Optimization of P3HT-PCBM Polymer Solar Cells Through Device Simulation and Manufacturing

James Boom
Department of Computer Engineering
California Polytechnic State University, San Luis Obispo
June 14, 2010

Table of Contents

Table of Figures

Table of Tables

***Abstract* – Given a good model and implementation of that model, computer simulation can be used to reduce the time and material costs of research. To this end I worked with other students to manufacture, test and simulate the single layer P3HT-PCBM solar cell. Using the data collected from this project, future work can then be done with the project's simulator to further optimize these types of solar cell devices.**

I. Project Goals, Motivation, Context and Justification

The demand for affordable, reliable, and clean power has increased over recent years and is expected to continue to do so in the future. I am hopeful that organic solar cells or organic photovoltiacs (OPVs) help to meet this demand in the future [1]. The costs of power generation potential for photovoltaics (PV) in general, however, are not currently competitive with existing power sources such as coal. In early 2009, First Solar announced it had achieved a cost of $0.98 / Watt [2], twice the cost of the $0.50 / Watt goal outlined in the PV industry road map [1]. OPVs need more device optimization research to advance the technology towards competetitive device efficiencies.

With this need for optimized devices in mind, I worked with three other students in continuing the optimization work of an ongoing organic solar cell project at Cal Poly. I took the project's existing current-voltage simulator for polymer solar cells and made it a more user-friendly application. I also worked in the polymer electronics lab at Cal Poly to manufacture and test organic solar cell devices, whose characteristics (thickness, absorption, current-density, etc..) we input into the program to obtain performance predictions of different device thicknesses.

II. Design Requirements and Specifications

In addition to the marketing requirements listed below in table I, engineering requirements can be found in table II of appendix I; table II details how each engineering requirement meets one or more marketing requirements that came out of my discussions with my primary client Dr. Bob Echols.

Table I:
Marketing Requirements

Requirement Number	Requirement
1	The simulator will be user friendly.
2	The simulator will run multiple simulations with one execution.
3	The simulator will produce reasonable approximations.
4	Simulation results and lab results will be compared.

To facilitate marketing requirements 1 and 2, I changed the file format used by the simulator as discussed in "Simulation." The simulator's previous authors wrote the code for the charge generation and flow in an organic polymer device. So given reasonable recombination and dissassociation correction factors, r and B respectively, one expects the simulator to generate reasonable current density (J) vs. Voltage (V) characteristics and subsequently power density (PD), which we calculate using (1), sought in marketing requirement 3.

$$PD = J*V \qquad (1)$$

To satisfy marketing requirement 4, three other students and I fabricated and tested several poly(3-hexylthiophene) – phenyl-C61-butyric acid methyl ester (P3HT – PCBM) solar cells. In order to perform the comparison between the manufactured devices and the simulator results as discussed in the "Testing" and "Simulation" sections, we gathered J-V, thickness, and optical density (OD) data.

III. Parts and Costs

The project required the materials and equipment, listed below in the "Required Materials" section and the person hours in table III. I found it difficult to pin down the total cost of materials, as some materials were already in the lab and not all of the materials purchased were used in the course of the project. One of the biggest cost contributions for materials comes from the active layer polymer P3HT.

$$\text{P3HT Cost} = \#\text{ devices} * (g_{P3HT} / \text{device}) * (\$ / g_{P3HT}) \qquad (2)$$

We made a total of 48 devices, with an average amount of 15mg of P3HT per device at a cost of about $500 per gram of P3HT; using (2) we find the approximate cost of the P3HT for the project to be $360.

The team completed four batches of solar cells over the course of this portion of the project. This component of the project took approximately 20 hours on my part and would cost $300 for my time. Given an appoximate cost of $1000 for materials plus the $750 cost of labor, this project would come to a total investment of $1750 and 50 person hours.

Table III:
Cost of Labor For Project

	Units	Cost / Unit	Total Cost
Coding	20 hrs.	$15.00/hr.	$300.00
Lab	20 hrs.	$15.00/hr.	$300.00
Documentation	10 hrs.	$15.00/hr.	$150.00
Total Labor	50 hrs.	$15.00/hr.	$750.00

In addition to each lab hour that I commited, I needed a lab partner as per the Department of Electrical Engineering lab safety guidelines. Normally I would include their time as well in calculating the project costs, but since the material engineering students of the team are turning in a separate project report for their work on the TEM, AFM and in the lab, I did not factor in their time for project costs.

Required Materials

1. Lab Coat: to provide a barrier between workers and materials and/or equipment
2. Glove Liners: to absorb sweat from wearing non-breathable gloves
3. Disposable Gloves: to provide a barrier between workers and materials and/or equipment. ~$8 / box
4. Fume Hood: to capture fumes from chemicals used outside glove box
5. Cotton Swabs: to clean large particulates from substrates
6. Sonic Baths: to vibrate loose smaller contaminants from the substrates
7. UV Ozone Reactor to further clean substrates after swabbing and sonic baths
8. Spin Coaters: to spin coat on polymer blend and PEDOT
9. Glove Box: to protect the devices from contamination through production and testing, until they are packaged
10. Hot Plates: to anneal devices and heat dissolved P3HT-PCBM blend
11. Clean Bottles: to hold polymer, solvent, etc...
12. Filters: to filter P3HT polymer
13. Evaporator: to evaporate on Aluminum electrode
14. Dolan Jenner MI-150 Spectrum Light Source: to shine on the completed devices for in box testing
15. Measurement Equipment (SourceMeter, etc...): to test the device performance
16. Lab computer with labview: to run test program and equipment
17. PEDOT:PSS: planarizes the surface for the active layer, and provides a slightly larger work function for the anode.
18. P3HT: an organic polymer which releases exitons when struck by incident photons. ~$400 / gram
19. PCBM: a fullerene derivative which collects electrons from disassociated exitons
20. Chlorobezene: to dissolve the P3HT-PCBM mixture
21. Aluminum pellets: Evaporated onto the top of device to form one electrode
22. ITO coated glass substrates: the ITO will act as a transparent bottom electrode for the device.

IV. ABET Senior Project Analysis

Ethical

The team's work in the Polymer Electronics Lab for this project stemed from work already completed by other Cal Poly students such as Chris France [3] and Erik Everson [4]. Additionally some of the manufactoring processes drew from techniques employed in Professor Heeger's lab in Santa Barbra. The software portion of the project also drew upon simulator source code work completed by Tim Hider, Eric Everson, Chris France, Robert Echols, Beat Ruhstaller, and Scott Cambell. All of these people deserve credit for their indirect contribution to the overall project.

Also since the devices created through out the course of the project utilized harmful substances, the risks associated with working with said materials, talked about more in the "Health and Safety," as well as "Environmental" sections below, were disclosed to the lab technicians

Environmental

Using solar cells to capture sunlight, as intense as 1000W / m^2 on the Earth's surface [5] at certian times of the day, may help to reduce the ongoing damage to the environment which energy sources like coal and fossil fuels inflict. Like the construction of coal fired plants, however, the construction of solar cells and the plants to manufacture them requires energy and some hazardous materials.

Although low dosages of aluminum are a natural occurrence, the aluminum used for the top electrode (cathode) of the project's devices poses some health risk if enough of it accumulates in ground water. In addition to aluminum, the indium used in the bottom electrode (anode) poses a risk to microbes in the soil according to (cite).

While small amounts of the metals used in the devices naturally occur in ground water and soil, we need proper reclamation of worn out or unwanted devices to prevent toxic concentrations of metals in the ground water surrounding landfills.

Besides the metals used, the Safety Officer in Physical Chemistry at Oxford University classifies our polymer solvent, chlorobezene, as toxic and a possible carigen [6]. While toxic, finished OPV devices contian trace amounts, if any, of chlorobenzene and so then we focus on proper handling during manufacturing and disposale of unused polymer-fullerene-solvent solution.

Health and Safety

As with any endeavor involving potentially hazardous materials such as solvents and lab equipment such as the glove box and spin coaters, we must consider any health and safety risks associated with them.

The glass substrates in these devices poses little health risk, but result in sharp edges which cut flesh and glove box seals. Likewise if broken, the glass bottles used to hold various solutions result in sharp edges and uncontained chemicals.

According to a 2005 paper by a group of Japanese researchers, fullerenes like PCBM pose no significant health risk. Their research showed no abnormal mutation rate and no noticable health differences between the control group of mice and the groups of mice administered 2000mg of C60 per kg of body mass [7]. Chlorobezene, a possible carcogen, does pose a health risk to those persons who might come in contact with it, i.e. lab technicians. As such the we store the container of chlorobezene in the glove box. Since we mix and spin coat the P3HT-PCBM solution onto a substrate on the chemical side of the glove box, we placed the solvent container on the same side to provide ready access to it and minimize the impact of any potential spills.

According to material safety data sheets from ScienceLab.com, isopropyl and acteone, used to clean substrates, also pose a health risk if they come into contact with skin or eyes or an open flame [8][9]. To mimize the ignition risk and protect technicians, we only used the two chemicals under the fume hood and then stored them in the flammables locker in closed containers.

In addition to the materials used in the project, care must be taken in operating the pumps for the glove box equipment and in moving things into or out of the box as to avoid a sudden change in pressure in any part of the box or the pumps. Additionally the hot plates used in the project easily burn human flesh or ignite flammable materials, such isopropyl and acetone vapors, so it will be important to be mindful of the placement of the plates and any other objects.

Sustainability

Although solar energy provides humans with a renewable energy resource of upto 1000 W/m2 of power during daylight hours, over time the performance of photovoltaic devices degrades and the packaging wears down from exposure to the elements. So then the issue of sustainability comes not from the energy source in this case, but the devices used to harnass the energy.

