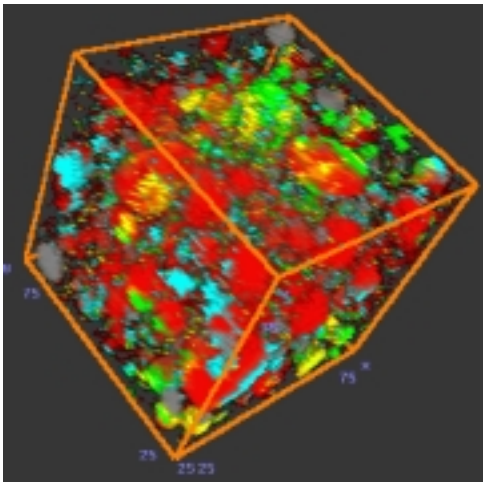
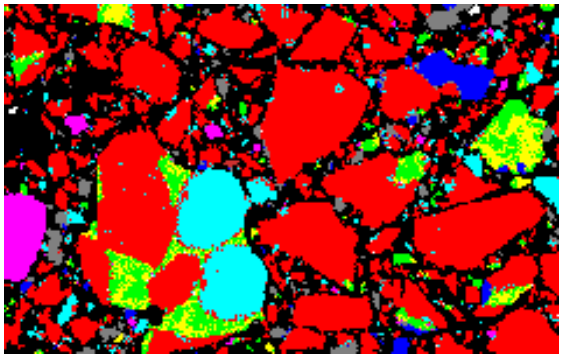
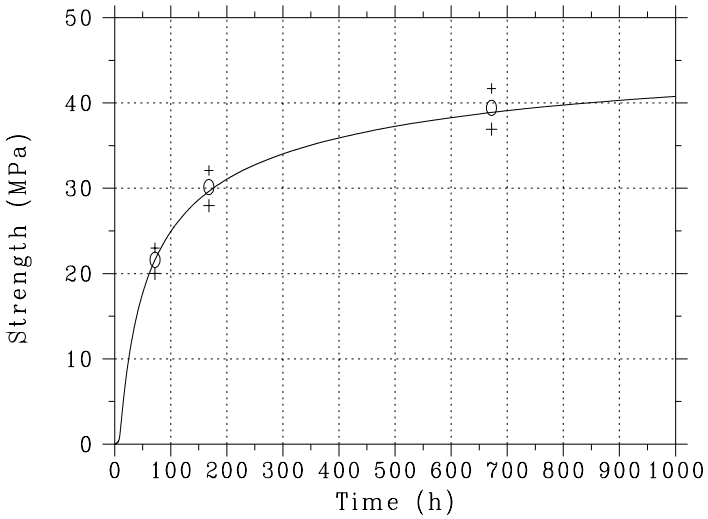
NISTIR 6485

CEMHYD3D: A Three-Dimensional Cement Hydration and Microstructure Development Modelling Package. Version 2.0

Dale P. Bentz







Building and Fire Research Laboratory
Gaithersburg, Maryland 20899

United States Department of Commerce
Technology Administration
National Institute of Standards and Technology

NISTIR 6485

CEMHYD3D: A Three-Dimensional Cement Hydration and Microstructure Development Modelling Package. Version 2.0

Dale P. Bentz

April 2000
Building and Fire Research Laboratory
National Institute of Standards and Technology
Gaithersburg, Maryland 20899

U.S. Department of Commerce
William M. Daley, *Secretary*
Technology Administration
Dr. Cheryl L. Shavers, *Under Secretary of Commerce for Technology*
National Institute of Standards and Technology
Raymond G. Kammer, *Director*

ABSTRACT

This user's manual provides updated documentation and computer program listings for version 2.0 of the three-dimensional cement hydration and microstructure development model (**CEMHYD3D**) developed at the National Institute of Standards and Technology. Originally documented and distributed in 1997, several substantial enhancements and modifications have been included in the new release. The following topics are covered in this documentation, as they were in the original version: acquisition and processing of two-dimensional scanning electron microscope and x-ray images; creation of a starting three-dimensional microstructure based on a measured particle size distribution for the cement powder and information extracted from the two-dimensional composite image; and execution of the cement hydration and microstructure development program. Additional noteworthy features of the new version of the model include: direct modelling of the induction period during cement hydration; isothermal, adiabatic, or user-programmed temperature curing conditions; a calcium silicate hydrate gel (C–S–H) whose molar stoichiometry and specific gravity (molar volume) vary with temperature; improved pozzolanic reactions between cement and silica fume (fly ash); and the incorporation of various forms of calcium sulfate (dihydrate, hemihydrate, and anhydrite) into the hydration model. In addition, a prototype internet-accessible database containing two-dimensional composite SEM images, quantitative phase analysis results, and measured cement particle size distributions is now available to provide key input information for the three-dimensional modelling process. Documented example datafiles are provided along with the examples given in the guide. Complete program listings are provided in the appendices. This manual and all of the computer programs it describes are freely available for downloading via anonymous ftp from NIST.

Keywords: Building technology, cement hydration, compressive strength, computer modelling, correlation, heat of hydration, image processing, microstructure, simulation.

Contents

List of Figures

List of Tables

1 Introduction

A revised package of computer programs for simulating cement hydration and cement paste microstructure development in **three dimensions** has been developed. A three-dimensional representation of microstructure is necessary for the computation of percolation and physical properties for comparison to experiment. While previous publications have focused on the validation of the computer model [1] and various extensions [2, 3], the purpose of this report is to provide a detailed documentation of the latest version of the computer codes so that other researchers may employ them in studying problems specific to their own interests. It is strongly suggested that potential users of these models also obtain copies of references [1, 2, 3] and the original user's manual [4] to obtain a better understanding of the technical basis underlying the computer programs documented herein. A NISTIR and a Ph. D. thesis dealing specifically with the incorporation of fly ash into the NIST 3-D cement hydration model are also available [5, 6]. The recent use of the model for simulating the microstructure and durability of high-performance cement matrices is described in another Ph. D. thesis [7].

The steps required for going from a two-dimensional image to a simulation of three-dimensional microstructure development, along with the corresponding computer program names, are provided in Table 1. The first two of these steps can now be replaced by accessing the cement images database available over the internet at
http://ciks.cbt.nist.gov/bentz/phpct/database/images. All of these programs will be described in detail in sections 2 through 5 of this user's manual. The computer codes are available via anonymous ftp as part of the Computer Integrated Knowledge System for High-Performance Concrete (HYPERCON) [8] being developed in the Building and Fire Research Laboratory at the National Institute of Standards and Technology. They may be accessed in the
pub/bfrl/bentz/CEMHYD3D/version20 subdirectory from ftp.nist.gov (129.6.13.25), by logging in as user "anonymous" and providing your e-mail address as the password. A postscript version of this manual is also available at ftp.nist.gov in the
pub/bfrl/bentz/CEMHYD3D/version20/manual subdirectory .

2 Two-dimensional Imaging of Cement Particles

A detailed description of the use of two-dimensional SEM/X-ray images as a starting point for modelling cement hydration and microstructure development can be found in the original version of this user's guide [4]. The analysis is based on the determination of phase area fractions, surface area fractions, and autocorrelation functions as described previously [4]. Because these programs (statsimp, corrcalc, and corrxy2r) have not changed from their original versions, they will not be further documented in this update.

2.1 Version 2.0 Update

A prototype database of 14 cements has been established to supercede this two-dimensional imaging process. The cements are available at
http://ciks.cbt.nist.gov/bentz/phpct/database/images. From the main selection form, the user simply selects their cement of interest by clicking on the "radio button" immediately to the left of each description. When this form is submitted, a color-coded 2-D image of the

Table 1: Steps in Execution of Three-Dimensional Cement Microstructure Model

Step	Programs
Acquire and process two-dimensional image	statsimp.c
Determine autocorrelation for	
$C_3S + C_2S$	corrcalc.c
C_3S	corrxy2r.c
C_3A or C_4AF	
Generate 3-D particle image	genpartnew.c
Distribute phases in 3-D image for	
silicates/aluminates	rand3d.f or rand3d.c
C_3S/C_2S	stat3d.c
C_3A/C_4AF	sinter3d.c
Execute hydration model	disrealnew.c

cement will be returned along with information on the quantitative phase analysis of the 2-D image(s) and a measured cement particle size distribution (PSD) for the cement. This information can be used to generate a three-dimensional starting microstructure as described in the section which follows. An example image for Cement 133 issued by the Cement and Concrete Reference Laboratory (CCRL) of NIST in June of 1999 [9] is shown in Fig. 1. The accompanying quantiative analysis information is provided in Fig. 2, with the measured and discretized cement PSD provided in Table 2.

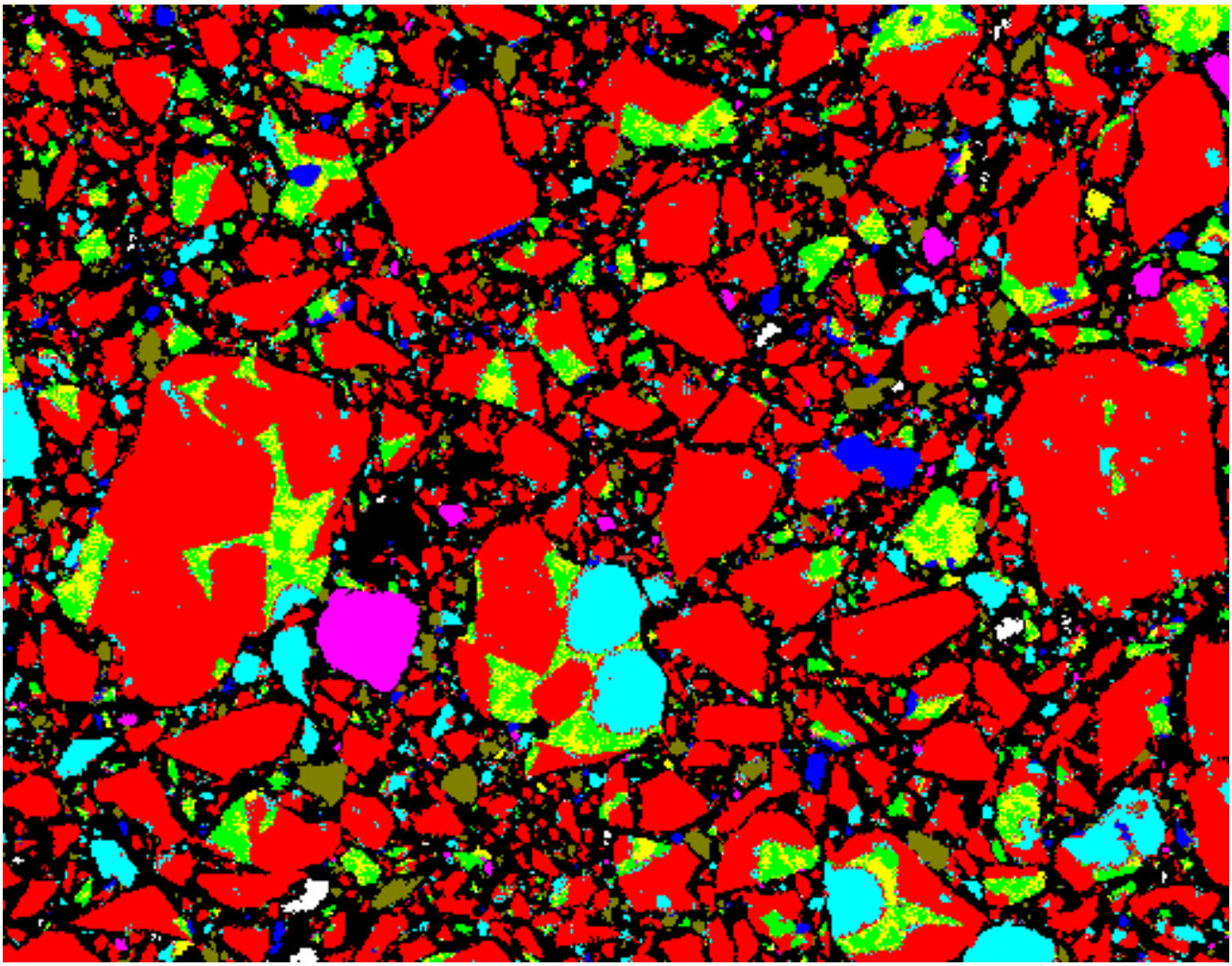


Figure 1: Two-dimensional processed SEM/X-ray image for cement 133 issued by the CCRL (NIST) in June of 1999. Color assignments are: red- C_3S, aqua- C_2S, green- C_3A, yellow- C_4AF, pale green- gypsum, white- free lime (CaO), dark blue (purple)- K_2SO_4, light magenta- periclase (magnesium containing phase). Image is 256 μm $\times$ 200 μm.

Information on CCRL Cement 133

Image and correlation files for CCRL Cement 133, a Type I/II ordinary portland cement with a Blaine fineness of about 350 m^2/kg

Color 2D image in cement133.gif (500X magnification- 256 µm by 200 µm) Red is C_3S, aqua is C_2S, green is C_3A, yellow is C_4AF, pale green is gypsum, white is free lime, dark blue is potassium sulfate, magenta is a MgCa phase, and burnt red is kaolin

Discretized particle size distribution is in cement133.psd

Extracted Correlation files (1 µm/pixel):
cm133r.sil --- C_3S and C_2S
cm133r.c3s --- C_3S
cm133r.c4f --- C_4AF

Phase Fractions for four major clinker phases:

PHASE	AREA	PERIMETER (SURFACE)
C_3S	0.7018	0.6491
C_2S	0.1315	0.1764
C_3A	0.0827	0.1138
C_4AF	0.0840	0.0607

Overall phase fractions (average of two images):

PHASE	AREA
C_3S	0.6216
C_2S	0.1168
C_3A	0.0735
C_4AF	0.0745
Gypsum	0.0550
Free lime	0.0232
Alkali sulfates	0.0191
Periclase	0.0151
'Kaolin'	0.00116

Gypsum typically added as 5.4% on a volume basis.

Use the back button on your Web browser to return to the cement image.

Figure 2: Description of the quantitative analysis for cement 133 issued by the CCRL (NIST) in June of 1999.

Table 2: Discretized Cement Particle Size Distribution for CCRL Cement 133

Diameter (μm)	Mass fraction
1.0	0.1348
3.0	0.1303
5.0	0.09456
7.0	0.07349
9.0	0.06085
11.0	0.0507
13.0	0.04221
15.0	0.03775
17.0	0.033935
19.0	0.028
21.0	0.02565
23.0	0.0233
25.0	0.02115
27.0	0.019
29.0	0.019
31.0	0.0228
35.0	0.0347
41.0	0.02658
47.0	0.0454
61.0	0.0358
73.0	0.040025

3 Two-dimensional to Three-dimensional Conversion

3.1 Generation of spherical particles following measured PSD

The program `genpartnew.c`, whose listing is provided in Appendix B, is used to place digitized spherical particles of a user specified PSD into a three-dimensional computational volume, typically 100 pixels on a side. Periodic boundaries are employed such that a particle that extends outward through one or more faces of the 3-D microstructure is completed extending inward through the opposite face(s). Digitized spherical particles are used to approximate the complex three-dimensional shapes of actual cement particles; previous results have indicated that this approximation is adequate if the actual cement PSD and phase volume and surface fractions are maintained in the 3-D spherical particle image [1, 2, 10]. Initially, the user must provide a negative integer to be used as the random number seed. Following this, the program is menu driven with the following main menu options:

1) Exit: Exit the program

2) Add spherical particles (cement, mineral admixtures, and calcium sulfate) to microstructure: allows the user to specify the discretized particle size distribution that should be used, the number of particles to place, the proportion of particles that should

be calcium sulfate (gypsum, hemihydrate, and/or anhydrite) as opposed to cement, and whether the particles should be dispersed. The particle size distribution should be on a number basis. Since most particle size analyzers provide the distribution on a mass basis (see Table 2 for example), a mass to number basis conversion (easily implemented in a spreadsheet) may be required. An example Excel spreadsheet is provided in Appendix A of this manual and is available for downloading from the anonymous ftp site as `cem133.xls`. For `genpartnew`, as input, the user must specifically supply the number of different size spheres to use (program parameter `NUMSIZES` determines the maximum number of different sizes allowed), a dispersion factor (0, 1, or 2) that specifies the minimum pixel separation to be maintained between all pairs of particles, a probability (0.0 to 1.0) for the generation of calcium sulfate particles instead of cement, and probabilities (0.0 to 1.0) for the hemihydrate and anhydrite versions of calcium sulfate (as opposed to gypsum/dihydrate). Following this, the user, for each different size class of spheres, provides the number, radius (size), and phase identifier characterizing that size class. The user should always begin with the largest particles and proceed consecutively to the smallest particles. Otherwise, after placing some of the smallest particles, no spaces where the largest particles can fit may remain in the 3-D microstructure. The dispersion capability has been included to simulate the effects of adding a superplasticizer or high range water reducer to the cement paste. However, it should be noted that for lower water-to-cement ratios (w/c < 0.45), it may not be possible to place all of the requested particles in a dispersed configuration. In this case, the program will exit after parameter `MAXTRIES` unsuccessful attempts at finding a random location for a particle. Table 3 provides a listing of the number of pixels contained in spheres of various diameters in pixels, that should prove useful in creating user specific PSDs. It should be noted that the $diameter = 2 * radius + 1$, so that all spheres may be centered exactly on a pixel. One pixel (or 1 μm) particles are typically added in the program `disrealnew`, just prior to executing the hydration, to significantly reduce the memory needed to execute the `genpartnew` program.

3) Flocculate system by reducing number of particle clusters: allows the user to create any desired number of flocs (clusters) by randomly moving each particle (cluster) centroid in one-pixel increments and aggregating any particles (clusters) that contact one another during this process. The only input required is the user requested number of clusters (flocs) to be present at the end of execution of the routine. Typically, if no superplasticizer or water reducing agent is used, the cement particles will have a great tendency to flocculate together [11], perhaps into a single floc structure.

4) Measure phase fractions: outputs the number of pixels of porosity, cement, the three forms of calcium sulfate, pozzolan (silica fume), inert filler, fly ash, and aggregate present in the 3-D microstructure. This selection is useful for checking the calcium sulfate volume fraction and the initial w/c ratio of the 3-D cement microstructure (neglecting the one-pixel particles that will be added at `disrealnew` execution time).

5) Add an aggregate to the microstructure: allows the user to add a single flat plate aggregate to the 3-D microstructure for studying the development of microstructure in the interfacial transition zone [12]. The only required input is the thickness of the aggregate to be placed, which must be an even integer. Typically, the user should place an aggregate (if desired) into the microstructure before placing any of the cement

Table 3: Volume in Pixels Occupied by Spheres of Various Diameters

Diameter (pixels)	Pixels per sphere
3	19
5	81
7	179
9	389
11	739
13	1189
15	1791
17	2553
19	3695
21	4945
23	6403
25	8217
27	10395
29	12893
31	15515
33	18853
35	22575
37	26745
39	31103
41	36137
43	41851
45	47833
47	54435
49	61565
51	69599
61	119009
73	203965

particles. Otherwise, the aggregate will simply overlap and replace any solid particle pixels contained within its boundaries.

6) Measure single phase connectivity: allows the user to employ a burning algorithm [13] to determine the percolation characteristics of either the porosity or the solids present in the 3-D microstructure. The user must specify the phase in which they are interested and the routine returns the number of pixels of that phase which are accessible from the top of the 3-D microstructure along with the number of pixels that are contained in pathways that traverse (span) the microstructure.

7) Measure phase fractions vs. distance from aggregate: outputs a listing of the number of pixels of each phase (cement, gypsum, porosity, etc.) present in parallel planes at various fixed distances (one pixel increments) from the aggregate surface. When this menu selection is executed, the results are reported to the standard output (screen)

and are also written in a datafile named `agglist.out`. If the user wishes to preserve this output file, they must rename it to a name of their choosing immediately upon leaving the `genpartnew` program.

8) Output microstructure to file: allows the user to save the created microstructure to files. The user must supply two filenames, one for the storage of the actual microstructure (cement, calcium sulfates, fillers, and porosity), and the second for storage of the individual particle IDs (to be used in assessing the setting of the cement during hydration using the `disrealnew` program).

When adding calcium sulfate to the cement, the user has basically two choices. If only the composite PSD for the cement and gypsum (sulfate) is known, the user can simply specify the volume fraction of particles that should be randomly assigned to be calcium sulfate. Conversely, if the actual separate PSD of the calcium sulfate is known, the user can use a value of 0.0 for this randomly-assigned volume fraction and specify the actual numbers of each size calcium sulfate particle to be placed, in a manner analogous to that used for cement. When the separate cement and calcium sulfate PSDs are "merged" in this fashion, the user should be sure to add all the largest radius particles (e.g., first the cement, then the calcium sulfate) first before proceeding to the next smaller radius particles. If this second option is to be utilised, in the program `genpartnew`, a phase ID of 1 corresponds to cement, 5 to calcium sulfate (dihydrate), 6 to hemihydrate, and 7 to anhydrite, as shown in the program listing in Appendix B.

An annotated example datafile for using `genpartnew` (with random calcium sulfate assignment) is as follows:

```
-3402                   random number seed
2                       menu choice to place spherical particles
16                      number of size classes to place
0                       dispersion distance in pixels
0.055                   calcium sulfate volume fraction
0.0 0.0          fractions of calcium sulfate that are hemihydrate and anhydrite
1                       number of spheres of size class 1
17                      radius of spheres of size class 1
1                       phase ID of spheres of size class 1 (cement=1)
1                       number of spheres of size class 2
15                      radius of spheres of size class 2
1                       phase ID of spheres of size class 2 (cement)
1                       number of spheres of size class 3
14                      radius of spheres of size class 3
1                       phase ID of spheres of size class 3 (cement)
1                       number of spheres of size class 4
13                      radius of spheres of size class 4
1                       phase ID of spheres of size class 4 (cement)
2                       number of spheres of size class 5
12                      radius of spheres of size class 5
1                       phase ID of spheres of size class 5 (cement)
2                       number of spheres of size class 6
11                      radius of spheres of size class 6
```

```
1                       phase ID of spheres of size class 6 (cement)
3                       number of spheres of size class 7
10                      radius of spheres of size class 7
1                       phase ID of spheres of size class 7 (cement)
4                       number of spheres of size class 8
9                       radius of spheres of size class 8
1                       phase ID of spheres of size class 8 (cement)
8                       number of spheres of size class 9
8                       radius of spheres of size class 9
1                       phase ID of spheres of size class 9 (cement)
12                      number of spheres of size class 10
7                       radius of spheres of size class 10
1                       phase ID of spheres of size class 10 (cement)
21                      number of spheres of size class 11
6                       radius of spheres of size class 11
1                       phase ID of spheres of size class 11 (cement)
41                      number of spheres of size class 12
5                       radius of spheres of size class 12
1                       phase ID of spheres of size class 12 (cement)
93                      number of spheres of size class 13
4                       radius of spheres of size class 13
1                       phase ID of spheres of size class 13 (cement)
243                     number of spheres of size class 14
3                       radius of spheres of size class 14
1                       phase ID of spheres of size class 14 (cement)
692                     number of spheres of size class 15
2                       radius of spheres of size class 15
1                       phase ID of spheres of size class 15 (cement)
4063                    number of spheres of size class 16
1                       radius of spheres of size class 16
1                       phase ID of spheres of size class 16 (cement)
4                       menu selection to report phase counts
8                       menu selection to output current microstructure to file
cem133wc030n1.img       filename to save image to
pcem133wc030n1.img      filename to save particle IDs to
1                       menu selection to end program
```

The output created by executing the program genpartnew with the above input datafile is as follows:

```
Enter random number seed value (a negative integer)
-3402

 Input User Choice
1) Exit
2) Add spherical particles (cement,gypsum, pozzolans, etc.) to microstructure
3) Flocculate system by reducing number of particle clusters
4) Measure global phase fractions
```

```
5) Add an aggregate to the microstructure
6) Measure single phase connectivity (pores or solids)
7) Measure phase fractions vs. distance from aggregate surface
8) Output current microstructure to file
2
Enter number of different size spheres to use(max. is 30)
16
Enter dispersion factor (separation distance in pixels) for spheres (0-2)
0 corresponds to totally random placement
0
Enter probability for gypsum particles on a random particle basis (0.0-1.0)
0.055000
Enter probabilities for hemihydrate and anhydrite forms of gypsum (0.0-1.0)
0.000000 0.000000
Enter number, radius, and phase ID for each sphere class (largest radius 1st)
Phases are 1- Cement and (random) calcium sulfate, 5- Gypsum, 6- hemihydrate
7- anhydrite 8- Pozzolanic, 9- Inert, 25- Fly Ash
Enter number of spheres of class 1
1
Enter radius of spheres of class 1
(Integer <=25 please)
17
Enter phase of spheres of class 1
1
Enter number of spheres of class 2
1
Enter radius of spheres of class 2
(Integer <=25 please)
15
Enter phase of spheres of class 2
1
Enter number of spheres of class 3
1
Enter radius of spheres of class 3
(Integer <=25 please)
14
Enter phase of spheres of class 3
1
Enter number of spheres of class 4
1
Enter radius of spheres of class 4
(Integer <=25 please)
13
Enter phase of spheres of class 4
1
Enter number of spheres of class 5
2
```

```
Enter radius of spheres of class 5
(Integer <=25 please)
12
Enter phase of spheres of class 5
1
Enter number of spheres of class 6
2
Enter radius of spheres of class 6
(Integer <=25 please)
11
Enter phase of spheres of class 6
1
Enter number of spheres of class 7
3
Enter radius of spheres of class 7
(Integer <=25 please)
10
Enter phase of spheres of class 7
1
Enter number of spheres of class 8
4
Enter radius of spheres of class 8
(Integer <=25 please)
9
Enter phase of spheres of class 8
1
Enter number of spheres of class 9
8
Enter radius of spheres of class 9
(Integer <=25 please)
8
Enter phase of spheres of class 9
1
Enter number of spheres of class 10
12
Enter radius of spheres of class 10
(Integer <=25 please)
7
Enter phase of spheres of class 10
1
Enter number of spheres of class 11
21
Enter radius of spheres of class 11
(Integer <=25 please)
6
Enter phase of spheres of class 11
1
```

```
Enter number of spheres of class 12
41
Enter radius of spheres of class 12
(Integer <=25 please)
5
Enter phase of spheres of class 12
1
Enter number of spheres of class 13
93
Enter radius of spheres of class 13
(Integer <=25 please)
4
Enter phase of spheres of class 13
1
Enter number of spheres of class 14
243
Enter radius of spheres of class 14
(Integer <=25 please)
3
Enter phase of spheres of class 14
1
Enter number of spheres of class 15
692
Enter radius of spheres of class 15
(Integer <=25 please)
2
Enter phase of spheres of class 15
1
Enter number of spheres of class 16
4063
Enter radius of spheres of class 16
(Integer <=25 please)
1
Enter phase of spheres of class 16
1

 Input User Choice
1) Exit
2) Add spherical particles (cement,gypsum, pozzolans, etc.) to microstructure
3) Flocculate system by reducing number of particle clusters
4) Measure global phase fractions
5) Add an aggregate to the microstructure
6) Measure single phase connectivity (pores or solids)
7) Measure phase fractions vs. distance from aggregate surface
8) Output current microstructure to file
4
```

```
 Phase counts are:
Porosity= 569660
Cement= 406682
Gypsum= 23658
Anhydrite= 0
Hemihydrate= 0
Pozzolan= 0
Inert= 0
Fly Ash= 0
Aggregate= 0

 Input User Choice
1) Exit
2) Add spherical particles (cement,gypsum, pozzolans, etc.) to microstructure
3) Flocculate system by reducing number of particle clusters
4) Measure global phase fractions
5) Add an aggregate to the microstructure
6) Measure single phase connectivity (pores or solids)
7) Measure phase fractions vs. distance from aggregate surface
8) Output current microstructure to file
8
Enter name of file to save microstructure to
cem133wc030n1.img
Enter name of file to save particle IDs to
pcem133wc030n1.img

 Input User Choice
1) Exit
2) Add spherical particles (cement,gypsum, pozzolans, etc.) to microstructure
3) Flocculate system by reducing number of particle clusters
4) Measure global phase fractions
5) Add an aggregate to the microstructure
6) Measure single phase connectivity (pores or solids)
7) Measure phase fractions vs. distance from aggregate surface
8) Output current microstructure to file
1
```

The user can easily calculate that the achieved calcium sulfate volume fraction (23658)/(23658+406682)=0.054975 is very close to the requested value of 0.055. A two-dimensional (postscript) image slice from the microstructure created using this datafile with `genpartnew` is shown in the left side of Fig. 3. The 2-D raw image file (ppm format) was created using the program `oneimage.c`, whose listing is also provided in Appendix B. The file created by this program can be easily read into an imaging program, resized, and output in a variety of file formats (gif, jpeg, postscript, etc.).

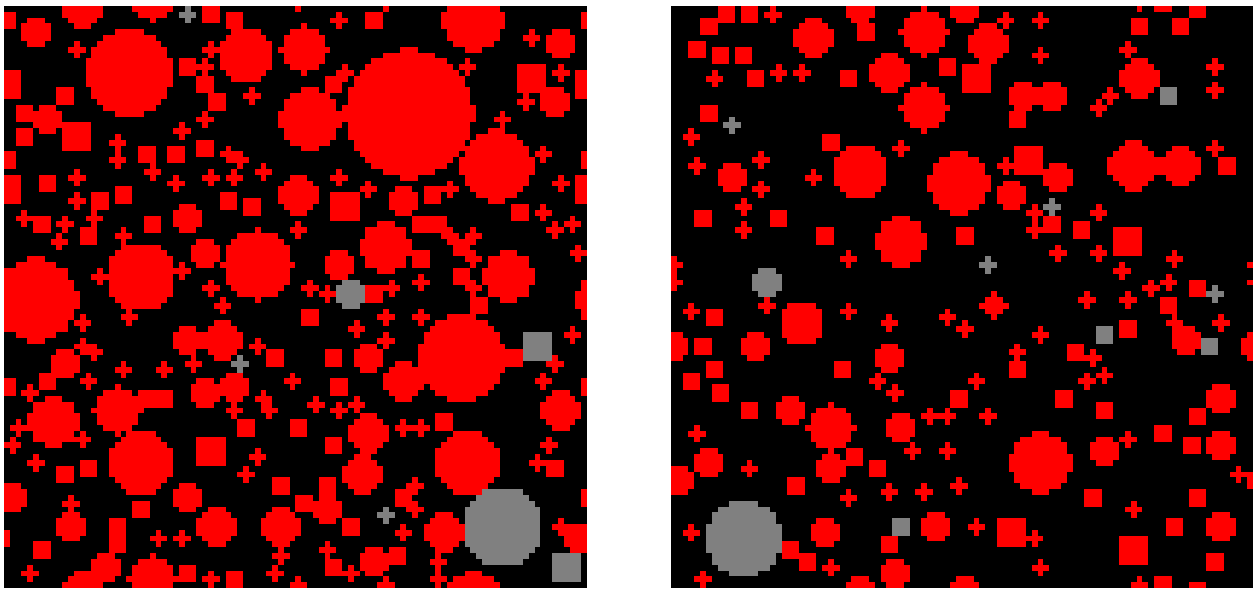


Figure 3: Two-dimensional slices from 3-D microstructures for cement 133 with w/c ratios of 0.30 (left) and 0.45 (right). At this point, only particles greater than 2 pixels (μm) in diameter have been placed in these microstructures. Color assignments are black- porosity, red- cement, and grey- gypsum. Gypsum volume fraction is approximately 5.5%. Images are 100 pixels $\times$ 100 pixels.

3.2 Filtering of random noise particle image

Once a 3-D image incorporating the desired PSD has been created, the next step is to introduce the appropriate phase volume fractions, phase surface area fractions, and correlation structure into the initially monophase cement (or fly ash) particles. For fly ash particles, two specialized programs for randomly distributing the fly ash phases on either a particle or pixel basis are described in reference [5]. Alternatively, the procedure described below for distributing phases amongst the cement particles could be employed for fly ash particles as well, as long as the spatial statistics (correlations, volume and surface fractions, etc.) for the fly ash particles had been previously determined.

For cement, the phase distribution process is accomplished in a series of steps using one Fortran (or C) program and two programs available only in C. The program available in Fortran or C, `rand3d.f` or `rand3d.c` (listings provided in Appendix B), is used to introduce the correct phase volume fraction and the correlation structure measured on the 2-D SEM image (Fig. 1) into the 3-D cement particle image. Starting with an image of random Gaussian noise, generated using the Box-Muller method [14], the measured autocorrelation function for the phase(s) of interest is used to filter the image, introducing the appropriate correlation structure. This correlation structure is then basically overlaid on the cement particle image to introduce the correlation structure and phase fractions into individual cement particles. Each time this program is executed, the user must specify what phase is to be subdivided into two phases and the value to be assigned to the new phase. A typical sequence is to separate the cement into silicates and aluminates, separate the silicates into C_3S[1] and C_2S, and separate the aluminates into C_3A and C_4AF, as illustrated in Fig. 4.

[1]Conventional cement chemistry notation is used throughout this manual: $C = CaO$, $S = SiO_2$, $H =$

To use `rand3d`, the user must input:

a random number seed (negative integer),

the initial phase ID to be subdivided into two phases,

the phase ID for the second (new) phase,

the name of the file containing the current 3-D microstructure to be processed,

the name of the file containing the correlation data for the phase(s) of interest,

the phase volume fraction (0.0 to 1.0) of the initial phase that is to retain its original phase value, and

the name of the file to be created to store the new resultant microstructure (file will contain one pixel/integer per line).

The input for the phase volume fraction can be determined from the 2-D SEM image or from a conventional Bogue analysis of the cement oxide composition [15], being sure to convert from a mass to a volume basis in the latter case. To facilitate this conversion, the specific gravities of the major four clinker phases [15] are provided in Table 4. Due to a few coding changes from the previously available version of `rand3d.f`, the new version will require only 10% to 50% of the CPU time required by the original version to execute a single filtering and phase assignment pass, a significant improvement. The C version is in general 3 to 4 times faster than the Fortran version.

Table 4: Specific Gravities of Major Clinker Phases

Phase	Specific Gravity
C_3S	3.21
C_2S	3.28
C_3A	3.03
C_4AF	3.73

The filtering process will be illustrated for cement 133 for the microstructure shown in the left side of Fig. 3. The input datafile to filter the image and separate the cement into silicates and aluminates (for the Fortran version of the code) is as follows:

```
-1088                  negative integer random number seed
1.0                    original phase to be segmented and reassigned (cement)
4.0                    new phase ID to be assigned to modified pixels
'cem133wc030n1.img'    filename of input 3-D microstructure
'cem133r.sil'          file containing 1-D correlation function for silicates
0.8333                 phase fraction (0.0-1.0) to remain as silicates
'cem133wc030n1a.img'   filename of output 3-D microstructure
```

H_2O, $A = Al_2O_3$, $F = Fe_2O_3$, and $\bar{S} = SO_3$

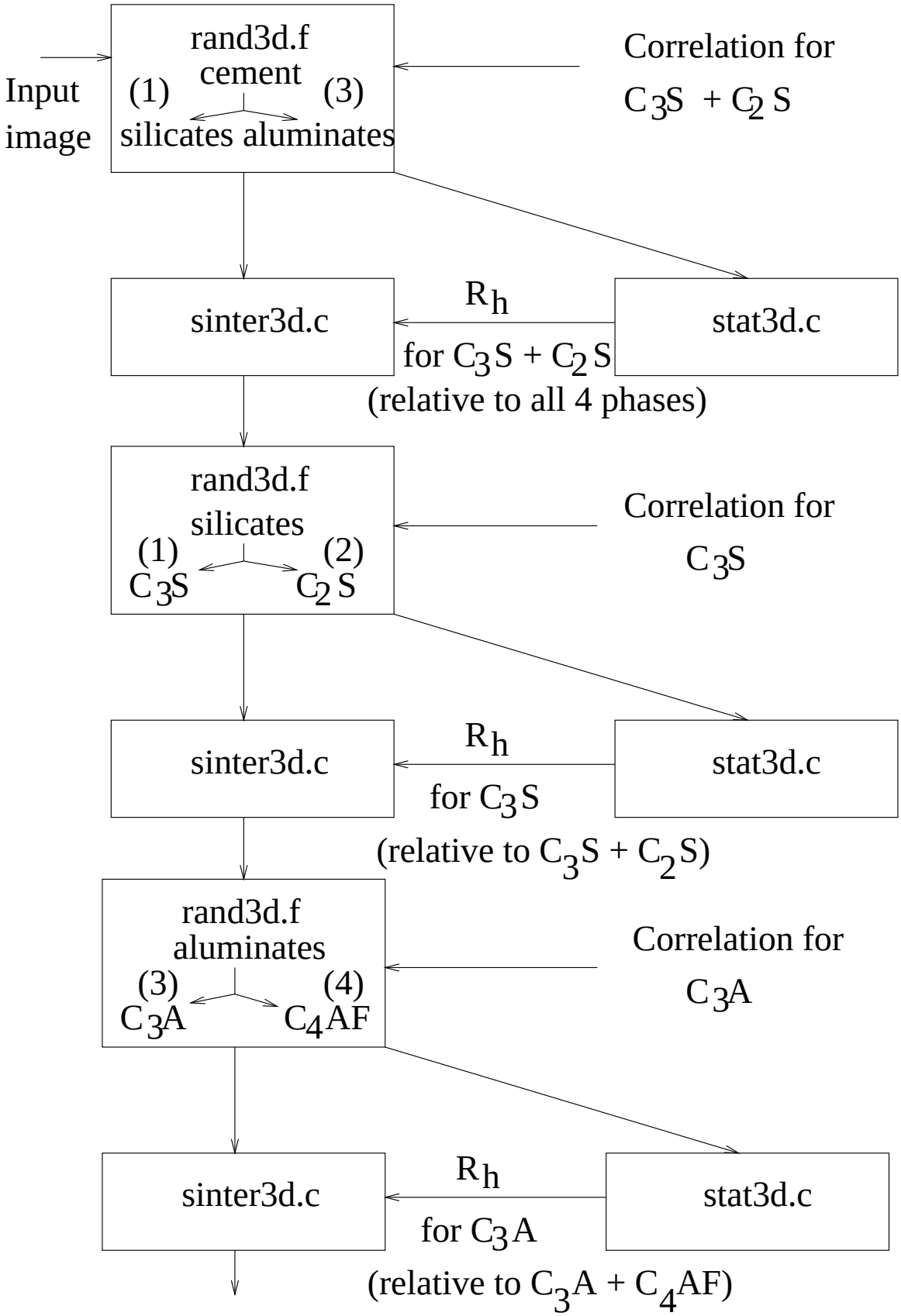


Figure 4: Flowchart specifying filtering algorithm for assigning pixel phase values to the three-dimensional starting cement microstructure image. Numbers in parentheses indicate phase IDs corresponding to the individual cement compounds.

