raytcho Lazarov
Head of Department,	Jerald Caton

May 2012

Major Subject: Mechanical Engineering

ABSTRACT

Modeling of Transport in Lithium Ion Battery Electrodes. (May 2012)

Michael Adam Martin, B.S., Texas A&M University

Co-Chairs of Advisory Committee: Dr. Devesh Ranjan
Dr. Partha P. Mukherjee

Lithium ion battery systems are promising solutions to current energy storage needs due to their high operating voltage and capacity. Numerous efforts have been conducted to model these systems in order to aid the design process and avoid expensive and time consuming prototypical experiments. Of the numerous processes occurring in these systems, solid state transport in particular has drawn a large amount of attention from the research community, as it tends to be one of the rate limiting steps in lithium ion battery performance. Recent studies have additionally indicated that purposeful design of battery electrodes using 3D microstructures offers new freedoms in design, better use of available cell area, and increased battery performance.

The following study is meant to serve as a first principles investigation into the behaviors of 3D electrode architectures by monitoring concentration and cycle behaviors under realistic operating conditions. This was accomplished using computational tools to model the solid state diffusion behavior in several generated electrode morphologies. Developed computational codes were used to generate targeted structures under prescribed conditions of particle shape, size, and overall morphology. The diffusion

processes in these morphologies were simulated under conditions prescribed from literature.

Primary results indicate that parameters usually employed to describe electrode geometry, such as volume to surface area ratio, cannot be solely relied upon to predict or characterize performance. Additionally, the interaction between particle shapes implies some design aspects that may be exploited to improve morphology behavior. Of major importance is the degree of particle isolation and overlap in 3D architectures, as these govern gradient development and lithium depletion within the electrode structures. The results of this study indicate that there are optimum levels of these parameters, and so purposeful design must make use of these behaviors.

DEDICATION

This work is first and foremost dedicated to my family, whose constant support and love gave me the strength and perseverance to see this work, and all else, to completion. I further wish to remember every teacher, professor, or mentor who has invested their time and energies in my development as a student and individual. I am forever in their debt. Lastly I would like to mention the other members of my research group, Jacob, Beth, Sarat, Bhanesh, and Bryce, who have been the greatest of friends to me throughout my time at Texas A&M University.

ACKNOWLEDGEMENTS

I would like to thank my co-chairs, Dr. Ranjan and Dr. Mukherjee, for their continued support throughout this work. Working as a team with both of them has been an absolute privilege, and their suggestions and guidance have been invaluable. I would never have had the opportunity to learn about electrochemical systems and renewable energy, subjects that I now find great joy in, without their help. I would also like to thank Dr. Lazarov for his input on several occasions with respect to numerical modeling and technique. My research group has been an invaluable resource for other ideas and possibilities, so I would like to thank Jacob, Bhanesh, Sarat, Beth, and Bryce for any and all help they gave to me. I owe Dr. Jeff Dietiker from NETL and Dr. Sreekanth Pannala from ORNL a large portion of my gratitude for their help with MFiX® and supplemental coding. The initial work for this project was performed during my summer stay with Dr. Mukherjee through the ORNL HERE program, and in my personal case was orchestrated by Dr. John Turner, who I owe a great amount of thanks for his willingness to help and make my stay at ORNL as pleasant as possible.

NOMENCLATURE

a	Activity Coefficient
$\hat{C}$	Specific Capacity
c_p	Specific Heat
D	Diffusion Coefficient
E	Equilibrium Potential
$\hat{E}$	Specific Energy
E	Elementary Charge
F	Faraday's Constant
ΔG	Free Energy of Reaction
ΔH	Enthalpy of Reaction
H	Heat Transfer Coefficient
I_{app}	Applied Cell Current
i	Current Density
j	Reaction Current, Species Flux
k	Reaction Constant
N	Species Flux
N_a	Avogadro's Number
n	Number of Electrons
q	Heat Generation, Heat Loss
R	Particle Radius, Universal Gas Constant

R	Radial Coordinate
S	Electroactive Surface Area
SOC	State of Charge
ΔS	Entropy of Reaction
T	Temperature
t	Transference Number, Time
U	Equilibrium Potential
V	Operating Cell Voltage
v	Velocity

Greek

α	Transfer Coefficient
γ	Activity Coefficient
ε	Porosity, Strain
H	Overpotential
Θ	Fractional Occupancy
κ	Conductivity of Electrolyte
λ	Thermal Conductivity
μ	Electrochemical Potential
ν	Stoichiometric Coefficient
σ	Conductivity
τ	Tortuosity

ϕ	Potential

Superscripts

a	Anode Property
c	Cathode Property
eff	Effective Property
e	Electrolyte Phase Property
p	Positive Electrode
n	Negative Electrode

Subscripts

r	Radial Direction
ref	Value at Reference State
s	Solid Phase Property
surf	Surface Value
max	Maximum Value
Θ	Tangential Direction

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER I

INTRODUCTION AND LITERATURE REVIEW

The growing need for alternative energy resources has been fueled over the past few decades by a growing concern over the environment and rising fuel costs. Both alternative energy storage and production are becoming the center of research for much of the scientific community as the demands of society continue to increase. For these reasons, mathematical and computational modeling of these complex systems is an absolute necessity for advancement, as governing parameters can be easily modified and their resulting effects determined to aid in the design process. Costly and time consuming prototypical experiments cannot be the sole source for the realization of new and important design issues, and so the construction of new, advanced models that can capture true physical effects are paramount to progress in this area.

The following discussion is an overview of some of the experimental and theoretical approaches to modeling the solid state diffusive processes in lithium ion batteries, which are currently promising solutions to current energy storage needs. The first section will introduce the complex nature of the battery, and some issues regarding the mathematical modeling of such an environment. The next section will introduce previous modeling work that has been completed, primarily focusing around that by Doyle and Newman(1), and later adaptations of their work, particularly concerning solid state diffusion. Finally, the advent of 3D microstructures for electrode design will also

---

This thesis follows the style of *The Journal of the Electrochemical Society*.

be introduced, along with previous works completed and design considerations for their implementation.

The Lithium Ion Battery

In general, a battery is composed of two electrodes, an anode and cathode, separated by a porous, electronically insulating layer called the separator, which, however, is conductive to ions. Additionally, these three components are pinned between two current collectors. These aspects are illustrated in Figure 1.

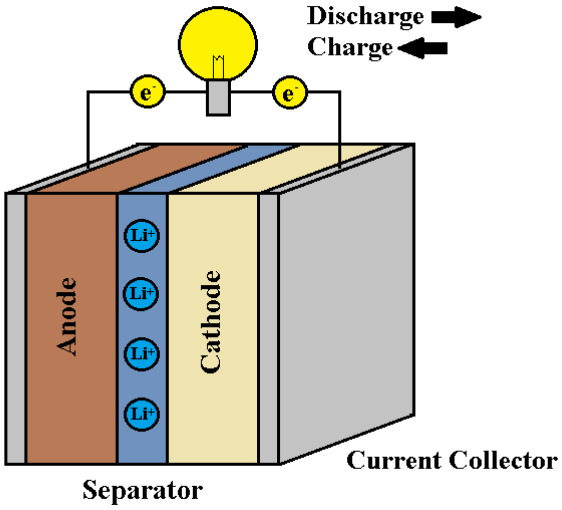

Figure 1. A simple schematic of a typical battery. During discharge, electrons are released from the anode to produce work through the applied load.

Additionally, an electrolyte fills the porous separator to serve as the medium for ion transport. The usefulness of a battery comes from the electrical work it can produce, achieved by movement of electrons through some load as depicted. In particular, the lithium ion battery functions in this way, where, upon discharge, electrons are stripped from lithium atoms in the anode, move through the applied load, and finally end in the

cathode. Simultaneously, the resulting lithium ions leave the anode, or deintercalate, and move into the electrolyte where they travel through the porous separator, and finally recombine with electrons in the cathode. The structure of the three mentioned regions can be varied based upon material and physical construction. Two prominent designs are shown in Figure 2.

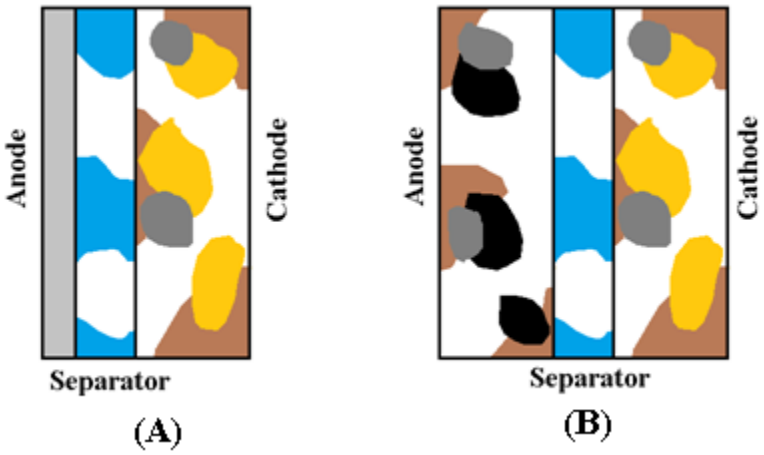

Figure 2. Two prominent designs for lithium ion batteries involve either a metal anode and insertion type cathode (A), or an insertion type electrode for both electrodes.

As can be seen from the above, the two designs employ either a lithium metal anode and what is known as an insertion cathode, as in Figure 2(A), or an insertion type for both electrodes as in Figure 2(B). Insertion type electrodes have drawn a large amount of attention because of their porous nature, which increases the surface area with which lithium ions can react, or intercalate, thereby improving performance. Furthermore, they have proven to be more reversible, or capable of being recharged. These electrodes, as depicted, are a compilation of several different materials, including active material, which provides intercalation sites for lithium ions, conductive filler, which is

electronically conductive, a binder that provides mechanical stability to the electrode, and pores that are filled with the acting electrolyte.

In the case where both electrodes are of the insertion type, which is currently the predominant design of choice, typical materials used are a metal oxide for the cathode, such as Li_yCoO_2, and carbon, Li_xC_6, for the anode. Here the subscripts *x* and *y* are the appropriate stoichiometric values. The reactions occurring at the cathode and anode for this example are represented by Equations (0.1) and (0.2), respectively(2).

$$Li^+ + e^- + CoO_2 \leftrightarrow LiCoO_2 \tag{0.1}$$

$$Li^+ + e^- + C_6 \leftrightarrow LiC_6 \tag{0.2}$$

Thermodynamically, the available work of a battery is determined by the Gibbs free energy, which is related to the enthalpy ΔH and entropy ΔS of reaction, as well as the temperature *T*(3).

$$\Delta G = \Delta H - T\Delta S \tag{0.3}$$

In the above, $T\Delta S$ is the term associated with heat generated during reaction, and the enthalpy and entropy are state values, meaning they are dependent only upon the initial and final states of reaction. However, electronically speaking, the work in a battery is accomplished by electrons moving through a potential, and therefore can be related to the Gibbs free energy through the following relation, where *n* is the number of electrons associated with the reaction, *F* is Faraday's Constant, and *E* is the potential.

$$\Delta G = -nFE \tag{0.4}$$

The relation between the Gibbs free energy and a reaction can additionally be expressed using the van't Hoff isotherm(4), where ΔG^0 is the free energy at the standard state with unit activities, a is the activity of either the products or reactants, R is the universal gas constant, and v_i is the stoichiometric coefficient in the reaction.

$$\Delta G = \Delta G^0 + RT \ln \frac{\prod a_{\text{products}}^{v_i}}{\prod a_{\text{reactants}}^{v_i}} \tag{0.5}$$

Using Equation (0.4), the above can be written into the following, which is the Nernst Equation.

$$E = E^0 - \frac{RT}{nF} \ln \frac{\prod a_{\text{products}}^{v_i}}{\prod a_{\text{reactants}}^{v_i}} \tag{0.6}$$

This is the fundamental relationship that dictates the maximum potential that can be produced by a particular reaction under equilibrium, also known as the open circuit potential.

However, in order to extract useful work from a battery system current must be produced, and so equilibrium is disturbed by biasing the direction of reaction. This departure from equilibrium results in potential loss at the anode and cathode. This is known as the activation overvoltage, or overpotential, and is given the symbol η. The relation between the overpotential, the equilibrium potential, and the potential while drawing current, V, is given below.

$$\eta = E - V \tag{0.7}$$

The overpotential is related to the current density, i, associated with a reaction at an electrode surface by the Butler-Volmer equation, as shown below.

$$i = i_0 \left[e^{\frac{\alpha F \eta}{RT}} - e^{\frac{(1-\alpha) F \eta}{RT}} \right] \tag{0.8}$$

In the above i_0 is the exchange current density, which is a function of the activity, a, and the reaction rate constant k_0 as shown below.

$$i_0 = k_0 F a \tag{0.9}$$

The transfer coefficient, α, when multiplied by the overpotential, governs how a change in overpotential will change the reaction. The above relations are the fundamental equations used in electrochemical processes. However, there are several other phenomena that occur in a battery, especially charge and species transport, as will be discussed later.

To judge battery performance, key parameters are voltage, capacity, power, and energy. Voltage is calculated by subtracting the anode potential from the cathode potential during battery operation. Further, capacity is the amount of charge that can be passed during discharge, and is usually a material property, as certain materials can only host so much lithium. Capacity is of significant importance as it determines the useful range for battery operation. Additionally, plotting the voltage of the cell versus capacity yields what is known as a discharge curve. An example of which is given below.

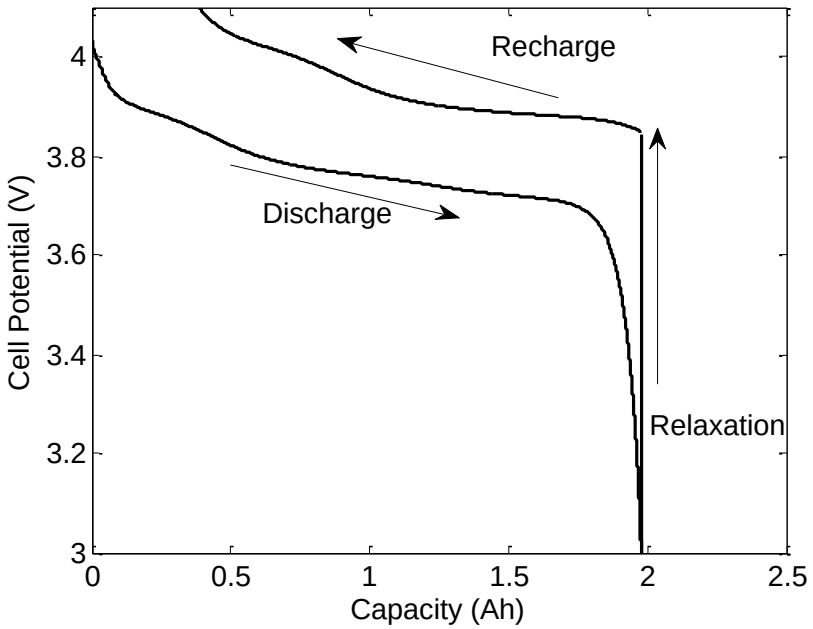

Figure 3. Discharge curve showing discharge to a cutoff potential of 3V, relaxation, and recharge to a cutoff potential of 4.1V.

The above figure illustrates a cycle in which the cell is first discharged at constant current, or under galvanostatic conditions, to a cutoff potential of 3V, where the battery is considered fully spent. It is to be noted here that this can sometimes be a source of confusion, as capacity is technically increasing as the cell is discharged. Instead, one might think of the capacity values along the discharge curve to be that which is spent or consumed. At the cutoff point, the cell is opened and no current flows. Thus the cell is allowed to 'relax', and this is typically associated with a rise in voltage. This occurs because concentration gradients within the electrodes, as well as in the electrolyte, are allowed to become uniform. When this happens, the anode surface concentration grows as lithium is brought to the surface from the core by diffusive processes, as the surface was previously more deprived of lithium due to the discharge process. Simultaneously,

the surface concentration of the cathode drops, as the deposited lithium from the previous discharge process diffuses further into the electrode, lowering the surface concentration. As will be explained later, the open circuit potential functions are usually a function of the surface concentration, and as the above processes occur, the battery approaches the open circuit potential for the resulting surface concentrations. Finally, the battery is recharged to a cutoff potential of 4.1V at constant current. Typically, this can be followed by a voltage hold across the cell, or potentiostatic operation, so that additional current can flow and the cells may be returned to their original state of charge. This is usually stopped after the current drops to some small value. Discharge curves are extremely valuable, as they not only highlight the voltage and capacity response, but also indicate how reversible the processes occurring in the cell are. One can consider the hysteresis in the recharge curve, as well as the capacity regained from the galvanostatic recharge, which would ideally be all of the capacity lost in the discharge process. To compare energy and power, a Ragone plot is usually employed, and an example of such is shown below.

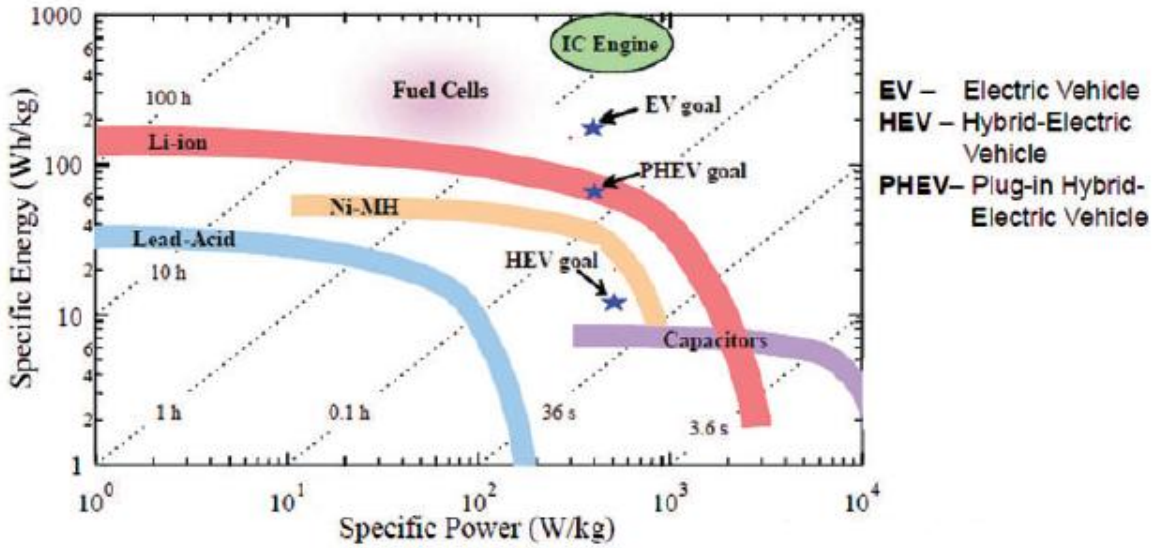

Figure 4. A Ragone plot comparing the specific power and energy of different electrochemical systems, such as batteries and fuel cells, versus other devices like capacitors and internal combustion (IC) engines. The goals for current hybrid and all electric vehicles are also illustrated. Adapted from (2).

The dashed lines in the above plot indicate discharge rates, where very short discharge times are to the lower right. These types of plots are very useful for illustrating the different realms of applicability for many devices. Note the units on the axes of the above plots, as both energy and power have been normalized by the weight of the device. The specific energy of an electrochemical device is calculated from the following equation(2).

$$\hat{E} = \frac{U_{pos} - U_{neg}}{\frac{1}{\hat{C}_{pos}} + \frac{1}{\hat{C}_{neg}}} \tag{0.10}$$

In the above equation U_{pos}, U_{neg}, $\hat{C}_{pos}$, and $\hat{C}_{neg}$ are the equilibrium potentials and specific capacities of the positive and negative electrodes, respectively. Specific power is

determined using the instantaneous voltage, which is again not equivalent to the open circuit potentials, due to the losses incurred in the cell during operation, such as the overpotential described earlier.

Battery Modeling

Equations that govern the interplay between the solid state regions in the electrodes, the liquid electrolyte, and the voltage across the cell are crucial to developing an accurate, physically representative model of the battery. In general, all governing equations are concerned with the conservation of charge and species, and their relation to the potential throughout the cell at each phase, and at the interfaces between each phase(5).

To begin, charge must be conserved in both phases, namely the electrolyte phase, and the solid phase. Because no charge is generated or consumed, the following must hold in the kth phase, where i_k is the current density in that phase.

$$\nabla \cdot i_k = 0 \tag{0.11}$$

In particular, for the solid phase, the current is carried via electrons, and therefore is related to the conductivity σ and the potential in the solid phase, ϕ_s, by Ohm's Law.

$$i_s = -\sigma \nabla \phi_s \tag{0.12}$$

Furthermore, in the electrolyte, the current is carried by the lithium ions as they move from one electrode to the other. This is influenced by the potential across the electrolyte, as well as by the concentration of lithium in the electrolyte, c_e.

$$i_e = -\kappa \nabla \phi_e - \kappa_D \nabla \left(\ln c_e \right) \tag{0.13}$$

Here, ϕ_e is the potential in the electrolyte, κ is the conductivity of the electrolyte, and κ_D is the proportionality constant that relates the movement of the charged species to diffusive processes. Substituting Equations (0.12) and (0.13) into (0.11), the following are obtained, respectively.

$$\nabla \cdot \left(\sigma \nabla \phi_s \right) = 0 \tag{0.14}$$

$$\nabla \cdot \left(\kappa \nabla \phi_e \right) + \nabla \cdot \left[\kappa_D \nabla \left(\ln c_e \right) \right] = 0 \tag{0.15}$$

Concerning species conservation, the following equation governs the rate of change of concentration to the flux of species at some point, assuming no generation or consumption of chemical species.

$$\frac{\partial c_k}{\partial t} = -\nabla \cdot N_k \tag{0.16}$$

In general, the flux of a species has three components, as shown below.

$$N_k = -D_k \nabla c_k + \frac{t_k}{zF} i_k + c_k v_k \tag{0.17}$$

The first component on the right hand side of the above equation represents diffusion, with a diffusion coefficient of D_k. The second term dictates how the electric field drives motion, otherwise known as migration, and is related to the charge on the species *z*, Faraday's Constant *F*, and the transference number t_k. The final component is the convective term, with velocity vector v_k. In the electrolyte and solid phases, convection is usually ignored. Additionally, in the solid phase, migration is additionally ignored,

yielding the characteristic Fick's Second Law, as shown below, when equation (0.17) is inserted into (0.16) under these assumptions.

$$\frac{\partial c_s}{\partial t} = \nabla \cdot \left(D_s \nabla c_s \right) \tag{0.18}$$

As will be shown later, thermal models for batteries are a very important consideration, especially with regards to performance and safety in operation. The governing equation for this additional phenomenon is an energy balance, as shown below.

$$\frac{\partial \left(\rho_k c_{p,k} T_k \right)}{\partial t} + v_k \cdot \nabla T_k = \nabla \cdot \left(\lambda_k \nabla T_k \right) + q \tag{0.19}$$

Here, ρ_k, $c_{p,k}$, q, and λ_k are the density, specific heat, heat generation term, and thermal conductivity, respectively. The heat generation term, q, is given by the following relation(6).

$$q = \sum_n I_n \left(U_n - T \frac{\partial U_n}{\partial T} \right) - IV + enthalpy\ of\ \ mixing + phase\ change \tag{0.20}$$

In the above, I_n is the reaction current at the nth electrode, and I is the applied current density, based on boundary conditions discussed below. The first term on the right hand side is associated with the enthalpy of charge transfer reactions, and the second is the electrical work produced by the battery, as V is the voltage across the cell. The last two terms represent those contributions from gradient development in the electrode, as well as those from phase transformations in the active material. Finally, now that governing equations for charge, species, and potential have been derived for each individual phase,

interface relations must now be considered. This is governed by the Butler-Volmer equation, as reproduced below.

$$i_n = i_{0,n}\left[e^{\frac{\alpha_{a,n}F}{RT}\eta_n} - e^{-\frac{\alpha_{c,n}F}{RT}\eta_n}\right] \tag{0.21}$$

In the above, i_n, is the current density, and $i_{0,n}$ is the exchange current density for the reaction at electrode n. Further, $\alpha_{a,n}$ and $\alpha_{c,n}$ are the anodic and cathodic transfer coefficients, and R is the universal gas constant. The overpotential for the reaction at the electrode, η_n, and again represents the amount of potential lost to bias the reaction direction, and is defined by the following equation.

$$\eta_n = \phi_s - \phi_e - U_n \tag{0.22}$$

Notice that this equation is slightly different than that shown in Equation (0.7), as this is the overpotential associated with the specific electrode n. In the above equation, U_n is the open circuit potential of the electrode, which can vary as a function of temperature, and is usually empirically fit to data as a function of the surface state of charge, as discussed later. Any temperature dependence can be modeled as a linear relation with the following equation, where $U_{n,ref}$ is the open circuit potential at some reference temperature.

$$U_n = U_{n,ref} + \left(T - T_{ref}\right)\frac{\partial U_n}{\partial T} \tag{0.23}$$

With these equations, all of the above relationships are now coupled together to form the basic framework for a battery model.

In the literature(5) several methods have been introduced to solve the above system, including Direct Numerical Simulation (DNS), and other volume averaged methods. In the DNS model, a discretization scheme is employed so that every computational cell in the domain represents a single phase, whether electrolyte or solid. Concerning a porous electrode, two discretiztions of this region are shown below, with different mesh sizes.

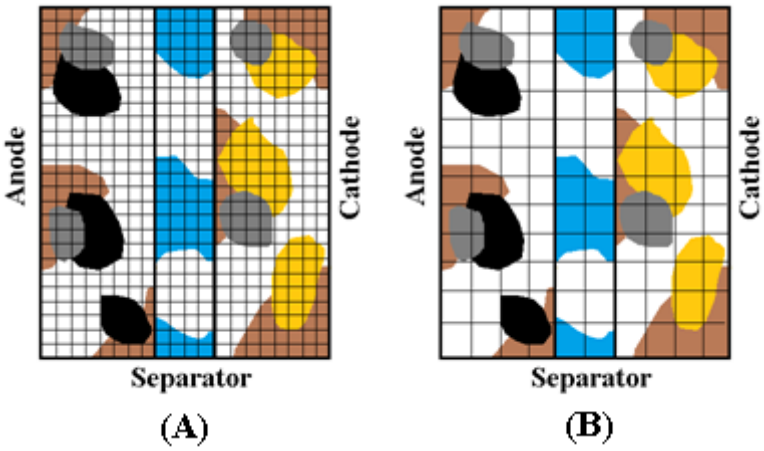

Figure 5. Different discretizations of insertion electrodes provide a particular level of resolution of the region. As discretization is coarsened, microstructure details are lost, and so volume averaged methods must be employed.

An obvious issue arises when using this particular method, as increasing the number of elements in the domain increases computational demand, but may offer increased accuracy. One of the primary advantages of this method, however, is that the governing equations discussed above can be directly applied to each appropriate cell as well as the appropriate coupling between computational cells.

As one coarsens the discretization, a volume averaged approach to the above equations must be taken as microstructure details are lost and governing equations can

no longer be directly applied(7-10). Because of this, effective properties must be determined, and are usually empirical in nature, such as by a Bruggeman relation. These are shown below, for some general transport property *P*.

$$P^{eff} = P\varepsilon^{\gamma} \tag{0.24}$$

$$P^{eff} = P\frac{\varepsilon}{\tau} \tag{0.25}$$

Here, ε and τ are the porosity and tortuosity, respectively, and γ is some empirical value. Using these volume averaged approach to modeling(11, 12), the above equations can now be rewritten in an appropriate form. Considering charge conservation in the solid phase, the following relation is used, where σ^{eff} can be determined from Equation (0.24) with γ=1(13-15).

$$\nabla \cdot \left(\sigma^{eff} \nabla \phi_s\right) - j^{Li} = 0 \tag{0.26}$$

For the charged species in the electrolyte phase, the following equations are used.

$$\nabla \cdot \left(\kappa^{eff} \nabla \phi_e\right) + \nabla \cdot \left[\kappa_D^{eff} \nabla \left(\ln c_e\right)\right] + j^{Li} = 0 \tag{0.27}$$

Here, κ_D^{eff} is expressed as a function of the transference number t_+^0, and mean molar activity coefficient $f_\pm$ (16, 17).

$$\kappa_D^{eff} = \frac{2RT\kappa^{eff}}{F}\left(t_+^0 - 1\right)\left(1 + \frac{d\ln f_\pm}{d\ln c_e}\right) \tag{0.28}$$

Additionally, κ^{eff} has been calculated using Equation (0.24) with γ=1.5. Analogously to the above, species are conserved in the electrolyte via the following relation.

$$\frac{\partial\left(\varepsilon_e c_e\right)}{\partial t} = \nabla \cdot \left(D_e^{eff} \nabla c_e\right) + \frac{1 - t_+^0}{F} j^{Li} - \frac{i_e \nabla t_+^0}{F} \tag{0.29}$$

In the above equation, the transference number is usually assumed constant(18), setting the last term on the right hand side of the above equation equal to zero. Within the solid phase, the species conservation equation is governed by the following equation.

$$\frac{\partial\left(\varepsilon_s c_s\right)}{\partial t} = \nabla \cdot \left(D_s^{eff} \nabla c_s\right) - \frac{j^{Li}}{F} \tag{0.30}$$

In both of the above equations, the effective diffusion coefficients have been determined using Equation (0.24) with γ=1.5. It is to be noted that, in all of the above volume averaged formulations, j^{Li} is the reaction current at the surface of the electrode *n*, multiplied by the specific interfacial specific area a_{sn}, or the reactive area per volume, of that electrode as shown below. Also note, that j^{Li} is zero in the separator.

$$j^{Li} = \begin{cases} j_n a_{s,n},\ n = a, c \\ 0,\ separator \end{cases} \tag{0.31}$$

Considering the energy balance in Equation (0.19), the thermophysical properties must now be altered to reflect the weight of the respective constituents. These are represented by the following equations.

$$\begin{aligned} \rho c_p &= \sum_k \varepsilon_k \rho_k c_{p,k} \\ \lambda &= \sum_k \varepsilon_k \lambda_k \end{aligned} \tag{0.32}$$

Further, the heat generation term, *q*, is also altered, and is now represented by the following.

$$q = a_{sn} j_n \left(\phi_s - \phi_e - U_n + T \frac{\partial U_n}{\partial T} \right) + \sigma^{eff} \nabla \phi_s \nabla \phi_s + \left(\kappa^{eff} \nabla \phi_e \nabla \phi_e + \kappa_D^{eff} \nabla \ln c_e \nabla \phi_e \right) \tag{0.33}$$

The electrode kinetics as governed by the Butler-Volmer Equation must also be changed, via the exchange current density, as shown below.

$$i_{0,n} = kc_e^{\alpha_{a,n}} \left(c_{s,\max} - c_{s,surf} \right)^{\alpha_{a,n}} c_{s,surf}^{\alpha_{c,n}} \tag{0.34}$$

In the above, $c_{s,surf}$ is the area averaged concentration of lithium at the interface between the solid and the electrolyte, and $c_{s,\max}$ is the maximum allowable concentration in the specific material for that electrode. The constant *k* is determined by concentrations and initial exchange current density.

During simulation of the cell, initial concentrations and temperatures in the solid and electrolyte phases are set to be uniform everywhere.

$$\left.\begin{aligned} c_e &= c_e^0 \\ c_s &= c_s^0 \\ T &= T^0 \end{aligned}\right\} at\ t = 0\ and\ x, y, z \geq 0 \tag{0.35}$$

Further, the following conditions are enforced on all boundaries, where *n* is the outward normal of the interface.

$$\frac{\partial c}{\partial n} = 0\ and\ \frac{\partial \phi_e}{\partial n} = 0\ at\ all\ boundaries \tag{0.36}$$

The conditions for the potential in the solid phase are dependent upon which tab the applied current density, *I*, is applied, but in general can be expressed as the following two equations.

$$\begin{aligned} -\sigma^{eff} \frac{\partial \phi_s}{\partial n} &= I\ at\ chosen\ tab \\ \frac{\partial \phi_s}{\partial n} &= 0\ at\ all\ other\ boundaries \end{aligned} \tag{0.37}$$

If the model does include thermal effects, Newton's Law of Cooling, reproduced below with a heat transfer coefficient h and ambient temperature T_{amb}, serves as the boundary condition.

$$-\lambda \frac{\partial T}{\partial n} = h\left(T - T_{amb}\right) \tag{0.38}$$

If thermal effects are considered, the temperature dependence of particular parameters may be modeled using the Arrhenius Equation, shown below, for some parameter P, activation energy E_{act}, and the reference value at some temperature, P_{ref}.

$$P = P_{ref} e^{-\left[\frac{E_{act}}{R}\left(\frac{1}{T_{ref}} - \frac{1}{T}\right)\right]} \tag{0.39}$$

To couple the solid state diffusion behavior to the rest of the cell, the structure of the electrode can be approximated using a pseudo-2D approach, like that shown in Figure 6.

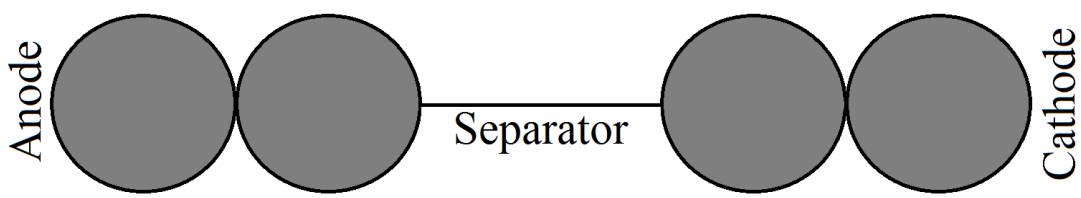

Figure 6. The insertion electrodes are approximated as being composed of small spheres of radius R_s.

This model was developed by Newman and coworkers in the early 1990s(1). In this model, the insertion electrodes are comprised of small spheres of radius R_s where, throughout the volume of each sphere, there is a superposition of electrolyte and solid phases. The figure below demonstrates these aspects of the model effort for an insertion electrode.

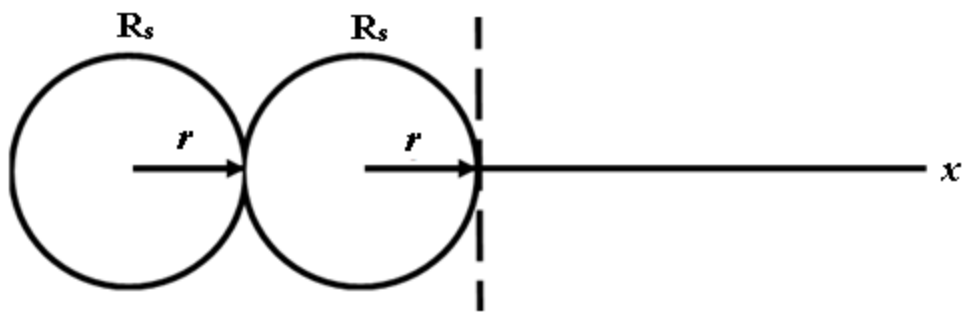

Figure 7. Doyle and Newman employ a pseudo-2D in the one-dimensional *x* direction and the pseudo-two-dimensional *r* direction.

In this figure it can be seen that there are the active particles as previously discussed, with two coordinate systems. This particular model employed only one dimension in the electrolyte, labeled *x*, and the coordinate *r* within the sphere. Thus the previously mentioned volume average equations are solved in the single *x* dimension, and the solid state diffusion problem is solved in spherical coordinates, as shown below.

$$\frac{\partial c_s}{\partial t} = D_s \left[\frac{\partial^2 c_s}{\partial r^2} + \frac{2}{r}\frac{\partial c_s}{\partial r} \right] \tag{0.40}$$

Assuming symmetry at the core of the spherical particles, the following boundary conditions apply, where the coupling to Butler-Volmer kinetics is accomplished by the condition at the surface of the particle.

$$\left.\frac{\partial c_s}{\partial r}\right|_{r=0} = 0 \; and \; -D_s \left.\frac{\partial c_s}{\partial r}\right|_{r=R_s} = \frac{j^{Li}}{a_s F} \tag{0.41}$$

One last simplification to the above model is to assume that the entirety of each electrode can be represented as a single particle with equivalent surface area(19-21).

This model assumes that transport within the electrolyte can be neglected, and solid state diffusion is the dominant process.

As is clear from the above discussion, a decrease in computational cost is accompanied by an increase in difficulty with respect to governing equations and parameters. This is best represented by the following figure.

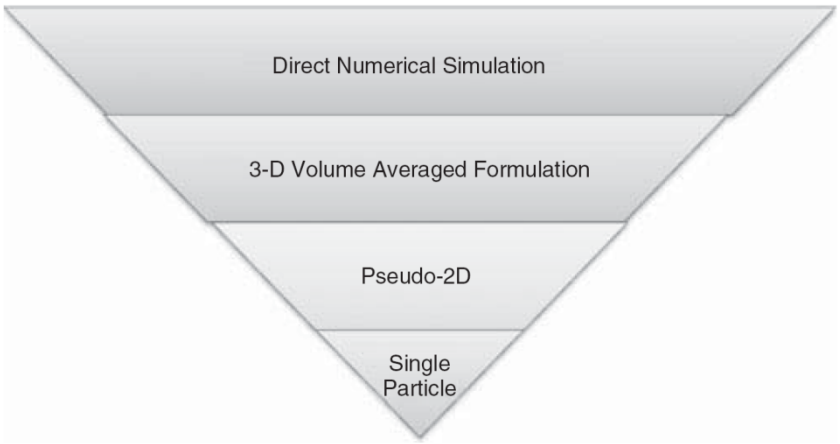

Figure 8. Inverted pyramid scheme illustrates decreasing computational difficulty but increasing model complexity due to volume averaged quantities. (Adapted from (5))

As one moves down the pyramid, there is a decrease of computational difficulty as the discretization process is coarsened, which, however, results in larger model complexity due to volume averaged quantities.

Solid State Diffusion Related Model Simplifications and Expansions

In solving the model previously discussed, Doyle and Newman employ the use of a Duhamel superposition integral to solve the solid state diffusion equation for the spheres. It has been found that in carrying out these computations a severe time cost is incurred, as the solution to every time step must remain in computer memory and accessed during every calculation. Others have developed less demanding methods to approximate this equation and thus reduce the time taken for computation. In general, the equation to be solved is Fick's law of diffusion, given below, in spherical coordinates.

$$\frac{\partial c_s}{\partial t} = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 D_s \frac{\partial c_s}{\partial r}\right) \tag{0.42}$$

In the above equation, the diffusion coefficient, D_s, may be a function of concentration, so it has been left in the differential. In some preliminary first approximations for batteries and fuel cells, Wang *et al.* implemented a diffusion length method to solve the above equation(22, 23), and did show accurate results when the model was compared against the literature. Further, Liu(24) employs a pseudo-steady-state method (PSS), and is able to obtain accurate results for several boundary conditions, including a sinusoidal flux at the surface of the particle. Another possibility for simplifying the above problem is to assume a particular solution for the concentration profile in the sphere. One such study was performed by Subramanian and coworkers(25, 26), where they employ either employ a two or three parameter polynomial model within the sphere as shown in Equations (0.43) and (0.44).

$$c(r,t) = a(t) + b(t)\left(\frac{r^2}{R_p^2}\right) \tag{0.43}$$

$$c(r,t) = a(t) + b(t)\left(\frac{r^2}{R_p^2}\right) + d(t)\left(\frac{r^4}{R_p^4}\right) \tag{0.44}$$

The above parameters *a*, *b*, *c*, and *d* are solved for using average bulk and surface concentrations, as well as volume-average concentration flux. Reasonable accuracy was obtained using these models for long times, however it was noted that special care would need to be taken when short times or pulsed currents were considered. Further expansions on the above approximations can be made, and are discussed in the References(27), and the uses of this model are very widespread, as can be viewed in the literature(19). A comparison study between the polynomial approximation, the PSS method, and a corrected diffusion length method was completed by Zhang *et al.*(28). Compared against the Duhamel Superposition Integral for a porous electrode, it was concluded that the higher order polynomial method or the PSS method should first be considered in approximate methods for porous electrodes. Smith and Wang employed a finite element method within the sphere to approximate the solution in their investigation of a cell to be used in a hybrid electric vehicle (HEV)(13). As illustrated in their work, the equation in spherical coordinates is first transformed to planar coordinates, where the elements are first established. This discretized system is then transformed back into spherical coordinates and represented in a state space form, where it is then rewritten as a transfer function and discretized in time. The results found indicate good accuracy under certain prescribed conditions, and studies are completed against the Duhamel

superposition integral, as well as the approximation from Reference(22). The authors in Reference (29) compare and contrast the above models, and investigate two additional methods for solving the solid phase diffusion problem in the spherical particles. One employs an eigenfunction based Galerkin collocation, and the other uses a finite difference method with uneven node spacing. In particular, for the latter case, the objective was to optimize the node spacing for the solution procedure, which was neglected in the work by Smith and Wang. The finite difference method is also developed below. In the literature a second order Taylor Series expansion of the equations were made, with variable step size. It was found that unequal node spacing can increase computational efficiency as there is a great computational cost for each additional node. To compare the performance of the two above methods with the full numerical solution, simulations were carried out at rates of 5C and 10C. Excellent agreement between the models was shown, and a very large reduction in computational time was realized. Additionally, it was also mentioned that in nonlinear cases, where the diffusion coefficient is a function of concentration, the finite difference method can be optimized in terms of node spacing to accommodate the higher complexity while still keeping the computational cost low. Employing more advanced methods for the single particle model, the authors in Reference (30) investigate the finite difference in addition to finite element and differential quadrature methodologies, and compare their results against the polynomial approximations. It was determined that the best results were obtained with the polynomial differential quadrature discretization scheme, and that

polynomial approximations perform very poorly when the applied currents were pulsated.

Inclusion of other Forces

Another important expansion on solid state diffusion is the inclusion of chemical potential within the activity coefficient, as well as the electrostatic forces that act during intercalation of the lithium ions. Portnyagin studies these other, non-diffusive, driving forces (31). The main portion of this work focused on cylindrical particles, instead of spherical ones, where the lithium intercalates along the radial direction only. In this model, the following governing equation and non-dimensional variables are used.

$$\frac{\partial y}{\partial \tau} = \frac{1}{R}\frac{\partial}{\partial R}\left(Rf\,\frac{\partial y}{\partial R}\right) \tag{0.45}$$

$$\tau = \frac{tD_s}{R_s^2} \quad y = \frac{c_s}{c_{s,max}} \quad R = \frac{r}{R_s} \tag{0.46}$$

Following the notation used in the literature, $c_{s,max}$, c_s, and R_s are the maximum allowable lithium concentration within the active material, the current lithium concentration, and the radius of the particle, respectively. From Equation (0.45) can be derived two forms of the solid state diffusion, one where the activity coefficient, f, is held constant at 1, or when Equation (0.47) is used.

$$f = \left(1 + \frac{d\ln\left(\gamma_+\right)}{dy}\right) = 1 + \sum_{s=2}^{7} \frac{\Omega_s}{RT} s\left(s-1\right)\left(y^{s-1} - y^s\right) \tag{0.47}$$

In the above, Ω_s are the parameters that described the deviation from ideal behavior due to ion-ion interaction. In addition to activity effects, migration due to electrostatic effects can be included with a different formulation of Equation (0.45), as shown below.

$$\frac{\partial y}{\partial t} = \frac{1}{R}\frac{\partial}{\partial R}\left(R\frac{D_s}{R_s^2}\frac{\partial y}{\partial R}\right) - \frac{1}{Fc_{s,max}} div(\sigma E) \tag{0.48}$$

In the above, σ is defined as follows,

$$\sigma = yc_{s,max}\frac{N_a D_s e^2}{kT} \tag{0.49}$$

where k, N_a, D_s, e, and T are Boltzmann's Constant, Avogadro's Number, the elementary charge, and temperature, respectively. Furthermore, the divergence of the electric field, E, is given by Equation (0.50).

$$divE = \frac{2}{R_s\sigma_{ef,1}} j_n^+ - \delta\frac{Fc_{s,max}}{\varepsilon_0}\left(y_{\text{avr}} - y\right) \tag{0.50}$$

Here, δ is the delocalization parameter, $\sigma_{ef,1}$ is the effective conductivity in the solid phase, and ε_0 is the dielectric constant. For cylindrical particles, the following equation is additionally used to determine the non-dimensional average concentration, y_{avr}.

$$y_{\text{avr}} = 2\int_0^1 yRdR \tag{0.51}$$

Using the other equations contained in the literature, four simulations, using different permutations of the above two effects, were solved at galvanostatic conditions. In every

case it was found that those models that incorporated the electrostatic components reflected higher battery capacitance and function time.

One important factor in the operation of lithium ion batteries is the stress that is generated as ions move in and out of insertion type electrodes. Such stresses can cause fragmentation of active material which leads to losses in capacity and battery life. Many studies have been performed on this particular issue, such as that performed by Christensen and Newman(32-34). Here, pressure diffusion is included in the transport equation due to stress formation, so that the flux equation becomes that shown below.

$$N_{LiS} = x_{LiS}\left(N_{LiS} + N_S\right) - cD_{LiS,S}\left[\alpha_{LiS}\frac{\partial x_{LiS}}{\partial r} + \frac{x_{LiS}}{RT}\left(\overline{V}_{LiS} - \frac{M_{LiS}}{\rho}\right)\frac{\partial p}{\partial r}\right] \qquad (0.52)$$

In the above, N_i is the flux of either occupied sites LiS, or unoccupied sites, S, and α_{LiS}, is a thermodynamic factor which influences the diffusivity of the lithium ions. Further, one will notice the influence of the local pressure, p, through $\overline{V}_{LiS}$, M_{LiS}, and ρ, which are the partial molar volume and molar mass of occupied sites, and the total density, respectively. Important conclusions drawn indicate that stresses will increase in the materials at high charge rates, and are also linked to particle size where fracture can be reduced with reduction in particle size. Cheng and Verbrugge(35, 36) approached stress as analogous to that generated during thermal diffusion, where the spherical particle is treated as an isotropic linear-elastic solid. Using this assumption, the relations between stress, σ, and strain, ε, in the radial and tangential directions r and θ can be expressed using the following equations.

$$\varepsilon_r = \frac{1}{E}\left(\sigma_r - 2\nu\sigma_\theta\right) + \frac{1}{3}\Omega C \qquad (0.53)$$

$$\varepsilon_\theta = \frac{1}{E}\left[\left(1-\nu\right)\sigma_\theta - \nu\sigma_r\right] + \frac{1}{3}\Omega C \tag{0.54}$$

In the above, E, C, ν, and Ω are Young's Modulus, the local concentration, Poisson's Ratio, and the partial molar volume of lithium, respectively. Their results indicate similar conclusions as before, specifically concerning the particle size, where the particle radius should be reduced to the nanometer range.

Finally, one more expansion of the solid state diffusion model is completed by White *et al.*(37), where energy equations are included to solve for the thermal behavior of the cell during operation. The approach used here is actually the single particle model as discussed previously, where the entire electrode is viewed as one particle. In this particular work, the focus is on the solid state diffusion processes only, and thus a uniform current density is assumed across the electrode. This assumption is only accurate for low to modest discharge rates. The potential drop in the electrolyte, R_{cell}, is modeled as a temperature dependent resistor that is based upon fits to data collected during experiment. The energy balance used to determine thermal variation in the cell is as shown below, and is similar to that discussed in Equations (0.19) and (0.20).

$$\rho v C_p \frac{dT}{dt} = IT\left[\frac{\partial U_p}{\partial T} - \frac{\partial U_n}{\partial T}\right] + I\left(\eta_p - \eta_n + IR_{cell}\right) - q \tag{0.55}$$

Here, ρ, v, and C_p are the density, volume, and specific heat capacity of the cell. Further, U_i is the open circuit potential of electrode i, and is determined by experimental fits to data as a function of the surface concentration, and q is the heat lost by the cell to the surroundings, modeled by Newton's Law of Cooling. The second term

in Equation (0.55) is the irreversible heat generated by electrode polarization. Using this thermal model for single particle electrodes, appropriate simulations were carried out and parameters were adjusted to fit experimentally acquired data. Further simulation showed good agreement with experimental data, as well as with another model developed by Kumaresan *et al.*(38).

Experimental Determination of Material Parameters

Computational models, while extremely powerful tools, inherently rely upon specific material characterization as inputs. While the literature on experimental methods for determining the behaviors of various materials to be used in battery modeling is vast, some significant work was performed by Tarascon and coworkers. Their studies focused on cells employing a $Li_xMn_2O_4$ cathode with a carbon anode(39-42). In their work, the cyclic behavior of the cell was studied under different temperatures, and they optimized the cell performance by using different electrolytes. It was found that cycle life could be maintained even at high temperatures, and that the cell could be safely discharged to 0V. Other electrolytes were further investigated and developed to continue to improve the performance of the system. In this investigation, experimental techniques were used to classify many important modeling parameters, including diffusion coefficients and capacities of materials.

A novel investigation was completed by Verbrugge and Koch (43). They additionally completed a follow-up study using particular models to isolate desired physiochemical properties (44), such as the open circuit potential function *U*. In these

two studies, the properties of a single carbon fiber electrode were isolated, and later used in a new mathematical model for intercalation into such a fiber. The main advantage of this study was that by using a single carbon fiber the effects of other additives and components like conductive binder and current collectors were not present. In this way the carbon itself and its respective properties could be directly identified. In this mathematical formulation, intercalation only occurs in the radial direction of a carbon fiber, modeled as a cylinder, and the open circuit potential needed to be determined as a function of the degree of intercalation, and is expressed in Equation (0.56).

$$FU = FU^{\Theta} + RTln\frac{\Theta_S}{\Theta_I} + RTln\frac{\gamma_S}{\gamma_I} \tag{0.56}$$

In the above, the subscripts 'S' and 'I' stand for those properties relating to the vacant site available for reaction and the intercalating species, respectively. Additionally, Θ and γ are the fractional occupancy and activity coefficients, respectively. The standard cell potential, U^{θ}, is given below.

$$FU^{\Theta} = \mu_{Li}^{0} + \mu_{S}^{0} - \mu_{I}^{\Theta} \tag{0.57}$$

The reference states chosen for this model require the following relations between the activity coefficients and the fractional occupancies.

$$\begin{aligned} \gamma_I &\rightarrow 1 \; as \; \Theta_I \rightarrow 0 \\ \gamma_S &\rightarrow 1 \; as \; \Theta_S \rightarrow 1 \end{aligned} \tag{0.58}$$

To solve for the above activity coefficients, a binary interaction equation is used to related I-I interactions with non-vanishing free energies, G^E, that cause deviation from ideal behavior. This is expressed as a series, shown in Equation (0.59).

$$G^E = \sum_{k=2}^{N} \Omega_k \Theta_{\mathrm{I}}^{\mathrm{k}} \tag{0.59}$$

In this series, Ω_k is the self interaction coefficient that characterizes I-I interactions and $\Theta_{\mathrm{I}}^{\mathrm{k}}$ is the frequency of such interaction. This series approaches a finite amount as the probability of larger scale interactions decreases with a larger number of interacting species. Using the above equation, the activity coefficients are given by the following.

$$RTln\gamma_{\mathrm{I}} = \frac{\partial}{\partial n_{\mathrm{I}}}\left(nG^E\right) = \sum_{k=2}^{N} \Omega_k \left[\mathrm{k}\Theta_{\mathrm{I}}^{\mathrm{k}-1} + \left(1-\mathrm{k}\right)\Theta_{\mathrm{I}}^{\mathrm{k}}\right] \tag{0.60}$$

$$RTln\gamma_{\mathrm{S}} = \sum_{k=2}^{N} \Omega_k \left(1-\mathrm{k}\right)\Theta_{\mathrm{I}}^{\mathrm{k}} \tag{0.61}$$

Substituting these quantities into Equation (0.56) expresses the open circuit potential.

$$FU = FU^{\theta} + RTln\frac{1-\Theta_{\mathrm{I}}}{\Theta_{\mathrm{I}}} - \sum_{k=2}^{N} \Omega_k \Theta_{\mathrm{I}}^{\mathrm{k}-1} \tag{0.62}$$

To determine U^{θ} and Ω_k, the authors used their previous experimental work with a least square polynomial regression fitting routine. Having obtained the open circuit potential function, the authors moved to test the validity of the model obtained within the carbon microfiber. Good agreement was made between experimental and theoretical data which can be reviewed in the literature. Another interesting application of this model was to determine the sensitivity of the diffusion coefficient as a function of concentration. Two

trials were conducted with a constant diffusion coefficient, and the theoretical results were found to be very different than those resulting from experiment.

Insertion Electrodes as Packed Beds

In the above discussion, the psedo-2D model has been employed in numerous studies to approximate the mesoscopic nature of insertion type electrodes. This is particularly well suited for these electrodes, as they may be considered analogous to packed bed rectors(2). They are traditionally assembled by mixing small active particles with conductive filler and binder as stated previously, giving them the properties of a packed bed. This mix is then applied to current collectors to form the planar, insertion type electrodes. Many materials have been investigated to employ here, first beginning with the carbon and cobalt oxide system previously described. Disadvantages of these materials included higher cost, and possibilities for safety problems as they became fire hazards if overcharged, overheated, or overdischarged. Additionally, new materials are being sought that are more environmentally benign. These included $LiMnO_2$, which eventually proved to be ineffective due to phase transformations in the crystal lattice, as well as mixes of Mn, Ni, and Co. One of the latest materials to be investigated is $LiFePO_4$, which is environmentally friendly and low cost, and has already been used in commercial electronics. However, this material does suffer from low electronic conductivity, and so other dopants have been investigated to improve performance.

3D Architectures

While insertion type electrodes have attracted a large amount of focus in terms of modeling and experimentation, some of the next generation of batteries may consist of electrodes that are no longer suited to be considered in the above manner, as microstructure details will become important. Recently, it has been realized that improved use of available cell area may greatly increase the performance of battery systems through the use of 3D electrode architectures, particularly for use with microelectromechanical devices (MEMS). As these devices are made smaller, the scaling of a 2D electrodes causes drops in power, and are thus unable to meet necessary requirements. As explained in the literature(45, 46), 3D electrodes intend to take advantage of the space available in the third dimension, as opposed to confining the geometry to a 2D, or planar electrode. This can be best understood by considering the following figure, which compares these two designs.

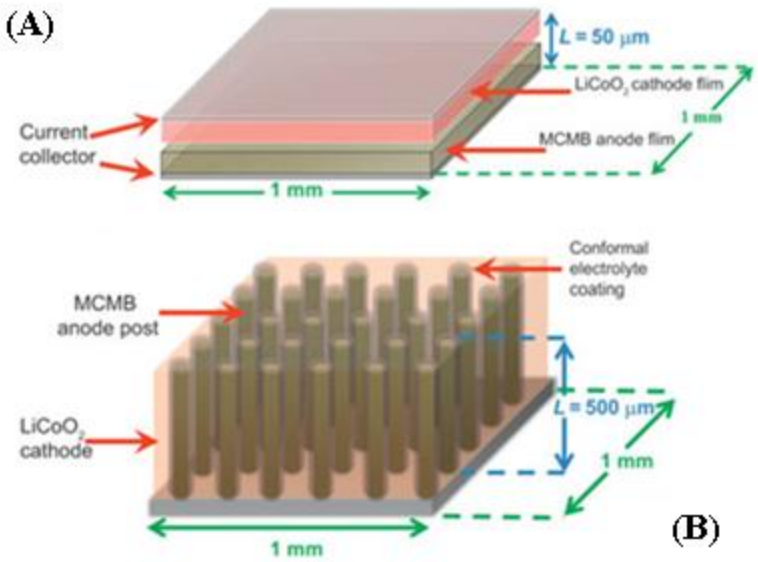

Figure 9. (A) A 2D planar electrode does not take advantage of the full area available. (B) A 3D structure using the 'height' dimension makes far better use of available cell area.(Adapted from (46))

Here, 2D and 3D electrode designs are confined to equivalent cell footprints. However, it is clear that greater use is made of available space by construction of anode posts or rods. With the advent of 3D microstructures, there are numerous possibilities for architecture, as shown below in Figure 10.

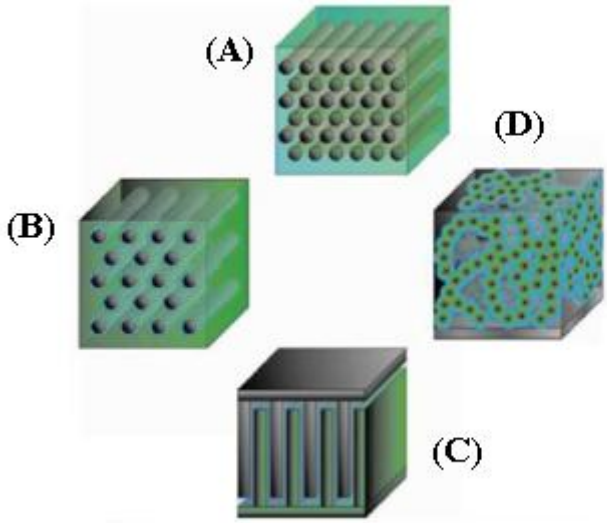

Figure 10. Four possible designs for 3D architectures, including (A) interdigitated rod arrays, (B) a continuous rod array, (C) interdigitated plates, and (D) sponge like geometry.(Adapted from (47))

Here is shown four of many microstructures that are probable designs for 3D electrodes. One of the more intuitive designs is the interdigitated plates (C), where the electrodes are meshed in the above configuration. However, there is no reason for both electrodes to have such a defined 'structure.' An example of this is illustrated by the continuous electrode (B), where a rod array serves as one electrode, is coated with a thin film of electrolyte, and the second electrode fills the remainder volume. This approach is also employed with the sponge geometry (D), where now one electrode is in a random configuration. Finally, a more challenging design is the interdigitated rod electrodes (A), as with this design there are many different possibilities, including staggered and aligned rod arrays, as well as the spacing between each rod type.