The plastic packaging materials used in industry OPVs may be recyclable and other groups are conducting research into the feasibility of reusing the P3HT and PCBM materials in the active layer. By recycling the components of OPVs, manufacturers need fewer new materials and fewer potentially toxic materials end up in landfills.

Social

As this project involved working with two faculty advisors and three other students, scheduling initially slowed the progress of the project. Fortunately our unofficial team leader Steve Hawks, took on much of the lab schedule coordination and all the team members set aside time regularly to meet for discussion and work.

Solar energy also impacts society at the national and gloabal level. Widespread availability of inexpensive solar modules allows people to produce some if not all the power they need for electronic applications such as a refrigirator or lighting at night. This effect might help to bring fiscally poorer regions to a higher standard of living.

In today's industrialized countries, where cell phones, computers, and other electronics prevade the society, cheap power, especially power viewed as sustainable to produce, may lead to an increase in power consumption and perhaps even demand on existing power generation like coal and nuclear plants.

Political

Politics deals with the guidelines for the distribution and consumption of common resources. The Department of Electrical Engineering at Cal Poly operates one Polymer Electronics Laboratory for everyone to share. Fortunately for this project, the polymer electronics lab only meets once a week and the rest of the time no other classes or projects needed the lab; this made getting access to this common resource much easier.

In a larger context, our nation's political body, the U.S. Government, encourages consumers to use solar energy to offset some of their consumption of grid supplied power which typically comes from coal fired or nuclear power plants. According to the U.S. Department of Energy, "Consumers who install solar energy systems (including solar water heating and solar electric systems), small wind systems, geothermal heat pumps, and residential fuel cell and microturbine systems can receive a 30% tax credit for systems placed in service before December 31, 2016" [10]. Such an ecomonic policy serves to increase the diffusion of solar technology and reduce the dependence on coal and nuclear fired power plants which also create byproducts harmful to the common resource of the earth.

Economic

Economics deals with the flow of goods and services from one entity to another. In the context of this project, we purchased polymer, syringes, and swabs among other things to carry out our work. Some of the manufacturing was delayed while we waited for companies to process and ship our orders.

Another important economic component to our work relates to the PV industry. Through innovation and consumer investment, the PV industry supplies thousands of workers with jobs, where the workers help to create a more environmentally friendly world.

Manufacturability

The P3HT-PCBM blend used for the OPVs in this project dissolves in chlorobenzene at near room temperature, whereas silicon needs to be heated to 1414 degrees Celsius for liquification [11]. Due to the low liquification temperature and solution based processing, OPV manufacturers might use the roll-to-roll processing technology used for newspapers. Due to the prohibitively large investment of a roll-to-roll setup, however, for the course of this project we stuck to spin coating our device layers in the polymer electronics lab.

Organic polymers degrade in the presence of contaminants such as water and oxygen. For this reason, once we cleaned a substrate and applied the PEDOT layer via spin coating, all further processing steps occured inside the polymer electronics lab's glove box.

In order to clean the substrates, we first subjected them to a distilled (DI) water and joy dish soap scrub. Then we sonicated the substrates in three seperate baths: DI water, acetone, and isopropyl alcohol. For our last cleaning step, we put the substrates into a UV Ozone reactor.

After thoroughly cleaning the substrates we spun coated an approximately 40 nm layer of PEDOT, which serves to reduce the flow of holes from the Indium Tin Oxide (ITO) anode to the P3HT. To remove the excess PEDOT around the edge of the device, we used swabs dipped in DI water and then subjected the substrates to an anneal at 140° C for 10 min to help evaporate the water used to dissolve the PEDOT. The substrates then entered the glove box for the rest of the processing steps as our active layer utilizes an organic polymer, P3HT.

Unlike the PEDOT, we needed an organic solvent to dissolve the P3HT-PCBM mixture; for our project we used the solvent chlorobezene. We mixed a 1:1 ratio of P3HT and PCBM with the solvent to obtain concentrations of 12mg/ml and 17mg/ml, depending on the device run. Using a magnetic stir bar, we mixed the solution for at least 24 hours at 150° C. Once thoroughly mixed we spun coated the blend on top of the PEDOT layer and again wiped away the excess away from the edges, this time using swabs dipped in chlorobezene.

After applying the polymer layers of the device, we performed the final processing step, cathode deposition. To accomplish this step, we moved the devices to a vacuum chamber, where under low pressure and high current conditions aluminum pellets heat to the point of vaporation. The vaporized aluminum floated upward and deposited itself on the devices in a pattern determined by the mask used. With completed devices, we then tested them as described below.

V. Testing

In order to characterize the devices made in the lab, we tested them under three different lighting conditions. To make electrical contact with a device we used an eight pin jig so that we could test all four pixels on a device without having to move it. The jig connects to a switch, which connects to a Keithly SourceMeter.

Using a computer running labview, we ran the source-meter from -1 to 1 volt for each pixel under each lighting condition. We collected the current and voltage characteristics and used a custom program which calculates from the data the current density, fill factor, open circuit voltage and power conversion effiency.

First we tested the devices under dark conditions. With the I-V data collected from this test we calculate the leakage current of a particular pixel according to (3) as the average current over the voltage interval -1 volt to 0 volts. I have not included dark current data here, but for the most part leakage current was small.

$$I_{LEAKAGE} = I_{AVERAGE}\ [-1 <= V_{BIAS} <= 0] \quad (3)$$

Under low light conditions or more precisely with a 40 W/m2 Dolan Jener light source, we obtained the seoncd set of I-V characteristics. Using the device's short circuit current density (Jsc) and open circuit voltage (Voc), we calculate its fill factor using (4), and power conversion effiecency (PCE) using (5) under lab lighting conditions.

$$\text{Fill Factor} = (PD / Jsc*Voc) * 100\% \quad (4)$$

$$PCE = (PD / I) * 100\% \quad (5)$$

The first device run the team completed yielded solar cells with PCEs ranging from 0.0 - 0.9% before annealing as shown in table IV. After the anneal step, we observed a marked, about 50%, degredation of effiency. We hypothesised that the low effieciency may have been due to too much aggregation of the PCBM.

Table IV:
Selected Results from Device Run 1

Pixel	J_{sc} (A/m^2)	V_{oc} (V)	FF (%)	Efficiency (%)
7A	-1.344	0.053	13.761	0.024
7B	-1.659	0.457	44.042	0.834
7C	-1.604	0.472	46.431	0.878
7D	-1.244	0.338	33.770	0.355
10A	-1.023	0.468	33.917	0.406
10B	-1.503	0.122	24.397	0.111
10C	-1.281	0.435	30.245	0.421
10D	-1.002	0.426	30.256	0.323

Table V:
Selected Results from Device Run 2

Pixel	J_{sc} (A/m^2)	V_{oc} (V)	FF (%)	Efficiency (%)
1A	-12.890	0.366	35.549	0.195
1B	-1.255	0.260	26.637	0.010
1C	-1.225	0.331	31.592	0.015
1D	-12.323	0.412	42.930	0.254
2A	-14.709	0.466	43.516	0.347
2B	-1.354	0.470	48.612	0.036
2C	-1.648	0.482	46.965	0.043
2D	-13.896	0.463	45.329	0.339

With new PCBM, we manufactured a second batch of devices which operated at about 0.0-0.4% power conversion effiecency, as seen in table V. A decrease, rather than increase. of performance with the younger PCBM indicates that some other factor than the PCBM contributed to low performance in the first device run. Perhaps then the P3HT aggregated too much and not the PCBM.

With a younger batch of P3HT, the team produced devices with PCEs from 0.2 – 1.0% as highlighted in table VI, with only a couple pixels near 0.0% and a single pixel at 1.4%. Perhaps then the older P3HT in the glove box degraded over time.

Table VI:
Selected Results from Device Run 3

Pixel	J_{sc} (A/m^2)	V_{oc} (V)	FF (%)	Efficiency (%)
1A	-1.6208	0.4732	53.3063	1.0222
1B	-1.9078	0.4560	45.1430	0.9818
1C	-1.6435	0.4262	47.1602	0.8259
1D	-1.4110	0.4401	51.8740	0.8054
2A	-1.5238	0.4591	53.8706	0.9423
2B	-1.8714	0.4480	34.0584	0.7138
2C	-1.8850	0.4704	49.6131	1.0999
2D	-1.4088	0.4597	56.2972	0.9114

Data from the fourth batch of solar cells seen in table VII shows no improvement with both younger P3HT and PCBM and in fact the performance goes down. At this time, we do not know why the old PCBM – new P3HT blend for the active layer produced the best devices. The tunneling electron microscope images taken at UCSB by Steve Hawks do not appear to show significant differences in topology.

Table VII:
Selected Results from Device Run 4

Pixel	J_{sc} (A/m^2)	V_{oc} (V)	FF (%)	Efficiency (%)
10A	-0.757	0.462	36.590	0.319
10B	-0.763	0.356	31.190	0.212
10C	-0.753	0.419	38.098	0.300
10D	-0.793	0.412	35.278	0.288
11A	-0.797	0.258	28.380	0.146
11B	-0.628	0.331	31.847	0.165
11C	-0.602	0.376	45.013	0.255
11D	-0.700	0.053	14.920	0.014

Using a mobile testing setup, we tested the devices under our third and final lighting condition, outside. Under these conditions, devices recieve as much as 1000W/m2 of input power from the sun [5]. We only tested select devices under this third lighting condition as the 1000W/m2 intensity occurs only at two points during the day and even then the light intensity flucated during these times due to cloud cover and other amospheric conditions. Of the devices tested under these conditions, the best device achieved approximately 2% PCE, nearly twice what it had achieved under the low light conditions in the lab. Unfortunately once we removed a device from the box, it began to degrade as it oxidized.