The value of 0.8333 was chosen for the silicates volume fraction based on the sum of the measured area fractions for C_3S (0.7018) and for C_2S (0.1315) provided in Fig. 2, obtained from analysis of the composite SEM image shown in Fig. 1. The file `cem133r.sil` (along with `cem133r.c3s` and `cem133r.c4f`) was downloaded from the cement images database described earlier. The value of 4.0 was chosen for the new phase assignment, because for this cement, the 3rd correlation file was determined for the C_4AF phase (and not C_3A). This means that the aluminates will subsequently be split into C_4AF (phase ID 4) and C_3A (phase ID 3). By assigning the aluminates an initial phase value of 4.0, simply executing `rand3d` again with an original phase ID of 4 and a new phase ID of 3 will conveniently and directly create the C_4AF and C_3A phases. If the correlation file had been determined for the C_3A instead of the C_4AF, one would simply use a phase ID value of 3.0 instead of 4.0 on line three of the datafile shown above, as illustrated in the flow diagram in Fig. 4. Finally, it should be noted that the quotation marks around the filenames are only needed when executing the Fortran version of `rand3d` and must be omitted when using the C version.

A 2-D slice from the resultant 3-D microstructure (e.g., `cem133wc030n1a.img`) obtained after the execution of `rand3d` is shown in the left side of Fig. 5. It can be clearly observed that the cement particles have been segmented into silicates and aluminates, as requested.

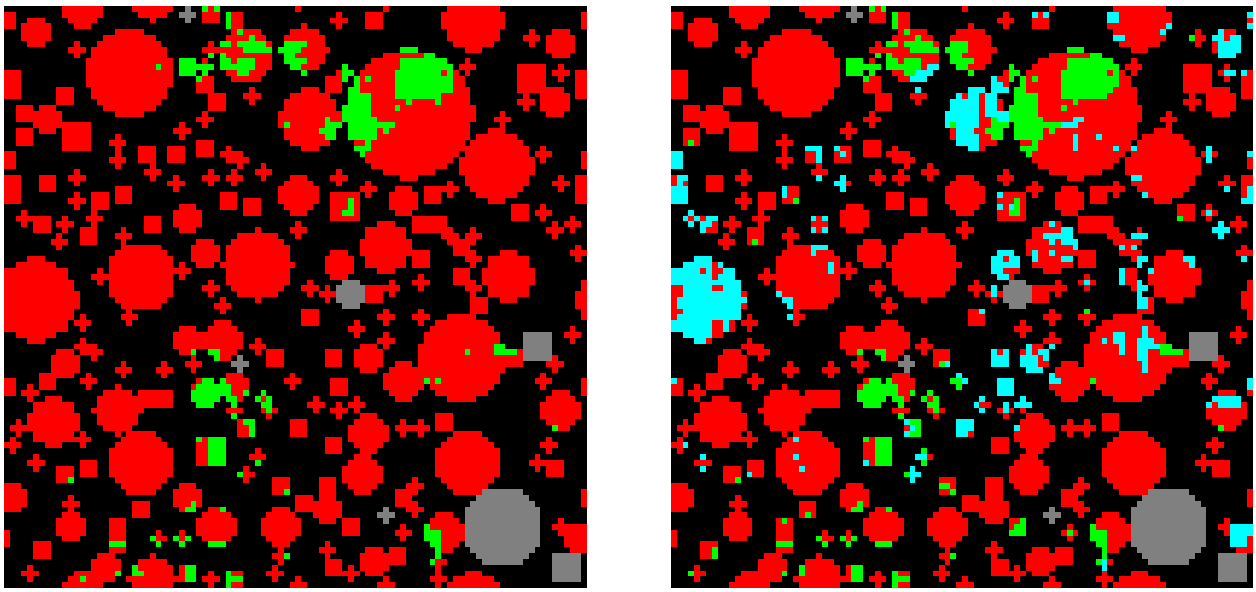


Figure 5: Two-dimensional slices from 3-D microstructures for cement 133 segmented into silicates and aluminates (left) and further segmented into C_3S, C_2S, and aluminates, with w/c=0.30. Color assignments are black- porosity, red- silicates (C_3S), aqua- C_2S, green- aluminates, and grey- gypsum. Images are 100 pixels $\times$ 100 pixels.

3.3 Correction of hydraulic radius

Because the correlation structure of the 3-D image produced by `rand3d` only approximates that of the input 2-D correlation function, a modification is utilized to directly match the surface area fraction (via the hydraulic radius) of the 3-D image to its 2-D counterpart, greatly improving the agreement between the 2-D and 3-D correlations [16]. Typically, the microstructure file produced by `rand3d` is first analyzed using the program `stat3d.c`. This

C program (listing provided in Appendix B), will determine the phase volume fractions and surface area fractions for each phase present in the 3-D microstructure. The user simply inputs the filename of the 3-D image to be quantified and the name of a file in which to store the results of the analysis. The surface area fractions are reported on a calcium sulfate-free basis, so the user must be sure that the 2-D image has also been analyzed on a sulfate-free basis (i.e., considering only the C_3S, C_2S, C_3A, and C_4AF, as in the first table in Fig. 2). The needed surface area count for a phase can be determined by multiplying the total surface area count by the surface (perimeter) fraction determined in analyzing the 2-D SEM image, as provided in Fig. 2. From this, the appropriate value of the hydraulic radius to be used in the execution of the program `sinter3d`, R_h, can be determined as:

$$R_h = \frac{6}{4} \times \frac{phase(s)\ volume\ in\ 3-D\ image\ in\ pixels}{needed\ surface\ area\ in\ pixels}. \tag{1}$$

The value of 6/4 is included in the equation to correct for the fact that a digitized 3-D sphere has a surface area of approximately $6\pi r^2$ as opposed to $4\pi r^2$ [16].

For example, the following output is obtained when executing the program `stat3d` to quantify the phases present in the 3-D microstructure contained in the image file `cem133wc030n1a.img`.

Phase ID	Volume count	Surface count	Volume fraction	Surface fraction
0	569660	0		
1	338509	380581	0.83237	0.84138
2	0	0	0.00000	0.00000
3	0	0	0.00000	0.00000
4	68173	71747	0.16763	0.15862
Total	406682	452328		
5	23658	23382		
6	0	0		
7	0	0		
8	0	0		
9	0	0		
24	0	0		
25	0	0		

Here, one can observe that the surface fraction of silicates (phase 1), 0.84138, is slightly greater than the surface fraction measured on the silicates in the real 2-D composite SEM image (Fig. 1, 0.8255 (the sum of the values for C_3S and C_2S for PERIMETER (SURFACE) in Fig. 2). The requisite hydraulic radius is thus calculated as $6. \times 338509./4./(0.8255 \times (380581 + 71747)) = 1.36$. For the second pass through the filtering process, to segment the silicates into C_3S and C_2S, the equation for R_h would be:

$$\frac{6 \times volume\ C_3S\ in\ 3-D\ image\ in\ pixels}{4 \times needed\ surface\ area\ fraction\ of\ C_3S \times (surface\ pixels\ C_3S\ +\ surface\ pixels\ C_2S)} \tag{2}$$

Note that in this case, the needed surface area fraction of C_3S is the ratio of the PERIMETER fraction for C_3S over the sum of the PERIMETERs for C_3S and C_2S. From Fig. 2, for example, this would be 0.6491/(0.6491+0.1764)=0.7863.

This hydraulic radius is input into the program `sinter3d.c` (listing provided in Appendix B), which interchanges pixels of two specific phases to obtain the specified hydraulic radius for the first phase. This code was originally developed to simulate the sintering of a ceramic powder by exchanging solid pixels of high curvature with porosity pixels of low curvature [17]. The program is menu driven and the user must first read in the microstructure previously output by execution of `rand3d`, using menu selection 2. After this, the sintering algorithm (menu selection 4) can be executed to adjust the hydraulic radius. For this algorithm, the following inputs are required:

the two phase IDs between which to execute the sintering algorithm; the first phase indicates the one whose hydraulic radius will be increased,

the number of pixels of each phase to exchange in each cycle of the sintering algorithm (default value=200),

the calculated target hydraulic radius (from equation 1), and

the radius of the digitized sphere [17] to be used in assessing curvature (default value=3).

For our example, these parameters would take the values:

```
1 4          (silicates and aluminates)
200
1.36
3
```

The sintering algorithm will be iteratively executed until the desired hydraulic radius is achieved or an equilibrium is reached. At this point, the user may elect to output the resulting microstructure to a file using menu selection 5. A file named by the user will be created for this output (for example `cem133wc030n1b.img`). 2-D slices from the 3-D microstructures obtained after "sintering" are provided in Fig. 6.

It should be noted that the sintering can only be used to increase the hydraulic radius of a phase, by decreasing its surface area fraction at constant volume fraction. If the desired hydraulic radius for a phase is less than the value present after executing `rand3d`, the user should simply "reverse" the phases being sintered. For example, in the example outlined above, while the hydraulic radius of phase 1 is being increased by the sintering, that of phase 4 is being decreased. Thus, one can also decrease the hydraulic radius of phase 1 by increasing the hydraulic radius of phase 4. In this case, the first line of input would be 4 1 (instead of 1 4) and the requested hydraulic radius would be the calculated desired value for phase 4 (instead of phase 1).

After the sintering is used to correct the hydraulic radius of the silicates, the rand3d/stat3d/sinter3d sequence is repeated to separate the silicates into C_3S and C_2S. In this case, the input datafile for `rand3d` would be:

```
-188                    negative integer random number seed
1.0                     original phase to be segmented and reassigned (silicates)
2.0                     new phase ID to be assigned to modified pixels (C2S)
'cem133wc030n1b.img'    filename of input 3-D microstructure
'cem133r.c3s'           file containing 1-D correlation function for C3S
```

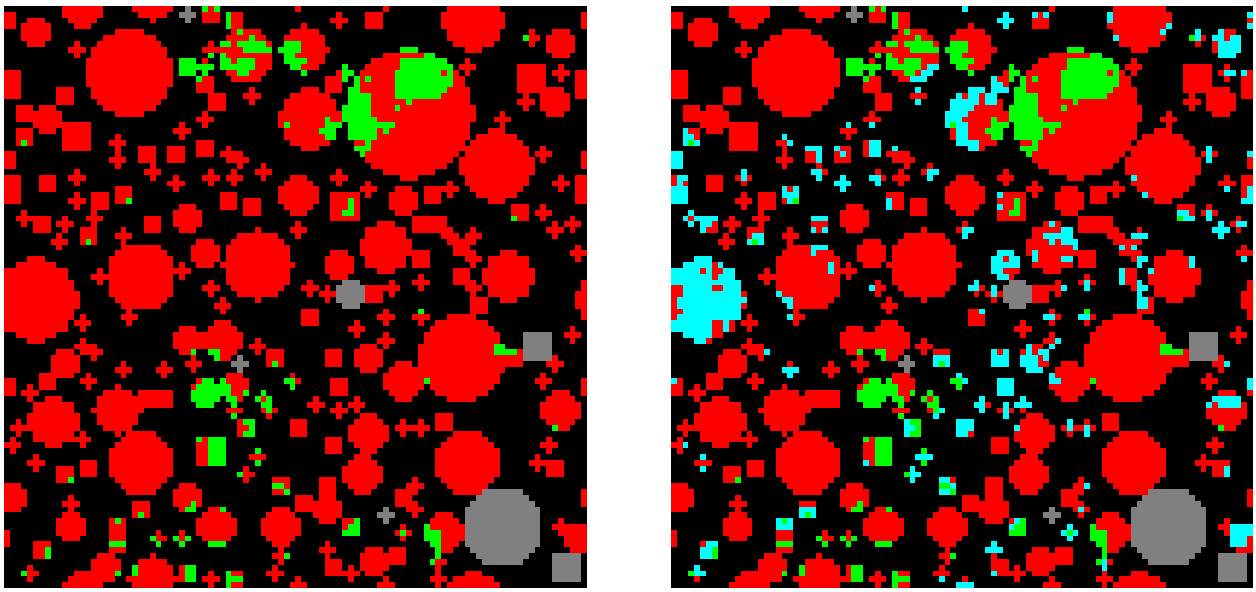


Figure 6: Two-dimensional slices from 3-D microstructures for cement 133 after "sintering" to correctly adjust surface area fractions (hydraulic radius) following segmentation into silicates and aluminates (left) and following further segmentation into C_3S, C_2S, and aluminates, with w/c=0.30. Color assignments are black- porosity, red- silicates (C_3S), aqua- C_2S, green- aluminates, and grey- gypsum. Images are 100 pixels $\times$ 100 pixels.

```
0.842194                phase fraction (0.0-1.0) to remain as C3S
'cem133wc030n1c.img'    filename of output 3-D microstructure
```

An image from the resultant microstructure is shown in the right side of Fig. 5.

When `stat3d` is executed on `cem133wc030n1c.img`, the following output is returned:

```
Phase     Volume       Surface      Volume     Surface
 ID       count         count       fraction   fraction
  0      569660             0
  1      285898        317785      0.70300    0.70255
  2       52611         55562      0.12937    0.12284
  3           0             0      0.00000    0.00000
  4       68173         78981      0.16763    0.17461
Total    406682        452328

  5       23658         23382
  6           0             0
  7           0             0
  8           0             0
  9           0             0
 24           0             0
 25           0             0
```

The desired count for the surface count of C_3S would be $0.7863 \times (317785 + 55562) = 293563$. Thus, the desired R_h to use in the sintering program can be calculated as $(6 \times 285898)/(4 \times$

293563) = 1.461. This value was used in `sinter3d` and the resultant microstructure output to a file called `cem133wc030n1d.img`, a 2-D slice from which is shown in the right side of Fig. 6. The above process is then repeated to separate the aluminates (phase 4) into C_4AF (phase 4) and C_3A (phase 3), using the `cem133wc030n1d.img` file and the correlation file `cem133r.c4f` as input for `rand3d`. After the final sintering, the phase fraction statistics are as follows:

Phase ID	Volume count	Surface count	Volume fraction	Surface fraction
0	569660	0		
1	285898	293384	0.70300	0.64861
2	52611	79963	0.12937	0.17678
3	33911	51701	0.08338	0.11430
4	34262	27280	0.08425	0.06031
Total	406682	452328		
5	23658	23382		
6	0	0		
7	0	0		
8	0	0		
9	0	0		
24	0	0		
25	0	0		

Note that all of the volume fractions and surface fractions are very close to the values determined for the real 2-D image and given in the first table in Fig. 2.

Final 2-D slices for the 3-D microstructures for cement 133 for both w/c ratios are shown in Fig. 7, before the addition of one-pixel particles, and in Fig. 8, after this additon. Finally, 3-D images of the central 50 pixel $\times$ 50 pixel $\times$ 50 pixel region of the initial microstructures can be found in Fig. 9 and Fig. 10, for the w/c=0.3 and w/c=0.45 systems, respectively. These 2-D and 3-D images can be compared directly to the real SEM image shown in Fig. 1.

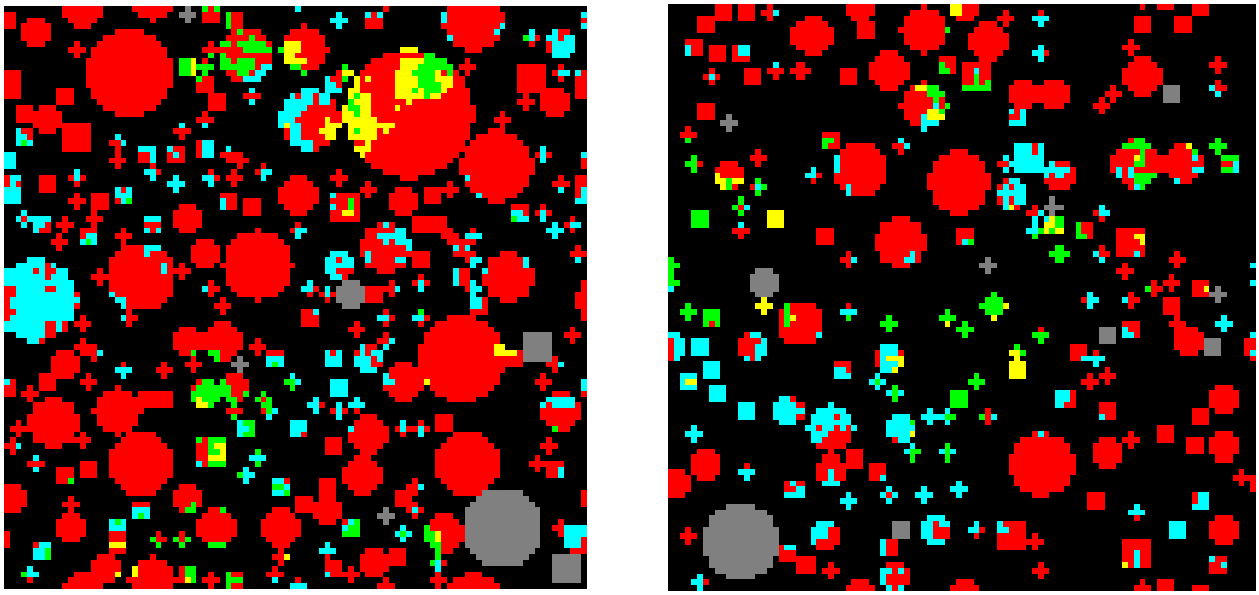


Figure 7: Two-dimensional slices from the final (before addition of any one-pixel particles) 3-D microstructures for cement 133 for w/c=0.30 (left image) and w/c=0.45 (right image). Color assignments are black- porosity, red- C_3S, aqua- C_2S, green- C_3A, yellow- C_4AF, and grey- gypsum. Images are 100 pixels $\times$ 100 pixels.

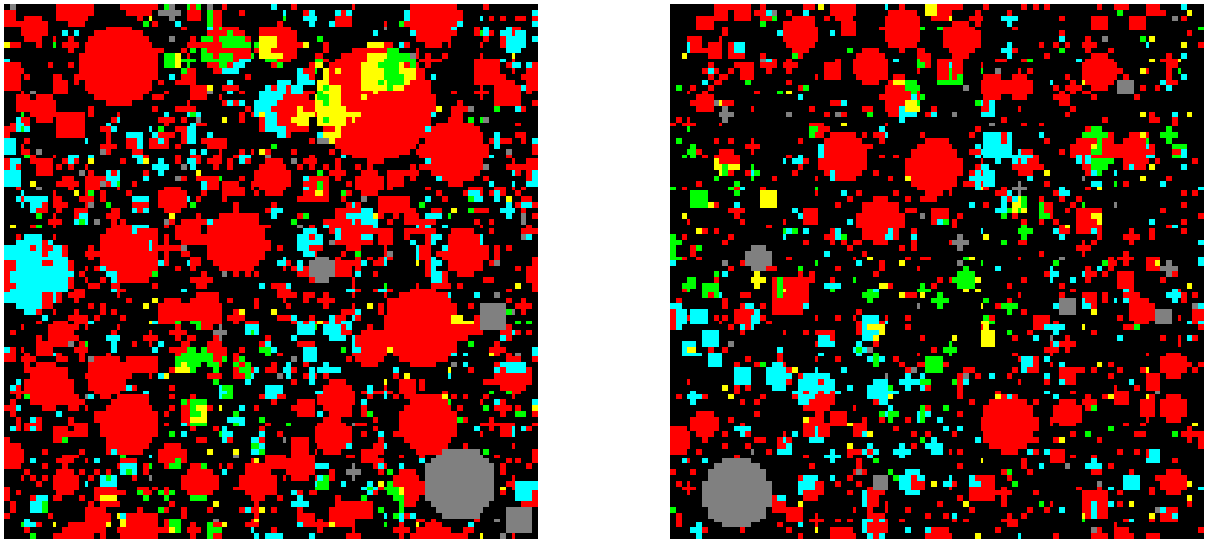


Figure 8: Two-dimensional slices from the final 3-D microstructures for cement 133 for w/c=0.30 (left image) and w/c=0.45 (right image) after the addition of the one-pixel particles by the `disrealnew` program. Color assignments are black- porosity, red- C_3S, aqua- C_2S, green- C_3A, yellow- C_4AF, and grey- gypsum. Images are 100 pixels $\times$ 100 pixels.

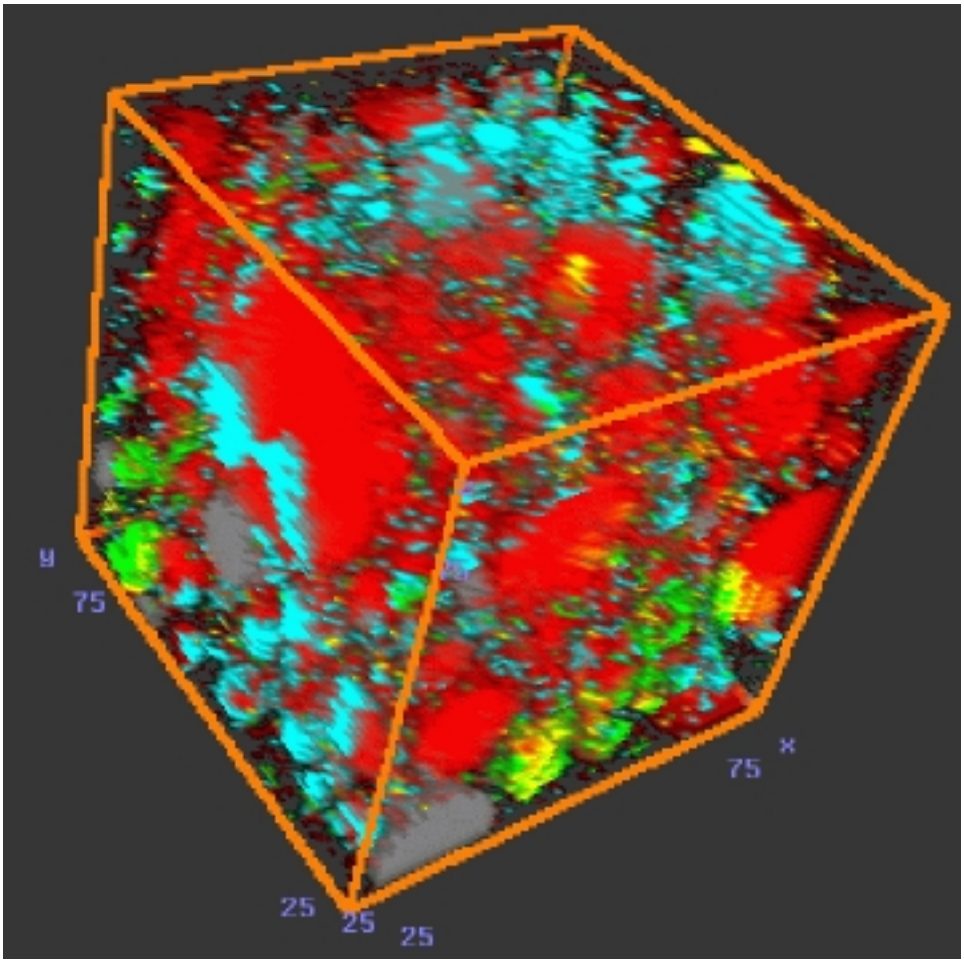


Figure 9: Three-dimensional central image from the final 3-D microstructure for cement 133 for w/c=0.30. Color assignments are black- porosity, red- C_3S, aqua- C_2S, green- C_3A, yellow- C_4AF, and grey- gypsum. Image is 50 pixels $\times$ 50 pixels $\times$ 50 pixels.

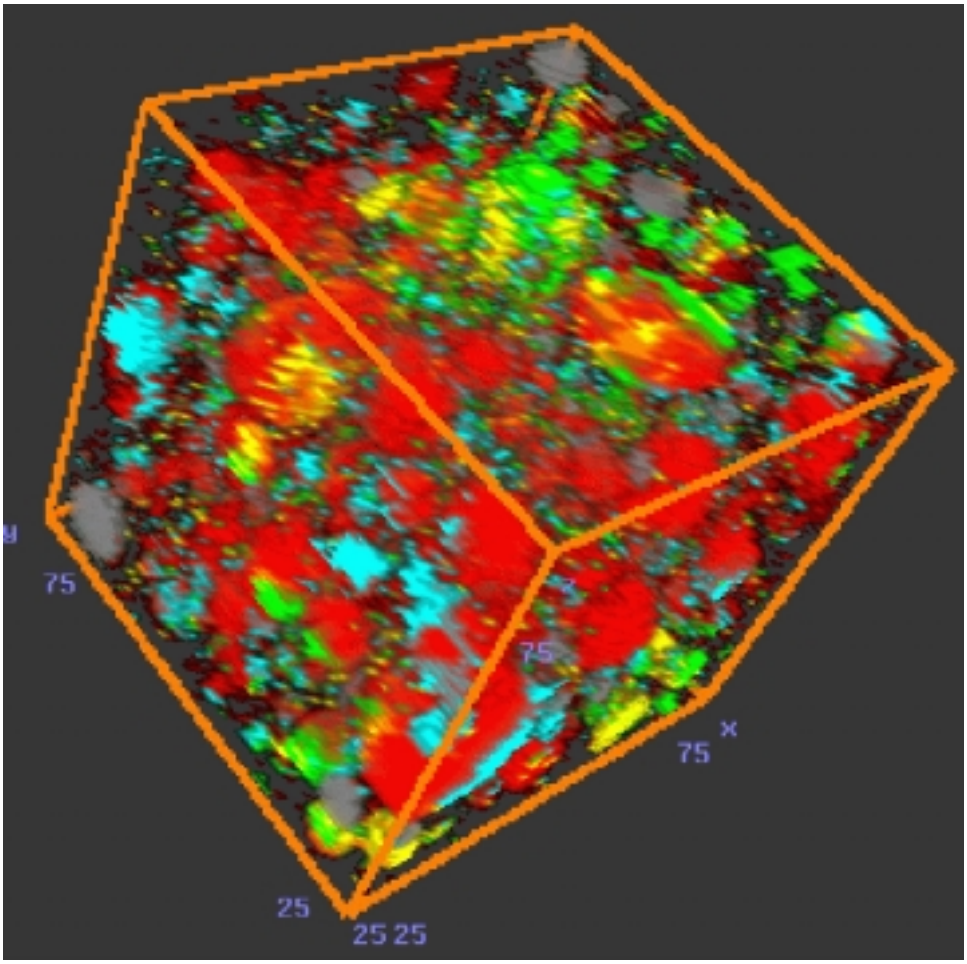


Figure 10: Three-dimensional central image from the final 3-D microstructure for cement 133 for w/c=0.45. Color assignments are black- porosity, red- C_3S, aqua- C_2S, green- C_3A, yellow- C_4AF, and grey- gypsum. Image is 50 pixels $\times$ 50 pixels $\times$ 50 pixels.

4 Enhancements to the Three-dimensional Cement Hydration Model in Version 2.0

4.1 Induction Period Modelling

One of the amazing properties of cement-based materials (both portland cements and pure tricalcium silicate) is the induction period they exhibit when mixed with water. After a short burst of hydration, the material undergoes a dormant period where very little reaction occurs. Then the reactions accelerate and the material sets and strengthens into the hardened concrete. While several mechanisms have been proposed for this induction period, the hypothesis gaining favor recently is that the induction period is controlled by the nucleation and growth of the *C–S–H* phase [18]. To implement this process in the CEMHYD3D model, the early-time dissolution probabilities for all of the cement clinker phases (C_3S, C_2S, C_3A, and C_4AF) are made to be proportional to the square of the amount of *C–S–H* that has formed. As the *C–S–H* volume fraction increases, the calculated dissolution probabilities can not exceed the base dissolution probabilities (set in the array `disbase` in the program `disrealnew`).

To demonstrate that making the early dissolution probabilities proportional to some function of the *C–S–H* which has formed is a viable approach, a study was conducted using pure tricalcium silicate (C_3S) paste. Isothermal calorimetry measurements were made for w/c=0.4 C_3S pastes at three different temperatures (20 °C, 30 °C, and 40 °C). For the hydration of C_3S, an activation energy of 33 kJ/mole was assumed. The measured heat release data was converted to degree of hydration assuming a heat of reaction of 517 J/g for C_3S [15]. In this case, the dissolution probabilities were made to be linearly proportional to the amount of *C–S–H* which has formed. As shown in Fig. 11, the agreement between the experimental measurements and model predictions is reasonable (particularly considering the signal/noise ratio present with the small calorimetry samples), suggesting that modelling the induction period by making initial dissolution probabilities proportional to the amount of formed *C–S–H* may be a reasonable approach. For cement, as opposed to pure C_3S, as will be illustrated in the Example Applications section, a better agreement to early time heat release data is obtained by using a parabolic instead of a linear proportionality between dissolution rates and amount of formed *C–S–H*.

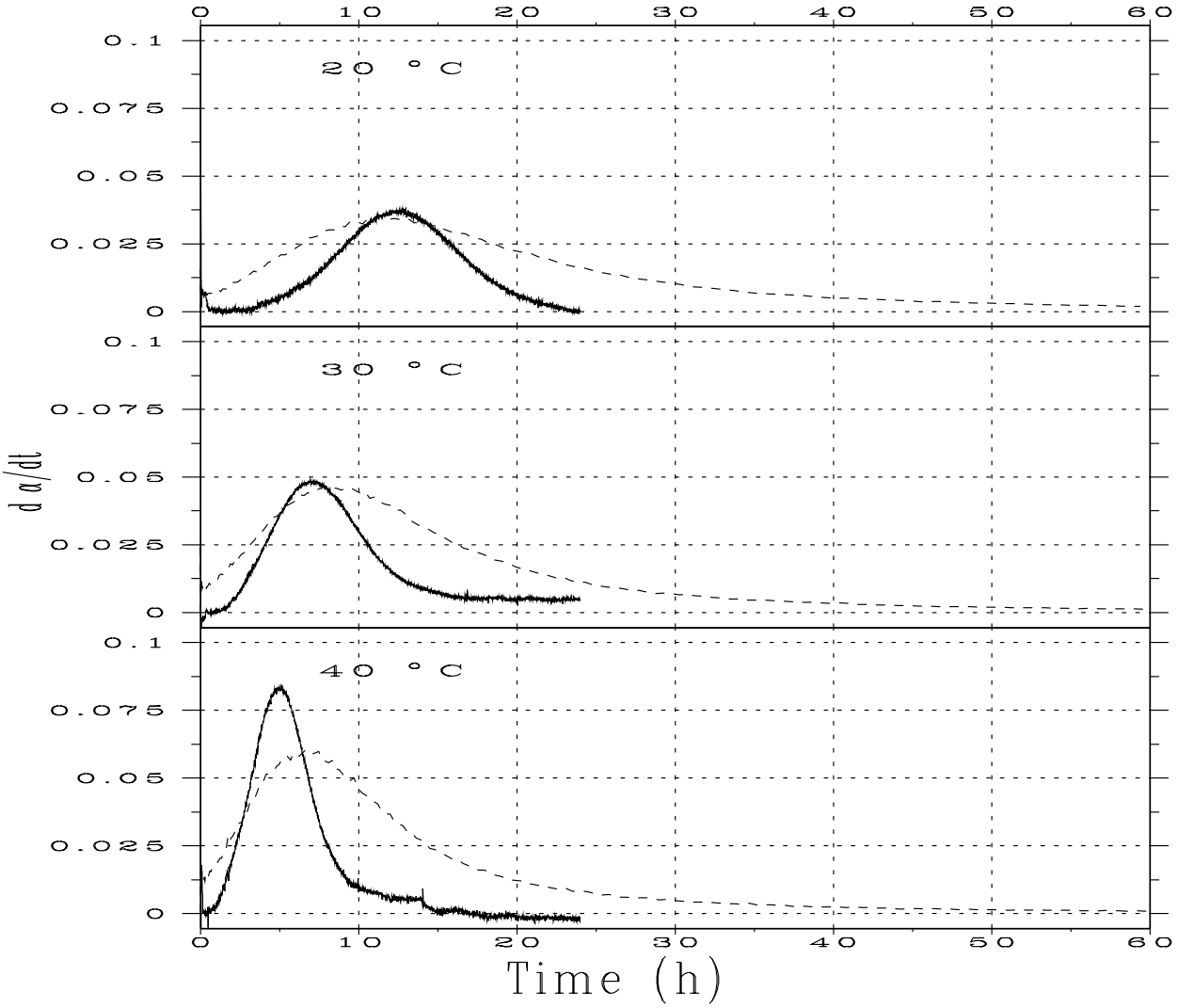


Figure 11: Experimental (solid lines) and model predicted (dashed lines) hydration rates as a function of hydration temperature for a C_3S paste with w/c=0.4.

4.2 Variable Forms of Calcium Sulfate

The initial version of the cement hydration model was developed to only consider the dihydrate form (gypsum) of calcium sulfate. To increase the utility of the computer codes for studying the influence of PSDs and sulfate forms on cement hydration and microstructure development, version 2.0 of the model also includes reactions between the two other major forms of calcium sulfate (hemihydrate and anhydrite) and the cement clinker phases. All three forms of sulfate participate in similar reactions, but their dissolution rates are generally different. Based on data provided in Uchikawa et al. [19], the dissolution probability for hemihydrate has been set at a value three times that used for dihydrate, while the anhydrite has been assigned a dissolution probablity that is 80% of the base value for the dihydrate. The cement hydration model allows for the conversion (hydration) of the anhydrite and hemihydrate forms of calcium sulfate to the dihydrate form during the hydration process; these two phases can also directly react with the aluminate phases present in the cement to form ettringite, etc.

Regarding ettringite formation, based on the experimental results of Odler and Abdul-Maula [20], the ettringite that forms in the model as a result of the reaction between C_4AF and sulfate is stable and does not convert to the monosulfoaluminate phase (Afm), regardless of the sulfate concentration remaining in the system. Ettringite formed from C_3A is only stable when sufficient unreacted sulfate is present in the system and the temperature is less than 70 °C, possibly converting to the Afm phase when these conditions are no longer met. To implement this change in the new version of the hydration model, the diffusing aluminate species created from C_4AF are given a different phase ID (designated diffusing C_4A - `DIFFC4A`) than those created from the C_3A (in the original version of the codes, both these phases created similar diffusing C_3A species- `DIFFC3A`).

An example of the ability of the new version of the model to predict the influence of hemihydrate additions on the hydration of portland cement [21] is provided in Figs. 12 to 14. In this case, the particle size distributions of both the cement and hemihydrate were known, along with the detailed phase composition of the cement (one of the Dyckerhoff cements found in the cement images database described previously). The consumption of C_3S and C_3A and the production of ettringite were assessed using quantitative x-ray diffraction analysis [21]. While not a perfect fit, the model definitely captures the quantitative trends in the experimental results. Improving the model's ability to predict the effects of sulfate form and concentration on cement hydration is a subject of current collaborative research between NIST and Dyckerhoff Zement.