In the above figure it is clear that the 3D architecture offers new design possibilities for battery design, and can offer improved performance. This is possible by

increased amounts of active material with large surface area, and intimacy between the two electrodes. This latter point is critical, as short diffusion lengths reduce the ohmic and other potential drops that occur during ion transport through the electrolyte. Other advantages are a high power density, due to the low losses incurred by the diffusion path lengths, and high effective mass utilization. These advantages, however, are not without physical conditions that must be considered carefully in design. For instance, with respect to separator thickness, there is a lower limit on the distance between the two electrodes, as electron tunneling may occur if the distance of separation is on the order of 1nm. This in essence would provide a short for the battery and render it useless. Also, even if the thickness is great enough to prevent electron shorting through tunneling, the impact of electric fields on transport in the electrolyte become significant, as the double layers originating from each electrode may encounter one another. Ion transport through such conditions are not well known, and appropriate relations including transport, electrostatics, and statistical mechanics will need to be employed(48, 49). Further, limitless extension into the third dimension to increase capacity is ultimately limited by the electronic conductivity of the electrode material, as ohmic drops within the electrodes themselves may outweigh the benefits of increased capacity, and this is discussed later. Also, 2D designs will always have a greater energy to volume ratio, as the electrolyte does not constitute a great amount of the cell volume, unlike in 3D batteries.

The approaches to manufacturing 3D architectures are wide, but are capable of producing almost any geometry imaginable. These methods include lithography,

chemical vapor deposition (CVD), electroless deposition, and electrodeposition. While a functional 3D battery has yet to be assembled, numerous studies have laid significant ground work towards such a goal. In particular, for creating the 'sponge geometry' in Figure 10, aerogels and ambigels can be employed(50). What is advantageous about this method is that the pores developed are through connected, meaning that developed structures are in good contact, which is key for transport. The interdigitated rod array has already been manufactured via micromachining methods, and the resulting carbon rods showed good reversibility(51). To create rods of different materials, silicon molds can be formed using photolithography and other methods, so that any powder may be employed. It is important to note here that during the fabrication process, errors may be incurred when a specific geometry is targeted. It therefore becomes important to understand stochastically the impact of such small perturbations.

To accompany the above experimental work in creating these architectures, computational and mathematical tools will need to play a key role in refining the design parameter space. Some studies have been completed on these types of structures using advanced computational tools, particularly finite element analysis. The authors in (52) made use of this approach to study the uniformity of the current distribution in several 3D microstructure designs. It has been mentioned that obtaining a uniform current distribution is key for battery performance when using 3D electrodes, especially in the utilization of active material. The geometries considered focused primarily around rod arrays where both the anode and cathode were in rod form, as shown in the interdigitated electrodes from Figure 10. Alternative designs were made by having parallel rows of

anode and cathode rods, or with an alternating pattern in each row. Further, a study was conducted where an anode rod was surrounded by six cathode nearest neighbors in a hexagonal fashion, as well as rods with a triangular cross-section. It was concluded that uniform current densities were very difficult to obtain, and can vary greatly from one geometry to another(52). The most uniform current densities were obtained with the hexagonal arrangement, but for the anode only. It was underlined, however, that this might be advantageous should one electrode material require a more uniform current density than the other, or if there need not be an equal number of both. Significant contributions have also been made by Zadin *et al.*, particularly with the interdigitated plate structures, otherwise known as the 'trench' design, as shown in Figure 10 (53, 54). Focused around the height of the plates, as well as the electrical conductivity for electron transport, the results indicated that the most favorable range for conductivity was such that the difference between the two electrodes should be no more than one order of magnitude. Additionally, it was found that even in this optimized regime, solid state transport was the limiting parameter for the battery as a whole, and that tuning the plate height exhibited a limited effect.

It is clear from the above discussion that computational tools will play a key role in identifying and gauging the performance of 3D electrode architectures. This is particularly true as these systems are currently in nascent stages, and the design space, as previously mentioned, is still extremely large. The aim of the current work is to aid in this process by using first principles approximations to gauge the performance of several types of 3D electrode architectures. The results obtained from this study will help to

isolate advantageous geometries, and develop strategies for studying system behavior that can be used in later, more developed models and tools.

CHAPTER II

ELECTRODE ARCHITECTURE GENERATION AND CELL DESCRIPTION

The objective of this study was to use first principle approximations to quantify the performance of various 3D structures through discharge behavior, and the globally averaged parameters of bulk and surface concentration. This was accomplished by developing several codes in MATLAB® software to first generate targeted microstructures to serve as the anode morphology. These 3D architectures were similar to the aperiodic structures described in Figure 10, and the effects of particle size distribution, particle shape, and overall morphology were included. Having generated the desired structures, MFiX® software was used to solve the diffusion problem in two dimensions using a finite volume formulation. A single particle cathode was additionally simulated in MATLAB® using the *pdepe* package. Post processing codes in MATLAB® were developed to solve for voltage profiles from the obtained surface concentrations. Further, two models were studied where both electrodes were of 3D design. The first was an aperiodic geometry, generated by additional MATLAB® codes, and the second was an interdigitated plate design with equal surface to volume ratios as the aperiodic design. From the results obtained, microstructures with advantageous properties may be targeted for implementation in fuller, more developed models.

Microstructure Generation

To develop 3D architectures in a controlled manner, the current work used an approach similar to those using fractal methods (55-63). The fractal structure was represented by branch lengths, parent particles, and child particles as shown in Figure 11.

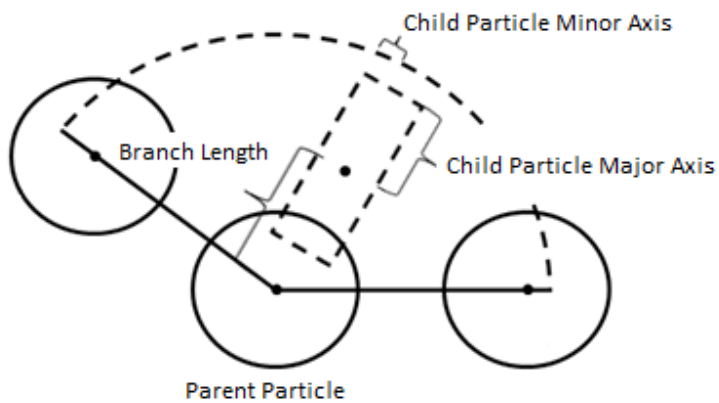

Figure 11. A cylindrical child particle is placed along the branch length only when the arc length swept between the parent particle's nearest neighbors is large enough to accommodate it.

Unlike previous efforts, a number of spawning particles was first specified by the user, and set as the 0th generation. Each particle then chose a random maximum number of branches to form, and each branch then chose a random particle type, whether spherical or cylindrical. The dimensions of that particle, or double the values of the major and minor axes as illustrated above, were based upon a normal distribution of values. One of these dimensions was placed along the branch direction, therefore setting the other perpendicular to it, if applicable, as spherical particles only required one dimension to be specified. The branch length, or the distance between the centers of the parent and child particle, was set to have a maximum value of the average of the parent and child

dimensions along the branch direction. A random percentage of that maximum was then chosen as the final branch length. This value then served as a radius about which a sweep was conducted to determine the parent particle's nearest neighbor particles. Based upon the location of those neighbors, an angle range was defined. An arc-length was then formed using this angle range, as shown by the dashed line in Figure 11, and the child particle was only allowed to be placed if this arc-length was larger than that particle's size by some criterion. The angle at which the particle was placed, relative to the coordinate system formed at the center of the parent particle, was then set to within a certain percentage of the midpoint of the angle range previously formed. This process was allowed to proceed until either a child particle could not be placed due to space restriction, or when the maximum allowable number of child generations was met. The coding for this procedure was completed in MATLAB® software, and is given in Appendix B. Further, after having created a base architecture, a code was developed to take this architecture and slightly perturb the values of the child angles and branch lengths, and is given in Appendix B. This procedure did not include initial spawning particles. This allowed for different realizations of one particular structure to be formed, and in doing so may lend insight to the impact of fabrication errors on particular microstructures in real cells. Typical output from the MATLAB codes are shown below, for a base structure in blue, and a perturbed realization in red.

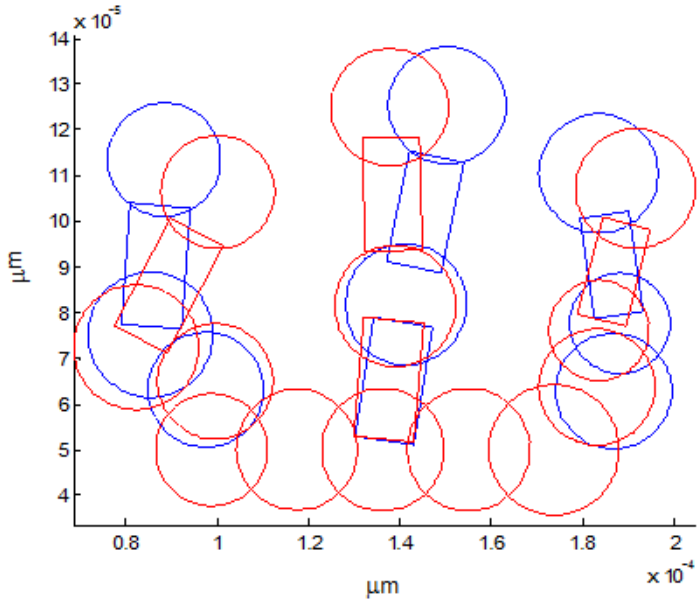

Figure 12. Typical output from the fractal geometry code developed in MATLAB®. A base structure in blue has been perturbed to yield a new realization in red.

Having completed generating a particular anode morphology, MFiX® software was used to solve the diffusion problem using the cut-cell option (64). MFiX® is a general purpose, computational code developed by the National Energy Technology Laboratory (NETL) capable of modeling various phenomena, including heat transfer, chemical reactions, and hydrodynamics(26). MFiX® software is unique in its ability to capture particular geometry using a Cartesian Grid by shaping the cells accordingly when they intersect with the problem geometry. This is illustrated in the following figure.

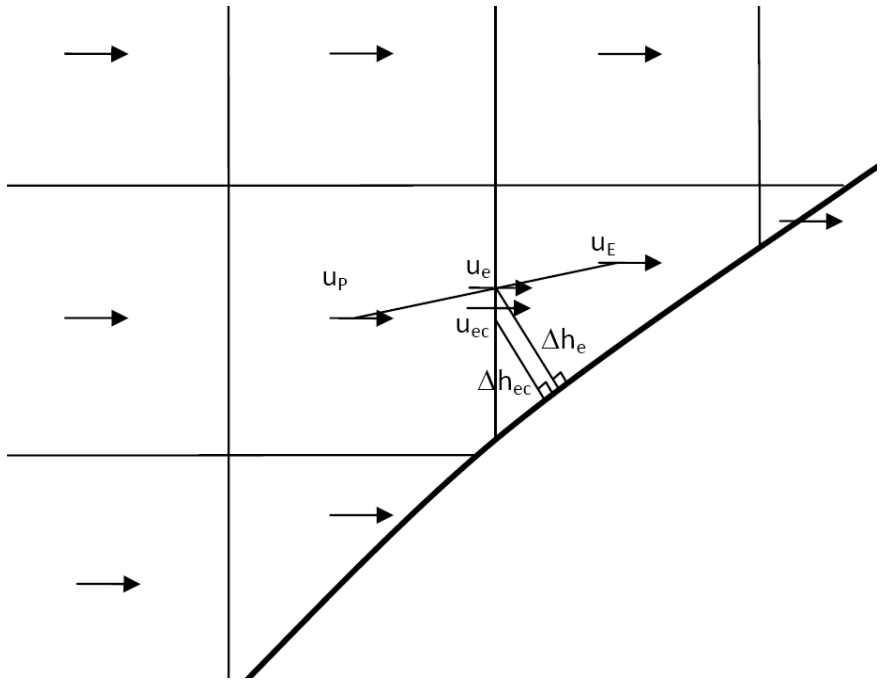

Figure 13. The cut-cell technique is used to shape the Cartesian Grid to the specified geometry. Here, the thick line intersects the Cartesian Grid, forming cut cells. The velocity component u_{ec} must be adjusted to account for the realignment of cell centers. Adapted from (64).

To generate the computational mesh using cut cells several procedures must take place. A search for cut cells is initially completed, and intersections points are calculated between the Cartesian Grid and the specified geometry. This is demonstrated in the following figure, and also underlines the importance of proper cell size, as only so many intersections will be considered by the mesh generation technique.

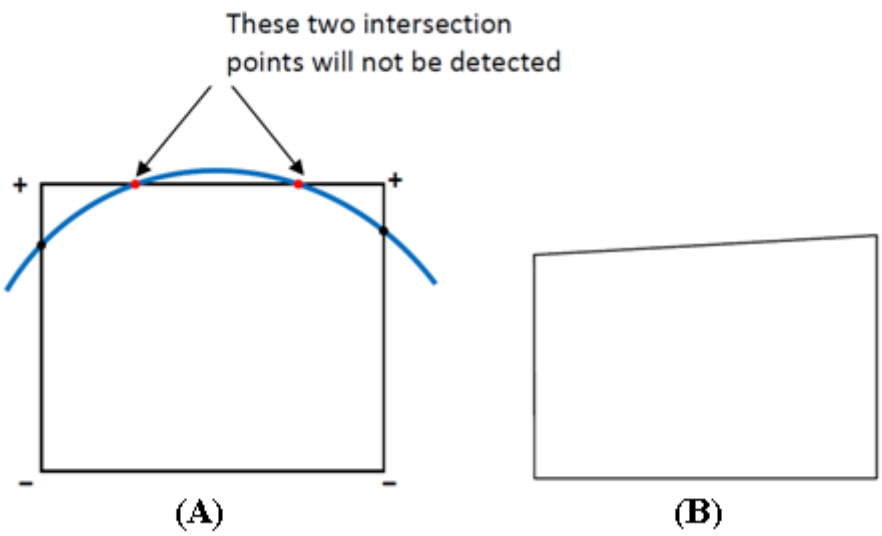

Figure 14. The intersections between the geometry defined (blue line in (A)) and the Cartesian Grid are determined to ultimately shape the resulting cut cell shown in (B). If improper cell size is used, some curvature aspects may be lost. Adapted from (64).

Once complete, the cell faces are computed similar to convex polygons and cell volumes are computed by splitting the cells into pyramids, computing their volume, and then adding them together. Because cut cells are generated, nodes and face centers are realigned, and so adjustments must be made to the quantities that are evaluated at those points. For example, as shown in Figure 13 and in the case of a no slip wall, the velocity on the east face of node *P* is not the same as the velocity component that is used for the east node. Therefore, a new component u_{ec} must be determined, and is approximated by using the ratio of the distances from the wall to the centers of the respective faces, as shown in the following equation.

$$u_{ec} = \frac{\Delta h_{ec}}{\Delta h_e} u_e \tag{0.63}$$

Other corresponding adjustments are made, and may be viewed in the literature(64).

Several studies have been completed to verify the cut cell technique, particularly within fluid dynamics and the study of fluidized beds (65, 66). The cut-cell technique is very flexible, as shown in the following figure, for generating almost any geometry imaginable.

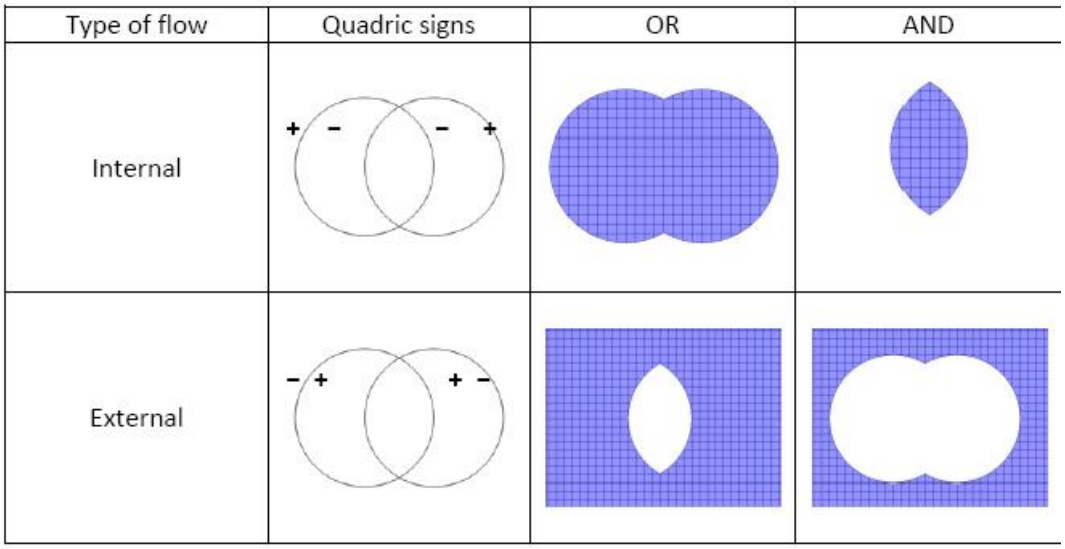

Figure 15. Depending on the type of flow and Boolean expression, the cut-cell technique is capable of capturing numerous geometrical shapes on a structured, Cartesian Grid. Adapted from (64).

Using the appropriate definition for the type of flow, and Boolean Expressions, the two circles in Figure 15 can be used to create very complicated geometry. Numerous, standard geometries, called quadrics, are available in MFiX for manipulation and combination to produce almost any geometry. Furthermore, quadrics can be grouped, and used with further Boolean expressions to form more complicated geometry, as shown in the following figure.

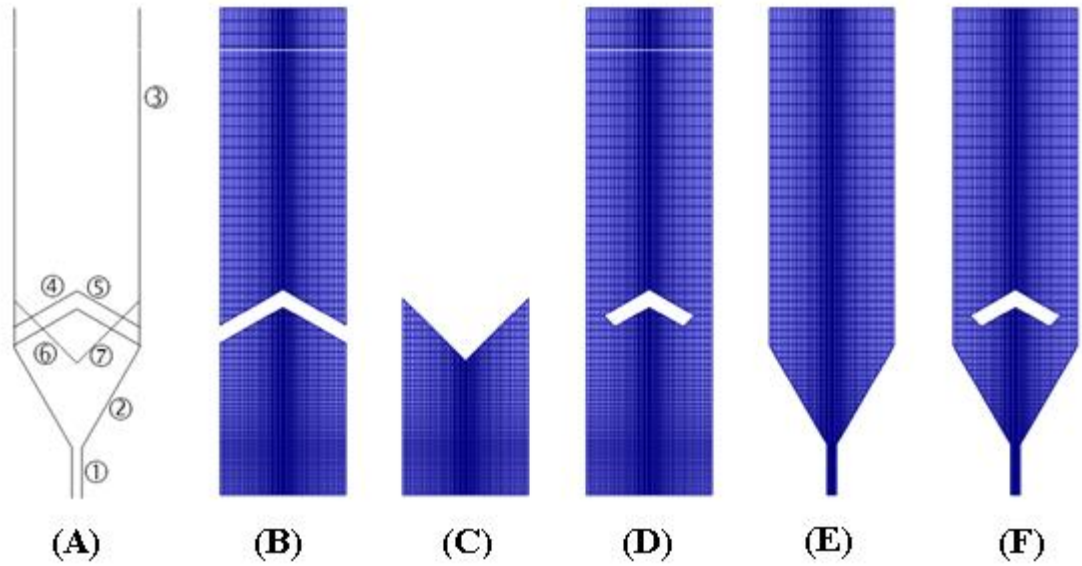

Figure 16. Geometries in the Cut Cell Technique in MFiX® can be described using quadrics. Here, several quadrics numbered in (A), as well as groups of quadrics (B) through (E), which are combined to form a spouted bed geometry with a stabilizer (F). Adapted from (64).

From the above discussion it is clear that MFiX®, especially with respect to the cut cell mesh generation technique, is ideal for this study, as simple shapes can be used to create very complicated geometries.

The scalar transport equation to be solved in this study is as shown below, where ε_m, ρ_m, and X_{mn} are the void fraction, density, and mass fraction of the nth species in the mth solid phase. Further, U_{mi}, D_{mn}, and R_{mn} are the velocity vector, diffusion coefficient, and rate of production due to chemical reaction, respectively(67).

$$\frac{\partial}{\partial t}\left(\varepsilon_m \rho_m X_{mn}\right) + \frac{\partial}{\partial x_i}\left(\varepsilon_m \rho_m U_{mi} X_{mn}\right) = \frac{\partial}{\partial x_i}\left(D_{mn}\frac{\partial X_{mn}}{\partial x_i}\right) + R_{mn} \qquad (0.64)$$

Using a finite volume formulation, the above differential equation is rewritten into an algebraic expression by integration over a control volume. Doing so on the transient

term on the left hand side over a control volume at the node *P* is approximated by the following expression.

$$\int \frac{\partial}{\partial t}\left(\varepsilon_m \rho_m \phi\right) dV \approx \left[\left(\varepsilon_m \rho_m \phi\right)_P - \left(\varepsilon_m \rho_m \phi\right)_P^o\right] \frac{\Delta V}{\Delta t} \tag{0.65}$$

Here, the superscript *o* indicates a previous time step, Δt is the discrete time step, and ΔV is the discretized volume. The next term on the left hand side of Equation (0.64) is the convective term, and is computed by the following equation.

$$\begin{aligned}
\int \frac{\partial}{\partial x_i}\left(\varepsilon_m \rho_m v_{mi} \phi\right) dV \approx & \\
& \left\{\xi_e \left(\varepsilon_m \rho_m \phi\right)_E + \bar{\xi}_e \left(\varepsilon_m \rho_m \phi\right)_P\right\} \left(u_m\right)_e A_e \\
& - \left\{\xi_w \left(\varepsilon_m \rho_m \phi\right)_E + \bar{\xi}_w \left(\varepsilon_m \rho_m \phi\right)_W\right\} \left(u_m\right)_w A_w \\
& + \left\{\xi_n \left(\varepsilon_m \rho_m \phi\right)_N + \bar{\xi}_n \left(\varepsilon_m \rho_m \phi\right)_P\right\} \left(v_m\right)_n A_n \\
& - \left\{\xi_s \left(\varepsilon_m \rho_m \phi\right)_P + \bar{\xi}_s \left(\varepsilon_m \rho_m \phi\right)_S\right\} \left(v_m\right)_s A_s \\
& + \left\{\xi_t \left(\varepsilon_m \rho_m \phi\right)_T + \bar{\xi}_t \left(\varepsilon_m \rho_m \phi\right)_P\right\} \left(w_m\right)_t A_t \\
& - \left\{\xi_b \left(\varepsilon_m \rho_m \phi\right)_P + \bar{\xi}_b \left(\varepsilon_m \rho_m \phi\right)_B\right\} \left(w_m\right)_b A_b
\end{aligned} \tag{0.66}$$

In the above equation, the standard notation for node locations (in three dimensions) have been used, where *N, E, S, W, T,* and *B,* represent nodes at the north, east, south, west, top, and bottom directions relative to the node at *P*. Lowercase letters indicate the faces of the control volume at node *P*, and *u, v,* and *w* are the three velocity components. Notice that the above equation is the computation of weighted fluxes at the faces of the node *P* with area *A*. In order to calculate these weights, ξ_e and $\bar{\xi}_e$,MFiX® utilizes various downwind factors to improve solution accuracy for the convective terms, and is dependent upon the scheme chosen as can be reviewed in the literature(68). Of primary

interest to this study is the diffusive term on the right hand side of Equation (0.64). Here, integration over the control volume yields the following.

$$\begin{aligned}\int \frac{\partial}{\partial x_i}\left(\Gamma_\phi \frac{\partial \phi}{\partial x_i}\right) dV \approx & \left(\Gamma_\phi \frac{\partial \phi}{\partial x_i}\right)_e A_e - \left(\Gamma_\phi \frac{\partial \phi}{\partial x_i}\right)_w A_w \\ & + \left(\Gamma_\phi \frac{\partial \phi}{\partial x_i}\right)_n A_n - \left(\Gamma_\phi \frac{\partial \phi}{\partial x_i}\right)_s A_s \\ & + \left(\Gamma_\phi \frac{\partial \phi}{\partial x_i}\right)_t A_t - \left(\Gamma_\phi \frac{\partial \phi}{\partial x_i}\right)_b A_b\end{aligned} \tag{0.67}$$

In the above, the diffusive fluxes are approximated by finite differences, and an example for the east face is as shown below.

$$\left(\Gamma_\phi \frac{\partial \phi}{\partial x_i}\right)_e A_e \approx \left(\Gamma_\phi\right)_e \frac{\phi_E - \phi_P}{\Delta x_e} \tag{0.68}$$

The diffusion coefficients, Γ_ϕ, as implied by the above equation, are determined at the particular face, using a harmonic mean of the properties at the two nodes, as shown below.

$$\left(\Gamma_\phi\right)_e = \left[\frac{1-f_e}{\left(\Gamma_\phi\right)_P} + \frac{f_e}{\left(\Gamma_\phi\right)_E}\right]^{-1} = \frac{\left(\Gamma_\phi\right)_P \left(\Gamma_\phi\right)_E}{f_e\left(\Gamma_\phi\right)_P + \left(1-f_e\right)\left(\Gamma_\phi\right)_E}, \; where \;\; f_e = \frac{\Delta x_E}{\Delta x_P + \Delta x_E}$$

(0.69)

Finally, the source term is usually nonlinear in nature, and is first linearized as shown below.

$$R_\phi \approx \overline{R}_\phi - R_\phi^{'} \phi_P \tag{0.70}$$

Once this is complete, integration over the control volume is approximated by the following equation.

$$\int R_\phi dV \approx \overline{R}_\phi \Delta V - R_\phi^{'} \phi_P \Delta V \tag{0.71}$$

Combining Equations (0.65), (0.66), (0.67), and (0.71) yield an equation of the following form, with coefficients a_{nb}, where the index corresponds to either *N, E, S, W, T,* or *B.*

$$a_P \phi = \sum_{nb} a_{nb} \phi_{nb} + b \tag{0.72}$$

It is important to mention further that in order to avoid large fluctuations in ϕ, the continuity equation should be multiplied by ϕ and subtracted from Equation (0.72). The reason for this is outlined in the literature(69), and yields the following requirement on the coefficients in Equation (0.72).

$$a_P = \sum_{nb} a_{nb} \tag{0.73}$$

For modeling the single particle cathode, as described previously for the anode study, MATLAB® software was used utilizing the built in partial differential equation solver, *pdepe*. This method is specifically tailored to solve boundary and initial value problems of parabolic or elliptical type, as shown in the following equation.

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t} = x^{-m}\frac{\partial}{\partial x}\left(x^m f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right) + s\left(x,t,u,\frac{\partial u}{\partial x}\right) \tag{0.74}$$

Here, *x* and *t,* are the single spatial and temporal variables, with solution *u*. The functions *f* and *s* are the flux and source terms, and *c* is responsible for coupling multiple equations if required. The geometry under consideration, whether plate, cylindrical, or spherical is modified by the choice of the exponent *m* to 0, 1, or 2, respectively. The expected boundary conditions are an initial value throughout the domain at the initial time t_0 and

appropriate Neumann or Dirichlet conditions on the domain boundaries, *a* and *b*. These are given by Equations (0.75) and (0.76).

$$u(x,t_0) = u_0(x) \ at \ t = t_0 \tag{0.75}$$

$$p(x,t,u) + q(x,t)\, f\left(x,t,u,\frac{\partial u}{\partial x}\right) = 0 \ at \ a \le x \le b \tag{0.76}$$

In *pdepe*, MATLAB® first uses a specified number of mesh points to discretize the problem in space, which then yields a system of ordinary differential equations in time. Another built in solver, named *ode15s*, is then used to solve the integration problem over time, which can use either numerical differentiation formulas, or backward differentiation formulas, which are also known as Gear's Method(42). As indicated by the usage of this package, orders of accuracy are only in the low to medium range, and thus a fine mesh is needed. More information on these methods can be found in the references (70-72).

The diffusion problem in spherical coordinates is easily adaptable to the above forms and boundary conditions, and is reproduced below in different forms for clarity of comparison for a time period between t_0 to t_f, and over the radial coordinate from the center of the particle to the surface.

$$\begin{gathered} \frac{\partial c}{\partial t} = r^{-2}\frac{\partial}{\partial r}\left(r^2 D_s \frac{\partial c}{\partial r}\right),\ t_0 \le t \le t_f \ \ and \ \ 0 \le r \le R_s \\ \frac{\partial c}{\partial r} = 0 \ at \ r = 0 \\ \frac{\partial c}{\partial r} + \frac{j_{Li,i}}{D_s} = 0 \ at \ r = R_s \end{gathered} \tag{0.77}$$

Note that in the above boundary condition at the surface of the particle, the sign of the flux must be correctly specified.

Cell Setup

In order to study the effects of microstructure, appropriate, realistic cell properties needed to be obtained. White *et. al*(19, 37, 38, 73) have completed numerous studies on LiCoO2-mesocarbon microbead pouch cells, and in the process have well documented needed material properties, initial conditions, and equilibrium potential functions. Relevant quantities for this study are shown in the following table.

Table 1. Material properties and other quantities for modeling the lithium ion cell(37). *Properties that apply to both the anode and cathode.

Parameter	Symbol	Anode (i=n)	Cathode (i=p)	Units
Electrode Active Area	S_i	0.782	1.12	m^2
Solid State Diffusion Coefficient	$D_{s,i}$	3.90×10^{-14}	1.00×10^{-14}	m^2/s
Particle Radius	R_p	12.5	8.5	μm
Thermal Rate Constant	k	1.80×10^{-11}	6.70×10^{-11}	$m^{2.5}mol^{-0.5}s^{-1}$
Maximum Concentration	$c_{s,max}$	31833	51410	mol/m^3
Initial State of Charge	SOC	.7522	.4952	-
Operating Temperature	T	298*		K
Electrolyte Concentration	c_e	1000*		mol/m^3
1C Rate	C	1.656*		A

Using an approach similar to their investigation in to the thermal behavior of these cells, this study first assumed that at slow discharge rates Equation (0.78) could be used to determine a uniform flux on the electrode surface.

$$j_{Li,i} = \frac{i_{app,i}}{nF} \tag{0.78}$$

$$i_{app,i} = \frac{I_{app}}{S_i} \tag{0.79}$$

In the above, I_{app}, $i_{app,i}$, S_i and j_i are the applied current to the cell, current density over the ith electrode, active surface area of the ith electrode, and lithium flux on the ith electrode. Knowing the flux on the structure, and with knowledge of the surface concentrations of both the anode and cathode at each temporal moment, the Butler-Volmer Equation was used to solve for the overpotential, η_i, at the ith electrode at that moment. Below is shown the appropriate form of this equation for this study.

$$j_{Li,i} = k_i c_e^{\alpha_n} \left(c_{\max,i} - c_{surf,i} \right)^{\alpha_n} c_{surf,i}^{\alpha_p} \left[e^{\frac{\alpha_n F}{RT}\eta_i} - e^{\frac{\alpha_p F}{RT}\eta_i} \right] \tag{0.80}$$

In the above equation, c_e, and $c_{surf,i}$ are the electrolyte concentration, taken to be constant at the initial value of 1000 mol/m^3, and the surface concentration of the ith electrode, respectively. Additionally, α_c and α_a are the transfer coefficients(35) taken to be .5 for both the anode and cathode. Furthermore, as shown in the literature, the equilibrium potentials for the anode and cathode at any point in time can be determined from empirically derived functions of surface concentration, as shown below.

$$\begin{aligned} U_n = {} & .13966 + .68920e^{-49.20361x_n} + .41903e^{-254.40067x_n} - e^{49.97886x_n - 43.37888} - \\ & .028221\arctan(22.52300x_n - 3.65328) \\ & -.01308\arctan(28.34801x_n - 13.43960) \end{aligned} \tag{0.81}$$

$$\begin{aligned} U_p = {} & 4.04596 + e^{-42.30027x_p + 16.56714} - .04880\arctan(50.01833x_p - 26.48897) - \\ & .05447\arctan(18.99678x_p - 12.32362) - e^{78.24095x_p - 78.68074} \end{aligned} \tag{0.82}$$

In the above equations, x_i is the ratio of the surface concentration to the maximum intercalatable concentration for the particular material, also known as the surface state of charge SOC_{surf}, as given in Equation (0.83).

$$x_i = \frac{c_{surf,i}}{c_{\max,i}},\ i = n, p \tag{0.83}$$

Having knowledge of the equilibrium potential and overpotential values at each electrode, the voltage across the cell can be calculated using Equation (0.84), assuming there is no potential drop across the electrolyte.

$$V = \left|\left(U_p - U_n\right) + \left(\eta_p - \eta_n\right)\right| \tag{0.84}$$

Additionally, the capacity of the cell, in amp-hours, is calculated via the following equation.

$$Cap = \frac{1}{3600}\int I_{app}\,dt \tag{0.85}$$

CHAPTER III

RESULTS AND DISCUSSION

In order to satisfy the objective of this study, the behavior of various 3D microstructures, in terms of concentration distribution and global cell performance, needed to be studied in detail. MATLAB® software was used to generate desired microstructures under provided conditions, which included variations in particle shape and size, as well as overall structure morphology. MFiX® software, for the first time as per the author's knowledge, was used to simulate and extract quantities of interest from these morphologies. Post processing codes then utilized these data to provide insight into global cell behavior through discharge performance. The following discussion will first analyze those results concerning 3D anode structures that were discharged versus a single cathode particle. These results include the depth of discharge, relaxation phenomena, and recharge efficacy over cyclical conditions. Additionally, voltage and performance curves will also be considered. Two 3D cell geometries, equivalent in terms of volume to surface area ratio, will analogously be analyzed over discharge and relaxation to compare their respective performances. Lastly, the results of a sensitivity study used to verify the simulation accuracy will be discussed.

3D Anode Architectures

The above described MATLAB® codes were used to generate four base microstructures, as well as five perturbed realizations of each, to model the anode morphology. These morphologies differed in particle size distribution, type, and overall structure. These are illustrated below. Figures 17 and 18 illustrate the Spherical Column 1 and 2 structures, respectively. The first of which was constructed using a mean particle radius of 12.5μm and a standard deviation of 1 μm, and the latter with 8.5μm and 2 μm, respectively. Spherical Tree-like structures are illustrated in Figure 19, and were constructed with the same distribution as Spherical Column 1, but with relaxed restrictions on branching behavior. The final microstructure, in Figure 20, is the Spherical/Cylindrical Column structure, which includes cylindrical and spherical particles, whose size was again drawn from the same distribution as Spherical Column 1.

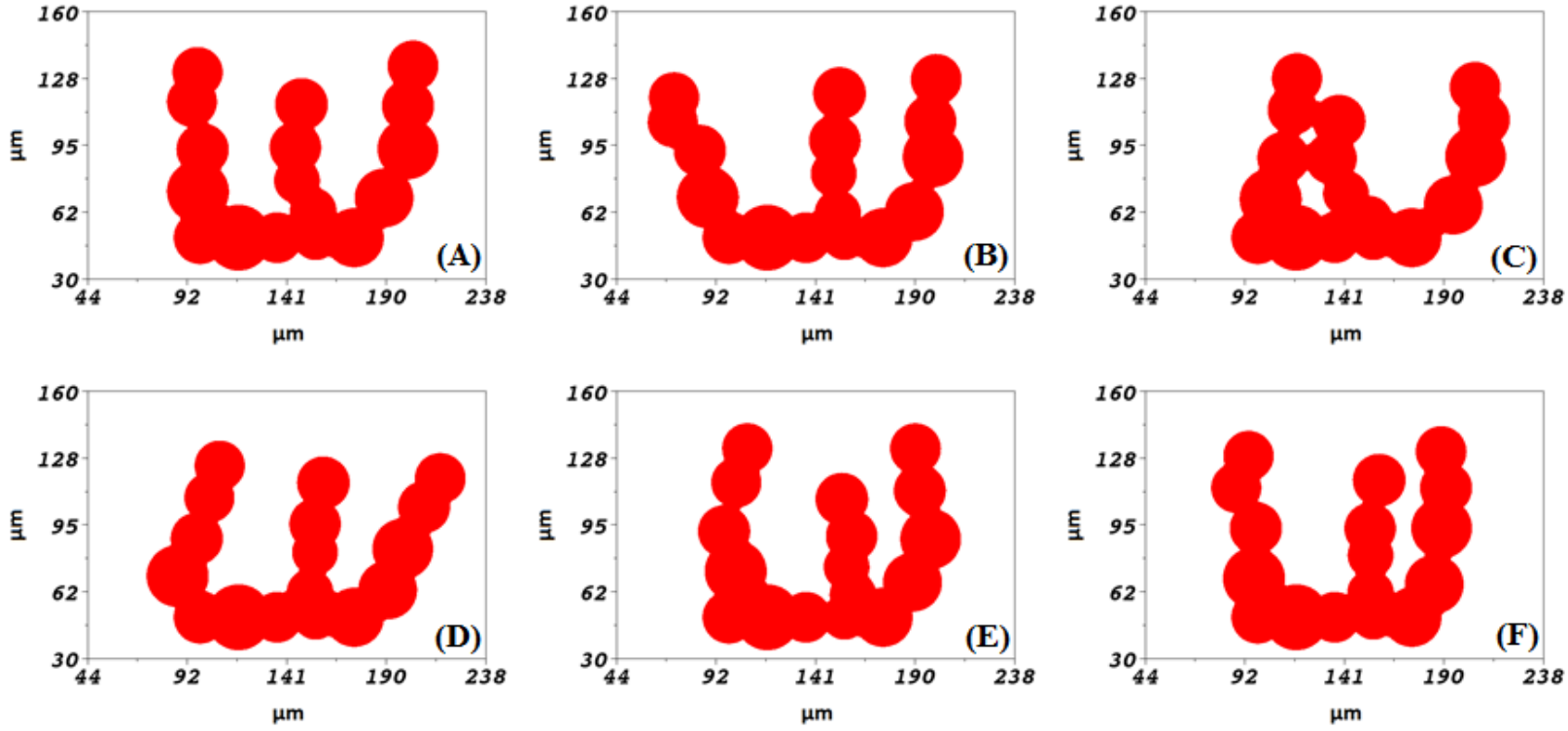

Figure 17. Spherical Column 1 Base(A) and Realizations 1 through 5(B-F). All dimensions in micron.

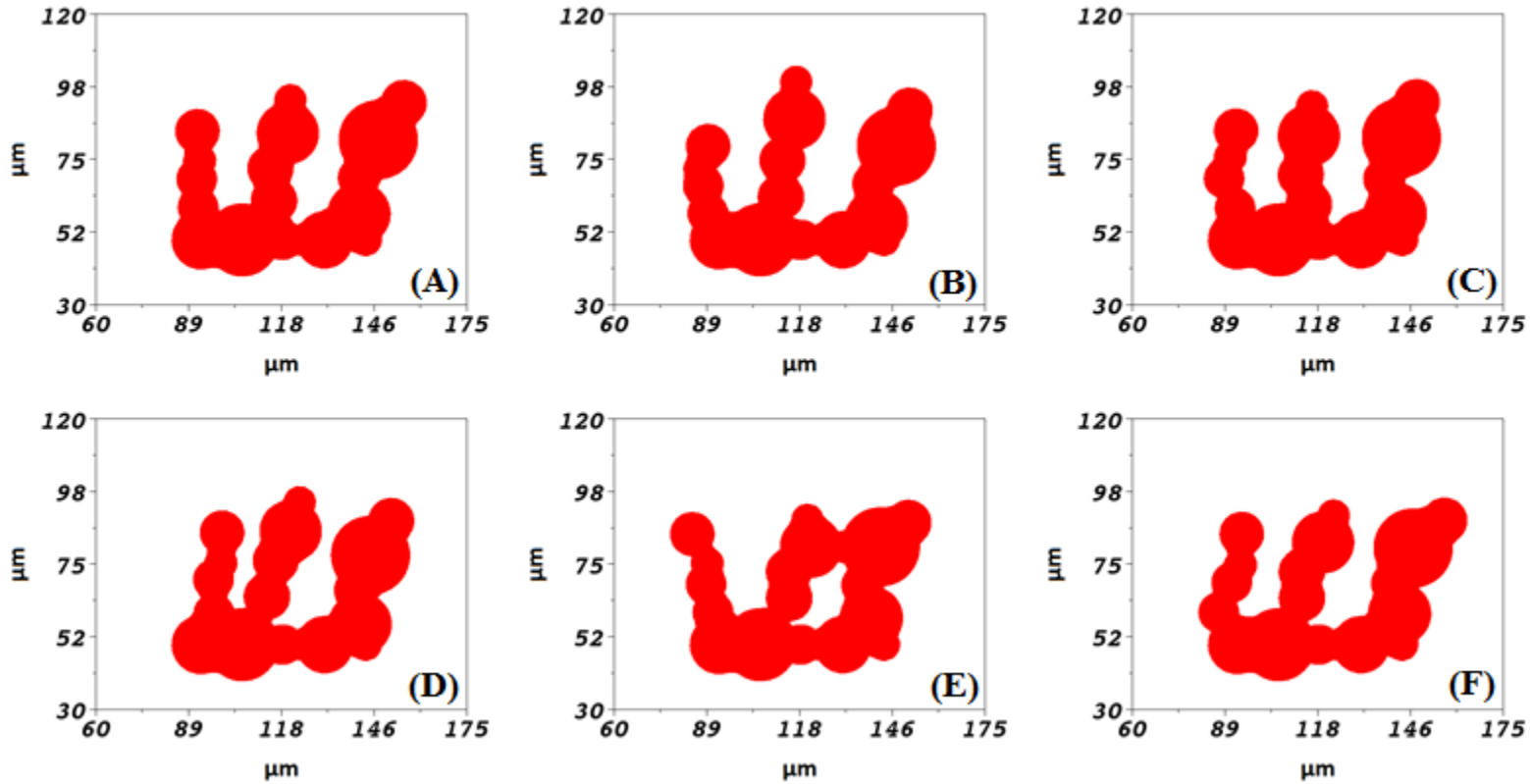

Figure 18. Spherical Column 2 Base(A) and Realizations 1 through 5(B-F). All dimensions in micron.

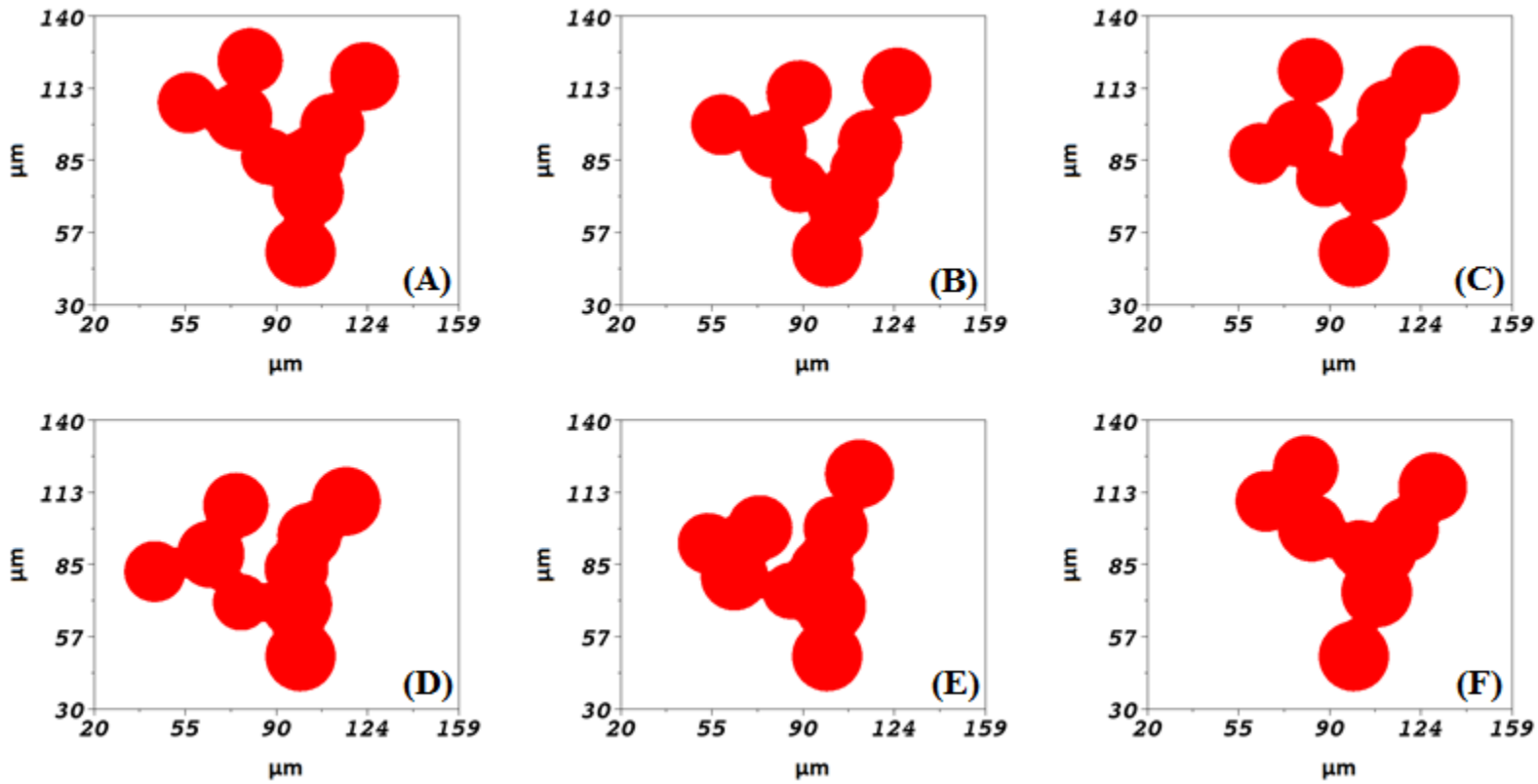

Figure 19. Tree Base(A) and Realizations 1 through 5(B-F). All dimensions in micron.

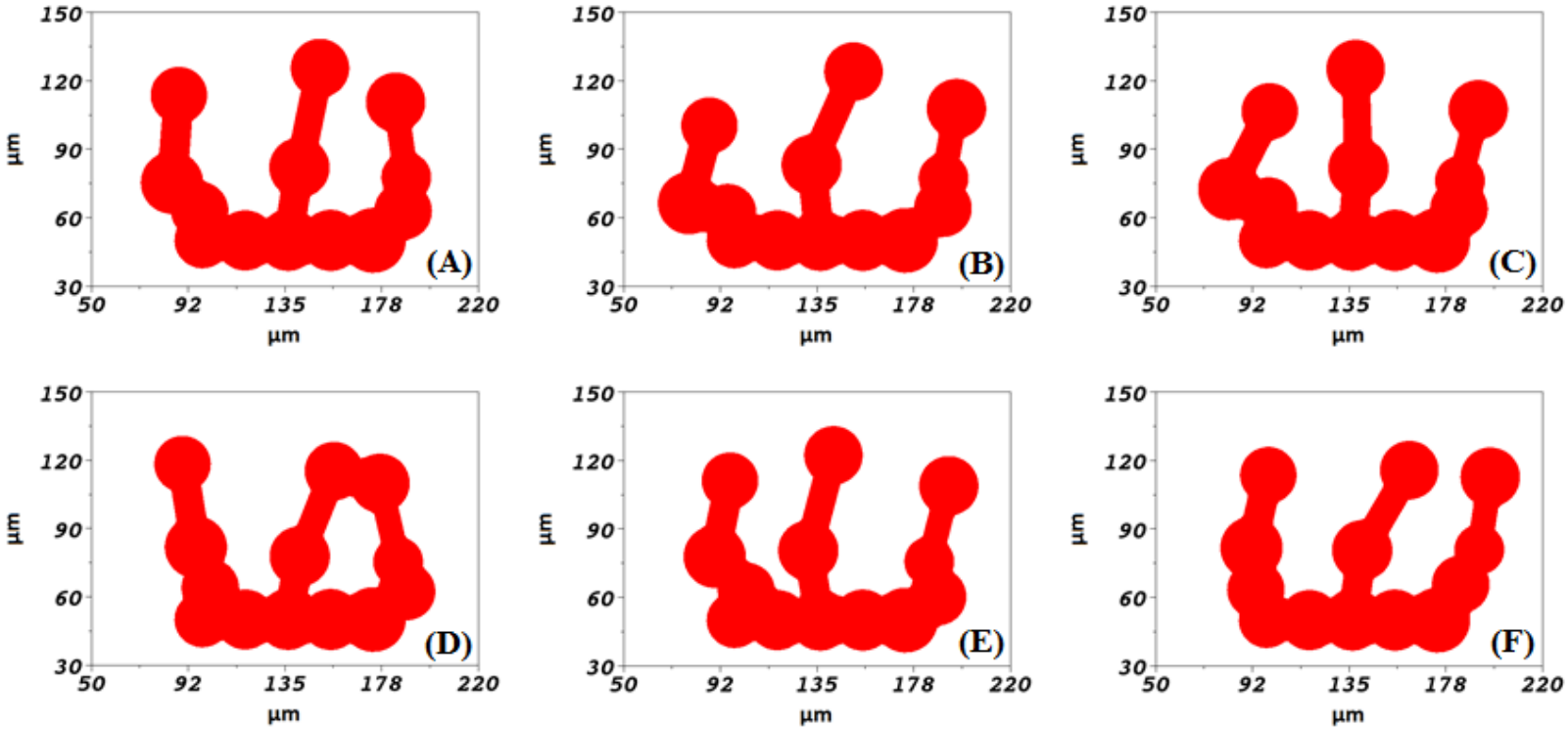

Figure 20. Spherical/Cylindrical Column Base(A) and Realizations 1 through 5(B-F). All dimensions in micron.

However, this parameter only determined the major axis for the cylindrical particles to be placed along the branch direction, while the minor axis, or radius of the cylinder, was determined by enforcing a 2 to 1 length to radius ratio. All branch lengths were determined by a random percentage between 95% and 100% of the mean of the child and parent major axis. Regarding the perturbed incarnations, the branch length and particle angles were allowed to change by a maximum of 20% and 30%, relative to the base case, respectively.

Concentration and Relaxation Behavior

Using MFiX, all structures were discharged at the 1C rate to a cutoff potential of 3V. A single particle model was employed for the cathode, using MATLAB® software to solve the diffusion problem in spherical coordinates as described previously. After discharge, the structures were allowed to relax for 2 hours, and the changes in overall bulk concentration were monitored. For the 1C discharge rate, the following figures illustrate this process over the two hour period for all four microstructure types.

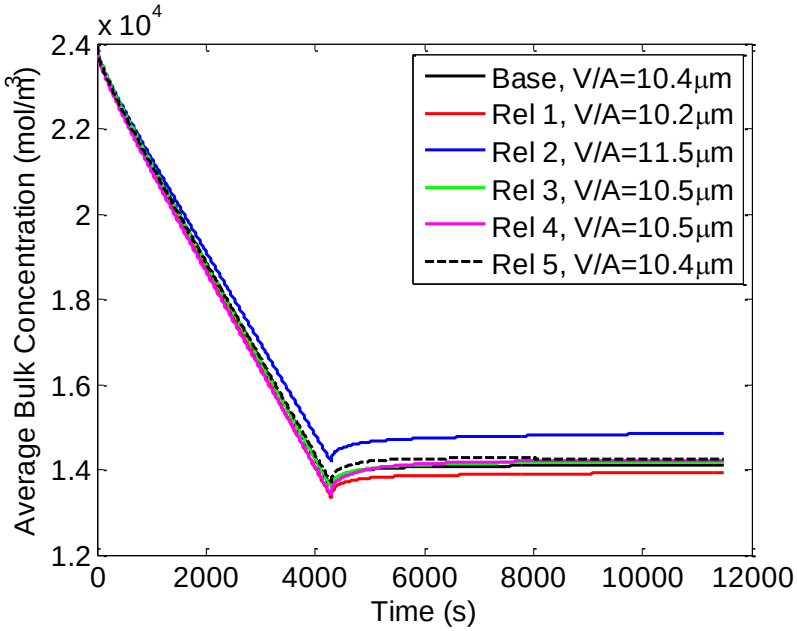

Figure 21. Average bulk concentration for the Spherical Column 1 structures after 1C discharge and 2 hour relaxation.

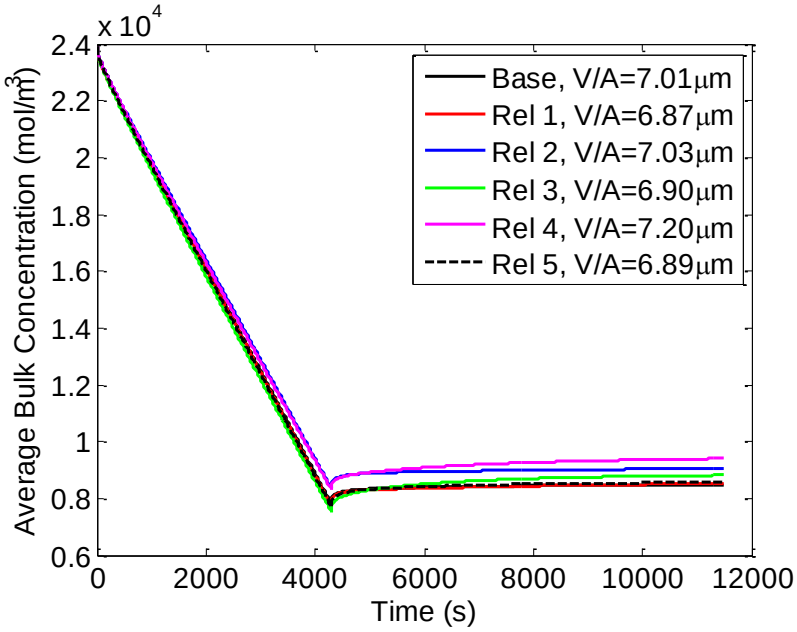

Figure 22. Average bulk concentration for the Spherical Column 2 structures after 1C discharge and 2 hour relaxation.

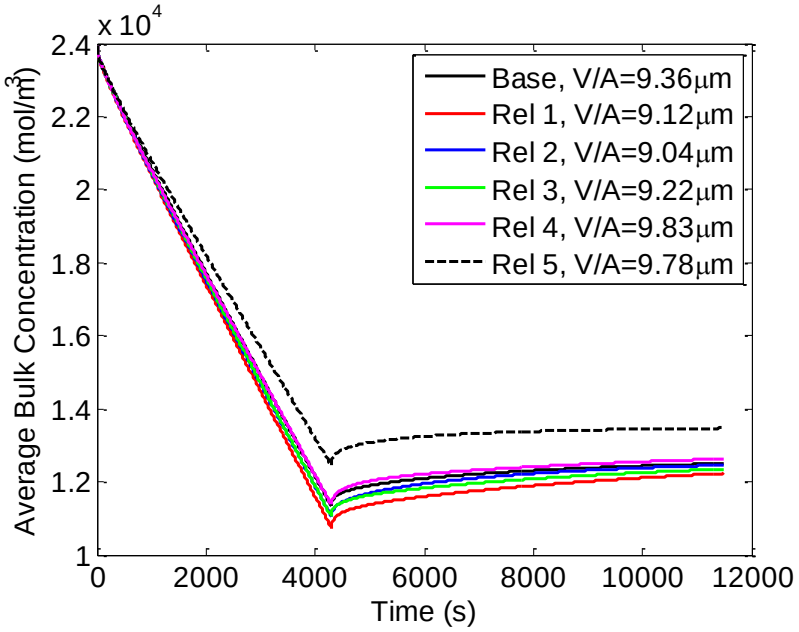

Figure 23. Average bulk concentration for the Spherical Tree structures after 1C discharge and 2 hour relaxation.

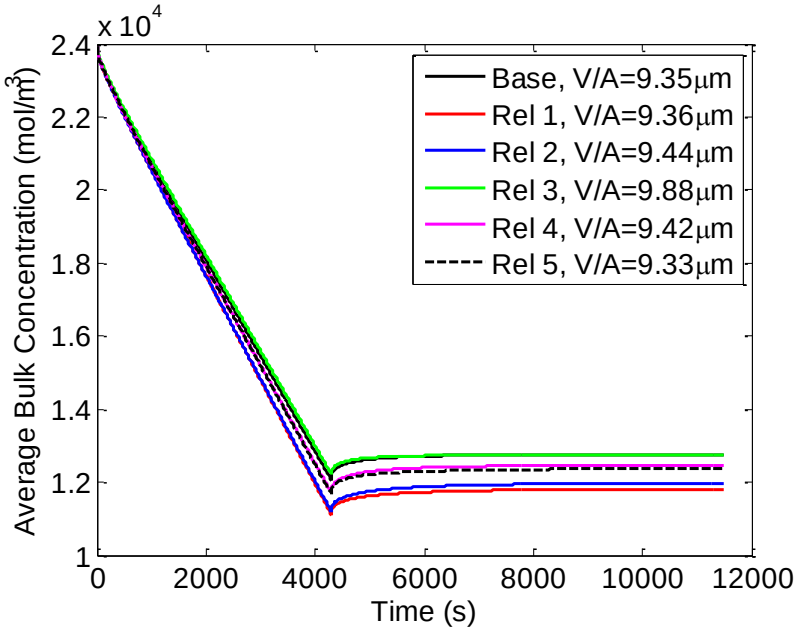

Figure 24. Average bulk concentration for the Spherical/Cylindrical Column structures after 1C discharge and 2 hour relaxation.