VI. Simulation

As the flow digram depicted in fig. 1 shows, the simulator software has a primary loop which executes code that satisfies marketing requirement 2 and a secondary loop to carry out the multiple iterations of the current at every bias voltage needed for each simulation.

The simulator previously read and wrote files as raw numbers like the example input file shown in fig 2 of the Appendix. This setup proved to be, even for experienced users, cumbersome because users had to look in the simulator source code to figure out which values corresponded to what parameter. With the modifications I made in Winter quarter 2010 as seen in fig. 3, the labels in the files make it much easier for someone famaliar with the basic physics of the P3HT-PCBM solar cell to interpret the values. Implementing these changes required modifying the Import_Values, Import_SweepParams, and main functions found in the demo.c file of appendix II.

The JV simulator now also runs multiple simulations from one device parameter file. The user specifies a range for the active layer thickness (L) and the recombination correction factor (ff_rec) with a starting value, ending value and step size as highlighted in fig. 3. To implement ranges for L and ff_rec, I wrote a function called runsets which resides in demo.c of the code appendix as well and added some glue code to the main function to facilitate its use.

For each of the simulations the program determines the PCE from the maximum PD and light intensity (I) shone on the device as decribed in (4). To find the maximum PD, the program calculates each J*V pairing and selects the minimum, in other words the J-V point where the device outputs the most power. Once the program has simulated each device, it then determines the simulation which had the highest PCE, i.e. the most negative PCE since the PCE routine doesn't drop the negative sign from the max power density value, and reports this value to the terminal before exiting as seen in mid left portion of the simulator flow in fig. 1.

Originally I purposed using the improved simulator between device manufacturing runs to tune the simulator's correction factors and provide a better guess for optimal thickness. Due to large variation in device performance in the lab however, this idea changed to using the simulator to determine the correction factors for recombination and dissassociation for the best reported P3HT-PCBM OPVs to date, the 4.4% PCE devices from Yang Yang Laboratories [12].

In order to simulate Yang Yang's devices, I needed an absorptivity profile, a Voc, and a thickness. The last two parameters came from [12] and the first parameter came from Emily Robertson's work. Emily calculated the absorptivity profile for our devices using (6) with the absorption and thickness measurements she took.

$$\alpha(\lambda) = OD(\lambda) / Thickness \qquad (6)$$

Using an external quantum effiency (EQE) simulator with this profile, a Voc of 0.6V, a thickness of 210nm, and zero recombination under low light conditions, I varied beta until the simulated EQE matched the EQE for Yang Yang's devices. A beta of 0.82 produced this matched.

Running the JV simulator with our absorptivity profile, the beta of 0.82, and a thickness of 210 nm yielded a PCE of 2.6% and a Jsc of 106A/m2 at 0.00156 recombination correction, while Yang Yang reported a PCE closer to 4.4% with the same Jsc. This sizable difference in PCE suggests that simulator does not achieve the 67% fill factor which Yang Yang reports for his best devices. With the beta and r factors, I took advantage of the new simulator capabilities and ran a range of thicknesses to determine that the 210 nm thickness reported for these devices achieved the maximum PCE.

$$r(\text{beta}) = \text{beta} * 6.061 - 1.697 \tag{7}$$

Using the simulator with the Yang Yang beta of 0.82 requires a recombination factor of 3.23 to achieve the Jsc of 2.058 A/m2 from our best pixel. Such a high r seems unlikely, rather our pixel probably falls in the r <= 1 range. With a r of 1.00, the simulator requires a beta of 0.445 to match the the Jsc from our pixel. On the lower end of the recombination spectrum, I assumed that our pixel had an r of at least 0.00156, the Yang Yang recombination. With this r, I found a beta of 0.28. Using these two endpoints with three additional beta-r pairs in this range lead to the linear realtionship of (7) for r and beta.

VII. Conclusions

In CPE461 and CPE462 I labored about 50 hours to manufacture, test and simulate single layer P3HT-PCBM solar cells. For the software portion of the project, I improved the user friendliness of the JV simulator by adding value labels and enabling parameter ranges for the thickness and recombination correction factor. I also improved the coding-friendliness by removing uneeded code and adding more comments about the function of the code. Using the improved simulator I found that the best P3HT-PCBM devices reported in the literature have high exiton dissassociation (0.82) and low recombination (0.00156) factors in the simulator. Our best devices on the other hand need additional improvement to achieve optimal performance to produce a relation similar to (7), but with a higher slope.

Additonally obtaining EQE data from our devices, rather than using EQE data from Yang Yang's paper will lead to a better estimation of beta and subsequently r for our devices.

For the harware portion of this project, I manfacutured several P3HT-PCBM OPVs and collected JV data, some of which I used in conjunction with the JV simulator. I found that at least for this project, these OPVs respond with large performance changes given small changes in manufacuring conditions, making results difficult to reproduce.

VIII. Acknowledgements

I acknowledge Dr. Robert Echols for his advising on this project as well as the Department of Physics at Cal Poly for funding. I also thank Dr. David Braun for the use of the Polymer Electronics Lab at Cal Poly. In addition to the faculty who helped make this project possible, I acknowledge the contribution of the other students on this project: Emily Robertson's work with manufacturing, AFM and absorption measurements, Steve Hawks's work with manufacturing and TEM, and Tim Hider's work with manufacturing and coding.

Appendix I: Figures and Tables

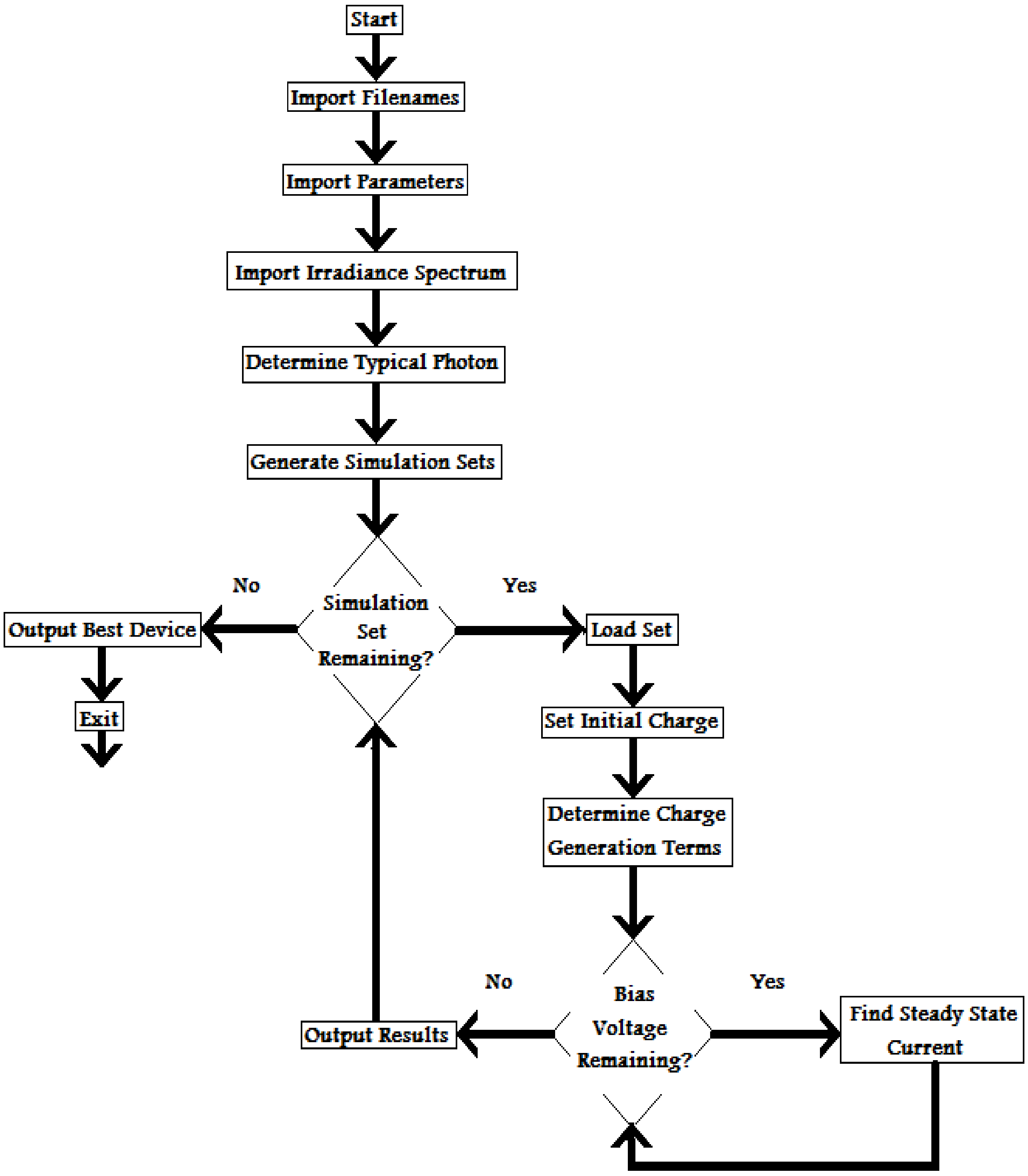

Fig. 1: Flow diagram of the polymer solar cell JV simulator. The program simulates the JV characteristics of several devices, each having a thickness-recombination correction factor pair calculated in the "Generate Simulation Sets". For bias voltage applied to the simulated device (the lower right loop) multiple iterations of the current generation steps in the program occur to find a steady state current. The bias voltages are specified in the sweep parameter file and the maximum number of iterations is specified in the input parameter file.

```
2-9-Win3B
40
1e7
1000000
200e-9
300
4.3
4.7
5.3
3.4
1e-9
3e-009
100
5
100
2.5e28
880
600
1E-24
1.31261E-24
0.34
0.38
0.15
0.82
0
```

Fig. 2: Example of an unlabeled input file used by the old version of the JV simulator. Notice how the file is simply a list of numbers, none of which contain a label indicating what the numbers mean. Users wanting to edit the parameters must look up the parameter order in the source code of the program.