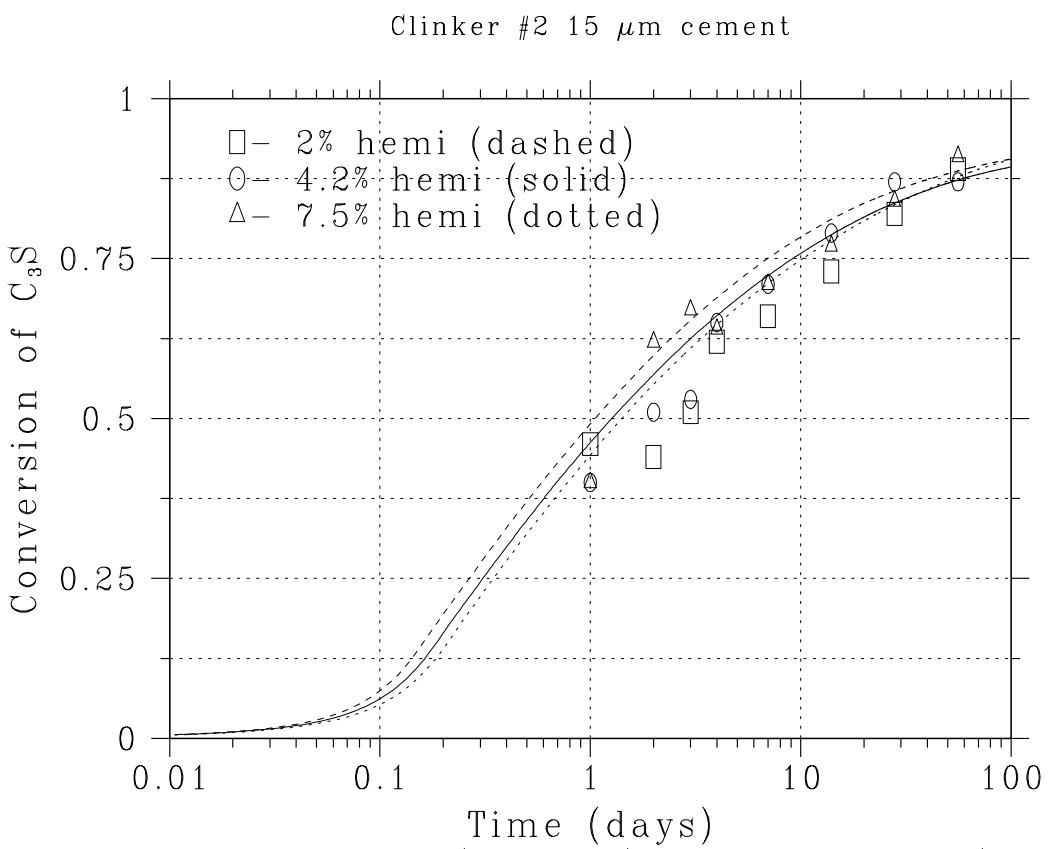


Figure 12: Experimentally measured (data points) and model predictions (lines) for C_3S consumption vs. time for different hemihydrate addition rates.

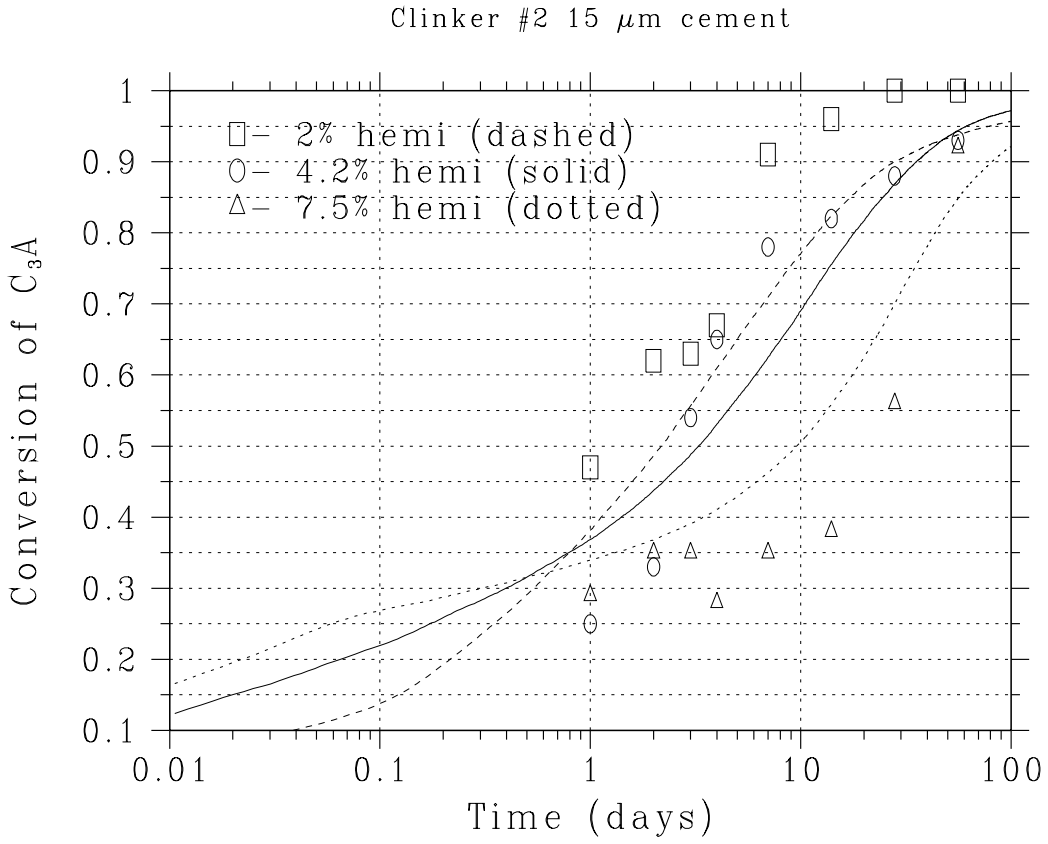


Figure 13: Experimentally measured (data points) and model predictions (lines) for C_3A consumption vs. time for different hemihydrate addition rates.

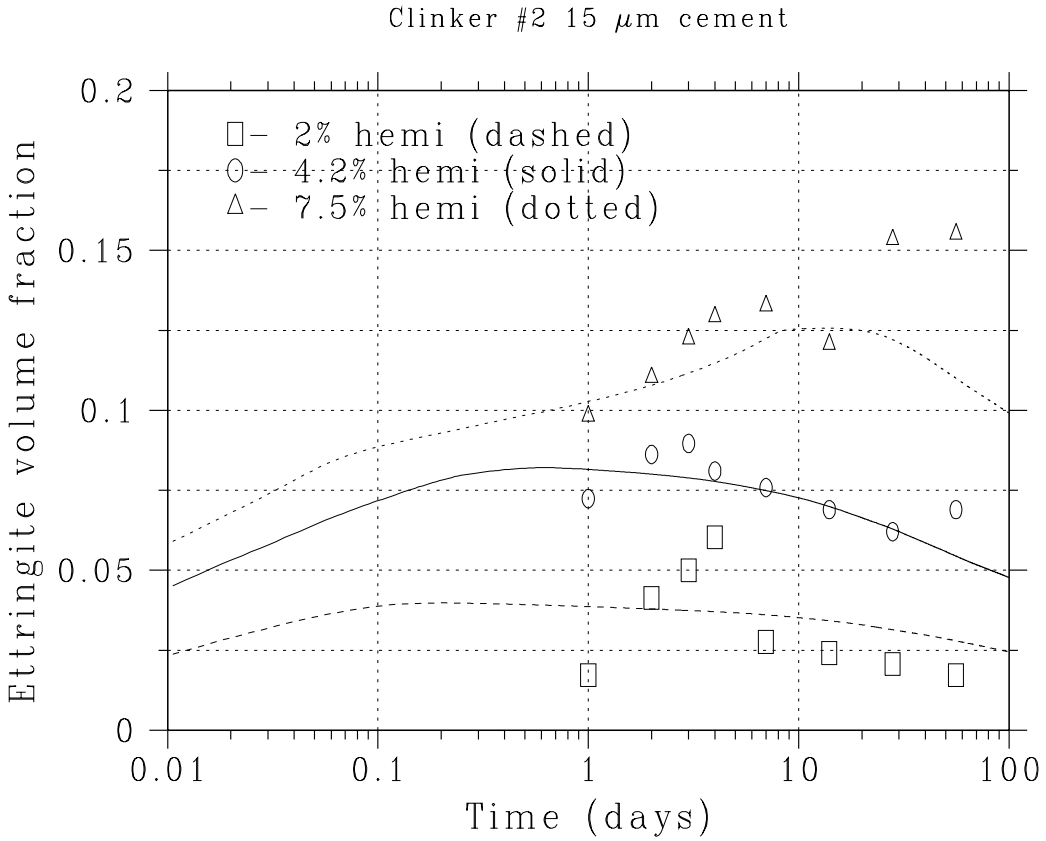


Figure 14: Experimentally measured (data points) and model predictions (lines) for ettringite formation vs. time for different hemihydrate addition rates.

4.3 Temperature-variable *C–S–H* Stoichiometry

It is well known that the pore structure of cement paste is a function of the temperature under which the hydration occurs [22]. One likely explanation for this effect is that the density of the *C–S–H* gel that forms is a function of temperature, with the general tendency to form a denser gel at higher temperatures. A denser gel would be consistent with the observed increase in and coarsening of the capillary porosity in the paste with increasing hydration temperature. Based on chemical shrinkage measurements and other data compiled by Geiker [23], version 2.0 of CEMHYD3D contains the following two functions to describe the relationships between molar volume (*molarv*) and temperature and between water content (*watercon*- the molar ratio of H to S in *C–S–H*) and temperature, respectively:

$$molarv[C\text{–}S\text{–}H] = 1000. \times (108. - 8. \times \frac{T - 20}{80 - 20})\ mm^3/mole \tag{3}$$

$$watercon[C\text{–}S\text{–}H] = 4.0 - 1.3 \times \frac{T - 20}{80 - 20} \tag{4}$$

where T is the hydration temperature in degrees Celsius. These equations result in the following predicted chemical shrinkages for the hydration of C_3S: 0.082 g H_2O / g C_3S at 5 °C, 0.067 at 20 °C, 0.034 at 50 °C, and 0.0 at 80 °C. The hydration model should be used with caution for hydration temperatures above 80 °C, as little experimental data is available above this temperature and phases such as ettringite are generally unstable above 70 °C, anyway. It should be noted that while the model varies the H/S molar ratio of the *C–S–H* with temperature, a constant molar ratio of C/S is assumed regardless of temperature, since the data of Bentur et al. [22] appears inconclusive in this area.

4.4 Variable Temperature Curing

In the first version of CEMHYD3D, curing could be conducted under isothermal or adiabatic conditions. In version 2.0, these capabilities have been augmented to include the possibility of curing that follows a user-programmed temperature profile, such as that employed in steam curing. The desired temperature profile is simply stored in a standard input file `temphist.dat`. This file, which must be created by the user before the execution of `disrealnew`, contains a line by line listing of the endpoints (low value and high value) for time and temperature describing a linearized temperature vs. time relationship. Times are given in hours and temperatures in degrees Celsius. An example datafile (for mild steam curing at 60 °C during the first 24 hours of hydration) is as follows:

```
0.0 4.0 25.0 25.0
4.0 5.0 25.0 60.0
5.0 24.0 60.0 60.0
24.0 28.0 60.0 25.0
28.0 10000.0 25.0 25.0
```

This file specifies that between 0 h and 4 h, the temperature is constant at 25 °C, between 4 h and 5 h, it is ramped from 25 °C to 60 °C, etc. To use this option in version 2.0 of CEMHYD3D, the user simply selects option 2 when prompted to enter "Hydration under 0) isothermal, 1)adiabatic, or 2) programmed temperature history conditions" and the file `temphist.dat` will automatically be consulted during program execution.

4.5 Enhanced Pozzolanic Reactions

Version 2.0 of CEMHYD3D also contains more accurate modelling of the reactions between silica (from silica fume or fly ash) and the cement hydration products. The silica typically reacts with the CH phase to form a pozzolanic *C–S–H* gel of a different specific gravity and molar stoichiometry than the *C–S–H* gel formed from conventional cement hydration. In fact, in the presence of excess silica, the conventional *C–S–H* may convert to the pozzolanic *C–S–H* form. Currently, the pozzolanic reactions included in the hydration model are:

$$1.1CH + S + 2.8H - - > C_{1.1}SH_{3.9}$$
$$C_{1.7}SH_{4.0} + 0.5H - - > C_{1.1}SH_{3.9} + 0.6CH$$

with an assumed molar volume of 101.8 cm^3/mole for the pozzolanic $C_{1.1}SH_{3.9}$ gel [24]. At this time, unlike the conventional *C–S–H*, the molar volume and stoichiometry of pozzolanic *C–S–H* are not temperature dependent. The current version of the model has been shown to provide good agreement with experimental data on the influence of silica fume additions on both the adiabatic temperature rise of concretes [24] and the chloride ion diffusivity of low w/c ratio cement-silica fume pastes [25].

The pozzolanic reactions are regulated by two parameters and one user input within the `disrealnew.c` program. The first parameter, `PPOZZ` (default value of 0.05), specifies the base probability for a reaction to occur when a diffusing CH species encounters a silica surface. The second parameter, `PCSH2CSH` (default value of 0.002), specifies the probability for primary or conventional *C–S–H* dissolution for conversion to the pozzolanic form of the *C–S–H*. If conversion to pozzolanic *C–S–H* is significantly increased due to a high temperature heat treatment, for example, as may be the case for reactive powder concrete [7], the parameter `PCSH2CSH` should be correspondingly increased to model this effect appropriately. The last user input value when initializing a run of CEMHYD3D, called `csh2flag`, indicates whether the conversion of conventional *C–S–H* to pozzolanic *C–S–H* is prohibited (value=0) or allowed (value=1).

5 Execution of the Three-Dimensional Cement Hydration Model

5.1 Inputs

The following inputs are required for execution of version 2.0 of the hydration model (program `disrealnew.c` provided in Appendix C):

a negative integer random number seed,

a flag (0=No, 1=Yes) indicating if the final microstructure is to be output to a file. If a value of 1 is entered for this parameter, the next entry must be the filename of the file to be created to store this microstructure.

the filename of the file containing the initial 3-D microstructure to be used in the hydration model,

for this file, the integer values assigned to C_3S, C_2S, C_3A, C_4AF, gypsum, hemihydrate, anhydrite, and aggregate (separated by spaces). Typical values following

execution of the `rand3d/stat3d/sinter3d` sequence would be 1, 2, 3, 4, 5, 6, 7, and 24.

the phase ID assigned to C_3A in any fly ash particles present in the starting microstructure [5]. Often, there is no fly ash present in the microstructure and this value can be set to its default value of 35.

the filename of the file containing the initial 3-D particle ID microstructure to be used in the hydration model (for assessing setting behavior). Typically, this file is created during the execution of the `genpartnew` program.

the number of one pixel (1 μm) particles of a phase to add. The program contains an iterative loop to continue to accept non-zero values for this parameter, so that the user can place different types of one pixel particles, at their discretion. This iterative process is terminated by the user inputting a value of 0 for the number of one pixel particles to add. Every time a non-zero value is input, the next entry must be the phase ID of the particles to be placed. The phase IDs corresponding to each phase can be found in a series of `#define` statements near the top of the `disrealnew.c` listing in Appendix C.

the number of cycles of the hydration model to execute. Note that this value can be set to zero if, for example, the user wants to output the initial microstructure before any hydration, but after addition of all of the one-pixel particles.

a flag indicating if hydration is to be under (0) saturated or (1) sealed conditions,

the maximum number of diffusion steps to take in a given dissolution cycle (typically a value of 500 is used here),

a prefactor and a scale factor for the nucleation probability of CH according to an exponential function [1],

a prefactor and a scale factor for the nucleation probability of calcium sulfate dihydrate (gypsum) forming from the hemihydrate and anhydrite forms of calcium sulfate,

a prefactor and a scale factor for the nucleation probability of C_3AH_6,

a prefactor and a scale factor for the nucleation probability of FH_3,

the frequency (in cycles) for examining the percolation properties of the capillary pore space (this examination can be totally avoided by setting this parameter to a value larger than the requested number of cycles),

the frequency (in cycles) for examining the percolation properties of the solids (set point),

the frequency (in cycles) for outputting the hydration characteristics of all cement particles present in the microstructure (these results are appended to the file named `partlist.hyd`),

the induction time (*time_induct*) in hours for use in converting model cycles to real time (now that the induction period is directly included in the hydration modelling, this value will often be set to zero),

the initial temperature of the system in degrees Celsius,

the activation energy (kJ/mole) for the cement hydration reactions,

the activation energy (kJ/mole) for pozzolanic reactions,

the calibration factor (β) for converting model cycles to real time in hours based on an equation of the form $time = time\_induct + \beta \times cycles \times cycles$ [1],

the mass fraction of aggregates (0.0 for cement paste hydration) in the concrete mixture proportions (not that present in the 3-D microstructure being used but that present in the concrete mixture being simulated; this is used for the direct simulation of adiabatic heat signature curves [24]),

a flag indicating if hydration is to be under (0) isothermal, (1) adiabatic, or (2) temperature-programmed conditions, and

a flag indicating if conversion of conventional $C{-}S{-}H$ to pozzolanic $C{-}S{-}H$ is (0) prohibited or (1) allowed.

An annotated example datafile for 100 cycles of sealed hydration of cement 133 (w/c=0.30) would be as follows:

```
-389                    random number seed
1                       flag indicating that final microstructure is to be saved
cem133wc030n1.100       filename in which to store final microstructure
cem133wc030n1f.img      filename containing input 3-D phase ID microstructure
1 2 3 4 5 6 7 24        phase assignments for C3S, C2S, etc.
35                      phase ID for C3A in fly ash particles
pcem133wc030n1.img      filename containing input 3-D particle microstructure
52470                   number of one-pixel particles to add
1                       add one-pixel particles of phase C3S
10791                   number of one-pixel particles to add
2                       add one-pixel particles of phase C2S
5962                    number of one-pixel particles to add
3                       add one-pixel particles of phase C3A
6238                    number of one-pixel particles to add
4                       add one-pixel particles of phase C4AF
4403                    number of one-pixel particles to add
5                       add one-pixel particles of phase Gypsum
0                       number of one-pixel particles to add
100                     number of cycles of hydration model to execute
1                       flag for executing model under sealed conditions
500                     maximum number of diffusion steps per cycle
0.01 9000.              nucleation parameters for CH
0.01 9000.              nucleation parameters for calcium sulfate dihydrate
0.002 10000.            nucleation parameters for C3AH6
0.2 2500.               nucleation parameters for FH3
10                      cycle frequency for checking pore space percolation
2                       cycle frequency for checking total solids percolation
```

```
50                    cycle frequency for outputting particle hydration stats
0.0                   induction time in hours
25.0                  initital hydration temperature in degrees Celsius
40.0                  activation energy (kJ/mole) for cement hydration
83.14                 activation energy (kJ/mole) for pozzolanic reactions
0.0003                conversion factor to go from cycles to time
0.72                  aggregate volume fraction in actual concrete mixture
0                     flag indicating hydration is under isothermal conditions
0                     flag indicating no conversion of conventional CSH
```

5.2 Model execution

To compile version 2.0 of CEMHYD3D a typical command line on a UNIX workstation might be:

```
cc disrealnew.c -o disrealnew -lm -O2.
```

Here, the `-lm` instructs the compiler to include the math libraries and the `-O2` (with a capital letter O) specifies the optimization level to be used by the compiler.

The 3-D cement hydration and microstructure development model is described in detail elsewhere [1, 2]. Once the initial 3-D microstructure and particle ID structures are read in, hydration is executed as a series of dissolution/diffusion/reaction cycles. In dissolution, any pixels of a soluble phase that are in contact with water-filled capillary pore space may dissolve and enter solution as one or more diffusing species. These diffusing species then undergo random walks within the pore space until a reaction occurs. After a specified number (typically 500) of diffusion steps have been taken, phase counts are tabulated, empty porosity created to account for the chemical shrinkage (if hydration is under sealed conditions), and another cycle of dissolution is executed. All diffusing species IDs and locations are stored in a dynamically allocated doubly linked list, so that the diffusing species may remain in solution from one dissolution cycle to the next. However, during the last diffusion step of the last cycle of the hydration, all diffusing species are converted into either the phase from which they dissolved or the appropriate hydration products.

In addition, after each dissolution cycle, the heat released by the reactions (as described in the next section) and the degree of hydration that has occurred are used to update the heat capacity and temperature of the system (if the hydration is being executed under adiabatic conditions). The heat capacity is a function of degree of hydration, and undergoes about a 10% decrease during the course of hydration due to changes in the state of the water present in the cement paste [26]. Dividing the incremental heat release by the current heat capacity determines the incremental temperature rise of the system under adiabatic conditions. In this way, the adiabatic response of a concrete system may be predicted by estimating temperature as a function of equivalent real time, calculated based on the application of the principles of the maturity method [27].

5.3 Outputs

In addition to the capability of writing the final hydrated microstructure to a file, depending on user input, between four and seven additional files are created during the execution of

`disrealnew`. The first of these contains a log of the program execution and is written to stdout (standard output). Thus, to save this to file, the user would typically pipe the output (using the > character), perhaps using a command line of the form:

```
disrealnew <disreal3d.dat >disreal3d.out
```

This file (`disreal3d.out`) will contain a listing of all user inputs, the number of diffusing species created during each dissolution, the number of pixels of each phase present during each cycle, data on self-desiccation, and information on the assessment of percolation properties of the pore space and the total solids.

The second file which is always created, `phases.out`, contains for each cycle the number of pixels of each phase present at the end of execution of the cycle and the number of pixels of water remaining in the system (for hydration under sealed or self-desiccating conditions). These phase counts are given in the order in which the phases are listed (0-25) in the `disrealnew.c` program provided in Appendix C. An example of the output generated during the first five cycles of a w/c=0.3 hydration run under sealed isothermal (25 °C) conditions is as follows:

```
0 489796 338368 63402 39873 40500 28061 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 489796
1 488899 338368 63402 39776 40395 27660 0 0 0 0 0 0 28 0 0 701 620 0 52 0 0 0 0
0 0 0 77 488899
2 488017 338288 63398 39686 40307 27306 0 0 0 0 0 0 58 135 0 1372 1102 0 109 0 0
 0 0 0 0 1 150 488017
3 487064 338197 63389 39590 40206 26936 0 0 0 0 0 0 99 301 0 2082 1617 0 164 0 0
 0 0 0 0 4 214 487064
4 486205 338116 63382 39499 40115 26600 0 0 0 0 0 0 152 437 2 2683 2119 0 212 0
0 0 0 0 0 5 287 486205
5 485304 338019 63376 39386 40018 26246 0 0 0 0 0 0 208 591 2 3322 2633 0 259 0
0 0 0 0 0 6 389 485304
```

The first column indicates the hydration cycle number, the second the pixel count for water-filled porosity, the third the pixel count for C_3S, etc. The final two columns indicate the empty porosity created due to self-desiccation and the value determined for the water left in the microstructure. For hydration under sealed conditions (as in this example), the value for the water left should be the same as the count for water-filled porosity in the first column of the same row. It can be seen that the phase counts for the clinker phases and gypsum (columns three through seven) are decreasing during the hydration, while those of the hydration products are increasing, as would be expected.

The third file, `heat.out`, contains for each cycle, the degree of hydration achieved both on a volume and a mass basis and the estimated heat release using four different methods [1], the first based on the specific enthalpy values of each phase and the latter three based on different values for the heats of hydration of each of the four major clinker phases, as documented in the source code listing provided in Appendix C. It is the fourth value that is currently utilized in calculating the temperature rise for hydration under adiabatic conditions [24], and has provided the best fit to experimental data [1, 2]. An example output file is as follows:

```
Cycle  alpha_vol  alpha_mass  heat1 heat2  heat3  heat4
0 0.000000 0.000000 -28530480.000000 0.000000 0.000000 0.000000
1 0.000419 0.000438 -28530894.000000 418.921326 499.942719 620.179260
2 0.000962 0.000994 -28531184.000000 925.209595 1085.318848 1306.325073
3 0.001578 0.001626 -28531820.000000 1489.566772 1734.313965 2070.976318
4 0.002138 0.002199 -28532094.000000 2007.340576 2332.071289 2772.938721
5 0.002788 0.002860 -28531524.000000 2617.215576 3041.136475 3593.079590
```

Since the entries for heat1 correspond to the current total enthalpy present in the microstructure, to determine the heat released based on these values, the user would need to subtract the first entry for this quantity from all subsequent entries. The values for heat2, heat3, and heat4 are in units of kJ $\times$ pixel/(cm^3 of paste $\times$ system). To convert to kJ/(cm^3 paste), one can simply divide by the number of pixels in the system (100^3). To further convert to kJ/kg cement, one needs to divide by the number of kg of cement in a cm^3 of paste. Thus, one calculates the final conversion factor (for a system with only Portland cement) to be:

$$heat4(\frac{kJ}{kg}) = heat4(modelvalue) \times 10^{-3} \times (0.3125 + \frac{w}{c}) \tag{5}$$

This equation becomes more complicated when mineral fillers are present, but the correct equation in this case can be found in the `disrealnew.c` code in the variable `heat_cf`, which is used in the adiabatic temperature rise calculation.

The fourth file, `adiabatic.out`, contains an estimate of the adiabatic heat signature for a concrete or mortar with the 3-D cement paste as its binder component. Here for each cycle, the file contains the estimated equivalent time (age) in hours, the temperature in degrees Celsius, the degree of hydration on a mass basis, the estimated reaction rate at the current temperature (relative to 25 °C), the current estimate of the heat capacity of the mortar or concrete in J/g/ °C, the current mass fraction of cement in the system, and the ratio of the rate constant for the pozzolanic reactions to that for the cement hydration. The current mass fraction of cement value should remain constant unless the hydration is being executed under saturated conditions, in which case the additional water imbibed into the hydrating cement paste will reduce the mass fraction of cement in the overall system. An example output file is as follows;

```
Time(h) Temperature  Alpha  Krate     Cp_now  Mass_cem kpozz/khyd
0.000000 25.000000 0.000000 1.000000 0.000000 0.215337 1.000000
0.000300 25.000000 0.000438 1.000000 0.000000 0.215337 1.000000
0.001200 25.000000 0.000994 1.000000 0.000000 0.215337 1.000000
0.002700 25.000000 0.001626 1.000000 0.000000 0.215337 1.000000
0.004800 25.000000 0.002199 1.000000 0.000000 0.215337 1.000000
0.007500 25.000000 0.002860 1.000000 0.000000 0.215337 1.000000
```

Note that the values for the heat capacity, Cp_{now}, are only determined when the hydration is executed under adiabatic conditions and thus are all zero in the example shown (isothermal conditions).

These last three files are in a format that can be easily imported into a spreadsheet or read directly into a plotting package. This allows the user to compare results from different cements, w/c ratios, or hydration conditions or to produce plots of various properties vs.

the number of hydration cycles or the estimated equivalent real time for comparison to experimental data, as shown in Figs. 15 and 16.

The fifth through seventh output files are created only when the user elects to regularly assess percolation of the pore space, percolation of total solids, or particle hydration characteristics. The fifth file, `percpore.out`, contains the results from analyzing the connectivity of the water-filled capillary porosity during the course of the hydration. This file consists of the following four columns: the number of cycles of hydration executed, the degree of hydration on a mass basis, the count for connected pores, and the total porosity count. Each time this option is executed, three lines of output are generated, one for executing the burning algorithm [13] to assess percolation in each of the three principal directions within the 3-D microstructure. Once the water-filled capillary porosity disconnects in all three of these directions (connected count=0), the percolation properties are no longer assessed as the hydration continues, to reduce the program's execution time. An example output file is as follows:

```
20 0.011002 471448 472885
20 0.011002 471453 472885
20 0.011002 471436 472885
40 0.020940 456515 458401
40 0.020940 456504 458401
40 0.020940 456501 458401
60 0.030379 443423 445768
60 0.030379 443402 445768
60 0.030379 443409 445768
80 0.040059 431470 434408
80 0.040059 431448 434408
80 0.040059 431439 434408
100 0.051172 419360 423136
100 0.051172 419357 423136
100 0.051172 419330 423136
```

Here, the percolation is being assessed every 20 cycles. One can observe that during the first 100 cycles of hydration, the water-filled capillary porosity remains highly connected.

The optional sixth file, `percset.out`, provides similar percolation results, but for the percolation of "total solids". This is useful to assess the setpoint as the cement hydrates. For the assessment of setting, "total solids" refers to the initial cement clinker phases (C_3S, C_2S, C_3A, and C_4AF) along with the $C{-}S{-}H$ and ettringite hydration products. It is thus assumed that setting is due to the building up of bridges of $C{-}S{-}H$ and/or ettringite between neighboring cement clinker particles. This file contains the same four columns as `percpore.out`, but the last two are the tabulations for "total solids" instead of water-filled capillary porosity. An example output file is as follows:

```
2  0.000994  0 484081
2  0.000994  0 484081
2  0.000994  0 484081
4  0.002199  0 486171
4  0.002199  0 486171
4  0.002199  0 486171
```

```
6  0.003494  0 488213
6  0.003494  0 488213
6  0.003494  0 488213
8  0.004646  0 490239
8  0.004646  0 490239
8  0.004646  0 490239
10  0.005705  0 492191
10  0.005705  0 492191
10  0.005705  0 492191
12  0.006752  0 494061
12  0.006752  0 494061
12  0.006752  0 494061
14  0.007805  0 495780
14  0.007805  0 495780
14  0.007805  0 495780
16  0.008887  0 497544
16  0.008887  0 497544
16  0.008887  0 497544
18  0.009978  125404 499339
18  0.009978  0 499339
18  0.009978  0 499339
20  0.011002  136632 501114
20  0.011002  144049 501114
20  0.011002  109638 501114
22  0.012058  168441 502960
22  0.012058  197793 502960
22  0.012058  123816 502960
24  0.013066  184665 504638
24  0.013066  227537 504638
24  0.013066  141164 504638
26  0.014084  271832 506333
26  0.014084  261729 506333
26  0.014084  264142 506333
28  0.015076  323770 507952
28  0.015076  318342 507952
28  0.015076  324153 507952
30  0.016026  344087 509449
30  0.016026  337642 509449
30  0.016026  340686 509449
```

Here, it can be observed that "setting" (connected fraction > 50 %) occurred when approximately 1.5 % of the initial cement phases had hydrated, in general agreement with experimental observations [11].

The optional seventh file, `partlist.hyd`, contains the information on the degree of hydration of each individual cement particle (greater than one pixel in diameter) within the 3-D microstructure. Each time this file is appended, the new first line contains the number of cycles of hydration that have been completed and the current degree of hydration on a mass basis. Subsequent lines contain four columns: the particle ID (100, 101, ...), the initial

number of cement pixels in the particle, the current number of cement pixels in the particle, and the calculated volumetric degree of hydration for this particle. The particle IDs start with 100, because within the program `genpartnew.c`, lower value IDs are used to signify aggregate, etc. The particle hydration data could be used to analyze the dependence of degree of hydration on particle size, as it is well known that while the smaller cement particles ($<$ 10 μm in diameter) hydrate completely within 28 days or so [15], the inner portions of larger particles may remain unhydrated indefinitely. Two small portions of an example output file are as follows:

```
100 0.050533
100 22575 22434 0.006
101 15515 15402 0.007
102 12893 12737 0.012
103 10395 10322 0.007
104 8217 8151 0.008
105 8217 8138 0.010
106 6403 6321 0.013
107 6403 6329 0.012
108 4945 4861 0.017
109 0 0 0.000
110 4945 4912 0.007
111 3695 3635 0.016
112 3695 3655 0.011
113 3695 3646 0.013
114 3695 3647 0.013
115 2553 2520 0.013
116 0 0 0.000
117 2553 2516 0.014
118 2553 2514 0.015
119 2553 2522 0.012
.
.
.
5100 19 18 0.053
5101 19 18 0.053
5102 19 17 0.105
5103 19 18 0.053
5104 19 19 0.000
5105 19 18 0.053
5106 19 18 0.053
5107 19 19 0.000
5108 19 18 0.053
5109 19 19 0.000
5110 19 16 0.158
5111 19 18 0.053
5112 19 15 0.211
5113 19 19 0.000
5114 19 17 0.105
```

```
5115 19 19 0.000
5116 19 18 0.053
5117 19 17 0.105
5118 19 19 0.000
5119 19 16 0.158
5120 19 17 0.105
```

Here, the top line indicates that after 100 cycles of hydration, the mass basis degree of hydration was about 0.05. Subsequent lines indicate that the individual particles have volume-based degrees of hydration between 0.006 and 0.017 for the larger particles (IDs 100 to 118) and generally between 0.053 and 0.211 for the smaller particles (IDs 5100 to 5120), consistent with the statement above that the smaller particles usually hydrate more completely than the larger ones. The particles whose last three columns are all zeros (e.g., 109 and 116 in the above list) indicate calcium sulfate particles (that contain none of the four cement clinker phases on which degree of hydration is based).

6 Example Applications

6.1 Calibration/Prediction of Degree of Hydration and Heat Release

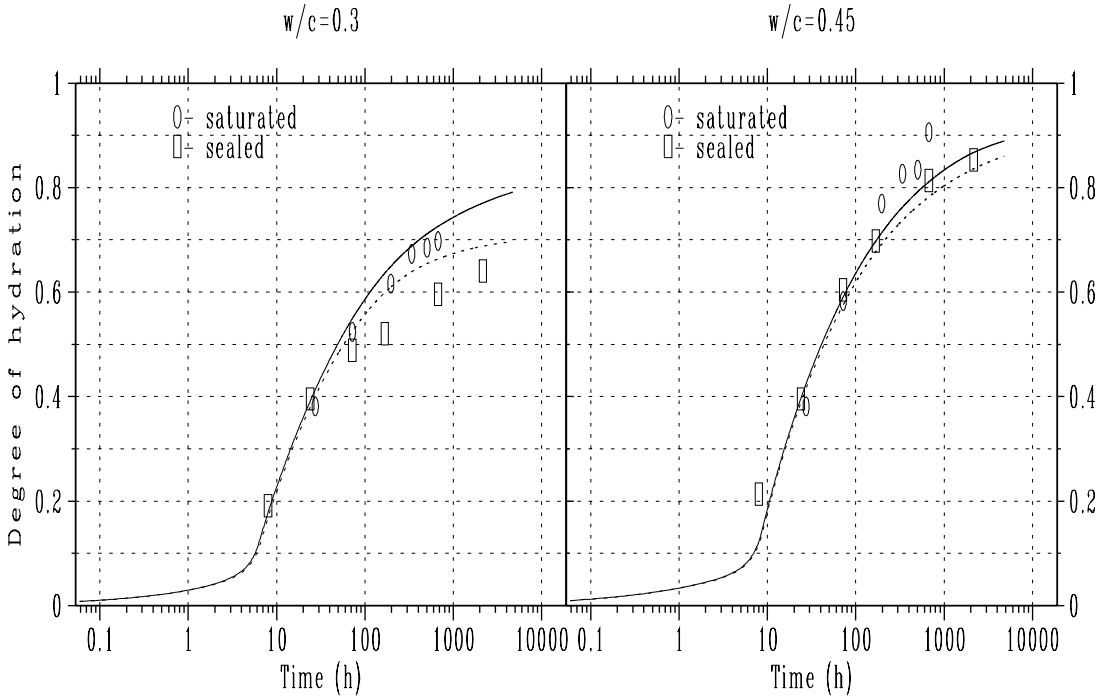


Figure 15: Experimental (data points) and model predicted (lines) degrees of hydration for cement 133, w/c=0.3 and 0.45, hydrated under both saturated (solid line) and sealed (dashed line) conditions.

To model the hydration behavior of a specific cement, a calibration is required to determine the input parameter β governing the relationship between time and cycles, as described

above. For cement 133, this calibration was performed by conducting studies of the degree of hydration vs. time achieved for two different w/c ratios (0.3 and 0.45) and two different curing conditions (saturated [28] and sealed). The degree of hydration was assessed experimentally based on the loss on ignition (non-evaporable water content) [1, 2] of samples hydrated for various ages (typically 8 h and (1, 3, 7, 14, 28, and 90) days). A value of 0.225 g H_2O/g cement was used to represent the non-evaporable water content at complete hydration, to convert from loss on ignition measurements to degrees of hydration. Based on these results and execution of the hydration model, it was determined that the most appropriate value for β in the equation: $time = \beta \times cycles^2$ is 0.0003. This can be compared with previously determined values of 0.0011 [1, 2], 0.0017 [24], and 0.001 [10]. The new value is lower than the previous values due to the incorporation of the induction period directly into the model, which required that the dissolution rates be slowed down (parameter `DISBIAS` was increased from 20.0 to 30.0 in `disrealnew.c`) to improve the early age hydration predictions. This means that each cycle of the hydration will represent a smaller value of real time, hence the reduction in β from about 0.001 to about 0.0003. As shown in Fig. 15 and Fig. 16, using this value for β results in very good fits to the experimental data for degree of hydration under both saturated and sealed conditions and the heat release under sealed conditions. Experimentally, the heat release was measured using isothermal calorimetry for small sealed samples (typically containing about 125 mg of cement paste). For heat release, the model predictions are excellent between 6 h and 30 h, but do overestimate the very early age ($<$ 6 h) heat release. The model appears to slightly underpredict the longterm hydration occuring under saturated conditions for the w/c=0.45 specimens, while providing an excellent prediction for the sealed specimens. However, results in section 6.2 will show that the model does an excellent job of predicting the strength development of mortar cubes prepared with w/c=0.485 and hydrated under saturated conditions.

6.2 Prediction of Strength Development

To predict compressive strength development using the CEMHYD3D model, the gel-space ratio theory of Powers is utilized [29]. This theory states that the compressive strength, σ_c, is related to the gel-space ratio, X, as shown in the following equation:

$$\sigma_c = \sigma_0 \ X^n \tag{6}$$

where σ_0 is an intrinsic strength that depends on the cement composition and PSD, and n generally assumes a value between 2.6 and 3.0. The gel-space ratio, X is defined by [29]:

$$X = \frac{0.68\alpha}{0.32\alpha \ + \ \frac{w}{c}} \tag{7}$$

where α is the mass-based degree of hydration of the cement. For strength predictions using CEMHYD3D, generally, a value of n=2.6 is used [1, 2] and σ_0 is calibrated based on one early-age (1 day or 3 day) strength determination.