As is evident, the small perturbations in the realizations of each microstructure do introduce differences in the discharge process, and this may in some cases be attributable to the differences in the volume to surface area ratio of each structure, as shown in each legend. However, there are several cases where behaviors are not explainable by this ratio. One may consider the Base and Realization 5 of Spherical Column 1, which have equivalent values of this ratio but do show differences in the depth of discharge. Realizations 4 and 5 of the Spherical Tree structures additionally have similar values for this ratio, but show extremely large differences in concentration. There are also simulations that indicate identical behavior, despite large differences in this ratio. This can be illustrated by Realizations 2 and 4 of the Spherical Column 2 structures, as well in the Base and Realization 3 of the Spherical/Cylindrical Column structures. These differences occur due to the different transport processes occurring in these structures as the cutoff potential is approached. This can best be visualized if the following contour plots in are considered, when looking at the lowest concentration values in the structures.

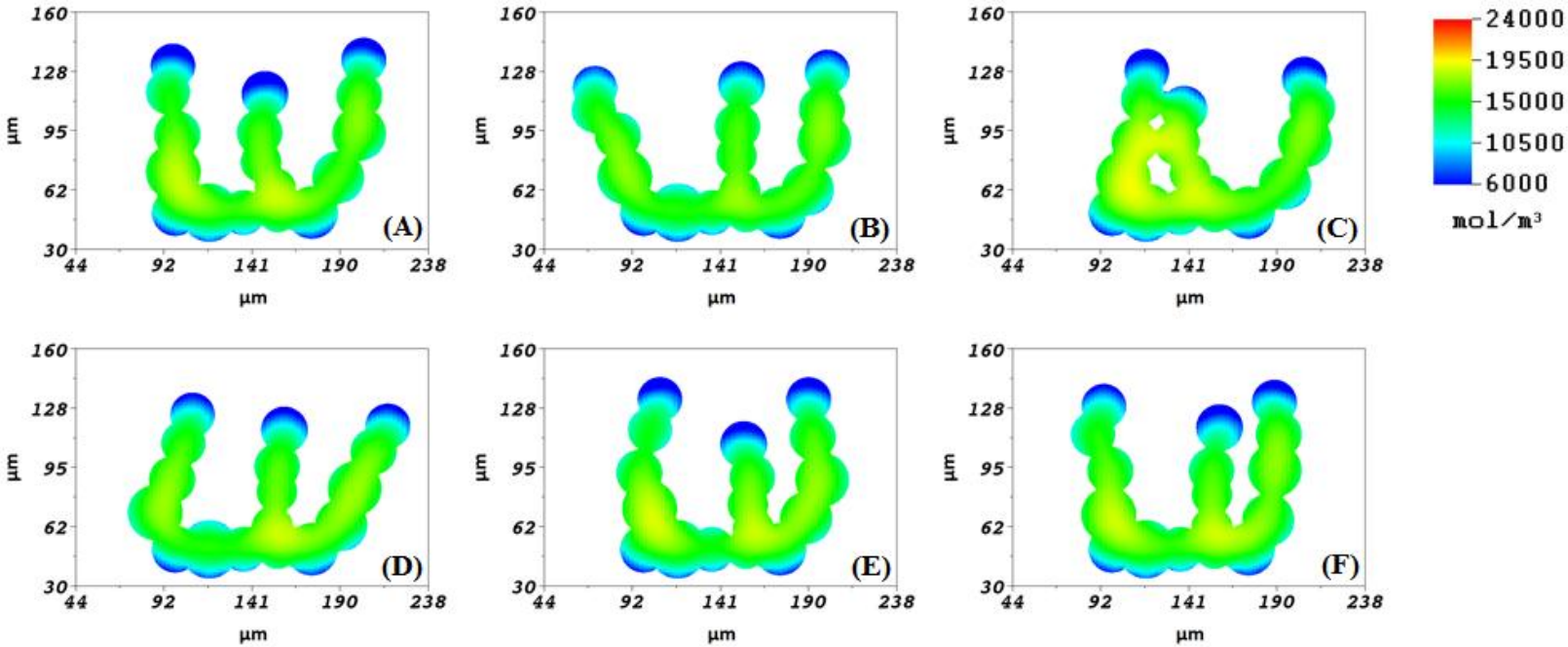

Figure 25. Spherical Column 1 Base(A) and Realizations 1 through 5(B-F) after discharge at 1C. All dimensions in micron and all concentration values in mol/m^3.

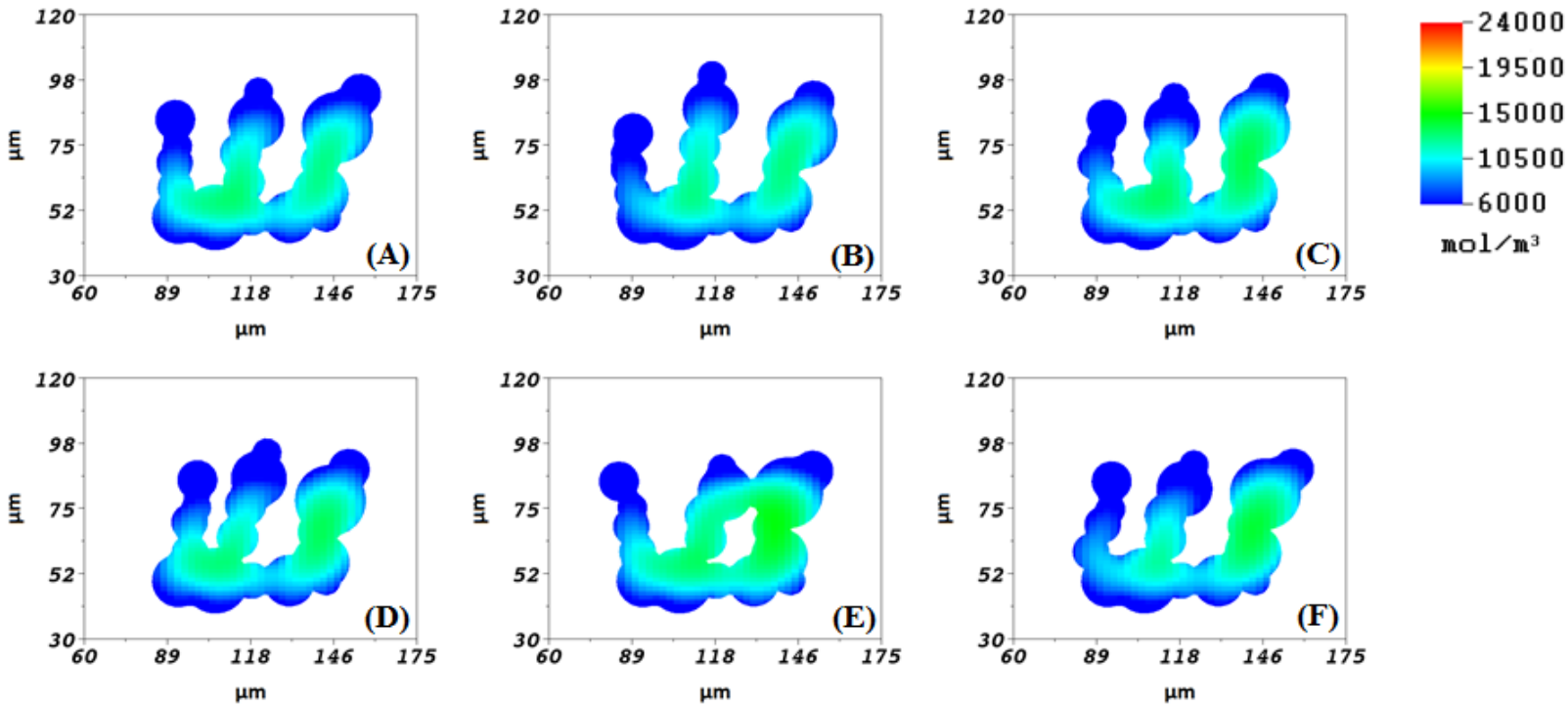

Figure 26. Spherical Column 2 Base(A) and Realizations 1 through 5(B-F) after discharge at 1C. All dimensions in micron and all concentration values in mol/m^3.

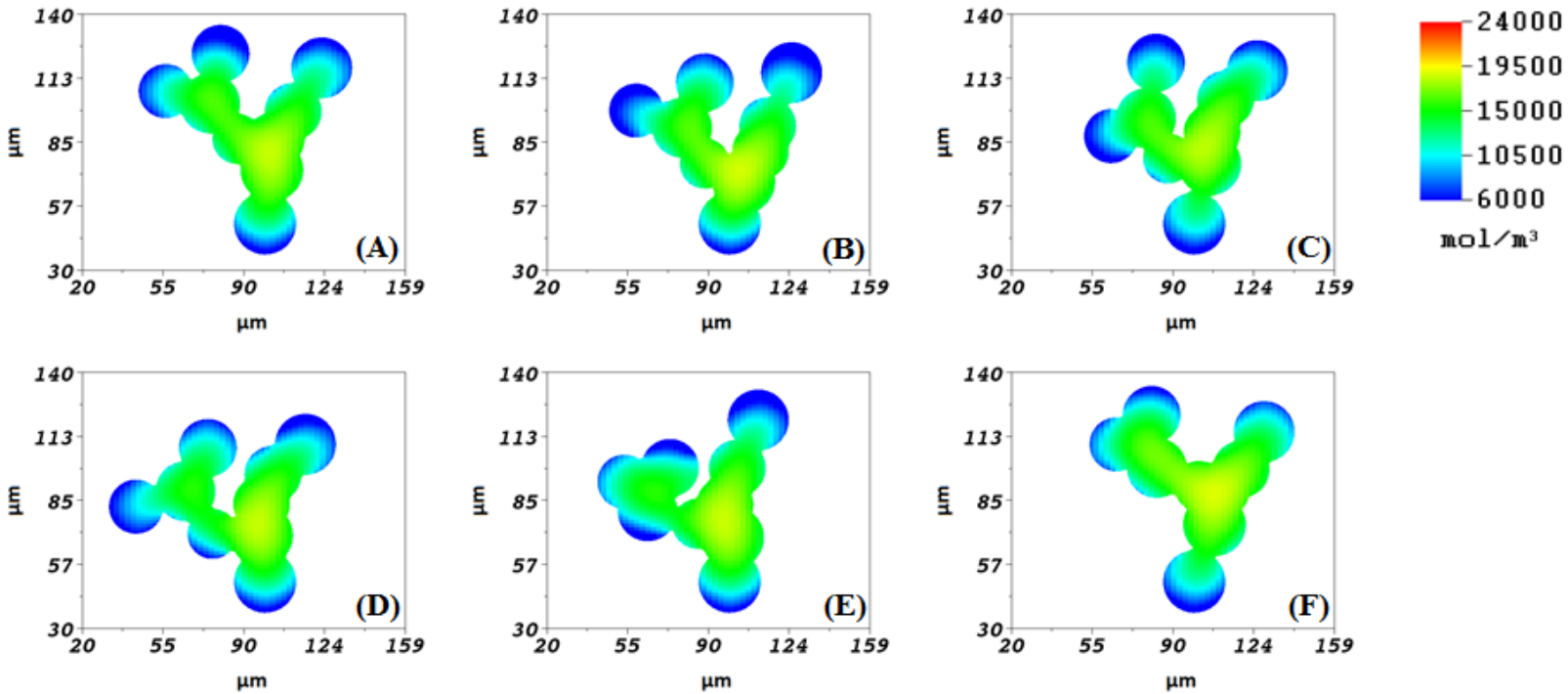

Figure 27. Tree Base(A) and Realizations 1 through 5(B-F) after discharge at 1C. All dimensions in micron and all concentration values in mol/m^3.

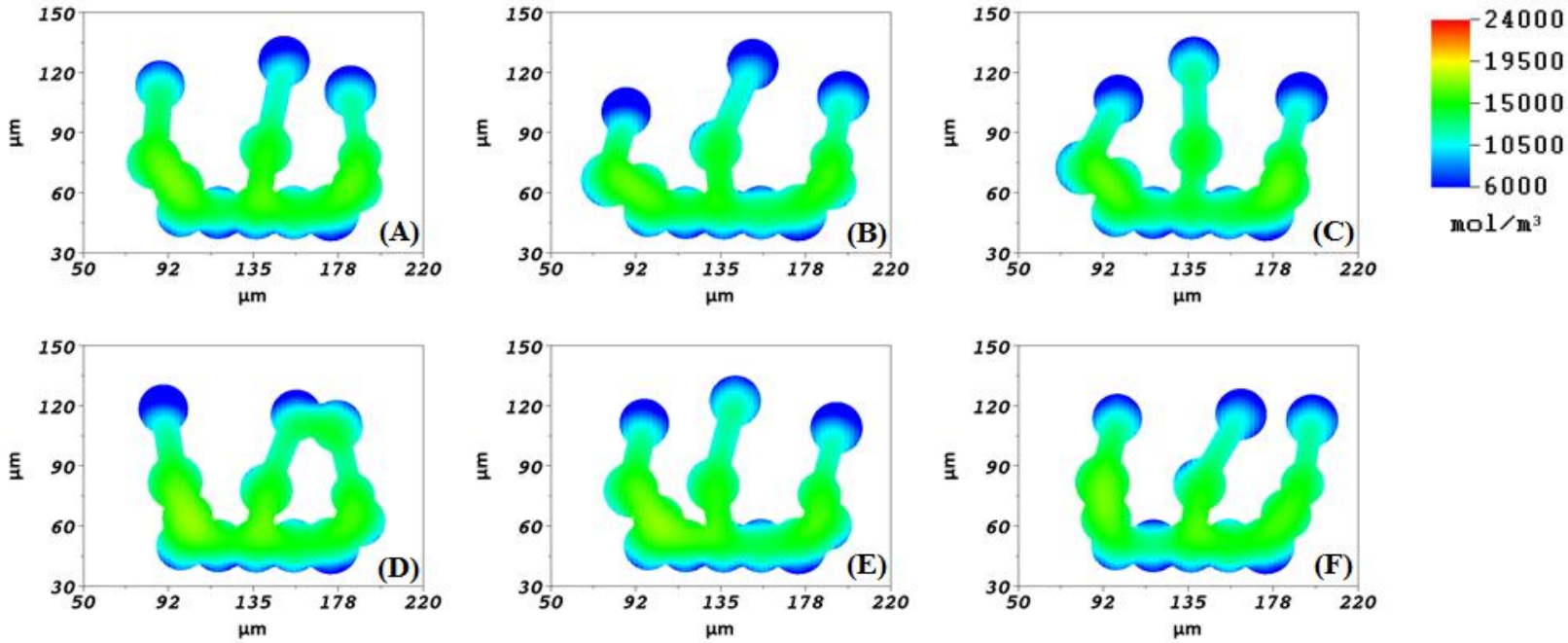

Figure 28. Spherical/Cylindrical Column Base(A) and Realizations 1 through 5(B-F) after discharge at 1C. All dimensions in micron and all concentration values in mol/m^3.

Considering the above figures, there are clear differences between each anode morphology, as well as between different realizations. As mentioned before, the Base and Realization 5 morphologies under Spherical Column 1 show significant differences between concentration values, despite similar volume to area ratios. Considering Figures 25(A) and 25(F), there are large areas of lithium depletion at the tops of the columns in the Base case that are not present in the other. Additionally, there are larger concentrations present in Realization 5 at the core of the structure that have remained due to differences in transport behavior. Of particular interest is the behavior of Realization 2, in Figure 25(C). This particular structure has an increased level of particle interconnectivity as is reflected in the volume to area ratio, as some of the active surface area has been lost due to the overlap in the upper areas of the columns. This structure has significant pockets of high concentration near these areas of overlap and this explains the low depth of discharge shown in Figure 21. This exact same behavior is seen in Realization 4 for Spherical Column 2, as there is the additional overlap in the column structures. Note here additionally, that because of the smaller particle mean size used for these structures, these structures show a far lower concentration after discharge. Considering the differences are between Realization 4 and Realization 5 of the Spherical Tree structures noted previously, Realization 4 shows a far more depleted profile, as shown in Figure 27(E), especially when one considers the left branch. This is due to increased particle isolation because of the small contact between the branch and the rest of the structure. Interestingly, with respect to the Spherical/Cylindrical Column structures, there are large areas of uniform concentration distribution in the cylindrical

particles composing the structure. This is due to the lower diffusion length perpendicular to the cylinder axis. In several cases there exist large concentrations in the centers of the top most spheres of the structures. This has occurred specifically due to the cylindrical particles, as the diffusion front has moved quickly through them and upwards to encapsulate these pockets in the spheres.

To better visualize some of the previously mentioned effects, one may consider the variance in bulk and surface concentration between the realizations of each morphology over the discharge period. Shown below in Figures 29 and 30 are four averaged bulk concentration curves for the four different morphologies where the average bulk concentration values for each base and respective realizations have been averaged at each temporal point. Further, a 1 standard deviation bar has been applied to each curve to illustrate the kind of variance in the concentration values.

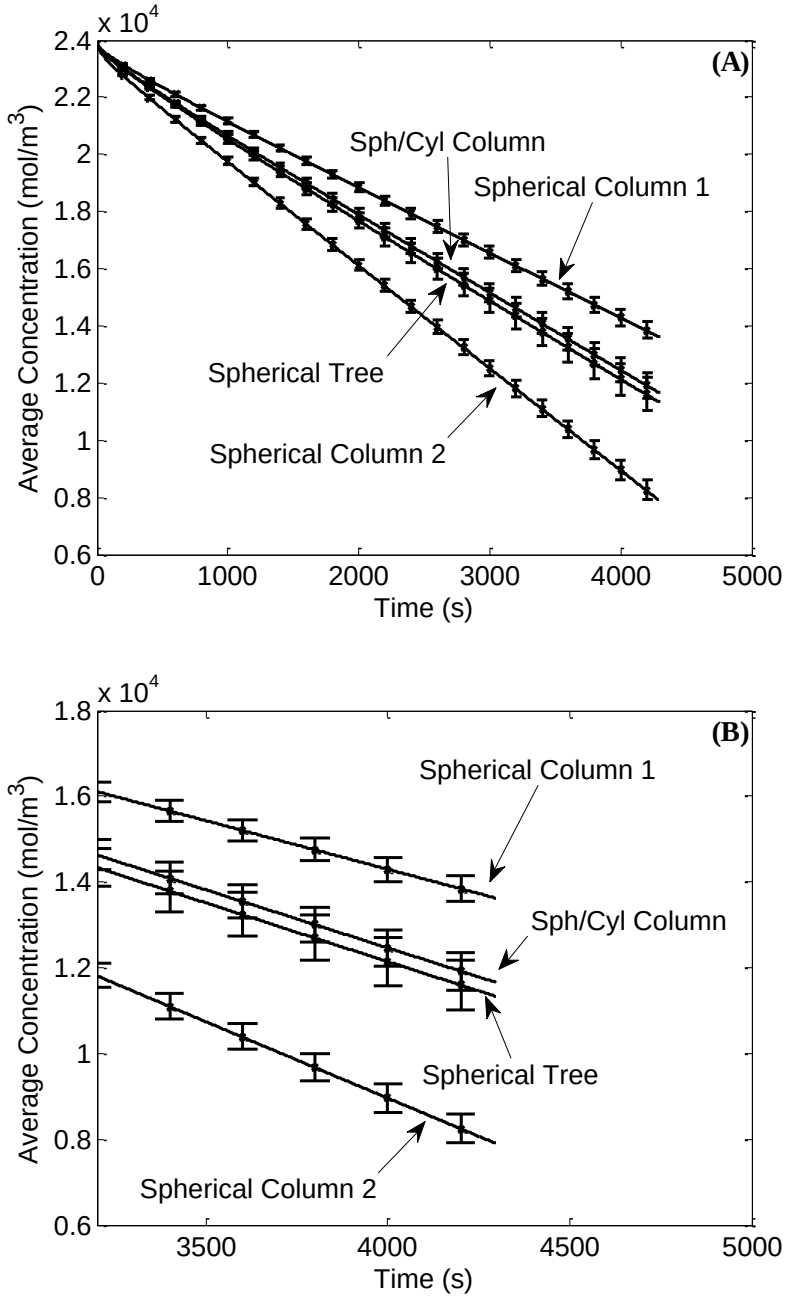

Figure 29. Average bulk concentration curves for the four morphologies. A 1 standard deviation bar has been applied at several times.

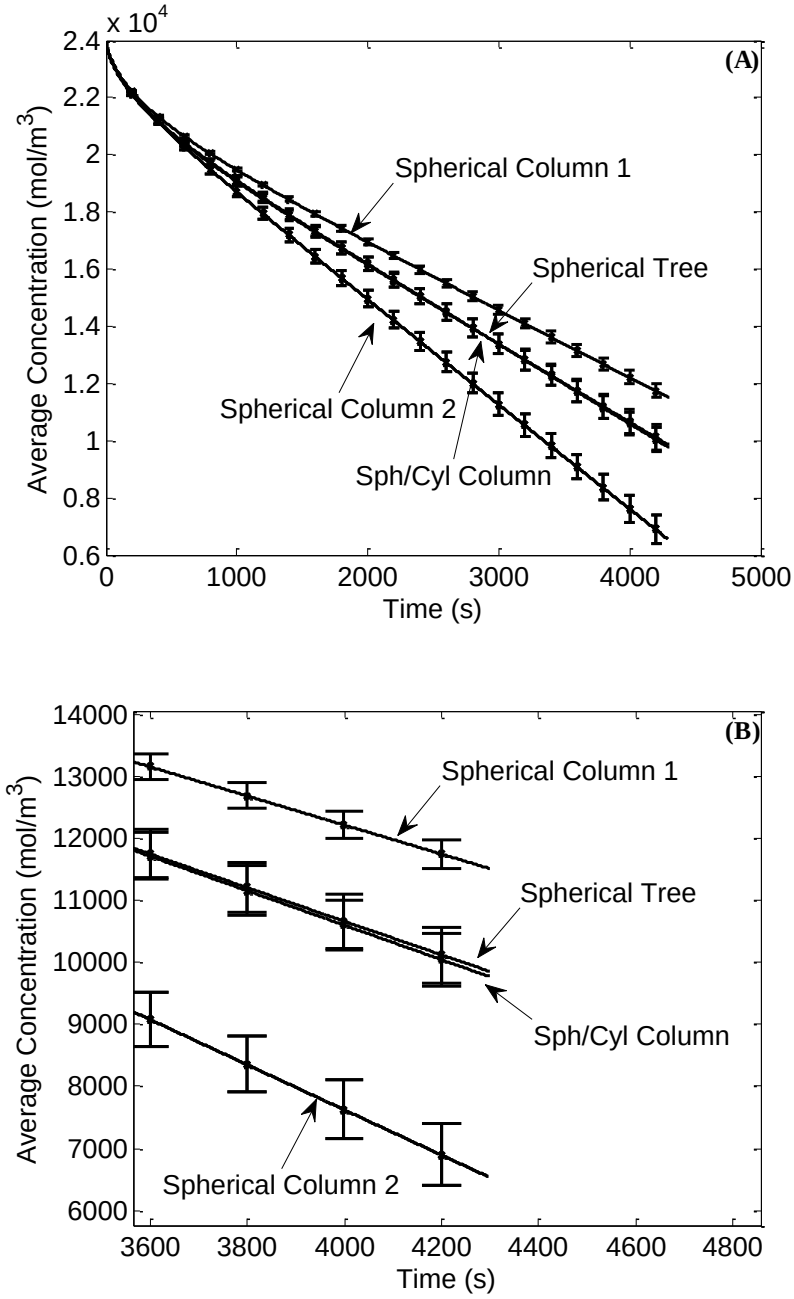

Figure 30. Average surface concentration curves for the four morphologies. A 1 standard deviation bar has been applied at several times.

As is clear from the bottom of Figure 29 the Spherical Tree structures have the largest variance in bulk concentration over the course of discharge. This can be explained visually if the images in Figure 19 are considered, as the Spherical Tree structures show the most apparent change in overall morphology when the described perturbations were applied. With the inclusion of multiple branches, and under the same perturbing conditions, these extra branches offer increased degrees of freedom by which the overall structure can be altered when compared to the others. Further, as illustrated in Figure 29, the Spherical/Cylindrical Column structures follow next in the greatest amount of variance for bulk concentration, but also closely match the average bulk concentration values of the Spherical Tree structures. By this evidence, the inclusion of different particle types is also an extra degree of freedom by which the variance in structure performance can increase. This result is especially striking, as the inclusion of a larger particle distribution, as in the case of the Spherical Column 2 structures, does not seem to have as large of an effect. Considering Figure 30, it is interesting that the greatest variance in average surface concentration occurs with the Spherical Column 2 structures, as opposed to before with average bulk concentration. This is likely due to the fact that the larger particles smear out the effects of low concentration in the bulk concentration calculation. While, for the surface concentration, the effects of both the large and small particles are on more equal footing. Here again, the Spherical Tree and Spherical/Cylindrical Column structures follow each other especially well.

If Realizations 2 and 3 of the Spherical/Cylindrical Column structures are considered in Figure 24, they are nearly indistinguishable from their discharge

behaviors, despite differences in their volume to surface are ratios. However, it is also clear that these two structures distinguish themselves during relaxation, indicating the need for another performance parameter, namely relaxation time. To investigate this parameter the temporal change in average bulk concentration was monitored until it dropped to .01 mol/m^3s or below and the structure was considered fully relaxed (See Appendix A). Displayed in Table 2 are the results of this calculation, and there are clearly wide ranges in relaxation behavior when overall structure is compared, as well as between base cases and their respective realizations. Additionally shown is the change in bulk concentration, Δc_s, between current cutoff and the point of complete relaxation. Additionally, contour plots of each structure are shown in Figures 31, 32, 33, and 34 after one hour of relaxation. Immediately noticeable is the fact that many of the Spherical Tree structures did not relax within the two hour period. This behavior can be best explained by the contour plots shown in Figures 27 and 33. The increased level of branching in the structure allows for more particle isolation, and therefore creates large lithium deprived zones in the structure relative to the main body. Another interesting development occurs in the Spherical/Cylindrical Column structures.

Table 2. Relaxation times for each microstructure and different realizations for the discharge rate of 1C to a cutoff potential of 3V.

Architecture	Relaxation Time	Δc_s	V/A Ratio
Spherical Column 1			
Base	1339.1	478.61	10.4
Realization 1	1342.8	505.78	10.2
Realization 2	1926.0	565.58	11.5
Realization 3	1732.5	545.15	10.5
Realization 4	2346.9	780.75	10.5
Realization 5	1421.4	568.04	10.4
Spherical Column 2			
Base	2550.6	616.35	7.01
Realization 1	5166.2	753.30	6.87
Realization 2	3057.4	618.31	7.03
Realization 3	5900.3	1252.43	6.90
Realization 4	7154.4	1087.37	7.20
Realization 5	4081.2	810.35	6.89
Spherical Tree			
Base	6392.2	1136.17	9.36
Realization 1	>7200.0	1489.09	9.12
Realization 2	>7200.0	1404.47	9.04
Realization 3	>7200.0	1299.80	9.22
Realization 4	>7200.0	1257.93	9.83
Realization 5	3926.6	1016.61	9.78
Spherical/Cylindrical			
Base	1795.2	652.57	9.35
Realization 1	2323.2	646.69	9.36
Realization 2	2700.5	738.15	9.44
Realization 3	1409.2	503.47	9.88
Realization 4	2106.3	698.16	9.42
Realization 5	1988.6	637.65	9.33

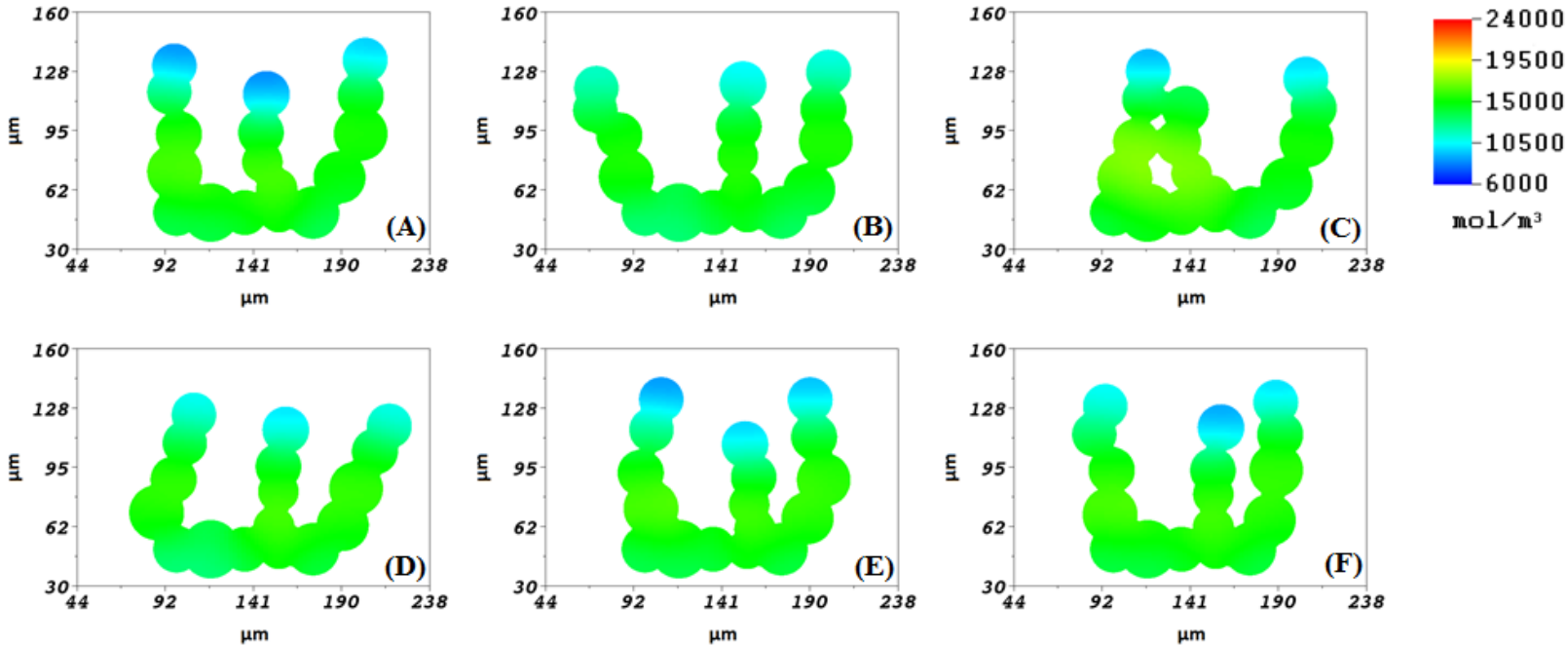

Figure 31. Spherical Column 1 Base(A) and Realizations 1 through 5(B-F) after relaxation for 1 hour following discharge at 1C. All dimensions in micron and all concentration values in mol/m^3.

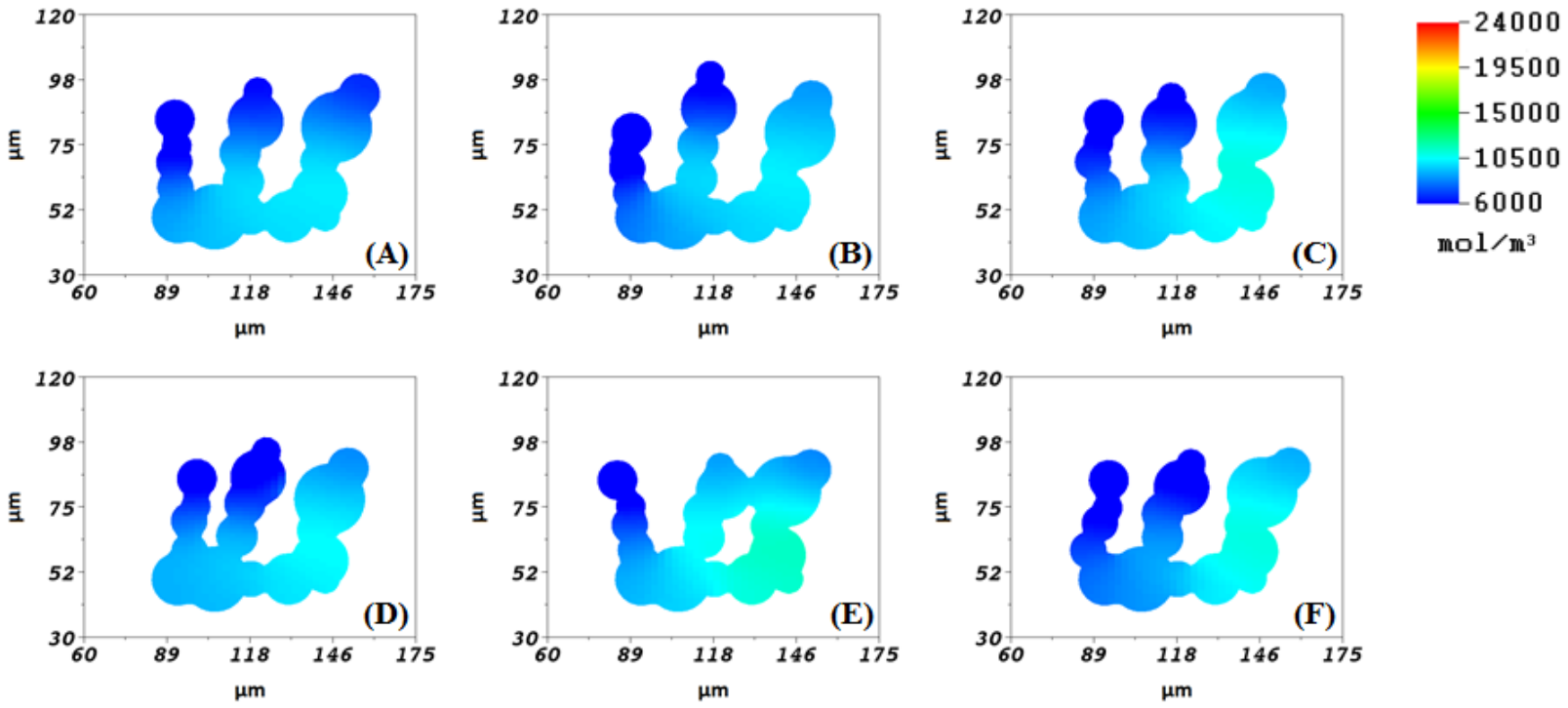

Figure 32. Spherical Column 2 Base(A) and Realizations 1 through 5(B-F) after relaxation for 1 hour following discharge at 1C. All dimensions in micron and all concentration values in mol/m^3.

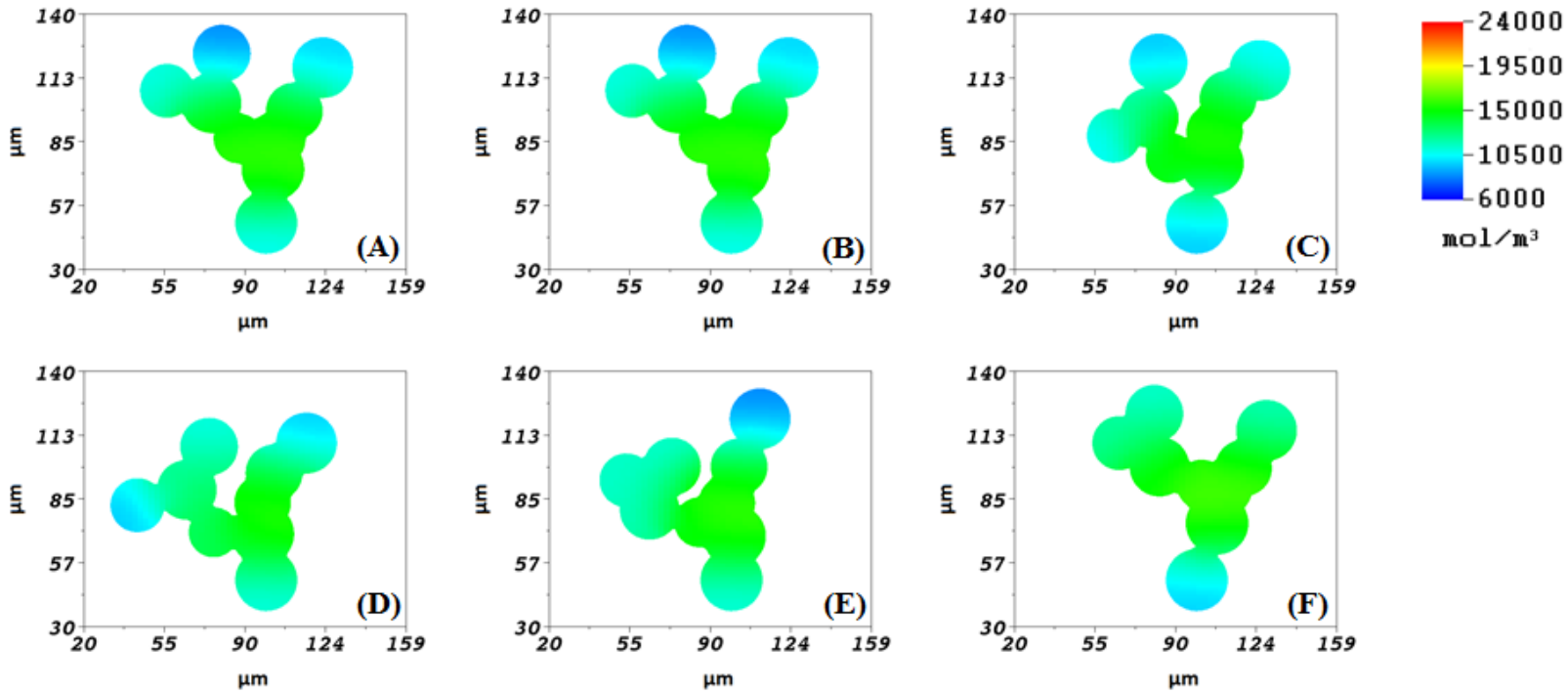

Figure 33. Tree Base(A) and Realizations 1 through 5(B-F) after relaxation for 1 hour following discharge at 1C. All dimensions in micron and all concentration values in mol/m^3.

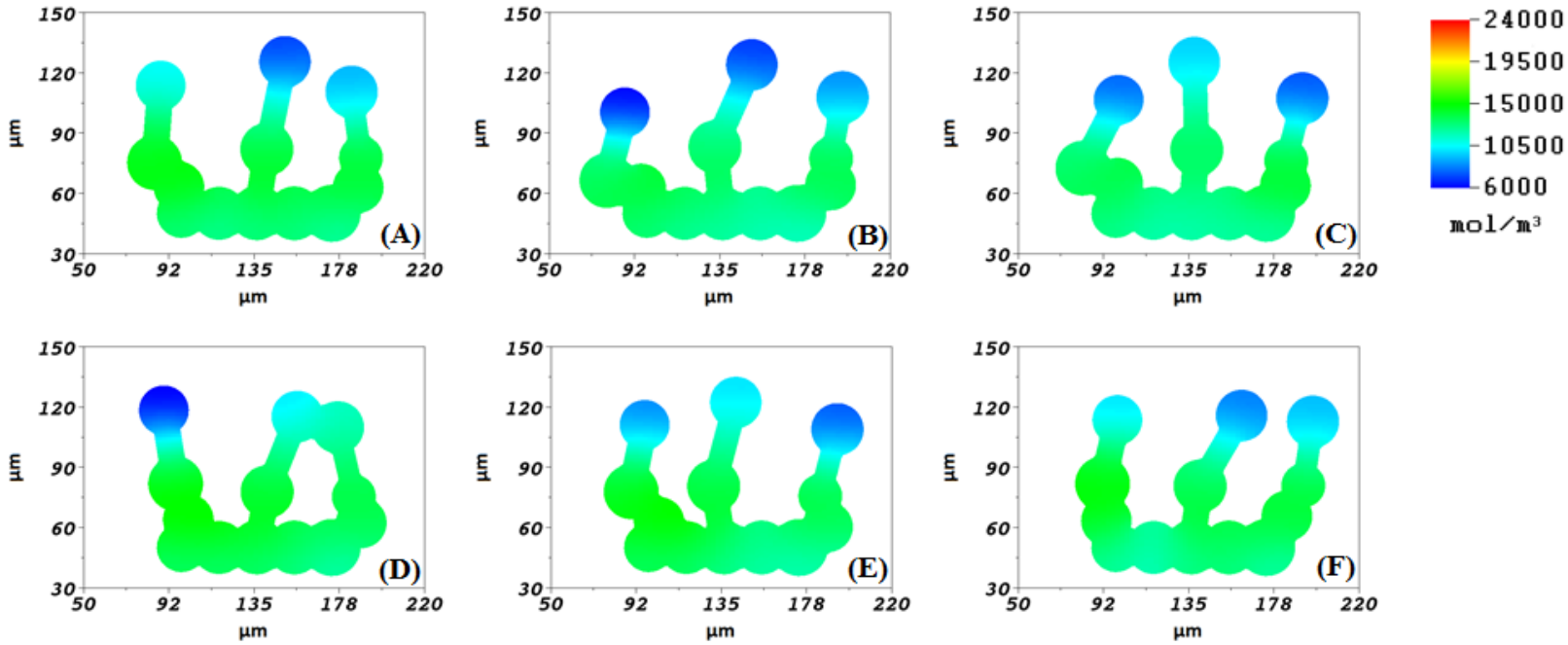

Figure 34. Spherical/Cylindrical Column Base(A) and Realizations 1 through 5(B-F) after relaxation for 1 hour following discharge at 1C. All dimensions in micron and all concentration values in mol/m^3.

The relaxation times for these are either on the order or slightly larger than those of the Spherical Column 1 structure. This is likely due to the fact that the most isolated particles, those at the tops of the columns, are, like the Spherical Tree structure, more isolated than those at the core due to the cylindrical particles. This is clearly visible when one compares Figures 28 and 34 and compares the top most spherical particles of Spherical/Cylindrical Structure with the Spherical Column 1 Structure. These cylinders create larger diffusion distances to these particles as lithium must now travel along the cylinder axis. While this is particularly detrimental to overall structure performance, cylindrical particles offer shorter diffusion distances perpendicular to their axis, as shown before, and thus take greatest advantage of the 3D geometry.

An excellent example of the kinds of variation in relaxation time between realizations is that seen in the Spherical Tree structures. Here, the majority do not relax in two hours, while Realization 5 does so in less than 1 hour. Considering the geometry of this structure in Figure 19, this realization has less particle isolation, due to the 'clumping' of the three particles on the left branch. Also, when comparing to Realization 4, which has a similar volume to surface area ratio, the particles on the leftmost branch have a strong connection to the structure, unlike the choke point in Realization 4. Further, this structure has the second highest volume to area ratio, meaning that less lithium is lost upon discharge, as illustrated by Figure 23, and therefore gradients can be relaxed quickly. Considering Figures 27 and 33, the uniformity in concentration seen in Realization 5 is also very clear. Another example of large variances in relaxation time is evident in the Spherical Column 2 structures. If Figures 26 and 32 are considered, upon

discharge the leftmost half of the structure is fully depleted for all realizations, while there are larger concentrations in the large particles composing the rightmost sections. This unevenness in depletion speed creates large gradients across the entirety of the structure, meaning that diffusion distances are long, causing relaxation time to be that much longer as well. It is also to be noted here that in the relaxation of Realization 3 of the Spherical Column 2 structures, there is a slightly higher concentration in the largest particle in the middle column at the right most edge, visible in Figure 32(D). This has been identified as the behavior of a small scalar cell, but has been viewed as being erroneous in the consideration of the averaged quantities considered here. Another interesting behavior is noted when those structures with overlap in the upper portions of the columns are considered, namely Realizations 2, 4, and 3 from the Spherical Column 1, Spherical Column 2, and Spherical/Cylindrical Column morphologies, respectively. It has already been noted that with the increased degree of overlap, less particle isolation occurs, but with a sacrifice in depth of discharge. It would be expected, however, that such structures would show some of the fastest relaxation times, as large gradients do not develop. While this seems to be true for the realization from the Spherical/Cylindrical Column morphology, this expected behavior is not evident in the other two. This result may actually be due to the fact that at these overlapped regions, while there is less gradient development, there is also less utilization of material, due to the local drop in surface area, which would tend to create regions of high concentration relative to the rest of the structure. Therefore, these regions of high concentration will take longer to redistribute. This is clearly visible in Figures 25(C) and 25(E), as well as

the Δc_s values from Table 2, as there is associated a large change in average bulk concentration for those structures with overlap. However, due to the performance of the Spherical/Cylindrical realization, there may be an optimum level of overlap to both prevent particle isolation and still maintain enough surface area to ensure uniformity of discharge. Finally, as discussed previously, the differences in Realizations 1 and 2 of the Spherical/Cylindrical Column are clear only when the relaxation behavior is considered, as displayed in Table 2.

After relaxation, the base cases of each morphology were recharged at 1C to a cutoff potential of 4.1V. After this was completed, an important difference to consider is that between the surface and bulk concentration values, as well as the amount of concentration returned upon recharge. These are indicative of how well a particular structure can recharge, and therefore are imperative for judging how well a system can cycle through many uses. Shown in Figure 36 are the values of average bulk and surface concentration after recharge from an initial discharge at 1C, and contour plots over the entire cycle are shown in the following figure.

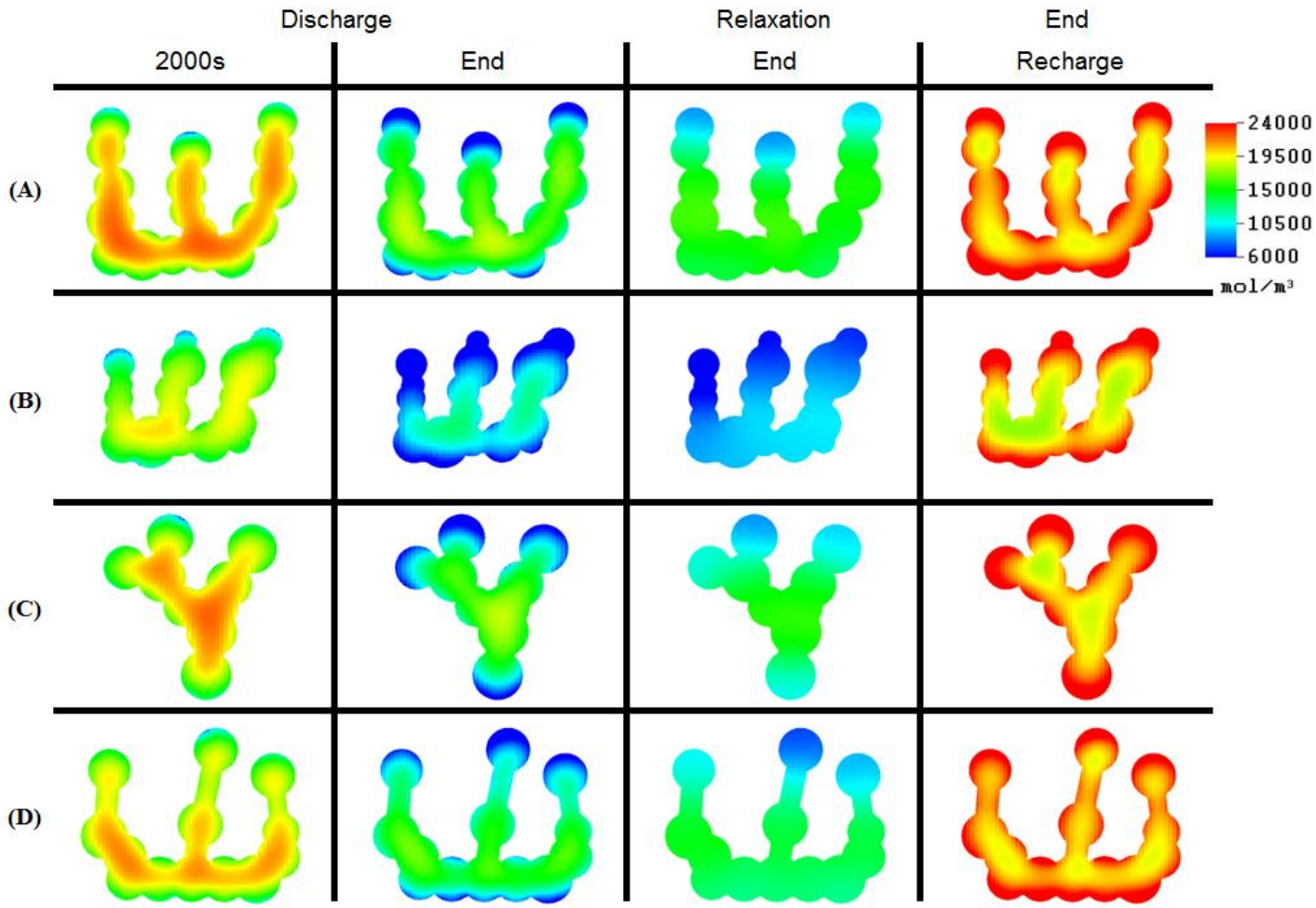

Figure 35. Spherical Column 1(Row A), Spherical Column 2(Row B), Tree(Row C) and Spherical/Cylindrical Column (Row D) over a cycle with initial discharge at 1C. All dimensions in micron and all concentration values in mol/m^3.(To Scale from Figtures 17 to 20)

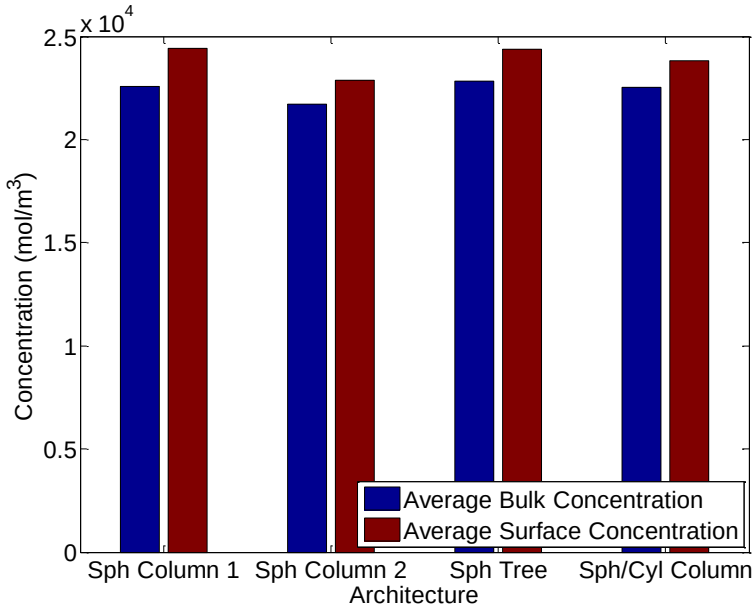

Figure 36. Surface and bulk concentrations in the base cases after relaxation and recharge to a potential of 4.1V from the initial 1C discharge rate.

The differences between bulk and surface concentrations after recharge is due to the fact that the surface will saturate faster than lithium can diffuse into the core of the structure. When this happens, as the voltage is based on the surface concentration, the cutoff voltage will be reached that much faster. Those structures that show the most similar values after recharge are therefore advantageous, because obtaining uniformity is key for maximum capacity recovery. Immediately discernible from the above results is how poorly the Spherical Column 2 structure performs on reclaiming available capacity despite the small difference between surface and bulk concentration. Considering the highly depleted profile after recharge, it is clear that the cathode particle saturates faster than this structure can reclaim capacity, forcing the recharge voltage to cutoff. Figure 35 illustrates these results very well, as it can be seen that in the Spherical Column 2 structure there exist large areas of low concentration in the core of the structure, which

are not present in the others. Furthermore, one may additionally notice how much more uniform the concentration is in the cylindrical particles of the Spherical/Cylindrical Column after recharge. Comparatively, the Spherical Column 1 structure only has such uniformity at points where the spheres in the columns are barely in contact. This is a consequence of the local diffusion length, which is very short for the cylindrical particles. Further, these cylindrical particles aid in creating more uniform concentrations in the spherical particles they are in contact with, as observed if one compares these spheres with those that are closely packed with other spheres in the same structure. This kind of interaction between particle shapes is a key design feature for 3D architectures that may be exploited.

In order to identify any rate dependencies in the above results, further cycles were conducted on the base cases with discharge rates of C/2 and C/10, one hour of relaxation, and recharge at 1C to the potential of 4.1V. Corresponding contour plots are shown below.

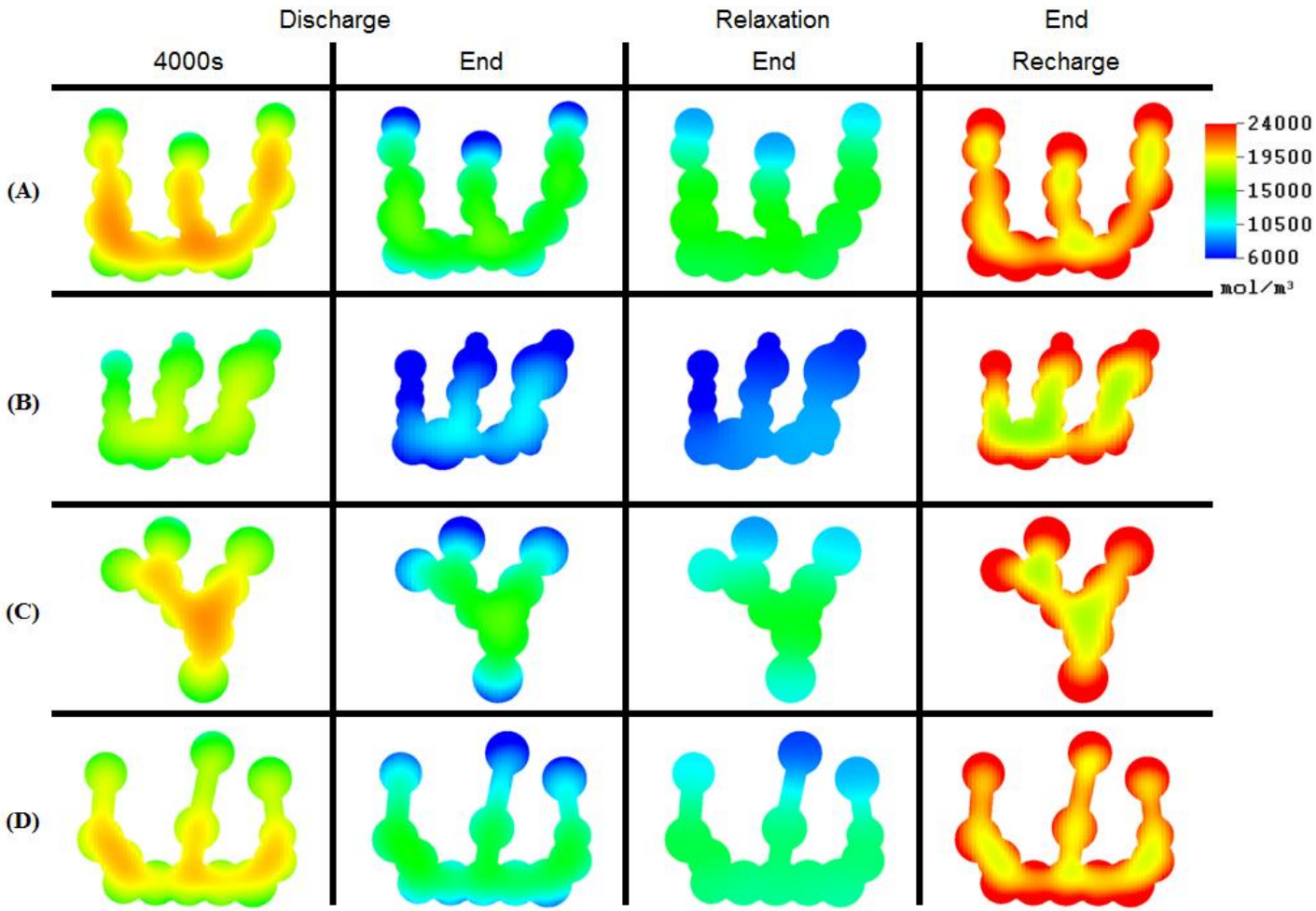

Figure 37. Spherical Column 1(Row A), Spherical Column 2(Row B), Tree(Row C) and Spherical/Cylindrical Column (Row D) over a cycle with initial discharge at C/2. All dimensions in micron and all concentration values in mol/m^3.