Output File: 2-9-Win3B
Illumination(W/m2): 40
Absorption(cu): 1e7
Iterations: 1000000
<u>Thickness(m): 200e-9 230e-9 10e-9</u>
Temperature(K): 300
Cathode Work Function(eV): 4.3
Anode Work Function(eV): 4.9
HOMO(eV): 5.3
LUMO(eV): 3.4
Zero Field n Mobility: 1e-9
Zero Field p Mobility: 3e-009
Number of Cells: 100
Accuracy: 5
Time Step: 100
Chargable Site Density: 2.5e28
Field Dependent n Mobility Temperature(K): 880
Field Dependent p Mobility Temperature(K): 600
Field Dependent n Mobility Constant: 1E-24
Field Dependent p Mobility Constant: 1.31261E-24
Zero Field n Mobility Delta: 0.34
Zero Field p Mobility Delta: 0.38
<u>Recombination Rate Factor(r): 0.01 0.15 0.01</u>
Exiton Recombination Factor(Beta): 0.82
Error Level: 0

Fig. 3: Example of a labeled input file used by the most recent working version of the JV simulator. The numbers in the file now contain labels indicating the paramter name and units associated with the parameter. The thickness and recombination rate factor now get entered as ranges, rather than single values.

Table II:
Engineering Requirements

Marketing Requirement(s)	Engineering Requirement	Justification
1	The simulator will utilize input files which have parameters labeled.	The end user will likely want to change the input parameters to the simulator and so the format for the input files needs to clearly indicate which numbers correspond to which parameters.
1	The simulator will output files which have results labeled	The end user needs to know what the simulator results indicate, i.e. the result 250 doesn't provide any context, whereas the result Current Density (mA/cm3): 250 does.
2	The simulator will accept parameter ranges from the input files	In order to run multiple simulations using one input file, the simulation must read in parameter ranges, not just parameter values
1, 3	The simulator software will require only one program be initialized for the simulation to run.	Having the user run more than one program could lead to out of order execution, which would yield potentially erroneous simulation results.
3	Lab results will be used to tune simulator after each batch run	Using real world data, the "fudge" factors in the simulator model can be tuned to better match the operation of actual solar cells
4	Devices created in the lab will be characterized using voltage, illuminated current density, dark current density, absorption, etc ... measurements	In order to compare the simulated devices to the real devices, we need performance data from the real devices which has a simulated counterpart.

Appendix II: Simulator Source Code

File: jvsim.h

```
/*******************************************************************************
 *  Project: Polymer Photovoltaic Modeling Code
 *  File: jvsim.h
 *  Author: James Boom, engineer.jboom@gmail.com
 *  Date: 2010-04-19
 *  Overview: This file contains all the library includes, function prototypes,
 *  #defines and other global variables needed for compilation of demo.c
 ******************************************************************************/

//Needed Libraries
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

//Constant Definitions
#define e 1.602e-19             // Fundamental Charge in units of Columbs
#define pi 3.141593             // You better know what this is
#define eo 8.8542e-12           // Permativity of Free Space
#define kb 1.3807e-23           // Boltzman constant
#define hc 1240                 // hc in units of eV*nm
#define toc_file "Xworking.toc"  // file that stores filenames of parameters
      /* For the irradiance spectrum and the absporption spectrum the file
       *      standard is: wavelength in first column in ascending order with
       *      units of nm target value in second column in SI units.
       */

//Global Variable Definitions
double wavelen[4002], illum[4002], photon[4002], photonbin[4002],
      Generate[4002], *a;
int  err;
char inf_swp[25], inf_par[25], inf_abs[25], inf_absx[25], inf_irr[25],
      outfile[25];
int x, z=0, i, T, savestep, cells, accuracy, Emax;
char autostop, *filename, datfile[25] = "newinput.dat";
char label[25], ivfile[25], volt[25], infile[25];
double n[2002], p[2002], E[2002], Jn[2002], Jp[2002], mobp[2002], mobn[2002],
      Ri[2002];
double dielec=3.5, V, L, dx, Temp, t, dt, c, dJn1, dJn2, dJp1, dJp2, Norm_E,
      Jold;
double Lbot, Ltop, deltaL, rbot, rtop, deltar;
double totn, totp, Norm_f, Norm_inj, Norm_surf;
double V0, ksi, phi_c, phi_a, HOMO, LUMO, No, nc, ff_rec, ff_int;
double psi, y, corr1, corr2, corr3, corr4;
double Jave, dJ, Rave, m0n, m0p, mob[2], vmax;
double V1, V2, dV,  bias[50], J[50], R[50];
// For calculating the field dep. mobility
double gam_n, gam_p, T0n, T0p, Bn, Bp, photoncheck;
FILE *f1, *f2;
double sum;  //used for debugging generate spectrum term and photon count term
double phottot, illnorm;
double delp, deln, mSTp, mSTn;
double JscInterp[4001];
//coefficients in spline routines
double splineB[1][4001], splineC[1][4002], splineD[1][4002];
static int search_interval;
double Iwavelen[4002], Illum[4002];
double Awavelen[4002], Abs[4002], Abs1[4002];
int Inumbins=0, Anumbins=0;
double *thicknesses, *recombs;
char **outfiles;

//Function Prototypes
double Import_Spectrum();
double Count_Photons();
void Import_Files(char *);
void Import_SweepParams(char *);
void Import_Values(char *);
void Generate_Spect();
void Import_ChargeDistr(int);
int runsets(double, double, double, double, double, double);
int maxpower(double *, int, int);
void Run_Loop();
void Export_Values(int);
int maxi(double *, int, int);
double max(double *, int);
void SplineCL(int, double *, double *, double *, double *, double *);
double SplineValCL(int, double, double *, double *, double *, double *,
      double *);
```