For cement 133, the compressive strengths of standard ASTM C109 [30] mortar cubes (w/c=0.485) have been determined by a number of testing laboratories (as part of the Cement and Concrete Reference Laboratory Proficiency Sample Program [9]) after 3 days, 7 days, and 28 days. The mean 3-day value, along with the model-predicted 3-day degree of hydration, was used to determine the value of σ_0 in equation 6 as 83.55 MPa (12,120 psi). This value

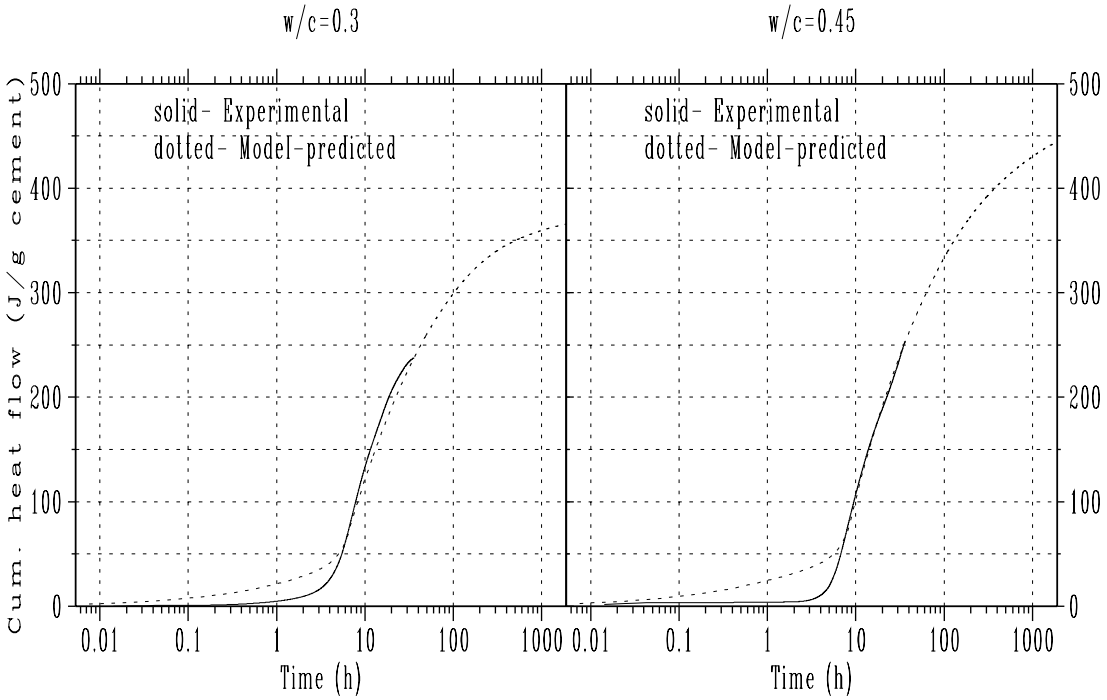


Figure 16: Experimental (solid lines) and model-predicted (dotted lines) cumulative heat release curves for hydration of cement 133 under sealed conditions, with w/c=0.3 (left) and 0.45 (right).

and the CEMHYD3D-predicted hydration development with time were used to predict the strength development of cement 133 (and specifically the strengths that would be achieved after 7 days and 28 days of hydration). The results shown in Fig. 17 clearly illustrate the ability of CEMHYD3D to predict longer term (7 day and/or 28 day) strengths from one early time assessment, in agreement with previous results [1, 2]. The predicted values clearly fall within the standard deviations determined in the CCRL testing program [9].

7 Acknowledgements

The author would like to thank Robin Haupt of CCRL for supplying the sample of cement 133, Paul Stutzman of BFRL/NIST for providing the source SEM and X-ray images used in producing the color composite image shown in Fig. 1, Helene Drud Madsen of the Technical University of Denmark for assistance in sample preparation and degree of hydration measurements, Dr. Xiuping Feng of BFRL/NIST for performing isothermal calorimetry measurements, Dr. Richard Livingston of the Federal Highway Administration for supplying the sample of tricalcium silicate, and Dr. Claus-Jochen Haecker of the Wilhelm Dyckerhoff Institut, Germany for measuring the particle size distribution of cement 133 and for providing the experimental data shown in Figs. 12 to 14. Funding for this research was provided by the NIST Partnership for High Performance Concrete Technology program.

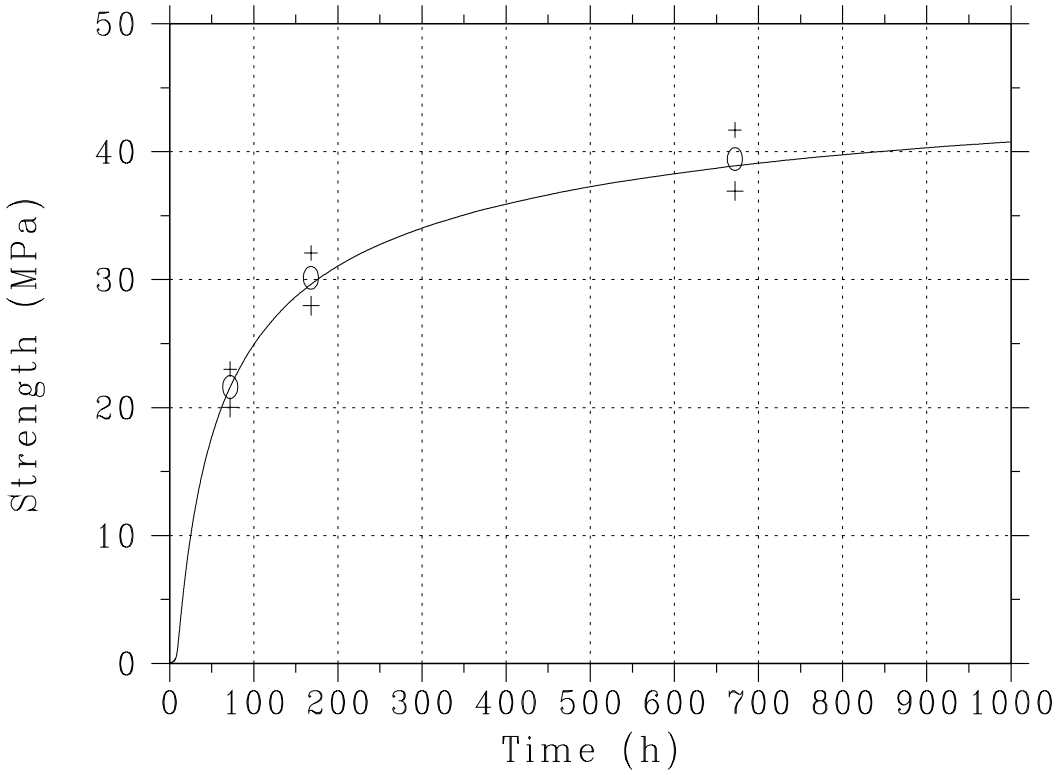


Figure 17: Experimentally measured (circles) and predicted (line) compressive strength development for mortar cubes prepared with cement 133 (w/c=0.485). Crosses indicate +/- one standard deviation from the mean, as determined in the CCRL testing program [9].

8 References

[1] Bentz, D.P., *A Three-Dimensional Cement Hydration and Microstructure Program. I. Hydration Rate, Heat of Hydration, and Chemical Shrinkage*, NISTIR **5756**, U.S. Department of Commerce, November 1995 (available from National Technical Information Service).

[2] Bentz, D.P., *Journal of the American Ceramic Society*, **80** (1), 3-21, 1997.

[3] Bentz, D.P., *Materials and Structures*, **32**, 187-195, 1999.

[4] Bentz, D.P., *Guide to Using CEMHYD3D: A Three-Dimensional Cement Hydration and Microstructure Development Modelling Package*, NISTIR **5977**, U.S. Department of Commerce, February 1997 (available from National Technical Information Service or online at http://ciks.cbt.nist.gov/bentz/cemhyd3d/useguide.html).

[5] Bentz, D.P., and Remond, S., *Incorporation of Fly Ash into a 3-D Cement Hydration Microstructure Model*, NISTIR **6050**, U.S. Department of Commerce, August 1997 (available from National Technical Information Service or online at http://ciks.cbt.nist.gov/bentz/flyash/flyash.html).

[6] Remond, S., "Evolution de la microstructure des betons contenant des dechets au cours de la lixiviation," Ph. D. thesis, L'ecole Normale Superieure de Cachan, 1998.

[7] Matte, V., "Durabilite des betons a ultra hautes performances: role de la matrice cimentaire," Ph. D. thesis, L'ecole Normale Superieure de Cachan, 1999.

[8] Bentz, D.P., Clifton, J.R., and Snyder, K.A., *Concrete International*, **18** (12), 42-47, 1996 (available online at http://ciks.cbt.nist.gov/bentz/ikscldif/iksfinal.html).

[9] Cement and Concrete Reference Laboratory Proficiency Sample Program: Final Report Portland Cement Proficiency Samples Number 133 and Number 134, Cement and Concrete Reference Laboratory, September, 1999.

[10] Bentz, D.P., Garboczi, E.J., Haecker, C.J., and Jensen, O.M., *Cement and Concrete Research*, **29** (10), 1663-1671, 1999.

[11] Jiang, S.P., Mutin, J.C., and Nonat, A., *Cement and Concrete Research*, **25** (4), 779-789, 1995.

[12] Bentz, D.P., Schlangen, E., and Garboczi, E.J., "Computer Simulation of Interfacial Zone Microstructure and Its Effect on the Properties of Cement-Based Composites," Materials Science of Concrete IV, edited by Jan P. Skalny and Sidney Mindess (American Ceramic Society, Westerville, OH, 1995), pp. 155-200.

[13] Stauffer, D., Introduction to Percolation Theory (Taylor and Francis, London, 1985).

[14] Law, A.M, and Kelton, W.D., Simulation Modeling and Analysis (McGraw-Hill, New York, 1982).

[15] Taylor, H.F.W., Cement Chemistry (Academic Press, London, 1990).

[16] Bentz, D.P. and Martys, N.S., *Transport in Porous Media*, **17** (3), 221-238, 1995 (available online at http://ciks.cbt.nist.gov/bentz/tipm/reconstr.html).

[17] Bullard, J.W., Garboczi, E.J., Carter, W.C., and Fuller, E.R., *Computational Materials Science*, **4**, 1-14, 1995 (available online at http://ciks.cbt.nist.gov/garbocz/paper68/paper68.html).

[18] Nonat, A., *Materials and Structures*, **27**, 187-195, 1994.

[19] Uchikawa, H., Uchida, S., and Hanehara, S., *Cement and Concrete Research*, **14**, 645-656, 1984.

[20] Odler, I., and Abdul-Maula, S., *Cement and Concrete Research*, **17**, 22-30, 1987.

[21] Haecker, C.J., unpublished results, 1999.

[22] Bentur, A., Berger, R.L., Kung, K.H., Milestone, N.B., and Young, J.F., *Journal of the American Ceramic Society*, **62** (7/8), 362-366, 1974.

[23] Geiker, M., *Studies of Portland Cement Hydration: Measurements of Chemical Shrinkage and a Systematic Evaluation of Hydration Curves by Means of the Dispersion Model*, Ph. D. Thesis, Technical University of Denmark, Lyngby, DENMARK, 1983.

[24] Bentz, D.P., Waller, V., and deLarrard, F., *Cement and Concrete Research*, **28** (2), 285-297, 1998 (available online at http://ciks.cbt.nist.gov/bentz/ccrahs/ccrahs.html).

[25] Bentz, D.P., Jensen, O.M., Coats, A.M., and Glasser, F.P., "Influence of Silica Fume on Diffusivity in Cement-Based Materials. I. Experimental and Computer Modelling Studies on Cement Paste," accepted by Cement and Concrete Research, 2000.

[26] Waller, V., DeLarrard, F., and Roussel, P., "Modelling the Temperature Rise in Massive HPC Structures," *4th International Symposium on Utilization of High-Strength/High-Performance Concrete*, Paris, Paper 170, 415-421, 1996.

[27] Carino, N.J., Knab, L.I., and Clifton, J.R., *Applicability of the Maturity Method to High-Performance Concrete*, NISTIR **4819**, U.S. Department of Commerce, May 1992.

[28] Carino, N.J., unpublished results, 1999.

[29] Powers, T.C., in 4th ISCC, Vol. 2, p. 577, 1962.

[30] Annual Book of ASTM Standards, Vol. 04.01. Cement; Lime; Gypsum (ASTM, Philadelphia, PA, 1998).

Appendices

A Example spreadsheet for converting cement PSD from a mass basis to a number basis

B Computer programs for two-dimensional to three-dimensional conversion

B.1 Listing for genpartnew.c

```
/*********************************************************************/
/*                                                                   */
/*      Program genpartnew.c to generate three-dimensional cement    */
/*              and gypsum particles in a 3-D box with periodic      */
/*              boundaries.                                          */
/*      Particles are composed of either cement clinker or gypsum,   */
/*              follow a user-specified size distribution, and can   */
/*              be either flocculated, random, or dispersed.         */
/*      Programmer: Dale P. Bentz                                    */
/*                  Building and Fire Research Laboratory            */
/*                  NIST                                             */
/*                  100 Bureau Drive Mail Stop 8621                  */
/*                  Gaithersburg, MD  20899-8621   USA               */
/*                  (301) 975-5865      FAX: 301-990-6891            */
/*                  E-mail: dale.bentz@nist.gov                      */
/*                                                                   */
/*********************************************************************/
/* Modified 3/97 to allow placement of pozzolanic, inert and fly ash particles */
/* Modified 9/98 to allow placement of various forms of gypsum */
/* Documented version produced 1/00 */
#include <stdio.h>
#include <math.h>

#define SYSSIZE 100     /* system size in pixels per dimension */
#define MAXTRIES 150000 /* maximum number of random tries for sphere placement */

/* phase identifiers */
#define POROSITY 0
/* Note that each particle must have a separate ID to allow for flocculation */
#define CEM 100         /* and greater */
#define CEMID 1         /* phase identifier for cement */
#define GYPID 5         /* phase identifier for gypsum */
#define HEMIHYDRATE 6  /* phase identifier for hemihydrate */
#define ANHYDRITE 7    /* phase identifier for anhydrite */
#define POZZID 8       /* phase identifier for pozzolanic material */
#define INERTID 9      /* phase identifier for inert material */
#define AGG 24          /* phase identifier for flat aggregate */
#define FLYASH 25       /* phase identifier for all fly ash components */
#define NPARTC 20000    /* maximum number of particles allowed in box*/
#define BURNT 24000     /* this value must be at least 100 > NPARTC */
#define NUMSIZES 30     /* maximum number of different particle sizes */
```

```
/* data structure for clusters to be used in flocculation */
struct cluster{
        int partid; /* index for particle */
        int clustid;   /* ID for cluster to which this particle belongs */
        int partphase;  /* phase identifier for this particle (CEMID or GYPID)*/
        int x,y,z,r; /* particle centroid and radius in pixels */
        struct cluster *nextpart;  /* pointer to next particle in cluster */
};

/* 3-D particle structure (each particle has own ID) stored in array cement */
/* 3-D microstructure is stored in 3-D array cemreal */
static unsigned short int cement [SYSSIZE+1] [SYSSIZE+1] [SYSSIZE+1];
static unsigned short int cemreal [SYSSIZE+1] [SYSSIZE+1] [SYSSIZE+1];
int npart,aggsize;     /* global number of particles and size of aggregate */
int *seed;      /* random number seed- global */
int dispdist;   /* dispersion distance in pixels */
int clusleft;   /* number of clusters in system */
/* parameters to aid in obtaining correct sulfate content */
long int n_sulfate=0,target_sulfate=0,n_total=0,target_total=0,volpart[37];
float probgyp,probhem,probanh; /* probability of gypsum particle instead of cement */
/* and probabilities of anhydrite and hemihydrate */
struct cluster *clust[NPARTC]; /* limit of NPARTC particles/clusters */

/* Random number generator ran1 from Computers in Physics */
/* Volume 6 No. 5, 1992, 522-524, Press and Teukolsky */
/* To generate real random numbers 0.0-1.0 */
/* Should be seeded with a negative integer */
#define IA 16807
#define IM 2147483647
#define IQ 127773
#define IR 2836
#define NTAB 32
#define EPS (1.2E-07)
#define MAX(a,b) (a>b)?a:b
#define MIN(a,b) (a<b)?a:b

double ran1(idum)
int *idum;
/* Calls: no routines */
/* Called by: gsphere,makefloc */
{
        int j,k;
        static int iv[NTAB],iy=0;
        void nrerror();
        static double NDIV = 1.0/(1.0+(IM-1.0)/NTAB);
        static double RNMX = (1.0-EPS);
        static double AM = (1.0/IM);
```

```
        if ((*idum <= 0) || (iy == 0)) {
                *idum = MAX(-*idum,*idum);
                for(j=NTAB+7;j>=0;j--) {
                        k = *idum/IQ;
                        *idum = IA*(*idum-k*IQ)-IR*k;
                        if(*idum < 0) *idum += IM;
                        if(j < NTAB) iv[j] = *idum;
                }
                iy = iv[0];
        }
        k = *idum/IQ;
        *idum = IA*(*idum-k*IQ)-IR*k;
        if(*idum<0) *idum += IM;
        j = iy*NDIV;
        iy = iv[j];
        iv[j] = *idum;
        return MIN(AM*iy,RNMX);
}
#undef IA
#undef IM
#undef IQ
#undef IR
#undef NTAB
#undef EPS
#undef MAX
#undef MIN

/* routine to add a flat plate aggregate in the microstructure */
void addagg()
/* Calls: no other routines */
/* Called by: main program */
{
        int ix,iy,iz;
        int agglo,agghi;

/* Be sure aggregate size is an even integer */
        do{
             printf("Enter thickness of aggregate to place (an even integer) \n");
             scanf("%d",&aggsize);
             printf("%d\n",aggsize);
        } while (((aggsize%2)!=0)||(aggsize>(SYSSIZE-2)));

        if(aggsize!=0){
                agglo=(SYSSIZE/2)-((aggsize-2)/2);
                agghi=(SYSSIZE/2)+(aggsize/2);

                /* Aggregate is placed in yz plane */
```

```
                for(ix=agglo;ix<=agghi;ix++){
                for(iy=1;iy<=SYSSIZE;iy++){
                for(iz=1;iz<=SYSSIZE;iz++){

        /* Mark aggregate into both particle and microstructure images */
                        cement [ix][iy][iz]=AGG;
                        cemreal [ix][iy][iz]=AGG;
                }
                }
                }
        }
}

/* routine to check or perform placement of sphere of ID phasein */
/* centered at location (xin,yin,zin) of radius radd */
/* wflg=1 check for fit of sphere */
/* wflg=2 place the sphere */
/* phasein and phase2 are phases to assign to cement and cemreal images resp. */
int chksph(xin,yin,zin,radd,wflg,phasein,phase2)
        int xin,yin,zin,radd,wflg,phasein,phase2;
/* Calls: no other routines */
/* Called by: gsphere */
{
        int nofits,xp,yp,zp,i,j,k;
        float dist,xdist,ydist,zdist,ftmp;

        nofits=0;       /* Flag indicating if placement is possible */
/* Check all pixels within the digitized sphere volume */
        for(i=xin-radd;((i<=xin+radd)&&(nofits==0));i++){
                xp=i;
                /* use periodic boundary conditions for sphere placement */
                if(xp<1) {xp+=SYSSIZE;}
                else if(xp>SYSSIZE) {xp-=SYSSIZE;}
                ftmp=(float)(i-xin);
                xdist=ftmp*ftmp;
        for(j=yin-radd;((j<=yin+radd)&&(nofits==0));j++){
                yp=j;
                /* use periodic boundary conditions for sphere placement */
                if(yp<1) {yp+=SYSSIZE;}
                else if(yp>SYSSIZE) {yp-=SYSSIZE;}
                ftmp=(float)(j-yin);
                ydist=ftmp*ftmp;
        for(k=zin-radd;((k<=zin+radd)&&(nofits==0));k++){
                zp=k;
                /* use periodic boundary conditions for sphere placement */
                if(zp<1) {zp+=SYSSIZE;}
                else if(zp>SYSSIZE) {zp-=SYSSIZE;}
```

```
                ftmp=(float)(k-zin);
                zdist=ftmp*ftmp;

        /* Compute distance from center of sphere to this pixel */
                dist=sqrt(xdist+ydist+zdist);
                if((dist-0.5)<=(float)radd){
                        /* Perform placement */
                        if(wflg==2){
                                cement [xp] [yp] [zp]=phasein;
                                cemreal [xp][yp][zp]=phase2;
                        }
                        /* or check placement */
                        else if((wflg==1)&&(cement [xp] [yp] [zp] !=POROSITY)){
                                nofits=1;
                        }
                }
                /* Check for overlap with aggregate */
      if((wflg==1)&&((fabs(xp-((float)(SYSSIZE+1)/2.0)))<((float)aggsize/2.0))){
                        nofits=1;
                }
        }
        }
        }

        /* return flag indicating if sphere will fit */
        return(nofits);
}

/* routine to place spheres of various sizes and phases at random */
/* locations in 3-D microstructure */
/* numgen is number of different size spheres to place */
/* numeach holds the number of each size class */
/* sizeeach holds the radius of each size class */
/* pheach holds the phase of each size class */
void gsphere(numgen,numeach,sizeeach,pheach)
        int numgen;
        long int numeach[NUMSIZES];
        int sizeeach[NUMSIZES],pheach[NUMSIZES];
/* Calls: makesph, ran1 */
/* Called by: create */
{
        int count,x,y,z,radius,ig,tries,phnow;
        long int jg;
        float rx,ry,rz,testgyp,typegyp;
        struct cluster *partnew;

/* Generate spheres of each size class in turn (largest first) */
```

```
        for(ig=0;ig<numgen;ig++){

                phnow=pheach[ig];        /* phase for this class */
                radius=sizeeach[ig];     /* radius for this class */
                /* loop for each sphere in this size class */
                for(jg=1;jg<=numeach[ig];jg++){

                        tries=0;
                        /* Stop after MAXTRIES random tries */
                        do{
                                tries+=1;
                        /* generate a random center location for the sphere */
                                x=(int)((float)SYSSIZE*ran1(seed))+1;
                                y=(int)((float)SYSSIZE*ran1(seed))+1;
                                z=(int)((float)SYSSIZE*ran1(seed))+1;
                                /* See if the sphere will fit at x,y,z */
                                /* Include dispersion distance when checking */
                            /* to insure requested separation between spheres */
                              count=chksph(x,y,z,radius+dispdist,1,npart+CEM,0);
                                if(tries>MAXTRIES){
printf("Could not place sphere %d after %d random attempts \n",npart,MAXTRIES);
                                        printf("Exitting program \n");
                                        exit(1);
                                }
                        } while(count!=0);

                        /* place the sphere at x,y,z */
                        npart+=1;
                        if(npart>=NPARTC){
                                printf("Too many spheres being generated \n");
           printf("User needs to increase value of NPARTC at top of C-code\n");
           printf("Exitting program \n");
                                exit(1);
                        }
                        /* Allocate space for new particle info */
                  clust[npart]=(struct cluster *)malloc(sizeof(struct cluster));
                        clust[npart]->partid=npart;
                        clust[npart]->clustid=npart;
                        /* Default to cement placement */
                        clust[npart]->partphase=CEMID;
                        clust[npart]->x=x;
                        clust[npart]->y=y;
                        clust[npart]->z=z;
                        clust[npart]->r=radius;
                        clusleft+=1;
                        if(phnow==1){
                        testgyp=ran1(seed);
```

```
                /* Do not use dispersion distance when placing particle */
                      if(((testgyp>probgyp)&&((target_sulfate-n_sulfate)<
                      (target_total-n_total)))||(n_sulfate>target_sulfate)||
                       (volpart[radius]>(target_sulfate-n_sulfate))){
                            count=chksph(x,y,z,radius,2,npart+CEM-1,CEMID);
                            n_total+=volpart[radius];
                      }
                      else{
                             /* Place particle as gypsum */
                             typegyp=ran1(seed);
                             n_total+=volpart[radius];
                             n_sulfate+=volpart[radius];
                             if(typegyp>(probanh+probhem)){
                              count=chksph(x,y,z,radius,2,npart+CEM-1,GYPID);
                              /* Correct phase ID of particle */
                              clust[npart]->partphase=GYPID;
                             }
                             else if(typegyp>probanh){
                        count=chksph(x,y,z,radius,2,npart+CEM-1,HEMIHYDRATE);
                                clust[npart]->partphase=HEMIHYDRATE;
                             }
                             else{
                        count=chksph(x,y,z,radius,2,npart+CEM-1,ANHYDRITE);
                                 clust[npart]->partphase=ANHYDRITE;
                             }
                      }
                      }
                      /* place as inert or pozzolanic material */
                      else{
                             n_total+=volpart[radius];
                             count=chksph(x,y,z,radius,2,npart+CEM-1,phnow);
                             /* Correct phase ID of particle */
                             clust[npart]->partphase=phnow;
                      }
                      clust[npart]->nextpart=NULL;
                }
        }
}

/* routine to obtain user input and create a starting microstructure */
void create()
/* Calls: gsphere */
/* Called by: main program */
{
        int numsize,sphrad [NUMSIZES],sphase[NUMSIZES];
        long int sphnum [NUMSIZES],inval1;
        int isph,inval;
```

```
        do{
printf("Enter number of different size spheres to use(max. is %d) \n",NUMSIZES);
                scanf("%d",&numsize);
                printf("%d \n",numsize);
        }while ((numsize>NUMSIZES)||(numsize<0));
        do{
printf
("Enter dispersion factor(separation distance in pixels) for spheres (0-2) \n");
                printf("0 corresponds to totally random placement \n");
                scanf("%d",&dispdist);
                printf("%d \n",dispdist);
        }while ((dispdist<0)||(dispdist>2));
        do{
printf
("Enter probability for gypsum particles on a random part. basis (0.0-1.0) \n");
                scanf("%f",&probgyp);
                printf("%f \n",probgyp);
        } while ((probgyp<0.0)||(probgyp>1.0));
do{
printf
("Enter probab. hemihydrate and anhydrite forms of gypsum (0.0-1.0) \n");
scanf("%f %f",&probhem,&probanh);
printf("%f %f\n",probhem,probanh);
         } while ((probhem<0.0)||(probhem>1.0)||(probanh<0.0)||(probanh>1.0)||
        ((probanh+probhem)>1.0));

        if((numsize>0)&&(numsize<(NUMSIZES+1))){
printf
("Enter number, radius, and phaseID for each sphere class (larg. rad. 1st) \n");
printf
("Phases are %d- Cement and (random) calcium sulfate, %d- Gypsum,",
CEMID,GYPID);
printf
("%d- hemihydrate", %d- anhydrite %d- Pozzolanic, %d- Inert, %d- Fly Ash \n",
HEMIHYDRATE,ANHYDRITE,POZZID,INERTID,FLYASH);

        /* Obtain input for each size class of spheres */
                for(isph=0;isph<numsize;isph++){
                       printf("Enter number of spheres of class %d \n",isph+1);
                        scanf("%ld",&inval1);
                        printf("%ld \n",inval1);
                        sphnum[isph]=inval1;
                        do{
                       printf("Enter radius of spheres of class %d \n",isph+1);
                                printf("(Integer <=%d please) \n",SYSSIZE/4);
                                scanf("%d",&inval);
                                printf("%d \n",inval);
```

```
                        } while ((inval<1)||(inval>(SYSSIZE/4)));
                        sphrad[isph]=inval;
                        do{
                        printf("Enter phase of spheres of class %d \n",isph+1);
                                scanf("%d",&inval);
                                printf("%d \n",inval);
        } while ((inval!=1)&&(inval!=5)&&(inval!=6)&&(inval!=7)&&
          (inval!=8)&&(inval!=9)&&(inval!=25));
                        sphase[isph]=inval;
                        target_total+=sphnum[isph]*volpart[sphrad[isph]];

                }
                target_sulfate=(int)((float)target_total*probgyp);
                gsphere(numsize,sphnum,sphrad,sphase);
        }
}

/* Routine to draw a particle during flocculation routine */
/* See routine chksph for definition of parameters */
void drawfloc(xin,yin,zin,radd,phasein,phase2)
        int xin,yin,zin,radd,phasein,phase2;
/* Calls: no other routines */
/* Called by: makefloc */
{
        int xp,yp,zp,i,j,k;
        float dist,xdist,ydist,zdist,ftmp;

/* Check all pixels within the digitized sphere volume */
        for(i=xin-radd;(i<=xin+radd);i++){
                xp=i;
                /* use periodic boundary conditions for sphere placement */
                if(xp<1) {xp+=SYSSIZE;}
                else if(xp>SYSSIZE) {xp-=SYSSIZE;}
                ftmp=(float)(i-xin);
                xdist=ftmp*ftmp;
        for(j=yin-radd;(j<=yin+radd);j++){
                yp=j;
                /* use periodic boundary conditions for sphere placement */
                if(yp<1) {yp+=SYSSIZE;}
                else if(yp>SYSSIZE) {yp-=SYSSIZE;}
                ftmp=(float)(j-yin);
                ydist=ftmp*ftmp;
        for(k=zin-radd;(k<=zin+radd);k++){
                zp=k;
                /* use periodic boundary conditions for sphere placement */
                if(zp<1) {zp+=SYSSIZE;}
                else if(zp>SYSSIZE) {zp-=SYSSIZE;}
```

```
                ftmp=(float)(k-zin);
                zdist=ftmp*ftmp;

        /* Compute distance from center of sphere to this pixel */
                dist=sqrt(xdist+ydist+zdist);
                if((dist-0.5)<=(float)radd){
                        /* Update both cement and cemreal images */
                        cement [xp] [yp] [zp]=phasein;
                        cemreal [xp] [yp] [zp]=phase2;
                }
        }
        }
        }
}

/* Routine to check particle placement during flocculation */
/* for particle of size radd centered at (xin,yin,zin) */
/* Returns flag indicating if placement is possible */
int chkfloc(xin,yin,zin,radd)
        int xin,yin,zin,radd;
/* Calls: no other routines */
/* Called by: makefloc */
{
        int nofits,xp,yp,zp,i,j,k;
        float dist,xdist,ydist,zdist,ftmp;

        nofits=0;       /* Flag indicating if placement is possible */

/* Check all pixels within the digitized sphere volume */
        for(i=xin-radd;((i<=xin+radd)&&(nofits==0));i++){
                xp=i;
                /* use periodic boundary conditions for sphere placement */
                if(xp<1) {xp+=SYSSIZE;}
                else if(xp>SYSSIZE) {xp-=SYSSIZE;}
                ftmp=(float)(i-xin);
                xdist=ftmp*ftmp;
        for(j=yin-radd;((j<=yin+radd)&&(nofits==0));j++){
                yp=j;
                /* use periodic boundary conditions for sphere placement */
                if(yp<1) {yp+=SYSSIZE;}
                else if(yp>SYSSIZE) {yp-=SYSSIZE;}
                ftmp=(float)(j-yin);
                ydist=ftmp*ftmp;
        for(k=zin-radd;((k<=zin+radd)&&(nofits==0));k++){
                zp=k;
                /* use periodic boundary conditions for sphere placement */
                if(zp<1) {zp+=SYSSIZE;}
```

```
                else if(zp>SYSSIZE) {zp-=SYSSIZE;}
                ftmp=(float)(k-zin);
                zdist=ftmp*ftmp;

        /* Compute distance from center of sphere to this pixel */
                dist=sqrt(xdist+ydist+zdist);
                if((dist-0.5)<=(float)radd){
                        if((cement [xp] [yp] [zp] !=POROSITY)){
                                /* Record ID of particle hit */
                                nofits=cement [xp] [yp] [zp];
                        }
                }
                /* Check for overlap with aggregate */
                if((fabs(xp-((float)(SYSSIZE+1)/2.0)))<((float)aggsize/2.0)){
                        nofits=AGG;
                }
        }
        }
        }
        /* return flag indicating if sphere will fit */
        return(nofits);
}

/* routine to perform flocculation of particles */
void makefloc()
/* Calls: drawfloc, chkfloc, ran1 */
/* Called by: main program */
{
        int partdo,numfloc;
        int nstart;
        int nleft,ckall;
        int xm,ym,zm,moveran;
        int xp,yp,zp,rp,clushit,valkeep;
        int iclus,cluspart[NPARTC];
        struct cluster *parttmp,*partpoint,*partkeep;

        nstart=npart; /* Counter of number of flocs remaining */
        for(iclus=1;iclus<=npart;iclus++){
                  cluspart[iclus]=iclus;
        }
        do{
          printf("Enter number of flocs desired at end of routine (>0) \n");
                scanf("%d",&numfloc);
                printf("%d\n",numfloc);
        } while (numfloc<=0);

        while(nstart>numfloc){
```

```
nleft=0;

/* Try to move each cluster in turn */
for(iclus=1;iclus<=npart;iclus++){
        if(clust[iclus]==NULL){
                nleft+=1;
        }
        else{
                xm=ym=zm=0;
/* Generate a random move in one of 6 principal directions */
moveran=6.*ran1(seed);
switch(moveran){
                case 0:
                        xm=1;
                        break;
                case 1:
                        xm=(-1);
                        break;
                case 2:
                        ym=1;
                        break;
                case 3:
                        ym=(-1);
                        break;
                case 4:
                        zm=1;
                        break;
                case 5:
                        zm=(-1);
                        break;
                default:
                        break;
}

/* First erase all particles in cluster */
partpoint=clust[iclus];
while(partpoint!=NULL){
        xp=partpoint->x;
        yp=partpoint->y;
        zp=partpoint->z;
        rp=partpoint->r;
        drawfloc(xp,yp,zp,rp,0,0);
        partpoint=partpoint->nextpart;
}

ckall=0;
/* Now try to draw cluster at new location */
```

```
partpoint=clust[iclus];
while((partpoint!=NULL)&&(ckall==0)){
        xp=partpoint->x+xm;
        yp=partpoint->y+ym;
        zp=partpoint->z+zm;
        rp=partpoint->r;
        ckall=chkfloc(xp,yp,zp,rp);
        partpoint=partpoint->nextpart;
}

if(ckall==0){
/* Place cluster particles at new location */
        partpoint=clust[iclus];
        while(partpoint!=NULL){
                xp=partpoint->x+xm;
                yp=partpoint->y+ym;
                zp=partpoint->z+zm;
                rp=partpoint->r;
                valkeep=partpoint->partphase;
                partdo=partpoint->partid;
                drawfloc(xp,yp,zp,rp,partdo+CEM-1,valkeep);
                /* Update particle location */
                partpoint->x=xp;
                partpoint->y=yp;
                partpoint->z=zp;
                partpoint=partpoint->nextpart;
        }
}
else{
        /* A cluster or aggregate was hit */
        /* Draw particles at old location */
        partpoint=clust[iclus];

        /* partkeep stores pointer to last particle in list */
        while(partpoint!=NULL){
                xp=partpoint->x;
                yp=partpoint->y;
                zp=partpoint->z;
                rp=partpoint->r;
                valkeep=partpoint->partphase;
                partdo=partpoint->partid;
                drawfloc(xp,yp,zp,rp,partdo+CEM-1,valkeep);
                partkeep=partpoint;
                partpoint=partpoint->nextpart;
        }
        /* Determine the cluster hit */
        if(ckall!=AGG){
                clushit=cluspart[ckall-CEM+1];
```

```
       /* Move all of the particles from cluster clushit to cluster iclus */
                                parttmp=clust[clushit];
                                /* Attach new cluster to old one */
                                partkeep->nextpart=parttmp;
                                while(parttmp!=NULL){
                                        cluspart[parttmp->partid]=iclus;
                           /* Relabel all particles added to this cluster */
                                        parttmp->clustid=iclus;
                                        parttmp=parttmp->nextpart;
                                }
                                /* Disengage the cluster that was hit */
                                clust[clushit]=NULL;
                                nstart-=1;
                           }
                  }
                  }
        }
  printf("Number left was %d but number of clusters is %d \n",nleft,nstart);
        }      /* end of while loop */
        clusleft=nleft;
}

/* routine to assess global phase fractions present in 3-D system */
void measure()
/* Calls: no other routines */
/* Called by: main program */
{
        long int npor,ngyp,ncem,nagg,npozz,ninert,nflyash,nanh,nhem;
        int i,j,k,valph;

/* counters for the various phase fractions */
        npor=0;
        ngyp=0;
        ncem=0;
        nagg=0;
        ninert=0;
        npozz=0;
        nflyash=0;
        nanh=0;
        nhem=0;

/* Check all pixels in 3-D microstructure */
        for(i=1;i<=SYSSIZE;i++){
        for(j=1;j<=SYSSIZE;j++){
        for(k=1;k<=SYSSIZE;k++){
                valph=cemreal [i] [j] [k];
                if(valph==POROSITY) {npor+=1;}
```

```
                else if(valph==CEMID){ncem+=1;}
                else if(valph==GYPID){ngyp+=1;}
                else if(valph==ANHYDRITE){nanh+=1;}
                else if(valph==HEMIHYDRATE){nhem+=1;}
                else if(valph==AGG) {nagg+=1;}
                else if(valph==POZZID) {npozz+=1;}
                else if(valph==INERTID) {ninert+=1;}
                else if(valph==FLYASH) {nflyash+=1;}
        }
        }
        }

/* Output results */
        printf("\n Phase counts are: \n");
        printf("Porosity= %ld \n",npor);
        printf("Cement= %ld \n",ncem);
        printf("Gypsum= %ld \n",ngyp);
        printf("Anhydrite= %ld \n",nanh);
        printf("Hemihydrate= %ld \n",nhem);
        printf("Pozzolan= %ld \n",npozz);
        printf("Inert= %ld \n",ninert);
        printf("Fly Ash= %ld \n",nflyash);
        printf("Aggregate= %ld \n",nagg);
}

/* Routine to measure phase fractions as a function of distance from */
/* aggregate surface      */
void measagg()
/* Calls: no other routines */
/* Called by: main program */
{
        int phase [40],ptot;
        int icnt,ixlo,ixhi,iy,iz,phid,idist;
        FILE *aggfile;

/* By default, results are sent to output file called agglist.out */
        aggfile=fopen("agglist.out","w");
printf
("Distance  Porosity  Cement  Gypsum  Anhydrite Hemihydrate  Pozzolan  Inert");
printf("   Fly Ash\n");
fprintf(aggfile,"Distance  Porosity  Cement  Gypsum  Anhydrite Hemihydrate ");
fprintf(aggfile,"Pozzolan  Inert   Fly Ash\n");

/* Increase distance from aggregate in increments of one */
        for(idist=1;idist<=(SYSSIZE-aggsize)/2;idist++){
                /* Pixel left of aggregate surface */
                ixlo=((SYSSIZE-aggsize+2)/2)-idist;
```

```
                /* Pixel right of aggregate surface */
                ixhi=((SYSSIZE+aggsize)/2)+idist;

        /* Initialize phase counts for this distance */
                for(icnt=0;icnt<39;icnt++){
                        phase[icnt]=0;
                }
                ptot=0;

/* Check all pixels which are this distance from aggregate surface */
                for(iy=1;iy<=SYSSIZE;iy++){
                for(iz=1;iz<=SYSSIZE;iz++){
                        phid=cemreal [ixlo] [iy] [iz];
                        ptot+=1;
                        if(phid<26){
                                phase[phid]+=1;
                        }
                        phid=cemreal [ixhi] [iy] [iz];
                        ptot+=1;
                        if(phid<26){
                                phase[phid]+=1;
                        }
                }
                }

                /* Output results for this distance from surface */
printf("%d   %d   %d  %d  %d %d %d  %d %d\n",idist,phase[0],phase[CEMID],
phase[GYPID],phase[ANHYDRITE],phase[HEMIHYDRATE],phase[POZZID],phase[INERTID],
phase[FLYASH]);
fprintf(aggfile,"%d   %d   %d  %d  %d %d %d  %d %d\n",idist,phase[0],
phase[CEMID],phase[GYPID],phase[ANHYDRITE],phase[HEMIHYDRATE],phase[POZZID],
phase[INERTID],phase[FLYASH]);


         }
         fclose(aggfile);
}

/* routine to assess the connectivity (percolation) of a single phase */
/* Two matrices are used here: one for the current burnt locations */
/*                the other to store the newly found burnt locations */
void connect()
/* Calls: no other routines */
/* Called by: main program */
{
        long int inew,ntop,nthrough,ncur,nnew,ntot;
        int i,j,k,nmatx[29000],nmaty[29000],nmatz[29000];
```

```
      int xcn,ycn,zcn,npix,x1,y1,z1,igood,nnewx[29000],nnewy[29000],nnewz[29000];
         int jnew,icur;

         do{
                 printf("Enter phase to analyze 0) pores 1) Cement  \n");
                 scanf("%d",&npix);
                 printf("%d \n",npix);
         } while ((npix!=0)&&(npix!=1));

         /* counters for number of pixels of phase accessible from top surface */
         /* and number which are part of a percolated pathway */
         ntop=0;
         nthrough=0;

         /* percolation is assessed from top to bottom only */
         /* and burning algorithm is nonperiodic in x and y directions */
         k=1;
         for(i=1;i<=SYSSIZE;i++){
                 for(j=1;j<=SYSSIZE;j++){
                         ncur=0;
                         ntot=0;
                         igood=0;     /* Indicates if bottom has been reached */
if(((cement [i] [j] [k]==npix)&&((cement [i] [j] [SYSSIZE]==npix)||
(cement [i] [j] [SYSSIZE]==(npix+BURNT))))||((cement [i] [j] [SYSSIZE]>=CEM)&&
(cement [i] [j] [k]>=CEM)&&(cement [i] [j] [k]<BURNT)&&(npix==1))){
                                 /* Start a burn front */
                                 cement [i] [j] [k]+=BURNT;
                                 ntot+=1;
                                 ncur+=1;
                                 /* burn front is stored in matrices nmat* */
                                 /* and nnew* */
                                 nmatx[ncur]=i;
                                 nmaty[ncur]=j;
                                 nmatz[ncur]=1;
                               /* Burn as long as new (fuel) pixels are found */
                                 do{
                                         nnew=0;
                                         for(inew=1;inew<=ncur;inew++){
                                                xcn=nmatx[inew];
                                                ycn=nmaty[inew];
                                                zcn=nmatz[inew];

                                                /* Check all six neighbors */
                                                for(jnew=1;jnew<=6;jnew++){
                                                        x1=xcn;
                                                        y1=ycn;
                                                        z1=zcn;
```

```
                                                    if(jnew==1){
                                                            x1-=1;
                                                            if(x1<1){
                                                                x1+=SYSSIZE;
                                                            }
                                                    }
                                                    else if(jnew==2){
                                                            x1+=1;
                                                            if(x1>SYSSIZE){
                                                                x1-=SYSSIZE;
                                                            }
                                                    }
                                                    else if(jnew==3){
                                                            y1-=1;
                                                            if(y1<1){
                                                                y1+=SYSSIZE;
                                                            }
                                                    }
                                                    else if(jnew==4){
                                                            y1+=1;
                                                            if(y1>SYSSIZE){
                                                                y1-=SYSSIZE;
                                                            }
                                                    }
                                                    else if(jnew==5){
                                                            z1-=1;
                                                    }
                                                    else if(jnew==6){
                                                            z1+=1;
                                                    }

/* Nonperiodic in z direction so be sure to remain in the 3-D box */
                                                if((z1>=1)&&(z1<=SYSSIZE)){
if((cement [x1] [y1] [z1]==npix)||((cement [x1] [y1] [z1]>=CEM)&&
(cement [x1] [y1] [z1]<BURNT)&&(npix==1))){
                                                            ntot+=1;
                                                cement [x1] [y1] [z1]+=BURNT;
                                                            nnew+=1;
                                                      if(nnew>=29000){
                                              printf("error in size of nnew \n");
                                                            }
                                                            nnewx[nnew]=x1;
                                                            nnewy[nnew]=y1;
                                                            nnewz[nnew]=z1;
                                        /* See if bottom of system has been reached */
                                                      if(z1==SYSSIZE){
                                                            igood=1;
```

```
                                                        }
                                                        }
                                                }
                                                }
                                        }
                                        if(nnew>0){
                                                ncur=nnew;
                                        /* update the burn front matrices */
                                                for(icur=1;icur<=ncur;icur++){
                                                        nmatx[icur]=nnewx[icur];
                                                        nmaty[icur]=nnewy[icur];
                                                        nmatz[icur]=nnewz[icur];
                                                }
                                        }
                                }while (nnew>0);

                                ntop+=ntot;
                                if(igood==1){
                                         nthrough+=ntot;
                                }
                        }
                }
        }

        printf("Phase ID= %d \n",npix);
        printf("Number accessible from top= %ld \n",ntop);
        printf("Number contained in through pathways= %ld \n",nthrough);

        /* return the burnt sites to their original phase values */
        for(i=1;i<=SYSSIZE;i++){
        for(j=1;j<=SYSSIZE;j++){
        for(k=1;k<=SYSSIZE;k++){
                if(cement [i] [j] [k]>=BURNT){
                        cement [i] [j] [k]-=BURNT;
                }
        }
        }
        }
}

/* Routine to output final microstructure to file */
void outmic()
/* Calls: no other routines */
/* Called by: main program */
{
        FILE *outfile,*partfile;
        char filen[80],filepart[80];
```

```
        int ix,iy,iz,valout;

        printf("Enter name of file to save microstructure to \n");
        scanf("%s",filen);
        printf("%s\n",filen);

        outfile=fopen(filen,"w");

        printf("Enter name of file to save particle IDs to \n");
        scanf("%s",filepart);
        printf("%s\n",filepart);

        partfile=fopen(filepart,"w");

        for(iz=1;iz<=SYSSIZE;iz++){
        for(iy=1;iy<=SYSSIZE;iy++){
        for(ix=1;ix<=SYSSIZE;ix++){
                valout=cemreal[ix][iy][iz];
                fprintf(outfile,"%1d\n",valout);
                valout=cement[ix][iy][iz];
                if(valout<0){valout=0;}
                fprintf(partfile,"%d\n",valout);
        }
        }
        }
        fclose(outfile);
        fclose(partfile);
}

main(){
        int userc;      /* User choice from menu */
        int nseed,ig,jg,kg;

        /* Initialize volume array */
        volpart[0]=1;
        volpart[1]=19;
        volpart[2]=81;
        volpart[3]=179;
        volpart[4]=389;
        volpart[5]=739;
        volpart[6]=1189;
        volpart[7]=1791;
        volpart[8]=2553;
        volpart[9]=3695;
        volpart[10]=4945;
        volpart[11]=6403;
        volpart[12]=8217;
```

```
        volpart[13]=10395;
        volpart[14]=12893;
        volpart[15]=15515;
        volpart[16]=18853;
        volpart[17]=22575;
        volpart[18]=26745;
        volpart[19]=31103;
        volpart[20]=36137;
        volpart[21]=41851;
        volpart[22]=47833;
        volpart[23]=54435;
        volpart[24]=61565;
        volpart[25]=69599;
        volpart[30]=119009;
        volpart[36]=203965;

        printf("Enter random number seed value (a negative integer) \n");
        scanf("%d",&nseed);
        printf("%d \n",nseed);
        seed=(&nseed);

        /* Initialize counters and system parameters */
        npart=0;
        aggsize=0;
        clusleft=0;

        /* clear the 3-D system to all porosity to start */
        for(ig=1;ig<=SYSSIZE;ig++){
        for(jg=1;jg<=SYSSIZE;jg++){
        for(kg=1;kg<=SYSSIZE;kg++){
                cement [ig] [jg] [kg]=POROSITY;
                cemreal [ig] [jg] [kg]=POROSITY;
        }
        }
        }

        /* present menu and execute user choice */
        do{
printf(" \n Input User Choice \n");
printf("1) Exit \n");
printf
("2) Add spherical particles (cement,gypsum, pozzolans) to microstructure \n");
printf("3) Flocculate system by reducing number of particle clusters \n");
printf("4) Measure global phase fractions \n");
printf("5) Add an aggregate to the microstructure \n");
printf("6) Measure single phase connectivity (pores or solids) \n");
printf("7) Measure phase fractions vs. distance from aggregate surface \n");
```

```
printf("8) Output current microstructure to file \n");

                scanf("%d",&userc);
                printf("%d \n",userc);
                fflush(stdout);

                switch (userc) {
                        case 2:
                                create();
                                break;
                        case 3:
                                makefloc();
                                break;
                        case 4:
                                measure();
                                break;
                        case 5:
                                addagg();
                                break;
                        case 6:
                                connect();
                                break;
                        case 7:
                                if(aggsize!=0){
                                        measagg();
                                }
                                else{
                                        printf("No aggregate present. \n");
                                }
                                break;
                        case 8:
                                outmic();
                                break;
                        default:
                                break;
                }
        } while (userc!=1);
}
```