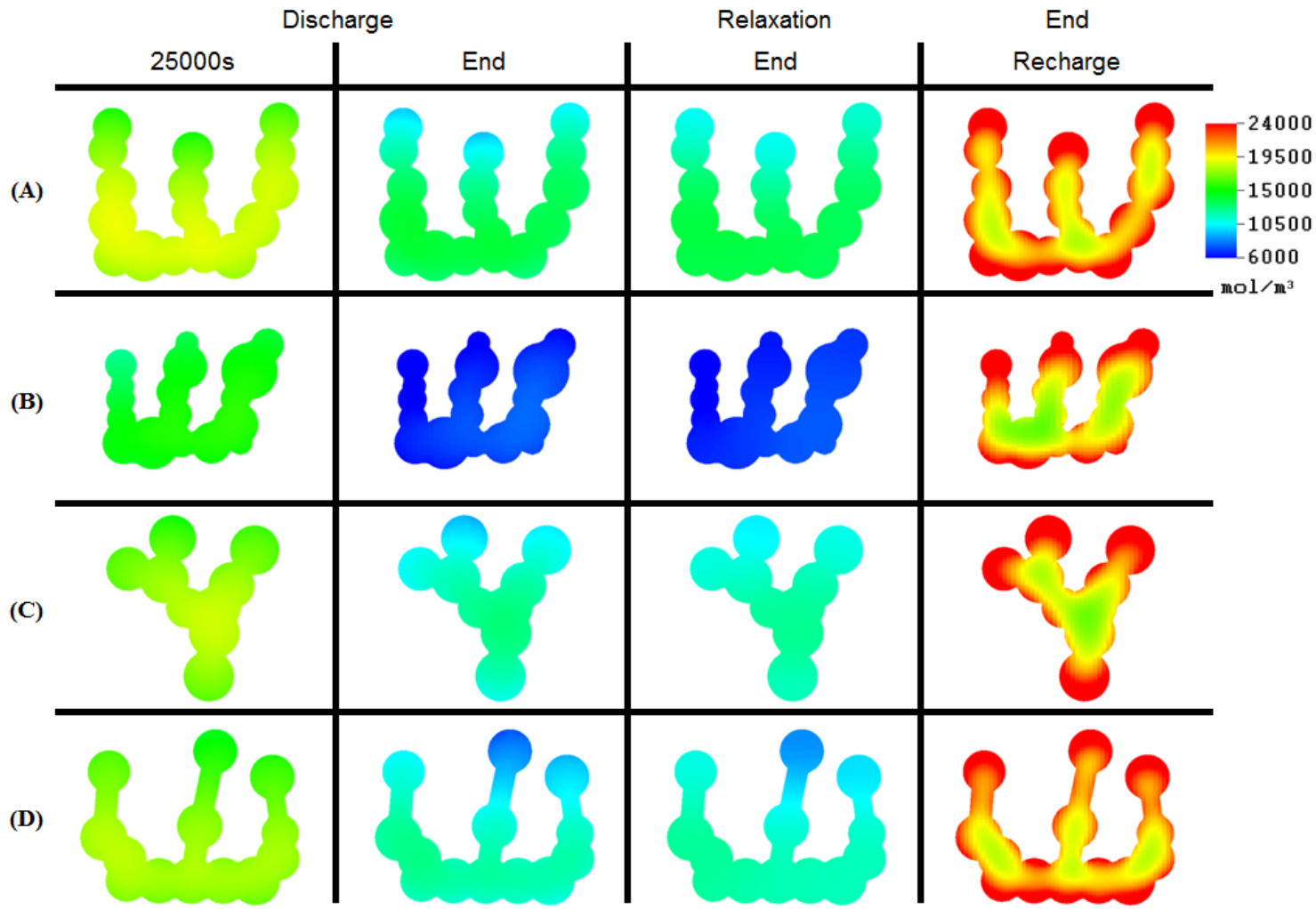

Figure 38. Spherical Column 1(Row A), Spherical Column 2(Row B), Tree(Row C) and Spherical/Cylindrical Column (Row D) over a cycle with initial discharge at C/10. All dimensions in micron and all concentration values in mol/m^3.

However, before recharging as completed before, the structures were given 2 hours of relaxation time and the same gradient analysis as before was conducted, and the results are shown below in the following Tables.

Table 3. Relaxation times for each base case microstructure at the discharge rate of C/2 to a cutoff potential of 3V.

Architecture	Relaxation Time	Δc_s	V/A Ratio
Spherical Column 1			
Base	958.70	233.50	10.4
Spherical Column 2			
Base	1764.10	303.35	7.01
Spherical Tree			
Base	5419.40	661.21	9.36
Spherical/Cylindrical			
Base	1194.00	310.68	9.35

Table 4. Relaxation times for each base case microstructure at the discharge rate of C/10 to a cutoff potential of 3V.

Architecture	Relaxation Time	Δc_s	V/A Ratio
Spherical Column 1			
Base	273.9	34.54	10.4
Spherical Column 2			
Base	447	52.01	7.01
Spherical Tree			
Base	1100.6	80.93	9.36
Spherical/Cylindrical			
Base	351.8	45.88	9.35

As is expected, there is a reduction in relaxation time as the discharge rate decreases because the discharge process becomes more uniform, as illustrated in Figures 37 and

38. However, despite the fact that the trends seen before are invariant with the discharge rate, in terms of structure ranking, the drop in relaxation time as a function of discharge rate does vary greatly from structure to structure, as depicted in the following figure.

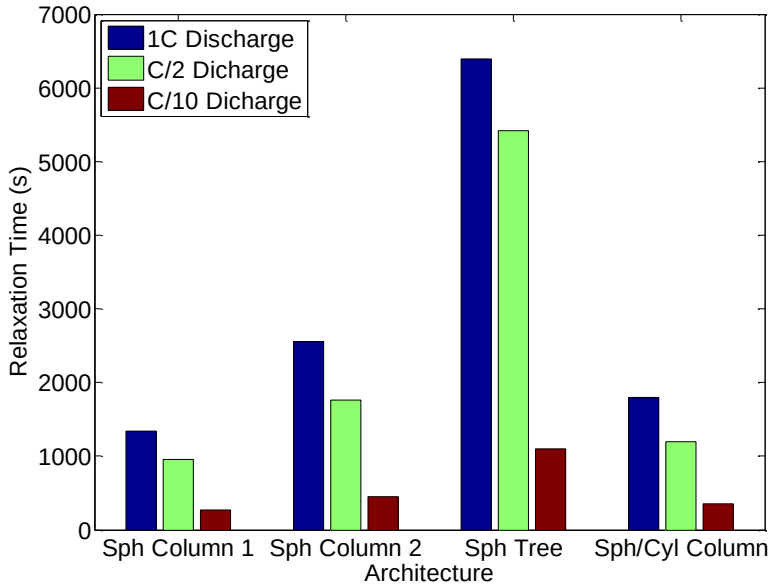

Figure 39. Relaxation times for the base cases for the different initial discharge rates of 1C, C/2, and C/10.

Clearly from the above comparison the Spherical Tree structure shows rapid reduction in relaxation time as the discharge rate decreases, however still displaying the largest relaxation times over all discharge rates. After relaxation, the structures were again recharged at the 1C rate to a cutoff potential of 4.1 volts.

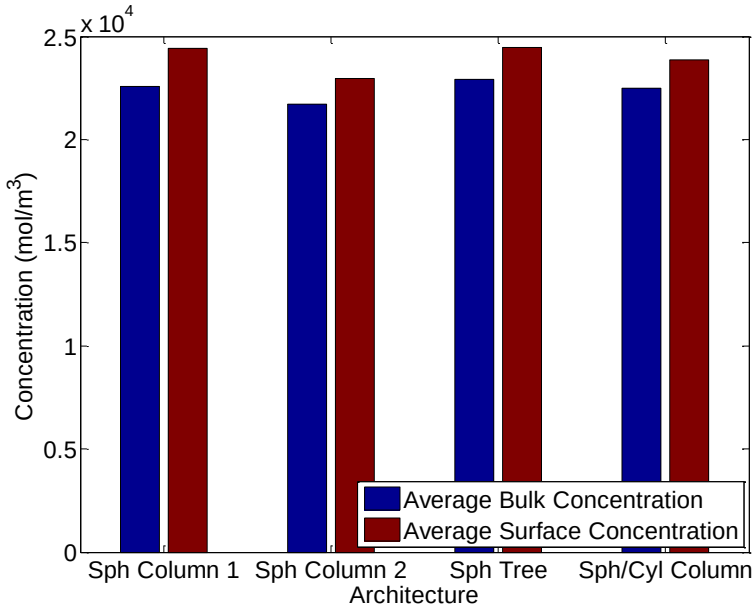

Figure 40. Surface and bulk concentrations in the base cases after relaxation and recharge to a potential of 4.1V from the initial C/2 discharge rate.

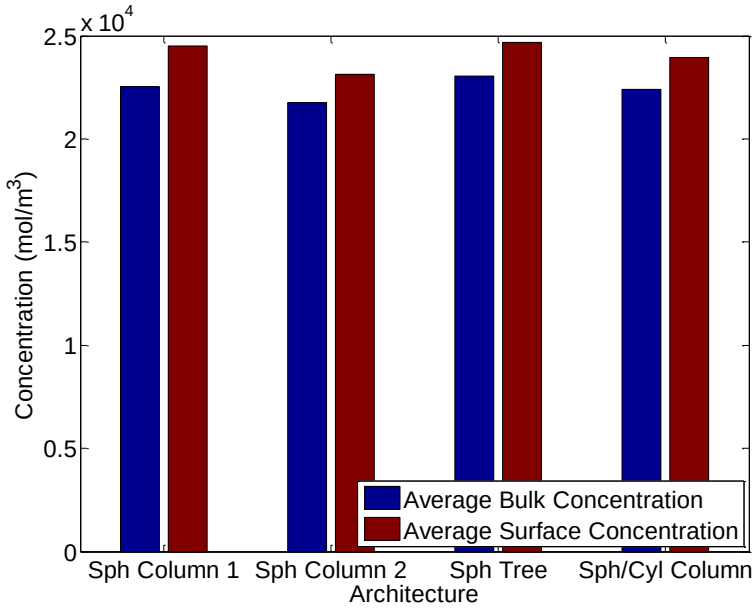

Figure 41. Surface and bulk concentrations in the base cases after relaxation and recharge to a potential of 4.1V from the initial C/10 discharge rate.

As is clear from Figures 40 and 41 the overall ranking of structure performance does not change over the discharge rate. However, it is to be noted that the Spherical Tree Structure does recover more material than any other over these cycles.

Additionally, with the decrease in discharge rate, the depth of discharge is a very important quantity for gauging the performance of a battery system. Shown in the following four figures are the average bulk concentrations for the base cases at each initial discharge rate and with two hours of relaxation time. For the majority of the structures there is an increase in depth of discharge with a decrease in discharge rate as expected. This occurs because with slower discharge rate, more lithium can diffuse to the surface to be extracted before the surface concentration drops to the point where the cutoff potential is reached. One of the more striking behaviors in the above results, however, is that seen in the Spherical Tree structure. As illustrated in Figure 44 there is initially a greater depth of discharge with a decrease in discharge rate, only to be followed by a larger concentration at the end of discharge for the C/10 initial discharge rate.

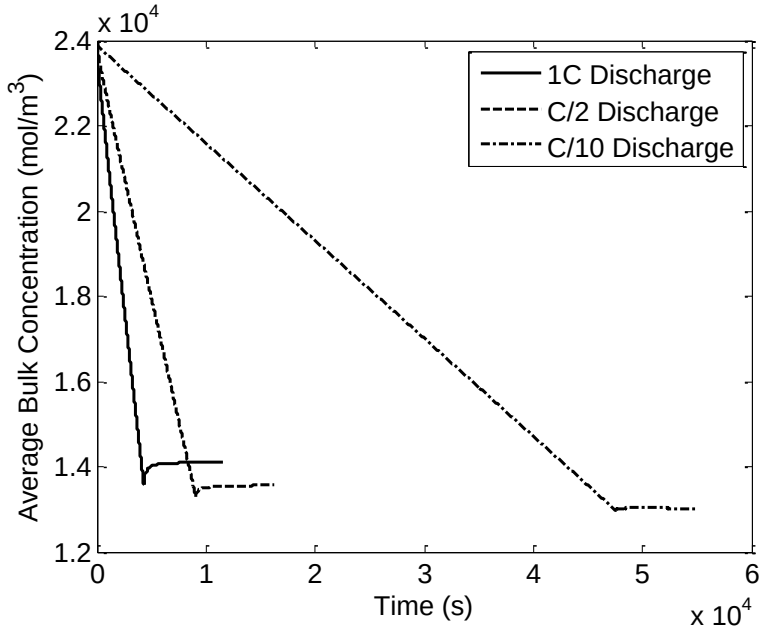

Figure 42. Depth of discharge for the Spherical Column 1 base structure at the 1C, C/2, and C/10 discharge rates.

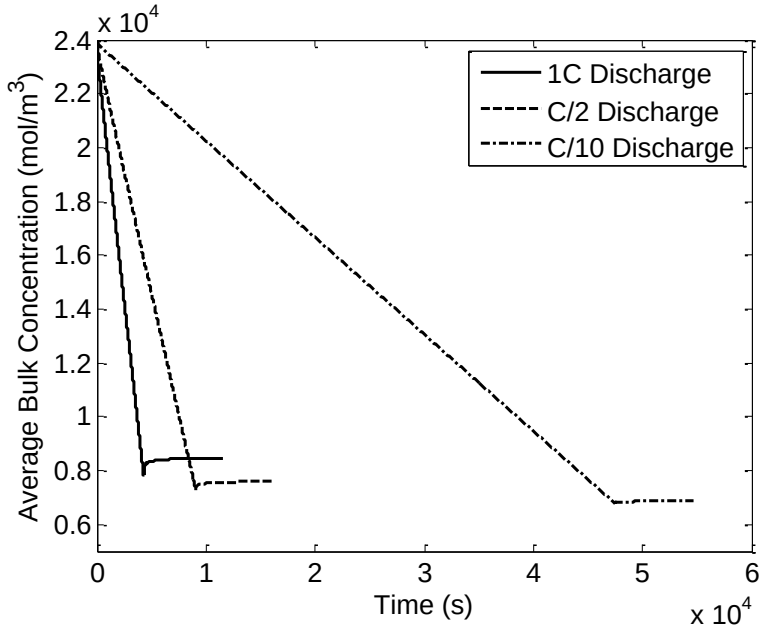

Figure 43. Depth of discharge for the Spherical Column 2 base structure at the 1C, C/2, and C/10 discharge rates.

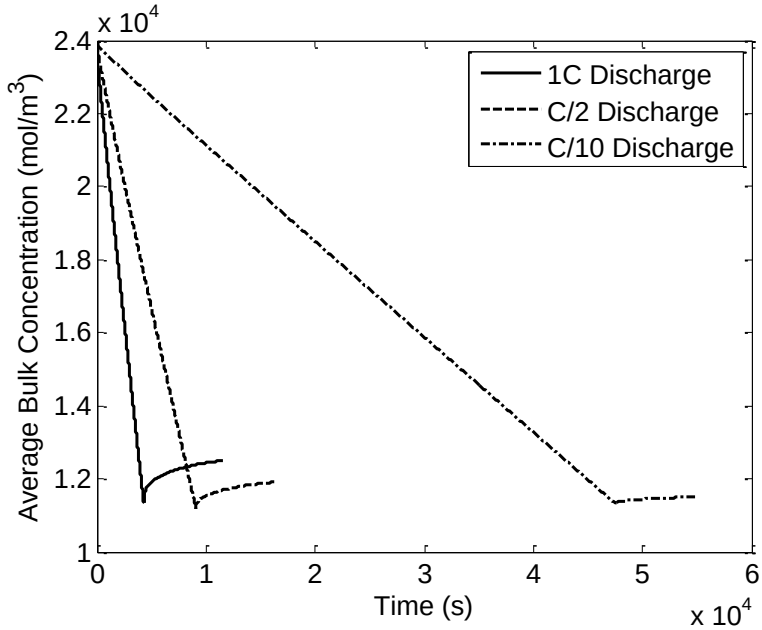

Figure 44. Depth of discharge for the Spherical Tree base structure at the 1C, C/2, and C/10 discharge rates.

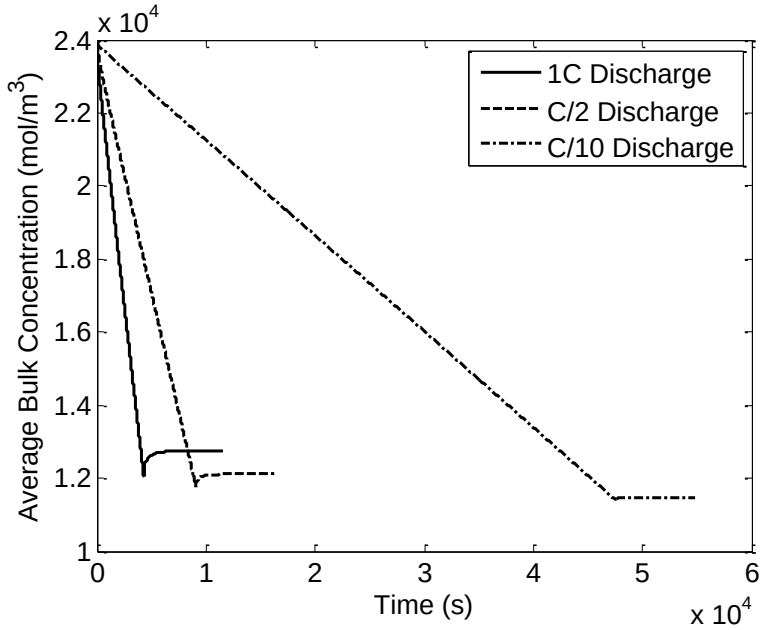

Figure 45. Depth of discharge for the Spherical/Cylindrical Column base structure at the 1C, C/2, and C/10 discharge rates.

This may indicate some rate dependence on the performance of this particular structure, and is therefore important for determining how best to utilize this particular geometry. However, the depth of discharge is more constant across all discharge rates for this structure when compared to the others.

Discharge Performance

Having collected surface concentration values from both the anode and cathode at each temporal moment, the voltage profiles over the discharge, relaxation, and recharge processes could be produced. It was found that all structures showed similar capacity values after discharge. However, it is important to consider how effective a particular structure is in terms of utilizing the amount of lithium it can store for a particular capacity. This is best shown by plotting voltage versus the state of charge (SOC). As the structure is depleted of lithium, the SOC will change according to the bulk concentration. Here, however, similar to capacity plots, it is important to remember that these plots need to be read as though the values on the horizontal axis are the amounts of SOC lost or spent. These are shown below.

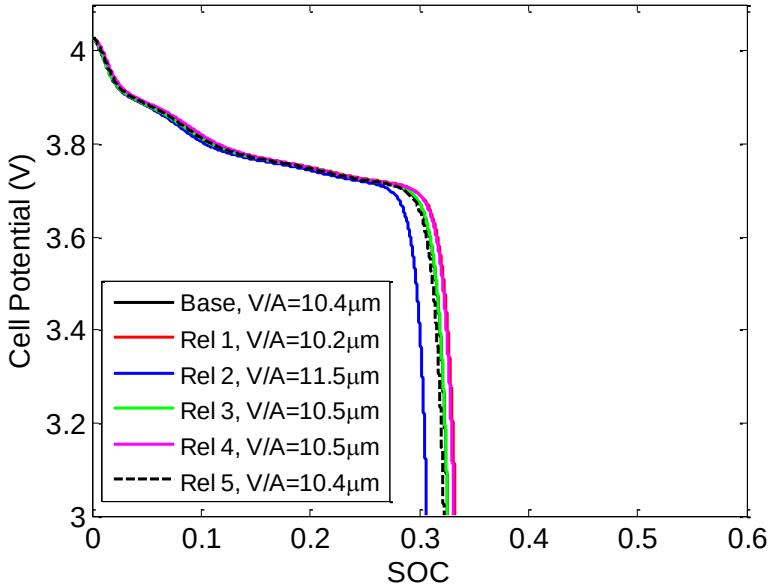

Figure 46. Discharge results for Spherical Column 1 base and realizations to a cutoff potential of 3V versus SOC.

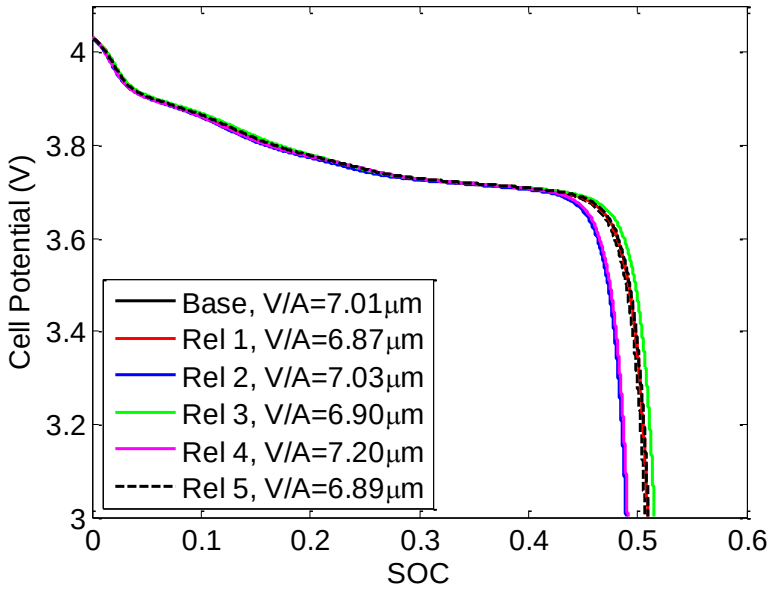

Figure 47. Discharge results for Spherical Column 2 base and realizations to a cutoff potential of 3V versus SOC.

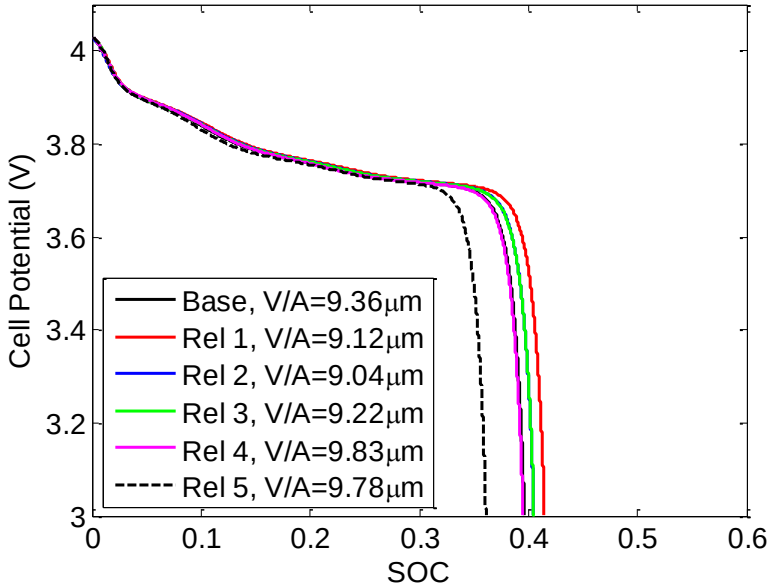

Figure 48. Discharge results for Spherical Tree base and realizations to a cutoff potential of 3V versus SOC.

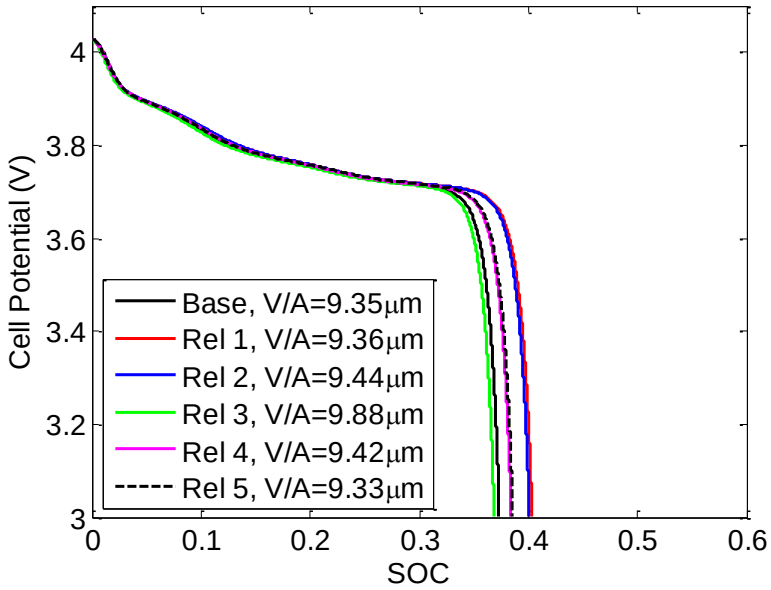

Figure 49. Discharge results for Spherical/Cylindrical Column base and realizations to a cutoff potential of 3V versus SOC.

These figures prove to be an excellent diagnostic tool for further gauging the performance of each of the above structures to each other. It is clear that although all of the structures show similar capacity after discharge, the utilization of available lithium is quite different. Spherical Column 1 shows the lowest SOC after discharge, meaning that although it has achieved similar capacity values to the other structures, it has been extremely inefficient. On the other hand, Spherical Column 2 shows the largest amount of SOC spent during discharge. This is likely attributable to the smaller volume to area ratios for these structures. A similar argument may also be appropriate for explaining the similarity in SOC behavior between the Spherical Tree and Spherical/Cylindrical Column structures. However, again, there are instances where volume to surface area ratios do not explain behavior, and many instances are similar to those for the bulk concentration analysis. For instance, Realizations 1 and 2 of the Spherical/Cylindrical Column structures do not distinguish themselves during discharge. This lack of distinction underlines the importance of analysis from multiple points of view, as completed previously. It is also to be noted that those structures showing increased particle overlap utilize less SOC over discharge, and this is due to a local drop in active surface area at those points. With this drop in surface area, lithium cannot be extracted as easily before the remaining surface area becomes depleted enough to force the structure to become fully discharged. This effect is also clear from the contour plots shown above.

For an initial discharge at the 1C rate, the base cases were allowed to relax for one hour, and then were recharged at the 1C rate. The voltage behavior for all base microstructures is shown below.

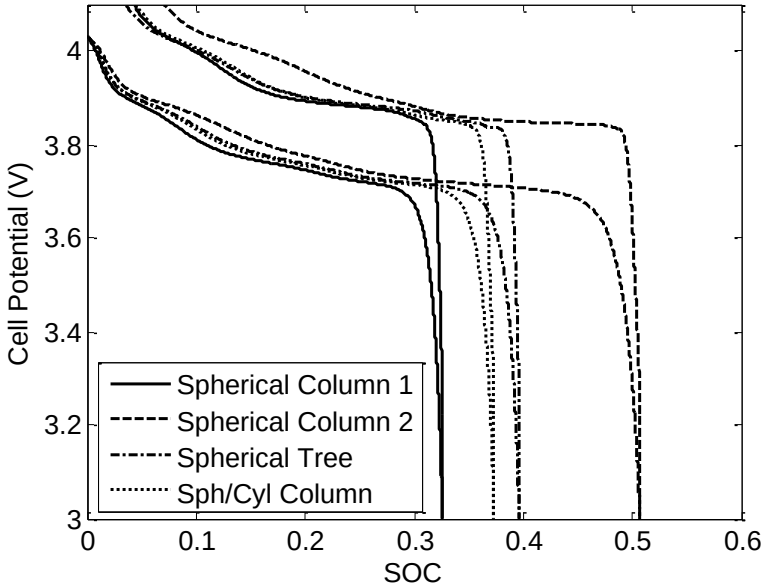

Figure 50. Discharge curves for the initial discharge at 1C, relaxation for one hour, and recharge at 1C, versus SOC.

It is to be noted that the ohmic drops between discharge and recharge have been drawn over to form a continuous curve over relaxation. These jumps occur in the data due to the fact that at the same surface and bulk state of charge, the flux is suddenly cutoff, causing the overpotential to instantaneously assume a zero value, while the equilibrium potential values remain the same. Analogously to what was discussed previously, the voltage versus SOC is shown in Figure 50 for the base structures. Here can be better visualized some of the effects of overall microstructure performance, as direct comparisons can now be drawn. As found previously, Spherical Column 2 shows the greatest use of available material for the same capacity, with Spherical Column 1 showing the least. It is to be noted however, that Spherical Column 2 does not recover the same amount of active material upon recharge, which is very detrimental to further use of the battery, as discussed with recharge efficacy.

The discharge curves over the full cycles for the base cases at the lower initial discharge rates are shown below, in Figures51 and 52.

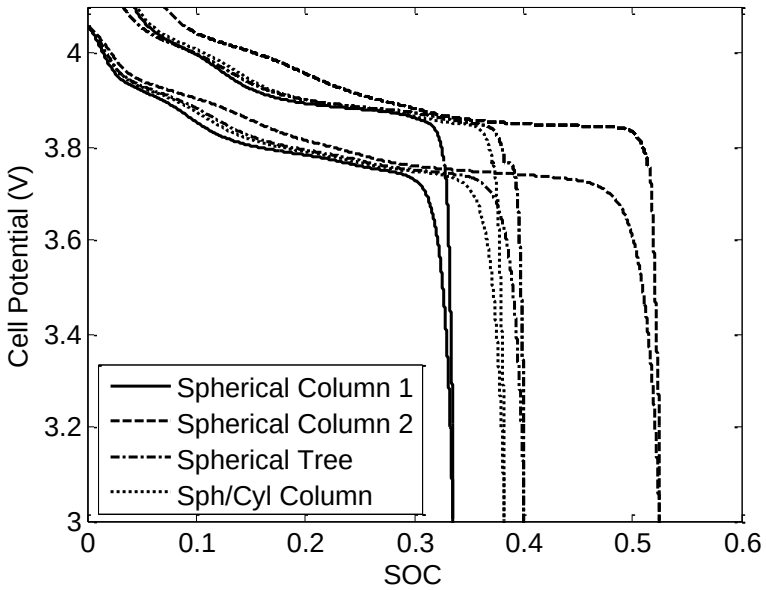

Figure 51. Discharge curve for the initial discharge at C/2, relaxation for one hour, and recharge at 1C versus SOC.

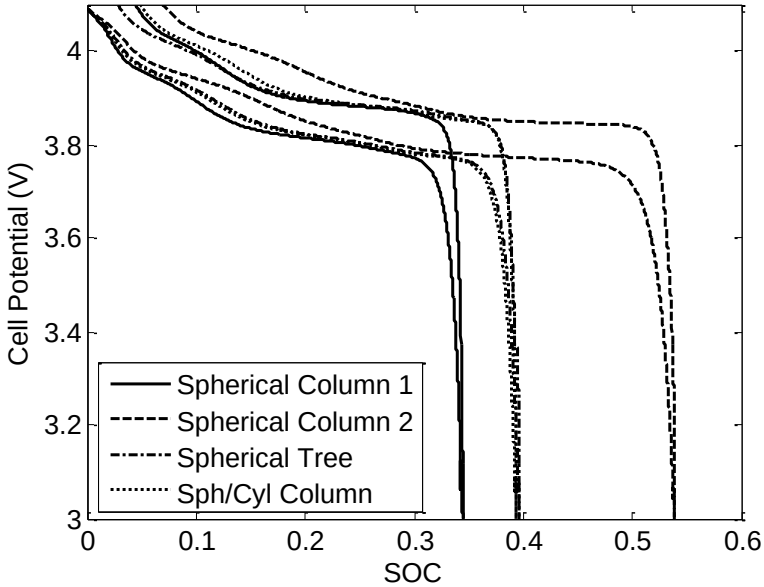

Figure 52. Discharge curve for the initial discharge at C/10, relaxation for one hour, and recharge at 1C versus SOC

The above figures clearly indicate the increase in capacity and the decrease in hysteresis as the discharge rate decreases. Also to be noted is the increase in recovered capacity as the discharge rate decreases. It is interesting to note that, as shown in Figure 52, the Spherical Tree and Spherical/Cylindrical Column structures have converged to show nearly identical behavior. However, the Spherical Tree structure does recover more material during recharge, as found during the recharge efficacy study, and utilizes nearly the same SOC over all discharge rates.

3D Cell Model

Having completed the above initial studies on anode architectures versus a single cathode particle, two 3D cell systems were developed with representative 3D electrodes serving for both the anode and cathode. The first of which was an Aperiodic structure

generated using the Spherical Column 1 base anode, and using another code written in MATLAB®, as given in Appendix B, to generate a cathode around this anode geometry. Having completed this cell, an interdigitated plate, or trench, design was generated where both the anode and cathode had equivalent volume to surface area ratios as the first cell design. This geometry was also restrained to a similar cell footprint as the former. These two geometries are illustrated in below, and relevant quantities are given in the following table.

Table 5. Geometrical properties of the 3D cell models.

Architecture	Volume(m^3)	Surface	V/A Ratio
Aperiodic			
Anode	2.34×10^{-13}	2.30×10^{-8}	10.2
Cathode	1.53×10^{-13}	1.93×10^{-8}	7.96
Interdigitated Plates			
Anode	1.95×10^{-13}	1.91×10^{-8}	10.2
Cathode	1.26×10^{-13}	1.58×10^{-8}	7.99

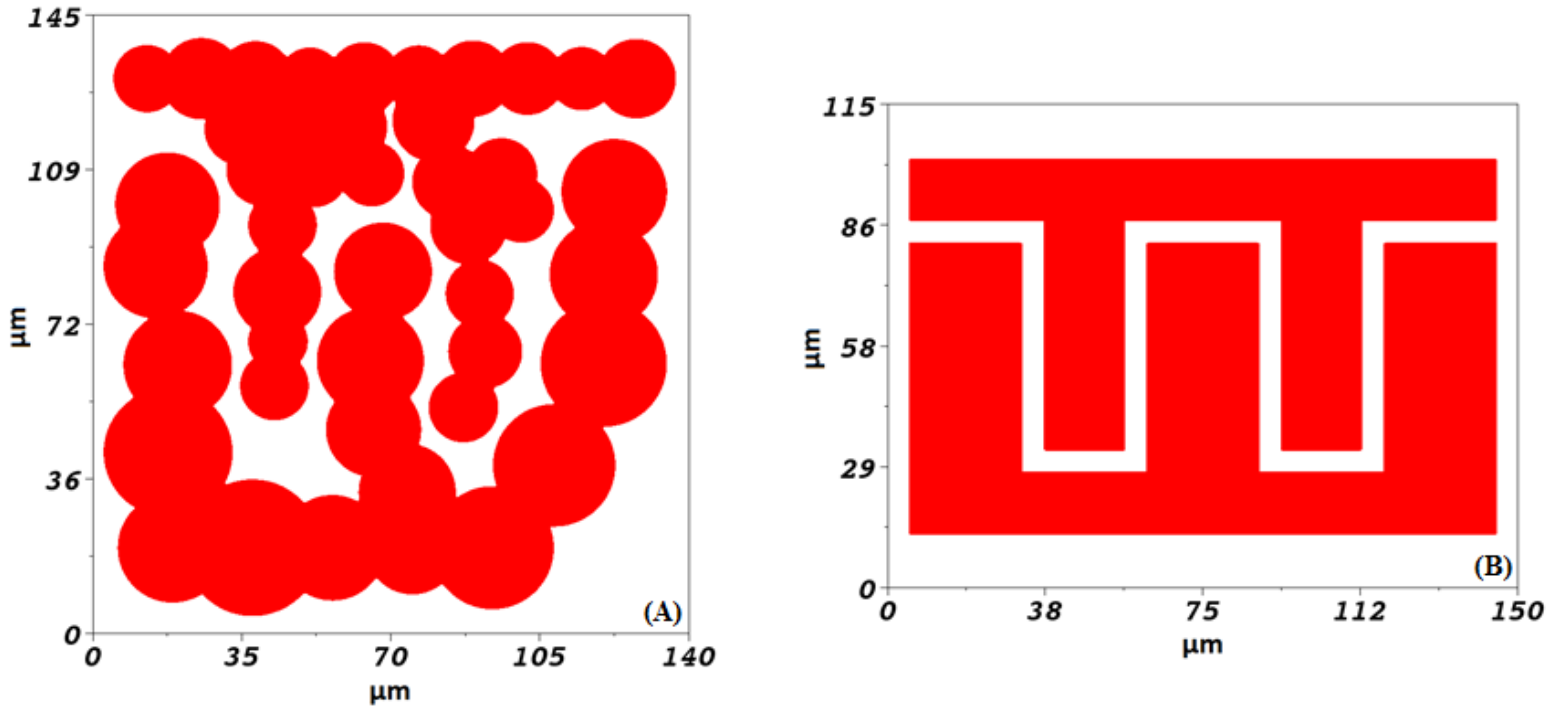

Figure 53. Aperiodic cell (A), and Interdigitated Plate cell (B). In both cases the bottom structure serves as the anode, and all dimensions are in micron.

Concentration and Relaxation Behaviors

Similarly to the anode morphology study, the structures were discharged at the 1C rate, and then allowed to relax for two hours after the cutoff potential was reached. The bulk and surface concentration values over this process as well as respective contour plots, are shown below. As is clear from the above figures, the Interdigitated Plate design has a longer discharge time before cutoff is reached, and additionally reaches a higher depth of discharge. This is especially striking, as this structure has a lower volume than the aperiodic structure. What this indicates, is that the transport phenomena occurring within the Interdigitated Plate design is far more efficient than the other. Considering the contour plots, this is easily noted. Facile transport throughout the structure is maintained by large flux areas, as opposed to the Aperiodic design, which has choke points between spherical particles.

It is to be noted at this point that during simulations, unphysical concentration cells of large lithium depletion were developed in both 3D cell configurations, despite the fact that bulk SOC values were in allowable ranges as provided in the literature(73).

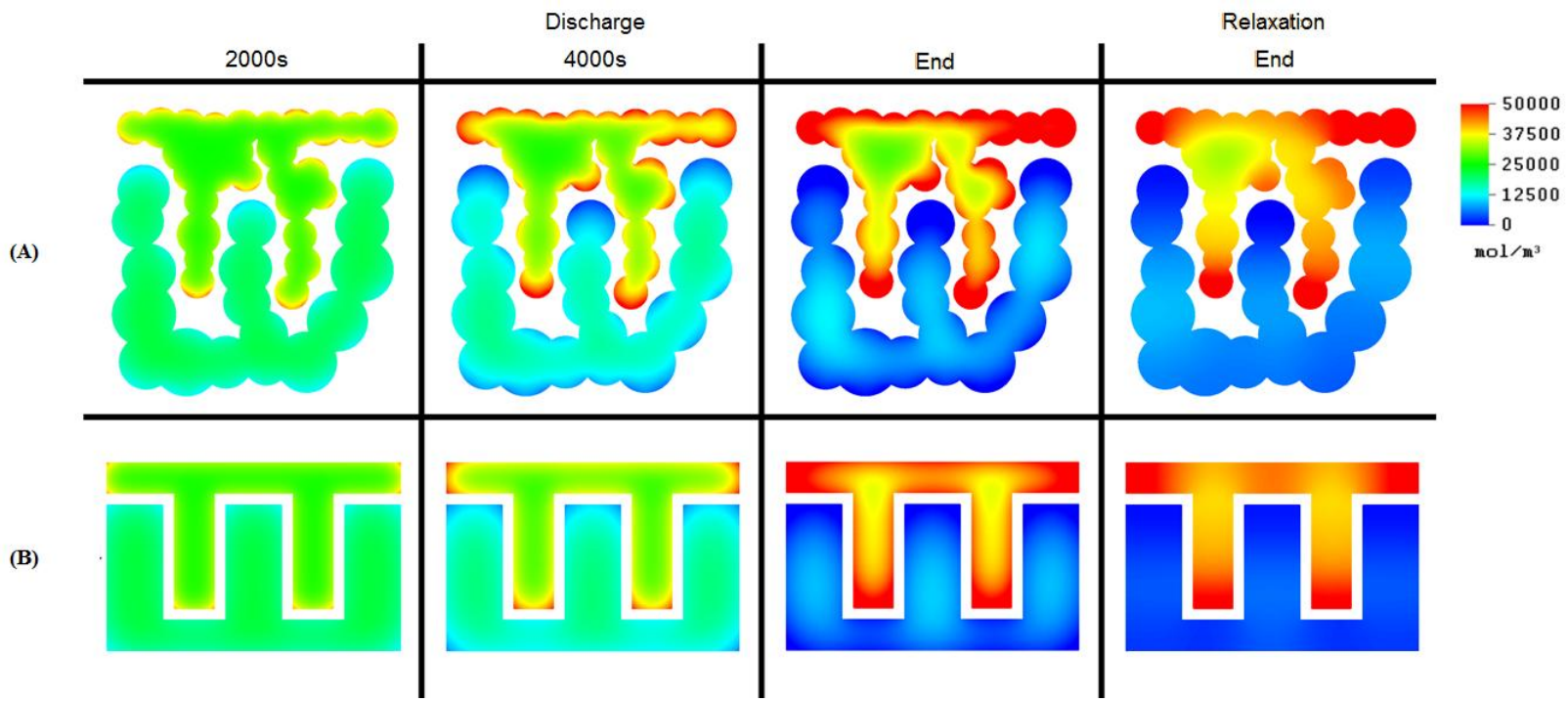

Figure 54. Aperiodic cell (Row A), and Interdigitated Plate cell (Row B) over discharge at 1C and relaxation. All dimensions in micron and all concentration values in mol/m^3. (To Scale from Figure 53)

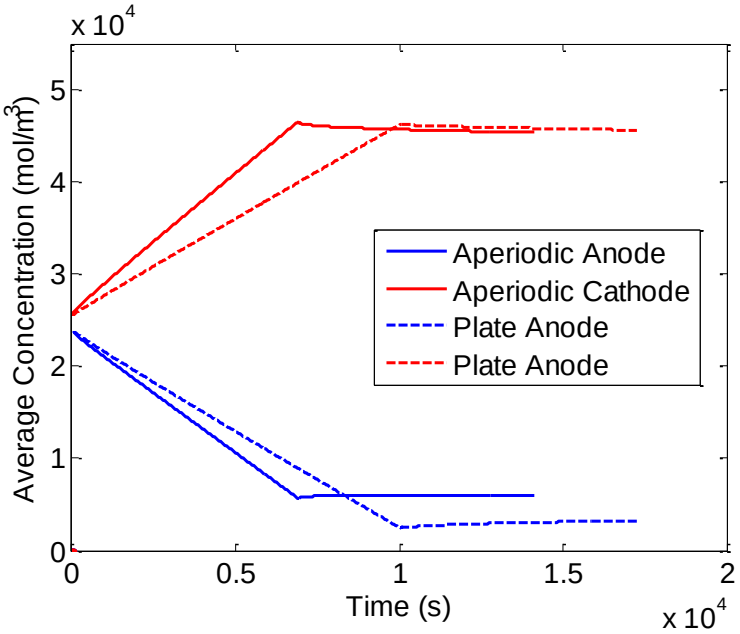

Figure 55. Bulk concentration curves for both 3D designs for a discharge at 1C to a cutoff potential of 3V and relaxation for two hours.

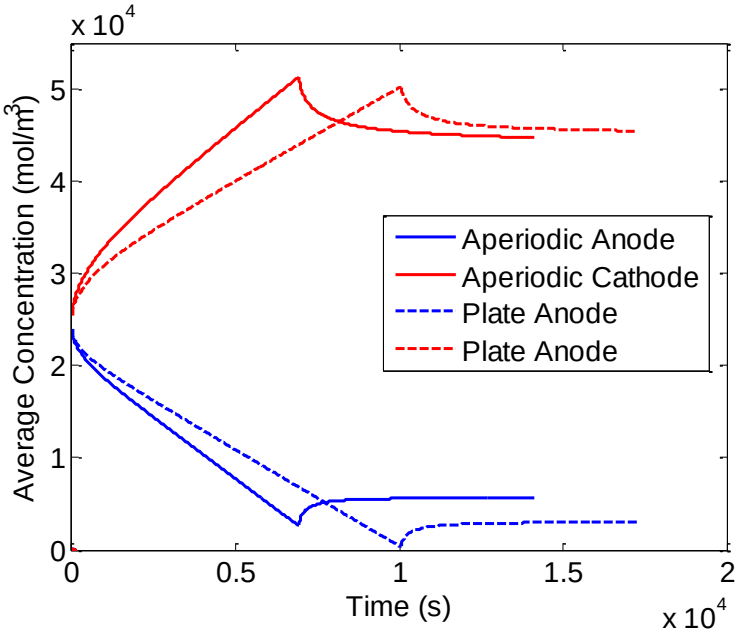

Figure 56. Surface concentration curves for both 3D designs for a discharge at 1C to a cutoff potential of 3V and relaxation for two hours.

These regions of low concentration were detected at the topmost spherical particles in the Aperiodic cell as well as at the sharp corners in the Interdigitated Plate cell. This behavior is a direct consequence of the constant flux assumption applied to the geometry. However, these results do not invalidate the value of this study, as this confirms many of the behaviors previously identified. Particle isolation at the top most spherical particles causes large depletion of lithium in these areas if particle overlap, as seen before, is not present. Additionally, Zadin *et. al*(53, 54) noted the exact same behavior in the trench design. What is important to mention, however, is that their study was performed with a full model, which included transport in the electrolyte. Here, using first principle approximations, the same conclusions were drawn at a greatly reduced computational cost. In order to mitigate this behavior in the interdigitated plate structure, rounding the sharp corners was found to be an effective method in the literature(53).

To compare the structures, relaxation behavior was considered over the two hour period. The results over this period for both structures and their respective anode and cathode are shown below.

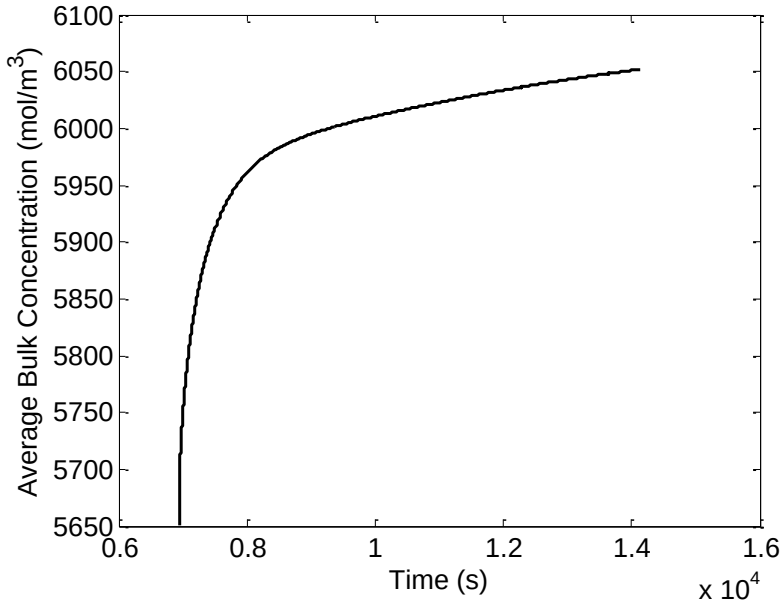

Figure 57. Relaxation in bulk concentration for the Aperiodic anode.

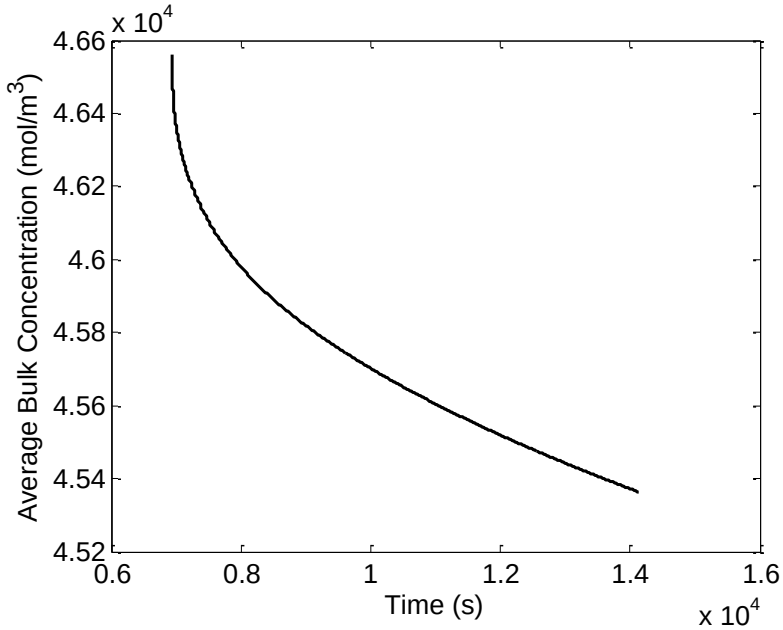

Figure 58. Relaxation in bulk concentration for the Aperiodic cathode.

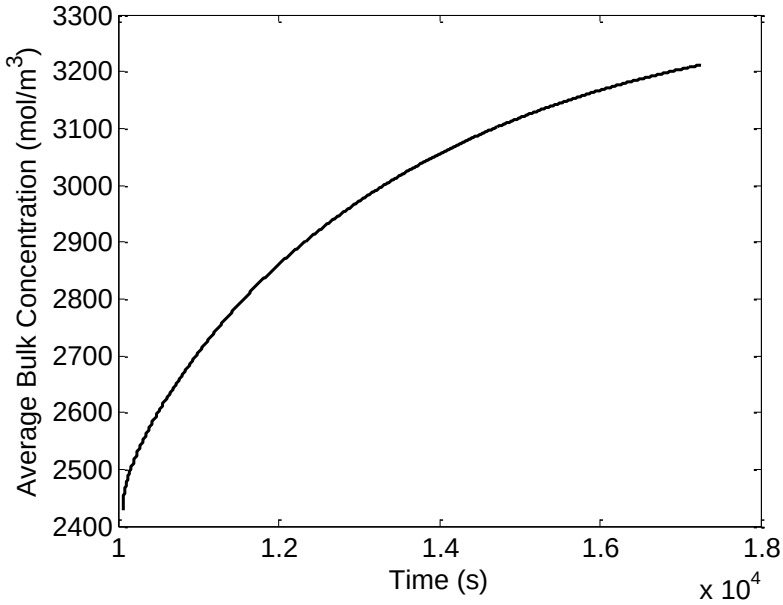

Figure 59. Relaxation in bulk concentration for the Interdigitated Plate anode.

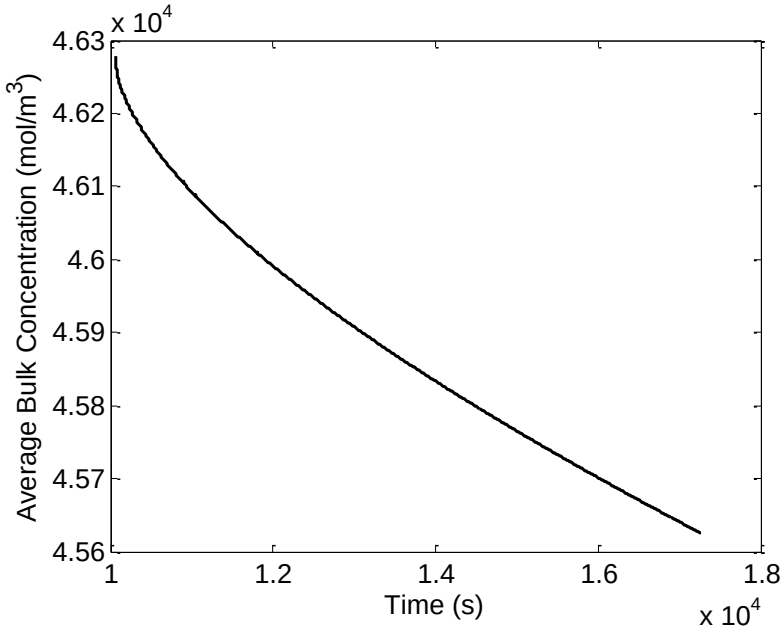

Figure 60. Relaxation in bulk concentration for the Interdigitated Plate cathode.

The above structures do not relax within two hours using the definition applied previously for the anode morphologies. This is especially true for the cathode structures, likely due to the fact that the diffusion coefficient is four times smaller. Considering the anodes, there is a far greater change in the bulk concentration for the Interdigitated Plate design over the same time period. This is again due to the geometry of the structure, as it promotes facile transport for redistribution, and this is evident in Figure 54(B), which shows that the plate structures have a far more uniform distribution over discharge and after relaxation.

Discharge Performance

Similarly to what was completed before, surface concentration values were used to compute cell potential over the course of discharge. These results are illustrated below, and here especially is noted a difference in the voltage versus capacity behavior.

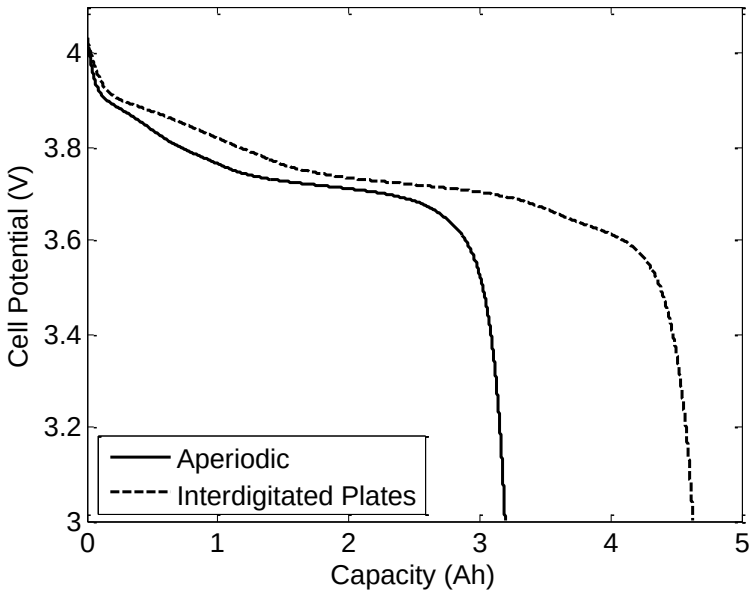

Figure 61. Discharge curve for the Aperiodic and Interdigitated Plate structures at 1C versus capacity.

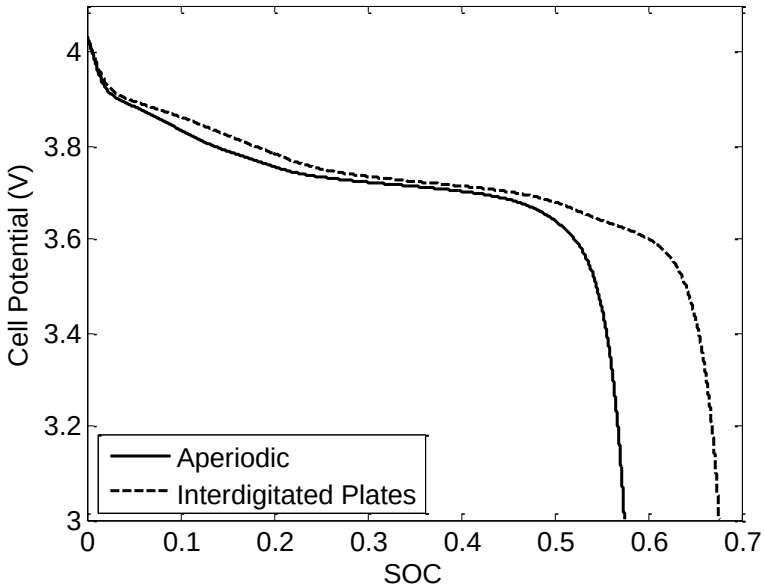

Figure 62. Discharge curve for the Aperiodic and Interdigitated Plate structures at 1C versus SOC.

With regard to the above results versus capacity, the aperiodic structure showed far larger capacity than the Spherical Column 1 Base case studied previously, despite the fact that the anode structure is the same. This is due to the increased capacity of the cathode, as it is no longer treated as a single spherical particle. When the Interdigitated Plate cell is considered, there is a very large difference in capacity relative to the Aperiodic structure, despite equivalent volume to surface area ratios. This finding is similar to what was discovered in the anode analysis, where, in many cases, global parameters were not sufficient to describe or predict cell behavior. Here again it is surprising to find that the plate design shows both increased capacity and utilization of active material, despite having a lower volume. The above results illustrate the importance of this behavior more clearly than what was discussed above. Facilitating

transport within these 3D structures has a profound effect on performance for a given volume of material, and so proper design must take these considerations into account.

Sensitivity Study

In order to validate the results acquired in the above study, a mesh and tolerance refinement study was completed on the Spherical Tree base structure, as well as the anodes of the two 3D cell geometries at the 1C rate over discharge. As noted in the procedure in Appendix B, the geometry in MFiX® software is controlled not only by the mesh size specified, but the value TOL_F, which specifies the tolerance at which the desired geometry intersects the Cartesian Grid.

To begin, all 3D anode morphology studies were completed on an equivalent mesh with 100 cells in both directions, with spatial extents of 240μm and 150μm in the horizontal and vertical directions, respectively. Additionally, a TOL_F value of 1×10^{-16} was employed. To determine the kinds of variability possible, the number of cells was both increased and decreased by approximately a factor of 2 relative to the base case (or 70 cells for the coarse grid, and 130 cells for the fine grid). Additionally, the TOL_F values were altered to 1×10^{-13} and 1×10^{-20} while keeping the base mesh. The results of the discharge process, in terms of average bulk and surface concentrations are displayed in the following figures.

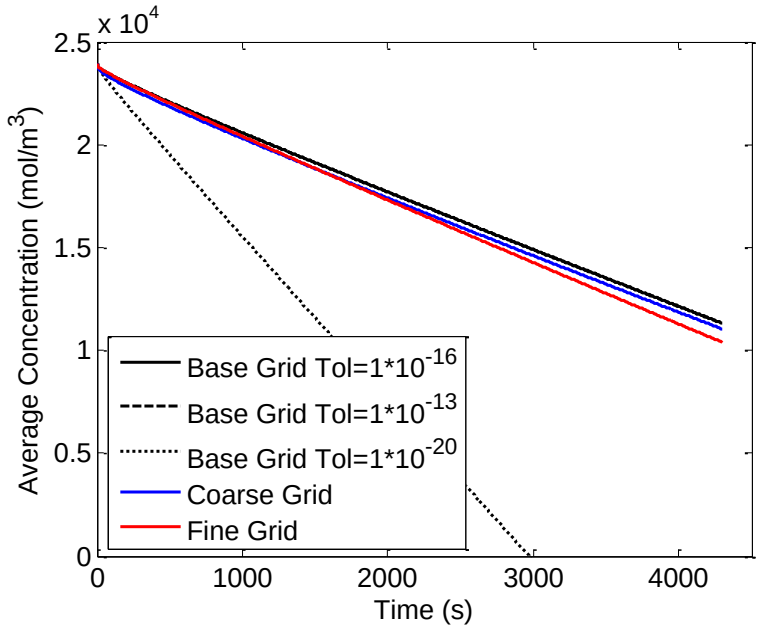

Figure 63. Average bulk concentration over discharge at 1C for the Spherical Tree base structure with various refinements to mesh and TOL_F.