File: demo.c

```
/*******************************************************************************
 *  Project: Polymer Photovoltaic Modeling Code
 *  File: demo.c
 *  Author: James Boom, engineer.jboom@gmail.com
 *  Previous Authors: Tim Hider, Eric Everson, Chris France, Robert Echols,
 *    Beat Ruhstaller, Scott Cambell
 *  Date: 2010-04-19
 *  Overview: This code is appropriate for simulating JV characteristics of
 *    blended materials such as M3EH-PPV with CN-ether-PPV when exiton
 *    dissociation is assumed to be taking place throughout the bulk.
 *  Notes: This version does not use non-dimensional quantities. It incorporates
 *    the injection/recombination currents as spelled out in Campbell Scott
 *    and George Malliaras's paper.
 ******************************************************************************/

//Contains all other #include, #define, prototypes and gloable variables
#include "jvsim.h"

/*******************************************************************************
 * Function: int main()
 *
 * Inputs: None
 *
 * Outputs: 0 if it exits successfully, nonzero otherwise
 *
 * Overview: Contains glue code and calls all functions needed to run simulation
 ******************************************************************************/

int main(int argc, char **argv)
{
  int i, runindex;              //Index variables for simulation number
  double *PCEs;                 //Array to store the PCEs of simulated devices
  int PCEi = 0;                 //Index in PCEs that has the maximum PCE
  double mPCE = 0;              //Value of the maximum PCE found at PCEs[PCEi]
  int verbose=0;                //Verbose flag
```

```
//Check for verbose mode
if(argv[1] != NULL){
    if(argv[1][1] == 'v') verbose = 1;
}

printf("Simulating a polymer-blend solar cell.\n\n");

//Prepare the simulator (import variables)
Norm_E=e/dielec/eo;
printf("Electric Field normalization=%1.2e\n",Norm_E);
filename = malloc(sizeof(char)*25);
a = malloc(sizeof(double)*4002);
// Using TOC file to find location of input and output files
Import_Files(toc_file);
// Imports sweep parameters from file inf_swp
Import_SweepParams(inf_swp);
// Imports parameters from file inf_par
Import_Values(inf_par);
// Imports irradiance and absorptivity from inf_irr and inf_abs
Inumbins = Import_Spectrum();
// Counts the number of photons in a bin based on wavelen
phottot = Count_Photons();
// Array of unique parameters to use
runindex = runsets(Lbot, Ltop, deltaL, rbot, rtop, deltar);
// Allocate enough space to store all the PCE of all simulations
PCEs = malloc(sizeof(double)*runindex);
// Start Simulation
for(i=0; i<runindex; i++)
{
    double PD[50];  // Holds power densities for each J-V pair
    int PDi = 0;    // Ends up with index in PD that has maximum power
    double PCE = 0; // PCE for this run

    if(verbose) printf("Running simulation %d...\n", i+1);

    //Set our two variables thickness and recombination factor here
    ff_rec = recombs[i];
    L = thicknesses[i];
    z=0;
    //Recalculate slice size for each L
    dx=L/cells;
    // Initial n, p charge distribution. chosen
    Import_ChargeDistr(cells);
    // Generates the charge generation term based on the illum and abs
    Generate_Spect();
    // subsequent n,p distribution's are taken from previous bias
    sprintf(label,"%s", outfiles[i]);
    sprintf(ivfile,"data/%s.iv",label);
    // opens file for storing current-density versus voltage info
    f2=fopen(ivfile,"w");
    V=V1;
    fprintf(f2,"V(Volt)\tJ(A/m2)\n"); // Print header for JV File
    // Fudge factor to z to solve rounding error when V2 is odd
  // When V2 was odd program stopped executing before iterating for V=V2
    while ((float) (z-0.000001)<=(V2-V1)/dV)
    {
      sprintf(volt, "%1.1lf",V);
    sprintf(filename,"%s%sV",label,volt);
    sprintf(datfile,"data/%s.dat",filename);
    printf("In process: %s...\n",filename);
    V=(V-(phi_a-phi_c));    // voltage across the device
```

```
        if (err > 0)
        {
            printf("*1*\tVoltage across device (before run loop) = %1.3f\n", V);
        }
        Run_Loop();           // Solve for steady state results
        Export_Values(i);     // Output results for run to file
        V=V+(phi_a-phi_c);   // Applied voltage

        // Record V, J, R for later use
        bias[z]=V;
        J[z]=Jave;
        R[z]=Rave;
        fprintf(f2,"%e\t %e\n",bias[z],J[z]);
        printf("\tTotal Photons: %1.3e%%\n", phottot);
        printf("...done\n");
        z++;
        V+=dV;
    }
      //Output Results Step
      printf("Outputting results...\n");
      printf("\tBias(V)\tJ(A/m2)\t\tPower Density(W/m2)\n");
      z=0;
    sprintf(filename, "%sJV.dat", outfiles[i]);   //Create JV output file name
      f1=fopen(filename,"w");             //Open file to store the JV curve data
    fprintf(f1,"Bias(V)\tJ(A/m2)\tPower Density (W/m2)\n");
      while ((float) (z-0.000001)<=(V2-V1)/dV)
      {
      //If in the 4th quadrant J becomes more negative, its most likely an error
        if((z > 1) && (bias[z] > 0) && (J[z] < J[z-1])) PD[z] = 0;
        // Otherwise record the simulated power density
        else PD[z] = bias[z]*J[z];
        // Screen output: V, J, PD
        printf("\t%1.2f\t%2.4e\t\t%2.3f\n", bias[z], J[z], PD[z]);
        // File output: V, J, PD
        fprintf(f1,"%e\t %e\t %e\n",bias[z],J[z],PD[z]);
        z++;
      }
      //Find the minimum point in PD, i.e. the most power out per unit area
      PDi = maxpower(PD,0,z);
      //PCE = (PDout / PDin)*100 (in %) Note: this is a negative number since
      //PDout is negative
      PCE = (PD[PDi]/illnorm)*100;
      //Store the PCE for each run in this array
      PCEs[i]= PCE;
      if(verbose){
        printf("\n\tMax Output Power Density:    %1.3f W/m2\n", PD[PDi]);
        //-PCE, since the negative carries through from PDout
        printf(  "\tPower Conversion Efficiency: %1.3f%%\n", -PCE);
      }

      fclose(f1);                         //Close jv.dat
      fclose(f2);                         //Close ivfile
      printf("\n"
        "...done\n");
  }
  //Since PCEs are negative, we can reuse the maxpower or min function
  PCEi = maxpower(PCEs,0,runindex);
  //-PCE, since the negative carries through from PDout
  mPCE = -PCEs[PCEi];
  if(verbose){
      for(i=0;i<runindex;i++){
        printf("Device %s had a PCE of %1.3f\n", outfiles[i], -PCEs[i]);
      }
  }
  // Report the maximum PCE from simulations and which device label is
  // associated with it
  printf("\n"
    "Maximum PCE: %1.3f\n"
      "From Device: %s\n"
      "\n"
      "Program Finished.\n", mPCE, outfiles[PCEi]);
  return 0;
}

/*******************************************************************************
 * Function: void Import_Files(char *toc)
 *
 * Inputs: toc, file which holds the names of the other files to use for
 *         simulation.
 *
 * Outputs: None
 *
 * Overview: Using the .toc file, imports the filenames for input and output of
 *          data.
 ******************************************************************************/

void Import_Files(char *toc)
{
      //Variable for reading used labels from files
  char *junk = malloc(60*sizeof(char));
  printf("Importing filenames...\n");

  sprintf(infile,"%s",toc);
  f1=fopen(infile, "r");
  // input sweep parameters filename
  fscanf(f1,"%s %s %s %s", junk, junk, junk, inf_swp);
  printf("\tSweep parameters filename is %s\n", inf_swp);
  // input parameters filename
  fscanf(f1,"%s %s %s %s",junk, junk, junk, inf_par);
  printf("\tGeneral parameters filename is %s\n", inf_par);
  // input AM 1.5 irradiance spectrum filename
  fscanf(f1,"%s %s %s %s",junk, junk, junk, inf_irr);
  printf("\tIrradiance spectrum filename is %s\n", inf_irr);
  // input absorptivity spectrum filename
  fscanf(f1,"%s %s %s %s",junk, junk, junk, inf_abs);
  printf("\tAbsorption spectrum filename is %s\n", inf_abs);
  printf("...done\n");
  free(junk);
}

/*******************************************************************************
 * Function: void Import_SweepParams(char *swpfile)
 *
 * Inputs: swpfile, file which contains the sweep voltage information
 *
 * Outputs: None
 *
 * Overview: Function to import the three voltage sweep parameters: V start,
 *         V end and V step.
 ******************************************************************************/
```

```
void Import_SweepParams(char *swpfile)
{
  //Variable for reading labels from files
  char *junk = malloc(60*sizeof(char));
  printf("Importing sweep parameters from %s...\n", swpfile);

  f1=fopen(swpfile,"r");
  fscanf(f1,"%s %s %lf",junk, junk, &V1);
  fscanf(f1,"%s %s %lf",junk, junk, &V2);
  fscanf(f1,"%s %s %lf",junk, junk, &dV);
  fclose(f1);
  printf("\tV1=%1.1lf\n\tV2=%1.1lf\n\tdV=%1.1lf\n",V1,V2,dV);

  printf("...done\n");
  free(junk);
}

/*****************************************************************************
 * Function: void Import_Values(char *devfile)
 *
 * Inputs: devfile, file which contains the device parameters
 *
 * Outputs: None
 *
 * Overview: Function to import device parameters.
 *****************************************************************************/

void Import_Values(char *devfile)
{
  //Variable for reading labels from files
  char *junk = malloc(60*sizeof(char));
  printf("Importing general parameters from %s...\n", devfile);

  f1=fopen(devfile,"r");
  fscanf(f1,"%s %s %s",junk,junk,filename); // output filename
  fscanf(f1,"%s %lf",junk,&illnorm);        // illumination
  fscanf(f1,"%s %lf",junk,a);               // absorption in c.u.=1e5 cm^-1
  fscanf(f1,"%s %d",junk,&T);               // # of iterations
  fscanf(f1,"%s %lf %lf %lf",junk,&Lbot, &Ltop, &deltaL);  // device thickness
  fscanf(f1,"%s %lf",junk,&Temp);           // temperature
  fscanf(f1,"%s %s %s %lf",junk,junk,junk,&phi_c);   // cathode work function
  fscanf(f1,"%s %s %s %lf",junk,junk,junk,&phi_a);   // anode work function
  fscanf(f1,"%s %lf",junk,&HOMO);        // highest occupied molecular level
  fscanf(f1,"%s %lf",junk,&LUMO);        // lowest unoccupied molecular level
  // electron zero field mobility @ 300K
  fscanf(f1,"%s %s %s %s %lf",junk,junk,junk,junk,&m0n);
  // hole zero field mobility @ 300K
  fscanf(f1,"%s %s %s %s %lf",junk,junk,junk,junk,&m0p);
  fscanf(f1,"%s %s %s %d",junk,junk,junk,&cells);          // number of cells
  fscanf(f1,"%s %d",junk,&accuracy);           // accuracy
  fscanf(f1,"%s %s %lf",junk,junk,&nc);        // time step factor