B.2 Listing for rand3d.f

```
C
C     Program RAND3D.F: to create a 100*100*100
C     microstructure based on filtering Gaussian
C     noise with autocorrelation filter of a 2-D
C     image
C     Programmer: Dale Bentz (dale.bentz@nist.gov)
C     Date: 1993
C     1995: Modified to filter random particle images
C     for cement hydration model
C
      INTEGER SEED,SYSSIZE,IRES
      REAL SNEW,S2,SS,SDIFF,PHASEIN,PHASEOUT,XTMP,YTMP
      REAL NORMM(100,100,100),RES(100,100,100)
      REAL MASK(100,100,100)
      REAL FILTER(31,31,31)
      INTEGER R(60)
      REAL S(60),XR(60),SUM(501)
      REAL T1,T2,X1,X2,U1,U2,XRAD,RESMAX,RESMIN
      INTEGER R1,R2,I1,I2,I3,I,J,K,J1,K1
      INTEGER IDO,III,JJJ,IX,IY,IZ,INDEX
      REAL PI,XTOT,FILVAL,RADIUS,XPT,SECT,SUMTOT,VCRIT
      CHARACTER*80 FILEN,FILEM

      PI=3.1415926
      SYSSIZE=100
      WRITE(6,*)'Input random seed (negative integer)'
      READ(5,*)SEED
      WRITE(6,*)SEED
C
C     In this version, only a portion of the original 3-D image is filtered
C     so that for instance, silicates may be separated into C3S and C2S
C     without modification of the aluminate and gypsum phases
C
      WRITE(6,*)'Input existing phase assignment for matching'
      READ(5,*)PHASEIN
      WRITE(6,*)PHASEIN
      WRITE(6,*)'Input phase assignment to be created'
      READ(5,*)PHASEOUT
      WRITE(6,*)PHASEOUT
C
C      Read in mask image (particles) from file
C
      WRITE(6,*)'Enter name of cement microstructure image file'
      READ(5,*)FILEM
      WRITE(6,*)FILEM
```

```
      OPEN(13,FILE=FILEM,STATUS='OLD')

      DO 19 K=1,SYSSIZE
      DO 19 J=1,SYSSIZE
      DO 19 I=1,SYSSIZE
  19  READ(13,*)MASK(I,J,K)

      CLOSE(13)
C
C      Create the gaussian noise image
C      by appropriately modifying uniform noise
C      image
C      Source: Law, A.M, and Kelton, W.D.,
C      Simulation Modeling and Analysis, McGraw-Hill, 1982.
C
      I1=1
      I2=1
      I3=1
      DO 70 I=1,500000
      U1=RAN1(SEED)
      U2=RAN1(SEED)
      T1=2.*PI*U2
      T2=(-2.*LOG(U1))**0.5
      X1=COS(T1)*(T2)
      X2=SIN(T1)*(T2)
      NORMM(I1,I2,I3)=X1
      I1=I1+1
      IF(I1.GT.SYSSIZE) THEN
        I1=1
        I2=I2+1
        IF(I2.GT.SYSSIZE) THEN
          I2=1
          I3=I3+1
        ENDIF
      ENDIF
      NORMM(I1,I2,I3)=X2
      I1=I1+1
      IF(I1.GT.SYSSIZE) THEN
        I1=1
        I2=I2+1
        IF(I2.GT.SYSSIZE) THEN
          I2=1
          I3=I3+1
        ENDIF
      ENDIF
  70  CONTINUE
C
```

```
C      Now convolve with the autocorrelation function of the goal 2-D image
C
       WRITE(6,*)'Enter filename to read in autocorrelation from'
       READ(5,*)FILEN
       WRITE(6,*)FILEN
       OPEN(8,FILE=FILEN,STATUS='OLD')
       READ(8,*)IDO
       WRITE(6,*)'Number of points in correlation file is',IDO
       DO 95 I=1,IDO
       READ(8,*)R(I),S(I)
       XR(I)=R(I)
  95   CONTINUE
       CLOSE(8)
       SS=S(1)
       S2=SS*SS
C
C      Load up the convolution matrix (31*31*31 in size)
C
C      OPEN(10,FILE='FILTER3D.IMG')
       SDIFF=SS-S2
       DO 14 I=0,30
       III=I*I
       DO 13 J=0,30
       JJJ=J*J
       DO 12 K=0,30
       XTMP=III+JJJ+K*K
C
C      Use Linear interpolation to estimate S(x,y,z) from available S(r)
C
       RADIUS=SQRT(XTMP)
       R1=RADIUS+1
       R2=R1+1
       XRAD=RADIUS+1-R1
       FILVAL=S(R1)+(S(R2)-S(R1))*XRAD
       FILTER(I+1,J+1,K+1)=(FILVAL-S2)/(SDIFF)
C      WRITE(10,*)FILTER(I+1,J+1,K+1)
  12   CONTINUE
  13   CONTINUE
  14   CONTINUE
C      CLOSE(10)
C
C      Now filter the image maintaining periodic boundaries
C
C      Store minimum (RESMIN) and maximum (RESMAX) values for later binning
C
       RESMAX=0.0
       RESMIN=1.0
```

```
      DO 97 I=1,SYSSIZE
      DO 97 J=1,SYSSIZE
      DO 97 K=1,SYSSIZE
      RES(I,J,K)=0.0
      IF(MASK(I,J,K).EQ.PHASEIN) THEN
      DO 98 IX=1,31
      I1=I+IX-1
C
C     Periodic boundaries
C
      IF(I1.LT.1) THEN
        I1=I1+SYSSIZE
      ELSE IF(I1.GT.SYSSIZE) THEN
        I1=I1-SYSSIZE
      ENDIF
      DO 98 IY=1,31
      J1=J+IY-1
      IF(J1.LT.1) THEN
        J1=J1+SYSSIZE
      ELSE IF(J1.GT.SYSSIZE) THEN
        J1=J1-SYSSIZE
      ENDIF
      DO 98 IZ=1,31
      K1=K+IZ-1
      IF(K1.LT.1) THEN
        K1=K1+SYSSIZE
      ELSE IF(K1.GT.SYSSIZE) THEN
        K1=K1-SYSSIZE
      ENDIF
      RES(I,J,K)=RES(I,J,K)+NORMM(I1,J1,K1)*FILTER(IX,IY,IZ)
  98  CONTINUE
C
C     Check for min/max if pixel is part of proper phase
C
      IF(RES(I,J,K).GT.RESMAX) THEN
        RESMAX=RES(I,J,K)
      ENDIF
      IF(RES(I,J,K).LT.RESMIN) THEN
        RESMIN=RES(I,J,K)
      ENDIF
      ENDIF
  97  CONTINUE
C
C      Now threshold the image
C
      WRITE(6,*)'Input desired threshold phase fraction'
      READ(5,*)XPT
```

```
      WRITE(6,*)XPT
C
C     Bin the resultant (filtered) values into 500 bins
C     so that adequate binarization (thresholding) can be performed
C
      SECT=(RESMAX-RESMIN)/500.0
      WRITE(6,*)'SECT is',SECT
      WRITE(6,*)'RESMAX and RESMIN are ',RESMAX,RESMIN
      DO 34 I=1,500
  34  SUM(I)=0.0
C
C     Fill in the 500-bin histogram
C
      XTOT=0.0
      DO 33 I=1,SYSSIZE
      DO 33 J=1,SYSSIZE
      DO 33 K=1,SYSSIZE
      IF(MASK(I,J,K).EQ.PHASEIN) THEN
        XTOT=XTOT+1.0
        INDEX=1+(RES(I,J,K)-RESMIN)/SECT
        IF(INDEX.GT.500) THEN
          INDEX=500
        ENDIF
        SUM(INDEX)=SUM(INDEX)+1.0
      ENDIF
 33   CONTINUE
C
C     Determine which bin to choose for correct thresholding
C
      SUMTOT=0.0
      VCRIT=0.0
      DO 35 I=1,500
      SUMTOT=SUMTOT+SUM(I)/(XTOT)
      IF(SUMTOT.GT.XPT) THEN
        YTMP=I
        VCRIT=RESMIN+(RESMAX-RESMIN)*(YTMP-0.5)/500.
        GOTO 36
      ENDIF
  35  CONTINUE

  36  IRES=0
      WRITE(6,*)'VCRIT is',VCRIT
C
C     Perform the segmentation and output the resultant image
C
      WRITE(6,*)'Enter name of new cement microstructure image file'
      WRITE(6,*)'to be created'
```

```
      READ(5,*)FILEM
      WRITE(6,*)FILEM
      OPEN(9,FILE=FILEM,STATUS='NEW')
      DO 32 K=1,SYSSIZE
      DO 32 J=1,SYSSIZE
      DO 32 I=1,SYSSIZE
C
C     Only set values that were originally the phase of choice
C
      IF(MASK(I,J,K).EQ.PHASEIN) THEN
        IF(RES(I,J,K).GT.VCRIT) THEN
          RES(I,J,K)=PHASEOUT
        ELSE
          RES(I,J,K)=PHASEIN
          IRES=IRES+1
        ENDIF
      ELSE
        RES(I,J,K)=MASK(I,J,K)
      ENDIF
      WRITE(9,65)INT(RES(I,J,K))
   65 FORMAT(I2)
   32 CONTINUE
      CLOSE(9)
C     WRITE(6,*)'Solids are ',FLOAT(IRES)/1000000.
      STOP
      END
      FUNCTION RAN1(IDUM)
C
C     Portable random number generator, RAN1
C     To generate uniform random deviates between 0 and 1.0
C     From: Computers in Physics, Vol. 6, (5), 1992, 522-524
C           Press and Teukolsky
C
C     CAll with idum set to a negative number to initialize
C     RNMAX should approximate the largest floating value that
C     is less than 1.0
      INTEGER IDUM,IA,IM,IQ,IR,NTAB,NDIV
      REAL RAN1,AM,EPS,RNMAX
      PARAMETER (IA=16807,IM=2147483647,AM=1./IM,IQ=127773,IR=2836,
     +     NTAB=32,NDIV=1+(IM-1)/NTAB,EPS=1.2E-7,RNMAX=1.-EPS)

      INTEGER J,K,IV(NTAB),IY
      SAVE IV,IY
      DATA IV /NTAB*0/, IY /0/

      IF(IDUM.LE.0.OR.IY.EQ.0) THEN
        IF(IDUM.LE.0) THEN
```

```
          IDUM=MAX(-IDUM,1)
        ENDIF
        DO 11 J=NTAB+8,1,-1
          K=IDUM/IQ
          IDUM=IA*(IDUM-K*IQ)-IR*K
          IF(IDUM.LT.0) IDUM=IDUM+IM
          IF(J.LE.NTAB) IV(J)=IDUM
11      CONTINUE
        IY=IV(1)
      ENDIF
      K=IDUM/IQ
      IDUM=IA*(IDUM-K*IQ)-IR*K
      IF(IDUM.LT.0) IDUM=IDUM+IM
      J=1+IY/NDIV
      IY=IV(J)
      IV(J)=IDUM
      RAN1=MIN(AM*IY,RNMAX)
      RETURN
      END
```

B.3 Listing for rand3d.c

```
/*****************************************************************/
/*                                                               */
/*      Program: rand3d.c                                        */
/*      Purpose: To read in a 3-D image and output               */
/*               a filtered image in which one phase from the original  */
/*               image has been separated into two distinct phases      */
/*      Programmer: Dale P. Bentz                                */
/*                  NIST                                         */
/*                  100 Bureau Drive Mail Stop 8621              */
/*                  Gaithersburg, MD  20899-0001                 */
/*                  Phone: (301) 975-5865                        */
/*                  E-mail: dale.bentz@.nist.gov                 */
/*                                                               */
/*      Created: From Fortran version  2/00                      */
/*****************************************************************/
#include <stdio.h>
#include <math.h>

/* Random number generator */
#define IA 16807
#define IM 2147483647
#define IQ 127773
#define IR 2836
#define NTAB 32
#define EPS (1.2E-07)
#define MAX(a,b) (a>b)?a:b
#define MIN(a,b) (a<b)?a:b

double ran1(idum)
int *idum;
{
        int j,k;
        static int iv[NTAB],iy=0;
        void nrerror();
        static double NDIV = 1.0/(1.0+(IM-1.0)/NTAB);
        static double RNMX = (1.0-EPS);
        static double AM = (1.0/IM);

        if ((*idum <= 0) || (iy == 0)) {
                *idum = MAX(-*idum,*idum);
                for(j=NTAB+7;j>=0;j--) {
                        k = *idum/IQ;
                        *idum = IA*(*idum-k*IQ)-IR*k;
                        if(*idum < 0) *idum += IM;
                        if(j < NTAB) iv[j] = *idum;
                }
```

```
                iy = iv[0];
        }
        k = *idum/IQ;
        *idum = IA*(*idum-k*IQ)-IR*k;
        if(*idum<0) *idum += IM;
        j = iy*NDIV;
        iy = iv[j];
        iv[j] = *idum;
        return MIN(AM*iy,RNMX);
}
#undef IA
#undef IM
#undef IQ
#undef IR
#undef NTAB
#undef EPS
#undef MAX
#undef MIN

#define SYSIZE 100

main(){
        int *seed,iseed,syssize;
        float snew,s2,ss,sdiff,phasein,phaseout,xtmp,ytmp;
        static float normm[SYSIZE+1][SYSIZE+1][SYSIZE+1];
        static float res[SYSIZE+1][SYSIZE+1][SYSIZE+1];
        static int mask[SYSIZE+1][SYSIZE+1][SYSIZE+1];
        static float filter [32][32][32];
        int done,r[61];
        static float s[61],xr[61],sum[502];
        float val2;
        float t1,t2,x1,x2,u1,u2,xrad,resmax,resmin;
        float pi,xtot,filval,radius,xpt,sect,sumtot,vcrit;
        int valin,r1,r2,i1,i2,i3,i,j,k,j1,k1;
        int ido,iii,jjj,ix,iy,iz,index;
        char filen[80],filem[80];
        FILE *infile,*outfile;

/* Get user input */
        pi=3.1415926;
        printf("Enter random number seed (negative integer) \n");
        scanf("%d",&iseed);
        printf("%d\n",iseed);
        seed=(&iseed);

        printf("Enter existing phase assignment for matching\n");
        scanf("%f",&phasein);
```

```
        printf("%f\n",phasein);
        printf("Enter phase assignment to be created by program\n");
        scanf("%f",&phaseout);
        printf("%f\n",phaseout);

        printf("Enter name of cement microstructure image file\n");
        scanf("%s",filen);
        printf("%s\n",filen);

/* Read in the original microstructure image file */
        infile=fopen(filen,"r");

        for(k=1;k<=SYSIZE;k++){
        for(j=1;j<=SYSIZE;j++){
        for(i=1;i<=SYSIZE;i++){
             fscanf(infile,"%d",&valin);
             mask[i][j][k]=valin;
        }
        }
        }
        fclose(infile);

/* Create the Gaussian noise image 2 elements at a time */
       i1=i2=i3=1;
       for(i=1;i<=500000;i++){
             u1=ran1(seed);
             u2=ran1(seed);
             t1=2.*pi*u2;
             t2=sqrt(-2.*log(u1));
             x1=cos(t1)*t2;
             x2=sin(t1)*t2;
             normm[i1][i2][i3]=x1;
             i1+=1;
             if(i1>SYSIZE){
                  i1=1;
                  i2+=1;
                  if(i2>SYSIZE){
                        i2=1;
                        i3+=1;
                  }
             }
             normm[i1][i2][i3]=x2;
             i1+=1;
             if(i1>SYSIZE){
                  i1=1;
                  i2+=1;
                  if(i2>SYSIZE){
```

```
                        i2=1;
                        i3+=1;
                    }
                }
        }

/* Now perform the convolution of */
/* the Gaussian noise image with the correlation filter */
      printf("Enter filename to read in autocorrelation from\n");
      scanf("%s",filen);
      printf("%s\n",filen);
      infile=fopen(filen,"r");

      fscanf(infile,"%d",&ido);
      printf("Number of points in correlation file is %d \n",ido);
      for(i=1;i<=ido;i++){
            fscanf(infile,"%d %f",&valin,&val2);
            r[i]=valin;
            s[i]=val2;
            xr[i]=(float)r[i];
       }
       fclose(infile);
       ss=s[1];
       s2=ss*ss;
/* Load up the convolution (filter) matrix with the correlation function*/
       sdiff=ss-s2;
       for(i=0;i<31;i++){
           iii=i*i;
           for(j=0;j<31;j++){
               jjj=j*j;
               for(k=0;k<31;k++){
                  xtmp=(float)(iii+jjj+k*k);
                  radius=sqrt(xtmp);
                  r1=(int)radius+1;
                  r2=r1+1;
                  xrad=radius+1-r1;
                  filval=s[r1]+(s[r2]-s[r1])*xrad;
                  filter[i+1][j+1][k+1]=(filval-s2)/sdiff;
               }
           }
       }

/* Now filter the image maintaining periodic boundaries */
       resmax=0.0;
       resmin=1.0;
       for(i=1;i<=SYSIZE;i++){
       for(j=1;j<=SYSIZE;j++){
```

```
        for(k=1;k<=SYSIZE;k++){
             res[i][j][k]=0.0;
             if((float)mask[i][j][k]==phasein){
                   for(ix=1;ix<=31;ix++){
                       i1=i+ix-1;
                       if(i1<1){i1+=SYSIZE;}
                       else if(i1>SYSIZE){i1-=SYSIZE;}
                       for(iy=1;iy<=31;iy++){
                            j1=j+iy-1;
                            if(j1<1){j1+=SYSIZE;}
                            else if(j1>SYSIZE){j1-=SYSIZE;}
                            for(iz=1;iz<=31;iz++){
                                 k1=k+iz-1;
                                 if(k1<1){k1+=SYSIZE;}
                                 else if(k1>SYSIZE){k1-=SYSIZE;}
                            res[i][j][k]+=normm[i1][j1][k1]*filter[ix][iy][iz];
                            }
                       }
                   }
                   if(res[i][j][k]>resmax){resmax=res[i][j][k];}
                   if(res[i][j][k]<resmin){resmin=res[i][j][k];}
             }
        }
        }
        }

/* Now threshold the image */
        printf("Input desired threshold phase fraction\n");
        scanf("%f",&xpt);
        printf("%f\n",xpt);

/* Divide the filtered image values into 500 separate bins */
        sect=(resmax-resmin)/500.;
        printf("SECT is %f\n",sect);
        printf("RESMAX and RESMIN are %f and %f\n",resmax,resmin);
        for(i=1;i<=500;i++){
            sum[i]=0.0;
        }
        xtot=0.0;
        for(i=1;i<=SYSIZE;i++){
        for(j=1;j<=SYSIZE;j++){
        for(k=1;k<=SYSIZE;k++){
            if((float)mask[i][j][k]==phasein){
                 xtot+=1;
                 index=1+(int)((res[i][j][k]-resmin)/sect);
                 if(index>500){index=500;}
                 sum[index]+=1.0;
```

```
            }
        }
        }
        }

/* Determine which bin to choose for correct thresholding */
        sumtot=vcrit=0.0;
        done=0;
        for(i=1;((i<=500)&&(done==0));i++){
            sumtot+=sum[i]/xtot;
            if(sumtot>xpt){
                 ytmp=(float)i;
                 vcrit=resmin+(resmax-resmin)*(ytmp-0.5)/500.;
                 done=1;
            }
        }
        printf("Critical volume fraction is %f\n",vcrit);

/* Write out the final resultant image */
        printf("Enter name of new cement microstructure image file\n");
        scanf("%s",filem);
        printf("%s\n",filem);
        outfile=fopen(filem,"w");

        for(k=1;k<=SYSIZE;k++){
        for(j=1;j<=SYSIZE;j++){
        for(i=1;i<=SYSIZE;i++){
             if((float)mask[i][j][k]==phasein){
                 if(res[i][j][k]>vcrit){
                        res[i][j][k]=phaseout;
                 }
                 else{
                        res[i][j][k]=phasein;
                 }
             }
             else{
                 res[i][j][k]=(float)mask[i][j][k];
             }
             fprintf(outfile,"%2d\n",(int)res[i][j][k]);
        }
        }
        }

        fclose(outfile);
}
```

B.4 Listing for stat3d.c

```
/*********************************************************************/
/*                                                                   */
/*      Program: stat3d.c                                            */
/*      Purpose: To read in a 3-D image and output phase volumes     */
/*               and report the volume and pore-exposed surface area */
/*               fractions                                           */
/*               Only processes the phase values 0-10, 24 and 25     */
/*      Programmer: Dale P. Bentz                                    */
/*                  NIST                                             */
/*                  100 Bureau Drive Mail Stop 8621                  */
/*                  Gaithersburg, MD  20899-0001                     */
/*                  Phone: (301) 975-5865                            */
/*                  E-mail: dale.bentz@.nist.gov                     */
/*                                                                   */
/*********************************************************************/
#include <stdio.h>
#include <math.h>

#define ISIZE 100


main(){
        static int mic [ISIZE] [ISIZE] [ISIZE];
        int valin,ix,iy,iz;
        int ix1,iy1,iz1,k;
        long int voltot,surftot,volume[37],surface [37];
        FILE *infile,*statfile;
        char filen[80],fileout[80];

        printf("Enter name of file to open \n");
        scanf("%s",filen);
        printf("%s \n",filen);
        printf("Enter name of file to write statistics to \n");
        scanf("%s",fileout);
        printf("%s \n",fileout);

        for(ix=0;ix<=36;ix++){
                volume[ix]=surface[ix]=0;
        }

        infile=fopen(filen,"r");
        statfile=fopen(fileout,"w");

/* Read in image and accumulate volume totals */
        for(iz=0;iz<ISIZE;iz++){
        for(iy=0;iy<ISIZE;iy++){
```

```
for(ix=0;ix<ISIZE;ix++){
        fscanf(infile,"%d",&valin);
        mic [ix] [iy] [iz]=valin;
        volume[valin]+=1;
}
}
}

fclose(infile);

for(iz=0;iz<ISIZE;iz++){
for(iy=0;iy<ISIZE;iy++){
for(ix=0;ix<ISIZE;ix++){
        if(mic [ix] [iy] [iz]!=0){
                valin=mic [ix] [iy] [iz];
        /* Check six neighboring pixels for porosity */
                for(k=1;k<=6;k++){

                        switch (k){
                               case 1:
                                       ix1=ix-1;
                                       if(ix1<0){ix1+=ISIZE;}
                                       iy1=iy;
                                       iz1=iz;
                                       break;
                               case 2:
                                       ix1=ix+1;
                                       if(ix1>=ISIZE){ix1-=ISIZE;}
                                       iy1=iy;
                                       iz1=iz;
                                       break;
                               case 3:
                                       iy1=iy-1;
                                       if(iy1<0){iy1+=ISIZE;}
                                       ix1=ix;
                                       iz1=iz;
                                       break;
                               case 4:
                                       iy1=iy+1;
                                       if(iy1>=ISIZE){iy1-=ISIZE;}
                                       ix1=ix;
                                       iz1=iz;
                                       break;
                               case 5:
                                       iz1=iz-1;
                                       if(iz1<0){iz1+=ISIZE;}
                                       iy1=iy;
```

```
                                        ix1=ix;
                                        break;
                                case 6:
                                        iz1=iz+1;
                                        if(iz1>=ISIZE){iz1-=ISIZE;}
                                        iy1=iy;
                                        ix1=ix;
                                        break;
                                default:
                                        break;
                                }
        if((ix1<0)||(iy1<0)||(iz1<0)||(ix1>=ISIZE)||(iy1>=ISIZE)||(iz1>=ISIZE)){
                                        printf("%d %d %d \n",ix1,iy1,iz1);
                                        exit(1);
                                }
                                if(mic[ix1] [iy1] [iz1]==0){
                                        surface[valin]+=1;
                                }
                        }
                }
        }
        }
        }

        printf("Phase    Volume      Surface     Volume    Surface \n");
        printf(" ID      count        count      fraction  fraction \n");
     fprintf(statfile,"Phase    Volume      Surface     Volume    Surface \n");
    fprintf(statfile," ID      count        count      fraction  fraction \n");
/* Only include clinker phases in surface area fraction calculation */
        surftot=surface[1]+surface[2]+surface[3]+surface[4];
        voltot=volume[1]+volume[2]+volume[3]+volume[4];
        k=0;
        printf("  %d    %8ld     %8ld  \n",k,volume[0],surface[0]);
        fprintf(statfile,"  %d    %8ld     %8ld  \n",k,volume[0],surface[0]);
        for(k=1;k<=4;k++){
        printf("  %d    %8ld     %8ld     %.5f   %.5f\n",k,volume[k],surface[k],
              (float)volume[k]/(float)voltot,(float)surface[k]/(float)surftot);
        fprintf(statfile,"  %d    %8ld     %8ld     %.5f   %.5f\n",k,volume[k],
surface[k],(float)volume[k]/(float)voltot,(float)surface[k]/(float)surftot);
        }
        printf("Total  %8ld     %8ld\n\n\n\n",voltot,surftot);
        fprintf(statfile,"Total  %8ld     %8ld\n\n\n\n",voltot,surftot);
        for(k=5;k<10;k++){
               printf("  %d    %8ld     %8ld\n",k,volume[k],surface[k]);
             fprintf(statfile,"  %d    %8ld     %8ld\n",k,volume[k],surface[k]);
        }
        for(k=24;k<=25;k++){
```

```
            printf(" %d    %8ld     %8ld\n",k,volume[k],surface[k]);
          fprintf(statfile," %d    %8ld     %8ld\n",k,volume[k],surface[k]);
        }
}
```