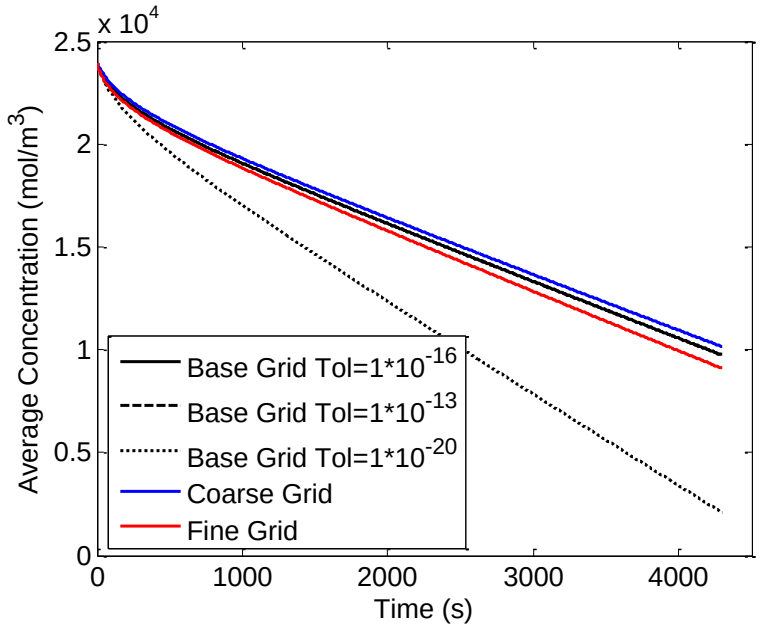

Figure 64. Average surface concentration over discharge at 1C for the Spherical Tree base structure with various refinements to mesh and TOL_F.

Immediately noticeable from the above results, is the variance in the results utilizing a TOL_F value of 1×10^{-20}. Upon further inspection of the data, these results were deemed to be erroneous, especially when the convergence behavior of the other studies was considered. The other results show excellent agreement over the discharge range, particularly considering both the scale and length of the simulation.

Regarding the Aperiodic and Interedigitated Plate cells, base simulations were performed on meshes with 200 cells on spatial extents of 150μm in both directions. As completed previously, the base TOL_F value was set to 1×10^{-16}. For the refinement studies, the number of cells was both increased and decreased by approximately a factor of 2, and the TOL_F values were varied similarly as previously. Here, only the anode has been analyzed. Shown in the following figures are analogous figures as previously shown for the anode morphologies for both the Aperiodic and Interdigitated Plate 3D cells. These results indicate more variability in the data, especially at the points where the cutoff potential was reached, due to the range at which the results cover. However, it is to be mentioned that unrealistic values for cell concentrations were detected in these simulations, as described previously, and these excessive gradients in the geometry may have caused increased variability. It is notable, however, that unlike in the 3D anode sensitivity study, those simulations with varying TOL_F values agree well with each other in these 3D cell designs.

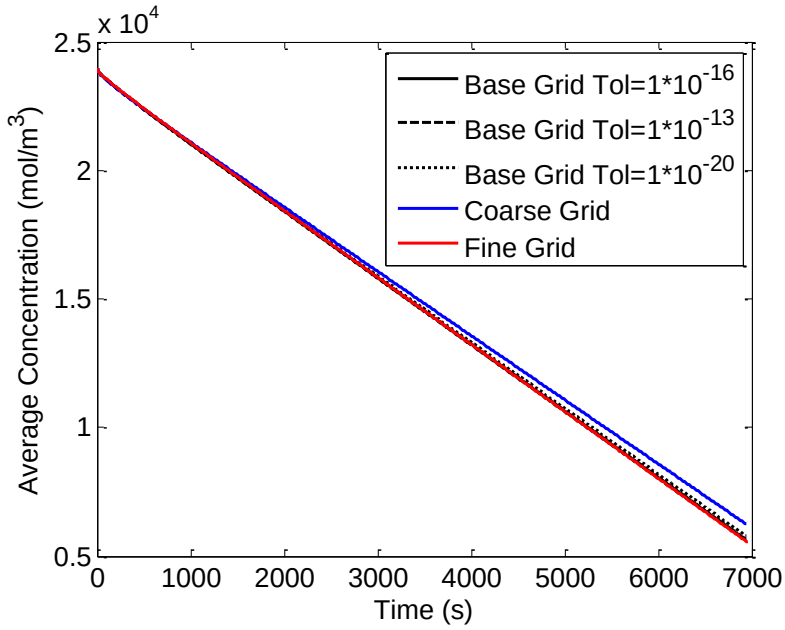

Figure 65. Average bulk concentration over discharge at 1C for the Aperiodic anode with various refinements to mesh and TOL_F.

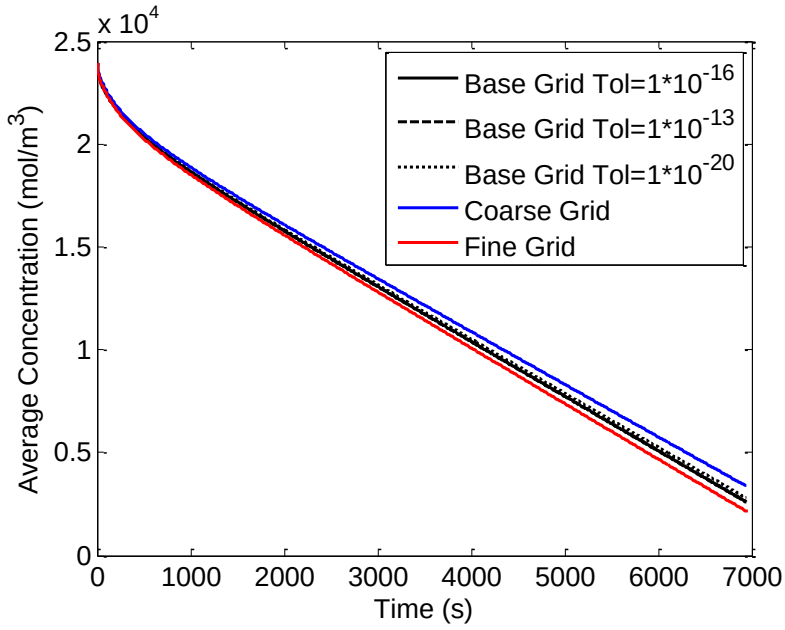

Figure 66. Average surface concentration over discharge at 1C for the Aperiodic anode with various refinements to mesh and TOL_F.

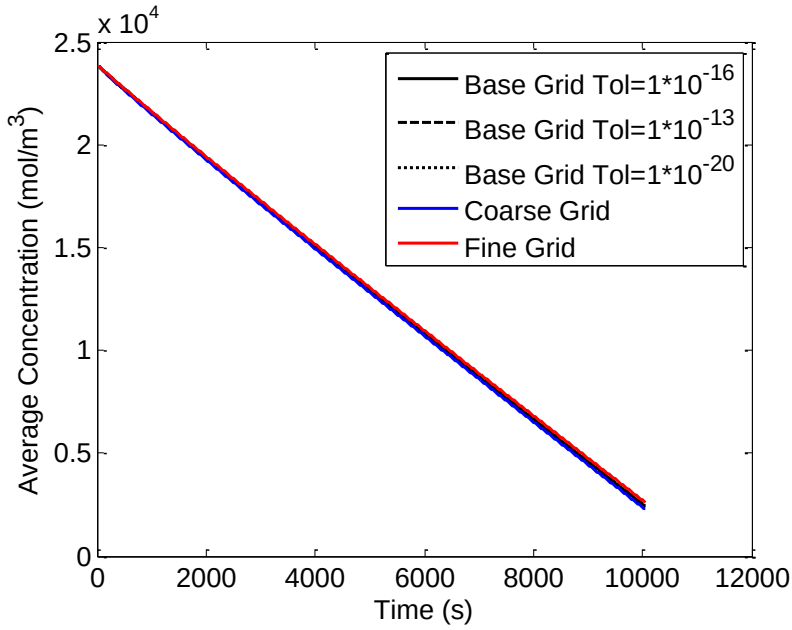

Figure 67. Average bulk concentration over discharge at 1C for the Interdigitated Plate anode with various refinements to mesh and TOL_F.

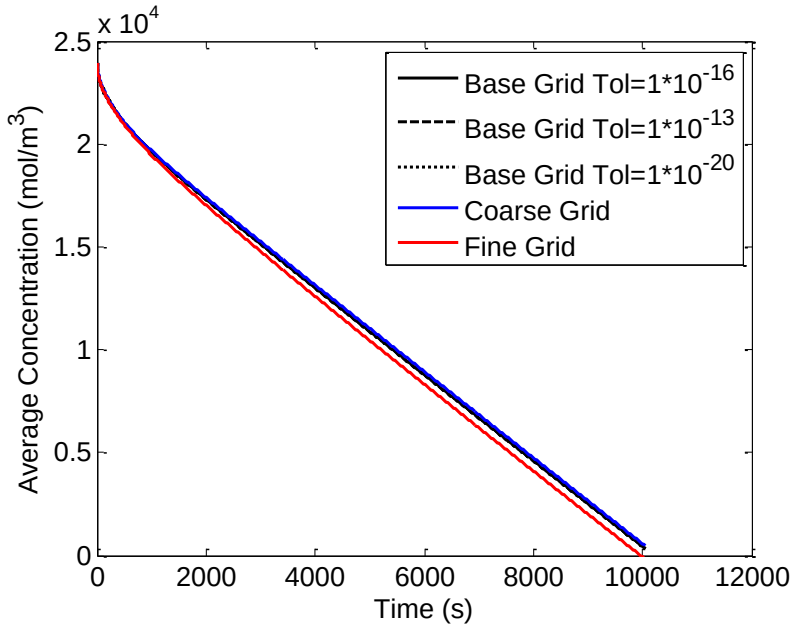

Figure 68. Average surface concentration over discharge at 1C for the Interdigitated Plate anode with various refinements to mesh and TOL_F.

CHAPTER IV

SUMMARY AND OUTLOOK

Energy storage has been a growing field of interest and importance for both society and the research community for the past several decades. Lithium ion batteries in particular have attracted a large amount of attention, as these devices are now widespread in everything from portable electronics to hybrid electric vehicles. In order for continued progress to be made in this area, computational and mathematical tools will play a key role in the continued search for improvement in these systems. The objective of the above study was to use first principle approximations to quantify the performances of various electrode geometries for use in 3D battery design, which shows promise as the next step towards more efficient battery systems in terms of capacity and performance.

Methodology

The above objective was satisfied by first developing several MATLAB® codes to generate aperiodic anode structures, which included variability in particle size, shape, and overall morphology. Additionally, further codes were developed to introduce perturbed realizations of each base morphology, and to produce 3D cells using a chosen morphology for the anode. MFiX® software, an open source, generic, computational code developed by the National Energy Technology Laboratory (NETL) was used to simulate the above structures under conditions from literature. The results of these

studies yielded significant results by consideration of average bulk and surface concentrations, relaxation behavior, as well as cycle performance.

Results

Discharge profiles at the 1C, C/2, and C/10 rates were performed for the anode morphologies and results indicated that global parameters like volume to surface area ratio, as well as actual geometry, were both equally important for characterizing behavior. Expected increases in capacity with decreases in discharge rate were evident, as well as a decrease in recharge hysteresis.

Furthermore, the variance in structure behavior under specified perturbed conditions were analyzed, and it was discovered that by increased branching in the aperiodic structures, as well as the inclusion of cylindrical particle shapes, produced the greatest differences in bulk concentration behavior. This finding was specifically important, as the inclusion of a wider particle size range did not produce such large differences. However, for surface concentration, those structures with greater particle distribution did show greatest variance.

Relaxation studies of the structures proved to be some of the more useful for gauging structure behavior, especially as volume to surface area ratios were shown to be incapable of doing so. It was concluded that increased particle isolation due to branching in the structures caused large gradients to develop, and thus relaxation times were accordingly increased. Additionally, with the inclusion of greater particle size variability, relaxation times also showed a wide range of variability, but also a general increase relative to structures with more uniform particle distribution. This was

attributed to the uneven discharge between large and small particles, thus causing redistribution of concentration within the structures to take significantly more time. With the inclusion of cylindrical particles in the electrode geometry, a slightly larger relaxation time was noted relative to those structures with solely spherical particles. This was explained by considering that cylindrical particles, like additional branches, cause increase particle isolation due to the fact that diffusion distances for redistribution are increased as the primary mode of transport is along the cylindrical axis.

After one hour of relaxation, all structures were recharged at the 1C rate to a cutoff potential of 4.1V. After the recharge process was completed, differences between average bulk and surface concentrations were considered. It was determined that structures with larger particle size distributions were ineffective at recharging when initially discharged at the 1C rate. Further, upon visual investigation of the results, it was found that with the inclusion of cylindrical particles more uniform concentrations were developed in the core of the structure, due to the decreased diffusion length along the radial direction of the cylindrical axis. This finding is especially important, as it is an attribute that might be exploited in the design of new geometries. Considering these differences after initial discharge rates of C/2 and C/10 another important discovery was realized. With the decrease in initial discharge rate, there was a reversal in performance for the structure with additional branching. This structure was able to obtain uniform and greater concentrations after recharge. This indicates the need for consideration in electrode geometry as it pertains to a specific application, as some geometries may only be suitable for low discharge rates over long periods of time.

Performance curves, in terms of voltage and either capacity or SOC were generated for the structures, and these were compared to yield significant findings. While all structures showed equivalent capacity over the discharge period, the utilization of available material showed large differences. The smallest structures investigated showed the greatest utilization, while the largest showed the poorest. It was additionally found that structures did show rate dependence on the utilization of material, and so careful consideration of these effects must be included in the design process.

After completion of the anode morphology study, two 3D cell geometries were developed with equivalent volume to surface area ratios. Simulated at a discharge rate of 1C, these structures showed large increases in capacity. This behavior was attributed to the fact that the cathode was no longer considered as a single spherical particle as with the anode morphologies, but rather as a full 3D electrode. During these simulations, severe lithium deprived zones were detected in certain regions of the structure. While unphysical, these results indicated that with the inclusion of 'sharp' edges in the microstructure, such lithium depletion may occur. This same result was concluded by other studies in the literature using more complicated computational models. This thus ultimately underscored the value of the first principle approximations used in this study, as computational time and expense were lessened only to arrive at similar conclusions.

Future Work

The results of this study should ultimately be used to target advantageous geometries for use in fuller, more developed models that include the other electrochemical behaviors not present here. The above study has used two dimensional

computational simulations to represent three dimensional slices of proposed geometry. Inclusion of these geometries in full, three dimensional models will be required to correct for the two dimensional approach employed here. This study has additionally assumed uniform current densities over the structures involved, and has neglected transport within the electrolyte. While suitable for low discharge rates, higher discharge rates will require knowledge of transport in the electrolyte as this becomes important, especially with regard to Butler-Volmer kinetics which governs current distribution over the structure.

Finally, experimental results are required to validate any numerical solution, whether it is a full cell simulation or an approximate approach as employed here. Representative architectures similar to those studied will need to be cycled over a wide range of discharge rates to generate results that can be compared against developed models. It is to be noted that these comparisons will likely be on the system level only, such as discharge performance and bulk behaviors. Once verification of system level results is complete, models can then be used to better investigate behavior at the scales considered in this study, which will lend further insight into architecture performance.

REFERENCES

1. C. M. Doyle, Ph. D. Thesis, University of California, Berkeley, CA (1995).

2. E. J. Cairns and P. Albertus, *Annual Review of Chemical and Biomolecular Engineering*, **1**, 299 (2010).

3. M. Winter and R. J. Brodd, *Chemical Reviews*, **105**, 1021 (2005).

4. S.-W. Cha, R.P O'Hayre, W. Colella and F. B. Prinz, *Fuel Cell Fundamentals*, John Wiley & Sons, New York (2006).

5. P. P. Mukherjee, S. Pannala and J. A. Turner, in *Handbook of Battery Materials*, 2nd ed., Wiley-VCH, Weinheim, Germany (2011).

6. D. Bernardi, E. Pawlikowski and J. Newman, *Journal of The Electrochemical Society*, **132**, 5 (1985).

7. J. C. Slattery, *Aiche J.*, **15**, 866 (1969).

8. S. Whitaker, *Industrial & Engineering Chemistry*, **61**, 14 (1969).

9. J. Bear and J. M. Buchlin, *Modeling and Applications of Transport Phenomena in Porous Media*, Kluwer Academic Publishers, Boston (1991).

10. J. C. Slattery, *Momentum, Energy and Mass Transfer in Continua*, R. E. Krieger Publishing Company, New York (1981).

11. W. B. Gu and C. Y. Wang, *Journal of The Electrochemical Society*, **147**, 2910 (2000).

12. P. M. Gomadam, J. W. Weidner, R. A. Dougal and R. E. White, *Journal of Power Sources*, **110**, 267 (2002).

13. K. Smith and C.-Y. Wang, *Journal of Power Sources*, **161**, 628 (2006).

14. V. Srinivasan and C. Y. Wang, *Journal of The Electrochemical Society*, **150**, A98 (2003).

15. W. B. Gu and C. Y. Wang, *Lithium Ion Batteries, Proceedings*, 748, (2000).

16. J. Newman, K. E. Thomas, R. M. Darling, in *Advances in Lithium-Ion Batteries*, Kluwer Academic/Plenum Publishers, New York (2002).

17. J. Newman, K.E. Thomas-Alyea, *Electrochemical Systems*, Wiley Interscience, Hoboken, N.J. (2004).

18. M. Doyle, J. Newman, A. S. Gozdz, C. N. Schmutz and J.-M. Tarascon, *Journal of The Electrochemical Society*, **143**, 1890 (1996).

19. S. Santhanagopalan, Q. Guo, P. Ramadass and R. E. White, *Journal of Power Sources*, **156**, 620 (2006).

20. B. S. Haran, B. N. Popov and R. E. White, *Journal of Power Sources*, **75**, 56 (1998).

21. G. Ning and B. N. Popov, *Journal of The Electrochemical Society*, **151**, A1584 (2004).

22. W. B. Gu, C. Y. Wang and B. Y. Liaw, *Journal of The Electrochemical Society*, **145**, 3418 (1998).

23. C. Y. Wang and V. Srinivasan, *Journal of Power Sources*, **110**, 364 (2002).

24. L. Shengyi, *Solid State Ionics*, **177**, 53 (2006).

25. V. R. Subramanian, J. A. Ritter and R. E. White, *Journal of The Electrochemical Society*, **148**, E444 (2001).

26. V. R. Subramanian, V. D. Diwakar and D. Tapriyal, *Journal of The Electrochemical Society*, **152**, A2002 (2005).

27. V. R. Subramanian and R. E. White, *Journal of Power Sources*, **96**, 385 (2001).

28. Q. Zhang and R. E. White, *Journal of Power Sources*, **165**, 880 (2007).

29. V. Ramadesigan, V. Boovaragavan, J. C. Pirkle, Jr. and V. R. Subramanian, *Journal of The Electrochemical Society*, **157**, A854 (2010).

30. A. Romero-Becerril and L. Alvarez-Icaza, *Journal of Power Sources*, **196**, 10267 (2011).

31. D. Portnyagin, *Russian Journal of Electrochemistry*, **46**, 144 (2010).

32. J. Christensen and J. Newman, *Journal of Solid State Electrochemistry*, **10**, 293 (2006).

33. J. Christensen and J. Newman, *Journal of The Electrochemical Society*, **153**, A1019 (2006).

34. J. Christensen, *Journal of The Electrochemical Society*, **157**, A366 (2010).

35. Y.-T. Cheng and M. W. Verbrugge, *Journal of Applied Physics*, **104**, 083521 (2008).

36. Y.-T. Cheng and M. W. Verbrugge, *Journal of The Electrochemical Society*, **157**, A508 (2010).

37. G. Sikha, M. Guo and R. E. White, *Journal of The Electrochemical Society*, **158**, A122 (2011).

38. G. Sikha, K. Kumaresan and R. E. White, *Journal of The Electrochemical Society*, **155**, A164 (2008).

39. D. Guyomard and J. M. Tarascon, *Journal of The Electrochemical Society*, **139**, 937 (1992).

40. J. M. Tarascon and D. Guyomard, *Solid State Ionics*, **69**, 293 (1994).

41. J. M. Tarascon and D. Guyomard, *Journal of The Electrochemical Society*, **138**, 2864 (1991).

42. D. Guyomard and J. M. Tarascon, *Journal of The Electrochemical Society*, **140**, 3071 (1993).

43. M. W. Verbrugge and B. J. Koch, *Journal of The Electrochemical Society*, **143**, 24 (1996).

44. M. W. Verbrugge and B. J. Koch, *Journal of The Electrochemical Society*, **143**, 600 (1996).

45. B. Dunn, J. Long, D. Rolison and H. White, *American Chemical Society*, **104**, 4463 (2004).

46. T. Arthur, D. Bates, N. Cirigliano, D. C. Johnson, P. Malati, J. M. Mosby, E. Perre, M. T. Rawls, A. L. Prieto and B. Dunn, *Materials Research Society*, **36**, 532 (2011).

47. D. R. Rolison, J. W. Long, J. C. Lytle, A. E. Fischer, C. P. Rhodes, T. M. McEvoy, M. E. Bourg and A. M. Lubers, *Chemical Society Reviews*, **38** (2009).

48. C. P. Smith and H. S. White, *Analytical Chemistry*, **65**, 3343 (1993).

49. V. G. Levich, *Physicochemical Hydrodynamics*, Prentice Hall, New York (1962).

50. D. R. Rolison and B. Dunn, *Journal of Materials Chemistry*, **11** (2001).

51. C. Wang, L. Taherabadi, G. Jia, M. Madou, Y. Yeh and B. Dunn, *Electrochemical and Solid-State Letters*, **7**, A435 (2004).

52. R. W. Hart, H. S. White, B. Dunn and D. R. Rolison, *Electrochemistry Communications*, **5**, 120 (2003).

53. V. Zadin, D. Brandell, H. Kasemägi, A. Aabloo and J. O. Thomas, *Solid State Ionics*, **192**, 279 (2011).

54. V. Zadin, H. Kasemägi, A. Aabloo and D. Brandell, *Journal of Power Sources*, **195**, 6218 (2010).

55. A. Le Mehaute and G. Crepy, *C. R. Acad. Sci.*, **294**, 685 (1982).

56. S. Havlin, and A. Bunde, *Fractals and Disordered Systems*, Springer-Verlag, Berlin (1996).

57. G. T. Teixidor, B. Y. Park, P. P. Mukherjee, Q. Kang and M. J. Madou *Electrochimica Acta*, **54**, 5928 (2009).

58. L. Fruchter, G. Crepy and A. Le Mehaute, *J. Power Sources*, **18** (1986).

59. L. Nyikos and T. Pajkossy, *Electrochimica Acta*, **30** (1985).

60. L. Nyikos and T. Pajkossy, *Electrochimica Acta*, **31** (1986).

61. M. Filoche and B. Sapoval *Phys. Rev. Lett.*, **84** (2000).

62. T. Pajkossy, *J. Electroanal. Chem*, **300**, 1 (1991).

63. T. Pajkossy and L. Nyikos, *Phys. Rev. B*, **42 (1)** (1990).

64. J. Dietiker, MFIX: Cartesian User Guide, https://mfix.netl.doe.gov/ (2009).

65. T. Li, J. Dietiker, Y. Zhang and M. Shahnam, *Chemical Engineering Science*, **66**, 6220 (2011).

66. T. McKeen and T. Pugsley, *Powder Technology*, **129**, 139 (2003).

67. S. Benyahia, M. Syamlal and T.J. Obrien, *Summary of MFIX Equations 2012-1,* https://mfix.netl.doe.gov/ (2012).

68. M. Syamlal, Mfix Documentation Numerical Technique, https://mfix.netl.doe.gov/ (1998).

69. S. V. Patankar, *Numerical Heat Transfer and Fluid Flow*, Hemisphere Publishing Corporation, New York (1980).

70. L. F. Shampine and M. W. Reichelt, *SIAM J. Sci. Comput.*, **18**, 1 (1997).

71. R. D. Skeel and M. Berzins, *SIAM Journal on Scientific and Statistical Computing*, **11**, 1 (1990).

72. L. F. Shampine, M. W. Reichelt and J. A. Kierzenka, *SIAM Review*, **41**, 538 (1999).

73. K. Kumaresan, Q. Guo, P. Ramadass and R. E. White, *Journal of Power Sources*, **158**, 679 (2006).

APPENDIX A

RELAXATION TIME CALCULATION

A typical result of the relaxation process is shown below, in Figure A.1, for the relaxation of the Spherical Column 1 Base structure over the two hour period. Additionally shown are the points at which a finite difference approximation was applied to calculate the derivative. To avoid potential 'stair-stepping' in the data causing an improper calculation of the derivative, smoothing has been applied by MATLAB® software to the data such that only the first appearance of a concentration value is recorded, and all others are neglected. Because this process may cause a non-uniform step size in time, as illustrated in the bottom of the figure, the following equation was used to calculate the derivative, and was derived using a second order Taylor Expansion with uneven node spacing.

$$\frac{dc}{dt} = \frac{-c_{i+1}h_i^2 + c_i h_i^2 + h_{i+1}^2 c_{i-1} - h_{i+1}^2 c_i}{h_{i+1}\left(h_i + h_{i+1}\right)h_i}$$

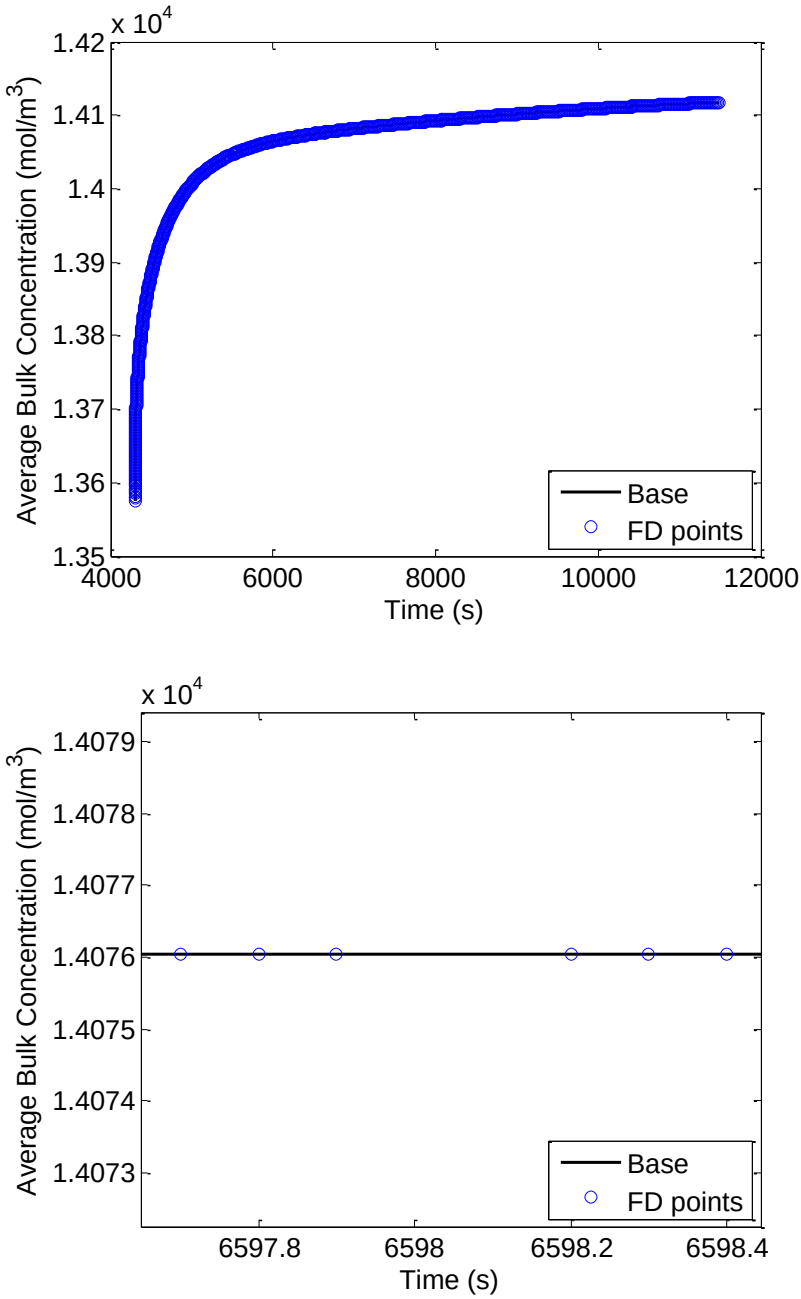

Figure A.1. Relaxation in average bulk concentration for the Base Spherical Column 1 structure. To avoid 'stair-stepping' in the data and an incorrect approximation in the derivative, some smoothing as been applied.

Alternatively, one may use a polynomial regression to calculate the derivate using MATLAB® software. For the above data, the following uses this approach, as shown in Figure A.2.

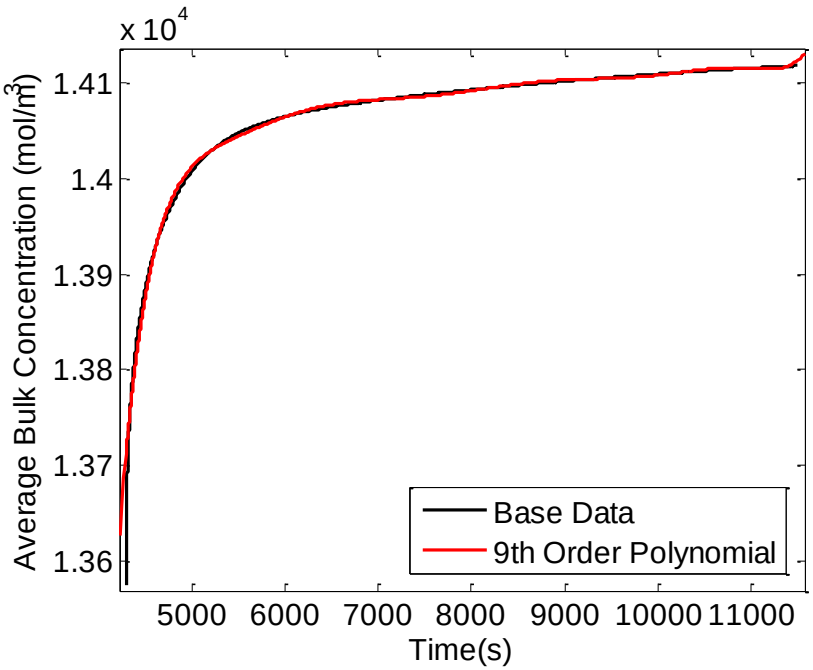

Figure A.2. Relaxation in average bulk concentration for the Base Spherical Column 1 structure. A 9th order polynomial has been applied to compute the derivative.

APPENDIX B

SIMULATION PROCEDURE

MATLAB GEOMETRY GENERATORS

The geometry files for MFiX® are generated using the *Fractal_Geometry_Generator.m* and *Fractal_Geometry_Reader.m* files. Here the user can specify the number of branches, generations, and other parameters to generate the desired geometry. This first will produce two text files *GEOMFORMFIX.txt* and *GEOMFORMATLAB.txt*. The second can be used to generate perturbed realizations by running the *Fractal_Geometry_Reader.m* file when the above text files are in the same directory, and will produce *INCGEOMFORMFIX.txt* and *INCGEOMFORMATLAB.txt*. Additionally, for the aperiodic 3D cell, a base anode structure is produced by using one of the above codes, and the *Fractal_Aperiodic_Generator.m* file can be used when the this case is in the same directory.These will be used in the *mfix.dat* file as discussed later.

MATLAB PDEPE

MATLAB® software was used to solve the fickian diffusion problem in spherical coordinates. The corresponding file for doing so is *FICKIANPDE.m*. Here the user will specify the operating current, the radius of the particle, the diffusion coefficient, the maximum intercalatable concentration, the initial concentration, Faraday's Constant, and the active surface area of the electrode on lines 21 through 26. The spatial discretization and time vector are specified on lines 34 and 35. Once run, this

file will produce three files: *CATH_SConc.txt*, *CATH_AConc.txt*, and *CATH_FConc.txt*, for the surface concentration, average bulk concentration, and final concentration after discharge, respectively.

In order to restart a simulation, such as when the relaxation studies were conducted, *FICKIANPDERS.m* file must be used. The *CATH_FConc.txt* file from above must be in the same directory, and renamed to *CATH_IConc.txt*, to serve as the initial conditions for the simulation. Once run, this file will produce the same as with *FICKIANPDE.m*.

MFiX

The following instructions assume the user has properly installed MFiX® as per the instructions on the download website. These instructions will also assume that one folder per simulation will be made, and that the following files are in such a folder: *mfix.dat*, *rrates.f*, *scalar_prop.f*, *usr0.f*, and *usr1.f*. Additionally, the following folders need to be included in the simulation folder: *cartesian_grid* and *post*. The following files need to be included in *cartesian_grid: allocate_cut_cell_arrays.f, cutcell_mod.f*, *vtk_out.f*. The following files need to be included in *post*: *post_cbar_time* and *post_epg*.

The *mfix.dat* file contains all geometry specifications as well as tolerance values used in the simulation. It will also specify intial concentrations as well as the constant flux condition. The simulation time is specified by a start time of *TIME* and stop time *TSTOP*. The timestep used is specified by *DT*. The geometry produced by the MATLAB® file *Fractal_Geometry_Geneartor.m* is copied and placed under the

statement *CARTESIAN_GRID=.TRUE.* After the goemetry is specified, *TOL_F* can be altered to specify the desired accuracy for geometry definition. The simulation space is altered by changing the valuees of *XLENGTH*, *YLENGTH*, and *ZLENGTH*. The number of cells in each direction are specifed by *IMAX*, *JMAX*, and *KMAX*, respectively. For a 2D simulation, the command *NO_K=.TRUE.* is entered under the cell definitions, and *KMAX* must be commented out. The initial conditions are enetered unter the *Initial Conditions Section*. Note that the spatial extents used before must be entered here, and the initial concentration is specified by *IC_SCALAR(1,1)*. Further, the boundary condition on the structure is specified first by setting it to be a no slip wall, or *BC_TYPE(12)='CG_NSW'*. Note there that the number twelve in the brackets corresponds to the group number of the geometry. *BC_SCALARW* can be used to specify a constant value of concentration on the surface of the geometry, but is shut off by setting *BC_HW_SCALAR* equal to 0. The constant flux condition is applied by *BC_C_SCALAR*, and it is to be noted that the flux desired must be divided by the diffusion coefficient and the corrseponding value enetered here. The output of the code can be controlled by *SPX_DT*, which saves prescribed properties of the simulation at every time step.

Once these values are altered, the *mfix.exe* executable file must first be generated by entering the following command in the terminal, where the directory is inside the simulation folder.

~/mfix/model/make_mfix (0.86)

This will produce the needed exectuable file. The simulation is started by typing the following commande

./mfix.exe (0.87)

The simulation will run until *TSTOP*. The resulting *.vtk* files can be used to visualize results. The *scalar_prop.f* file is used to specify the diffusion coefficients within the structure. The file , *usr0.f* is used to calculate the surface concentration at every time step and will produce the file *AVGSURF_CONC.dat*. The file *usr1.f* is used to calculate the volume and surface area of the structure, and will be displayed in the terminal immediately after the simulation starts. Note, that for 2D simulations, these values are actually an area and length, multiplied by *ZLENGTH* as specified in the *mfix.dat* file. Once a simulation is complete, the following command is issued to produce the file *cbar_c.dat*.

~/mfix/post_mfix/post_mfix<./post/post_cbar_time (0.88)

Additionally, the file *void.dat* is generated similarly by typing the following into the terminal.

~/mfix/post_mfix/post_mfix<./post/post_epg (0.89)

These two files are used in conjuncture to calculate the average concentration in the geometry. The file *cbar_c.dat* contains average concentration values that have been multiplied by the void fraction of the simulation, as shown in *void.dat*. Therefore, to extract the average concentration values, the values in *cbar_c.dat* must be divided by the value calcualted by *void.dat*. It is noted that *void.dat* will contain void fraction values at

every time step, but these are the same as no change in geometry is occuring for these simulations.

In order to restart an MFiX simulation, a restart file at the desired time step must be produced. This is accomplished by entering the following command.

~/mfix/post_mfix/post_mfix (0.90)

Following the onscreen instructions, one must produce a *.RES* file from the data stored in the *.SPX* files. Following the instructions, the user will be asked to retrieve the data from the desired time step. Once this is complete, the user will be asked for the time at which the restart file is to start, and the time step *DT* to be used. The *mfix.dat* file then needs to be altered accordingly if required, with the proper time and flux values. Additionally, when running a restart, the line in *mfix.dat RUN_TYPE='new'* needs to be changed to *RUN_TYPE='Restart_1'*.

Processing

After corresponding simulations are completed, the file *Data_Processor.m* is used to calculate the voltage values based on the surface concentrations stored in *AVGSURF_CONC.dat*. To use this code, the required files from both MFiX and MATLAB need to be in the same directory. Also, the user may wish to rename the files in a more descriptive fashion, especially if multiple realizations of the same geometry have been used, as this processing code is capable of processing any number of simulations. In the code attached, the anode bulk, surface, and volume fraction files have been renamed, so that for the instance of the base case these are *Base_AConc.txt*,

Base_SConc.txt, and *Base_Void.txt*. Correspondingly, the cathode simulations have been renamed to *CATH_AConc.txt*, *CATH_SConc.txt*, and *CATH_Void.txt*. It is to be noted, that when an MFiX simulation is restarted, the intial average bulk concentration value will not be recorded in *cbar_c.dat*. Therefore the user will need to take this value from the *cbar_c.dat* file from the original simulation. In *Data_Processor.m* the user will specify the bounds of the simulation time, the timestep, the applied current, and the bounds of the computational domain. Further, using the *void.dat* files, the volume fractions are specified beginning on line 27. The appropriate text files that specify bulk and surface concentrations are read in by MATLAB® beginning on line 36. It is to be noted that *cbar_c.dat* has headers and timestamps that need to be stripped before MATLAB® can read it. This can be accomplished easily in Microsoft Excel®. All of these concentration values are stored in a matrix called *Sims_Conc*. The global cell properties are defined beginning on line 54. Once this code is run, text files containing the voltage values will be produced at each time step.

Fracatal_Geometry_Generator.m

```
%This m-file is meant to generate a fractal based microstructure for a
%Lithium Ion Battery electrode

clc, clear all, close all, format long,
%%User Inputs
    %Specify the maximum number of generations
        max_gen=7;

    %Specify the mean and standard deviation of the particle axes
        axes_mean=12.5/8.5*10^-6;
        axes_stdev=1/8.5*10^-6;

%%Fractal Generation
%Set a particle counters
    part_count_new=0;
    part_count_old=0;

%Generate a number of spherical spawning particles, that are
equidistant from each
%other
    for i=1:10
        part_count_new=part_count_new+1;

part_mjr_axis(1,part_count_new)=normrnd(axes_mean,axes_stdev,1);

part_mnr_axis(1,part_count_new)=part_mjr_axis(1,part_count_new);
        part_coord_x(1,part_count_new)=3*10^-6+1.5*axes_mean*i;
        part_coord_y(1,part_count_new)=3*10^-6;
        part_branlen(1,part_count_new)=0;
        part_angle(1,part_count_new)=0;
        part_type(1,part_count_new)=1;
        part_parent_x(1,part_count_new)=0;
        part_parent_y(1,part_count_new)=0;
        part_parent_type(1,part_count_new)=0;
        part_parent_angle(1,part_count_new)=0;
        part_generation(1,part_count_new)=0;
    end

%Produce the number of specified generations
    for i=1:max_gen
        part_sweep_first=part_count_old+1;
        part_sweep_last=part_count_old+part_count_new;
        part_count_old=part_count_old+part_count_new;
        part_count_new=0;

        %Sweep over previous generation
            for j=part_sweep_first:part_sweep_last
                parent_x=part_coord_x(1,j);
                parent_y=part_coord_y(1,j);
                parent_mjr_axis=part_mjr_axis(1,j);
```

```
                parent_mnr_axis=part_mnr_axis(1,j);
                parent_type=part_type(1,j);
                parent_angle=part_angle(1,j);

                %Determine the number of branches the parent particle
will
                %produce
                if parent_type~=3
                    num_bran=randi([1,1],1);
                else
                    num_bran=1;
                end

                if num_bran>0

                    for k=1:num_bran

                        %Determine the child particle type 1=sphere,
2=ellipse,
                        %3=cylinder. However, prevent 2 cylinders from
                        %forming consecutively.
                        if part_type(1,j)~=3
                            type=randi([1,1],1);
                            if type ==2
                                type =1;
                            end
                        else
                            type=randi([1,1],1);
                        end


                        if type==1
                            %Determine the radius of the sphere

mjr_axis=normrnd(axes_mean,axes_stdev,1);
                                mnr_axis=mjr_axis;

                            %Determine the branch length

bran_length=randi([50,95],1)/100*(parent_mjr_axis+mjr_axis);
                        end

                        if type==3
                            %The major axis of the cylinder is half its
length,
                            %the minor is its radius. Maintain an L/d
ratio
                            %of 2.

mjr_axis=normrnd(axes_mean,axes_stdev,1);
                                mnr_axis=mjr_axis/2;
```

```
                            %Determine the branch length

bran_length=randi([50,95],1)/100*(parent_mjr_axis+mjr_axis);
                        end

                        if type==2
                            %The major axis is half the major axis of
an elipse

mjr_axis=normrnd(axes_mean,axes_stdev,1);

mnr_axis=normrnd(axes_mean,axes_stdev,1);

                            %Determine the branch length

bran_length=randi([50,95],1)/100*(parent_mjr_axis+mjr_axis);
                        end

                        neigh_x=zeros(0,0);
                        neigh_y=zeros(0,0);
                        neigh_mnr_axis=zeros(0,0);
                        bran_angle_low=0;
                        bran_angle_up=pi;
                        near=1;
                        %Determine if particle placement is possible by
                        %searching for nearest particle neighbors that
are
                        %within (branch+major_axis) distance
                        for l=1:length(part_coord_x)
                            dist=sqrt((parent_x-
part_coord_x(1,l))^2+(parent_y-part_coord_y(1,l))^2);
                            if dist>0 &&
dist<(bran_length+mjr_axis+part_mjr_axis(1,l))
                                neigh_x(1,near)=part_coord_x(1,l);
                                neigh_y(1,near)=part_coord_y(1,l);

neigh_mjr_axis(1,near)=part_mjr_axis(1,l);
                                near=near+1;
                            end
                        end

                        if prod(size(neigh_x))~=0
                            for l=1:length(neigh_x)
                                angle=atan2(neigh_y(1,l)-
parent_y,neigh_x(1,l)-parent_x);
                                if angle>=0 && angle<pi/2
                                    if angle>bran_angle_low
                                        bran_angle_low=0;

bran_angle_low=bran_angle_low+angle+2*atan2(neigh_mjr_axis(1,l),bran_le
ngth);
                                    end
                                end
```

```
                                if angle>=pi/2 &&angle<=pi
                                    if angle<bran_angle_up
                                        bran_angle_up=pi;
                                        bran_angle_up=bran_angle_up-
(pi-angle)-2*atan2(neigh_mjr_axis(1,l),bran_length);
                                    end
                                end
                            end
                        end

                        if bran_length*(bran_angle_up-
bran_angle_low)>2*mnr_axis
                            if parent_type~=3

bran_angle=(bran_angle_up+bran_angle_low)/2+randi([-
15,15],1)/100*(bran_angle_up+bran_angle_low)/2;
                            else
                                bran_angle=parent_angle;
                            end
                            part_count_new=part_count_new+1;

part_mjr_axis(1,part_count_new+part_count_old)=mjr_axis;

part_mnr_axis(1,part_count_new+part_count_old)=mnr_axis;

part_type(1,part_count_new+part_count_old)=type;

part_coord_x(1,part_count_new+part_count_old)=parent_x+bran_length*cos(
bran_angle);

part_coord_y(1,part_count_new+part_count_old)=parent_y+bran_length*sin(
bran_angle);

part_branlen(1,part_count_new+part_count_old)=bran_length;

part_angle(1,part_count_new+part_count_old)=bran_angle;

part_parent_x(1,part_count_new+part_count_old)=parent_x;

part_parent_y(1,part_count_new+part_count_old)=parent_y;

part_parent_type(1,part_count_new+part_count_old)=parent_type;

part_parent_angle(1,part_count_new+part_count_old)=parent_angle;

part_generation(1,part_count_new+part_count_old)=i;
                        end

                    end
                end
            end
    end
```

```
%Plot the particles
    figure(1)
    hold on
    axis equal
    for i=1:length(part_coord_x)
        %Extract the particle type
            type=part_type(1,i);
        if type==1
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            [x y]=draw_circle(part_x, part_y, mjr_axis);
            plot(x,y)
        end

        if type==3
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            a=[part_x+mnr_axis*cos(theta+3*pi/2)
part_y+mnr_axis*sin(theta+3*pi/2)];
            b=[part_x+mnr_axis*cos(theta+pi/2)
part_y+mnr_axis*sin(theta+pi/2)];
            line([a(1)+mjr_axis*cos(theta)
b(1)+mjr_axis*cos(theta)],[a(2)+mjr_axis*sin(theta)
b(2)+mjr_axis*sin(theta)])
            line([b(1)+mjr_axis*cos(theta) b(1)-
mjr_axis*cos(theta)],[b(2)+mjr_axis*sin(theta) b(2)-
mjr_axis*sin(theta)])
            line([b(1)-mjr_axis*cos(theta) a(1)-
mjr_axis*cos(theta)],[b(2)-mjr_axis*sin(theta) a(2)-
mjr_axis*sin(theta)])
            line([a(1)-mjr_axis*cos(theta)
a(1)+mjr_axis*cos(theta)],[a(2)-mjr_axis*sin(theta)
a(2)+mjr_axis*sin(theta)])
        end

        if type==2
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            ellipse(mjr_axis,mnr_axis,theta,part_x,part_y)
        end
    end

%Write the geometry so that MFIX can read it
```

```
    fid=fopen('GEOMFORMFIX.txt','w');

    str1=['N_QUADRIC = ',num2str(length(part_coord_x))];
    str2=[' '];
    fprintf(fid,'%s\n',str1);
    fprintf(fid,'%s\n',str2);

    for i=1:length(part_coord_x)
        type=part_type(1,i);

        if type==1
            str1=['QUADRIC_FORM(',num2str(i),') = ''Z_CYL_INT'''];
            str2=['RADIUS(',num2str(i),') =
',num2str(part_mjr_axis(1,i))];
            str3=[' '];
            str4=['t_x(',num2str(i),') = ',num2str(part_coord_x(1,i))];
            str5=['t_y(',num2str(i),') = ',num2str(part_coord_y(1,i))];
            str6=[' '];
            fprintf(fid,'%s\n',str1);
            fprintf(fid,'%s\n',str2);
            fprintf(fid,'%s\n',str3);
            fprintf(fid,'%s\n',str4);
            fprintf(fid,'%s\n',str5);
            fprintf(fid,'%s\n',str6);
        end

        if type==3
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            a=[part_x+mnr_axis*cos(theta+3*pi/2)
part_y+mnr_axis*sin(theta+3*pi/2)];
            b=[part_x+mnr_axis*cos(theta+pi/2)
part_y+mnr_axis*sin(theta+pi/2)];
            y1=a(2)+mjr_axis*sin(theta);
            y2=b(2)+mjr_axis*sin(theta);
            y3=b(2)-mjr_axis*sin(theta);
            y4=a(2)-mjr_axis*sin(theta);
            clip_ymax=max([y1, y2, y3, y4]);
            clip_ymin=min([y1, y2, y3, y4]);
            str1=['QUADRIC_FORM(',num2str(i),') = ''X_CYL_INT'''];
            str2=['RADIUS(',num2str(i),') = ',num2str(mnr_axis)];
            str3=[' '];
            str4=['t_x(',num2str(i),') = ',num2str(part_coord_x(1,i))];
            str5=['t_y(',num2str(i),') = ',num2str(part_coord_y(1,i))];
            str6=[' '];
            str7=['THETA_Z(',num2str(i),')=',num2str(theta*180/pi)];
            str8=[' '];
            str9=['clip_ymin(',num2str(i),')=',num2str(clip_ymin)];
            str10=['clip_ymax(',num2str(i),')=',num2str(clip_ymax)];
```

```
            str11=['FLUID_IN_CLIPPED_REGION(',num2str(i),') =
.FALSE.'];
            str12=[' '];
            fprintf(fid,'%s\n',str1);
            fprintf(fid,'%s\n',str2);
            fprintf(fid,'%s\n',str3);
            fprintf(fid,'%s\n',str4);
            fprintf(fid,'%s\n',str5);
            fprintf(fid,'%s\n',str6);
            fprintf(fid,'%s\n',str7);
            fprintf(fid,'%s\n',str8);
            fprintf(fid,'%s\n',str9);
            fprintf(fid,'%s\n',str10);
            fprintf(fid,'%s\n',str11);
            fprintf(fid,'%s\n',str12);
        end

        if type==2
            theta=part_angle(1,i);
            a=part_mjr_axis(1,i);
            b=part_mnr_axis(1,i);
            str1=['lambda_x(',num2str(i),') = ',num2str(b^2)];
            str2=['lambda_y(',num2str(i),') = ',num2str(a^2)];
            str3=['lambda_z(',num2str(i),') = ',num2str(0)];
            str4=['dquadric(',num2str(i),')= ',num2str(-a^2*b^2)];
            str5=[' '];
            str6=['t_x(',num2str(i),') = ',num2str(part_coord_x(1,i))];
            str7=['t_y(',num2str(i),') = ',num2str(part_coord_y(1,i))];
            str8=[' '];
            str9=['THETA_Z(',num2str(i),')=',num2str(theta*180/pi)];
            str10=[' '];
            fprintf(fid,'%s\n',str1);
            fprintf(fid,'%s\n',str2);
            fprintf(fid,'%s\n',str3);
            fprintf(fid,'%s\n',str4);
            fprintf(fid,'%s\n',str5);
            fprintf(fid,'%s\n',str6);
            fprintf(fid,'%s\n',str7);
            fprintf(fid,'%s\n',str8);
            fprintf(fid,'%s\n',str9);
            fprintf(fid,'%s\n',str10);
        end
    end

    str8=['N_GROUP = 1'];
    str9=['GROUP_SIZE(1) = ',num2str(length(part_coord_x))];
    fprintf(fid,'%s\n',str8);
    fprintf(fid,'%s\n',str9);
    for i=1:length(part_coord_x)
        str=['GROUP_Q(1,',num2str(i),') = ',num2str(i)];
        fprintf(fid,'%s\n',str);
    end
    str=['GROUP_RELATION(1) = ''OR'''];
```

```
    fprintf(fid,'%s\r',str);
    str=['BC_ID_Q(1) = '];
    fprintf(fid,str);
    for i=1:length(part_coord_x)
        str=[num2str(12),' '];
        fprintf(fid,str);
    end

fclose(fid);

%Write geometry in a format that matlab can read for post processing if
%needed

for i=1:length(part_coord_x)

M(i,[1:12])=[part_coord_x(1,i),part_coord_y(1,i),part_mjr_axis(1,i),par
t_mnr_axis(1,i),...
        part_type(1,i), part_branlen(1,i), part_angle(1,i),
part_parent_x(1,i),...

part_parent_y(1,i),part_parent_type(1,i),part_parent_angle(1,i),part_ge
neration(1,i)];
end

dlmwrite('GEOMFORMATLAB.txt',M,'precision',18)
```

Fractal_Geometry_Reader.m

```
%The following m-file is meant to read in fractal geometry produced by
%Fractal_Geometry_Generator, plot it, and, if required, perturb it to
%create new incarnations.

clc, clear all, close all, format long

%Read in the fractal geometry from GEOMFORMATLAB.txt
    M=dlmread('GEOMFORMATLAB.txt');

%Separate the columns of 'M' into appropriate arrays
    part_coord_x(1,:)=M(:,1);
    part_coord_y(1,:)=M(:,2);
    part_mjr_axis(1,:)=M(:,3);
    part_mnr_axis(1,:)=M(:,4);
    part_type(1,:)=M(:,5);
    part_branlen(1,:)=M(:,6);
    part_angle(1,:)=M(:,7);
    part_parent_x(1,:)=M(:,8);
    part_parent_y(1,:)=M(:,9);
    part_parent_type(1,:)=M(:,10);
    part_parent_angle(1,:)=M(:,11);
    part_generation(1,:)=M(:,12);

%Plot the particles
    figure(1)
    hold on
    axis equal
    for i=1:length(part_coord_x)
        %Extract the particle type
            type=part_type(1,i);
        if type==1
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            [x y]=draw_circle(part_x, part_y, mjr_axis);
            plot(x,y,'b')
        end

        if type==3
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            a=[part_x+mnr_axis*cos(theta+3*pi/2)
part_y+mnr_axis*sin(theta+3*pi/2)];
            b=[part_x+mnr_axis*cos(theta+pi/2)
part_y+mnr_axis*sin(theta+pi/2)];
```

```
            line([a(1)+mjr_axis*cos(theta)
b(1)+mjr_axis*cos(theta)],[a(2)+mjr_axis*sin(theta)
b(2)+mjr_axis*sin(theta)])
            line([b(1)+mjr_axis*cos(theta) b(1)-
mjr_axis*cos(theta)],[b(2)+mjr_axis*sin(theta) b(2)-
mjr_axis*sin(theta)])
            line([b(1)-mjr_axis*cos(theta) a(1)-
mjr_axis*cos(theta)],[b(2)-mjr_axis*sin(theta) a(2)-
mjr_axis*sin(theta)])
            line([a(1)-mjr_axis*cos(theta)
a(1)+mjr_axis*cos(theta)],[a(2)-mjr_axis*sin(theta)
a(2)+mjr_axis*sin(theta)])
        end

        if type==2
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            ellipse(mjr_axis,mnr_axis,theta,part_x,part_y)
        end
    end

%Introduce pertubations in the branch angles and lengths to create new
incarnations
%of the geometry

    %Specify angle pertubation limits. These are percentages of how
much the
    %angle can increase or decrease
        angle_pert_up=130;
        angle_pert_low=70;
        length_pert_up=120;
        length_pert_low=80;

    %Recalculate new coordinates based upon new angles
    for i=1:length(part_coord_x)
        parent_x=part_parent_x(1,i);
        parent_y=part_parent_y(1,i);
        parent_type=part_parent_type(1,i);
        parent_angle=part_parent_angle(1,i);
        part_gen=part_generation(1,i);
        part_x=part_coord_x(1,i);
        part_y=part_coord_y(1,i);
        bran_length=part_branlen(1,i);

        if part_gen~=0
            bran_angle=part_angle(1,i);
            if parent_type~=3

part_angle(1,i)=randi([angle_pert_low,angle_pert_up],1)/100*bran_angle;
```

```
            else
                part_angle(1,i)=part_parent_angle(1,i);
            end

part_branlen(1,i)=randi([length_pert_low,length_pert_up],1)/100*bran_le
ngth;
            %Find the children of this particle, and reset their
parent's
            %coordinates
            bran_length=part_branlen(1,i);
                for j=1:length(part_coord_x)
                    if part_generation(1,j)==part_gen+1
                        if part_parent_x(1,j)==part_x &&
part_parent_y(1,j)==part_y

part_parent_x(1,j)=parent_x+bran_length*cos(part_angle(1,i));

part_parent_y(1,j)=parent_y+bran_length*sin(part_angle(1,i));
                           part_parent_angle(1,j)=part_angle(1,i);
                        end
                    end
                end

part_coord_x(1,i)=parent_x+bran_length*cos(part_angle(1,i));

part_coord_y(1,i)=parent_y+bran_length*sin(part_angle(1,i));
        end
    end

    %Replot the particles in red over the orignal blue
    for i=1:length(part_coord_x)
        %Extract the particle type
            type=part_type(1,i);
        if type==1
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            [x y]=draw_circle(part_x, part_y, mjr_axis);
            plot(x,y,'r')
        end

        if type==3
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            a=[part_x+mnr_axis*cos(theta+3*pi/2)
part_y+mnr_axis*sin(theta+3*pi/2)];
            b=[part_x+mnr_axis*cos(theta+pi/2)
part_y+mnr_axis*sin(theta+pi/2)];
```

```
            line([a(1)+mjr_axis*cos(theta)
b(1)+mjr_axis*cos(theta)],[a(2)+mjr_axis*sin(theta)
b(2)+mjr_axis*sin(theta)],'Color','r')
            line([b(1)+mjr_axis*cos(theta) b(1)-
mjr_axis*cos(theta)],[b(2)+mjr_axis*sin(theta) b(2)-
mjr_axis*sin(theta)],'Color','r')
            line([b(1)-mjr_axis*cos(theta) a(1)-
mjr_axis*cos(theta)],[b(2)-mjr_axis*sin(theta) a(2)-
mjr_axis*sin(theta)],'Color','r')
            line([a(1)-mjr_axis*cos(theta)
a(1)+mjr_axis*cos(theta)],[a(2)-mjr_axis*sin(theta)
a(2)+mjr_axis*sin(theta)],'Color','r')
        end

        if type==2
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            ellipse(mjr_axis,mnr_axis,theta,part_x,part_y,'r')
        end
    end

    %Write the geometry so that MFIX can read it
    fid=fopen('INCGEOMFORMFIX.txt','w');

    str1=['N_QUADRIC = ',num2str(length(part_coord_x))];
    str2=[' '];
    fprintf(fid,'%s\n',str1);
    fprintf(fid,'%s\n',str2);

    for i=1:length(part_coord_x)
        type=part_type(1,i);

        if type==1
            str1=['QUADRIC_FORM(',num2str(i),') = ''Z_CYL_INT'''];
            str2=['RADIUS(',num2str(i),') =
',num2str(part_mjr_axis(1,i))];
            str3=[' '];
            str4=['t_x(',num2str(i),') = ',num2str(part_coord_x(1,i))];
            str5=['t_y(',num2str(i),') = ',num2str(part_coord_y(1,i))];
            str6=[' '];
            fprintf(fid,'%s\n',str1);
            fprintf(fid,'%s\n',str2);
            fprintf(fid,'%s\n',str3);
            fprintf(fid,'%s\n',str4);
            fprintf(fid,'%s\n',str5);
            fprintf(fid,'%s\n',str6);
        end

        if type==3
```

```
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            a=[part_x+mnr_axis*cos(theta+3*pi/2)
part_y+mnr_axis*sin(theta+3*pi/2)];
            b=[part_x+mnr_axis*cos(theta+pi/2)
part_y+mnr_axis*sin(theta+pi/2)];
            y1=a(2)+mjr_axis*sin(theta);
            y2=b(2)+mjr_axis*sin(theta);
            y3=b(2)-mjr_axis*sin(theta);
            y4=a(2)-mjr_axis*sin(theta);
            clip_ymax=max([y1, y2, y3, y4]);
            clip_ymin=min([y1, y2, y3, y4]);
            str1=['QUADRIC_FORM(',num2str(i),') = ''X_CYL_INT'''];
            str2=['RADIUS(',num2str(i),') = ',num2str(mnr_axis)];
            str3=[' '];
            str4=['t_x(',num2str(i),') = ',num2str(part_coord_x(1,i))];
            str5=['t_y(',num2str(i),') = ',num2str(part_coord_y(1,i))];
            str6=[' '];
            str7=['THETA_Z(',num2str(i),')=',num2str(theta*180/pi)];
            str8=[' '];
            str9=['clip_ymin(',num2str(i),')=',num2str(clip_ymin)];
            str10=['clip_ymax(',num2str(i),')=',num2str(clip_ymax)];
            str11=['FLUID_IN_CLIPPED_REGION(',num2str(i),') =
.FALSE.'];
            str12=[' '];
            fprintf(fid,'%s\n',str1);
            fprintf(fid,'%s\n',str2);
            fprintf(fid,'%s\n',str3);
            fprintf(fid,'%s\n',str4);
            fprintf(fid,'%s\n',str5);
            fprintf(fid,'%s\n',str6);
            fprintf(fid,'%s\n',str7);
            fprintf(fid,'%s\n',str8);
            fprintf(fid,'%s\n',str9);
            fprintf(fid,'%s\n',str10);
            fprintf(fid,'%s\n',str11);
            fprintf(fid,'%s\n',str12);
        end

        if type==2
            theta=part_angle(1,i);
            a=part_mjr_axis(1,i);
            b=part_mnr_axis(1,i);
            str1=['lambda_x(',num2str(i),') = ',num2str(b^2)];
            str2=['lambda_y(',num2str(i),') = ',num2str(a^2)];
            str3=['lambda_z(',num2str(i),') = ',num2str(0)];
            str4=['dquadric(',num2str(i),')= ',num2str(-a^2*b^2)];
            str5=[' '];
            str6=['t_x(',num2str(i),') = ',num2str(part_coord_x(1,i))];
            str7=['t_y(',num2str(i),') = ',num2str(part_coord_y(1,i))];
```

```
            str8=[' '];
            str9=['THETA_Z(',num2str(i),')=',num2str(theta*180/pi)];
            str10=[' '];
            fprintf(fid,'%s\n',str1);
            fprintf(fid,'%s\n',str2);
            fprintf(fid,'%s\n',str3);
            fprintf(fid,'%s\n',str4);
            fprintf(fid,'%s\n',str5);
            fprintf(fid,'%s\n',str6);
            fprintf(fid,'%s\n',str7);
            fprintf(fid,'%s\n',str8);
            fprintf(fid,'%s\n',str9);
            fprintf(fid,'%s\n',str10);
        end
    end

    str8=['N_GROUP = 1'];
    str9=['GROUP_SIZE(1) = ',num2str(length(part_coord_x))];
    fprintf(fid,'%s\n',str8);
    fprintf(fid,'%s\n',str9);
    for i=1:length(part_coord_x)
        str=['GROUP_Q(1,',num2str(i),') = ',num2str(i)];
        fprintf(fid,'%s\n',str);
    end
    str=['GROUP_RELATION(1) = ''OR'''];
    fprintf(fid,'%s\r',str);
    str=['BC_ID_Q(1) = '];
    fprintf(fid,str);
    for i=1:length(part_coord_x)
        str=[num2str(12),' '];
        fprintf(fid,str);
    end

fclose(fid);

%Write geometry in a format that matlab can read for post processing if
%needed

for i=1:length(part_coord_x)

M(i,[1:12])=[part_coord_x(1,i),part_coord_y(1,i),part_mjr_axis(1,i),par
t_mnr_axis(1,i),...
        part_type(1,i), part_branlen(1,i), part_angle(1,i),
part_parent_x(1,i),...

part_parent_y(1,i),part_parent_type(1,i),part_parent_angle(1,i),part_ge
neration(1,i)];
end

dlmwrite('INCGEOMFORMATLAB.txt',M,'precision',18)
```