  // density of chargeable sites
  fscanf(f1,"%s %s %s %lf",junk,junk,junk,&No);
  // Temp. for calc. field dep. mob (elec)
  fscanf(f1,"%s %s %s %s %s %lf",junk,junk,junk,junk,junk,&T0n);
  // Temp. for calc. field dep. mob (hole)
  fscanf(f1,"%s %s %s %s %s %lf",junk,junk,junk,junk,junk,&T0p);
  // Const. term for calc field dep mob (elec)
  fscanf(f1,"%s %s %s %s %s %lf",junk,junk,junk,junk,junk,&Bn);
  // Const. term for calc field dep mob (hole)
  fscanf(f1,"%s %s %s %s %s %lf",junk,junk,junk,junk,junk,&Bp);
  // delta term used to find 0-field mob. at any temp
  fscanf(f1,"%s %s %s %s %s %lf",junk,junk,junk,junk,junk,&deln);
  // delta term used to find 0-field mob. at any temp
  fscanf(f1,"%s %s %s %s %s %lf",junk,junk,junk,junk,junk,&delp);
  // fudge factor (r-increases recombination rate)
  fscanf(f1,"%s %s %s %lf %lf %lf",junk,junk,junk,&rbot,&rtop,&deltar);
  // fudge factor (Beta-simulates the exiton recombination)
  fscanf(f1,"%s %s %s %lf",junk,junk,junk,&ff_int);
  // flag to indicate level of error checking: 0=none, 1=low, 3=high
  fscanf(f1, "%s %s %d",junk,junk,&err);
  fclose(f1);

  printf("...done\n");

  if (err == 1)
    printf("***Running code in error level 1***\n");
  else if (err == 2)
    printf("***Running code in error level 2***\n");


  printf("Calculating constants...\n");
  ksi=kb*Temp;
  printf("\t(diffusion coef./mobility) ksi=%1.2e\n",ksi);
  Norm_f=e*e*e/(4*pi*dielec*eo*ksi*ksi);
  printf("\tNorm_f=%1.3e\n",Norm_f);
  Norm_inj=16*pi*dielec*eo*No*ksi*ksi/e/e;
  printf("\tNorm_inj=%1.3e\n",Norm_inj);
  Norm_surf=16*pi*dielec*eo*ksi*ksi/e/e;
  printf("\tNorm_surf=%1.3e\n",Norm_surf);
  gam_n = Bn*(1/(kb*Temp)-1/(kb*T0n)); // gam_n = (E_0)^(-1/2)
  printf("\tgam_n=%1.3e\n",gam_n);
  gam_p = Bp*(1/(kb*Temp)-1/(kb*T0p)); // gam_p = (E_0)^(-1/2)
  printf("\tgam_p=%1.3e\n",gam_p);

  if (Temp != 300) {
    printf("\tMobilities @ 300K:\tm0n=%1.3e\tm0p=%1.3e\n",  m0n, m0p);
    printf("\tChanging mobilities from 300K to %1.1fK\n", Temp);
    mSTn = m0n*exp(deln/(kb*300/e));
    mSTp = m0p*exp(delp/(kb*300/e));
    m0n = mSTn*exp(-deln/(kb*Temp/e));
    m0p = mSTp*exp(-delp/(kb*Temp/e));
    printf("\tMobilities @ %1.0fK:\tm0n=%1.3e\tm0p=%1.3e\n", Temp, m0n, m0p);
  }
  else printf("\tRunning at 300K, no mobility recalculation necessary.\n");

  printf("...done\n");
  free(junk);
}

/*****************************************************************************
 * Function: void Import_ChargeDistr(int numslices)
 *
 * Inputs: numslices, the number of slices to simulate at. Each slice is
 *         L/numsilces thick.
 *
 * Outputs: None
 *
 * Overview: Function to generate the initial charge distribution. Currently
 *         this is 0 everywhere
 ****************************************************************************/
```

```
void Import_ChargeDistr(int numslices)
{
  printf("Importing charge distribution (no charge)...");

  //0 out charge distribution
  for (x=0;x<=numslices;x++) {
       n[x]=0.0;
       p[x]=0.0;
  }

  printf("done\n");
}

/*******************************************************************************
 * Function: void Import_Spectrum()
 *
 * Inputs: None
 *
 * Outputs: None
 *
 * Overview: Function to import the absorption and illumination spectrums.
 *          Also normalizes the illumination according to the desired total
 *          illumination
 ******************************************************************************/

double Import_Spectrum()
{
  int i=0;
  double IllumT=0.0;

  //Reading illumination spectrum
  printf("Reading irradiance spectrum from %s ", inf_irr);

  f1=fopen(inf_irr,"r");
  fscanf(f1, "%lf", &Iwavelen[i]);
  while (!feof(f1)) {
    fscanf(f1, "%lf", &Illum[i]);
    i++;
    fscanf(f1, "%lf", &Iwavelen[i]);
  }
  Inumbins=i-1;  //counts the number of values in the illumination file
  fclose(f1);
  i=0;

  printf("done.\n");

  //Reading the absorption spectrum
  printf("Reading absorption spectrum from %s", inf_abs);

  f1=fopen(inf_abs,"r");
  fscanf(f1, "%lf", &Awavelen[i]);
  while (!feof(f1)) {
    fscanf(f1, "%lf", &Abs[i]);
    i++;
    fscanf(f1, "%lf", &Awavelen[i]);
  }
  Anumbins=i-1; //counts the number of values in the absorption file
  fclose(f1);

  printf("...done.\n");
  // Normalizing using the value specified in the parameter file
  // Finding total illumination of unnormalized spectrum

  printf("Normalizing irradiance to %1.0f W/m^2...", illnorm);
  for(i=1;i<Inumbins;i++) {
    IllumT += (Iwavelen[i+1]-Iwavelen[i-1])*Illum[i]/2;
  }
  printf("\nInumbins: %d\n", Inumbins);
  IllumT += (Iwavelen[1]-Iwavelen[0])*Illum[0];
  IllumT += (Iwavelen[Inumbins]-Iwavelen[Inumbins-1])*Illum[Inumbins];
  // Normalizing

  for(i=0;i<=Inumbins;i++) {
    Illum[i]=Illum[i]*illnorm/IllumT;

  }
  printf("IllumT: %1.2e\tIllumNorm: %1.2e\n", IllumT ,illnorm);
  printf("done.\n");

  SplineCL(Anumbins, Awavelen, Abs, splineB[0], splineC[0], splineD[0]);

  for (i=0;i<=Inumbins;i++)
  {
      a[i] = SplineValCL(Anumbins, Iwavelen[i],Awavelen, Abs, splineB[0],
                  splineC[0], splineD[0]);
  }

  return Inumbins; //returns the number of entries in the abs and illum arrays.
}

/*****************************************************************************
 * Function: double Count_Photons()
 *
 * Inputs: None
 *
 * Outputs: double tot - the total number of photons from all "bins"
 *
 * Overview: This routine counts the number of photons, find the photon density
 *          and the photons per bin.
 ****************************************************************************/

double Count_Photons() {
  int i;
  double tot=0;
  printf("Counting photons...");

  // calculating for first boundary
  photon[0] = Illum[0]*Iwavelen[0]*1.0e-9/3.0e8/6.626e-34;
  photonbin[0]=photon[0]*(Iwavelen[1]-Iwavelen[0]);
  // calculating for bulk
  i=0;
  for(i=1;i<Inumbins;i++) {
    photon[i] = Illum[i]*Iwavelen[i]*1.0e-9/3.0e8/6.626e-34;
    photonbin[i]=photon[i]*(Iwavelen[i+1]-Iwavelen[i-1])/2;

  }

  // calculating for the second boundary
  photon[Inumbins] = Illum[Inumbins]*Iwavelen[Inumbins]*1.0e-9/3.0e8/6.626e-34;
  photonbin[Inumbins]=
        photon[Inumbins]*(Iwavelen[Inumbins]-Iwavelen[Inumbins-1]);
  i=0;

  printf("done.\n");
  fflush(stdout);

  if (err > 0)
    {
      printf("*1*\ti\tIwavelen\tphotonbin\ta\n");
      for (i=0;i<=Inumbins;i++) {
      printf("*1*\t%d\t%1.0f\t%1.4e\t%1.4e\n",
                  i, Iwavelen[i], photonbin[i], Abs[i]);
      }
      printf("*1*\ttotal photons : %1.3e\n", tot);
    }
  for (i=0;i<=Inumbins;i++) {
    tot += photonbin[i];
  }
  return tot;
}
/******************************************************************************
 * Function: void Generate_Spect()
 *
 * Inputs: None
 *
 * Outputs: None
 *
 * Overview: This routine calculates the generation terms for each slice
 ******************************************************************************/

void Generate_Spect()
{


  printf("Creating charge generation term...");
  for (x=1;x<=cells;x++) {
    Generate[x]=0;
    for(i=0;i<Inumbins;i++) {

      Generate[x] =
        Generate[x]+ff_int*(a[i]*photonbin[i]*e*exp(-a[i]*L*x/cells)+
            0.90*a[i]*photonbin[i]*e*exp(-a[i]*L)*exp(-a[i]*L*(cells-x)/cells));
    }
  }
  sprintf(filename, "data/absorbtion.dat");  //debugging print of absorbtion
  f1=fopen(filename,"w");      // absorbtion data
  for (x=1;x<=Anumbins-5;x++)
    fprintf(f1,"%d \t %lf\n",x, a[x]);
  fclose(f1);



  sprintf(filename, "data/absorbtion2.dat");  //debugging print of absorbtion
  f1=fopen(filename,"w");      // absorbtion data
  for (x=1;x<=Anumbins;x++)
    fprintf(f1,"%d \t %lf\n",x, photonbin[x]);
  fclose(f1);

  sum=0;
  for(i=0;i<Inumbins;i++){
        sum =+ sum+Generate[i];
  }
  printf("done\n");
  return;
}
/*****************************************************************************
 * Function: void Run_Loop()
 *
 * Inputs: None
 *
 * Outputs: None
 *
 * Overview: Function that handles most calculation. This is called once for
 *          each voltage value iterated through.
 ****************************************************************************/

void Run_Loop()
{
  t=0.00;
  i=0;
  Jave=Rave=0;
  savestep=100;
  autostop='n';
  while (autostop=='n') {
    if (i<30000) {
      c=nc*100.0;
    }
    else {
      c=nc;
    }

    // Calc. the electric field by integrating Poisson's Eq. and
    // satisfying the B.C. Calculate the mobilities
    ///Bulk of Device
    E[0]=0.00;
    for (x=1;x<=cells;x++)  // Integrate
      E[x]=E[x-1]+dx*Norm_E*(p[x]-n[x]);
    V0=0.00;
    for (x=1;x<=cells;x++)  // Find potenital
      V0=V0+E[x]*dx;
    E[0]=(V-V0)/L;
    for (x=1;x<=cells;x++) {
      // Adding constant to integral result
      E[x]=E[x]+E[0];
      // Calc. field dep. mobility
      mobp[x]=m0p*exp(gam_p*sqrt(fabs(E[x]+E[x-1])/2));
      mobn[x]=m0n*exp(gam_n*sqrt(fabs(E[x]+E[x-1])/2));
      // Calc. Recombination rate
      Ri[x]=Norm_E*(mobn[x]+mobp[x])*n[x]*p[x];
    }


       for (x=1;x<=cells-1;x++) {
      Jn[x]=e*(mobn[x]+mobn[x+1])*0.25*(n[x]+n[x+1])*
              E[x]+ksi*(mobn[x]+mobn[x+1])*0.5*(n[x+1]-n[x])/dx;
      Jp[x]=e*(mobp[x]+mobp[x+1])*0.25*(p[x]+p[x+1])*
              E[x]-ksi*(mobp[x]+mobp[x+1])*0.5*(p[x+1]-p[x])/dx;
    }

    ///Anode
    if (E[0]>0.0) {
      y=E[0]*Norm_f;
      psi=1.0/y+pow(y,-0.5)-pow(1.0+2.0*pow(y,0.5),0.5)/y;
      corr1=exp(pow(y,0.5));
      corr2=(pow(psi,-2.0) - y)/4.0;
      corr3=1.0;  // corr3 and 4 not used
      corr4=1.0;
      Jp[0]= Norm_inj*mobp[1]*exp(-(HOMO-phi_a)*e/ksi)*corr1-
      Norm_surf*mobp[1]*p[1]*corr2;
      Jn[0]=e*E[0]*n[1]*mobn[1];
    }
    else if (E[0]==0) {
      corr1=1.0;
      corr2=1.0;
      corr3=1.0;
      corr4=1.0;
      Jp[0]= Norm_inj*mobp[1]*exp(-(HOMO-phi_a)*e/ksi)*corr1-
      Norm_surf*mobp[1]*p[1]*corr2;
      Jn[0]= -Norm_inj*mobn[1]*exp(-(phi_a-LUMO)*e/ksi)*corr3+
      Norm_surf*mobn[1]*n[1]*corr4;
    }
    else {
      corr1=1.0;  // corr1 and 2 not used
      corr2=1.0;
      y=-E[0]*Norm_f;
      psi=1.0/y+pow(y,-0.5)-pow(1.0+2.0*pow(y,0.5),0.5)/y;
      corr3=exp(pow(y,0.5));
      corr4=(pow(psi,-2.0) - y)/4.0;
      Jp[0]=e*E[0]*p[1]*mobp[1];
      Jn[0]= -Norm_inj*mobn[1]*exp(-(phi_a-LUMO)*e/ksi)*corr3+
      Norm_surf*mobn[1]*n[1]*corr4;

    }
    if (corr2<=0||corr4<=0)
      printf("alert: psi^(-2)-f < 0 at anode \n");

    ///Cathode
    if (E[cells]>0.0) {
      y=E[cells]*Norm_f;
      psi=1.0/y+pow(y,-0.5)-pow(1.0+2.0*pow(y,0.5),0.5)/y;