B.5 Listing for sinter3d.c

```
/*************************************************************/
/*                                                           */
/*      Program sinter3d.c                                   */
/*      To simulate 3-D curvature controlled sintering       */
/*      of a loosely packed powder                           */
/*      This version to sinter between two phases in         */
/*      a multi-phase system                                 */
/*      This version is periodic in all three directions     */
/*      Application: Modify local structure of generated     */
/*      3-D porous media by matching a specific hydraulic    */
/*      radius                                               */
/*                                                           */
/*      Programmer: Dale P. Bentz                            */
/*                  NIST                                     */
/*                  100 Bureau Drive Mail Stop 8621          */
/*                  Gaithersburg, MD  20899-8621             */
/*                  Phone: (301) 975-5865                    */
/*                  E-mail: dale.bentz@.nist.gov             */
/*      Date: Summer 1994                                    */
/*                                                           */
/*************************************************************/

#include <stdio.h>
#include <math.h>

#define SYSSIZE 100
#define MAXCYC 1000    /* maximum sintering cycles to use */
#define MAXSPH 10000   /* maximum number of elements in a spherical template */

/* array phase stores the microstructure representation */
/* array curvature stores the local curvature values */
static unsigned short int phase [SYSSIZE+1] [SYSSIZE+1] [SYSSIZE+1];
static unsigned short int curvature [SYSSIZE+1] [SYSSIZE+1] [SYSSIZE+1];
int nsph,xsph[MAXSPH],ysph[MAXSPH],zsph[MAXSPH];
int *seed;
long int nsolid[500],nair[500];

/* Random number generator ran1 from Computers in Physics */
/* Volume 6 No. 5, 1992, 522-524, Press and Teukolsky */
/* To generate real random numbers 0.0-1.0 */
/* Should be seeded with a negative integer */
#define IA 16807
#define IM 2147483647
#define IQ 127773
#define IR 2836
#define NTAB 32
```

```
#define EPS (1.2E-07)
#define MAX(a,b) (a>b)?a:b
#define MIN(a,b) (a<b)?a:b

double ran1(idum)
int *idum;
/* Calls: no routines */
/* Called by: gsphere,makefloc */
{
        int j,k;
        static int iv[NTAB],iy=0;
        void nrerror();
        static double NDIV = 1.0/(1.0+(IM-1.0)/NTAB);
        static double RNMX = (1.0-EPS);
        static double AM = (1.0/IM);

        if ((*idum <= 0) || (iy == 0)) {
                *idum = MAX(-*idum,*idum);
                for(j=NTAB+7;j>=0;j--) {
                        k = *idum/IQ;
                        *idum = IA*(*idum-k*IQ)-IR*k;
                        if(*idum < 0) *idum += IM;
                        if(j < NTAB) iv[j] = *idum;
                }
                iy = iv[0];
        }
        k = *idum/IQ;
        *idum = IA*(*idum-k*IQ)-IR*k;
        if(*idum<0) *idum += IM;
        j = iy*NDIV;
        iy = iv[j];
        iv[j] = *idum;
        return MIN(AM*iy,RNMX);
}
#undef IA
#undef IM
#undef IQ
#undef IR
#undef NTAB
#undef EPS
#undef MAX
#undef MIN

/* routine to create a template for the sphere of interest of radius size */
/* to be used in curvature evaluation */
/* Called by: runsint */
/* Calls no other routines */
```

```
int maketemp(size)
        int size;
{
        int icirc,xval,yval,zval;
        float xtmp,ytmp;
        float dist;

/* determine and store the locations of all pixels in the 3-D sphere */
        icirc=0;
        for(xval=(-size);xval<=size;xval++){
                xtmp=(float)(xval*xval);
        for(yval=(-size);yval<=size;yval++){
                ytmp=(float)(yval*yval);
        for(zval=(-size);zval<=size;zval++){
                dist=sqrt(xtmp+ytmp+(float)(zval*zval));
                if(dist<=((float)size+0.5)){
                        icirc+=1;
                        if(icirc>=MAXSPH){
                            printf("Too many elements in sphere \n");
                            printf("Must change value of MAXSPH parameter \n");
                            printf("Currently set at %d \n",MAXSPH);
                            exit(1);
                        }
                        xsph[icirc]=xval;
                        ysph[icirc]=yval;
                        zsph[icirc]=zval;
                }
        }
        }
        }

/* return the number of pixels contained in sphere of radius (size+0.5) */
        return(icirc);
}

/* routine to count phase fractions (porosity and solids) */
/* Called by main routine */
/* Calls no other routines */
void phcount()
{
        long int npore,nsolid [37];
        int ix,iy,iz;

        npore=0;
        for(ix=1;ix<37;ix++){
                nsolid[ix]=0;
        }
```

```
        /* check all pixels in the 3-D system */
        for(ix=0;ix<SYSSIZE;ix++){
        for(iy=0;iy<SYSSIZE;iy++){
        for(iz=0;iz<SYSSIZE;iz++){
                if(phase [ix] [iy] [iz]==0){
                        npore+=1;
                }
                else{
                        nsolid[phase [ix] [iy] [iz]]+=1;
                }
        }
        }
        }

        printf("Pores are: %ld \n",npore);
        printf("Solids are: %ld %ld %ld %ld %ld %ld\n",nsolid[1],nsolid[2],
             nsolid[3],nsolid[4],nsolid[5],nsolid[6]);
}

/* routine to return number of surface faces exposed to porosity */
/* for pixel located at (xin,yin,zin) */
/* Called by rhcalc */
/* Calls no other routines */
int surfpix(xin,yin,zin)
        int xin,yin,zin;
{
        int npix,ix1,iy1,iz1;

        npix=0;

        /* check each of the six immediate neighbors */
        /* using periodic boundary conditions */
        ix1=xin-1;
        if(ix1<0){ix1+=SYSSIZE;}
        if(phase[ix1][yin][zin]==0){
                npix+=1;
        }
        ix1=xin+1;
        if(ix1>=SYSSIZE){ix1-=SYSSIZE;}
        if(phase[ix1][yin][zin]==0){
                npix+=1;
        }
        iy1=yin-1;
        if(iy1<0){iy1+=SYSSIZE;}
        if(phase[xin][iy1][zin]==0){
                npix+=1;
        }
```

```
        iy1=yin+1;
        if(iy1>=SYSSIZE){iy1-=SYSSIZE;}
        if(phase[xin][iy1][zin]==0){
                npix+=1;
        }
        iz1=zin-1;
        if(iz1<0){iz1+=SYSSIZE;}
        if(phase[xin][yin][iz1]==0){
                npix+=1;
        }
        iz1=zin+1;
        if(iz1>=SYSSIZE){iz1-=SYSSIZE;}
        if(phase[xin][yin][iz1]==0){
                npix+=1;
        }
        return(npix);
}

/* routine to return the current hydraulic radius for phase phin */
/* Calls surfpix */
/* Called by runsint */
float rhcalc(phin)
        int phin;
{
        int ix,iy,iz;
        long int porc,surfc;
        float rhval;

        porc=surfc=0;

        /* Check all pixels in the 3-D volume */
        for(ix=0;ix<SYSSIZE;ix++){
        for(iy=0;iy<SYSSIZE;iy++){
        for(iz=0;iz<SYSSIZE;iz++){
                if(phase [ix] [iy] [iz]==phin){
                        porc+=1;
                        surfc+=surfpix(ix,iy,iz);
                }
        }
        }
        }

        printf("Phase area count is %ld \n",porc);
        printf("Phase surface count is %ld \n",surfc);
        rhval=(float)porc*6./(4.*(float)surfc);
        printf("Hydraulic radius is %f \n",rhval);
        return(rhval);
```

```
}

/* routine to return count of pixels in a spherical template which are phase */
/* phin or porosity (phase=0) */
/* Calls no other routines */
/* Called by sysinit */
int countem(xp,yp,zp,phin)
        int xp,yp,zp,phin;
{
        int xc,yc,zc;
        int cumnum,ic;

        cumnum=0;
        for(ic=1;ic<=nsph;ic++){
                xc=xp+xsph[ic];
                yc=yp+ysph[ic];
                zc=zp+zsph[ic];
                /* Use periodic boundaries */
                if(xc<0){xc+=SYSSIZE;}
                else if(xc>=SYSSIZE){xc-=SYSSIZE;}
                if(yc<0){yc+=SYSSIZE;}
                else if(yc>=SYSSIZE){yc-=SYSSIZE;}
                if(zc<0){zc+=SYSSIZE;}
                else if(zc>=SYSSIZE){zc-=SYSSIZE;}

                if((xc!=xp)||(yc!=yp)||(zc!=zp)){

                if((phase [xc] [yc] [zc]==phin)||(phase [xc] [yc] [zc]==0)){
                        cumnum+=1;
                }
                }
        }
        return(cumnum);
}

/* routine to initialize system by determining local curvature */
/* of all phase 1 and phase 2 pixels */
/* Calls countem */
/* Called by runsint */
void sysinit(ph1,ph2)
        int ph1,ph2;
{
        int count,xl,yl,zl;

        count=0;
        /* process all pixels in the 3-D box */
        for(xl=0;xl<SYSSIZE;xl++){
```

```
        for(yl=0;yl<SYSSIZE;yl++){
        for(zl=0;zl<SYSSIZE;zl++){

                /* determine local curvature */
                /* For phase 1 want to determine number of porosity pixels */
                /* (phase=0) in immediate neighborhood */
                if(phase [xl] [yl] [zl]==ph1){
                        count=countem(xl,yl,zl,0);
                }
                /* For phase 2 want to determine number of porosity or phase */
                /* 2 pixels in immediate neighborhood */
                if(phase [xl] [yl] [zl]==ph2){
                        count=countem(xl,yl,zl,ph2);
                }
                if((count<0)||(count>=nsph)){
                        printf("Error count is %d \n",count);
                        printf("xl %d  yl %d  zl %d \n",xl,yl,zl);
                }

                /* case where we have a phase 1 surface pixel */
                /* with non-zero local curvature */
                if((count>=0)&&(phase [xl] [yl] [zl]==ph1)){

                        curvature [xl] [yl] [zl]=count;
                        /* update solid curvature histogram */
                        nsolid[count]+=1;
                }

                /* case where we have a phase 2 surface pixel */
                if((count>=0)&&(phase [xl] [yl] [zl]==ph2)){

                        curvature [xl] [yl] [zl]=count;
                        /* update air curvature histogram */
                        nair[count]+=1;
                }

        }
        }
        } /* end of xl loop */
}

/* routine to scan system and determine nsolid (ph2) and nair (ph1) */
/* histograms based on values in phase and curvature arrays */
/* Calls no other routines */
/* Called by runsint */
void sysscan(ph1,ph2)
        int ph1,ph2;
```

```
{
        int xd,yd,zd,curvval;

        /* Scan all pixels in 3-D system */
        for(xd=0;xd<SYSSIZE;xd++){
        for(yd=0;yd<SYSSIZE;yd++){
        for(zd=0;zd<SYSSIZE;zd++){

                curvval=curvature [xd] [yd] [zd];

                if(phase [xd] [yd] [zd]==ph2){
                        nair[curvval]+=1;
                }
                else if (phase [xd] [yd] [zd]==ph1){
                        nsolid[curvval]+=1;
                }
        }
        }
        }
}

/* routine to return how many cells of solid curvature histogram to use */
/* to accomodate nsearch pixels moving */
/* want to use highest values first */
/* Calls no other routines */
/* Called by movepix */
int procsol(nsearch)
        int nsearch;
{
        int valfound,i,stop;
        long int nsofar;

        /* search histogram from top down until cumulative count */
        /* exceeds nsearch */
        valfound=nsph-1;
        nsofar=0;
        stop=0;
        for(i=(nsph-1);((i>=0)&&(stop==0));i--){
                nsofar+=nsolid[i];
                if(nsofar>nsearch){
                        valfound=i;
                        stop=1;
                }
        }
        return(valfound);
}
```

```
/* routine to determine how many cells of air curvature histogram to use */
/* to accomodate nsearch moving pixels */
/* want to use lowest values first */
/* Calls no other routines */
/* Called by movepix */
int procair(nsearch)
        int nsearch;
{
        int valfound,i,stop;
        long int nsofar;

        /* search histogram from bottom up until cumulative count */
        /* exceeds nsearch */
        valfound=0;
        nsofar=0;
        stop=0;
        for(i=0;((i<nsph)&&(stop==0));i++){
                nsofar+=nair[i];
                if(nsofar>nsearch){
                        valfound=i;
                        stop=1;
                }
        }
        return(valfound);
}

/* routine to move requested number of pixels (ntomove) from highest */
/* curvature phase 1 (ph1) sites to lowest curvature phase 2 (ph2) sites */
/* Calls procsol and procair */
/* Called by runsint */
int movepix(ntomove,ph1,ph2)
        int ntomove,ph1,ph2;
{
        int xloc[2100],yloc[2100],zloc[2100];
        int count1,count2,ntot,countc,i,xp,yp,zp;
        int cmin,cmax,cfg;
        int alldone;
        long int nsolc,nairc,nsum,nsolm,nairm,nst1,nst2,next1,next2;
        float pck,plsol,plair;

        alldone=0;
        /* determine critical values for removal and placement */
        count1=procsol(ntomove);
        nsum=0;
        cfg=0;
        cmax=count1;
        for(i=nsph;i>count1;i--){
```

```
        if((nsolid[i]>0)&&(cfg==0)){
                cfg=1;
                cmax=i;
        }
        nsum+=nsolid[i];
}
/* Determine movement probability for last cell */
plsol=(float)(ntomove-nsum)/(float)nsolid[count1];
next1=ntomove-nsum;
nst1=nsolid[count1];

count2=procair(ntomove);
nsum=0;
cmin=count2;
cfg=0;
for(i=0;i<count2;i++){
        if((nair[i]>0)&&(cfg==0)){
                cfg=1;
                cmin=i;
        }
        nsum+=nair[i];
}
/* Determine movement probability for last cell */
plair=(float)(ntomove-nsum)/(float)nair[count2];
next2=ntomove-nsum;
nst2=nair[count2];

/* Check to see if equilibrium has been reached --- */
/* no further increase in hydraulic radius is possible */
if(cmin>=cmax){
        alldone=1;
        printf("Stopping - at equilibrium \n");
        printf("cmin- %d  cmax- %d \n",cmin,cmax);
        return(alldone);
}

/* initialize counters for performing sintering */
ntot=0;
nsolc=0;
nairc=0;
nsolm=0;
nairm=0;

/* Now process each pixel in turn */
for(xp=0;xp<SYSSIZE;xp++){
for(yp=0;yp<SYSSIZE;yp++){
for(zp=0;zp<SYSSIZE;zp++){
```

```
countc=curvature [xp] [yp] [zp];
/* handle phase 1 case first */
if(phase [xp] [yp] [zp]==ph1){
        if(countc>count1){
                /* convert from phase 1 to phase 2 */
                phase [xp] [yp] [zp]=ph2;

                /* update appropriate histogram cells */
                nsolid[countc]-=1;
                nair[countc]+=1;
                /* store the location of the modified pixel */
                ntot+=1;
                xloc[ntot]=xp;
                yloc[ntot]=yp;
                zloc[ntot]=zp;
        }
        if(countc==count1){
                nsolm+=1;
             /* generate probability for pixel being removed */
                pck=ran1(seed);
                if((pck<0)||(pck>1.0)){pck=1.0;}

 if(((pck<plsol)&&(nsolc<next1))||((nst1-nsolm)<(next1-nsolc))){
                        nsolc+=1;
                        /* convert phase 1 pixel to phase 2 */
                        phase [xp] [yp] [zp]=ph2;

                        /* update appropriate histogram cells */
                        nsolid[count1]-=1;
                        nair[count1]+=1;
                  /* store the location of the modified pixel */
                        ntot+=1;
                        xloc[ntot]=xp;
                        yloc[ntot]=yp;
                        zloc[ntot]=zp;
                 }
                 }
        }

        /* handle phase 2 case here */
        else if (phase [xp] [yp] [zp]==ph2){
                if(countc<count2){
                        /* convert phase 2 pixel to phase 1 */
                        phase [xp] [yp] [zp]=ph1;

                        nsolid[countc]+=1;
                        nair[countc]-=1;
```

```
                                        ntot+=1;
                                        xloc[ntot]=xp;
                                        yloc[ntot]=yp;
                                        zloc[ntot]=zp;
                                }
                                if(countc==count2){
                                        nairm+=1;
                                        pck=ran1(seed);
                                        if((pck<0)||(pck>1.0)){pck=1.0;}

            if(((pck<plair)&&(nairc<next2))||((nst2-nairm)<(next2-nairc))){
                                                nairc+=1;
                                                /* convert phase 2 to phase 1 */
                                                phase [xp] [yp] [zp]=ph1;

                                                nsolid[count2]+=1;
                                                nair[count2]-=1;
                                                ntot+=1;
                                                xloc[ntot]=xp;
                                                yloc[ntot]=yp;
                                                zloc[ntot]=zp;
                                        }
                                }
                        }

        } /* end of zp loop */
        } /* end of yp loop */
        } /* end of xloop */
        printf("ntot is %d \n",ntot);
        return(alldone);
}

/* routine to execute user input number of cycles of sintering algorithm */
/* Calls maketemp, rhcalc, sysinit, sysscan, and movepix */
/* Called by main routine */
void runsint()
{
        int natonce,ncyc,i,rade,j,rflag;
        int ph1id,ph2id,keepgo;
        long int curvsum1,curvsum2,pixsum1,pixsum2;
        float rhnow,rhtarget,avecurv1,avecurv2;

        /* initialize the solid and air count histograms */
        for(i=0;i<=499;i++){
                nsolid[i]=0;
                nair[i]=0;
        }
```

```
        /* Obtain needed user input */
        printf("Enter phases to execute sintering between \n");
        scanf("%d %d",&ph1id,&ph2id);
        printf("%d %d \n",ph1id,ph2id);
        printf("Enter number of pixels to move at once (e.g. 200)\n");
        scanf("%d",&natonce);
        printf("%d \n",natonce);
        printf("Enter target hydraulic radius \n");
        scanf("%f",&rhtarget);
        printf("%f \n",rhtarget);
printf("Enter sphere radius to use for surface energy calculation (e.g. 3)\n");
        scanf("%d",&rade);
        printf("%d \n",rade);
        rflag=0;   /* always initialize system */

        nsph=maketemp(rade);
        printf("nsph is %d \n",nsph);
        if(rflag==0){
                sysinit(ph1id,ph2id);
        }
        else{
                sysscan(ph1id,ph2id);
        }
        i=0;
        rhnow=rhcalc(ph1id);
        while((rhnow<rhtarget)&&(i<MAXCYC)){
                printf("Now: %f  Target: %f \n",rhnow,rhtarget);
                i+=1;
                printf("Cycle: %d \n",i);
                keepgo=movepix(natonce,ph1id,ph2id);
                /* If equilibrium is reached, then return to calling routine */
                if(keepgo==1){
                        return;
                }
                curvsum1=0;
                curvsum2=0;
                pixsum1=0;
                pixsum2=0;
                /* Determine average curvatures for phases 1 and 2 */
                for(j=0;j<=nsph;j++){
                        pixsum1+=nsolid[j];
                        curvsum1+=(j*nsolid[j]);
                        pixsum2+=nair[j];
                        curvsum2+=(j*nair[j]);
                }
                avecurv1=(float)curvsum1/(float)pixsum1;
                avecurv2=(float)curvsum2/(float)pixsum2;
```

```
                printf("Ave. solid curvature: %f \n",avecurv1);
                printf("Ave. air curvature: %f \n",avecurv2);
                rhnow=rhcalc(ph1id);
        }
}

/* routine to read in microstructure from a file */
/* Calls no other routines */
/* Called by main routine */
void readmic()
{
        FILE *infile;
        char filen[80];
        int iout,i1,i2,i3;
        long int porcnt,totcnt;

        porcnt=totcnt=0;
        printf("Enter name of file to read in \n");
        scanf("%s",filen);
        infile=fopen(filen,"r");

        for(i3=0;i3<SYSSIZE;i3++){
        for(i2=0;i2<SYSSIZE;i2++){
        for(i1=0;i1<SYSSIZE;i1++){
                fscanf(infile,"%d",&iout);
                totcnt+=1;
                phase [i1] [i2] [i3]=iout;
                if(iout==0){
                        porcnt+=1;
                }
        }
        }
        }
        printf("Count for porosity is %ld out of %ld \n",porcnt,totcnt);
        fclose(infile);
}

/* routine to output microstructure to file */
/* Calls no other routines */
/* Called by main routine */
void savemic()
{
        FILE *outfile;
        int iout,i1,i2,i3;
        long int porcnt,totcnt;
        char fileout[80];
```

```
        porcnt=totcnt=0;
        printf("Enter name of file to write to \n");
        scanf("%s",fileout);
        outfile=fopen(fileout,"w");

        for(i3=0;i3<SYSSIZE;i3++){
        for(i2=0;i2<SYSSIZE;i2++){
        for(i1=0;i1<SYSSIZE;i1++){
                totcnt+=1;
                iout=phase [i1] [i2] [i3];
                if(iout==0){
                       porcnt+=1;
                }
                fprintf(outfile,"%d\n",iout);
        }
        }
        }
        printf("Count for porosity is %ld out of %ld \n",porcnt,totcnt);
        fclose(outfile);
}

/* Calls phcount, runsint, readmic, and savemic */
main()
{
        int nseed,menuch;

        printf("Enter random number seed (integer < 0): \n");
        scanf("%d",&nseed);
        printf("%d \n",nseed);
        seed=(&nseed);

        menuch=6;
        /* Present menu and obtain user choice */
        while(menuch!=1){
                printf("Enter choice: \n");
                printf("1) Exit program \n");
                printf("2) Read in microstructure from file \n");
                printf("3) Measure phase fractions \n");
                printf("4) Perform sintering algorithm \n");
                printf("5) Output resultant microstructure to file \n");
                scanf("%d",&menuch);
                printf("%d \n",menuch);

                switch(menuch){
                        case 2:
                                readmic();
                                break;
```

```
                        case 3:
                                phcount();
                                break;
                        case 4:
                                runsint();
                                break;
                        case 5:
                                savemic();
                                break;
                        default:
                                break;
                }
        }
}
```

B.6 Listing for oneimage.c

```
/*************************************************************/
/*                                                           */
/*      Program oneimage.c                                   */
/*      To create a 2-D raw image file from one slice        */
/*      of a 3-D model microstructure                        */
/*                                                           */
/*      Programmer: Dale P. Bentz                            */
/*                  NIST                                     */
/*                  100 Bureau Drive Mail Stop 8621          */
/*                  Gaithersburg, MD  20899-8621             */
/*                  Phone: (301) 975-5865                    */
/*                  E-mail: dale.bentz@.nist.gov             */
/*      Date: Summer 1999                                    */
/*                                                           */
/*************************************************************/

#include <stdio.h>
#include <math.h>

main(){
       FILE *infile,*outfile;
        char filein[80],fileout[80];
        int valin;
        int valout,i1,j1;
        int i,nskip,dx,dy,j,scf,iscale,dxtot,dytot;
        static short int image[1000][1000];
        static short int red[50],green[50],blue[50];

        for(i1=0;i1<50;i1++){
                red[i1]=0;
                green[i1]=0;
                blue[i1]=0;
        }
        for(i1=10;i1<24;i1++){
                red[i1]=192;
                green[i1]=255;
                blue[i1]=128;
        }
        red[12]=0;
        green[12]=0;
        blue[12]=255;
        red[13]=255;
        green[13]=128;
        blue[13]=0;
        red[1]=255;
        green[2]=255;
```

```
blue[2]=255;
green[3]=255;
red[4]=255;
green[4]=255;
red[5]=128;
green[5]=128;
blue[5]=128;
red[6]=64;
green[6]=64;
blue[6]=64;
red[7]=192;
green[7]=192;
blue[7]=192;
red[8]=255;
green[8]=128;
blue[8]=64;
red[9]=128;
green[9]=128;
blue[9]=255;
red[24]=255;
blue[24]=255;
red[35]=255;
green[35]=255;
blue[35]=255;
red[38]=255;
green[38]=255;
blue[38]=255;

printf("Enter name of file with raw (3-D image) data \n");
scanf("%s",filein);
printf("%s\n",filein);
printf("Enter name of image file to create \n");
scanf("%s",fileout);
printf("%s\n",fileout);

printf("Enter number of pixels to skip at start of image file \n");
scanf("%d",&nskip);
printf("%d\n",nskip);
dx=dy=100;

printf("Enter factor to scale image by\n");
scanf("%d",&iscale);
printf("%d\n",iscale);

infile=fopen(filein,"r");
outfile=fopen(fileout,"w");
```

```
        for(i=0;i<nskip;i++){
                fscanf(infile,"%d",&valin);
        }

        fprintf(outfile,"P3\n");
        fprintf(outfile,"%d %d\n",dx*iscale,dy*iscale);
        fprintf(outfile,"255\n");
        dxtot=dx*iscale;
        dytot=dy*iscale;
        for(j=0;j<dy;j++){
        for(i=0;i<dx;i++){

                fscanf(infile,"%d",&valin);
                valout=valin;
                for(j1=0;j1<iscale;j1++){
                for(i1=0;i1<iscale;i1++){
                        image[i*iscale+i1][j*iscale+j1]=valout;
                }
                }
        }
        }
        fclose(infile);

        for(j=0;j<dytot;j++){
        for(i=0;i<dxtot;i++){
            fprintf(outfile,"%d %d %d\n",red[image[i][j]],green[image[i][j]],
            blue[image[i][j]]);
        }
        }

        fclose(outfile);
}
```

C Computer programs for three-dimensional cement hydration model

C.1 Code for assessing percolation of pore space

```
#define BURNT 70      /* label for a burnt pixel */
#define SIZE2D 49000 /* size of matrices for holding burning locations */
/* functions defining coordinates for burning in any of three directions */
#define cx(x,y,z,a,b,c) (1-b-c)*x+(1-a-c)*y+(1-a-b)*z
#define cy(x,y,z,a,b,c) (1-a-b)*x+(1-b-c)*y+(1-a-c)*z
#define cz(x,y,z,a,b,c) (1-a-c)*x+(1-a-b)*y+(1-b-c)*z

/* routine to assess the connectivity (percolation) of a single phase */
/* Two matrices are used here: one to store the recently burnt locations */
/*                the other to store the newly found burnt locations */
int burn3d(npix,d1,d2,d3)
        int npix;    /* ID of phase to perform burning on */
        int d1,d2,d3;   /* directional flags */
{
        long int ntop,nthrough,ncur,nnew,ntot,nphc;
        int i,inew,j,k,nmatx[SIZE2D],nmaty[SIZE2D],nmatz[SIZE2D];
        int xl,xh,j1,k1,px,py,pz,qx,qy,qz,xcn,ycn,zcn;
        int x1,y1,z1,igood,nnewx[SIZE2D],nnewy[SIZE2D],nnewz[SIZE2D];
        int jnew,icur;
        int bflag;
        float mass_burn=0.0,alpha_burn=0.0;
        FILE *fileperc;

/* counters for number of pixels of phase accessible from surface #1 */
/* and number which are part of a percolated pathway to surface #2 */
        ntop=0;
        bflag=0;
        nthrough=0;
        nphc=0;

        /* percolation is assessed from top to bottom only */
        /* and burning algorithm is periodic in other two directions */
        /* use of directional flags allow transformation of coordinates */
        /* to burn in direction of choosing (x, y, or z) */
        i=0;

        for(k=0;k<SYSIZE;k++){
        for(j=0;j<SYSIZE;j++){

                igood=0;
                ncur=0;
                ntot=0;
```

```
/* Transform coordinates */
px=cx(i,j,k,d1,d2,d3);
py=cy(i,j,k,d1,d2,d3);
pz=cz(i,j,k,d1,d2,d3);
if(mic [px] [py] [pz]==npix){
        /* Start a burn front */
        mic [px] [py] [pz]=BURNT;
        ntot+=1;
        ncur+=1;
        /* burn front is stored in matrices nmat* */
        /* and nnew* */
        nmatx[ncur]=i;
        nmaty[ncur]=j;
        nmatz[ncur]=k;
        /* Burn as long as new (fuel) pixels are found */
        do{
                nnew=0;
                for(inew=1;inew<=ncur;inew++){
                        xcn=nmatx[inew];
                        ycn=nmaty[inew];
                        zcn=nmatz[inew];

                        /* Check all six neighbors */
                        for(jnew=1;jnew<=6;jnew++){
                                x1=xcn;
                                y1=ycn;
                                z1=zcn;
                                if(jnew==1){x1-=1;}
                                if(jnew==2){x1+=1;}
                                if(jnew==3){y1-=1;}
                                if(jnew==4){y1+=1;}
                                if(jnew==5){z1-=1;}
                                if(jnew==6){z1+=1;}
                                /* Periodic in y and */
                                if(y1>=SYSIZE){y1-=SYSIZE;}
                                else if(y1<0){y1+=SYSIZE;}
                                /* Periodic in z direction */
                                if(z1>=SYSIZE){z1-=SYSIZE;}
                                else if(z1<0){z1+=SYSIZE;}

        /* Nonperiodic so be sure to remain in the 3-D box */
                                if((x1>=0)&&(x1<SYSIZE)){
                                /* Transform coordinates */
                                px=cx(x1,y1,z1,d1,d2,d3);
                                py=cy(x1,y1,z1,d1,d2,d3);
                                pz=cz(x1,y1,z1,d1,d2,d3);
                                if(mic [px] [py] [pz]==npix){
```

```
                                        ntot+=1;
                                        mic [px] [py] [pz]=BURNT;
                                        nnew+=1;
                                        if(nnew>=SIZE2D){
                                   printf("error in size of nnew \n");
                                        }
                                        nnewx[nnew]=x1;
                                        nnewy[nnew]=y1;
                                        nnewz[nnew]=z1;
                                    }
                            }
                            }
                    }
                    if(nnew>0){
                            ncur=nnew;
                            /* update the burn front matrices */
                            for(icur=1;icur<=ncur;icur++){
                                    nmatx[icur]=nnewx[icur];
                                    nmaty[icur]=nnewy[icur];
                                    nmatz[icur]=nnewz[icur];
                            }
                    }
            }while (nnew>0);

            ntop+=ntot;
            xl=0;
            xh=SYSIZE-1;
         /* See if current path extends through the microstructure */
            for(j1=0;j1<SYSIZE;j1++){
            for(k1=0;k1<SYSIZE;k1++){
                    px=cx(xl,j1,k1,d1,d2,d3);
                    py=cy(xl,j1,k1,d1,d2,d3);
                    pz=cz(xl,j1,k1,d1,d2,d3);
                    qx=cx(xh,j1,k1,d1,d2,d3);
                    qy=cy(xh,j1,k1,d1,d2,d3);
                    qz=cz(xh,j1,k1,d1,d2,d3);
        if((mic [px] [py] [pz]==BURNT)&&(mic [qx] [qy] [qz]==BURNT)){
                            igood=2;
                    }
                    if(mic [px] [py] [pz]==BURNT){
                            mic [px] [py] [pz]=BURNT+1;
                    }
                    if(mic [qx] [qy] [qz]==BURNT){
                            mic [qx] [qy] [qz]=BURNT+1;
                    }
            }
            }
```

```
                        if(igood==2){
                                nthrough+=ntot;
                        }
                }
        }
        }
/* return the burnt sites to their original phase values */
        for(i=0;i<SYSIZE;i++){
        for(j=0;j<SYSIZE;j++){
        for(k=0;k<SYSIZE;k++){
                if(mic [i] [j] [k]>=BURNT){
                        nphc+=1;
                        mic [i] [j] [k]=npix;
                }
                else if(mic[i][j][k]==npix){
                        nphc+=1;
                }
        }
        }
        }

        printf("Phase ID= %d \n",npix);
        printf("Number accessible from first surface = %ld \n",ntop);
        printf("Number contained in through pathways= %ld \n",nthrough);
        fileperc=fopen("percpore.out","a");
        mass_burn+=specgrav[C3S]*count[C3S];
        mass_burn+=specgrav[C2S]*count[C2S];
        mass_burn+=specgrav[C3A]*count[C3A];
        mass_burn+=specgrav[C4AF]*count[C4AF];
        alpha_burn=1.-(mass_burn/cemmass);
        fprintf(fileperc,"%ld %f %ld %ld \n",cyccnt,alpha_burn,nthrough,nphc);
        fclose(fileperc);
        if(nthrough>0){
                bflag=1;
        }
        return(bflag);
}
```

C.2 Code for assessing percolation of total solids- set point

```
#define BURNT 70    /* label for burnt pixels */
#define SIZESET 100000
/* Transformation functions for changing direction of burn propagation */
#define cx(x,y,z,a,b,c) (1-b-c)*x+(1-a-c)*y+(1-a-b)*z
#define cy(x,y,z,a,b,c) (1-a-b)*x+(1-b-c)*y+(1-a-c)*z
#define cz(x,y,z,a,b,c) (1-a-c)*x+(1-a-b)*y+(1-b-c)*z

/* routine to assess connectivity (percolation) of solids for set estimation*/
/* Definition of set is a through pathway of cement particles connected */
/* together by CSH or ettringite */
/* Two matrices are used here: one to store the recently burnt locations */
/*                the other to store the newly found burnt locations */
int burnset(d1,d2,d3)
        int d1,d2,d3;
{
        long int ntop,nthrough,icur,inew,ncur,nnew,ntot;
        int i,j,k,setyet;
        static int nmatx[SIZESET],nmaty[SIZESET],nmatz[SIZESET];
        int xl,xh,j1,k1,px,py,pz,qx,qy,qz;
        int xcn,ycn,zcn,x1,y1,z1,igood;
        static int nnewx[SIZESET],nnewy[SIZESET],nnewz[SIZESET];
        int jnew;
        float mass_burn=0.0,alpha_burn=0.0;
        FILE *percfile;
        static char newmat [SYSIZE] [SYSIZE] [SYSIZE];

/* counters for number of pixels of phase accessible from surface #1 */
/* and number which are part of a percolated pathway to surface #2 */
        ntop=0;
        nthrough=0;
        setyet=0;
        for(k=0;k<SYSIZE;k++){
        for(j=0;j<SYSIZE;j++){
        for(i=0;i<SYSIZE;i++){
                newmat[i][j][k]=mic[i][j][k];
        }
        }
        }

        /* percolation is assessed from top to bottom only */
        /* in transformed coordinates */
        /* and burning algorithm is periodic in other two directions */
        i=0;

        for(k=0;k<SYSIZE;k++){
        for(j=0;j<SYSIZE;j++){
```

```
                igood=0;
                ncur=0;
                ntot=0;
                /* Transform coordinates */
                px=cx(i,j,k,d1,d2,d3);
                py=cy(i,j,k,d1,d2,d3);
                pz=cz(i,j,k,d1,d2,d3);
/* start from a cement clinker, ettringite, or CSH pixel */
        if((mic [px] [py] [pz]==C3S) ||
                (mic [px] [py] [pz]==C2S) ||
                (mic [px] [py] [pz]==CSH) ||
                (mic [px] [py] [pz]==ETTR) ||
                (mic [px] [py] [pz]==ETTRC4AF) ||
                (mic [px] [py] [pz]==C3A) ||
                (mic [px] [py] [pz]==C4AF)){
                        /* Start a burn front */
                        mic [px] [py] [pz]=BURNT;
                        ntot+=1;
                        ncur+=1;
                        /* burn front is stored in matrices nmat* */
                        /* and nnew* */
                        nmatx[ncur]=i;
                        nmaty[ncur]=j;
                        nmatz[ncur]=k;
                        /* Burn as long as new (fuel) pixels are found */
                        do{
                                nnew=0;
                                for(inew=1;inew<=ncur;inew++){
                                        xcn=nmatx[inew];
                                        ycn=nmaty[inew];
                                        zcn=nmatz[inew];
                                        /* Convert to directional coordinates */
                                        qx=cx(xcn,ycn,zcn,d1,d2,d3);
                                        qy=cy(xcn,ycn,zcn,d1,d2,d3);
                                        qz=cz(xcn,ycn,zcn,d1,d2,d3);

                                        /* Check all six neighbors */
                                        for(jnew=1;jnew<=6;jnew++){
                                                x1=xcn;
                                                y1=ycn;
                                                z1=zcn;
                                                if(jnew==1){x1-=1;}
                                                if(jnew==2){x1+=1;}
                                                if(jnew==3){y1-=1;}
                                                if(jnew==4){y1+=1;}
                                                if(jnew==5){z1-=1;}
                                                if(jnew==6){z1+=1;}
```

```
                                                /* Periodic in y and */
                                                if(y1>=SYSIZE){y1-=SYSIZE;}
                                                else if(y1<0){y1+=SYSIZE;}
                                                /* Periodic in z direction */
                                                if(z1>=SYSIZE){z1-=SYSIZE;}
                                                else if(z1<0){z1+=SYSIZE;}

/* Nonperiodic so be sure to remain in the 3-D box */
                                                if((x1>=0)&&(x1<SYSIZE)){
                                                px=cx(x1,y1,z1,d1,d2,d3);
                                                py=cy(x1,y1,z1,d1,d2,d3);
                                                pz=cz(x1,y1,z1,d1,d2,d3);
                /* Conditions for propagation of burning */
                /* 1) new pixel is CSH or ETTR */
                if((mic[px][py][pz]==CSH)||(mic[px][py][pz]==ETTRC4AF)
                      ||(mic[px][py][pz]==ETTR)){
                        ntot+=1;
                        mic [px] [py] [pz]=BURNT;
                        nnew+=1;
                        if(nnew>=SIZESET){
                              printf("error in size of nnew %d\n", nnew);
                        }
                        nnewx[nnew]=x1;
                        nnewy[nnew]=y1;
                        nnewz[nnew]=z1;
                }
/* 2) old pixel is CSH or ETTR and new pixel is one of cement clinker phases */
                else if(((newmat[qx][qy][qz]==CSH)||
                (newmat[qx][qy][qz]==ETTRC4AF)||(newmat[qx][qy][qz]==ETTR))
                &&((mic [px] [py] [pz]==C3S) ||
                (mic [px] [py] [pz]==C2S) ||
                (mic [px] [py] [pz]==C3A) ||
                (mic [px] [py] [pz]==C4AF))){
                                ntot+=1;
                                mic [px] [py] [pz]=BURNT;
                                nnew+=1;
                                if(nnew>=SIZESET){
                                   printf("error in size of nnew %d\n", nnew);
                                }
                                nnewx[nnew]=x1;
                                nnewy[nnew]=y1;
                                nnewz[nnew]=z1;
                }
        /* 3) old and new pixels belong to one of cement clinker phases and */
        /* are contained in the same initial cement particle */
                else if((micpart[qx][qy][qz]==micpart[px][py][pz])
                &&((mic [px] [py] [pz]==C3S) ||
```

```
(mic [px] [py] [pz]==C2S) ||
(mic [px] [py] [pz]==C3A) ||
(mic [px] [py] [pz]==C4AF))&&((newmat[qx][qy][qz]==C3S)||
(newmat [qx] [qy] [qz]==C2S) ||
(newmat [qx] [qy] [qz]==C3A) ||
(newmat[qx][qy][qz]==C4AF))){
                        ntot+=1;
                        mic [px] [py] [pz]=BURNT;
                        nnew+=1;
                        if(nnew>=SIZESET){
                    printf("error in size of nnew %d\n", nnew);
                        }
                        nnewx[nnew]=x1;
                        nnewy[nnew]=y1;
                        nnewz[nnew]=z1;
}
                       } /* nonperiodic if delimiter */
                       } /* neighbors loop */
                } /* propagators loop */
                if(nnew>0){
                        ncur=nnew;
                        /* update the burn front matrices */
                        for(icur=1;icur<=ncur;icur++){
                                nmatx[icur]=nnewx[icur];
                                nmaty[icur]=nnewy[icur];
                                nmatz[icur]=nnewz[icur];
                        }
                }
        }while (nnew>0);

        ntop+=ntot;
        xl=0;
        xh=SYSIZE-1;
        /* Check for percolated path through system */
        for(j1=0;j1<SYSIZE;j1++){
        for(k1=0;k1<SYSIZE;k1++){
                px=cx(xl,j1,k1,d1,d2,d3);
                py=cy(xl,j1,k1,d1,d2,d3);
                pz=cz(xl,j1,k1,d1,d2,d3);
                qx=cx(xh,j1,k1,d1,d2,d3);
                qy=cy(xh,j1,k1,d1,d2,d3);
                qz=cz(xh,j1,k1,d1,d2,d3);
  if((mic [px] [py] [pz]==BURNT)&&(mic [qx] [qy] [qz]==BURNT)){
                        igood=2;
                }
                if(mic [px] [py] [pz]==BURNT){
                        mic [px] [py] [pz]=BURNT+1;
```

```
                                }
                                if(mic [qx] [qy] [qz]==BURNT){
                                        mic [qx] [qy] [qz]=BURNT+1;
                                }
                        }
                        }

                        if(igood==2){
                                nthrough+=ntot;
                        }
                }
        }
        }

        printf("Phase ID= Solid Phases \n");
        printf("Number accessible from first surface = %ld \n",ntop);
        printf("Number contained in through pathways= %ld \n",nthrough);
        percfile=fopen("percset.out","a");
        mass_burn+=specgrav[C3S]*count[C3S];
        mass_burn+=specgrav[C2S]*count[C2S];
        mass_burn+=specgrav[C3A]*count[C3A];
        mass_burn+=specgrav[C4AF]*count[C4AF];
        alpha_burn=1.-(mass_burn/cemmass);
        fprintf(percfile,"%ld  %f  %ld %ld\n",cyccnt,alpha_burn,nthrough,
         count[C3S]+count[C2S]++count[C3A]+count[C4AF]+
         count[ETTR]+count[ETTRC4AF]+count[CSH]);
        fclose(percfile);
        if(nthrough>0){setyet=1;}

/* return the burnt sites to their original phase values */
        for(i=0;i<SYSIZE;i++){
        for(j=0;j<SYSIZE;j++){
        for(k=0;k<SYSIZE;k++){
                if(mic [i] [j] [k]>=BURNT){
                        mic [i] [j] [k]= newmat [i] [j] [k];
                }
        }
        }
        }
        /* Return flag indicating if set has indeed occurred */
        return(setyet);
}
```