Fractal_Aperiodic_Generator.m

```
%The following m-file is meant to read in fractal geometry produced by
%Fractal_Geometry_Generator, then produce an interdigitated cathode
%structure

clc, clear all, close all, format long

%%User Inputs
    %Specify the maximum number of generations of the cathode structure
        max_gen=10;

    %Specify the mean and standard deviation of the particle axes
        axes_mean=8.5/8.5*10^-6;
        axes_stdev=1/8.5*10^-6;

%%Generate the Cathode Structure

%Read in the fractal geometry from GEOMFORMATLAB.txt
    M=dlmread('GEOMFORMATLAB.txt');

%Separate the columns of 'M' into appropriate arrays
    part_coord_x(1,:)=M(:,1);
    part_coord_y(1,:)=M(:,2);
    part_mjr_axis(1,:)=M(:,3);
    part_mnr_axis(1,:)=M(:,4);
    part_type(1,:)=M(:,5);
    part_branlen(1,:)=M(:,6);
    part_angle(1,:)=M(:,7);
    part_parent_x(1,:)=M(:,8);
    part_parent_y(1,:)=M(:,9);
    part_parent_type(1,:)=M(:,10);
    part_parent_angle(1,:)=M(:,11);
    part_generation(1,:)=M(:,12);

%Set a particle counters
    part_count_new=0;
    part_count_old=length(part_coord_x);
    anode_part=length(part_coord_x);

%Generate a number of spherical spawning particles, that are
equidistant from each
%other
    for i=1:14
        part_count_new=part_count_new+1;

part_mjr_axis(1,part_count_old+part_count_new)=normrnd(axes_mean,axes_s
tdev,1);

part_mnr_axis(1,part_count_old+part_count_new)=part_mjr_axis(1,part_cou
nt_new);
```

```
        part_coord_x(1,part_count_old+part_count_new)=5*10^-
6+1.5*axes_mean*i;
        part_coord_y(1,part_count_old+part_count_new)=21.5*10^-6;
        part_branlen(1,part_count_old+part_count_new)=0;
        part_angle(1,part_count_old+part_count_new)=0;
        part_type(1,part_count_old+part_count_new)=1;
        part_parent_x(1,part_count_old+part_count_new)=0;
        part_parent_y(1,part_count_old+part_count_new)=0;
        part_parent_type(1,part_count_old+part_count_new)=0;
        part_parent_angle(1,part_count_old+part_count_new)=0;
        part_generation(1,part_count_old+part_count_new)=0;
    end

%Produce the number of specified generations
    for i=1:max_gen
        part_sweep_first=part_count_old+1;
        part_sweep_last=part_count_old+part_count_new;
        part_count_old=part_count_old+part_count_new;
        part_count_new=0;

        %Sweep over previous generation
            for j=part_sweep_first:part_sweep_last
                parent_x=part_coord_x(1,j);
                parent_y=part_coord_y(1,j);
                parent_mjr_axis=part_mjr_axis(1,j);
                parent_mnr_axis=part_mnr_axis(1,j);
                parent_type=part_type(1,j);
                parent_angle=part_angle(1,j);

                %Determine the number of branches the parent particle
will
                %produce
                if parent_type~=3
                    num_bran=randi([1,1],1);
                else
                    num_bran=1;
                end

                if num_bran>0

                    for k=1:num_bran

                        %Determine the child particle type 1=sphere,
2=ellipse,
                        %3=cylinder. However, prevent 2 cylinders from
                        %forming consecutively.
                        if part_type(1,j)~=3
                            type=randi([1,1],1);
                            if type ==2
                                type =1;
                            end
                        else
```

```
                            type=randi([1,1],1);
                        end


                        if type==1
                            %Determine the radius of the sphere

mjr_axis=normrnd(axes_mean,axes_stdev,1);
                                mnr_axis=mjr_axis;

                            %Determine the branch length

bran_length=randi([50,95],1)/100*(parent_mjr_axis+mjr_axis);
                        end

                        if type==3
                            %The major axis of the cylinder is half its
length,
                            %the minor is its radius. Maintain an L/d
ratio
                            %of 2.

mjr_axis=normrnd(axes_mean,axes_stdev,1);
                                mnr_axis=mjr_axis/2;

                            %Determine the branch length

bran_length=randi([50,95],1)/100*(parent_mjr_axis+mjr_axis);
                        end

                        if type==2
                            %The major axis is half the major axis of
an elipse

mjr_axis=normrnd(axes_mean,axes_stdev,1);

mnr_axis=normrnd(axes_mean,axes_stdev,1);

                            %Determine the branch length

bran_length=randi([50,95],1)/100*(parent_mjr_axis+mjr_axis);
                        end

                        neigh_x=zeros(0,0);
                        neigh_y=zeros(0,0);
                        neigh_mnr_axis=zeros(0,0);
                        bran_angle_low=-pi;
                        bran_angle_up=0;
                        near=1;
                        %Determine if particle placement is possible by
                        %searching for nearest particle neighbors that
are
```

```
                        %within (branch+major_axis) distance
                        for l=1:length(part_coord_x)
                            dist=sqrt((parent_x-
part_coord_x(1,l))^2+(parent_y-part_coord_y(1,l))^2);
                            if dist>0 &&
dist<(bran_length+mjr_axis+part_mjr_axis(1,l))
                                neigh_x(1,near)=part_coord_x(1,l);
                                neigh_y(1,near)=part_coord_y(1,l);

neigh_mjr_axis(1,near)=part_mjr_axis(1,l);
                                near=near+1;
                            end
                        end

                        if prod(size(neigh_x))~=0
                            for l=1:length(neigh_x)
                                angle=atan2(neigh_y(1,l)-
parent_y,neigh_x(1,l)-parent_x);
                                if angle>=-pi && angle<-pi/2
                                    if angle>bran_angle_low
                                        bran_angle_low=angle

%+2*atan2(neigh_mjr_axis(1,l),bran_length);
                                    end
                                end

                                if angle>=-pi/2 &&angle<=0
                                    if angle<bran_angle_up
                                        bran_angle_up=pi;
                                        bran_angle_up=angle
                                        %-
2*atan2(neigh_mjr_axis(1,l),bran_length);
                                    end
                                end
                            end
                        end

                        if bran_length*(bran_angle_up-
bran_angle_low)>3*mnr_axis
                            if parent_type~=3

bran_angle=(bran_angle_up+bran_angle_low)/2+randi([-
15,15],1)/100*(bran_angle_up+bran_angle_low)/2;
                            else
                                bran_angle=parent_angle;
                            end
                            part_count_new=part_count_new+1;

part_mjr_axis(1,part_count_new+part_count_old)=mjr_axis;

part_mnr_axis(1,part_count_new+part_count_old)=mnr_axis;

part_type(1,part_count_new+part_count_old)=type;
```

```
part_coord_x(1,part_count_new+part_count_old)=parent_x+bran_length*cos(
bran_angle);

part_coord_y(1,part_count_new+part_count_old)=parent_y+bran_length*sin(
bran_angle);

part_branlen(1,part_count_new+part_count_old)=bran_length;

part_angle(1,part_count_new+part_count_old)=bran_angle;

part_parent_x(1,part_count_new+part_count_old)=parent_x;

part_parent_y(1,part_count_new+part_count_old)=parent_y;

part_parent_type(1,part_count_new+part_count_old)=parent_type;

part_parent_angle(1,part_count_new+part_count_old)=parent_angle;

part_generation(1,part_count_new+part_count_old)=i;
                        end

                    end
                end
            end
    end

%Plot the particles
    figure(1)
    hold on
    axis equal
    for i=1:length(part_coord_x)
        %Extract the particle type
            type=part_type(1,i);
        if type==1
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            [x y]=draw_circle(part_x, part_y, mjr_axis);
            if i<=anode_part
                plot(x,y)
            else
                plot(x,y,'r')
            end
        end

        if type==3
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
```

```
            part_y=part_coord_y(1,i);
            a=[part_x+mnr_axis*cos(theta+3*pi/2)
part_y+mnr_axis*sin(theta+3*pi/2)];
            b=[part_x+mnr_axis*cos(theta+pi/2)
part_y+mnr_axis*sin(theta+pi/2)];
            if i<=anode_part
                line([a(1)+mjr_axis*cos(theta)
b(1)+mjr_axis*cos(theta)],[a(2)+mjr_axis*sin(theta)
b(2)+mjr_axis*sin(theta)])
                line([b(1)+mjr_axis*cos(theta) b(1)-
mjr_axis*cos(theta)],[b(2)+mjr_axis*sin(theta) b(2)-
mjr_axis*sin(theta)])
                line([b(1)-mjr_axis*cos(theta) a(1)-
mjr_axis*cos(theta)],[b(2)-mjr_axis*sin(theta) a(2)-
mjr_axis*sin(theta)])
                line([a(1)-mjr_axis*cos(theta)
a(1)+mjr_axis*cos(theta)],[a(2)-mjr_axis*sin(theta)
a(2)+mjr_axis*sin(theta)])
            else
                line([a(1)+mjr_axis*cos(theta)
b(1)+mjr_axis*cos(theta)],[a(2)+mjr_axis*sin(theta)
b(2)+mjr_axis*sin(theta)],'Color','r')
                line([b(1)+mjr_axis*cos(theta) b(1)-
mjr_axis*cos(theta)],[b(2)+mjr_axis*sin(theta) b(2)-
mjr_axis*sin(theta)],'Color','r')
                line([b(1)-mjr_axis*cos(theta) a(1)-
mjr_axis*cos(theta)],[b(2)-mjr_axis*sin(theta) a(2)-
mjr_axis*sin(theta)],'Color','r')
                line([a(1)-mjr_axis*cos(theta)
a(1)+mjr_axis*cos(theta)],[a(2)-mjr_axis*sin(theta)
a(2)+mjr_axis*sin(theta)],'Color','r')
            end
        end

        if type==2
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            ellipse(mjr_axis,mnr_axis,theta,part_x,part_y)
        end
    end



    %Write the geometry so that MFIX can read it
    fid=fopen('INCGEOMFORMFIX.txt','w');

    str1=['N_QUADRIC = ',num2str(length(part_coord_x))];
    str2=[' '];
    fprintf(fid,'%s\n',str1);
```

```
    fprintf(fid,'%s\n',str2);

    for i=1:length(part_coord_x)
        type=part_type(1,i);

        if type==1
            str1=['QUADRIC_FORM(',num2str(i),') = ''Z_CYL_INT'''];
            str2=['RADIUS(',num2str(i),') =
',num2str(part_mjr_axis(1,i))];
            str3=[' '];
            str4=['t_x(',num2str(i),') = ',num2str(part_coord_x(1,i))];
            str5=['t_y(',num2str(i),') = ',num2str(part_coord_y(1,i))];
            str6=[' '];
            fprintf(fid,'%s\n',str1);
            fprintf(fid,'%s\n',str2);
            fprintf(fid,'%s\n',str3);
            fprintf(fid,'%s\n',str4);
            fprintf(fid,'%s\n',str5);
            fprintf(fid,'%s\n',str6);
        end

        if type==3
            theta=part_angle(1,i);
            mjr_axis=part_mjr_axis(1,i);
            mnr_axis=part_mnr_axis(1,i);
            part_x=part_coord_x(1,i);
            part_y=part_coord_y(1,i);
            a=[part_x+mnr_axis*cos(theta+3*pi/2)
part_y+mnr_axis*sin(theta+3*pi/2)];
            b=[part_x+mnr_axis*cos(theta+pi/2)
part_y+mnr_axis*sin(theta+pi/2)];
            y1=a(2)+mjr_axis*sin(theta);
            y2=b(2)+mjr_axis*sin(theta);
            y3=b(2)-mjr_axis*sin(theta);
            y4=a(2)-mjr_axis*sin(theta);
            clip_ymax=max([y1, y2, y3, y4]);
            clip_ymin=min([y1, y2, y3, y4]);
            str1=['QUADRIC_FORM(',num2str(i),') = ''X_CYL_INT'''];
            str2=['RADIUS(',num2str(i),') = ',num2str(mnr_axis)];
            str3=[' '];
            str4=['t_x(',num2str(i),') = ',num2str(part_coord_x(1,i))];
            str5=['t_y(',num2str(i),') = ',num2str(part_coord_y(1,i))];
            str6=[' '];
            str7=['THETA_Z(',num2str(i),')=',num2str(theta*180/pi)];
            str8=[' '];
            str9=['clip_ymin(',num2str(i),')=',num2str(clip_ymin)];
            str10=['clip_ymax(',num2str(i),')=',num2str(clip_ymax)];
            str11=['FLUID_IN_CLIPPED_REGION(',num2str(i),') =
.FALSE.'];
            str12=[' '];
            fprintf(fid,'%s\n',str1);
            fprintf(fid,'%s\n',str2);
            fprintf(fid,'%s\n',str3);
```

```
        fprintf(fid,'%s\n',str4);
        fprintf(fid,'%s\n',str5);
        fprintf(fid,'%s\n',str6);
        fprintf(fid,'%s\n',str7);
        fprintf(fid,'%s\n',str8);
        fprintf(fid,'%s\n',str9);
        fprintf(fid,'%s\n',str10);
        fprintf(fid,'%s\n',str11);
        fprintf(fid,'%s\n',str12);
    end

    if type==2
        theta=part_angle(1,i);
        a=part_mjr_axis(1,i);
        b=part_mnr_axis(1,i);
        str1=['lambda_x(',num2str(i),') = ',num2str(b^2)];
        str2=['lambda_y(',num2str(i),') = ',num2str(a^2)];
        str3=['lambda_z(',num2str(i),') = ',num2str(0)];
        str4=['dquadric(',num2str(i),')= ',num2str(-a^2*b^2)];
        str5=[' '];
        str6=['t_x(',num2str(i),') = ',num2str(part_coord_x(1,i))];
        str7=['t_y(',num2str(i),') = ',num2str(part_coord_y(1,i))];
        str8=[' '];
        str9=['THETA_Z(',num2str(i),')=',num2str(theta*180/pi)];
        str10=[' '];
        fprintf(fid,'%s\n',str1);
        fprintf(fid,'%s\n',str2);
        fprintf(fid,'%s\n',str3);
        fprintf(fid,'%s\n',str4);
        fprintf(fid,'%s\n',str5);
        fprintf(fid,'%s\n',str6);
        fprintf(fid,'%s\n',str7);
        fprintf(fid,'%s\n',str8);
        fprintf(fid,'%s\n',str9);
        fprintf(fid,'%s\n',str10);
    end
end

str8=['N_GROUP = 1'];
str9=['GROUP_SIZE(1) = ',num2str(length(part_coord_x))];
fprintf(fid,'%s\n',str8);
fprintf(fid,'%s\n',str9);
for i=1:length(part_coord_x)
    str=['GROUP_Q(1,',num2str(i),') = ',num2str(i)];
    fprintf(fid,'%s\n',str);
end
str=['GROUP_RELATION(1) = ''OR'''];
fprintf(fid,'%s\r',str);
str=['BC_ID_Q(1) = '];
fprintf(fid,str);
for i=1:length(part_coord_x)
    str=[num2str(12),' '];
    fprintf(fid,str);
```

```
    end

fclose(fid);

%Write geometry in a format that matlab can read for post processing if
%needed

for i=1:length(part_coord_x)

M(i,[1:12])=[part_coord_x(1,i),part_coord_y(1,i),part_mjr_axis(1,i),par
t_mnr_axis(1,i),...
        part_type(1,i), part_branlen(1,i), part_angle(1,i),
part_parent_x(1,i),...

part_parent_y(1,i),part_parent_type(1,i),part_parent_angle(1,i),part_ge
neration(1,i)];
end

dlmwrite('INCGEOMFORMATLAB.txt',M,'precision',18)
```

Data_Processor.m

```
%The following m-file is meant to read in 2 files produced by MFIX
%simulation and post processed by the user, *_CONC.txt and *_SCONC.txt,
%and to produce needed quantities. NOTE: MFIX will produce cbar_c.dat
and
%AVGSURF_CONC.dat. These need to be trimmed of any headers so that
MATLAB
%can read it. Use excel and create the above text file names.

clc, clear all, close all, format long

%%User Inputs:

    %Specify the bounds of simulation time and the time step used
        t_start = 0;
        t_stop = 7200;
        dt = .1;

    %Specify the Discharge/Charge Current (positive is discharging, and
    %vice versa)
        I_app=1.656; %A

    %Specify the bounds of the computational domain
        xmin=0;
        ymin=0;
        xmax=2.4*10^-4;
        ymax=15.0*10^-5;

    %Specify the void volume fractions of each simulation
        Base_ep=.56;

    %Read-in average concentration and surface average concentration
values
    %of each simulation, and group them in one matrix for processing
        Base_AConc=dlmread('Base_AConc.txt');
        Base_SConc=dlmread('Base_SConc.txt');
        CATH_AConc=dlmread('CATH_AConc.txt');
        CATH_SConc=dlmread('CATH_SConc.txt');
        Sims_Conc=horzcat(Base_AConc, Base_SConc);

     %Specify the Li-Ion Cell Properties
            %Anode Properties
                A_n=.08; %m^2
                c_max_n=31833; %mol/m^3
                D_s_n=3.9*10^-14; %m^2/s
                alpha_n=.5;
                k_n=1.764*10^-11;
                c_0_n=.7522*c_max_n; %mol/m^3
                S_n=.7824; %m^2
```

```
            %Mock Cathode Properties
                R_p_p=8.5*10^-6; %m
                c_max_p=51410; %mol/m^3
                D_s_p=1.0*10^-14;
                alpha_p=.5;
                k_p=6.6667*10^-11;
                c_0_p=.4952*c_max_p;
                S_p=1.1167;

             %Assume a constant concentration in the electrolyte for
both
             %electrodes
                c_e_n=1000;
                c_e_p=1000;

             %Other Properties
                nodes_p_r=100;
                sample_freq=1;
                F=96487; %C/mol
                R_gas=8.3143; %J/mol*K
                N_A=6.022*10^23; %atom/mol
                k=1.380*10^-23; %m^2 kg/(s^2 K)
                e=1.9*10^-19; %C
                T=298; %K
                eps_0=8.85*10^-12; %C^2/(N m^2)

%%Process

    %Generate time vector
        t=t_start:dt:t_stop;

    %Calculate the Equilibrium Potentials, Overpotentials, and Overall
Cell
    %Voltage vs time.

        %Equilibrium Potentials
            Eq_Pot_n=zeros(length(t),length(Sims_Conc(1,:))/2);
            for i=1:length(Sims_Conc(1,:))/2
                Eq_Pot_n(:,i)=U_n(Sims_Conc(:,(i+i)),c_max_n);
            end
            Eq_Pot_p=U_p(CATH_SConc,c_max_p);

        %Overpotentials
            eta_n=zeros(length(t),length(Sims_Conc(1,:))/2);
            eta_p=zeros(length(t),1);
            for i=1:length(Sims_Conc(1,:))/2
                for j=1:length(Sims_Conc(:,1))
                    if c_max_n-Sims_Conc(j,i+i)>0
                    Butler_Volmer_n= @(eta)
k_n.*c_e_n.^alpha_n.*Sims_Conc(j,(i+i)).^...
                        alpha_p.*(c_max_n-
Sims_Conc(j,(i+i))).^alpha_n*(exp(alpha_n.*...
```

```
                        F./(R_gas.*T).*eta)-exp(-
alpha_p.*F./(R_gas.*T).*eta))-I_app/(F*S_n);
                        eta_n(j,i)=fzero(Butler_Volmer_n,0);
                    end
                end
            end

            for i=1:length(CATH_SConc(:,1))
                if c_max_p-CATH_SConc(i,1)>0
                    Butler_Volmer_p= @(eta)
k_p.*c_e_p.^alpha_n.*CATH_SConc(i,1).^...
                        alpha_p.*(c_max_p-
CATH_SConc(i,1)).^alpha_n*(exp(alpha_n.*...
                        F./(R_gas.*T).*eta)-exp(-
alpha_p.*F./(R_gas.*T).*eta))-(-I_app)/(F*S_p);
                        eta_p(i,1)=fzero(Butler_Volmer_p,0);
                end
            end

            %Calculate Overall Cell Potential
                Base_V=(eta_p(:,1)-eta_n(:,1))+(Eq_Pot_p(:,1)-...
                    Eq_Pot_n(:,1));
            %Record Overall Cell Voltage Values
                dlmwrite('Base_V.txt',Base_V,'precision',18)
```

FICKIANPDE.m

```
function FickianPDE
clc, close all, clear all
%   This function file solves the fickian diffusion problem in
spherical
%   coordinates where concentration gradients exist in the radial
direction
%   only. This will output a file meant to be used in the
Data_Processor
%   m-file
%
%   In the form expected by PDEPE, the single PDE is written as
%
%       1        dc_s        d^2c_s         2    dc_s
%      ---       ----    =   ------    +   ---   ----
%      D_Li       dt          dr^2          r     dr
%      ----      ----    -------------    -------------
%       c         u      f(x,t,u,Du/Dx)   s(x,t,u,Du/Dx)


%Declare and define global variables. Operating current, Particle
radius, diffusion
%coefficient,  maximum lithium concentration, initial lithium
%concentration, Faraday's Constant, and active surface area.
    global R_p_p D_s_p c_0_p c_max_p j_n;
    I_app=1.656; %A
    R_p_p=8.5*10^-6; %m
    D_s_p=1*10^-14;
    c_max_p=51410; %mol/m^3
    c_0_p=.4952*c_max_p;
    F=96487; %C/mol
    S_p=1.1167; %m^2
    j_n=-I_app/(F*S_p);

%PDEPE expects a particular power on the spacial variable. Define 'm'
    m = 0;

%Define spacial and time limits
    r = linspace(0,R_p_p,100);
    t = 0:.1:4298.5;

%Solve the PDE
    options=odeset('RelTol',1*10^-6,'AbsTol',1*10^-12);
    sol = pdepe(m,@pdex1pde,@pdex1ic,@pdex1bc,r,t,options);

%Extract Solution
    c_s = sol(:,:,1);

%Plot final concentration profile
    figure(1)
    plot(r, c_s(end,:))
```

```
    title('Final Concentration Profile')
    xlabel('Radial Position (m)')
    ylabel('Concentration (mol/m^3)')

%Calculate and plot mass conservation vs. time
    Mass_NOM=zeros(1,length(t));
    Mass_Fick=zeros(1,length(t));
    sum=0;
    for i=1:length(t)
        Mass_Fick(i)=trapz(r,4*pi*r.^2.*c_s(i,:));
        Mass_NOM(i)=4/3*pi*R_p_p^3*c_0_p-4*pi*R_p_p^2*(t(i)-t(1))*j_n;
        sum=sum+(Mass_Fick(i)-Mass_NOM(i))^2;
    end
    RMS=sqrt(sum);
    figure(2)
    hold on
    plot(t,Mass_NOM,'r')
    plot(t,Mass_Fick,'k--')
    title('Mass Conservation')
    xlabel('Time (s)')
    ylabel('Moles of Li (mol)')
    str=['Fickain, RMS=',num2str(RMS)];
    legend('Theoretical',str)

%Record Average and Surface Concentrations
    c_s_avg=zeros(1,length(t));
    for i=1:length(t)
        c_s_avg(1,i)=trapz(r,4*pi*r.^2.*c_s(i,:));
    end
    dlmwrite('CATH_SConc.txt', c_s(:,end));
    dlmwrite('CATH_AConc.txt',c_s_avg);
    dlmwrite('CATH_FConc.txt',c_s(end,:));
    t(end)
%Construct the PDE as Matlab requires
function [c,f,s] = pdex1pde(r,t,c_s,Dc_sDr)
    global D_s_p
    c = 1/D_s_p;
    f = Dc_sDr;
    s = 2/r.*Dc_sDr;

%Set initial Concentration in the Sphere
function c_s_0 = pdex1ic(r)
    global c_0_p dr
    c_s_0 = c_0_p; %mol/m^3

%Set Boundary Conditions on the 'left' and 'right' sides
%NOTE: MUST define the pore wall flux of Li 'j_n'
function [pl,ql,pr,qr] = pdex1bc(rl,c_sl,rr,c_sr,t)
    global D_s_p c_0 j_n
    c_s_0=c_0; %mol/m^3
    pl = c_sl.*0;
    ql = D_s_p;
    pr = j_n;
```

```
qr = D_s_p;
```

FICKIANPDERS.m

```
function FickianPDERS
clc, close all, clear all
%   This function file solves the fickian diffusion problem in
spherical
%   coordinates where concentration gradients exist in the radial
direction
%   only. This will output a file meant to be used in the
Data_Processor
%   m-file
%
%   In the form expected by PDEPE, the single PDE is written as
%
%       1        dc_s        d^2c_s         2    dc_s
%      ---       ----    =   ------    +   ---   ----
%      D_Li       dt          dr^2          r     dr
%      ----      ----    -------------    -------------
%       c         u      f(x,t,u,Du/Dx)   s(x,t,u,Du/Dx)


%Declare and define global variables. Particle radius, diffusion
%coefficient, initial lithium concentration, maximum lithium
concentration,
% Faraday's Constant, the universal gas constant, and operating
temperature.
    global R_p_p D_s_p c_0_p c_max_p j_n dr;
    I_app=0; %A
    R_p_p=8.5*10^-6; %m
    c_max_p=51410; %mol/m^3
    D_s_p=1*10^-14;
    c_0_p=dlmread('CATH_IConcB.txt');
    S_p=1.1167; %m^2
    F=96487; %C/mol
    j_n=-I_app/(F*S_p);

%PDEPE expects a particular power on the spacial variable. Define 'm'
    m = 0;

%Define spacial and time limits
    r = linspace(0,R_p_p,100);
    t = (4298.3):.1:(4298.3+3600);
    dr = R_p_p/99;

%Solve the PDE
    options=odeset('RelTol',1*10^-6,'AbsTol',1*10^-12);
    sol = pdepe(m,@pdex1pde,@pdex1ic,@pdex1bc,r,t,options);

%Extract Solution
    c_s = sol(:,:,1);

%Plot final concentration profile
```

```
    figure(1)
    plot(r, c_s(end,:))
    title('Final Concentration Profile')
    xlabel('Radial Position (m)')
    ylabel('Concentration (mol/m^3)')

%Calculate and plot mass conservation vs. time
    Mass_NOM=zeros(1,length(t));
    Mass_Fick=zeros(1,length(t));
    sum=0;
    for i=1:length(t)
        Mass_Fick(i)=trapz(r,4*pi*r.^2.*c_s(i,:));
        Mass_NOM(i)=trapz(r,4*pi*r.^2.*c_0_p(1,:))-4*pi*R_p_p^2*(t(i)-
t(1))*j_n;
        sum=sum+(Mass_Fick(i)-Mass_NOM(i))^2;
    end
    RMS=sqrt(sum);
    figure(2)
    hold on
    plot(t,Mass_NOM,'r')
    plot(t,Mass_Fick,'k--')
    title('Mass Conservation')
    xlabel('Time (s)')
    ylabel('Moles of Li (mol)')
    str=['Fickain, RMS=',num2str(RMS)];
    legend('Theoretical',str)

%Record Average and Surface Concentrations
    c_s_avg=zeros(1,length(t));
    for i=1:length(t)
        c_s_avg(1,i)=trapz(r,4*pi*r.^2.*c_s(i,:));
    end
    dlmwrite('CATH_SConcB.txt', c_s(:,end));
    dlmwrite('CATH_AConcB.txt',c_s_avg);
    dlmwrite('CATH_FConcB.txt',c_s(end,:));
    c_s(1,end)

%Construct the PDE as Matlab requires
function [c,f,s] = pdex1pde(r,t,c_s,Dc_sDr)
    global D_s_p
    c = 1/D_s_p;
    f = Dc_sDr;
    s = 2/r.*Dc_sDr;

%Set initial Concentration in the Sphere
function c_s_0 = pdex1ic(r)
    global c_0_p dr
    c_s_0 = c_0_p(round(r/dr)+1); %mol/m^3

%Set Boundary Conditions on the 'left' and 'right' sides
%NOTE: MUST define the pore wall flux of Li 'j_n'
function [pl,ql,pr,qr] = pdex1bc(rl,c_sl,rr,c_sr,t)
```

```
global D_s_p c_0 j_n
pl = c_sl.*0;
ql = D_s_p;
pr = j_n;
qr = D_s_p;
```

mfix.dat

```
! Run-control section

   RUN_NAME              =  'PBED'

   DESCRIPTION           = 'Diffusion in a Packed Bed'
   RUN_TYPE              = 'new'
   UNITS                 = 'SI'
   TIME                  = 0.0                !start time
   TSTOP                 = 7200.0
   DT                    = 0.1               !time step
   ENERGY_EQ             = .FALSE.            !do not solve
energy eq
   SPECIES_EQ            = .FALSE.    .FALSE. !do not solve
species eq
   CALL_USR               = .TRUE.            !Call usr1.f

   DT_FAC = 1.0
   DETECT_STALL = .FALSE.

   TOL_RESID_Scalar      = 1.0e-3
   TOL_RESID             = 1.0e-3

   GRAVITY = 0.0

   DISCRETIZE(1) = 2
   DISCRETIZE(3) = 2
   DISCRETIZE(4) = 2

   DEF_COR       = .TRUE.
   FPFOI         = .FALSE.

   TOL_RESID     = 1.0E-6

   MAX_NIT       = 5000

   MOMENTUM_X_EQ(1) = .FALSE.
   MOMENTUM_Y_EQ(1) = .FALSE.
   MOMENTUM_Z_EQ(1) = .FALSE.

   CYCLIC_X = .TRUE.
   CYCLIC_Y = .TRUE.
   CYCLIC_Z = .TRUE.

!=================================================================
================
! Cartesian Grid - Quadric definition:
```

```
! Quadric surface Normal form :
! f(x,y,z) = lambda_x * x^2 + lambda_y * y^2 + lambda_z * z^2 +
d = 0
! Regions where f(x,y,z) < 0 are part of the computational
domain.
! Regions where f(x,y,z) > 0 are excluded from the computational
domain.
!
! Predefined quadrics: set QUADRIC_FORM to one of the following:
! Plane:                     'PLANE'
! Cylinder (internal flow): 'X_CYL_INT' or 'Y_CYL_INT' or
'Z_CYL_INT'
! Cylinder (external flow): 'X_CYL_EXT' or 'Y_CYL_EXT' or
'Z_CYL_EXT'
! Cone     (internal flow): 'X_CONE'    or 'Y_CONE'    or
'Z_CONE'
!================================================================
================

  CARTESIAN_GRID = .TRUE.

  N_QUADRIC = 17

QUADRIC_FORM(1) = 'Z_CYL_INT'
RADIUS(1) = 1.2866e-005

t_x(1) = 9.875e-005
t_y(1) = 5e-005

QUADRIC_FORM(2) = 'Z_CYL_INT'
RADIUS(2) = 1.6027e-005

t_x(2) = 0.0001175
t_y(2) = 5e-005

QUADRIC_FORM(3) = 'Z_CYL_INT'
RADIUS(3) = 1.2388e-005

t_x(3) = 0.00013625
t_y(3) = 5e-005

QUADRIC_FORM(4) = 'Z_CYL_INT'
RADIUS(4) = 1.0943e-005

t_x(4) = 0.000155
t_y(4) = 5e-005

QUADRIC_FORM(5) = 'Z_CYL_INT'
RADIUS(5) = 1.4415e-005
```

```
t_x(5) = 0.00017375
t_y(5) = 5e-005

QUADRIC_FORM(6) = 'Z_CYL_INT'
RADIUS(6) = 1.5117e-005

t_x(6) = 9.7695e-005
t_y(6) = 7.2362e-005

QUADRIC_FORM(7) = 'Z_CYL_INT'
RADIUS(7) = 1.1353e-005

t_x(7) = 0.00015378
t_y(7) = 6.3032e-005

QUADRIC_FORM(8) = 'Z_CYL_INT'
RADIUS(8) = 1.4354e-005

t_x(8) = 0.00018826
t_y(8) = 6.9325e-005

QUADRIC_FORM(9) = 'Z_CYL_INT'
RADIUS(9) = 1.2744e-005

t_x(9) = 9.9958e-005
t_y(9) = 9.2855e-005

QUADRIC_FORM(10) = 'Z_CYL_INT'
RADIUS(10) = 1.1177e-005

t_x(10) = 0.00014589
t_y(10) = 7.7722e-005

QUADRIC_FORM(11) = 'Z_CYL_INT'
RADIUS(11) = 1.4796e-005

t_x(11) = 0.00019988
t_y(11) = 9.3168e-005

QUADRIC_FORM(12) = 'Z_CYL_INT'
RADIUS(12) = 1.2219e-005

t_x(12) = 9.4785e-005
t_y(12) = 0.000116

QUADRIC_FORM(13) = 'Z_CYL_INT'
RADIUS(13) = 1.2567e-005
```

```
t_x(13) = 0.00014513
t_y(13) = 9.385e-005

QUADRIC_FORM(14) = 'Z_CYL_INT'
RADIUS(14) = 1.2695e-005

t_x(14) = 0.00019988
t_y(14) = 0.00011406

QUADRIC_FORM(15) = 'Z_CYL_INT'
RADIUS(15) = 1.2256e-005

t_x(15) = 9.7536e-005
t_y(15) = 0.00013042

QUADRIC_FORM(16) = 'Z_CYL_INT'
RADIUS(16) = 1.2853e-005

t_x(16) = 0.0001481
t_y(16) = 0.00011474

QUADRIC_FORM(17) = 'Z_CYL_INT'
RADIUS(17) = 1.2383e-005

t_x(17) = 0.00020234
t_y(17) = 0.00013347

N_GROUP = 1
GROUP_SIZE(1) = 17
GROUP_Q(1,1) = 1
GROUP_Q(1,2) = 2
GROUP_Q(1,3) = 3
GROUP_Q(1,4) = 4
GROUP_Q(1,5) = 5
GROUP_Q(1,6) = 6
GROUP_Q(1,7) = 7
GROUP_Q(1,8) = 8
GROUP_Q(1,9) = 9
GROUP_Q(1,10) = 10
GROUP_Q(1,11) = 11
GROUP_Q(1,12) = 12
GROUP_Q(1,13) = 13
GROUP_Q(1,14) = 14
GROUP_Q(1,15) = 15
GROUP_Q(1,16) = 16
GROUP_Q(1,17) = 17
GROUP_RELATION(1) = 'OR'
BC_ID_Q(1) = 12
BC_ID_Q(2) = 12
```

```
BC_ID_Q(3) = 12
BC_ID_Q(4) = 12
BC_ID_Q(5) = 12
BC_ID_Q(6) = 12
BC_ID_Q(7) = 12
BC_ID_Q(8) = 12
BC_ID_Q(9) = 12
BC_ID_Q(10) = 12
BC_ID_Q(11) = 12
BC_ID_Q(12) = 12
BC_ID_Q(13) = 12
BC_ID_Q(14) = 12
BC_ID_Q(15) = 12
BC_ID_Q(16) = 12
BC_ID_Q(17) = 12


  TOL_F = 1.0D-16

  PRINT_WARNINGS = .TRUE.

  PRINT_PROGRESS_BAR = .TRUE.
  WRITE_DASHBOARD = .TRUE.

!=================================================================
================
! VTK file options
!=================================================================
================
  WRITE_VTK_FILES  = .TRUE.
  TIME_DEPENDENT_FILENAME = .TRUE.
  VTK_DT = 1000

! Available flags for VTK_VAR are :
!  1 : Void fraction (EP_g)
!  2 : Gas pressure, solids pressure (P_g, P_star)
!  3 : Gas velocity (U_g, V_g, W_g)
!  4 : Solids velocity (U_s, V_s, W_s)
!  5 : Solids density (ROP_s)
!  6 : Gas and solids temperature (T_g, T_s1, T_s2)
!  7 : Gas and solids mass fractions (X_g, X_s)
!  8 : Granular temperature (G)
! 11 : Turbulence quantities (k and Îµ)
! 12 : Gas Vorticity magnitude and Lambda_2 (VORTICITY,
LAMBDA_2)
!100 : Processor assigned to scalar cell (Partition)
!101 : Boundary condition flag for scalar cell (BC_ID)
```

```
  VTK_VAR = 9 101 102

! Geometry Section

  COORDINATES           = 'cartesian'

  XLENGTH               =  2.4d-4               ! length
  YLENGTH               =  15.0d-5               ! height
  ZLENGTH               =  30.0d-6               ! depth

  IMAX                  =  100             ! cells in i
direction
  JMAX                  =  100             ! cells in j
direction
  !KMAX                  =  50             ! cells in k
direction

  NO_K=.TRUE.

! Scalar field definition

   NSCALAR               = 1
   PHASE4SCALAR(1)       = 0

# GAS SECTION

  NMAX(0) = 1
  MW_avg = 29.0
  MU_g0  = 1.8e-4
  RO_g0  = 1.0d0 !

! Solids-phase Section

  MMAX = 0

! Initial Conditions Section

IC_X_w(1)             =     0.0
IC_X_e(1)             =     2.4d-4
IC_Y_s(1)             =     0.0
IC_Y_n(1)             =     15.0d-5
IC_Z_b(1)             =     0.0
IC_Z_t(1)             =     30.0d-6

IC_EP_g(1)            =     1.0
IC_U_g(1)             =     0.0
IC_V_g(1)             =     0.0
IC_W_g(1)             =     0.0
IC_SCALAR(1,1)        =     23944.7826
```

```
! cut-cell boundary condition
  BC_TYPE(12)           = 'CG_NSW'

  BC_SCALARW(12,1)      = 23944.7826
  BC_HW_SCALAR(12,1)  = 0.0
  BC_C_SCALAR(12,1)     = -5.62e8

!
!  Output Control
!
  OUT_DT                = 10.                   !write text file
CYL.OUT
                                                !  every 10 s
  RES_DT                = 100.0                 !write binary
restart file
                                                !  CYl.RES every
100.0 s
  NLOG                  = 25                    !write logfile
CYL.LOG
                                                !every 25 time
steps
  FULL_LOG              = .TRUE.                !display
residuals on screen

  Resid_string    = "P0", "U0", "V0" , "S0"

!
        ! EP_g P_g       U_g  U_s  ROP_s     T_g  X_g
        !      P_star    V_g  V_s            T_s  X_s     Theta
Scalar
        !                W_g  W_s
 SPX_DT = 100. 100000 100000 100000 100000 100000 100000 100000.
0.1

!  The decomposition in I, J, and K directions for a Distributed
Memory Parallel machine

  NODESI = 1   NODESJ = 1  NODESK = 1


!  Sweep Direction

 LEQ_SWEEP(1) = 'ISIS'
 LEQ_SWEEP(2) = 'ISIS'
 LEQ_SWEEP(3) = 'ISIS'
 LEQ_SWEEP(4) = 'ISIS'
 LEQ_SWEEP(5) = 'ISIS'
```

```
 LEQ_SWEEP(6) = 'ISIS'
 LEQ_SWEEP(7) = 'ISIS'
 LEQ_SWEEP(8) = 'ISIS'
 LEQ_SWEEP(9) = 'ISIS'

! # Maximum iterations for linear solve
```

rrates.f

```
!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvC
!
C
!  Module name: RRATES(IER)
C
!  Purpose: Calculate reaction rates for various reactions in
cell ijk C
!
C
!  Author:                                            Date:
C
!  Reviewer:                                          Date:
C
!
C
!  Revision Number:
C
!  Purpose:
C
!  Author:                                            Date: dd-
mmm-yy  C
!  Reviewer:                                          Date: dd-
mmm-yy  C
!
C
!  Literature/Document References:
C
!
C
!  Variables referenced: MMAX, IJK, T_g, T_s1, D_p, X_g, X_s,
EP_g,    C
!            P_g, HOR_g, HOR_s
C
!
C
!
C
!  Variables modified: M, N, R_gp, R_sp, RoX_gc, RoX_sc,
SUM_R_g,      C
!                      SUM_R_s
C
!
C
!  Local variables:
C
```

```
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
!
!
      SUBROUTINE RRATES(IER)
!...Translated by Pacific-Sierra Research VAST-90 2.06G5
12:17:31  12/09/98
!...Switches: -xf
!-----------------------------------------------
!   M o d u l e s
!-----------------------------------------------
      USE param
      USE param1
      USE parallel
      USE fldvar
      USE rxns
      USE energy
      USE geometry
      USE run
      USE indices
      USE physprop
      USE constant
      USE funits
      USE compar        !//d
      USE sendrecv      !// 400
      IMPLICIT NONE
!-----------------------------------------------
!   G l o b a l   P a r a m e t e r s
!-----------------------------------------------
!-----------------------------------------------
!   D u m m y   A r g u m e n t s
!-----------------------------------------------
!
!                      Error index
      INTEGER          IER
!
!                      Local phase and species indices
      INTEGER          L, LM, M, N

!                      cell index
      INTEGER          IJK

      DOUBLE PRECISION R_tmp(0:MMAX, 0:MMAX), RxH_xfr(0:MMAX,
0:MMAX)
      DOUBLE PRECISION X_tmp(0:MMAX, 0:MMAX, Dimension_n_all)
      DOUBLE PRECISION RXNA, Trxn
      DOUBLE PRECISION, EXTERNAL ::calc_h
```

```
!
!-----------------------------------------------
      INCLUDE 'function.inc'


!*******  REMOVE THE FOLLOWING LINES to activate the routine
**************
! The following section is provided so that species equation
calculations are
! NOT accidentally performed with the default routine.  To
activate this routine
! remove the following two lines and insert information in
sections 1-4.

      IF(CALL_DI.OR.CALL_ISAT) THEN ! These use functions
external to this routine for rates calculations

         RETURN

      ELSE

!        IER = 1
         RETURN

      ENDIF

!*****************************************************************
*********
      R_tmp = UNDEFINED
!
!  ---  Remember to include all the local variables here for
parallel
!  ---- processing
!$omp  parallel do firstprivate(R_tmp), &
!$omp  private(ijk, L, LM, M, N)

      DO IJK = IJKSTART3, IJKEND3

         IF (FLUID_AT(IJK)) THEN
!
!
!  User input is required in sections 1 through 4.
!
!1111111111111111111111111111111111111111111111111111111111111111
1111111111111111
!
! 1. Write the rates of various reactions:
!    Write the reaction rates for each of the reactions as RXNxF
and RXNxB (both
```

```
!    quantities >= 0), where x identifies the reaction, F stands
for forward
!    rate, and B stands for the backward rate.  The rates can be
in
!    g-mole/(cm^3.s) or g/(cm^3.s).  For the sake of clarity,
give the reaction
!    scheme and the units in a comment statement above the rate
expression.
!    The volume (cm^3) is that of the computational cell.
Therefore, for
!    example, the rate term of a gas phase reaction will have a
multiplicative
!    factor of epsilon. Note that X_g and X_s are mass fractions
!
!
!
!22222222222222222222222222222222222222222222222222222222222222
222222222222222
!
! 2. Write the formation and consumption rates of various
species:
!    Obtain the rates of formation and consumption of various
species
!    in g/(cm^3.s) from the rate expressions RXNxF and RXNxB
obtained in the
!    previous section.  Pay attention to the units of RXNxF and
RXNxB.
!    the formation rates for gas species n are added to get R_gp
(IJK, n).
!    All the consumption rates are added and then divided by
X_g(IJK, n) to
!    get RoX_gc(IJK, n).  If X_g(IJK, n) is zero and species n
is likely
!    to be consumed in a reaction then it is recommended that
RoX_gc (IJK, n)
!    be initialized to the derivative of the consumption rate
with respect to
!    X_g at X_g=0.
!    If the slope is not known analytically a small value such
as 1.0e-9 may
!    instead be used.  A similar procedure is used for all the
species in the
!    solids phases also.
!
!  GAS SPECIES
!
!
!  SOLIDS SPECIES
!
```

```
!
!3333333333333333333333333333333333333333333333333333333333333333333
3333333333333333
!
! 3.  Determine the g/(cm^3.s) transferred from one phase to the
other.
!          R_tmp(To phase #, From phase #)
!     e.g. R_tmp(0,1) -  mass generation of gas phase from
solids-1,
!          R_tmp(0,2) -  mass generation of gas phase from
solids-2,
!          R_tmp(1,0) -  mass generation of solid-1 from gas = -
R_tmp(0,1)
!          R_tmp(1,2) -  mass generation of solid-1 from solids-
2.
!     Note, for example, that if gas is generated from solids-1
then
!     R_tmp(0,1) > 0.
!     The R-phase matrix is skew-symmetric and diagonal elements
are not needed.
!     Only one of the two skew-symmetric elements -- e.g.,
R_tmp(0,1) or
!     R_tmp(1,0) -- needs to be specified.
!
!     X_tmp(M,L, N) is the mass fraction of species N in the
material transferred
!     between phase-M and phase-L. If the destination phase is
M, then N is the
!     index of the species in phase-M, otherwise N is the index
of the species
!     in phase-L; e.g. (1) In the reaction C+1/2O2 --> CO,
!     the destination phase is gas phase. Then N will be equal
to the index of CO
!     in gas phase. (2) If H2O is trasferred between liquid and
gas phases either
!     evaporation or condensation, then the index must change
depending upon the
!     direction of mass transfer. For condensation, N is the
species index
!     of H2O in the liquid phase and for evaporation it is the
species index
!     of H2O in the gas phase. Also Sum_over_N (X_tmp(M,L, N))
should be equal to 1.
!
      if(MMAX > 0) then
        R_tmp(0,1) =  ZERO
      X_tmp(0, 1, :) = ZERO
      if(R_tmp(0,1) > 0) then
        X_tmp(0, 1, 1) = 1.0
```

```
      else
        X_tmp(0, 1, 1) = 1.0
      endif
      endif
!
!444444444444444444444444444444444444444444444444444444444444444444
4444444444444444
!
! 4.  Determine the heat of reactions in cal/(cm^3.s) at the
!     temperature T_g or T_s1.  Note that for exothermic
reactions
!     HOR_g (or HOR_s) will be negative. The assignment of heat
of reaction
!     is user defined as it depends upon the microphysics near
the interface,
!     which is averaged out in the multiphase flow equations.
For example,
!     heat of Reaction for the C + O2 reaction is split into
parts;
!     CO formation is assigned to the solid phase and CO2
formation from CO to
!     the gas phase.
!     *** This section is no longer needed as the heats of
reactions are
!         calculated below.  If you need to override the
automatic calculation,
!         comment out the calculations below.
!
!==================================================================
===============
!
!     No user input is required below this line
!------------------------------------------------------------------
--------------
!   Determine g/(cm^3.s) of mass generation for each of the
phases by adding
!   the reaction rates of all the individual species.

            SUM_R_G(IJK) = ZERO
            IF (SPECIES_EQ(0)) THEN
               IF (NMAX(0) > 0) THEN
                  SUM_R_G(IJK) = SUM_R_G(IJK) +
SUM(R_GP(IJK,:NMAX(0))-ROX_GC(&
                     IJK,:NMAX(0))*X_G(IJK,:NMAX(0)))
               ENDIF
          ELSE
            DO M = 1, MMAX
              IF(R_tmp(0,M) .NE. UNDEFINED)THEN
             SUM_R_G(IJK) = SUM_R_G(IJK) + R_tmp(0,M)
```

```
            ELSEIF(R_tmp(M,0) .NE. UNDEFINED)THEN
              SUM_R_G(IJK) = SUM_R_G(IJK) - R_tmp(M,0)
            ENDIF
             ENDDO
             ENDIF
!
             DO M = 1, MMAX
                SUM_R_S(IJK,M) = ZERO
                IF (SPECIES_EQ(M)) THEN
                   IF (NMAX(M) > 0) THEN
                      SUM_R_S(IJK,M) = SUM_R_S(IJK,M) +
SUM(R_SP(IJK,M,:NMAX(M))&
                                  -
ROX_SC(IJK,M,:NMAX(M))*X_S(IJK,M,:NMAX(M)))
                   ENDIF
              ELSE
                DO L = 0, MMAX
                  IF(R_tmp(M,L) .NE. UNDEFINED)THEN
                  SUM_R_s(IJK,M) = SUM_R_s(IJK,M) + R_tmp(M,L)
               ELSEIF(R_tmp(L,M) .NE. UNDEFINED)THEN
                  SUM_R_s(IJK,M) = SUM_R_s(IJK,M) - R_tmp(L,M)
               ENDIF
                ENDDO
                 ENDIF
             END DO

!
!            Calculate the enthalpy of transferred material
!

             DO M = 0, MMAX-1
            DO L = M+1, MMAX
              RxH_xfr(M, L) = zero
              IF(R_tmp(M,L) .NE. UNDEFINED)THEN
               IF(R_tmp(M,L) > ZERO) then ! phase-M is generated
from phase-L
                      DO N = 1, NMAX(M)
                        RxH_xfr(M, L) =  RxH_xfr(M, L) +
R_tmp(M,L) * X_tmp(M,L, N) * &
                                                CALC_H(IJK, M, N)

                      END DO
               else    ! phase-L is generated from phase-M
                      DO N = 1, NMAX(L)
                        RxH_xfr(M, L) =  RxH_xfr(M, L) +
R_tmp(M,L) * X_tmp(M,L, N) * &
                                                CALC_H(IJK, L, N)
                      END DO
               endif
```

```
           ELSEIF(R_tmp(L,M) .NE. UNDEFINED)THEN
              IF(R_tmp(L,M)> ZERO) then ! phase-L is generated
from phase-M
                     DO N = 1, NMAX(L)
                       RxH_xfr(M, L) =  RxH_xfr(M, L) -
R_tmp(L,M) * X_tmp(L,M, N) * &
                                                CALC_H(IJK, L, N)
                     END DO
              else ! phase-M is generated from phase-L
                     DO N = 1, NMAX(M)
                       RxH_xfr(M, L) =  RxH_xfr(M, L) -
R_tmp(L,M) * X_tmp(L,M, N) * &
                                                CALC_H(IJK, M, N)
                     END DO
              endif
             ENDIF
            ENDDO
            END DO

            DO M = 1, MMAX
            DO L = 0, M-1
              RxH_xfr(M, L) = -RxH_xfr(L, M)
            ENDDO
            END DO

!
!           Calculate heats of reactions
!
            HOR_G(IJK) = zero
            DO N = 1, NMAX(0)
              HOR_G(IJK) = HOR_G(IJK) + &
               (R_gp(IJK, N) - RoX_gc(IJK, N) * X_g(IJK, N)) *
CALC_H(IJK, 0, N)
            END DO
            DO L = 1, MMAX
            HOR_G(IJK) = HOR_G(IJK) - RxH_xfr(0, L)
          ENDDO
            IF (UNITS == 'SI') HOR_G(IJK) =
4183.925D0*HOR_G(IJK)    !in J/kg K

            DO M = 1, MMAX
              HOR_s(IJK, M) = zero
              DO N = 1, NMAX(M)
                HOR_s(IJK, M) = HOR_s(IJK, M) + &
             (R_sp(IJK, M, N) - RoX_sc(IJK, M, N) * X_s(IJK, M,
N)) * CALC_H(IJK, M, N)
              END DO
              DO L = 0, MMAX
```

```
             if(M .NE. L) HOR_s(IJK, M) = HOR_s(IJK, M) -
RxH_xfr(M, L)
           ENDDO
              IF (UNITS == 'SI') HOR_s(IJK, M) =
4183.925D0*HOR_s(IJK, M)    !in J/kg K
          END DO
!
!
!     Store R_tmp values in an array.  Only store the upper
triangle without
!     the diagonal of R_tmp array.
!
            DO L = 0, MMAX
               DO M = L + 1, MMAX
                  LM = L + 1 + (M - 1)*M/2
                  IF (R_TMP(L,M) /= UNDEFINED) THEN
                     R_PHASE(IJK,LM) = R_TMP(L,M)
                  ELSE IF (R_TMP(M,L) /= UNDEFINED) THEN
                     R_PHASE(IJK,LM) = -R_TMP(M,L)
                  ELSE
                     CALL START_LOG
                     IF(.not.DMP_LOG)call open_pe_log(ier)
                     WRITE (UNIT_LOG, 1000) L, M
                     CALL END_LOG
                     call mfix_exit(myPE)
                  ENDIF
               END DO
            END DO

         ENDIF
      END DO

 1000 FORMAT(/1X,70('*')//' From: RRATES',/&
         ' Message: Mass transfer between phases ',I2,' and
',I2,&
         ' (R_tmp) not specified',/1X,70('*')/)
      RETURN
      END SUBROUTINE RRATES
```