      corr1=exp(pow(y,0.5));
      corr2=(pow(psi,-2.0) - y)/4.0;
      corr3=1.0;  // corr3 and 4 not used
      corr4=1.0;
      Jn[cells]= Norm_inj*mobn[cells]*exp(-(phi_c-LUMO)*e/ksi)*corr1-
      Norm_surf*mobn[cells]*n[cells]*corr2;
      Jp[cells]=e*E[cells]*p[cells]*mobp[cells];
    }
    else if (E[cells]==0) {
      corr1=1.0;
      corr2=1.0;
      corr3=1.0;
      corr4=1.0;
      Jn[cells]=Norm_inj*mobn[cells]*exp(-(phi_c-LUMO)*e/ksi)*corr1-
      Norm_surf*mobn[cells]*n[cells]*corr2;
      Jp[cells]=-Norm_inj*mobp[cells]*exp(-(HOMO-phi_c)*e/ksi)*corr3+
      Norm_surf*mobp[cells]*p[cells]*corr4;
      }
    else {
      corr1=1.0;  // corr1 and 2 not used
      corr2=1.0;
      y=-E[cells]*Norm_f;
      psi=1.0/y+pow(y,-0.5)-pow(1.0+2.0*pow(y,0.5),0.5)/y;
```

```
    corr3=exp(pow(y,0.5));
    corr4=(pow(psi,-2.0) - y)/4.0;
    Jn[cells]=e*E[cells]*n[cells]*mobn[cells]; // careful E < 0
    Jp[cells]=-Norm_inj*mobp[cells]*exp(-(HOMO-phi_c)*e/ksi)*corr3+
    Norm_surf*mobp[cells]*p[cells]*corr4;
  }

  // choose dt=max_cell_transit_time/c
  ///Bulk of Device cont.
  Emax=maxi(E,0,cells);
  mob[0]=m0n;
  mob[1]=m0p;
  vmax=max(mob,2)*E[Emax];
  dt=fabs(dx/c/vmax);

  // Calculate new charge densities
  for (x=1;x<=cells;x++) {
    dJn1=(Jn[x]-Jn[x-1])/dx;
    dJp1=(Jp[x-1]-Jp[x])/dx;
    dJn2=dJp2=ff_rec*e*Ri[x];
    // do not allow negative number densities
    n[x]=n[x]+(dJn1+Generate[x]-dJn2)*dt/e;
    p[x]=p[x]+(dJp1+Generate[x]-dJp2)*dt/e;
  }
  ///////////BCs with injection

  // Calculate averages, check autostop and transient
  if (i+1>=savestep) {
    if (err > 2)
    {
      printf("*1*\tJavg = %1.3e\n", Jave);
    }

    Jold=Jave;
    Jave=Jn[0]+Jp[0];
    Rave=0.0;
    dJ=fabs(Jn[0]+Jp[0]-Jold);

    for (x=1;x<=cells;x++) {
    Jave=Jave+Jn[x]+Jp[x];
    Rave=Rave+Ri[x];
    dJ=dJ+fabs(Jn[x]+Jp[x]-Jold);
    }
    Jave=Jave/(cells+1);
    dJ=dJ/(cells+1);
    Rave=Rave/cells;
    if (Jave!=0){
    if (fabs(dJ/Jave)<pow(10.0,-accuracy)) {
      autostop='y';
      printf("\tsuccessful convergence after %f (sec) and %d iterations !\n",
               t,i);
      printf("\tconvergence criteria dJ/J:  %f \n",dJ/Jave);
    }
    else if (i>=T) {
      autostop='y';
      printf("\tNO convergence after %f (sec) and %d iterations!\n",t,i);
      printf("\tconvergence criteria dJ/J: %f \t final value %f \n",
               pow(10,-accuracy), dJ/Jave);
    }
    else if (Jave!=Jave) {
      autostop='y';
```

```
        printf("\tnumerical error after %1.3e (usec) and %d
iterations !\n",t,i);
      }
      }
    }
    t=t+dt;
    // debugging purposes (printing itteration)
    i++;
  }
  printf("\t(last) dt = %e, total time= %e\n",dt,t);
  return;
}
/**************************************************************************
 * Function: void Export_Values(int)
 *
 * Inputs: index - int that represents the simulation number
 *
 * Outputs: None
 *
 * Overview: Function to output the results to data files. This is called for
 *          each iteration through the applied voltage
 *************************************************************************/

void Export_Values(int index)
{
  sprintf(filename, "data/%sgenerate.dat", outfiles[index]);
  f1=fopen(filename,"w");      // recombination data
  for (x=1;x<=cells;x++)
    fprintf(f1,"%e %e\n",L*x/cells, Generate[x]);
  fclose(f1);
  sprintf(filename, "data/recomb%s%s.dat",outfiles[index],volt);
  f1=fopen(filename,"w");      // recombination data
  for (x=1;x<=cells;x++)
    fprintf(f1,"%e %e\n",L*x/cells,Ri[x]);
  fclose(f1);
  sprintf(filename, "data/pdensity%s%s.dat",outfiles[index],volt);
  f1=fopen(filename,"w");    // pos. charge density
  for (x=1;x<=cells;x++)
    fprintf(f1,"%e %e\n",L*x/cells,p[x]);
  fclose(f1);
  sprintf(filename, "data/ndensity%s%s.dat",outfiles[index],volt);
  f1=fopen(filename,"w");    // neg. charge density
  for (x=1;x<=cells;x++)
    fprintf(f1,"%e %e\n",L*x/cells,n[x]);
  fclose(f1);
  sprintf(filename, "data/electric%s%s.dat",outfiles[index],volt);
  f1=fopen(filename,"w");    // electric field data
  for (x=0;x<=cells;x++)
    fprintf(f1,"%e %e\n",L*x/cells,E[x]);
  fclose(f1);
  sprintf(filename, "data/pcurrent%s%s.dat",outfiles[index],volt);
  f1=fopen(filename,"w");    // pos. current data
  for (x=0;x<=cells;x++)
    fprintf(f1,"%e %e\n",L*x/cells,Jp[x]);
  fclose(f1);
  sprintf(filename, "data/ncurrent%s%s.dat",outfiles[index],volt);
  f1=fopen(filename,"w");    // neg. current data
  for (x=0;x<=cells;x++)
    fprintf(f1,"%e %e\n",L*x/cells,Jn[x]);
  fclose(f1);
  sprintf(filename, "data/tcurrent%s%s.dat",outfiles[index],volt);
  f1=fopen(filename,"w");    // total current data
  for (x=0;x<=cells;x++)
    fprintf(f1,"%e %e\n",L*x/cells,Jn[x]+Jp[x]);
  fclose(f1);
  sprintf(filename, "data/n-p%s%s.dat",outfiles[index],volt);
  f1=fopen(filename,"w");          // difference in charge
  for (x=0;x<=cells;x++)
    fprintf(f1,"%e %e\n",L*x/cells,n[x]-p[x]);
  fclose(f1);
}

/* Finds the location of the last value in an array between positions l and n
   where the value of the array is greater than it is at the l^th position. */
int maxi(double a[], int l, int n)
{
  int j,index=l;
  double m=fabs(a[l]);

  for (j=l+1;j<=n;j++)
    if (m<fabs(a[j])) {
      index=j;
      m = a[j];
    }
  return index;
}

/* Finds the maximum value in the first n elements of an array */
double max(double a[], int n)
{
  int j;
  double m=fabs(a[0]);

  for (j=0;j<n;j++)
    if (m<fabs(a[j]))
      m=fabs(a[j]);
  return m;
}

/*****************************************************************************
 * Function: double runsets(double Ll, double Lh, double dL, double rl,
 *          double rh, double dr)
 *
 * Inputs: Ll - the lower bound of the thickness range
 *             Lh - the upper bound of the thickness range
 *             dL - the thickness step size
 *             rl - the lower bound of the recombination range
 *             rh - the upper bound of the recombination range
 *             dr - the recombination step size
 *
 * Outputs: int product, the number of runsets to simulate
 *
 * Overview: Function to create all permutations of L and r given their ranges
 *          and step sizes
 *****************************************************************************/
int runsets(double Ll, double Lh, double dL, double rl, double rh, double dr)
{
      int i, j;
      int l, R, product;
      double nL, nr;

      nL = (Lh - Ll) / dL;
	l = ((int)nL) + 1;
	printf("Number of thicknesses to simulate: %d\n", l);

	nr = (rh - rl) / dr;
	R = ((int)nr) + 1;
	printf("Number of recombination factors to simulate: %d\n", R);

	product = R*l;
	printf("Total number of simulations: %d\n", product);

	outfiles = malloc(product*sizeof(char *));
	for(i=0; i<product; i++)
	{
		outfiles[i] = (char *)malloc(24*sizeof(char));
	}
	thicknesses = malloc(product*sizeof(double));
	recombs = malloc(product*sizeof(double));
	for(i=0; i<l; i++)
	{
		for(j=0; j<R; j++)
		{
			recombs[(i*R)+j] = rl + j*dr;
			thicknesses[(i*R)+j] = Ll + i*dL;
			sprintf(outfiles[(i*R)+j],"%3.0fnm%1.2lfrec",
					thicknesses[(i*R)+j]*1e9,recombs[(i*R)+j]);
		}
	}
	return product;
}

/*****************************************************************************
 * Function: int maxpower(double[], int)
 *
 * Inputs:  P - double [] of J*V values
 *		start - int which represents the index of P to start at
 *		n - int which represents the index+1 of P to stop at
 *
 * Outputs: index - the index of the maximum power point (i.e. the most
 *	    negative power density) in the range
 *
 * Overview: compare each of the elements from index start to n-1 in P and
 *	    report the index of the most negative one
 ****************************************************************************/
int maxpower(double P[], int start, int n)
{
	int i;
	double max;
	int index = start;
	max = P[index];
	for(i=start; i<n; i++)
	{
		if(P[i] < max){
			index = i;
			max = P[index];
		}
	}
	return index;
}

/*** The following two spline interpolation routines are taken from the
C CATAM Software Library (CCATSL). These functions are freely available
```

```
under the GNU General Public License. Visit
http://www.maths.cam.ac.uk/undergrad/tripos/catam/ccatsl/
for more information ***/

void SplineCL(int n, double *x, double *y, double *b, double *c, double *d)
{
 int i;
 double t;

 /* tri-diagonal coefficients : b[] diagonal, d[] off-d, c[] RHS ... */

 d[0] = x[1] - x[0];
 if (d[0] <= 0.0) {   /*'x[i+1] not > x[i] for some i'*/

   return;
 }
 c[1] = (y[1] - y[0]) / d[0];
 for (i = 2; i < n; i++) {
   d[i - 1] = x[i] - x[i - 1];
   if (d[i - 1] <= 0.0) {   /*'x[i+1] not > x[i] for some i'*/

     return;
   }
   b[i - 1] = 2 * (d[i - 2] + d[i - 1]);
   c[i] = (y[i] - y[i - 1]) / d[i - 1];
   c[i - 1] = c[i] - c[i - 1];
 }

 /* end condition ... */

 b[0] = -d[0];
 b[n - 1] = -d[n - 2];

 if (n == 3) {
   c[0] = 0.0;
   c[n - 1] = 0.0;
 } else {
   c[0] = c[2] / (x[3] - x[1]) - c[1] / (x[2] - x[0]);
   c[n - 1] = c[n-2] / (x[n-1] - x[n-3]) - c[n - 3] / (x[n-2] - x[n-4]);
   c[0] = c[0] * d[0] * d[0] / (x[3] - x[0]);
   c[n - 1] = c[n-1] * d[n-2] * d[n-2] / (x[n-4] - x[n-1]);
 }

 /* forward elimination ... */

 for (i = 1; i < n; i++) {
   t = d[i - 1] / b[i - 1];
   b[i] -= t * d[i - 1];
   c[i] -= t * c[i - 1];
 }

 /* back substitution ... */

 c[n - 1] /= b[n - 1];
 for (i = n - 2; i >= 0; i--)
   c[i] = (c[i] - d[i] * c[i + 1]) / b[i];

 /* ... now find polynomial coefficients ... */

 b[n - 1] = (y[n - 1] - y[n - 2]) / d[n - 2] +
            d[n - 2] * (c[n - 2] + 2 * c[n - 1]);
 for (i = 0; i <= n - 2; i++) {
   b[i] = (y[i + 1] - y[i]) / d[i] - d[i] * (c[i + 1] + 2 * c[i]);
   d[i] = (c[i + 1] - c[i]) / d[i];
   c[i] = 3 * c[i];
 }
 c[n - 1] = 3 * c[n - 1];
 d[n - 1] = d[n - 2];

 search_interval = 1;

 /* no errors */

}  /*of spline*/

double SplineValCL(int n, double xx, double *x, double *y, double *b,
            double *c, double *d)
{
 double Result;
 int i, j, k;
 double t;


 /* add this to prevent C compilation warnings*/

// Result = 0.0;

 i = search_interval;
 if (i < 1 || i > n)
   i = 1;

 /* find interval ... */

 if (xx <= x[i - 1] || xx > x[i]) {
   i = 1;
   j = n;
   do {
     k = (i + j) / 2;
     if (xx < x[k - 1])
       j = k;
     else
       i = k;
   } while (j != i + 1);
 }

 t = xx - x[i - 1];
 search_interval = i;
 Result = y[i - 1] + t * (b[i - 1] + t * (c[i - 1] + t * d[i - 1]));

 /* no errors */
 return Result;
}  /*of seval*/
```