C.3 Code for monitoring hydration of individual particles

```
/* Routine to assess relative particle hydration */
void parthyd(){
       int norig[50000],nleft[50000];
       int ix,iy,iz;
       char valmic,valmicorig;
       int valpart,partmax;
       float alpart;
       FILE *phydfile;

       /* Initialize the particle count arrays */
       for(ix=0;ix<50000;ix++){
              nleft[ix]=norig[ix]=0;
       }
       phydfile=fopen("partlist.hyd","a");
       fprintf(phydfile,"%d %f\n",cyccnt,alpha_cur);

       partmax=0;
       /* Scan the microstructure pixel by pixel and update counts */
       for(ix=0;ix<SYSIZE;ix++){
       for(iy=0;iy<SYSIZE;iy++){
       for(iz=0;iz<SYSIZE;iz++){

              if(micpart[ix][iy][iz]!=0){
                     valpart=micpart[ix][iy][iz];
                     if(valpart>partmax){partmax=valpart;}
                     valmic=mic[ix][iy][iz];
              if((valmic==C3S)||(valmic==C2S)||(valmic==C3A)||(valmic==C4AF)){
                            nleft[valpart]+=1;
                     }
                     valmicorig=micorig[ix][iy][iz];
                     if((valmicorig==C3S)||(valmicorig==C2S)||(valmicorig==C3A)
                         ||(valmicorig==C4AF)){
                            norig[valpart]+=1;
                      }
              }
       }
       }
       }

       /* Output results to end of particle hydration file */
       for(ix=100;ix<=partmax;ix++){
              alpart=0.0;
              if(norig[ix]!=0){
                    alpart=1.-(float)nleft[ix]/(float)norig[ix]);
              }
              fprintf(phydfile,"%d %d %d %.3f\n",ix,norig[ix],nleft[ix],alpart);
```

```
        }
        fclose(phydfile);
}
```

C.4 Code for random number generation

```
/* Random number generator ran1 from Computers in Physics */
/* Volume 6 No. 5, 1992, 522-524, Press and Teukolsky */
/* To generate real random numbers 0.0-1.0 */
/* Should be seeded with a negative integer */
#define IA 16807
#define IM 2147483647
#define IQ 127773
#define IR 2836
#define NTAB 32
#define EPS (1.2E-07)
#define MAX(a,b) (a>b)?a:b
#define MIN(a,b) (a<b)?a:b

double ran1(idum)
int *idum;
{
        int j,k;
        static int iv[NTAB],iy=0;
        void nrerror();
        static double NDIV = 1.0/(1.0+(IM-1.0)/NTAB);
        static double RNMX = (1.0-EPS);
        static double AM = (1.0/IM);

        if ((*idum <= 0) || (iy == 0)) {
                *idum = MAX(-*idum,*idum);
                for(j=NTAB+7;j>=0;j--) {
                        k = *idum/IQ;
                        *idum = IA*(*idum-k*IQ)-IR*k;
                        if(*idum < 0) *idum += IM;
                        if(j < NTAB) iv[j] = *idum;
                }
                iy = iv[0];
        }
        k = *idum/IQ;
        *idum = IA*(*idum-k*IQ)-IR*k;
        if(*idum<0) *idum += IM;
        j = iy*NDIV;
        iy = iv[j];
        iv[j] = *idum;
        return MIN(AM*iy,RNMX);
}
#undef IA
#undef IM
#undef IQ
#undef IR
#undef NTAB
```

```
#undef EPS
#undef MAX
#undef MIN
```