scalarprop.f

```
!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvC
!
C
!  Module name: Scalar_PROP(IER)
C
!  Purpose: Calculate diffusion coefficeint and sources for
user-defined
!           scalars
!
C
!  Author:
Date:    C
!  Reviewer:
Date:    C
!
C
!
C
!  Literature/Document References:
C
!
C
!  Variables referenced: None
C
!  Variables modified: None
C
!
C
!  Local variables: None
C
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
!
      SUBROUTINE SCALAR_PROP( IER)
!...Translated by Pacific-Sierra Research VAST-90 2.06G5
12:17:31  12/09/98
!...Switches: -xf
!
!-----------------------------------------------
!   M o d u l e s
!-----------------------------------------------
      USE param
      USE param1
```

```
      USE fldvar
      USE physprop
      USE geometry
      USE indices
      USE run
      USE scalars
      USE toleranc
      USE compar
      USE sendrecv
      IMPLICIT NONE
!-----------------------------------------------
!   G l o b a l   P a r a m e t e r s
!-----------------------------------------------
!-----------------------------------------------
!   D u m m y   A r g u m e n t s
!-----------------------------------------------
!
!                      Error index
      INTEGER          IER

      INTEGER          L,IJK
!
!-----------------------------------------------

      INCLUDE 'function.inc'

      IF(NScalar == 0) RETURN
!
!  ---  Remember to include all the local variables here for
parallel
!  ---- processing
!$omp  parallel do private(ijk, L)
      DO IJK = IJKSTART3, IJKEND3
         IF (FLUID_AT(IJK)) THEN
           DO L = 1, NScalar

!            d (Scalar)/dt = S
!            S is linearized as S = Scalar_c - Scalar_p * Scalar
!            Scalar_c and Scalar_p must be >= 0
!            *** Uncomment next two lines ***
             Scalar_c (IJK, L) = ZERO
             Scalar_p (IJK, L) = ZERO
!
!            Diffusion coefficient for User-defined Scalars
!            *** Uncomment next one line ***
!              Dif_Scalar(IJK, L) = 1.0e-6
             Dif_Scalar(IJK, L) = 3.9e-14 !Anode
!             Dif_Scalar(IJK, L) = 1.0e-14 !Cathode
           END DO
```

```
!
         ENDIF
      END DO
!\\Sendrecv operations - just to make sure all the variables
computed are
!  are passed and updated locally - fool-proof approach -
Sreekanth - 102199

!      call send_recv(Scalar_c,2)
!      call send_recv(Scalar_p,2)
!      call send_recv(Dif_Scalar,2)
      RETURN
      END SUBROUTINE SCALAR_PROP
```

usr0.f

```
!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvC
!
C
!  Module name: USR0
C
!  Purpose: This routine is called before the time loop starts
and is  C
!           user-definable.  The user may insert code in this
routine  C
!           or call appropriate user defined subroutines.  This
C
!           can be used for setting constants and checking
errors in   C
!           data.  This routine is not called from an IJK loop,
hence  C
!           all indices are undefined.
C
!
C
!  Author:                                              Date: dd-
mmm-yy  C
!  Reviewer:                                            Date: dd-
mmm-yy  C
!
C
!  Revision Number:
C
!  Purpose:
C
!  Author:                                              Date: dd-
mmm-yy  C
!  Reviewer:                                            Date: dd-
mmm-yy  C
!
C
!  Literature/Document References:
C
!
C
!  Variables referenced:
C
!  Variables modified:
C
!
C
```

```
!  Local variables:
C
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
!
      SUBROUTINE USR0
!...Translated by Pacific-Sierra Research VAST-90 2.06G5
12:17:31  12/09/98
!...Switches: -xf
      Use usr
      USE param
      USE param1
      USE parallel
      USE matrix
      USE scales
      USE constant
      USE physprop
      USE fldvar
      USE visc_s
      USE rxns
      USE run
      USE toleranc
      USE geometry
      USE indices
      USE is
      USE tau_s
      USE bc
      USE compar
      USE sendrecv
      use kintheory
      USE ghdtheory
      USE drag
      USE cutcell
      USE quadric
      USE scalars
      IMPLICIT NONE
!-----------------------------------------------
!
!  Include files defining common blocks here
!
!
!  Define local variables here

      INTEGER :: I,J,K,IJK
      INTEGER :: BCV,ACTIVE_ZONE,FZ
      INTEGER, PARAMETER :: MAX_ZONES = 1000
```

```
      DOUBLE PRECISION, DIMENSION(MAX_ZONES) ::VOL_FLUID_ZONE  !
Volume in 3D, Surface area times ZLENGTH in 2D
      DOUBLE PRECISION, DIMENSION(MAX_ZONES) ::AREA_FLUID_ZONE !
Surface area in 3D, Perimeter times ZLENGTH in 2D
!
!
!  Include files defining statement functions here
!
      INCLUDE 'fun_avg1.inc'
      INCLUDE 'function.inc'
!
!  Insert user-defined code here


      FLUID_ZONE = 0

      DO K = KSTART3, KEND3            ! Loop through cells and
assign a fluid zone ID to contiguous cells
         DO J = JSTART3, JEND3         ! in regions surrounded
by the same BC_ID value
            DO I = ISTART3, IEND3      ! Works only for closed
domains !!

               IJK = FUNIJK(I,J,K)

               BCV = BC_ID(IJK)

               IF(BCV > 0 ) ACTIVE_ZONE = BCV

               IF(FLUID_AT(IJK)) FLUID_ZONE(IJK) = ACTIVE_ZONE

            END DO
         END DO
      ENDDO


      VOL_FLUID_ZONE = ZERO
      AREA_FLUID_ZONE  = ZERO


      DO IJK = IJKSTART3, IJKEND3    ! Compute volume and and
surface area of boundary for each fluid zone identified above

         FZ = FLUID_ZONE(IJK)

         IF(FZ>0) THEN

            VOL_FLUID_ZONE(FZ) = VOL_FLUID_ZONE(FZ) + VOL(IJK)
```

```
            IF(CUT_CELL_AT(IJK)) AREA_FLUID_ZONE(FZ) =
AREA_FLUID_ZONE(FZ) + AREA_CUT(IJK)

         ENDIF

      END DO


WRITE(*,100)'=================================================
==========================='
      WRITE(*,*)' FLUID ZONE   VOLUME          AREA'
      DO FZ = 1,MAX_ZONES
         IF (VOL_FLUID_ZONE(FZ)>ZERO)
WRITE(*,110)FZ,VOL_FLUID_ZONE(FZ),AREA_FLUID_ZONE(FZ)
      ENDDO

WRITE(*,100)'=================================================
==========================='



!
      RETURN

100   FORMAT(1X,A)
110   FORMAT(1X,I4,10X,E14.8,2X,E14.8)

      END SUBROUTINE USR0
```

usr1.f

```
!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvC
!
C
!  Module name: USR1
C
!  Purpose: This routine is called after the time loop ends and
is
!           user-definable.  The user may insert code in this
routine
!           or call appropriate user defined subroutines.
!           This routine is not called from an IJK loop, hence
!           all indices are undefined.
C
!
C
!  Author:                                          Date: dd-
mmm-yy  C
!  Reviewer:                                        Date: dd-
mmm-yy  C
!
C
!  Revision Number:
C
!  Purpose:
C
!  Author:                                          Date: dd-
mmm-yy  C
!  Reviewer:                                        Date: dd-
mmm-yy  C
!
C
!  Literature/Document References:
C
!
C
!  Variables referenced:
C
!  Variables modified:
C
!
C
!  Local variables:
C
!
C
```

```
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
!
      SUBROUTINE USR1
!...Translated by Pacific-Sierra Research VAST-90 2.06G5
12:17:31  12/09/98
!...Switches: -xf
      USE param
      USE param1
      USE parallel
      USE matrix
      USE scales
      USE constant
      USE physprop
      USE fldvar
      USE visc_s
      USE rxns
      USE run
      USE toleranc
      USE geometry
      USE indices
      USE is
      USE tau_s
      USE bc
      USE compar
      USE sendrecv
      use kintheory
      USE ghdtheory
      USE drag
      USE cutcell
      USE quadric
      USE scalars
      IMPLICIT NONE
!-----------------------------------------------
!
!  Include files defining common blocks here
!
!
!  Define local variables here
!
      INTEGER IJK, BCV
      DOUBLE PRECISION VOL_TOTAL, WTSUM_CONC, AVGSURF_CONC
      CHARACTER(LEN=9) :: BCT
!
!  Include files defining statement functions here
!
      INCLUDE 'fun_avg1.inc'
      INCLUDE 'function.inc'
!
```

```
!  Insert user-defined code here
!
!  Loop over all cells looking for boundary cells. Compute the
volume weighted
!  average concentration of those cells.

   VOL_TOTAL = 0.0
   WTSUM_CONC  = 0.0
   AVGSURF_CONC = 0.0

      DO IJK = IJKSTART3, IJKEND3

!         BCV = BC_V_ID(IJK)
         BCV = BC_ID(IJK)

         IF(BCV > 0 ) THEN
            BCT = BC_TYPE(BCV)
         ELSE
            BCT = 'NONE'
         ENDIF

         SELECT CASE (BCT)

            CASE ('CG_NSW')
               VOL_TOTAL = VOL_TOTAL + VOL(IJK)
               WTSUM_CONC = WTSUM_CONC+VOL(IJK)*Scalar(IJK,1)

         END SELECT

      END DO

      AVGSURF_CONC = WTSUM_CONC/VOL_TOTAL

!  Write the results to a text file
      Open(5,File='AVGSURF_CONC.dat',position ='append')
      write(5,'(//f12.5//)'), AVGSURF_CONC
      Close(5)
      RETURN
      END SUBROUTINE USR1
```

allocate_cut_cell_arrays.f

```
      SUBROUTINE ALLOCATE_CUT_CELL_ARRAYS

!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
!
!  Module name: ALLOCATE_ARRAYS
!  Purpose: allocate arrays
!
C
!  Author: Jeff Dietiker                                Date: 21-
Feb-08  C
!  Reviewer:
!
!-----------------------------------------------
!   M o d u l e s
!-----------------------------------------------
      USE param
      USE param1
      Use indices

      USE cutcell
      USE stl

      IMPLICIT NONE

      DIMENSION_MAX_CUT_CELL =
INT(FAC_DIM_MAX_CUT_CELL*DIMENSION_3G)

      Allocate(  INTERIOR_CELL_AT  (DIMENSION_3) )

      Allocate( XG_E(0:DIMENSION_I) )
      Allocate( YG_N(0:DIMENSION_J) )
      Allocate( ZG_T(0:DIMENSION_K) )

      Allocate(  X_U (DIMENSION_3) )
      Allocate(  Y_U (DIMENSION_3) )
      Allocate(  Z_U (DIMENSION_3) )

      Allocate(  X_V (DIMENSION_3) )
      Allocate(  Y_V (DIMENSION_3) )
      Allocate(  Z_V (DIMENSION_3) )

      Allocate(  X_W (DIMENSION_3) )
      Allocate(  Y_W (DIMENSION_3) )
      Allocate(  Z_W (DIMENSION_3) )
```

```
Allocate(  INTERSECT_X  (DIMENSION_3) )
Allocate(  INTERSECT_Y  (DIMENSION_3) )
Allocate(  INTERSECT_Z  (DIMENSION_3) )

Allocate(  X_int (DIMENSION_3) )
Allocate(  Y_int (DIMENSION_3) )
Allocate(  Z_int (DIMENSION_3) )

Allocate(  X_NEW_POINT  (DIMENSION_MAX_CUT_CELL) )
Allocate(  Y_NEW_POINT  (DIMENSION_MAX_CUT_CELL) )
Allocate(  Z_NEW_POINT  (DIMENSION_MAX_CUT_CELL) )

Allocate(  X_NEW_U_POINT  (DIMENSION_MAX_CUT_CELL) )
Allocate(  Y_NEW_U_POINT  (DIMENSION_MAX_CUT_CELL) )
Allocate(  Z_NEW_U_POINT  (DIMENSION_MAX_CUT_CELL) )

Allocate(  X_NEW_V_POINT  (DIMENSION_MAX_CUT_CELL) )
Allocate(  Y_NEW_V_POINT  (DIMENSION_MAX_CUT_CELL) )
Allocate(  Z_NEW_V_POINT  (DIMENSION_MAX_CUT_CELL) )

Allocate(  X_NEW_W_POINT  (DIMENSION_MAX_CUT_CELL) )
Allocate(  Y_NEW_W_POINT  (DIMENSION_MAX_CUT_CELL) )
Allocate(  Z_NEW_W_POINT  (DIMENSION_MAX_CUT_CELL) )

Allocate(  NUMBER_OF_NODES  (DIMENSION_3) )
Allocate(  NUMBER_OF_U_NODES  (DIMENSION_3) )
Allocate(  NUMBER_OF_V_NODES  (DIMENSION_3) )
Allocate(  NUMBER_OF_W_NODES  (DIMENSION_3) )

Allocate(  CONNECTIVITY  (DIMENSION_3,15) )
Allocate(  CONNECTIVITY_U  (DIMENSION_3,15) )
Allocate(  CONNECTIVITY_V  (DIMENSION_3,15) )
Allocate(  CONNECTIVITY_W  (DIMENSION_3,15) )

Allocate(  PARTITION  (DIMENSION_3) )

Allocate(  WALL_U_AT (DIMENSION_3) )
Allocate(  WALL_V_AT (DIMENSION_3) )
Allocate(  WALL_W_AT (DIMENSION_3) )

Allocate( Area_CUT  (DIMENSION_3) )
Allocate( Area_U_CUT  (DIMENSION_3) )
Allocate( Area_V_CUT  (DIMENSION_3) )
Allocate( Area_W_CUT  (DIMENSION_3) )


Allocate( DELX_Ue  (DIMENSION_3) )
Allocate( DELX_Uw  (DIMENSION_3) )
```

```
Allocate( DELY_Un  (DIMENSION_3) )
Allocate( DELY_Us  (DIMENSION_3) )
Allocate( DELZ_Ut  (DIMENSION_3) )
Allocate( DELZ_Ub  (DIMENSION_3) )

Allocate( DELX_Ve  (DIMENSION_3) )
Allocate( DELX_Vw  (DIMENSION_3) )
Allocate( DELY_Vn  (DIMENSION_3) )
Allocate( DELY_Vs  (DIMENSION_3) )
Allocate( DELZ_Vt  (DIMENSION_3) )
Allocate( DELZ_Vb  (DIMENSION_3) )

Allocate( DELX_We  (DIMENSION_3) )
Allocate( DELX_Ww  (DIMENSION_3) )
Allocate( DELY_Wn  (DIMENSION_3) )
Allocate( DELY_Ws  (DIMENSION_3) )
Allocate( DELZ_Wt  (DIMENSION_3) )
Allocate( DELZ_Wb  (DIMENSION_3) )

Allocate( X_U_ec  (DIMENSION_3) )
Allocate( Y_U_ec  (DIMENSION_3) )
Allocate( Z_U_ec  (DIMENSION_3) )
Allocate( X_U_nc  (DIMENSION_3) )
Allocate( Y_U_nc  (DIMENSION_3) )
Allocate( Z_U_nc  (DIMENSION_3) )
Allocate( X_U_tc  (DIMENSION_3) )
Allocate( Y_U_tc  (DIMENSION_3) )
Allocate( Z_U_tc  (DIMENSION_3) )

Allocate( X_V_ec  (DIMENSION_3) )
Allocate( Y_V_ec  (DIMENSION_3) )
Allocate( Z_V_ec  (DIMENSION_3) )
Allocate( X_V_nc  (DIMENSION_3) )
Allocate( Y_V_nc  (DIMENSION_3) )
Allocate( Z_V_nc  (DIMENSION_3) )
Allocate( X_V_tc  (DIMENSION_3) )
Allocate( Y_V_tc  (DIMENSION_3) )
Allocate( Z_V_tc  (DIMENSION_3) )

Allocate( X_W_ec  (DIMENSION_3) )
Allocate( Y_W_ec  (DIMENSION_3) )
Allocate( Z_W_ec  (DIMENSION_3) )
Allocate( X_W_nc  (DIMENSION_3) )
Allocate( Y_W_nc  (DIMENSION_3) )
Allocate( Z_W_nc  (DIMENSION_3) )
Allocate( X_W_tc  (DIMENSION_3) )
Allocate( Y_W_tc  (DIMENSION_3) )
Allocate( Z_W_tc  (DIMENSION_3) )
```

```
Allocate( DELH_U  (DIMENSION_3) )
Allocate( Theta_Ue  (DIMENSION_3) )
Allocate( Theta_Ue_bar (DIMENSION_3) )
Allocate( Theta_U_ne  (DIMENSION_3) )
Allocate( Theta_U_nw  (DIMENSION_3) )
Allocate( Theta_U_te  (DIMENSION_3) )
Allocate( Theta_U_tw  (DIMENSION_3) )
Allocate( ALPHA_Ue_c  (DIMENSION_3) )
Allocate( NOC_U_E  (DIMENSION_3) )
Allocate( Theta_Un  (DIMENSION_3) )
Allocate( Theta_Un_bar (DIMENSION_3) )
Allocate( ALPHA_Un_c  (DIMENSION_3) )
Allocate( NOC_U_N  (DIMENSION_3) )
Allocate( Theta_Ut  (DIMENSION_3) )
Allocate( Theta_Ut_bar (DIMENSION_3) )
Allocate( ALPHA_Ut_c  (DIMENSION_3) )
Allocate( NOC_U_T  (DIMENSION_3) )
Allocate( A_UPG_E (DIMENSION_3) )
Allocate( A_UPG_W (DIMENSION_3) )

Allocate( DELH_V  (DIMENSION_3) )
Allocate( Theta_V_ne  (DIMENSION_3) )
Allocate( Theta_V_se  (DIMENSION_3) )
Allocate( Theta_Vn  (DIMENSION_3) )
Allocate( Theta_Vn_bar (DIMENSION_3) )
Allocate( Theta_V_nt  (DIMENSION_3) )
Allocate( Theta_V_st (DIMENSION_3) )
Allocate( Theta_Ve  (DIMENSION_3) )
Allocate( Theta_Ve_bar (DIMENSION_3) )
Allocate( ALPHA_Ve_c  (DIMENSION_3) )
Allocate( NOC_V_E  (DIMENSION_3) )
Allocate( ALPHA_Vn_c  (DIMENSION_3) )
Allocate( NOC_V_N  (DIMENSION_3) )
Allocate( Theta_Vt  (DIMENSION_3) )
Allocate( Theta_Vt_bar (DIMENSION_3) )
Allocate( ALPHA_Vt_c  (DIMENSION_3) )
Allocate( NOC_V_T  (DIMENSION_3) )
Allocate( A_VPG_N (DIMENSION_3) )
Allocate( A_VPG_S (DIMENSION_3) )

Allocate( DELH_W (DIMENSION_3) )
Allocate( Theta_W_te (DIMENSION_3) )
Allocate( Theta_W_be (DIMENSION_3) )
Allocate( Theta_W_tn (DIMENSION_3) )
Allocate( Theta_W_bn (DIMENSION_3) )
Allocate( Theta_Wt (DIMENSION_3) )
Allocate( Theta_Wt_bar (DIMENSION_3) )
Allocate( Theta_We (DIMENSION_3) )
Allocate( Theta_We_bar (DIMENSION_3) )
```

```
Allocate( ALPHA_We_c (DIMENSION_3) )
Allocate( NOC_W_E (DIMENSION_3) )
Allocate( Theta_Wn (DIMENSION_3) )
Allocate( Theta_Wn_bar (DIMENSION_3) )
Allocate( ALPHA_Wn_c (DIMENSION_3) )
Allocate( NOC_W_N (DIMENSION_3) )
Allocate( ALPHA_Wt_c (DIMENSION_3) )
Allocate( NOC_W_T (DIMENSION_3) )
Allocate( A_WPG_T (DIMENSION_3) )
Allocate( A_WPG_B (DIMENSION_3) )


Allocate( NORMAL_S (DIMENSION_3,3) )
Allocate( NORMAL_U (DIMENSION_3,3) )
Allocate( NORMAL_V (DIMENSION_3,3) )
Allocate( NORMAL_W (DIMENSION_3,3) )

Allocate( REFP_S (DIMENSION_3,3) )
Allocate( REFP_U (DIMENSION_3,3) )
Allocate( REFP_V (DIMENSION_3,3) )
Allocate( REFP_W (DIMENSION_3,3) )

Allocate(  ONEoDX_E_U (DIMENSION_3) )
Allocate(  ONEoDY_N_U (DIMENSION_3) )
Allocate(  ONEoDZ_T_U (DIMENSION_3) )

Allocate(  ONEoDX_E_V (DIMENSION_3) )
Allocate(  ONEoDY_N_V (DIMENSION_3) )
Allocate(  ONEoDZ_T_V (DIMENSION_3) )

Allocate(  ONEoDX_E_W (DIMENSION_3) )
Allocate(  ONEoDY_N_W (DIMENSION_3) )
Allocate(  ONEoDZ_T_W (DIMENSION_3) )

Allocate(  Xn_int (DIMENSION_3) )
Allocate(  Xn_U_int (DIMENSION_3) )
Allocate(  Xn_V_int (DIMENSION_3) )
Allocate(  Xn_W_int (DIMENSION_3) )

Allocate(  Ye_int (DIMENSION_3) )
Allocate(  Ye_U_int (DIMENSION_3) )
Allocate(  Ye_V_int (DIMENSION_3) )
Allocate(  Ye_W_int (DIMENSION_3) )

Allocate(  Zt_int (DIMENSION_3) )
Allocate(  Zt_U_int (DIMENSION_3) )
Allocate(  Zt_V_int (DIMENSION_3) )
Allocate(  Zt_W_int (DIMENSION_3) )
```

```
Allocate(  SNAP (DIMENSION_3) )


Allocate(  CUT_TREATMENT_AT (DIMENSION_3) )
Allocate(  CUT_U_TREATMENT_AT (DIMENSION_3) )
Allocate(  CUT_V_TREATMENT_AT (DIMENSION_3) )
Allocate(  CUT_W_TREATMENT_AT (DIMENSION_3) )

Allocate(  CUT_CELL_AT (DIMENSION_3) )
Allocate(  CUT_U_CELL_AT (DIMENSION_3) )
Allocate(  CUT_V_CELL_AT (DIMENSION_3) )
Allocate(  CUT_W_CELL_AT (DIMENSION_3) )


Allocate( SMALL_CELL_AT  (DIMENSION_3) )

Allocate( SMALL_CELL_FLAG  (DIMENSION_3) )

Allocate(  BLOCKED_CELL_AT (DIMENSION_3) )
Allocate(  BLOCKED_U_CELL_AT (DIMENSION_3) )
Allocate(  BLOCKED_V_CELL_AT (DIMENSION_3) )
Allocate(  BLOCKED_W_CELL_AT (DIMENSION_3) )

Allocate(  STANDARD_CELL_AT (DIMENSION_3) )
Allocate(  STANDARD_U_CELL_AT (DIMENSION_3) )
Allocate(  STANDARD_V_CELL_AT (DIMENSION_3) )
Allocate(  STANDARD_W_CELL_AT (DIMENSION_3) )


Allocate(  VORTICITY (DIMENSION_3) )
Allocate(  LAMBDA2 (DIMENSION_3) )

Allocate(  TRD_G_OUT (DIMENSION_3) )
Allocate(  PP_G_OUT (DIMENSION_3) )
Allocate(  EPP_OUT (DIMENSION_3) )

Allocate(  dudx_OUT (DIMENSION_3) )
Allocate(  dvdy_OUT (DIMENSION_3) )
Allocate(  delv_OUT (DIMENSION_3) )

Allocate(  U_MASTER_OF (DIMENSION_3) )
Allocate(  V_MASTER_OF (DIMENSION_3) )
Allocate(  W_MASTER_OF (DIMENSION_3) )

Allocate(  BC_ID (DIMENSION_3) )
Allocate(  BC_U_ID (DIMENSION_3) )
Allocate(  BC_V_ID (DIMENSION_3) )
Allocate(  BC_W_ID (DIMENSION_3) )
```

```
Allocate(  DEBUG_CG (DIMENSION_3,15) )

Allocate(  U_g_CC (DIMENSION_3) )
Allocate(  V_g_CC (DIMENSION_3) )
Allocate(  W_g_CC (DIMENSION_3) )

Allocate(  U_s_CC (DIMENSION_3, DIMENSION_M) )
Allocate(  V_s_CC (DIMENSION_3, DIMENSION_M) )
Allocate(  W_s_CC (DIMENSION_3, DIMENSION_M) )


ALLOCATE(N_FACET_AT(DIMENSION_3))
ALLOCATE(LIST_FACET_AT(DIMENSION_3,10))


Allocate(  FLUID_ZONE (DIMENSION_3) )

RETURN
END SUBROUTINE ALLOCATE_CUT_CELL_ARRAYS
```

cutcell_mod.f

```
      MODULE cutcell

      Use param
      Use param1
      USE progress_bar

!     CUT_CELL.LOG unit number
      INTEGER  UNIT_CUT_CELL_LOG
      PARAMETER (UNIT_CUT_CELL_LOG = 111)

!     Flag to activate Cartesian grid

      LOGICAL :: CARTESIAN_GRID

!     maximum number of cut cells
      INTEGER :: DIMENSION_MAX_CUT_CELL

!     Factor used to allocate cut cells arrays
      DOUBLE PRECISION :: FAC_DIM_MAX_CUT_CELL

!     Flag to identify interior cells
      LOGICAL, DIMENSION(:), ALLOCATABLE :: INTERIOR_CELL_AT

!     One-Dimensional Arrays for East, North, Top location of
!     original (uncut) scalar cells
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::XG_E
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::YG_N
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::ZG_T

!     location of U-momentum nodes
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_U
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_U
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_U

!     location of V-momentum nodes
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_V
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_V
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_V

!     location of W-momentum nodes
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_W
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_W
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_W

!     Intersection flags
      LOGICAL, DIMENSION(:), ALLOCATABLE :: INTERSECT_X
```

```
      LOGICAL, DIMENSION(:), ALLOCATABLE :: INTERSECT_Y
      LOGICAL, DIMENSION(:), ALLOCATABLE :: INTERSECT_Z

!     Location of intersections
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_int
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_int
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_int

!     Location of original (uncut) corner cell nodes
      DOUBLE PRECISION, DIMENSION(0:15):: X_NODE
      DOUBLE PRECISION, DIMENSION(0:15):: Y_NODE
      DOUBLE PRECISION, DIMENSION(0:15):: Z_NODE
      DOUBLE PRECISION, DIMENSION(0:15):: F_NODE
      INTEGER, DIMENSION(0:15) :: IJK_OF_NODE

!     Location of new (along intersecting edges) nodes
      INTEGER :: NUMBER_OF_NEW_POINTS
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
X_NEW_POINT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Y_NEW_POINT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Z_NEW_POINT

!     Location of new (along intersecting edges) nodes
      INTEGER :: NUMBER_OF_NEW_U_POINTS
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
X_NEW_U_POINT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Y_NEW_U_POINT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Z_NEW_U_POINT

!     Location of new (along intersecting edges) nodes
      INTEGER :: NUMBER_OF_NEW_V_POINTS
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
X_NEW_V_POINT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Y_NEW_V_POINT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Z_NEW_V_POINT

!     Location of new (along intersecting edges) nodes
      INTEGER :: NUMBER_OF_NEW_W_POINTS
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
X_NEW_W_POINT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Y_NEW_W_POINT
```

```
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Z_NEW_W_POINT

!     Number of nodes
      INTEGER, DIMENSION(:), ALLOCATABLE ::  NUMBER_OF_NODES
      INTEGER, DIMENSION(:), ALLOCATABLE ::  NUMBER_OF_U_NODES
      INTEGER, DIMENSION(:), ALLOCATABLE ::  NUMBER_OF_V_NODES
      INTEGER, DIMENSION(:), ALLOCATABLE ::  NUMBER_OF_W_NODES

!     Connectivity
      INTEGER, DIMENSION(:,:), ALLOCATABLE :: CONNECTIVITY
      INTEGER, DIMENSION(:,:), ALLOCATABLE :: CONNECTIVITY_U
      INTEGER, DIMENSION(:,:), ALLOCATABLE :: CONNECTIVITY_V
      INTEGER, DIMENSION(:,:), ALLOCATABLE :: CONNECTIVITY_W

!     Processor assign to cell IJK
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: PARTITION

!     Normal Vector Defining cut face in Scalar Cell
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::  NORMAL_S

!     Reference point Defining cut face in Scalar Cell
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::  REFP_S

!     Flags for Wall momentum cells
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  WALL_U_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  WALL_V_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  WALL_W_AT

!     Areas of cut faces
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Area_CUT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Area_U_CUT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Area_V_CUT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Area_W_CUT

!     Distances from cell center to face center
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELX_Ue
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELX_Uw
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELY_Un
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELY_Us
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELZ_Ut
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELZ_Ub

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELX_Ve
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELX_Vw
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELY_Vn
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELY_Vs
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELZ_Vt
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELZ_Vb
```

```
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELX_We
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELX_Ww
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELY_Wn
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELY_Ws
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELZ_Wt
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELZ_Wb

!     Location of face centers
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_U_ec
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_U_ec
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_U_ec

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_U_nc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_U_nc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_U_nc

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_U_tc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_U_tc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_U_tc

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_V_ec
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_V_ec
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_V_ec

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_V_nc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_V_nc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_V_nc

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_V_tc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_V_tc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_V_tc

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_W_ec
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_W_ec
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_W_ec

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_W_nc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_W_nc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_W_nc

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  X_W_tc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Y_W_tc
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Z_W_tc


!     Distance to cut face in U-Momentum Cell
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELH_U
```

```
!     Normal Vector Defining cut face in U-Momentum Cell
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::  NORMAL_U

!     Reference point Defining cut face in W-Momentum Cell
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::  REFP_U


!     Correction factors for U-Momentum Cell
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_Ue
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Theta_Ue_bar

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_U_ne
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_U_nw

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_U_te
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_U_tw

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ALPHA_Ue_c
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  NOC_U_E

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_Un
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Theta_Un_bar

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ALPHA_Un_c
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  NOC_U_N

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_Ut
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Theta_Ut_bar

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ALPHA_Ut_c
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  NOC_U_T

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  A_UPG_E
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  A_UPG_W

!     Distance to cut face in V-Momentum Cell
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELH_V

!     Normal Vector Defining cut face in V-Momentum Cell
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::  NORMAL_V

!     Reference point Defining cut face in V-Momentum Cell
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::  REFP_V


!     Correction factors for V-Momentum Cell
```

```
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_V_ne
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_V_se

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_Vn
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Theta_Vn_bar

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_V_nt
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_V_st

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_Ve
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Theta_Ve_bar

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ALPHA_Ve_c
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  NOC_V_E

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ALPHA_Vn_c
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  NOC_V_N

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_Vt
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Theta_Vt_bar

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ALPHA_Vt_c
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  NOC_V_T

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  A_VPG_N
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  A_VPG_S


!     Distance to cut face in W-Momentum Cell
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  DELH_W

!     Normal Vector Defining cut face in W-Momentum Cell
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::  NORMAL_W

!     Reference point Defining cut face in W-Momentum Cell
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::  REFP_W


!     Correction factors for W-Momentum Cell
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_W_te
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_W_be

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_W_tn
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_W_bn

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_Wt
```

```
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Theta_Wt_bar

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_We
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Theta_We_bar

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ALPHA_We_c
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  NOC_W_E

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Theta_Wn
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
Theta_Wn_bar

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ALPHA_Wn_c
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  NOC_W_N

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ALPHA_Wt_c
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  NOC_W_T

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  A_WPG_T
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  A_WPG_B

!     1/dx, 1/dy, 1/dz
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ONEoDX_E_U
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ONEoDY_N_U
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ONEoDZ_T_U


      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ONEoDX_E_V
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ONEoDY_N_V
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ONEoDZ_T_V

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ONEoDX_E_W
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ONEoDY_N_W
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  ONEoDZ_T_W



      LOGICAL, DIMENSION(:), ALLOCATABLE ::
ALONG_DOMAIN_BOUNDARY

!     Location of intersection points
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Xn_int
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Xn_U_int
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Xn_V_int
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Xn_W_int

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Ye_int
```

```
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Ye_U_int
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Ye_V_int
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Ye_W_int

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Zt_int
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Zt_U_int
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Zt_V_int
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  Zt_W_int

!     Cut cell treatment flags
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  CUT_TREATMENT_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  CUT_U_TREATMENT_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  CUT_V_TREATMENT_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  CUT_W_TREATMENT_AT

!     Various cell flags
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  CUT_CELL_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  CUT_U_CELL_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  CUT_V_CELL_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  CUT_W_CELL_AT

      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  SMALL_CELL_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  BLOCKED_CELL_AT
      INTEGER,  DIMENSION(:), ALLOCATABLE ::  SMALL_CELL_FLAG

      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  BLOCKED_U_CELL_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  BLOCKED_V_CELL_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  BLOCKED_W_CELL_AT

      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  STANDARD_CELL_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  STANDARD_U_CELL_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  STANDARD_V_CELL_AT
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  STANDARD_W_CELL_AT

!     Tolerance for snapping procedure
      DOUBLE PRECISION, DIMENSION(3) :: TOL_SNAP

!     Tolerances for wall distance
      DOUBLE PRECISION :: TOL_DELH

!     Tolerance for detecting small scalar cells
      DOUBLE PRECISION :: TOL_SMALL_CELL
      DOUBLE PRECISION :: TOL_SMALL_AREA
!     Maximum value of ALPHA correction factor
      DOUBLE PRECISION :: ALPHA_MAX

!     Flags to include effect of cut cells
      LOGICAL ::
NOC,NOC_UG,NOC_VG,NOC_WG,NOC_US,NOC_VS,NOC_WS,NOC_TRDG,NOC_TRDS
```

```
      LOGICAL ::
CUT_TAU_UG,CUT_TAU_VG,CUT_TAU_WG,CUT_TAU_US,CUT_TAU_VS,CUT_TAU_W
S

!     pressure gradient option flag
      INTEGER :: PG_OPTION

!     Number of cells
      INTEGER NUMBER_OF_U_CUT_CELLS
      INTEGER NUMBER_OF_V_CUT_CELLS
      INTEGER NUMBER_OF_W_CUT_CELLS
      INTEGER NUMBER_OF_SMALL_CELLS

      INTEGER NUMBER_OF_U_WALL_CELLS
      INTEGER NUMBER_OF_V_WALL_CELLS
      INTEGER NUMBER_OF_W_WALL_CELLS

!     Vorticity and lambda2
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  VORTICITY
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  LAMBDA2

!     Re-ordering array
      INTEGER, DIMENSION(15) :: ORDER

!     Snapping flag
      LOGICAL,  DIMENSION(:), ALLOCATABLE ::  SNAP

      INTEGER, DIMENSION(10) :: CG_SAFE_MODE
      LOGICAL :: PRINT_WARNINGS
      LOGICAL :: SET_CORNER_CELLS

!     Master cell of wall cell (FSW)
      INTEGER,  DIMENSION(:), ALLOCATABLE ::  U_MASTER_OF
      INTEGER,  DIMENSION(:), ALLOCATABLE ::  V_MASTER_OF
      INTEGER,  DIMENSION(:), ALLOCATABLE ::  W_MASTER_OF

      INTEGER :: N_USR_DEF

      LOGICAL :: USE_POLYGON

      LOGICAL :: USE_STL

      LOGICAL :: USE_MSH

!     Boundary condition flag
      INTEGER,  DIMENSION(:), ALLOCATABLE ::  BC_ID
      INTEGER,  DIMENSION(:), ALLOCATABLE ::  BC_U_ID
      INTEGER,  DIMENSION(:), ALLOCATABLE ::  BC_V_ID
      INTEGER,  DIMENSION(:), ALLOCATABLE ::  BC_W_ID
```

```
      INTEGER :: NSW_GHOST_BC_ID

!     Under-relaxation flag applied to cut cells
      DOUBLE PRECISION, DIMENSION(9):: CG_UR_FAC

!     Debugging_variables
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::DEBUG_CG

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::U_g_CC
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::V_g_CC
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::W_g_CC

      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::U_s_CC
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::V_s_CC
      DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE ::W_s_CC

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  TRD_G_OUT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  PP_G_OUT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  EPP_OUT

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  dudx_OUT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  dvdy_OUT
      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::  delv_OUT

!     Fluid zone flag
      INTEGER,  DIMENSION(:), ALLOCATABLE ::  FLUID_ZONE

      END MODULE cutcell
```