References

[1] Author Unknown, "Solar Electric Power: The U.S. Photovoltaic Industry Roadmap," Sandia National Labrotories, May 2001. [Online] Available: http://photovoltaics.sandia.gov/docs/PDF/PV_Road_Map.pdf [Accessed: June 8, 2010].

[2] Author Unknown, "First Solar Passes $1 Per Watt Industry Milestone: Company Cuts Manufacturing Cost to 98 Cents Per Watt in Fourth Quarter," First Solar News Release, Februaury 24, 2009.[Online] Available: http://phx.corporate-ir.net/phoenix.zhtml?c=201491&p=irol-newsArticle&ID=1259614 [Accessed: June 8, 2010].

[3] Chris France, "Numerical Modeling of Polymer-based Bulk Heterojunction Photovoltaics," Cal Poly 2005.

[4] Erik Everson, "Numerical Simulation of ITO/P3HT:PCBM/Au Devices to Determine Electron and Hole Mobility," Cal Poly 2006.

[5] Author Unknown, "Insolation," Wikipedia, The Free Encyclopedia. [Online] Available: http://en.wikipedia.org/wiki/Insolation. [Accessed June 8, 2010].

[6] Author Unknown, "Safety data for chlorobenzene," The Physical and Theoretical Chemistry Laboratory Oxford University, October 30, 2009. [Online] Available: http://msds.chem.ox.ac.uk/CH/chlorobenzene.html. [Accessed: January 19, 2009].

[7] Tomohisa Mori, Hiroya Takada, Shinobu Ito, Kenji Matsubayashi, Nobuhiko Miwa, Toshiko Sawaguchi, "Preclinical studies on safety of fullerene upon acute oral administration and evaluation for no mutagenesis," Toxicology, Volume 225, Issue 1, 1 August 2006, Pages 48-54. [Online]. Available: http://www.sciencedirect.com/science/article/B6TCN-4JXXR7C-1/2/8f596933ed3c37ff3ecda6c55bbe355d . [Accessed: January 14, 2009].

[8] Author Unknown, "Isopropyl alcohol MSDS," ScienceLab.com, May 22, 2009. [Online] Available: http://www.sciencelab.com/xMSDS-Isopropyl_alcohol-9924412. [Accessed: March 11, 2010].

[9] Author Unknown, "Acetone MSDS," ScienceLab.com, November 6, 2008. [Online] Available: http://www.sciencelab.com/xMSDS-Acetone-9927062. [Accessed: March 11, 2010].

[10] Author Unknown, "Consumer Energy Tax Incentives," U.S. Department of Energy, 2008. [Online] Available: http://www.energy.gov/taxbreaks.htm. [Accessed: January 14, 2009].

[11] Author Unknown, "Silicon," Wikipedia, The Free Encyclopedia. [Online] Available: http://en.wikipedia.org/wiki/Silicon#Alloys. [Accessed: March 2, 2010].

[12] Gang Li, Vishal Shrotriya, Jinsong Huang, Yan Yao, Tommoriarty, Keith Emery, Yang Yang, "High-efficiency solution processable polymer photovoltaic cells by self-organization of polymer blends," Yang Yang Laboratory. October 9, 2005. [Online] Available: http://yylab.seas.ucla.edu/papers/YY-PV-7-Slow%20Growth-NM-05.pdf. [Accessed January 11, 2010].