vtk_out.f

```
!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvC
!
C
!  Module name: WRITE_VTK_FILE
C
!  Purpose: Writes the cut cell grid in VTK format
C
!
C
!  Author: Jeff Dietiker                              Date: 21-
Feb-08  C
!  Reviewer:                                          Date:
C
!
C
!  Revision Number #                                  Date: ##-
###-##  C
!  Author: #
C
!  Purpose: #
C
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
  SUBROUTINE WRITE_VTK_FILE

      USE param
      USE param1
      USE parallel
      USE constant
      USE run
      USE toleranc
      USE geometry
      USE indices
      USE compar
      USE sendrecv
      USE quadric
      USE cutcell
      USE fldvar
      USE visc_s
      USE physprop
      USE pgcor
      USE vtk
      USE rxns
```

```
      USE output
      USE scalars

      USE pgcor
      USE pscor


      IMPLICIT NONE
      DOUBLE PRECISION:: Xw,Xe,Yn,Ys
      INTEGER :: I,J,K,L,M,N,IM,JM,KM,IP,JP,KP,IJK
      INTEGER :: IMJK,IJMK,IJKM,IMJMK,IMJKM,IJMKM,IMJMKM

      INTEGER sw,se,ne,nw
      INTEGER, DIMENSION(10) :: additional_node
      DOUBLE PRECISION, DIMENSION(2*DIMENSION_3) ::  X_OF
      DOUBLE PRECISION, DIMENSION(2*DIMENSION_3) ::  Y_OF
      DOUBLE PRECISION, DIMENSION(2*DIMENSION_3) ::  Z_OF
      INTEGER, DIMENSION(DIMENSION_3) ::  INDEX_OF_E_ADD_NODE
      INTEGER, DIMENSION(DIMENSION_3) ::  INDEX_OF_N_ADD_NODE
      INTEGER :: SPECIES_COUNTER,LT

      CHARACTER (LEN=32) :: SUBM,SUBN
      CHARACTER (LEN=64) :: VAR_NAME

      DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE ::
DP_BC_ID,DP_FLUID_ZONE

      include "function.inc"

      IF(.NOT.CARTESIAN_GRID) RETURN

      DX(IEND3+1) = DX(IEND3)
      DY(JEND3+1) = DY(JEND3)
      DZ(KEND3+1) = DZ(KEND3)

!     Location of U-momentum cells for original (uncut grid)
      IF (DO_I) THEN
        XG_E(1) = ZERO
        DO I = IMIN1, IMAX2
           XG_E(I) = XG_E(I-1) + DX(I)
        END DO
      ENDIF

!     Location of V-momentum cells for original (uncut grid)
      IF (DO_J) THEN
        YG_N(1) = ZERO
        DO J = JMIN1, JMAX2
           YG_N(J) = YG_N(J-1) + DY(J)
        END DO
```

```
      ENDIF

!     Location of W-momentum cells for original (uncut grid)
      IF (DO_K) THEN
        ZG_T(1) = ZERO
        DO K = KMIN1, KMAX2
           ZG_T(K) = ZG_T(K-1) + DZ(K)
        END DO
      ELSE
         ZG_T = ZERO
      ENDIF

      IF(WRITE_ANI_CUTCELL) THEN
         CALL OPEN_VTK_FILE
         CALL WRITE_GEOMETRY_IN_VTK
         CALL CLOSE_VTK_FILE
         IF (FULL_LOG) THEN
            WRITE(*,30)'WROTE VTK FILE : ani_cutcell.vtk'
         ENDIF
         WRITE_ANI_CUTCELL = .FALSE.
         RETURN
      ENDIF

      CALL OPEN_VTK_FILE

      CALL WRITE_GEOMETRY_IN_VTK

      DO L = 1, DIM_VTK_VAR

         SELECT CASE (VTK_VAR(L))

            CASE (1)
               CALL WRITE_SCALAR_IN_VTK('EP_G',EP_G)
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'

            CASE (2)
               CALL WRITE_SCALAR_IN_VTK('P_G',P_G)
               CALL WRITE_SCALAR_IN_VTK('P_S',P_S)
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'

            CASE (3)
               CALL
WRITE_VECTOR_IN_VTK('Gas_Velocity',U_G,V_G,W_G)
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'

            CASE (4)
               DO M = 1,MMAX
                  WRITE(SUBM,*)M
```

```
                  CALL
WRITE_VECTOR_IN_VTK('Solids_Velocity_'//ADJUSTL(SUBM),U_S(:,M),V
_S(:,M),W_S(:,M))
               END DO
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'

            CASE (5)
               DO M = 1,MMAX
                  WRITE(SUBM,*)M
                  CALL
WRITE_SCALAR_IN_VTK('Solids_density_'//ADJUSTL(SUBM),ROP_S(:,M))
               END DO
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'

            CASE (6)
               CALL WRITE_SCALAR_IN_VTK('Gas_temperature',T_g)
               DO M = 1,MMAX
                  WRITE(SUBM,*)M
                  CALL
WRITE_SCALAR_IN_VTK('Solids_temperature_'//ADJUSTL(SUBM),T_S(:,M
))
               END DO
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'

            CASE (7)
               SPECIES_COUNTER = 0
               DO N = 1,NMAX(0)
                  WRITE(SUBN,*)N
                  SPECIES_COUNTER = SPECIES_COUNTER + 1
                  VAR_NAME =
ADJUSTL(SPECIES_NAME(SPECIES_COUNTER))
                  LT =
LEN_TRIM(ADJUSTL(SPECIES_NAME(SPECIES_COUNTER)))
                  VAR_NAME =
VAR_NAME(1:LT)//'_Gas_mass_fractions_'//ADJUSTL(SUBN)
                  CALL WRITE_SCALAR_IN_VTK(VAR_NAME,X_g(:,N))
               END DO

               DO M = 1, MMAX
                  WRITE(SUBM,*)M
                  DO N = 1,NMAX(M)
                     WRITE(SUBN,*)N
                     SPECIES_COUNTER = SPECIES_COUNTER + 1
                     VAR_NAME =
ADJUSTL(SPECIES_NAME(SPECIES_COUNTER))
                     LT =
LEN_TRIM(ADJUSTL(SPECIES_NAME(SPECIES_COUNTER)))
```

```
                     VAR_NAME =
VAR_NAME(1:LT)//'_Solids_mass_fractions_'//TRIM(ADJUSTL(SUBM))//
'_'//ADJUSTL(SUBN)
                     CALL
WRITE_SCALAR_IN_VTK(VAR_NAME,X_s(:,M,N))
                  END DO
               END DO
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'

            CASE (8)
               DO M = 1,MMAX
                  WRITE(SUBM,*)M
                  CALL
WRITE_SCALAR_IN_VTK('Granular_temperature_'//ADJUSTL(SUBM),Theta
_m(:,M))
               END DO
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'

            CASE (9)
               SPECIES_COUNTER = 0
               DO N = 1,NSCALAR
                  WRITE(SUBN,*)N
                  SPECIES_COUNTER = SPECIES_COUNTER + 1
                  VAR_NAME = 'Scalar_'//ADJUSTL(SUBN)
                  CALL WRITE_SCALAR_IN_VTK(VAR_NAME,Scalar(:,N))
               END DO
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'

            CASE (11)
               IF(K_EPSILON) THEN
                  CALL WRITE_SCALAR_IN_VTK('K_Turb_G',K_Turb_G)
                  CALL WRITE_SCALAR_IN_VTK('E_Turb_G',E_Turb_G)
                  IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'
               ENDIF

            CASE (12)
               CALL CALC_VORTICITY

               CALL
WRITE_SCALAR_IN_VTK('VORTICITY_MAG',VORTICITY)
               CALL WRITE_SCALAR_IN_VTK('LAMBDA_2',LAMBDA2)
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'

            CASE (100)
               CALL WRITE_SCALAR_IN_VTK('PARTITION',PARTITION)
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'

            CASE (101)
```

```
               Allocate(DP_BC_ID(DIMENSION_3))
               DP_BC_ID = DFLOAT(BC_ID)
!               CALL WRITE_SCALAR_IN_VTK('BC_ID',DFLOAT(BC_ID))
               CALL WRITE_SCALAR_IN_VTK('BC_ID',DP_BC_ID)
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'
               DeAllocate(DP_BC_ID)

            CASE (102)

               Allocate(DP_FLUID_ZONE(DIMENSION_3))
               DP_FLUID_ZONE = DFLOAT(FLUID_ZONE)
               CALL
WRITE_SCALAR_IN_VTK('FLUID_ZONE',DP_FLUID_ZONE)
               IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,10)'.'
               DeAllocate(DP_FLUID_ZONE)

            CASE (0) ! do nothing

            CASE (UNDEFINED_I) ! do nothing

            CASE DEFAULT

               WRITE(*,30) ' Unknown VTK variable flag
',L,':',VTK_VAR(L)
               WRITE(*,30) ' Available flags are : '
               WRITE(*,30) ' 1 : Void fraction (EP_g)'
               WRITE(*,30) ' 2 : Gas pressure, solids pressure
(P_g, P_star)'
               WRITE(*,30) ' 3 : Gas velocity (U_g, V_g, W_g)'
               WRITE(*,30) ' 4 : Solids velocity (U_s, V_s,
W_s)'
               WRITE(*,30) ' 5 : Solids density (ROP_s)'
               WRITE(*,30) ' 6 : Gas and solids temperature
(T_g, T_s1, T_s2)'
               WRITE(*,30) ' 7 : Gas and solids mass fractions
(X_g, X-s)'
               WRITE(*,30) ' 8 : Granular temperature (G)'
!               write(*,30) ' 9 : User defined scalars'
!               write(*,30) '10 : Reaction Rates'
               write(*,30) '11 : Turbulence quantities (k and
Îµ)'
               write(*,30) '12 : Gas Vorticity magnitude and
Lambda_2 (VORTICITY, LAMBDA_2)'
               write(*,30) '100: Processor assigned to scalar
cell (Partition)'
               write(*,30) '101: Boundary condition flag for
scalar cell (BC_ID)'
               write(*,30) 'MFiX will exit now.'
               CALL MFIX_EXIT(myPE)
```

```
            END SELECT

      END DO

      CALL CLOSE_VTK_FILE


     IF (FULL_LOG.AND.myPE == PE_IO) WRITE(*,20)' DONE.'

10    FORMAT(A,$)
20    FORMAT(A,1X/)
30    FORMAT(1X,A)
      RETURN

      END SUBROUTINE WRITE_VTK_FILE

!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvC
!
C
!  Module name: OPEN_VTK_FILE
C
!  Purpose: Open a vtk file and writes the header
C
!
C
!  Author: Jeff Dietiker                              Date: 21-
Feb-08  C
!  Reviewer:                                          Date:
C
!
C
!  Revision Number #                                  Date: ##-
###-##  C
!  Author: #
C
!  Purpose: #
C
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
  SUBROUTINE OPEN_VTK_FILE

      USE param
      USE param1
      USE parallel
      USE constant
```

```
      USE run
      USE toleranc
      USE geometry
      USE indices
      USE compar
      USE sendrecv
      USE output
      USE quadric
      USE cutcell
      USE fldvar
      USE vtk

      IMPLICIT NONE
      DOUBLE PRECISION:: Xw,Xe,Yn,Ys
      INTEGER :: I,J,K,L,IM,JM,KM,IP,JP,KP,IJK
      INTEGER :: IMJK,IJMK,IJKM,IMJMK,IMJKM,IJMKM,IMJMKM

      include "function.inc"

      IF (myPE /= PE_IO) RETURN

      IF(.NOT.WRITE_ANI_CUTCELL) THEN

         VTK_FILENAME = TRIM(RUN_NAME)

         IF(TIME_DEPENDENT_FILENAME) THEN
            FRAME = FRAME + 1
            WRITE(FRAME_CHAR,*) FRAME
            FRAME_CHAR = ADJUSTL(FRAME_CHAR)
            VTK_FILENAME = TRIM(VTK_FILENAME) // '_' //
TRIM(FRAME_CHAR) // '.vtk'
         ELSE
            VTK_FILENAME = TRIM(VTK_FILENAME) // '.vtk'
         ENDIF

         IF (FULL_LOG) THEN
            WRITE(*,10)' WRITING VTK FILE : ',
TRIM(VTK_FILENAME),' .'
         ENDIF

      ELSE

         VTK_FILENAME = 'ani_cutcell.vtk'

      ENDIF

      VTK_UNIT   = 123

      OPEN(UNIT     = VTK_UNIT,           &
```

```
           FILE     = TRIM(VTK_FILENAME), &
           FORM     = 'UNFORMATTED',    &  ! works with gfortran
4.3.4 and ifort 10.1 but may not be supported by all compilers
                                           ! use 'BINARY' if
'UNFORMATTED' is not supported
           ACCESS   = 'STREAM',   &        ! works with gfortran
4.3.4 and ifort 10.1 but may not be supported by all compilers
                                           ! use 'SEQUENTIAL' if
'STREAM' is not supported
           ACTION   = 'WRITE', &
           CONVERT  = 'BIG_ENDIAN')

      WRITE(UNIT=VTK_UNIT)'# vtk DataFile Version 2.0'//END_REC
      WRITE(BUFFER,FMT='(A,A,E14.8)')TRIM(RUN_NAME),', Time =
',TIME
      WRITE(UNIT=VTK_UNIT)TRIM(BUFFER)//END_REC
      WRITE(UNIT=VTK_UNIT)TRIM('BINARY')//END_REC
      IF(NO_K) THEN
         WRITE(UNIT=VTK_UNIT)'DATASET POLYDATA'//END_REC
      ELSE
         WRITE(UNIT=VTK_UNIT)'DATASET
UNSTRUCTURED_GRID'//END_REC
      ENDIF

10    FORMAT(/1X,3A,$)

      RETURN

      END SUBROUTINE OPEN_VTK_FILE


!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvC
!
C
!  Module name: WRITE_GEOMETRY_IN_VTK
C
!  Purpose: Write Geometry and connectivity in a vtk file
C
!
C
!  Author: Jeff Dietiker                                  Date: 21-
Feb-08  C
!  Reviewer:                                              Date:
C
!
C
!  Revision Number #                                      Date: ##-
###-##  C
```

```
!  Author: #
C
!  Purpose: #
C
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
  SUBROUTINE WRITE_GEOMETRY_IN_VTK

      USE param
      USE param1
      USE parallel
      USE constant
      USE run
      USE toleranc
      USE geometry
      USE indices
      USE compar
      USE mpi_utility
      USE sendrecv
      USE quadric
      USE cutcell
      USE fldvar
      USE vtk


      IMPLICIT NONE

      INTEGER :: IJK,I,J,K,L
      INTEGER :: IJK_OFFSET

      INTEGER :: iproc,IERR
      INTEGER, DIMENSION(0:numPEs-1) :: disp,rcount
      INTEGER, DIMENSION(:,:), ALLOCATABLE ::
SHIFTED_CONNECTIVITY

      include "function.inc"

      IF (myPE /= PE_IO) RETURN

         NUMBER_OF_VTK_CELLS = NUMBER_OF_CELLS -
NUMBER_OF_BLOCKED_CELLS

         WRITE(BUFFER,FMT='(A,I8,A)')'POINTS
',NUMBER_OF_POINTS,' double'
         WRITE(UNIT=VTK_UNIT)TRIM(BUFFER)//END_REC
```

```
         WRITE(UNIT=VTK_UNIT)
(XG_E(GLOBAL_I_OF(IJK)),YG_N(GLOBAL_J_OF(IJK)),ZG_T(GLOBAL_K_OF(
IJK)), IJK = 1,IJKMAX3), &

(GLOBAL_X_NEW_POINT(IJK),GLOBAL_Y_NEW_POINT(IJK),GLOBAL_Z_NEW_PO
INT(IJK),IJK = 1,&
                              GLOBAL_NUMBER_OF_NEW_POINTS)
         WRITE(UNIT=VTK_UNIT) END_REC

         IF(NO_K) THEN
            WRITE(BUFFER,FMT='(A,2(I8,2X))')'POLYGONS
',NUMBER_OF_VTK_CELLS,POLY_COUNTER
         ELSE
            WRITE(BUFFER,FMT='(A,2(I8,2X))')'CELLS
',NUMBER_OF_VTK_CELLS,POLY_COUNTER
         ENDIF

         WRITE(UNIT=VTK_UNIT)TRIM(BUFFER)//END_REC

         DO IJK = 1,IJKMAX3
            IF (GLOBAL_INTERIOR_CELL_AT(IJK))      THEN
              IF (.NOT.GLOBAL_BLOCKED_CELL_AT(IJK))
WRITE(UNIT=VTK_UNIT) GLOBAL_NUMBER_OF_NODES(IJK),&
              (GLOBAL_CONNECTIVITY(IJK,L)-
1,L=1,GLOBAL_NUMBER_OF_NODES(IJK))
            ENDIF
         END DO
         WRITE(UNIT=VTK_UNIT) END_REC

         IF(DO_K) THEN
            WRITE(BUFFER,FMT='(A,I8)')'CELL_TYPES
',NUMBER_OF_VTK_CELLS
            WRITE(UNIT=VTK_UNIT)TRIM(BUFFER)//END_REC

            DO IJK = 1,IJKMAX3
               IF (GLOBAL_INTERIOR_CELL_AT(IJK))      THEN
                  IF (.NOT.GLOBAL_BLOCKED_CELL_AT(IJK)) THEN
                     IF(GLOBAL_CUT_CELL_AT(IJK)) THEN
                        WRITE(UNIT=VTK_UNIT) 41
                     ELSE
                        WRITE(UNIT=VTK_UNIT) 11
                     ENDIF
                  ENDIF
               ENDIF
            END DO
            WRITE(UNIT=VTK_UNIT) END_REC
         ENDIF
```

```
         WRITE(BUFFER,FMT='(A,I8)') 'CELL_DATA
',NUMBER_OF_VTK_CELLS
         WRITE(UNIT=VTK_UNIT)TRIM(BUFFER)//END_REC

      RETURN

      END SUBROUTINE WRITE_GEOMETRY_IN_VTK



!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvC
!
C
!  Module name: WRITE_SCALAR_IN_VTK
C
!  Purpose: Write Scalar variable in a vtk file
C
!
C
!  Author: Jeff Dietiker                                Date: 21-
Feb-08  C
!  Reviewer:                                            Date:
C
!
C
!  Revision Number #                                    Date: ##-
###-##  C
!  Author: #
C
!  Purpose: #
C
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
  SUBROUTINE WRITE_SCALAR_IN_VTK(VAR_NAME,VAR)

      USE param
      USE param1
      USE parallel
      USE constant
      USE run
      USE toleranc
      USE geometry
      USE indices
      USE compar
      USE mpi_utility
      USE sendrecv
```

```
      USE quadric
      USE cutcell
      USE fldvar
      USE vtk

      IMPLICIT NONE
      INTEGER :: I,IJK,L

      CHARACTER (*) :: VAR_NAME
      DOUBLE PRECISION, DIMENSION(DIMENSION_3) ::  VAR
      DOUBLE PRECISION, ALLOCATABLE :: GLOBAL_VAR(:)

      include "function.inc"


      IF (myPE == PE_IO) THEN
         allocate (GLOBAL_VAR(ijkmax3))
      ELSE
         allocate (GLOBAL_VAR(1))
      ENDIF

      call gather (VAR,GLOBAL_VAR,root)


      IF (myPE /= PE_IO) RETURN


      DO I = 1,LEN_TRIM(VAR_NAME)
         IF(VAR_NAME(I:I) == ' ') VAR_NAME(I:I) = '_'
      ENDDO


      WRITE(BUFFER,FMT='(A)')'SCALARS '//TRIM(VAR_NAME)//'
double 1'
      WRITE(UNIT=VTK_UNIT)TRIM(BUFFER)//END_REC
      WRITE(BUFFER,FMT='(A)')'LOOKUP_TABLE default'
      WRITE(UNIT=VTK_UNIT)TRIM(BUFFER)//END_REC

      DO IJK = 1,IJKMAX3
         IF (GLOBAL_INTERIOR_CELL_AT(IJK))      THEN
            IF (.NOT.GLOBAL_BLOCKED_CELL_AT(IJK))
WRITE(UNIT=VTK_UNIT) GLOBAL_VAR(IJK)
         ENDIF
      ENDDO
      WRITE(UNIT=VTK_UNIT)END_REC

      Deallocate (GLOBAL_VAR)

      RETURN
```

```
      END SUBROUTINE WRITE_SCALAR_IN_VTK

!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvC
!
C
!  Module name: WRITE_VECTOR_IN_VTK
C
!  Purpose: Write Vector variable in a vtk file
C
!
C
!  Author: Jeff Dietiker                              Date: 21-
Feb-08  C
!  Reviewer:                                          Date:
C
!
C
!  Revision Number #                                  Date: ##-
###-##  C
!  Author: #
C
!  Purpose: #
C
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
  SUBROUTINE WRITE_VECTOR_IN_VTK(VAR_NAME,VARX,VARY,VARZ)

      USE param
      USE param1
      USE parallel
      USE constant
      USE run
      USE toleranc
      USE geometry
      USE indices
      USE compar
      USE mpi_utility
      USE sendrecv
      USE quadric
      USE cutcell
      USE fldvar
      USE vtk

      IMPLICIT NONE
```

```
      INTEGER :: IJK,L

      CHARACTER (*) :: VAR_NAME
      DOUBLE PRECISION, DIMENSION(DIMENSION_3) ::
VARX,VARY,VARZ
      DOUBLE PRECISION, ALLOCATABLE ::
GLOBAL_VARX(:),GLOBAL_VARY(:),GLOBAL_VARZ(:)

      include "function.inc"


      IF (myPE == PE_IO) THEN
         allocate (GLOBAL_VARX(ijkmax3))
         allocate (GLOBAL_VARY(ijkmax3))
         allocate (GLOBAL_VARZ(ijkmax3))
      ELSE
         allocate (GLOBAL_VARX(1))
         allocate (GLOBAL_VARY(1))
         allocate (GLOBAL_VARZ(1))
      ENDIF

      call gather (VARX,GLOBAL_VARX,root)
      call gather (VARY,GLOBAL_VARY,root)
      call gather (VARZ,GLOBAL_VARZ,root)

      IF (myPE /= PE_IO) RETURN


      WRITE(BUFFER,FMT='(A)')'VECTORS '//TRIM(VAR_NAME)//'
double'
      WRITE(UNIT=VTK_UNIT)TRIM(BUFFER)//END_REC

      DO IJK = 1,IJKMAX3
         IF (GLOBAL_INTERIOR_CELL_AT(IJK))      THEN
            IF (.NOT.GLOBAL_BLOCKED_CELL_AT(IJK))
WRITE(UNIT=VTK_UNIT)
GLOBAL_VARX(IJK),GLOBAL_VARY(IJK),GLOBAL_VARZ(IJK)
         ENDIF
      ENDDO
      WRITE(UNIT=VTK_UNIT)END_REC


      Deallocate (GLOBAL_VARX)
      Deallocate (GLOBAL_VARY)
      Deallocate (GLOBAL_VARZ)

      RETURN

      END SUBROUTINE WRITE_VECTOR_IN_VTK
```

```
!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvC
!
C
!  Module name: CLOSE_VTK_FILE
C
!  Purpose: Close a vtk file
C
!
C
!  Author: Jeff Dietiker                              Date: 21-
Feb-08  C
!  Reviewer:                                          Date:
C
!
C
!  Revision Number #                                  Date: ##-
###-##  C
!  Author: #
C
!  Purpose: #
C
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
  SUBROUTINE CLOSE_VTK_FILE

      USE vtk

      IF (myPE /= PE_IO) RETURN

      CLOSE(VTK_UNIT)

      RETURN

      END SUBROUTINE CLOSE_VTK_FILE



!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvC
!
C
```

```
!  Module name: WRITE_CUT_SURFACE_VTK
C
!  Purpose: Writes the cut cell surface in VTK format
C
!
C
!  Author: Jeff Dietiker                              Date: 21-
Feb-08  C
!  Reviewer:                                          Date:
C
!
C
!  Revision Number #                                  Date: ##-
###-##  C
!  Author: #
C
!  Purpose: #
C
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
  SUBROUTINE WRITE_CUT_SURFACE_VTK

      USE param
      USE param1
      USE parallel
      USE constant
      USE run
      USE toleranc
      USE geometry
      USE indices
      USE compar
      USE sendrecv
      USE quadric
      USE cutcell
      USE fldvar
      USE vtk
      USE polygon
      USE stl

      IMPLICIT NONE

      INTEGER :: I,J,K,L,IM,JM,KM,IP,JP,KP,IJK,NODE
      INTEGER :: IMJK,IJMK,IJKM,IMJMK,IMJKM,IJMKM,IMJMKM
      INTEGER :: POINT_ID,POLY_COUNT,FACE_ID,Q_ID,Q_ID2
      INTEGER :: N_CUT_FACE_NODES,BCID2

      INTEGER NUMBER_OF_FACES
```

```
      INTEGER NUMBER_OF_SURFACE_POINTS

      DOUBLE PRECISION, DIMENSION(15,3) :: COORD_CUT_FACE_NODES
      DOUBLE PRECISION, DIMENSION(3)    :: NORMAL

      INTEGER, DIMENSION(DIMENSION_MAX_CUT_CELL,6) ::
FACE_CONNECTIVITY
      INTEGER, DIMENSION(DIMENSION_MAX_CUT_CELL)   ::
NUMBER_OF_CUT_FACE_POINTS

      DOUBLE PRECISION, DIMENSION(DIMENSION_MAX_CUT_CELL) ::
X_FACE_POINT
      DOUBLE PRECISION, DIMENSION(DIMENSION_MAX_CUT_CELL) ::
Y_FACE_POINT
      DOUBLE PRECISION, DIMENSION(DIMENSION_MAX_CUT_CELL) ::
Z_FACE_POINT

      DOUBLE PRECISION :: X_COPY,Y_COPY,Z_COPY,F_COPY,F2

      LOGICAL :: CLIP_FLAG,INTERSECT_FLAG,PRINT_FLAG

      CHARACTER (LEN=32) :: FILENAME

      include "function.inc"

      IF(myPE/=0) RETURN

!==================================================================
=======
!  Set-up connectivity for each cell, i.e., regular cells and
cut cells
!==================================================================
=======

      POLY_COUNT = 0

      NUMBER_OF_SURFACE_POINTS = 0

      NUMBER_OF_FACES = 0

      DO IJK = 1,IJKMAX3

         IF(GLOBAL_CUT_CELL_AT(IJK)) THEN

!==================================================================
=======
!  Filter the connectivity to identify nodes belonging to cut
face
```

```
!=================================================================
=======


            NUMBER_OF_FACES = NUMBER_OF_FACES + 1

            N_CUT_FACE_NODES = 0

            CALL GET_GLOBAL_CELL_NODE_COORDINATES(IJK,'SCALAR')

            DO L = 1, GLOBAL_NUMBER_OF_NODES(IJK)
               IF(GLOBAL_CONNECTIVITY(IJK,L)>IJKMAX3) THEN   !
One of the new point
                  X_COPY =
GLOBAL_X_NEW_POINT(GLOBAL_CONNECTIVITY(IJK,L)-IJKMAX3)
                  Y_COPY =
GLOBAL_Y_NEW_POINT(GLOBAL_CONNECTIVITY(IJK,L)-IJKMAX3)
                  Z_COPY =
GLOBAL_Z_NEW_POINT(GLOBAL_CONNECTIVITY(IJK,L)-IJKMAX3)
               ELSE                                    ! An
existing point
                  DO NODE = 1,8
                     IF(GLOBAL_CONNECTIVITY(IJK,L) ==
IJK_OF_NODE(NODE)) THEN
                        X_COPY = X_NODE(NODE)
                        Y_COPY = Y_NODE(NODE)
                        Z_COPY = Z_NODE(NODE)


                        IF (GLOBAL_SNAP(IJK_OF_NODE(NODE))) THEN
! One of the snapped corner point which now belongs to the cut
face
                           N_CUT_FACE_NODES = N_CUT_FACE_NODES +
1

COORD_CUT_FACE_NODES(N_CUT_FACE_NODES,1) = X_COPY

COORD_CUT_FACE_NODES(N_CUT_FACE_NODES,2) = Y_COPY

COORD_CUT_FACE_NODES(N_CUT_FACE_NODES,3) = Z_COPY
                        ENDIF
                     ENDIF
                  END DO

               ENDIF

               Q_ID = 1
```

```
               CALL
EVAL_F('QUADRIC',X_COPY,Y_COPY,Z_COPY,Q_ID,F_COPY,CLIP_FLAG)

               CALL
EVAL_F('POLYGON',X_COPY,Y_COPY,Z_COPY,N_POLYGON,F_COPY,CLIP_FLAG
)

               CALL
EVAL_F('USR_DEF',X_COPY,Y_COPY,Z_COPY,N_USR_DEF,F_COPY,CLIP_FLAG
)

               X_NODE(15) = X_COPY
               Y_NODE(15) = Y_COPY
               Z_NODE(15) = Z_COPY

               CALL
EVAL_STL_FCT_AT('SCALAR',IJK,15,F_COPY,CLIP_FLAG,BCID2)

               IF (ABS(F_COPY) < TOL_F ) THEN ! belongs to cut
face
                  N_CUT_FACE_NODES = N_CUT_FACE_NODES + 1
                  COORD_CUT_FACE_NODES(N_CUT_FACE_NODES,1) =
X_COPY
                  COORD_CUT_FACE_NODES(N_CUT_FACE_NODES,2) =
Y_COPY
                  COORD_CUT_FACE_NODES(N_CUT_FACE_NODES,3) =
Z_COPY
               ENDIF

            END DO

            CALL
REORDER_POLYGON(N_CUT_FACE_NODES,COORD_CUT_FACE_NODES,NORMAL)

            NUMBER_OF_CUT_FACE_POINTS(NUMBER_OF_FACES) =
N_CUT_FACE_NODES
            POLY_COUNT = POLY_COUNT + N_CUT_FACE_NODES + 1
            DO NODE = 1,N_CUT_FACE_NODES
               NUMBER_OF_SURFACE_POINTS =
NUMBER_OF_SURFACE_POINTS + 1


IF(NUMBER_OF_SURFACE_POINTS>=DIMENSION_MAX_CUT_CELL) THEN
                  WRITE(*,3000) 'ERROR IN SUBROUTINE
WRITE_3DCUT_SURFACE_VTK:'
                  WRITE(*,3000)
'NUMBER_OF_SURFACE_POINTS>=DIMENSION_MAX_CUT_CELL:'
                  WRITE(*,3000) 'INCREASE VALUE OF
FAC_DIM_MAX_CUT_CELL.'
```

```
                  WRITE(*,3010) 'CURRENT VALUE OF
FAC_DIM_MAX_CUT_CELL =',FAC_DIM_MAX_CUT_CELL
                  WRITE(*,3020) 'CURRENT VALUE OF
DIMENSION_MAX_CUT_CELL =',DIMENSION_MAX_CUT_CELL
                  WRITE(*,3000) 'MFiX will exit now.'
                  CALL MFIX_EXIT(myPE)
               ENDIF

               X_FACE_POINT(NUMBER_OF_SURFACE_POINTS) =
COORD_CUT_FACE_NODES(NODE,1)
               Y_FACE_POINT(NUMBER_OF_SURFACE_POINTS) =
COORD_CUT_FACE_NODES(NODE,2)
               Z_FACE_POINT(NUMBER_OF_SURFACE_POINTS) =
COORD_CUT_FACE_NODES(NODE,3)
               FACE_CONNECTIVITY(NUMBER_OF_FACES,NODE) =
NUMBER_OF_SURFACE_POINTS
            ENDDO

         ENDIF

      END DO



      FILENAME= TRIM(RUN_NAME) // '_boundary.vtk'
      FILENAME = TRIM(FILENAME)
      OPEN(UNIT = 123, FILE = FILENAME)
      WRITE(123,1001)'# vtk DataFile Version 2.0'
      WRITE(123,1001)'3D CUT-CELL SURFACE'
      WRITE(123,1001)'ASCII'

      IF(NO_K) THEN   ! 2D GEOMETRY
         WRITE(123,1001)'DATASET UNSTRUCTURED_GRID'
      ELSE            ! 3D GEOMETRY
         WRITE(123,1001)'DATASET POLYDATA'
      ENDIF

      WRITE(123,1010)'POINTS ',NUMBER_OF_SURFACE_POINTS,' float'

      DO POINT_ID = 1,NUMBER_OF_SURFACE_POINTS
         WRITE(123,1020)
X_FACE_POINT(POINT_ID),Y_FACE_POINT(POINT_ID),Z_FACE_POINT(POINT
_ID)
      ENDDO

      IF(NO_K) THEN   ! 2D GEOMETRY

         WRITE(123,1030)'CELLS ',NUMBER_OF_FACES,POLY_COUNT
         DO FACE_ID = 1 , NUMBER_OF_FACES
```

```
            WRITE(123,1040)
NUMBER_OF_CUT_FACE_POINTS(FACE_ID),(FACE_CONNECTIVITY(FACE_ID,L)
-1,&
            L=1,NUMBER_OF_CUT_FACE_POINTS(FACE_ID))
         ENDDO
         WRITE(123,1030)'CELL_TYPES ',NUMBER_OF_FACES
         DO FACE_ID = 1 , NUMBER_OF_FACES
            WRITE(123,1040) 3
         ENDDO

      ELSE            ! 3D GEOMETRY

         WRITE(123,1030)'POLYGONS ',NUMBER_OF_FACES,POLY_COUNT
         DO FACE_ID = 1 , NUMBER_OF_FACES
            WRITE(123,1040)
NUMBER_OF_CUT_FACE_POINTS(FACE_ID),(FACE_CONNECTIVITY(FACE_ID,L)
-1,&
            L=1,NUMBER_OF_CUT_FACE_POINTS(FACE_ID))
         ENDDO

      ENDIF

1001  FORMAT(A)
1010  FORMAT(A,I8,A)
1020  FORMAT(3(E16.8,2X))
1030  FORMAT(A,2(I8,2X))
1040  FORMAT(20(I8,2X))
1050  FORMAT(A,I8)
1060  FORMAT(E16.8)
1070  FORMAT(3(E16.8,2X))
1080  FORMAT(I5)
3000  FORMAT(1X,A)
3010  FORMAT(1X,A,F8.4)
3020  FORMAT(1X,A,I8)
3030  FORMAT(1X,A,A)
      CLOSE (123)


      WRITE(*,3030)'WROTE BOUNDARY IN VTK FILE : ',FILENAME
      RETURN


      END SUBROUTINE WRITE_CUT_SURFACE_VTK


!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvC
!
C
```

```
!  Module name: GATHER_DATA
C
!  Purpose: Gather data from all processes in preparation of
C
!           Writing VTK files
C
!
C
!  Author: Jeff Dietiker                             Date: 21-
Feb-08  C
!  Reviewer:                                         Date:
C
!
C
!  Revision Number #                                 Date: ##-
###-##  C
!  Author: #
C
!  Purpose: #
C
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
  SUBROUTINE GATHER_DATA

      USE param
      USE param1
      USE parallel
      USE constant
      USE run
      USE toleranc
      USE geometry
      USE indices
      USE compar
      USE mpi_utility
      USE sendrecv
      USE quadric
      USE cutcell
      USE fldvar
      USE vtk


      IMPLICIT NONE

      INTEGER :: IJK,I,J,K,L
      INTEGER :: IJK_OFFSET

      INTEGER :: iproc,IERR
```

```
      INTEGER, DIMENSION(0:numPEs-1) :: disp,rcount
      INTEGER, DIMENSION(:,:), ALLOCATABLE ::
SHIFTED_CONNECTIVITY

      include "function.inc"


!=================================================================
=======
!  parallel processing
!=================================================================
=======

      CALL allgather_1i (NUMBER_OF_NEW_POINTS,rcount,IERR)

      IF (myPE == 0) THEN
         IJK_OFFSET = 0
      ELSE
         IJK_OFFSET = 0
         DO iproc=0,myPE-1
            IJK_OFFSET = IJK_OFFSET + rcount(iproc)
         ENDDO
      ENDIF

      CALL allgather_1i (IJK_OFFSET,disp,IERR)

      IF(.NOT.GLOBAL_VAR_ALLOCATED) THEN

         IF (myPE == PE_IO) THEN
            allocate (GLOBAL_I_OF(ijkmax3))
            allocate (GLOBAL_J_OF(ijkmax3))
            allocate (GLOBAL_K_OF(ijkmax3))
            allocate (GLOBAL_CONNECTIVITY(ijkmax3,15))
            allocate (GLOBAL_NUMBER_OF_NODES(ijkmax3))
            allocate (GLOBAL_INTERIOR_CELL_AT(ijkmax3))
            allocate (GLOBAL_BLOCKED_CELL_AT(ijkmax3))
            allocate (GLOBAL_STANDARD_CELL_AT(ijkmax3))
            allocate (GLOBAL_CUT_CELL_AT(ijkmax3))
            allocate (GLOBAL_SNAP(ijkmax3))
            allocate (GLOBAL_X_NEW_POINT(ijkmax3))
            allocate (GLOBAL_Y_NEW_POINT(ijkmax3))
            allocate (GLOBAL_Z_NEW_POINT(ijkmax3))

         ELSE
            allocate (GLOBAL_I_OF(1))
            allocate (GLOBAL_J_OF(1))
            allocate (GLOBAL_K_OF(1))
            allocate (GLOBAL_CONNECTIVITY(1,1))
            allocate (GLOBAL_NUMBER_OF_NODES(1))
```

```
            allocate (GLOBAL_INTERIOR_CELL_AT(1))
            allocate (GLOBAL_BLOCKED_CELL_AT(1))
            allocate (GLOBAL_STANDARD_CELL_AT(1))
            allocate (GLOBAL_CUT_CELL_AT(1))
            allocate (GLOBAL_SNAP(1))
            allocate (GLOBAL_X_NEW_POINT(1))
            allocate (GLOBAL_Y_NEW_POINT(1))
            allocate (GLOBAL_Z_NEW_POINT(1))
         ENDIF

         GLOBAL_VAR_ALLOCATED = .TRUE.

      ENDIF

      call gatherv_1d( X_NEW_POINT, NUMBER_OF_NEW_POINTS,
GLOBAL_X_NEW_POINT, rcount, disp, PE_IO, ierr )
      call gatherv_1d( Y_NEW_POINT, NUMBER_OF_NEW_POINTS,
GLOBAL_Y_NEW_POINT, rcount, disp, PE_IO, ierr )
      call gatherv_1d( Z_NEW_POINT, NUMBER_OF_NEW_POINTS,
GLOBAL_Z_NEW_POINT, rcount, disp, PE_IO, ierr )

      call global_sum(NUMBER_OF_NEW_POINTS,
GLOBAL_NUMBER_OF_NEW_POINTS,  PE_IO, ierr )

      Allocate(  SHIFTED_CONNECTIVITY  (DIMENSION_3,15) )

      SHIFTED_CONNECTIVITY = CONNECTIVITY

      WHERE (SHIFTED_CONNECTIVITY > IJKEND3)
         SHIFTED_CONNECTIVITY = SHIFTED_CONNECTIVITY - IJKEND3 +
IJKMAX3 + disp(myPE)
      END WHERE

      DO IJK = IJKSTART3,IJKEND3
         DO L=1,NUMBER_OF_NODES(IJK)
            IF(CONNECTIVITY(IJK,L) <= IJKEND3) THEN
               I = I_OF(CONNECTIVITY(IJK,L))
               J = J_OF(CONNECTIVITY(IJK,L))
               K = K_OF(CONNECTIVITY(IJK,L))
               SHIFTED_CONNECTIVITY(IJK,L) = funijk_gl(I,J,K)
            ENDIF
         ENDDO
      ENDDO

      call gather (I_OF,GLOBAL_I_OF,root)
      call gather (J_OF,GLOBAL_J_OF,root)
      call gather (K_OF,GLOBAL_K_OF,root)
      call gather
(SHIFTED_CONNECTIVITY,GLOBAL_CONNECTIVITY,root)
```

```
      call gather (NUMBER_OF_NODES,GLOBAL_NUMBER_OF_NODES,root)
      call gather
(INTERIOR_CELL_AT,GLOBAL_INTERIOR_CELL_AT,root)
      call gather (BLOCKED_CELL_AT,GLOBAL_BLOCKED_CELL_AT,root)
      call gather
(STANDARD_CELL_AT,GLOBAL_STANDARD_CELL_AT,root)
      call gather (CUT_CELL_AT,GLOBAL_CUT_CELL_AT,root)
      call gather (SNAP,GLOBAL_SNAP,root)

      Deallocate(  SHIFTED_CONNECTIVITY )

      IF (myPE == PE_IO) THEN

         POLY_COUNTER = 0

         NUMBER_OF_CELLS = 0

         NUMBER_OF_CUT_CELLS = 0

         NUMBER_OF_BLOCKED_CELLS = 0

         NUMBER_OF_STANDARD_CELLS = 0

         DO IJK = 1, IJKMAX3

            IF (GLOBAL_INTERIOR_CELL_AT(IJK)) THEN

               NUMBER_OF_CELLS = NUMBER_OF_CELLS + 1

               IF (GLOBAL_BLOCKED_CELL_AT(IJK))
NUMBER_OF_BLOCKED_CELLS  = NUMBER_OF_BLOCKED_CELLS + 1
               IF (GLOBAL_STANDARD_CELL_AT(IJK))
NUMBER_OF_STANDARD_CELLS = NUMBER_OF_STANDARD_CELLS + 1
               IF (GLOBAL_CUT_CELL_AT(IJK))
NUMBER_OF_CUT_CELLS = NUMBER_OF_CUT_CELLS + 1

               IF (.NOT.GLOBAL_BLOCKED_CELL_AT(IJK))
POLY_COUNTER = POLY_COUNTER + GLOBAL_NUMBER_OF_NODES(IJK) + 1

            ENDIF

         END DO


         NUMBER_OF_POINTS = IJKMAX3 +
GLOBAL_NUMBER_OF_NEW_POINTS

      ENDIF
```

```
      RETURN

      END SUBROUTINE GATHER_DATA

!vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
vvvvvvvC
!
C
!  Module name: PRINT_GRID_STATISTICS
C
!  Purpose: PRINT_GRID_STATISTICS ON SCREEN
C
!
C
!
C
!  Author: Jeff Dietiker                                Date: 21-
Feb-08  C
!  Reviewer:                                            Date:
C
!
C
!  Revision Number #                                    Date: ##-
###-##  C
!  Author: #
C
!  Purpose: #
C
!
C
!^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^C
  SUBROUTINE PRINT_GRID_STATISTICS

      USE param
      USE param1
      USE parallel
      USE constant
      USE run
      USE toleranc
      USE geometry
      USE indices
      USE compar
      USE mpi_utility
      USE sendrecv
      USE quadric
      USE cutcell
```

```
      USE fldvar
      USE vtk


      IMPLICIT NONE

      INTEGER :: IJK,I,J,K,L
      INTEGER :: IJK_OFFSET

      INTEGER :: iproc,IERR

      DOUBLE PRECISION :: MIN_VOL, MAX_VOL,
GLOBAL_MIN_VOL,GLOBAL_MAX_VOL
      DOUBLE PRECISION :: MIN_AYZ, MAX_AYZ,
GLOBAL_MIN_AYZ,GLOBAL_MAX_AYZ
      DOUBLE PRECISION :: MIN_AXZ, MAX_AXZ,
GLOBAL_MIN_AXZ,GLOBAL_MAX_AXZ
      DOUBLE PRECISION :: MIN_AXY, MAX_AXY,
GLOBAL_MIN_AXY,GLOBAL_MAX_AXY
      DOUBLE PRECISION :: MIN_CUT, MAX_CUT,
GLOBAL_MIN_CUT,GLOBAL_MAX_CUT
      DOUBLE PRECISION :: LOCAL_MIN_Q,LOCAL_MAX_Q,
GLOBAL_MIN_Q,GLOBAL_MAX_Q


      include "function.inc"

      IF (myPE == PE_IO) THEN

         IF(.NOT.GRID_INFO_PRINTED_ON_SCREEN) THEN
            WRITE(*,5) 'GRID STATISTICS:'
            WRITE(*,5) 'NUMBER OF CELLS          = ',
NUMBER_OF_CELLS
            WRITE(*,10)'NUMBER OF STANDARD CELLS = ', &

NUMBER_OF_STANDARD_CELLS,DFLOAT(NUMBER_OF_STANDARD_CELLS) /
DFLOAT(NUMBER_OF_CELLS) * 100.0D0
            WRITE(*,10)'NUMBER OF CUT CELLS      = ', &

NUMBER_OF_CUT_CELLS,DFLOAT(NUMBER_OF_CUT_CELLS) /
DFLOAT(NUMBER_OF_CELLS) * 100.0D0
            WRITE(*,10)'NUMBER OF BLOCKED CELLS  = ', &

NUMBER_OF_BLOCKED_CELLS,DFLOAT(NUMBER_OF_BLOCKED_CELLS) /
DFLOAT(NUMBER_OF_CELLS) * 100.0D0

5           FORMAT(1X,A,I8)
10          FORMAT(1X,A,I8,' (',F6.2,' % of Total)')
```

```
         ENDIF

         GRID_INFO_PRINTED_ON_SCREEN = .TRUE.

      ENDIF


!=================================================================
=======
!  Scalar Cell volumes and areas
!=================================================================
=======

      MIN_VOL =   LARGE_NUMBER
      MAX_VOL = - LARGE_NUMBER
      MIN_AYZ =   LARGE_NUMBER
      MAX_AYZ = - LARGE_NUMBER
      MIN_AXZ =   LARGE_NUMBER
      MAX_AXZ = - LARGE_NUMBER
      MIN_AXY =   LARGE_NUMBER
      MAX_AXY = - LARGE_NUMBER

      DO IJK = IJKSTART3, IJKEND3
         IF(STANDARD_CELL_AT(IJK)) THEN              ! STANDARD
CELLS

            MIN_VOL =   DMIN1(MIN_VOL,VOL(IJK))
            MAX_VOL =   DMAX1(MAX_VOL,VOL(IJK))
            MIN_AYZ =   DMIN1(MIN_AYZ,AYZ(IJK))
            MAX_AYZ =   DMAX1(MAX_AYZ,AYZ(IJK))
            MIN_AXZ =   DMIN1(MIN_AXZ,AXZ(IJK))
            MAX_AXZ =   DMAX1(MAX_AXZ,AXZ(IJK))
            MIN_AXY =   DMIN1(MIN_AXY,AXY(IJK))
            MAX_AXY =   DMAX1(MAX_AXY,AXY(IJK))

         ENDIF
      END DO

      call global_min(MIN_VOL, GLOBAL_MIN_VOL,  PE_IO, ierr )
      call global_max(MAX_VOL, GLOBAL_MAX_VOL,  PE_IO, ierr )
      call global_min(MIN_AYZ, GLOBAL_MIN_AYZ,  PE_IO, ierr )
      call global_max(MAX_AYZ, GLOBAL_MAX_AYZ,  PE_IO, ierr )
      call global_min(MIN_AXZ, GLOBAL_MIN_AXZ,  PE_IO, ierr )
      call global_max(MAX_AXZ, GLOBAL_MAX_AXZ,  PE_IO, ierr )
      call global_min(MIN_AXY, GLOBAL_MIN_AXY,  PE_IO, ierr )
      call global_max(MAX_AXY, GLOBAL_MAX_AXY,  PE_IO, ierr )

      IF (myPE == PE_IO) THEN
```

```
         WRITE(UNIT_CUT_CELL_LOG,1000)
'##############################################################
#'
         WRITE(UNIT_CUT_CELL_LOG,1000)  '
CELLS STATISTICS                         '
         WRITE(UNIT_CUT_CELL_LOG,1000)
'##############################################################
#'
         WRITE(UNIT_CUT_CELL_LOG,1000)  'SCALAR STANDARD CELLS:'
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXY
= ', GLOBAL_MIN_AXY,GLOBAL_MAX_AXY
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXZ
= ', GLOBAL_MIN_AXZ,GLOBAL_MAX_AXZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AYZ
= ', GLOBAL_MIN_AYZ,GLOBAL_MAX_AYZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF VOLUME
= ', GLOBAL_MIN_VOL,GLOBAL_MAX_VOL
      ENDIF


      MIN_VOL =   LARGE_NUMBER
      MAX_VOL = - LARGE_NUMBER
      MIN_AYZ =   LARGE_NUMBER
      MAX_AYZ = - LARGE_NUMBER
      MIN_AXZ =   LARGE_NUMBER
      MAX_AXZ = - LARGE_NUMBER
      MIN_AXY =   LARGE_NUMBER
      MAX_AXY = - LARGE_NUMBER

      DO IJK = IJKSTART3, IJKEND3
         IF(CUT_CELL_AT(IJK)) THEN                  ! CUT CELLS

            MIN_VOL =   DMIN1(MIN_VOL,VOL(IJK))
            MAX_VOL =   DMAX1(MAX_VOL,VOL(IJK))
            MIN_AYZ =   DMIN1(MIN_AYZ,AYZ(IJK))
            MAX_AYZ =   DMAX1(MAX_AYZ,AYZ(IJK))
            MIN_AXZ =   DMIN1(MIN_AXZ,AXZ(IJK))
            MAX_AXZ =   DMAX1(MAX_AXZ,AXZ(IJK))
            MIN_AXY =   DMIN1(MIN_AXY,AXY(IJK))
            MAX_AXY =   DMAX1(MAX_AXY,AXY(IJK))

         ENDIF
      END DO

      call global_min(MIN_VOL, GLOBAL_MIN_VOL,  PE_IO, ierr )
      call global_max(MAX_VOL, GLOBAL_MAX_VOL,  PE_IO, ierr )
      call global_min(MIN_AYZ, GLOBAL_MIN_AYZ,  PE_IO, ierr )
      call global_max(MAX_AYZ, GLOBAL_MAX_AYZ,  PE_IO, ierr )
      call global_min(MIN_AXZ, GLOBAL_MIN_AXZ,  PE_IO, ierr )
```

```
      call global_max(MAX_AXZ, GLOBAL_MAX_AXZ,  PE_IO, ierr )
      call global_min(MIN_AXY, GLOBAL_MIN_AXY,  PE_IO, ierr )
      call global_max(MAX_AXY, GLOBAL_MAX_AXY,  PE_IO, ierr )

      IF (myPE == PE_IO) THEN
         WRITE(UNIT_CUT_CELL_LOG,1000)  'SCALAR CUT CELLS:'
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXY
= ', GLOBAL_MIN_AXY,GLOBAL_MAX_AXY
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXZ
= ', GLOBAL_MIN_AXZ,GLOBAL_MAX_AXZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AYZ
= ', GLOBAL_MIN_AYZ,GLOBAL_MAX_AYZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF VOLUME
= ', GLOBAL_MIN_VOL,GLOBAL_MAX_VOL
         WRITE(UNIT_CUT_CELL_LOG,1010)  'NUMBER OF SMALL SCALAR
CELLS   = ', NUMBER_OF_SMALL_CELLS
         WRITE(UNIT_CUT_CELL_LOG,1000)
'##############################################################
#'
      ENDIF


1000 FORMAT(A,E14.8,2X,E14.8)
1010 FORMAT(A,I8)

!==============================================================
=======
!  U-Momentum Cell volumes and areas
!==============================================================
=======



      MIN_VOL =   LARGE_NUMBER
      MAX_VOL = - LARGE_NUMBER
      MIN_AYZ =   LARGE_NUMBER
      MAX_AYZ = - LARGE_NUMBER
      MIN_AXZ =   LARGE_NUMBER
      MAX_AXZ = - LARGE_NUMBER
      MIN_AXY =   LARGE_NUMBER
      MAX_AXY = - LARGE_NUMBER

      DO IJK = IJKSTART3, IJKEND3
         IF(STANDARD_U_CELL_AT(IJK)) THEN              !
STANDARD CELLS

            MIN_VOL =   DMIN1(MIN_VOL,VOL_U(IJK))
            MAX_VOL =   DMAX1(MAX_VOL,VOL_U(IJK))
            MIN_AYZ =   DMIN1(MIN_AYZ,AYZ_U(IJK))
```

```
            MAX_AYZ =   DMAX1(MAX_AYZ,AYZ_U(IJK))
            MIN_AXZ =   DMIN1(MIN_AXZ,AXZ_U(IJK))
            MAX_AXZ =   DMAX1(MAX_AXZ,AXZ_U(IJK))
            MIN_AXY =   DMIN1(MIN_AXY,AXY_U(IJK))
            MAX_AXY =   DMAX1(MAX_AXY,AXY_U(IJK))

         ENDIF
      END DO

      call global_min(MIN_VOL, GLOBAL_MIN_VOL,  PE_IO, ierr )
      call global_max(MAX_VOL, GLOBAL_MAX_VOL,  PE_IO, ierr )
      call global_min(MIN_AYZ, GLOBAL_MIN_AYZ,  PE_IO, ierr )
      call global_max(MAX_AYZ, GLOBAL_MAX_AYZ,  PE_IO, ierr )
      call global_min(MIN_AXZ, GLOBAL_MIN_AXZ,  PE_IO, ierr )
      call global_max(MAX_AXZ, GLOBAL_MAX_AXZ,  PE_IO, ierr )
      call global_min(MIN_AXY, GLOBAL_MIN_AXY,  PE_IO, ierr )
      call global_max(MAX_AXY, GLOBAL_MAX_AXY,  PE_IO, ierr )

      IF (myPE == PE_IO) THEN
         WRITE(UNIT_CUT_CELL_LOG,1000)  'U-MOMENTUM STANDARD
CELLS:'
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXY
= ', GLOBAL_MIN_AXY,GLOBAL_MAX_AXY
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXZ
= ', GLOBAL_MIN_AXZ,GLOBAL_MAX_AXZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AYZ
= ', GLOBAL_MIN_AYZ,GLOBAL_MAX_AYZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF VOLUME
= ', GLOBAL_MIN_VOL,GLOBAL_MAX_VOL
      ENDIF

      MIN_VOL =   LARGE_NUMBER
      MAX_VOL = - LARGE_NUMBER
      MIN_AYZ =   LARGE_NUMBER
      MAX_AYZ = - LARGE_NUMBER
      MIN_AXZ =   LARGE_NUMBER
      MAX_AXZ = - LARGE_NUMBER
      MIN_AXY =   LARGE_NUMBER
      MAX_AXY = - LARGE_NUMBER
      MIN_CUT =   LARGE_NUMBER
      MAX_CUT = - LARGE_NUMBER

      DO IJK = IJKSTART3, IJKEND3
         IF(CUT_U_CELL_AT(IJK).AND.(.NOT.WALL_U_AT(IJK))) THEN
! CUT CELLS

            MIN_VOL =   DMIN1(MIN_VOL,VOL_U(IJK))
            MAX_VOL =   DMAX1(MAX_VOL,VOL_U(IJK))
            MIN_AYZ =   DMIN1(MIN_AYZ,AYZ_U(IJK))
```

```
            MAX_AYZ =   DMAX1(MAX_AYZ,AYZ_U(IJK))
            MIN_AXZ =   DMIN1(MIN_AXZ,AXZ_U(IJK))
            MAX_AXZ =   DMAX1(MAX_AXZ,AXZ_U(IJK))
            MIN_AXY =   DMIN1(MIN_AXY,AXY_U(IJK))
            MAX_AXY =   DMAX1(MAX_AXY,AXY_U(IJK))
            MIN_CUT =   DMIN1(MIN_CUT,AREA_U_CUT(IJK))
            MAX_CUT =   DMAX1(MAX_CUT,AREA_U_CUT(IJK))

         ENDIF
      END DO

      call global_min(MIN_VOL, GLOBAL_MIN_VOL,  PE_IO, ierr )
      call global_max(MAX_VOL, GLOBAL_MAX_VOL,  PE_IO, ierr )
      call global_min(MIN_AYZ, GLOBAL_MIN_AYZ,  PE_IO, ierr )
      call global_max(MAX_AYZ, GLOBAL_MAX_AYZ,  PE_IO, ierr )
      call global_min(MIN_AXZ, GLOBAL_MIN_AXZ,  PE_IO, ierr )
      call global_max(MAX_AXZ, GLOBAL_MAX_AXZ,  PE_IO, ierr )
      call global_min(MIN_AXY, GLOBAL_MIN_AXY,  PE_IO, ierr )
      call global_max(MAX_AXY, GLOBAL_MAX_AXY,  PE_IO, ierr )
      call global_min(MIN_CUT, GLOBAL_MIN_CUT,  PE_IO, ierr )
      call global_max(MAX_CUT, GLOBAL_MAX_CUT,  PE_IO, ierr )

      IF (myPE == PE_IO) THEN
         WRITE(UNIT_CUT_CELL_LOG,1000)  'U-MOMENTUM CUT CELLS:'
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXY
= ', GLOBAL_MIN_AXY,GLOBAL_MAX_AXY
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXZ
= ', GLOBAL_MIN_AXZ,GLOBAL_MAX_AXZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AYZ
= ', GLOBAL_MIN_AYZ,GLOBAL_MAX_AYZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF CUT AREA
= ', GLOBAL_MIN_CUT,GLOBAL_MAX_CUT
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF VOLUME
= ', GLOBAL_MIN_VOL,GLOBAL_MAX_VOL
         WRITE(UNIT_CUT_CELL_LOG,1010)  'NUMBER OF U WALL CELLS
= ', NUMBER_OF_U_WALL_CELLS
         WRITE(UNIT_CUT_CELL_LOG,1000)
'##############################################################
#'
      ENDIF
!==============================================================
=======
!  V-Momentum Cell volumes and areas
!==============================================================
=======


      MIN_VOL =   LARGE_NUMBER
      MAX_VOL = - LARGE_NUMBER
```

```
      MIN_AYZ =   LARGE_NUMBER
      MAX_AYZ = - LARGE_NUMBER
      MIN_AXZ =   LARGE_NUMBER
      MAX_AXZ = - LARGE_NUMBER
      MIN_AXY =   LARGE_NUMBER
      MAX_AXY = - LARGE_NUMBER

      DO IJK = IJKSTART3, IJKEND3
         IF(STANDARD_V_CELL_AT(IJK)) THEN              !
STANDARD CELLS

            MIN_VOL =   DMIN1(MIN_VOL,VOL_V(IJK))
            MAX_VOL =   DMAX1(MAX_VOL,VOL_V(IJK))
            MIN_AYZ =   DMIN1(MIN_AYZ,AYZ_V(IJK))
            MAX_AYZ =   DMAX1(MAX_AYZ,AYZ_V(IJK))
            MIN_AXZ =   DMIN1(MIN_AXZ,AXZ_V(IJK))
            MAX_AXZ =   DMAX1(MAX_AXZ,AXZ_V(IJK))
            MIN_AXY =   DMIN1(MIN_AXY,AXY_V(IJK))
            MAX_AXY =   DMAX1(MAX_AXY,AXY_V(IJK))

         ENDIF
      END DO

      call global_min(MIN_VOL, GLOBAL_MIN_VOL,  PE_IO, ierr )
      call global_max(MAX_VOL, GLOBAL_MAX_VOL,  PE_IO, ierr )
      call global_min(MIN_AYZ, GLOBAL_MIN_AYZ,  PE_IO, ierr )
      call global_max(MAX_AYZ, GLOBAL_MAX_AYZ,  PE_IO, ierr )
      call global_min(MIN_AXZ, GLOBAL_MIN_AXZ,  PE_IO, ierr )
      call global_max(MAX_AXZ, GLOBAL_MAX_AXZ,  PE_IO, ierr )
      call global_min(MIN_AXY, GLOBAL_MIN_AXY,  PE_IO, ierr )
      call global_max(MAX_AXY, GLOBAL_MAX_AXY,  PE_IO, ierr )

      IF (myPE == PE_IO) THEN
         WRITE(UNIT_CUT_CELL_LOG,1000)  'V-MOMENTUM STANDARD
CELLS:'
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXY
= ', GLOBAL_MIN_AXY,GLOBAL_MAX_AXY
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXZ
= ', GLOBAL_MIN_AXZ,GLOBAL_MAX_AXZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AYZ
= ', GLOBAL_MIN_AYZ,GLOBAL_MAX_AYZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF VOLUME
= ', GLOBAL_MIN_VOL,GLOBAL_MAX_VOL
      ENDIF

      MIN_VOL =   LARGE_NUMBER
      MAX_VOL = - LARGE_NUMBER
      MIN_AYZ =   LARGE_NUMBER
      MAX_AYZ = - LARGE_NUMBER
```

```
      MIN_AXZ =   LARGE_NUMBER
      MAX_AXZ = - LARGE_NUMBER
      MIN_AXY =   LARGE_NUMBER
      MAX_AXY = - LARGE_NUMBER
      MIN_CUT =   LARGE_NUMBER
      MAX_CUT = - LARGE_NUMBER

      DO IJK = IJKSTART3, IJKEND3
         IF(CUT_V_CELL_AT(IJK).AND.(.NOT.WALL_V_AT(IJK))) THEN
! CUT CELLS

            MIN_VOL =   DMIN1(MIN_VOL,VOL_V(IJK))
            MAX_VOL =   DMAX1(MAX_VOL,VOL_V(IJK))
            MIN_AYZ =   DMIN1(MIN_AYZ,AYZ_V(IJK))
            MAX_AYZ =   DMAX1(MAX_AYZ,AYZ_V(IJK))
            MIN_AXZ =   DMIN1(MIN_AXZ,AXZ_V(IJK))
            MAX_AXZ =   DMAX1(MAX_AXZ,AXZ_V(IJK))
            MIN_AXY =   DMIN1(MIN_AXY,AXY_V(IJK))
            MAX_AXY =   DMAX1(MAX_AXY,AXY_V(IJK))
            MIN_CUT =   DMIN1(MIN_CUT,AREA_V_CUT(IJK))
            MAX_CUT =   DMAX1(MAX_CUT,AREA_V_CUT(IJK))

         ENDIF
      END DO

      call global_min(MIN_VOL, GLOBAL_MIN_VOL,  PE_IO, ierr )
      call global_max(MAX_VOL, GLOBAL_MAX_VOL,  PE_IO, ierr )
      call global_min(MIN_AYZ, GLOBAL_MIN_AYZ,  PE_IO, ierr )
      call global_max(MAX_AYZ, GLOBAL_MAX_AYZ,  PE_IO, ierr )
      call global_min(MIN_AXZ, GLOBAL_MIN_AXZ,  PE_IO, ierr )
      call global_max(MAX_AXZ, GLOBAL_MAX_AXZ,  PE_IO, ierr )
      call global_min(MIN_AXY, GLOBAL_MIN_AXY,  PE_IO, ierr )
      call global_max(MAX_AXY, GLOBAL_MAX_AXY,  PE_IO, ierr )
      call global_min(MIN_CUT, GLOBAL_MIN_CUT,  PE_IO, ierr )
      call global_max(MAX_CUT, GLOBAL_MAX_CUT,  PE_IO, ierr )

      IF (myPE == PE_IO) THEN
         WRITE(UNIT_CUT_CELL_LOG,1000)  'V-MOMENTUM CUT CELLS:'
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXY
= ', GLOBAL_MIN_AXY,GLOBAL_MAX_AXY
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXZ
= ', GLOBAL_MIN_AXZ,GLOBAL_MAX_AXZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AYZ
= ', GLOBAL_MIN_AYZ,GLOBAL_MAX_AYZ
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF CUT AREA
= ', GLOBAL_MIN_CUT,GLOBAL_MAX_CUT
         WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF VOLUME
= ', GLOBAL_MIN_VOL,GLOBAL_MAX_VOL
```

```
         WRITE(UNIT_CUT_CELL_LOG,1010)  'NUMBER OF V WALL CELLS
= ', NUMBER_OF_V_WALL_CELLS
         WRITE(UNIT_CUT_CELL_LOG,1000)
'##############################################################
#'
      ENDIF

!==============================================================
=======
!  W-Momentum Cell volumes and areas
!==============================================================
=======


      IF(DO_K) THEN

         MIN_VOL =   LARGE_NUMBER
         MAX_VOL = - LARGE_NUMBER
         MIN_AYZ =   LARGE_NUMBER
         MAX_AYZ = - LARGE_NUMBER
         MIN_AXZ =   LARGE_NUMBER
         MAX_AXZ = - LARGE_NUMBER
         MIN_AXY =   LARGE_NUMBER
         MAX_AXY = - LARGE_NUMBER

         DO IJK = IJKSTART3, IJKEND3
            IF(STANDARD_W_CELL_AT(IJK)) THEN              !
STANDARD CELLS

               MIN_VOL =   DMIN1(MIN_VOL,VOL_W(IJK))
               MAX_VOL =   DMAX1(MAX_VOL,VOL_W(IJK))
               MIN_AYZ =   DMIN1(MIN_AYZ,AYZ_W(IJK))
               MAX_AYZ =   DMAX1(MAX_AYZ,AYZ_W(IJK))
               MIN_AXZ =   DMIN1(MIN_AXZ,AXZ_W(IJK))
               MAX_AXZ =   DMAX1(MAX_AXZ,AXZ_W(IJK))
               MIN_AXY =   DMIN1(MIN_AXY,AXY_W(IJK))
               MAX_AXY =   DMAX1(MAX_AXY,AXY_W(IJK))

            ENDIF
         END DO

         call global_min(MIN_VOL, GLOBAL_MIN_VOL,  PE_IO, ierr )
         call global_max(MAX_VOL, GLOBAL_MAX_VOL,  PE_IO, ierr )
         call global_min(MIN_AYZ, GLOBAL_MIN_AYZ,  PE_IO, ierr )
         call global_max(MAX_AYZ, GLOBAL_MAX_AYZ,  PE_IO, ierr )
         call global_min(MIN_AXZ, GLOBAL_MIN_AXZ,  PE_IO, ierr )
         call global_max(MAX_AXZ, GLOBAL_MAX_AXZ,  PE_IO, ierr )
         call global_min(MIN_AXY, GLOBAL_MIN_AXY,  PE_IO, ierr )
         call global_max(MAX_AXY, GLOBAL_MAX_AXY,  PE_IO, ierr )
```

```
         IF (myPE == PE_IO) THEN
            WRITE(UNIT_CUT_CELL_LOG,1000)  'W-MOMENTUM STANDARD
CELLS:'
            WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXY
= ', GLOBAL_MIN_AXY,GLOBAL_MAX_AXY
            WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXZ
= ', GLOBAL_MIN_AXZ,GLOBAL_MAX_AXZ
            WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AYZ
= ', GLOBAL_MIN_AYZ,GLOBAL_MAX_AYZ
            WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF VOLUME
= ', GLOBAL_MIN_VOL,GLOBAL_MAX_VOL
         ENDIF

         MIN_VOL =   LARGE_NUMBER
         MAX_VOL = - LARGE_NUMBER
         MIN_AYZ =   LARGE_NUMBER
         MAX_AYZ = - LARGE_NUMBER
         MIN_AXZ =   LARGE_NUMBER
         MAX_AXZ = - LARGE_NUMBER
         MIN_AXY =   LARGE_NUMBER
         MAX_AXY = - LARGE_NUMBER
         MIN_CUT =   LARGE_NUMBER
         MAX_CUT = - LARGE_NUMBER

         DO IJK = IJKSTART3, IJKEND3
            IF(CUT_W_CELL_AT(IJK).AND.(.NOT.WALL_W_AT(IJK)))
THEN      ! CUT CELLS

               MIN_VOL =   DMIN1(MIN_VOL,VOL_W(IJK))
               MAX_VOL =   DMAX1(MAX_VOL,VOL_W(IJK))
               MIN_AYZ =   DMIN1(MIN_AYZ,AYZ_W(IJK))
               MAX_AYZ =   DMAX1(MAX_AYZ,AYZ_W(IJK))
               MIN_AXZ =   DMIN1(MIN_AXZ,AXZ_W(IJK))
               MAX_AXZ =   DMAX1(MAX_AXZ,AXZ_W(IJK))
               MIN_AXY =   DMIN1(MIN_AXY,AXY_W(IJK))
               MAX_AXY =   DMAX1(MAX_AXY,AXY_W(IJK))
               MIN_CUT =   DMIN1(MIN_CUT,AREA_W_CUT(IJK))
               MAX_CUT =   DMAX1(MAX_CUT,AREA_W_CUT(IJK))

            ENDIF
         END DO

         call global_min(MIN_VOL, GLOBAL_MIN_VOL,  PE_IO, ierr )
         call global_max(MAX_VOL, GLOBAL_MAX_VOL,  PE_IO, ierr )
         call global_min(MIN_AYZ, GLOBAL_MIN_AYZ,  PE_IO, ierr )
         call global_max(MAX_AYZ, GLOBAL_MAX_AYZ,  PE_IO, ierr )
         call global_min(MIN_AXZ, GLOBAL_MIN_AXZ,  PE_IO, ierr )
         call global_max(MAX_AXZ, GLOBAL_MAX_AXZ,  PE_IO, ierr )
```

```
         call global_min(MIN_AXY, GLOBAL_MIN_AXY,  PE_IO, ierr )
         call global_max(MAX_AXY, GLOBAL_MAX_AXY,  PE_IO, ierr )
         call global_min(MIN_CUT, GLOBAL_MIN_CUT,  PE_IO, ierr )
         call global_max(MAX_CUT, GLOBAL_MAX_CUT,  PE_IO, ierr )

         IF (myPE == PE_IO) THEN
            WRITE(UNIT_CUT_CELL_LOG,1000)  'W-MOMENTUM CUT
CELLS:'
            WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXY
= ', GLOBAL_MIN_AXY,GLOBAL_MAX_AXY
            WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AXZ
= ', GLOBAL_MIN_AXZ,GLOBAL_MAX_AXZ
            WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF AYZ
= ', GLOBAL_MIN_AYZ,GLOBAL_MAX_AYZ
            WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF CUT AREA
= ', GLOBAL_MIN_CUT,GLOBAL_MAX_CUT
            WRITE(UNIT_CUT_CELL_LOG,1000)  'RANGE OF VOLUME
= ', GLOBAL_MIN_VOL,GLOBAL_MAX_VOL
            WRITE(UNIT_CUT_CELL_LOG,1010)  'NUMBER OF W WALL
CELLS         = ', NUMBER_OF_W_WALL_CELLS
            WRITE(UNIT_CUT_CELL_LOG,1000)
'##############################################################
#'
         ENDIF

      ENDIF



      LOCAL_MIN_Q = MINVAL(Alpha_Ue_c)
      LOCAL_MAX_Q = MAXVAL(Alpha_Ue_c)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO)  WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM Alpha_Ue_c = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q

      LOCAL_MIN_Q = MINVAL(Alpha_Un_c)
      LOCAL_MAX_Q = MAXVAL(Alpha_Un_c)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM Alpha_Un_c = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q

      LOCAL_MIN_Q = MINVAL(Alpha_Ut_c)
      LOCAL_MAX_Q = MAXVAL(Alpha_Ut_c)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM Alpha_Ut_c = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
```

```
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)

      LOCAL_MIN_Q = MINVAL(Theta_Ue)
      LOCAL_MAX_Q = MAXVAL(Theta_Ue)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM Theta_Ue   = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q

      LOCAL_MIN_Q = MINVAL(Theta_Un)
      LOCAL_MAX_Q = MAXVAL(Theta_Un)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM Theta_Un   = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q

      LOCAL_MIN_Q = MINVAL(Theta_Ut)
      LOCAL_MAX_Q = MAXVAL(Theta_Ut)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM Theta_Ut   = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)

      LOCAL_MIN_Q = MINVAL(Theta_U_ne)
      LOCAL_MAX_Q = MAXVAL(Theta_U_ne)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM Theta_U_ne = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q

      LOCAL_MIN_Q = MINVAL(Theta_U_te)
      LOCAL_MAX_Q = MAXVAL(Theta_U_te)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM Theta_U_te = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)

      LOCAL_MIN_Q = MINVAL(NOC_U_E)
      LOCAL_MAX_Q = MAXVAL(NOC_U_E)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM NOC_U_E    = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q

      LOCAL_MIN_Q = MINVAL(NOC_U_N)
      LOCAL_MAX_Q = MAXVAL(NOC_U_N)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
```

```
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM NOC_U_N    = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q

      LOCAL_MIN_Q = MINVAL(NOC_U_T)
      LOCAL_MAX_Q = MAXVAL(NOC_U_T)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM NOC_U_T    = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)

      LOCAL_MIN_Q = MINVAL(DELH_U)
      LOCAL_MAX_Q = MAXVAL(DELH_U)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF U-MOMENTUM DELH_U     = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)
'##############################################################
#'



      LOCAL_MIN_Q = MINVAL(Alpha_Ve_c)
      LOCAL_MAX_Q = MAXVAL(Alpha_Ve_c)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM Alpha_Ve_c = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      LOCAL_MIN_Q = MINVAL(Alpha_Vn_c)
      LOCAL_MAX_Q = MAXVAL(Alpha_Vn_c)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM Alpha_Vn_c = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      LOCAL_MIN_Q = MINVAL(Alpha_Vt_c)
      LOCAL_MAX_Q = MAXVAL(Alpha_Vt_c)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM Alpha_Vt_c = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)
      LOCAL_MIN_Q = MINVAL(Theta_Ve)
      LOCAL_MAX_Q = MAXVAL(Theta_Ve)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM Theta_Ve   = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
```

```
      LOCAL_MIN_Q = MINVAL(Theta_Vn)
      LOCAL_MAX_Q = MAXVAL(Theta_Vn)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM Theta_Vn   = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      LOCAL_MIN_Q = MINVAL(Theta_Vt)
      LOCAL_MAX_Q = MAXVAL(Theta_Vt)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM Theta_Vt   = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)
      LOCAL_MIN_Q = MINVAL(Theta_V_ne)
      LOCAL_MAX_Q = MAXVAL(Theta_V_ne)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM Theta_V_ne = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      LOCAL_MIN_Q = MINVAL(Theta_V_nt)
      LOCAL_MAX_Q = MAXVAL(Theta_V_nt)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM Theta_V_nt = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)
      LOCAL_MIN_Q = MINVAL(NOC_V_E)
      LOCAL_MAX_Q = MAXVAL(NOC_V_E)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM NOC_V_E    = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      LOCAL_MIN_Q = MINVAL(NOC_V_N)
      LOCAL_MAX_Q = MAXVAL(NOC_V_N)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM NOC_V_N    = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      LOCAL_MIN_Q = MINVAL(NOC_V_T)
      LOCAL_MAX_Q = MAXVAL(NOC_V_T)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM NOC_V_T    = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)
      LOCAL_MIN_Q = MINVAL(DELH_V)
      LOCAL_MAX_Q = MAXVAL(DELH_V)
      call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr )
      call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr )
```

```
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  ' RANGE
OF V-MOMENTUM DELH_V     = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
      IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)
'##########################################################
#'


      IF(DO_K) THEN

         LOCAL_MIN_Q = MINVAL(Alpha_We_c)
         LOCAL_MAX_Q = MAXVAL(Alpha_We_c)
         call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
         call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM Alpha_We_c = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         LOCAL_MIN_Q = MINVAL(Alpha_Wn_c)
         LOCAL_MAX_Q = MAXVAL(Alpha_Wn_c)
         call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
         call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM Alpha_Wn_c = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         LOCAL_MIN_Q = MINVAL(Alpha_Wt_c)
         LOCAL_MAX_Q = MAXVAL(Alpha_Wt_c)
         call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
         call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM Alpha_Wt_c = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)
         LOCAL_MIN_Q = MINVAL(Theta_We)
         LOCAL_MAX_Q = MAXVAL(Theta_We)
         call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
         call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM Theta_We   = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         LOCAL_MIN_Q = MINVAL(Theta_Wn)
         LOCAL_MAX_Q = MAXVAL(Theta_Wn)
         call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
         call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
```

```
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM Theta_Wn   = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         LOCAL_MIN_Q = MINVAL(Theta_Wt)
         LOCAL_MAX_Q = MAXVAL(Theta_Wt)
         call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
         call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM Theta_Wt   = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)
         LOCAL_MIN_Q = MINVAL(Theta_W_te)
         LOCAL_MAX_Q = MAXVAL(Theta_W_te)
         call global_min(LOCAL_MAX_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
         call global_max(LOCAL_MIN_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM Theta_W_te = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         LOCAL_MIN_Q = MINVAL(Theta_W_tn)
         LOCAL_MAX_Q = MAXVAL(Theta_W_tn)
         call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
         call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM Theta_W_tn = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)
         LOCAL_MIN_Q = MINVAL(NOC_W_E)
         LOCAL_MAX_Q = MAXVAL(NOC_W_E)
         call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
         call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM NOC_W_E    = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         LOCAL_MIN_Q = MINVAL(NOC_W_N)
         LOCAL_MAX_Q = MAXVAL(NOC_W_N)
         call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
         call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM NOC_W_N    = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         LOCAL_MIN_Q = MINVAL(NOC_W_T)
         LOCAL_MAX_Q = MAXVAL(NOC_W_T)
         call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
```

```
         call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM NOC_W_T    = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)
         LOCAL_MIN_Q = MINVAL(DELH_W)
         LOCAL_MAX_Q = MAXVAL(DELH_W)
         call global_min(LOCAL_MIN_Q, GLOBAL_MIN_Q,  PE_IO, ierr
)
         call global_max(LOCAL_MAX_Q, GLOBAL_MAX_Q,  PE_IO, ierr
)
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)  '
RANGE OF W-MOMENTUM DELH_W     = ', GLOBAL_MIN_Q, GLOBAL_MAX_Q
         IF (myPE == PE_IO) WRITE(UNIT_CUT_CELL_LOG,1000)
'##############################################################
#'

      ENDIF

      RETURN


      END SUBROUTINE PRINT_GRID_STATISTICS
```

post_cbar_time

```
PBED
1
T
10
10
0.0, 7200.0 !Start, Stop Time
N
Scalar
1
10
2,101 !xmax+1
Y
2,101 !ymax+1
Y
1,1
cbar_c.dat
-1
0
```

post_epg

```
PBED
1
T
10
10
0.0, 7200.0 !Start, Stop Time
N
EP_g
10
2,101 !xmax+1
Y
2,101 !ymax+1
Y
1,1
void.dat
-1
0
```

VITA

Name: Michael Adam Martin

Address: Texas A&M University
Department of Mechanical Engineering
3123 TAMU
College Station TX 77843-3123
Phone: (979) 845-1251
Fax: (979) 845-3081

Education: B.S., Mechanical Engineering, Texas A&M University 2010
M.S., Mechanical Engineering, Texas A&M University 2012