C.5 Listing for hydrealnew.c

```
#define AGRATE 0.25        /* Probability of gypsum absorption by CSH */

/* routine to select a new neighboring location to (xloc, yloc, zloc) */
/* for a diffusing species */
/* Returns a prime number flag indicating direction chosen */
/* Calls ran1 */
/* Called by movecsh, extettr, extfh3, movegyp, extafm, moveettr, */
/* extpozz, movefh3, movech, extc3ah6, movec3a */
/* extfreidel, movecacl2, extstrat, moveas */
int moveone(xloc,yloc,zloc,act,sumold)
        int *xloc,*yloc,*zloc,*act,sumold;
{
        int plok,sumnew,xl1,yl1,zl1,act1;

        sumnew=1;
        /* store the input values for location */
        xl1=(*xloc);
        yl1=(*yloc);
        zl1=(*zloc);
        act1=(*act);

        /* Choose one of six directions (at random) for the new */
        /* location */
        plok=6.*ran1(seed);
        if((plok>5)||(plok<0)){plok=5;}

        switch (plok){
                case 0:
                        xl1-=1;
                        act1=1;
                        if(xl1<0){xl1=(SYSIZEM1);}
                        if(sumold%2!=0){sumnew=2;}
                        break;
                case 1:
                        xl1+=1;
                        act1=2;
                        if(xl1>=SYSIZE){xl1=0;}
                        if(sumold%3!=0){sumnew=3;}
                        break;
                case 2:
                        yl1-=1;
                        act1=3;
                        if(yl1<0){yl1=(SYSIZEM1);}
                        if(sumold%5!=0){sumnew=5;}
                        break;
                case 3:
```

```
                        yl1+=1;
                        act1=4;
                        if(yl1>=SYSIZE){yl1=0;}
                        if(sumold%7!=0){sumnew=7;}
                        break;
                case 4:
                        zl1-=1;
                        act1=5;
                        if(zl1<0){zl1=(SYSIZEM1);}
                        if(sumold%11!=0){sumnew=11;}
                        break;
                case 5:
                        zl1+=1;
                        act1=6;
                        if(zl1>=SYSIZE){zl1=0;}
                        if(sumold%13!=0){sumnew=13;}
                        break;
                default:
                        break;
        }

        /* Return the new location */
        *xloc=xl1;
        *yloc=yl1;
        *zloc=zl1;
        *act=act1;
        /* sumnew returns a prime number indicating that a specific direction */
        /* has been chosen */
        return(sumnew);
}

/* routine to return count of number of neighboring pixels for pixel */
/* (xck,yck,zck) which are not phase ph1, ph2, or ph3 which are input as */
/* parameters */
/* Calls no other routines */
/* Called by extettr, extfh3, extch, extafm, extpozz, extc3ah6 */
/* extfreidel, extcsh, and extstrat */
int edgecnt(xck,yck,zck,ph1,ph2,ph3)
        int xck,yck,zck,ph1,ph2,ph3;
{
        int ixe,iye,ize,edgeback,x2,y2,z2,check;

/* counter for number of neighboring pixels which are not ph1, ph2, or ph3 */
        edgeback=0;

/* Examine all pixels in a 3*3*3 box centered at (xck,yck,zck) */
/* except for the central pixel */
```

```
        for(ixe=(-1);ixe<=1;ixe++){
                x2=xck+ixe;
        for(iye=(-1);iye<=1;iye++){
                y2=yck+iye;
        for(ize=(-1);ize<=1;ize++){

                 if((ixe!=0)||(iye!=0)||(ize!=0)){
                        z2=zck+ize;
                        /* adjust to maintain periodic boundaries */
                        if(x2<0){x2=(SYSIZEM1);}
                        else if(x2>=SYSIZE){x2=0;}
                        if(y2<0){y2=(SYSIZEM1);}
                        else if(y2>=SYSIZE){y2=0;}
                        if(z2<0){z2=(SYSIZEM1);}
                        else if(z2>=SYSIZE){z2=0;}
                        check=mic[x2][y2][z2];
                        if((check!=ph1)&&(check!=ph2)&&(check!=ph3)){
                                  edgeback+=1;
                        }
                }
        }
        }
        }
        /* return number of neighboring pixels which are not ph1, ph2, or ph3 */
        return(edgeback);
}

/* routine to add extra CSH when diffusing CSH reacts */
/* Called by movecsh */
/* Calls edgecnt */
void extcsh()
{
        int numnear,sump,xchr,ychr,zchr,fchr,i1,plok,check;
        long int tries;

        fchr=0;
        tries=0;
        /* locate CSH at random location */
        /* in pore space in contact with at least another CSH or C3S or C2S */
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
```

```
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];

                /* if location is porosity, locate the CSH there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,CSH,C3S,C2S);
                        /* be sure that at least one neighboring pixel */
                        /* is C2S, C3S, or diffusing CSH */
                        if((numnear<26)||(tries>5000)){
                                mic[xchr][ychr][zchr]=CSH;
                                count[CSH]+=1;
                                count[POROSITY]-=1;
                                cshage[xchr][ychr][zchr]=cyccnt;
                                fchr=1;
                        }
                }
        }
}

/* routine to move a diffusing CSH species */
/* Inputs: current location (xcur,ycur,zcur) and flag indicating if final */
/* step in diffusion process */
/* Returns flag indicating action taken (reaction or diffusion/no movement) */
/* Calls moveone,extcsh */
/* Called by hydrate */
int movecsh(xcur,ycur,zcur,finalstep,cycorig)
        int xcur,ycur,zcur,finalstep,cycorig;
{
        int xnew,ynew,znew,plok,action,sumback,sumin,check;
        float prcsh,prtest;

        action=0;
        /* Store current location of species */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        sumin=1;
        sumback=moveone(&xnew,&ynew,&znew,&action,sumin);

        if(action==0){printf("Error in value of action \n");}
        check=mic[xnew][ynew][znew];

/* if new location is solid C3S, C2S, or CSH, then convert */
/* diffusing CSH species to solid CSH */
        if((check==C3S)||(check==C2S)||(check==CSH)||(finalstep==1)){
                /* decrement count of diffusing CSH species */
                count[DIFFCSH]-=1;
```

```
                /* and increment count of solid CSH if needed */
                prcsh=ran1(seed);
                prtest=molarvcsh[cyccnt]/molarvcsh[cycorig];
                if(prcsh<=prtest){
                mic[xcur][ycur][zcur]=CSH;
                        cshage[xcur][ycur][zcur]=cyccnt;
                 count[CSH]+=1;
                }
                else{
                        mic[xcur][ycur][zcur]=POROSITY;
                        count[POROSITY]+=1;
                }
           /* May need extra solid CSH if temperature goes down with time */
                if(prtest>1.0){
                        if(prcsh<(prtest-1.0)){
                                 extcsh();
                        }
                }
                action=0;
        }

        if(action!=0){
        /* if diffusion step is possible, perform it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFCSH;
                }
                else{
                        /* indicate that diffusing CSH species remained */
                      /* at original location */
                        action=7;
                }
        }
        return(action);
}

/* routine to add extra FH3 when gypsum, hemihydrate, anhydrite, CAS2, or */
/* CaCl2 reacts with C4AF at location (xpres,ypres,zpres) */
/* Called by movegyp, moveettr, movecas2, movehem, moveanh, and movecacl2 */
/* Calls moveone and edgecnt */
void extfh3(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int multf,numnear,sump,xchr,ychr,zchr,check,fchr,i1,plok,newact;
        long int tries;

/* first try 6 neighboring locations until      */
```

```
/*      a) successful                           */
/*      b) all 6 sites are tried and full or    */
/*      c) 500 tries are made                   */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=500)&&(fchr==0)&&(sump!=30030));i1++){

                /* choose a neighbor at random */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                newact=0;
                multf=moveone(&xchr,&ychr,&zchr,&newact,sump);
                if(newact==0){printf("Error in value of newact in extfh3 \n");}
                check=mic[xchr][ychr][zchr];

                /* if neighbor is porosity   */
                /* then locate the FH3 there */
                if(check==POROSITY){
                        mic[xchr][ychr][zchr]=FH3;
                        count[FH3]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
                else{
                        sump*=multf;
                }
        }

/* if no neighbor available, locate FH3 at random location */
/* in pore space in contact with at least one FH3 */
        tries=0;
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the FH3 there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,FH3,FH3,DIFFFH3);
                        /* be sure that at least one neighboring pixel */
                        /* is FH3 or diffusing FH3 */
```

```
                        if((numnear<26)||(tries>5000)){
                                mic[xchr][ychr][zchr]=FH3;
                                count[FH3]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
}

/* routine to add extra ettringite when gypsum, anhydrite, or hemihydrate */
/* reacts with aluminates, addition adjacent to location (xpres,ypres,zpres) */
/* in a fashion to preserve needle growth */
/* etype=0 indicates primary ettringite */
/* etype=1 indicates iron-rich stable ettringite */
/* Returns flag indicating action taken */
/* Calls moveone and edgecnt */
/* Called by movegyp, movehem, moveanh, and movec3a */
int extettr(xpres,ypres,zpres,etype)
        int xpres,ypres,zpres,etype;
{
        int check,newact,multf,numnear,sump,xchr,ychr,zchr,fchr,i1,plok;
        int numalum;
        float pneigh,ptest;
        long int tries;

/* first try neighboring locations until        */
/*      a) successful                           */
/*      b) 1000 tries are made                  */
        fchr=0;
        sump=1;
        /* Note that 30030 = 2*3*5*7*11*13 */
        /* indicating that all six sites have been tried */
        for(i1=1;((i1<=1000)&&(fchr==0));i1++){

/* determine location of neighbor (using periodic boundaries) */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                newact=0;
                multf=moveone(&xchr,&ychr,&zchr,&newact,sump);
                if(newact==0){printf("Error in value of action \n");}

                check=mic[xchr][ychr][zchr];

                /* if neighbor is porosity, and conditions are favorable */
                /* based on number of neighboring ettringite, C3A, or C4AF */
                /* pixels then locate the ettringite there */
```

```
                if(check==POROSITY){
                        if(etype==0){
                                numnear=edgecnt(xchr,ychr,zchr,ETTR,ETTR,ETTR);
                                numalum=edgecnt(xchr,ychr,zchr,C3A,C3A,C3A);
                                numalum=26-numalum;
                        }
                        else{
                     numnear=edgecnt(xchr,ychr,zchr,ETTRC4AF,ETTRC4AF,ETTRC4AF);
                                numalum=edgecnt(xchr,ychr,zchr,C4AF,C4AF,C4AF);
                                numalum=26-numalum;
                        }
                        pneigh=(float)(numnear+1)/26.0;
                        pneigh*=pneigh;
                        if(numalum==0){pneigh=0.0;}
                        if(numalum>=2){pneigh+=0.5;}
                        if(numalum>=3){pneigh+=0.25;}
                        if(numalum>=5){pneigh+=0.25;}
                        ptest=ran1(seed);
                        if(pneigh>=ptest){
                                if(etype==0){
                                        mic[xchr][ychr][zchr]=ETTR;
                                        count[ETTR]+=1;
                                }
                                else{
                                        mic[xchr][ychr][zchr]=ETTRC4AF;
                                        count[ETTRC4AF]+=1;
                                }
                                fchr=1;
                                count[POROSITY]-=1;
                        }
                }
        }

/* if no neighbor available, locate ettringite at random location */
/* in pore space in contact with at least another ettringite */
/* or aluminate surface  */
        tries=0;
        while(fchr==0){
                tries+=1;
                newact=7;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
```

```
                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the ettringite there */
                if(check==POROSITY){
                        if(etype==0){
                              numnear=edgecnt(xchr,ychr,zchr,ETTR,C3A,C4AF);
                        }
                        else{
                              numnear=edgecnt(xchr,ychr,zchr,ETTRC4AF,C3A,C4AF);
                        }
                        /* be sure that at least one neighboring pixel */
                        /* is ettringite, or aluminate clinker */
                        if((tries>5000)||(numnear<26)){
                              if(etype==0){
                                      mic[xchr][ychr][zchr]=ETTR;
                                      count[ETTR]+=1;
                                }
                                else{
                                      mic[xchr][ychr][zchr]=ETTRC4AF;
                                      count[ETTRC4AF]+=1;
                                }
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
        return(newact);
}
/* routine to add extra CH when gypsum, hemihydrate, anhydrite, CaCl2, or */
/* diffusing CAS2  reacts with C4AF */
/* Called by movegyp, movehem, moveanh, moveettr, movecas2, and movecacl2 */
/* Calls edgecnt */
void extch()
{
        int numnear,sump,xchr,ychr,zchr,fchr,i1,plok,check;
        long int tries;

        fchr=0;
        tries=0;
        /* locate CH at random location */
        /* in pore space in contact with at least another CH */
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
```

```
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];

                /* if location is porosity, locate the CH there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,CH,DIFFCH,CH);
                        /* be sure that at least one neighboring pixel */
                        /* is CH or diffusing CH */
                        if((numnear<26)||(tries>5000)){
                                mic[xchr][ychr][zchr]=CH;
                                count[CH]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
}
/* routine to add extra gypsum when hemihydrate or anhydrite hydrates */
/* Called by movehem and moveanh */
/* Calls moveone and edgecnt */
void extgyps(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int multf,numnear,sump,xchr,ychr,zchr,check,fchr,i1,plok,newact;
        long int tries;

/* first try 6 neighboring locations until      */
/*      a) successful                          */
/*      b) all 6 sites are tried and full or   */
/*      c) 500 tries are made                  */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=500)&&(fchr==0)&&(sump!=30030));i1++){

                /* choose a neighbor at random */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                newact=0;
                multf=moveone(&xchr,&ychr,&zchr,&newact,sump);
                if(newact==0){printf("Error in value of newact in extfh3 \n");}
                check=mic[xchr][ychr][zchr];

                /* if neighbor is porosity   */
                /* then locate the GYPSUMS there */
                if(check==POROSITY){
```

```
                        mic[xchr][ychr][zchr]=GYPSUMS;
                        count[GYPSUMS]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
                else{
                        sump*=multf;
                }
        }

/* if no neighbor available, locate GYPSUMS at random location */
/* in pore space in contact with at least one GYPSUMS */
        tries=0;
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the GYPSUMS there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,HEMIHYD,GYPSUMS,ANHYDRITE);
                        /* be sure that at least one neighboring pixel */
                        /* is Gypsum in some form */
                        if((numnear<26)||(tries>5000)){
                                mic[xchr][ychr][zchr]=GYPSUMS;
                                count[GYPSUMS]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
}

/* routine to move a diffusing ANHYDRITE species */
/* Inputs: current location (xcur,ycur,zcur) and flag indicating if final */
/* step in diffusion process */
/* Returns flag indicating action taken (reaction or diffusion/no movement) */
/* Calls moveone */
/* Called by hydrate */
int moveanh(xcur,ycur,zcur,finalstep,nucprgyp)
        int xcur,ycur,zcur,finalstep;
        float nucprgyp;
{
```

```
        int xnew,ynew,znew,plok,action,sumback,sumin,check;
        int nexp,iexp,xexp,yexp,zexp,newact,sumold,sumgarb,ettrtype;
        float pgen,pexp,pext,p2diff;

        action=0;
        /* first check for nucleation */
        pgen=ran1(seed);
        p2diff=ran1(seed);
        if((nucprgyp>=pgen)||(finalstep==1)){
                action=0;
                mic[xcur][ycur][zcur]=GYPSUMS;
                count[DIFFANH]-=1;
                count[GYPSUMS]+=1;
                pexp=ran1(seed);
                if(pexp<0.4){
                       extgyps(xcur,ycur,zcur);
                }
        }
        else{
        /* Store current location of species */
         xnew=xcur;
         ynew=ycur;
         znew=zcur;
         sumin=1;
         sumback=moveone(&xnew,&ynew,&znew,&action,sumin);

         if(action==0){printf("Error in value of action \n");}
         check=mic[xnew][ynew][znew];

/* if new location is solid GYPSUM(S) or diffusing GYPSUM, then convert */
/* diffusing ANHYDRITE species to solid GYPSUM */
        if((check==GYPSUM)||(check==GYPSUMS)||(check==DIFFGYP)){
                        mic[xcur][ycur][zcur]=GYPSUMS;
                        /* decrement count of diffusing ANHYDRITE species */
                        /* and increment count of solid GYPSUMS */
                        count[DIFFANH]-=1;
                        count[GYPSUMS]+=1;
                        action=0;
                        /* Add extra gypsum as necessary */
                        pexp=ran1(seed);
                        if(pexp<0.4){
                             extgyps(xnew,ynew,znew);
                        }
         }
        /* if new location is C3A or diffusing C3A, execute conversion */
        /* to ettringite (including necessary volumetric expansion) */
        else if((check==C3A)||((check==DIFFC3A)&&(p2diff<0.01))||
```

```
      ((check==DIFFC4A)&&(p2diff<0.01))){
/* Convert diffusing gypsum to an ettringite pixel */
        ettrtype=0;
        mic[xcur][ycur][zcur]=ETTR;
        if(check==DIFFC4A){
                ettrtype=1;
                mic[xcur][ycur][zcur]=ETTRC4AF;
        }
        action=0;
        count[DIFFANH]-=1;
        count[check]-=1;

        /* determine if C3A should be converted to ettringite */
        /* 1 unit of hemihydrate requires 0.569 units of C3A */
        /* and should form 4.6935 units of ettringite */
        pexp=ran1(seed);
        nexp=3;
        if(pexp<=0.569){
                if(ettrtype==0){
                        mic[xnew][ynew][znew]=ETTR;
                        count[ETTR]+=1;
                }
                else{
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                }
                nexp=2;
        }
        else{
                /* maybe someday, use a new FIXEDC3A here */
                /* so it won't dissolve later */
                if(check==C3A){
                        mic[xnew][ynew][znew]=C3A;
                        count[C3A]+=1;
                }
                else{
                        if(ettrtype==0){
                                count[DIFFC3A]+=1;
                                mic[xnew][ynew][znew]=DIFFC3A;
                        }
                        else{
                                count[DIFFC4A]+=1;
                                mic[xnew][ynew][znew]=DIFFC4A;
                        }
                }
                nexp=3;
        }
```

```
/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,ettrtype);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                        }
                }

                /* probabilistic-based expansion for last ettringite pixel */
                pexp=ran1(seed);
                if(pexp<=0.6935){
                        newact=extettr(xexp,yexp,zexp,ettrtype);
                }
        }

        /* if new location is C4AF execute conversion */
```

```
        /* to ettringite (including necessary volumetric expansion) */
        if(check==C4AF){
                mic[xcur][ycur][zcur]=ETTRC4AF;
                count[ETTRC4AF]+=1;
                count[DIFFANH]-=1;

                /* determine if C4AF should be converted to ettringite */
                /* 1 unit of gypsum requires 0.8174 units of C4AF */
                /* and should form 4.6935 units of ettringite */
                pexp=ran1(seed);
                nexp=3;
                if(pexp<=0.8174){
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                        count[C4AF]-=1;
                        nexp=2;
                        pext=ran1(seed);
                        /* Addition of extra CH */
                        if(pext<0.2584){
                                        extch();
                        }
                        pext=ran1(seed);
                        /* Addition of extra FH3 */
                        if(pext<0.5453){
                                        extfh3(xnew,ynew,znew);
                        }

                }
                else{
                        /* maybe someday, use a new FIXEDC4AF here */
                        /* so it won't dissolve later */
                        mic[xnew][ynew][znew]=C4AF;
                        nexp=3;
                }

/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,1);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
```

```
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                        }
                }

                /* probabilistic-based expansion for last ettringite pixel */
                pexp=ran1(seed);
                if(pexp<=0.6935){
                        newact=extettr(xexp,yexp,zexp,1);
                }
                action=0;
        }
}

        if(action!=0){
        /* if diffusion step is possible, perform it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFANH;
                }
                else{
                        /* indicate that diffusing ANHYDRITE species remained */
                      /* at original location */
                        action=7;
                }
```

```
        }
        return(action);
}

/* routine to move a diffusing HEMIHYDRATE species */
/* Inputs: current location (xcur,ycur,zcur) and flag indicating if final */
/* step in diffusion process */
/* Returns flag indicating action taken (reaction or diffusion/no movement) */
/* Calls moveone, extettr, extch, and extfh3 */
/* Called by hydrate */
int movehem(xcur,ycur,zcur,finalstep,nucprgyp)
        int xcur,ycur,zcur,finalstep;
        float nucprgyp;
{
        int xnew,ynew,znew,plok,action,sumback,sumin,check;
        int nexp,iexp,xexp,yexp,zexp,newact,sumold,sumgarb,ettrtype;
        float pgen,pexp,pext,p2diff;

        action=0;
        /* first check for nucleation */
        pgen=ran1(seed);
        p2diff=ran1(seed);
        if((nucprgyp>=pgen)||(finalstep==1)){
                action=0;
                mic[xcur][ycur][zcur]=GYPSUMS;
                count[DIFFHEM]-=1;
                count[GYPSUMS]+=1;
                /* Add extra gypsum as necessary */
                pexp=ran1(seed);
                if(pexp<0.4){
                      extgyps(xcur,ycur,zcur);
                }
        }
        else{
        /* Store current location of species */
                xnew=xcur;
                ynew=ycur;
                znew=zcur;
                sumin=1;
                sumback=moveone(&xnew,&ynew,&znew,&action,sumin);

                if(action==0){printf("Error in value of action \n");}
                check=mic[xnew][ynew][znew];

/* if new location is solid GYPSUM(S) or diffusing GYPSUM, then convert */
/* diffusing HEMIHYDRATE species to solid GYPSUM */
         if((check==GYPSUM)||(check==GYPSUMS)||(check==DIFFGYP)){
```

```
        mic[xcur][ycur][zcur]=GYPSUMS;
        /* decrement count of diffusing HEMIHYDRATE species */
        /* and increment count of solid GYPSUMS */
        count[DIFFHEM]-=1;
        count[GYPSUMS]+=1;
        action=0;
                /* Add extra gypsum as necessary */
         pexp=ran1(seed);
                if(pexp<0.4){
                      extgyps(xnew,ynew,znew);
                }
}
/* if new location is C3A or diffusing C3A, execute conversion */
/* to ettringite (including necessary volumetric expansion) */
else if((check==C3A)||((check==DIFFC3A)&&
    (p2diff<0.01))||((check==DIFFC4A)&&(p2diff<0.01))){
/* Convert diffusing gypsum to an ettringite pixel */
        ettrtype=0;
        mic[xcur][ycur][zcur]=ETTR;
        if(check==DIFFC4A){
                ettrtype=1;
                mic[xcur][ycur][zcur]=ETTRC4AF;
        }
        action=0;
        count[DIFFHEM]-=1;
        count[check]-=1;

        /* determine if C3A should be converted to ettringite */
        /* 1 unit of hemihydrate requires 0.5583 units of C3A */
        /* and should form 4.6053 units of ettringite */
        pexp=ran1(seed);
        nexp=3;
        if(pexp<=0.5583){
                if(ettrtype==0){
                        mic[xnew][ynew][znew]=ETTR;
                        count[ETTR]+=1;
                }
                else{
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                }
                nexp=2;
        }
        else{
                /* maybe someday, use a new FIXEDC3A here */
                /* so it won't dissolve later */
                if(check==C3A){
```

```
                        mic[xnew][ynew][znew]=C3A;
                        count[C3A]+=1;
                }
                else{
                        if(ettrtype==0){
                                count[DIFFC3A]+=1;
                                mic[xnew][ynew][znew]=DIFFC3A;
                        }
                        else{
                                count[DIFFC4A]+=1;
                                mic[xnew][ynew][znew]=DIFFC4A;
                        }
                }
                nexp=3;
        }

/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
        xexp=xcur;
        yexp=ycur;
        zexp=zcur;
        for(iexp=1;iexp<=nexp;iexp++){
                newact=extettr(xexp,yexp,zexp,ettrtype);
                /* update xexp, yexp and zexp as needed */
                switch (newact){
                        case 1:
                                xexp-=1;
                                if(xexp<0){xexp=(SYSIZEM1);}
                                break;
                        case 2:
                                xexp+=1;
                                if(xexp>=SYSIZE){xexp=0;}
                                break;
                        case 3:
                                yexp-=1;
                                if(yexp<0){yexp=(SYSIZEM1);}
                                break;
                        case 4:
                                yexp+=1;
                                if(yexp>=SYSIZE){yexp=0;}
                                break;
                        case 5:
                                zexp-=1;
                                if(zexp<0){zexp=(SYSIZEM1);}
                                break;
                        case 6:
```

```
                                zexp+=1;
                                if(zexp>=SYSIZE){zexp=0;}
                                break;
                        default:
                                break;
                }
        }

        /* probabilistic-based expansion for last ettringite pixel */
        pexp=ran1(seed);
        if(pexp<=0.6053){
                newact=extettr(xexp,yexp,zexp,ettrtype);
        }
}

/* if new location is C4AF execute conversion */
/* to ettringite (including necessary volumetric expansion) */
if(check==C4AF){
        mic[xcur][ycur][zcur]=ETTRC4AF;
        count[ETTRC4AF]+=1;
        count[DIFFHEM]-=1;

        /* determine if C4AF should be converted to ettringite */
        /* 1 unit of gypsum requires 0.802 units of C4AF */
        /* and should form 4.6053 units of ettringite */
        pexp=ran1(seed);
        nexp=3;
        if(pexp<=0.802){
                mic[xnew][ynew][znew]=ETTRC4AF;
                count[ETTRC4AF]+=1;
                count[C4AF]-=1;
                nexp=2;
                pext=ran1(seed);
                /* Addition of extra CH */
                if(pext<0.2584){
                                extch();
                }
                pext=ran1(seed);
                /* Addition of extra FH3 */
                if(pext<0.5453){
                                extfh3(xnew,ynew,znew);
                }

        }
        else{
                /* maybe someday, use a new FIXEDC4AF here */
                /* so it won't dissolve later */
```

```
                        mic[xnew][ynew][znew]=C4AF;
                        nexp=3;
                }

/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,1);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                       }
                }

                /* probabilistic-based expansion for last ettringite pixel */
                pexp=ran1(seed);
                if(pexp<=0.6053){
                        newact=extettr(xexp,yexp,zexp,1);
```

```
                }
                action=0;
        }
}

        if(action!=0){
        /* if diffusion step is possible, perform it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFHEM;
                }
                else{
                        /* indicate that diffusing HEMIHYDRATE species */
                      /* remained at original location */
                        action=7;
                }
        }
        return(action);
}

/* routine to add extra Freidel's salt when CaCl2 reacts with */
/* C3A or C4AF at location (xpres,ypres,zpres) */
/* Called by movecacl2 and movec3a */
/* Calls moveone and edgecnt */
int extfreidel(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int multf,numnear,sump,xchr,ychr,zchr,check,fchr,i1,plok,newact;
        long int tries;

/* first try 6 neighboring locations until      */
/*      a) successful                          */
/*      b) all 6 sites are tried and full or   */
/*      c) 500 tries are made                  */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=500)&&(fchr==0)&&(sump!=30030));i1++){

                /* choose a neighbor at random */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                newact=0;
                multf=moveone(&xchr,&ychr,&zchr,&newact,sump);
                if(newact==0){printf("Error in value of newact in extfreidel \n");}
                check=mic[xchr][ychr][zchr];

                /* if neighbor is porosity   */
```

```
                /* then locate the freidel's salt there */
                if(check==POROSITY){
                        mic[xchr][ychr][zchr]=FREIDEL;
                        count[FREIDEL]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
                else{
                        sump*=multf;
                }
        }

/* if no neighbor available, locate FREIDEL at random location */
/* in pore space in contact with at least one FREIDEL */
        tries=0;
        while(fchr==0){
                tries+=1;
                newact=7;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the FREIDEL there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,FREIDEL,FREIDEL,DIFFCACL2);
                        /* be sure that at least one neighboring pixel */
                        /* is FREIDEL or diffusing CACL2 */
                        if((numnear<26)||(tries>5000)){
                                mic[xchr][ychr][zchr]=FREIDEL;
                                count[FREIDEL]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
return(newact);
}
/* routine to add extra stratlingite when AS reacts with */
/* CH at location (xpres,ypres,zpres) */
/* or when diffusing CAS2 reacts with aluminates */
/* Called by moveas, movech, and movecas2 */
/* Calls moveone and edgecnt */
int extstrat(xpres,ypres,zpres)
```

```
        int xpres,ypres,zpres;
{
        int multf,numnear,sump,xchr,ychr,zchr,check,fchr,i1,plok,newact;
        long int tries;

/* first try 6 neighboring locations until      */
/*      a) successful                          */
/*      b) all 6 sites are tried and full or   */
/*      c) 500 tries are made                  */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=500)&&(fchr==0)&&(sump!=30030));i1++){

                /* choose a neighbor at random */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                newact=0;
                multf=moveone(&xchr,&ychr,&zchr,&newact,sump);
                if(newact==0){printf("Error in value of newact in extstrat \n");}
                check=mic[xchr][ychr][zchr];

                /* if neighbor is porosity   */
                /* then locate the stratlingite there */
                if(check==POROSITY){
                        mic[xchr][ychr][zchr]=STRAT;
                        count[STRAT]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
                else{
                        sump*=multf;
                }
        }

/* if no neighbor available, locate STRAT at random location */
/* in pore space in contact with at least one STRAT */
        tries=0;
        while(fchr==0){
                tries+=1;
                newact=7;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
```

```
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the STRAT there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,STRAT,DIFFCAS2,DIFFAS);
                        /* be sure that at least one neighboring pixel */
                        /* is STRAT, diffusing CAS2, or diffusing AS */
                        if((numnear<26)||(tries>5000)){
                                mic[xchr][ychr][zchr]=STRAT;
                                count[STRAT]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
return(newact);
}
/* routine to move a diffusing gypsum species */
/* from current location (xcur,ycur,zcur) */
/* Returns action flag indicating response taken */
/* Called by hydrate */
/* Calls moveone, extettr, extch, and extfh3 */
int movegyp(xcur,ycur,zcur,finalstep)
        int xcur,ycur,zcur,finalstep;
{
        int check,xnew,ynew,znew,plok,action,nexp,iexp;
        int xexp,yexp,zexp,newact,sumold,sumgarb,ettrtype;
        float pexp,pext,p2diff;

        sumold=1;

/* First be sure that a diffusing gypsum species is located at xcur,ycur,zcur */
/* if not, return to calling routine */
        if(mic[xcur][ycur][zcur]!=DIFFGYP){
                action=0;
                return(action);
        }

/* Determine new coordinates (periodic boundaries are used) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in movegyp \n");}
        check=mic[xnew][ynew][znew];
        p2diff=ran1(seed);
```

```
/* if new location is CSH, check for absorption of gypsum */
if((check==CSH)&&((float)count[ABSGYP]<(gypabsprob*(float)count[CSH]))){
        pexp=ran1(seed);
        if(pexp<AGRATE){
                /* update counts for absorbed and diffusing gypsum */
                count[ABSGYP]+=1;
                count[DIFFGYP]-=1;
                mic[xcur][ycur][zcur]=ABSGYP;
                action=0;
        }
}

/* if new location is C3A or diffusing C3A, execute conversion */
/* to ettringite (including necessary volumetric expansion) */
/* Use p2diff to try to favor formation of ettringite on */
/* aluminate surfaces as opposed to in solution */
else if((check==C3A)||((check==DIFFC3A)&&(p2diff<0.01))||
    ((check==DIFFC4A)&&(p2diff<0.01))){
/* Convert diffusing gypsum to an ettringite pixel */
        ettrtype=0;
        mic[xcur][ycur][zcur]=ETTR;
        if(check==DIFFC4A){
                ettrtype=1;
                mic[xcur][ycur][zcur]=ETTRC4AF;
        }
        action=0;
        count[DIFFGYP]-=1;
        count[check]-=1;

        /* determine if C3A should be converted to ettringite */
        /* 1 unit of gypsum requires 0.40 units of C3A */
        /* and should form 3.30 units of ettringite */
        pexp=ran1(seed);
        nexp=2;
        if(pexp<=0.40){
                if(ettrtype==0){
                        mic[xnew][ynew][znew]=ETTR;
                        count[ETTR]+=1;
                }
                else{
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                }
                nexp=1;
        }
        else{
                /* maybe someday, use a new FIXEDC3A here */
```

```
                        /* so it won't dissolve later */
                        if(check==C3A){
                                mic[xnew][ynew][znew]=C3A;
                                count[C3A]+=1;
                        }
                        else{
                                if(ettrtype==0){
                                        count[DIFFC3A]+=1;
                                        mic[xnew][ynew][znew]=DIFFC3A;
                                }
                                else{
                                        count[DIFFC4A]+=1;
                                        mic[xnew][ynew][znew]=DIFFC4A;
                                }
                        }
                        nexp=2;
                }

/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,ettrtype);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
```

```
                                break;
                        case 6:
                                zexp+=1;
                                if(zexp>=SYSIZE){zexp=0;}
                                break;
                        default:
                                break;
                }
        }

        /* probabilistic-based expansion for last ettringite pixel */
        pexp=ran1(seed);
        if(pexp<=0.30){
                newact=extettr(xexp,yexp,zexp,ettrtype);
        }
}

/* if new location is C4AF execute conversion */
/* to ettringite (including necessary volumetric expansion) */
if(check==C4AF){
        mic[xcur][ycur][zcur]=ETTRC4AF;
        count[ETTRC4AF]+=1;
        count[DIFFGYP]-=1;

        /* determine if C4AF should be converted to ettringite */
        /* 1 unit of gypsum requires 0.575 units of C4AF */
        /* and should form 3.30 units of ettringite */
        pexp=ran1(seed);
        nexp=2;
        if(pexp<=0.575){
                mic[xnew][ynew][znew]=ETTRC4AF;
                count[ETTRC4AF]+=1;
                count[C4AF]-=1;
                nexp=1;
                pext=ran1(seed);
                /* Addition of extra CH */
                if(pext<0.2584){
                                extch();
                }
                pext=ran1(seed);
                /* Addition of extra FH3 */
                if(pext<0.5453){
                                extfh3(xnew,ynew,znew);
                }

        }
        else{
```

```
                        /* maybe someday, use a new FIXEDC4AF here */
                        /* so it won't dissolve later */
                        mic[xnew][ynew][znew]=C4AF;
                        nexp=2;
                }

/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,1);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                        }
                }

                /* probabilistic-based expansion for last ettringite pixel */
                pexp=ran1(seed);
```

```
                if(pexp<=0.30){
                        newact=extettr(xexp,yexp,zexp,1);
                }
                action=0;
        }

        /* if last diffusion step and no reaction, convert back to */
        /* primary solid gypsum */
        if((action!=0)&&(finalstep==1)){
                action=0;
                count[DIFFGYP]-=1;
                count[GYPSUM]+=1;
                mic[xcur][ycur][zcur]=GYPSUM;
        }

        if(action!=0){
                /* if diffusion is possible, execute it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFGYP;
                }
                else{
                        /* indicate that diffusing gypsum remained at */
                        /* original location */
                        action=7;
                }
        }
        return(action);
}
/* routine to move a diffusing CaCl2 species */
/* from current location (xcur,ycur,zcur) */
/* Returns action flag indicating response taken */
/* Called by hydrate */
/* Calls moveone, extfreidel, extch, and extfh3 */
int movecacl2(xcur,ycur,zcur,finalstep)
        int xcur,ycur,zcur,finalstep;
{
        int check,xnew,ynew,znew,plok,action,nexp,iexp;
        int xexp,yexp,zexp,newact,sumold,sumgarb,keep;
        float pexp,pext;

        sumold=1;
        keep=0;

/* First be sure that a diffusing CaCl2 species is located at xcur,ycur,zcur */
/* if not, return to calling routine */
        if(mic[xcur][ycur][zcur]!=DIFFCACL2){
```

```
                action=0;
                return(action);
        }

/* Determine new coordinates (periodic boundaries are used) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in movecacl2 \n");}
        check=mic[xnew][ynew][znew];

        /* if new location is C3A or diffusing C3A, execute conversion */
        /* to freidel's salt (including necessary volumetric expansion) */
        if((check==C3A)||(check==DIFFC3A)||(check==DIFFC4A)){
        /* Convert diffusing C3A or C3A to a freidel's salt pixel */
                action=0;
                mic[xnew][ynew][znew]=FREIDEL;
                count[FREIDEL]+=1;
                count[check]-=1;

                /* determine if diffusing CaCl2 should be converted to FREIDEL */
                /* 0.5793 unit of CaCl2 requires 1 unit of C3A */
                /* and should form 3.3295 units of FREIDEL */
                pexp=ran1(seed);
                nexp=2;
                if(pexp<=0.5793){
                        mic[xcur][ycur][zcur]=FREIDEL;
                        count[FREIDEL]+=1;
                        count[DIFFCACL2]-=1;
                        nexp=1;
                }
                else{
                        keep=1;
                        nexp=2;
                }

/* create extra Freidel's salt pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added FREIDEL */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extfreidel(xexp,yexp,zexp);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
```

```
                        case 1:
                                xexp-=1;
                                if(xexp<0){xexp=(SYSIZEM1);}
                                break;
                        case 2:
                                xexp+=1;
                                if(xexp>=SYSIZE){xexp=0;}
                                break;
                        case 3:
                                yexp-=1;
                                if(yexp<0){yexp=(SYSIZEM1);}
                                break;
                        case 4:
                                yexp+=1;
                                if(yexp>=SYSIZE){yexp=0;}
                                break;
                        case 5:
                                zexp-=1;
                                if(zexp<0){zexp=(SYSIZEM1);}
                                break;
                        case 6:
                                zexp+=1;
                                if(zexp>=SYSIZE){zexp=0;}
                                break;
                        default:
                                break;
                }
        }

        /* probabilistic-based expansion for last FREIDEL pixel */
        pexp=ran1(seed);
        if(pexp<=0.3295){
                newact=extfreidel(xexp,yexp,zexp);
        }
}

/* if new location is C4AF execute conversion */
/* to freidel's salt (including necessary volumetric expansion) */
else if(check==C4AF){
        mic[xnew][ynew][znew]=FREIDEL;
        count[FREIDEL]+=1;
        count[C4AF]-=1;

        /* determine if CACL2 should be converted to FREIDEL */
        /* 0.4033 unit of CaCl2 requires 1 unit of C4AF */
        /* and should form 2.3176 units of FREIDEL */
        /* Also 0.6412 units of CH and 1.3522 units of FH3 */
```

```
                /* per unit of CACL2 */
                pexp=ran1(seed);
                nexp=1;
                if(pexp<=0.4033){
                        mic[xcur][ycur][zcur]=FREIDEL;
                        count[FREIDEL]+=1;
                        count[DIFFCACL2]-=1;
                        nexp=0;
                        pext=ran1(seed);
                        /* Addition of extra CH */
                        if(pext<0.6412){
                                        extch();
                        }
                        pext=ran1(seed);
                        /* Addition of extra FH3 */
                        if(pext<0.3522){
                                        extfh3(xnew,ynew,znew);
                        }
                        extfh3(xnew,ynew,znew);

                }
                else{
                        nexp=1;
                        keep=1;
                }

/* create extra freidel's salt pixels to maintain volume stoichiometry */
/* xexp, yexp and zexp hold coordinates of most recently added FREIDEL */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extfreidel(xexp,yexp,zexp);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
```

```
                        case 4:
                                yexp+=1;
                                if(yexp>=SYSIZE){yexp=0;}
                                break;
                        case 5:
                                zexp-=1;
                                if(zexp<0){zexp=(SYSIZEM1);}
                                break;
                        case 6:
                                zexp+=1;
                                if(zexp>=SYSIZE){zexp=0;}
                                break;
                        default:
                                break;
                }
        }

        /* probabilistic-based expansion for last FREIDEL pixel */
        pexp=ran1(seed);
        if(pexp<=0.3176){
                newact=extfreidel(xexp,yexp,zexp);
        }
        action=0;
}

/* if last diffusion step and no reaction, convert back to */
/* solid CaCl2 */
if((action!=0)&&(finalstep==1)){
        action=0;
        count[DIFFCACL2]-=1;
        count[CACL2]+=1;
        mic[xcur][ycur][zcur]=CACL2;
}

if(action!=0){
        /* if diffusion is possible, execute it */
        if(check==POROSITY){
                mic[xcur][ycur][zcur]=POROSITY;
                mic[xnew][ynew][znew]=DIFFCACL2;
        }
        else{
                /* indicate that diffusing CACL2 remained at */
                /* original location */
                action=7;
        }
}
if(keep==1){action=7;}
```

```
        return(action);
}
/* routine to move a diffusing CAS2 species */
/* from current location (xcur,ycur,zcur) */
/* Returns action flag indicating response taken */
/* Called by hydrate */
/* Calls moveone, extstrat, extch, and extfh3 */
int movecas2(xcur,ycur,zcur,finalstep)
        int xcur,ycur,zcur,finalstep;
{
        int check,xnew,ynew,znew,plok,action,nexp,iexp;
        int xexp,yexp,zexp,newact,sumold,sumgarb,keep;
        float pexp,pext;

        sumold=1;
        keep=0;

/* First be sure that a diffusing CAS2 species is located at xcur,ycur,zcur */
/* if not, return to calling routine */
        if(mic[xcur][ycur][zcur]!=DIFFCAS2){
                action=0;
                return(action);
        }

/* Determine new coordinates (periodic boundaries are used) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in movecas2 \n");}
        check=mic[xnew][ynew][znew];

        /* if new location is C3A or diffusing C3A, execute conversion */
        /* to stratlingite (including necessary volumetric expansion) */
        if((check==C3A)||(check==DIFFC3A)||(check==DIFFC4A)){
        /* Convert diffusing CAS2 to a stratlingite pixel */
                action=0;
                mic[xcur][ycur][zcur]=STRAT;
                count[STRAT]+=1;
                count[DIFFCAS2]-=1;

                /* determine if diffusing or solid C3A should be converted to STRAT*/
                /* 1 unit of CAS2 requires 0.886 units of C3A */
                /* and should form 4.286 units of STRAT */
                pexp=ran1(seed);
                nexp=3;
```

```
                if(pexp<=0.886){
                        mic[xnew][ynew][znew]=STRAT;
                        count[STRAT]+=1;
                        count[check]-=1;
                        nexp=2;
                }

/* create extra stratlingite pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added STRAT */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extstrat(xexp,yexp,zexp);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                        }
                }

                /* probabilistic-based expansion for last STRAT pixel */
                pexp=ran1(seed);
```

```
                if(pexp<=0.286){
                        newact=extstrat(xexp,yexp,zexp);
                }
        }

        /* if new location is C4AF execute conversion */
        /* to stratlingite (including necessary volumetric expansion) */
        else if(check==C4AF){
                mic[xnew][ynew][znew]=STRAT;
                count[STRAT]+=1;
                count[C4AF]-=1;

                /* determine if CAS2 should be converted to STRAT */
                /* 0.786 units of CAS2 requires 1 unit of C4AF */
                /* and should form 3.37 units of STRAT */
                /* Also 0.2586 units of CH and 0.5453 units of FH3 */
                /* per unit of C4AF */
                pexp=ran1(seed);
                nexp=2;
                if(pexp<=0.786){
                        mic[xcur][ycur][zcur]=STRAT;
                        count[STRAT]+=1;
                        count[DIFFCAS2]-=1;
                        nexp=1;
                        pext=ran1(seed);
                        /* Addition of extra CH */
                        /* 0.329= 0.2586/0.786 */
                        if(pext<0.329){
                                        extch();
                        }
                        pext=ran1(seed);
                        /* Addition of extra FH3 */
                        /* 0.6938= 0.5453/0.786 */
                        if(pext<0.6938){
                                        extfh3(xnew,ynew,znew);
                        }

                }
                else{
                        nexp=2;
                        keep=1;
                }

/* create extra stratlingite pixels to maintain volume stoichiometry */
/* xexp, yexp and zexp hold coordinates of most recently added STRAT */
                xexp=xcur;
                yexp=ycur;
```

```
        zexp=zcur;
        for(iexp=1;iexp<=nexp;iexp++){
                newact=extstrat(xexp,yexp,zexp);
                /* update xexp, yexp and zexp as needed */
                switch (newact){
                        case 1:
                                xexp-=1;
                                if(xexp<0){xexp=(SYSIZEM1);}
                                break;
                        case 2:
                                xexp+=1;
                                if(xexp>=SYSIZE){xexp=0;}
                                break;
                        case 3:
                                yexp-=1;
                                if(yexp<0){yexp=(SYSIZEM1);}
                                break;
                        case 4:
                                yexp+=1;
                                if(yexp>=SYSIZE){yexp=0;}
                                break;
                        case 5:
                                zexp-=1;
                                if(zexp<0){zexp=(SYSIZEM1);}
                                break;
                        case 6:
                                zexp+=1;
                                if(zexp>=SYSIZE){zexp=0;}
                                break;
                        default:
                                break;
               }
        }

        /* probabilistic-based expansion for last STRAT pixel */
        pexp=ran1(seed);
        if(pexp<=0.37){
                newact=extstrat(xexp,yexp,zexp);
        }
        action=0;
}

/* if last diffusion step and no reaction, convert back to */
/* solid CAS2 */
if((action!=0)&&(finalstep==1)){
        action=0;
        count[DIFFCAS2]-=1;
```

```
                count[CAS2]+=1;
                mic[xcur][ycur][zcur]=CAS2;
        }

        if(action!=0){
                /* if diffusion is possible, execute it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFCAS2;
                }
                else{
                        /* indicate that diffusing CAS2 remained at */
                        /* original location */
                        action=7;
                }
        }
        if(keep==1){action=7;}
        return(action);
}
/* routine to move a diffusing AS species */
/* from current location (xcur,ycur,zcur) */
/* Returns action flag indicating response taken */
/* Called by hydrate */
/* Calls moveone, extstrat */
int moveas(xcur,ycur,zcur,finalstep)
        int xcur,ycur,zcur,finalstep;
{
        int check,xnew,ynew,znew,plok,action,nexp,iexp;
        int xexp,yexp,zexp,newact,sumold,sumgarb,keep;
        float pexp,pext;

        sumold=1;
        keep=0;

/* First be sure that a diffusing AS species is located at xcur,ycur,zcur */
/* if not, return to calling routine */
        if(mic[xcur][ycur][zcur]!=DIFFAS){
                action=0;
                return(action);
        }

/* Determine new coordinates (periodic boundaries are used) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
```

```
        if(action==0){printf("Error in value of action in moveas \n");}
        check=mic[xnew][ynew][znew];

        /* if new location is CH or diffusing CH, execute conversion */
        /* to stratlingite (including necessary volumetric expansion) */
        if((check==CH)||(check==DIFFCH)){
        /* Convert diffusing CH or CH to a stratlingite pixel */
                action=0;
                mic[xnew][ynew][znew]=STRAT;
                count[STRAT]+=1;
                count[check]-=1;

                /* determine if diffusing AS should be converted to STRAT */
                /* 0.7538 unit of AS requires 1 unit of CH */
                /* and should form 3.26 units of STRAT */
                pexp=ran1(seed);
                nexp=2;
                if(pexp<=0.7538){
                        mic[xcur][ycur][zcur]=STRAT;
                        count[STRAT]+=1;
                        count[DIFFAS]-=1;
                        nexp=1;
                }
                else{
                        keep=1;
                        nexp=2;
                }

/* create extra stratlingite pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added STRAT */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extstrat(xexp,yexp,zexp);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
```

```
                                if(yexp<0){yexp=(SYSIZEM1);}
                                break;
                        case 4:
                                yexp+=1;
                                if(yexp>=SYSIZE){yexp=0;}
                                break;
                        case 5:
                                zexp-=1;
                                if(zexp<0){zexp=(SYSIZEM1);}
                                break;
                        case 6:
                                zexp+=1;
                                if(zexp>=SYSIZE){zexp=0;}
                                break;
                        default:
                                break;
                }
        }

        /* probabilistic-based expansion for last stratlingite pixel */
        pexp=ran1(seed);
        if(pexp<=0.326){
                newact=extstrat(xexp,yexp,zexp);
        }
}

/* if last diffusion step and no reaction, convert back to */
/* solid ASG */
if((action!=0)&&(finalstep==1)){
        action=0;
        count[DIFFAS]-=1;
        count[ASG]+=1;
        mic[xcur][ycur][zcur]=ASG;
}

if(action!=0){
        /* if diffusion is possible, execute it */
        if(check==POROSITY){
                mic[xcur][ycur][zcur]=POROSITY;
                mic[xnew][ynew][znew]=DIFFAS;
        }
        else{
                /* indicate that diffusing AS remained at */
                /* original location */
                action=7;
        }
}
```

```
        if(keep==1){action=7;}
        return(action);
}

/* routine to add extra AFm phase when diffusing ettringite reacts */
/* with C3A (diffusing or solid) at location (xpres,ypres,zpres) */
/* Called by moveettr and movec3a */
/* Calls moveone and edgecnt */
void extafm(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int check,sump,xchr,ychr,zchr,fchr,i1,plok,newact,numnear;
        long int tries;

/* first try 6 neighboring locations until      */
/*      a) successful                          */
/*      b) all 6 sites are tried or            */
/*      c) 100 tries are made                  */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=100)&&(fchr==0)&&(sump!=30030));i1++){

                /* determine location of neighbor (using periodic boundaries) */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                newact=0;
                sump*=moveone(&xchr,&ychr,&zchr,&newact,sump);
                if(newact==0){printf("Error in value of newact in extafm \n");}
                check=mic[xchr][ychr][zchr];

                /* if neighbor is porosity, locate the AFm phase there */
                if(check==POROSITY){
                        mic[xchr][ychr][zchr]=AFM;
                        count[AFM]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
         }

         /* if no neighbor available, locate AFm phase at random location */
         /* in pore space */
         tries=0;
         while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
```

```
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];

                /* if location is porosity, locate the extra AFm there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,AFM,C3A,C4AF);
                        /* Be sure that at least one neighboring pixel is */
                        /* Afm phase, C3A, or C4AF */
                        if((tries>5000)||(numnear<26)){
                                mic[xchr][ychr][zchr]=AFM;
                                count[AFM]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
}
/* routine to move a diffusing ettringite species */
/* currently located at (xcur,ycur,zcur) */
/* Called by hydrate */
/* Calls moveone, extch, extfh3, and extafm */
int moveettr(xcur,ycur,zcur,finalstep)
        int xcur,ycur,zcur,finalstep;
{
        int check,xnew,ynew,znew,plok,action,nexp,iexp;
        int xexp,yexp,zexp,newact,sumold,sumgarb;
        float pexp,pafm,pgrow;

/* First be sure a diffusing ettringite species is located at xcur,ycur,zcur */
/* if not, return to calling routine */
        if(mic[xcur][ycur][zcur]!=DIFFETTR){
                action=0;
                return(action);
        }

/* Determine new coordinates (periodic boundaries are used) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumold=1;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in moveettr \n");}
```

```
check=mic[xnew][ynew][znew];

/* if new location is C4AF, execute conversion */
/* to AFM phase (including necessary volumetric expansion) */
if(check==C4AF){
        /* Convert diffusing ettringite to AFM phase */
        mic[xcur][ycur][zcur]=AFM;
        count[AFM]+=1;
        count[DIFFETTR]-=1;

        /* determine if C4AF should be converted to Afm */
        /* or FH3- 1 unit of ettringite requires 0.348 units */
        /* of C4AF to form 1.278 units of Afm, */
        /* 0.0901 units of CH and 0.1899 units of FH3 */
        pexp=ran1(seed);

        if(pexp<=0.278){
                mic[xnew][ynew][znew]=AFM;
                count[AFM]+=1;
                count[C4AF]-=1;
                pafm=ran1(seed);
                /* 0.3241= 0.0901/0.278 */
                if(pafm<=0.3241){
                        extch();
                }
                pafm=ran1(seed);
                /* 0.4313= ((.1899-(.348-.278))/.278)   */
                if(pafm<=0.4313){
                        extfh3(xnew,ynew,znew);
                }
        }
        else if (pexp<=0.348){
                mic[xnew][ynew][znew]=FH3;
                count[FH3]+=1;
                count[C4AF]-=1;
        }
        action=0;
}

/* if new location is C3A or diffusing C3A, execute conversion */
/* to AFM phase (including necessary volumetric expansion) */
else if((check==C3A)||(check==DIFFC3A)){
        /* Convert diffusing ettringite to AFM phase */
        action=0;
        mic[xcur][ycur][zcur]=AFM;
        count[DIFFETTR]-=1;
        count[AFM]+=1;
```

```
                count[check]-=1;

                /* determine if C3A should be converted to AFm */
                /* 1 unit of ettringite requires 0.2424 units of C3A */
                /* and should form 1.278 units of AFm phase */
                pexp=ran1(seed);
                if(pexp<=0.2424){
                        mic[xnew][ynew][znew]=AFM;
                        count[AFM]+=1;
                        pafm=(-0.1);
                }
                else{
                        /* maybe someday, use a new FIXEDC3A here */
                        /* so it won't dissolve later */
                        if(check==C3A){
                                mic[xnew][ynew][znew]=C3A;
                                count[C3A]+=1;
                        }
                        else{
                                count[DIFFC3A]+=1;
                                mic[xnew][ynew][znew]=DIFFC3A;
                        }
/*                      pafm=(0.278-0.2424)/(1.0-0.2424);  */
                        pafm=0.04699;
                }

                /* probabilistic-based expansion for new AFm phase pixel */
                pexp=ran1(seed);
                if(pexp<=pafm){
                        extafm(xcur,ycur,zcur);
                }
        }

        /* Check for conversion back to solid ettringite */
        else if(check==ETTR){
                pgrow=ran1(seed);
                if(pgrow<=ETTRGROW){
                        mic[xcur] [ycur] [zcur]=ETTR;
                        count[ETTR]+=1;
                        action=0;
                        count[DIFFETTR]-=1;
                }
        }

        /* if last diffusion step and no reaction, convert back to */
        /* solid ettringite */
        if((action!=0)&&(finalstep==1)){
```

```
                action=0;
                count[DIFFETTR]-=1;
                count[ETTR]+=1;
                mic[xcur][ycur][zcur]=ETTR;
        }

        if(action!=0){
                /* if diffusion is possible, execute it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFETTR;
                }
                else{
                        /* indicate that diffusing ettringite remained at */
                        /* original location */
                        action=7;
                }
        }
        return(action);
}
/* routine to add extra pozzolanic CSH when CH reacts at */
/* pozzolanic surface (e.g. silica fume) located at (xpres,ypres,zpres) */
/* Called by movech */
/* Calls moveone and edgecnt */
void extpozz(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int check,sump,xchr,ychr,zchr,fchr,i1,plok,action,numnear;
        long int tries;

/* first try 6 neighboring locations until      */
/*      a) successful                          */
/*      b) all 6 sites are tried or            */
/*      c) 100 tries are made                  */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=100)&&(fchr==0)&&(sump!=30030));i1++){

                /* determine location of neighbor (using periodic boundaries) */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                action=0;
                sump*=moveone(&xchr,&ychr,&zchr,&action,sump);
                if(action==0){printf("Error in value of action in extpozz \n");}
                check=mic[xchr][ychr][zchr];
```

```
                /* if neighbor is porosity, locate the pozzolanic CSH there */
                if(check==POROSITY){
                        mic[xchr][ychr][zchr]=POZZCSH;
                        count[POZZCSH]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
        }

        /* if no neighbor available, locate pozzolanic CSH at random location */
        /* in pore space */
        tries=0;
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
           /* if location is porosity, locate the extra pozzolanic CSH there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,POZZ,CSH,POZZCSH);
                        /* Be sure that one neighboring species is CSH or */
                        /* pozzolanic material */
                        if((tries>5000)||(numnear<26)){
                                mic[xchr][ychr][zchr]=POZZCSH;
                                count[POZZCSH]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
}
/* routine to move a diffusing FH3 species */
/* from location (xcur,ycur,zcur) with nucleation probability nucprob */
/* Called by hydrate */
/* Calls moveone */
int movefh3(xcur,ycur,zcur,finalstep,nucprob)
        int xcur,ycur,zcur,finalstep;
        float nucprob;
{
        int check,xnew,ynew,znew,plok,action,sumold,sumgarb;
        float pgen;
```

```
        /* first check for nucleation */
        pgen=ran1(seed);

        if((nucprob>=pgen)||(finalstep==1)){
                action=0;
                mic[xcur][ycur][zcur]=FH3;
                count[FH3]+=1;
                count[DIFFFH3]-=1;
        }
        else{

                /* determine new location (using periodic boundaries) */
                xnew=xcur;
                ynew=ycur;
                znew=zcur;
                action=0;
                sumold=1;
                sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
                if(action==0){printf("Error in value of action in movefh3 \n");}
                check=mic[xnew][ynew][znew];

                /* check for growth of FH3 crystal */
                if(check==FH3){
                        mic[xcur][ycur][zcur]=FH3;
                        count[FH3]+=1;
                        count[DIFFFH3]-=1;
                        action=0;
                }

                if(action!=0){
                        /* if diffusion is possible, execute it */
                        if(check==POROSITY){
                                mic[xcur][ycur][zcur]=POROSITY;
                                mic[xnew][ynew][znew]=DIFFFH3;
                        }
                        else{
                                /* indicate that diffusing FH3 species */
                                /* remained at original location */
                                action=7;
                        }
                }
        }
        return(action);
}
/* routine to move a diffusing CH species */
/* from location (xcur,ycur,zcur) with nucleation probability nucprob */
/* Called by hydrate */
```

```
/* Calls moveone and extpozz */
int movech(xcur,ycur,zcur,finalstep,nucprob)
        int xcur,ycur,zcur,finalstep;
        float nucprob;
{
        int check,xnew,ynew,znew,plok,action,sumgarb,sumold;
        float pexp,pgen,pfix;

        /* first check for nucleation */
        pgen=ran1(seed);
        if((nucprob>=pgen)||(finalstep==1)){
                action=0;
                mic[xcur][ycur][zcur]=CH;
                count[DIFFCH]-=1;
                count[CH]+=1;
        }
        else{

                /* determine new location (using periodic boundaries) */
                xnew=xcur;
                ynew=ycur;
                znew=zcur;
                action=0;
                sumold=1;
                sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
                if(action==0){printf("Error in value of action in movech \n");}
                check=mic[xnew][ynew][znew];

                /* check for growth of CH crystal */
                if((check==CH)&&(pgen<=CHGROW)){
                        mic[xcur][ycur][zcur]=CH;
                        count[DIFFCH]-=1;
                        count[CH]+=1;
                        action=0;
                }

                /* check for pozzolanic reaction */
/* 36.41 units CH can react with 27 units of S */
                else if((pgen<=ppozz)&&(check==POZZ)&&(npr<=(int)(
                      (float)nfill*1.35))){
                        action=0;
                        mic[xcur][ycur][zcur]=POZZCSH;
                        count[POZZCSH]+=1;
                        /* update counter of number of diffusing CH */
                        /* which have reacted pozzolanically */
                        npr+=1;
                        count[DIFFCH]-=1;
```

```
        /* Convert pozzolan to pozzolanic CSH as needed */
        pfix=ran1(seed);
        if(pfix<=(1./1.35)){
                mic[xnew][ynew][znew]=POZZCSH;
                count[POZZ]-=1;
                count[POZZCSH]+=1;
        }
        /* allow for extra pozzolanic CSH as needed */
        pexp=ran1(seed);
        /* should form 101.81 units of pozzolanic CSH for */
        /* each 36.41 units of CH and 27 units of S */
        /* 1.05466=(101.81-36.41-27)/36.41 */
        extpozz(xcur,ycur,zcur);
        if(pexp<=0.05466){
                extpozz(xcur,ycur,zcur);
        }
}
else if(check==DIFFAS){
        action=0;
        mic[xcur][ycur][zcur]=STRAT;
        count[STRAT]+=1;
        /* update counter of number of diffusing CH */
        /* which have reacted to form stratlingite */
        nasr+=1;
        count[DIFFCH]-=1;
        /* Convert DIFFAS to STRAT as needed */
        pfix=ran1(seed);
        if(pfix<=0.7538){
                mic[xnew][ynew][znew]=STRAT;
                count[STRAT]+=1;
                count[DIFFAS]-=1;
        }
        /* allow for extra stratlingite as needed */
        /* 1.5035=(215.63-66.2-49.9)/66.2 */
        extstrat(xcur,ycur,zcur);
        pexp=ran1(seed);
        if(pexp<=0.5035){
                extstrat(xcur,ycur,zcur);
        }
}

if(action!=0){
        /* if diffusion is possible, execute it */
        if(check==POROSITY){
                mic[xcur][ycur][zcur]=POROSITY;
                mic[xnew][ynew][znew]=DIFFCH;
        }
```

```
                        else{
                                /* indicate that diffusing CH species */
                                /* remained at original location */
                                action=7;
                        }
                }
        }
        return(action);
}
/* routine to add extra C3AH6 when diffusing C3A nucleates or reacts at */
/* C3AH6 surface at location (xpres,ypres,zpres) */
/* Called by movec3a */
/* Calls moveone and edgecnt */
void extc3ah6(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int check,sump,xchr,ychr,zchr,fchr,i1,plok,action,numnear;
        long int tries;

/* First try 6 neighboring locations until      */
/*      a) successful                          */
/*      b) all 6 sites are tried or            */
/*      c) 100 random attempts are made        */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=100)&&(fchr==0)&&(sump!=30030));i1++){

                /* determine new coordinates (using periodic boundaries) */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                action=0;
                sump*=moveone(&xchr,&ychr,&zchr,&action,sump);
                if(action==0){printf("Error in action value in extc3ah6 \n");}
                check=mic[xchr][ychr][zchr];

                /* if neighbor is pore space, convert it to C3AH6 */
                if(check==POROSITY){
                        mic[xchr][ychr][zchr]=C3AH6;
                        count[C3AH6]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
        }

        /* if unsuccessful, add C3AH6 at random location in pore space */
        tries=0;
```

```
        while(fchr==0){
                tries+=1;
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];

                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,C3AH6,C3A,C3AH6);
                        /* Be sure that new C3AH6 is in contact with */
                        /* at least one C3AH6 or C3A */
                        if((tries>5000)||(numnear<26)){
                                mic[xchr][ychr][zchr]=C3AH6;
                                count[C3AH6]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
}
/* routine to move a diffusing C3A species */
/* from location (xcur,ycur,zcur) with nucleation probability of nucprob */
/* Called by hydrate */
/* Calls extc3ah6, moveone, extettr, and extafm */
int movec3a(xcur,ycur,zcur,finalstep,nucprob)
        int xcur,ycur,zcur,finalstep;
        float nucprob;
{
        int check,xnew,ynew,znew,plok,action,sumgarb,sumold;
        int xexp,yexp,zexp,nexp,iexp,newact;
        float pgen,pexp,pafm,pgrow,p2diff;

        /* First be sure that a diffusing C3A species is at (xcur,ycur,zcur) */
        if(mic[xcur][ycur][zcur]!=DIFFC3A){
                action=0;
                return(action);
        }

        /* Check for nucleation into solid C3AH6 */
        pgen=ran1(seed);
        p2diff=ran1(seed);

        if((nucprob>=pgen)||(finalstep==1)){
                action=0;
```

```
        mic[xcur][ycur][zcur]=C3AH6;
        count[C3AH6]+=1;
        /* decrement count of diffusing C3A species */
        count[DIFFC3A]-=1;
        /* allow for probabilistic-based expansion of C3AH6 */
        /* crystal to account for volume stoichiometry */
        pexp=ran1(seed);
        if(pexp<=0.69){
                extc3ah6(xcur,ycur,zcur);
        }
}
else{
        /* determine new coordinates (using periodic boundaries) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumold=1;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in movec3a \n");}
        check=mic[xnew][ynew][znew];

        /* check for growth of C3AH6 crystal */
        if(check==C3AH6){
        pgrow=ran1(seed);
                /* Try to slow down growth of C3AH6 crystals to */
                /* promote ettringite and Afm formation */
                if(pgrow<=C3AH6GROW){
                        mic[xcur][ycur][zcur]=C3AH6;
                        count[C3AH6]+=1;
                        count[DIFFC3A]-=1;
                        action=0;
                 /* allow for probabilistic-based expansion of C3AH6 */
                 /* crystal to account for volume stoichiometry */
                        pexp=ran1(seed);
                        if(pexp<=0.69){
                                extc3ah6(xcur,ycur,zcur);
                        }
                }
        }

        /* examine reaction with diffusing gypsum to form ettringite */
        /* Only allow reaction with diffusing gypsum */
        else if((check==DIFFGYP)&&(p2diff<0.01)){
                /* convert diffusing gypsum to ettringite */
                mic[xnew][ynew][znew]=ETTR;
                count[ETTR]+=1;
```

```
                        /* decrement counts of diffusing gypsum */
                        count[DIFFGYP]-=1;
                        action=0;

/* convert diffusing C3A to solid ettringite or else leave as a diffusing C3A */
                        pexp=ran1(seed);
                        nexp=2;
                        if(pexp<=0.40){
                                mic[xcur][ycur][zcur]=ETTR;
                                count[ETTR]+=1;
                                count[DIFFC3A]-=1;
                                nexp=1;
                        }
                        else{
                /* indicate that diffusing species remains in current location */
                                action=7;
                                nexp=2;
                        }

        /* Perform expansion that occurs when ettringite is formed */
        /* xexp, yexp and zexp are the coordinates of the last ettringite */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extettr(xexp,yexp,zexp,0);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
                                                if(xexp>=SYSIZE){xexp=0;}
                                                break;
                                        case 3:
                                                yexp-=1;
                                                if(yexp<0){yexp=(SYSIZEM1);}
                                                break;
                                        case 4:
                                                yexp+=1;
                                                if(yexp>=SYSIZE){yexp=0;}
                                                break;
                                        case 5:
                                                zexp-=1;
```

```
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                                 }
                        }

                  /* probabilistic-based expansion for last ettringite pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.30){
                                newact=extettr(xexp,yexp,zexp,0);
                        }
                }
            /* examine reaction with diffusing hemihydrate to form ettringite */
                /* Only allow reaction with diffusing hemihydrate */
                else if((check==DIFFHEM)&&(p2diff<0.01)){
                        /* convert diffusing hemihydrate to ettringite */
                        mic[xnew][ynew][znew]=ETTR;
                        count[ETTR]+=1;
                        /* decrement counts of diffusing hemihydrate */
                        count[DIFFHEM]-=1;
                        action=0;

/* convert diffusing C3A to solid ettringite or else leave as a diffusing C3A */
                        pexp=ran1(seed);
                        nexp=3;
                        if(pexp<=0.5583){
                                mic[xcur][ycur][zcur]=ETTR;
                                count[ETTR]+=1;
                                count[DIFFC3A]-=1;
                                nexp=2;
                        }
                        else{
              /* indicate that diffusing species remains in current location */
                                action=7;
                                nexp=3;
                        }

        /* Perform expansion that occurs when ettringite is formed */
        /* xexp, yexp and zexp are the coordinates of the last ettringite */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
```

```
                zexp=znew;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,0);
                        /* update xexp, yexp and zexp */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                         }
                }

          /* probabilistic-based expansion for last ettringite pixel */
                pexp=ran1(seed);
                if(pexp<=0.6053){
                        newact=extettr(xexp,yexp,zexp,0);
                }
           }
        /* examine reaction with diffusing anhydrite to form ettringite */
           /* Only allow reaction with diffusing anhydrite */
           else if((check==DIFFANH)&&(p2diff<0.01)){
                   /* convert diffusing anhydrite to ettringite */
                   mic[xnew][ynew][znew]=ETTR;
                   count[ETTR]+=1;
                   /* decrement counts of diffusing anhydrite */
```

```
                        count[DIFFANH]-=1;
                        action=0;

/* convert diffusing C3A to solid ettringite or else leave as a diffusing C3A */
                        pexp=ran1(seed);
                        nexp=3;
                        if(pexp<=0.569){
                                mic[xcur][ycur][zcur]=ETTR;
                                count[ETTR]+=1;
                                count[DIFFC3A]-=1;
                                nexp=2;
                        }
                        else{
                /* indicate that diffusing species remains in current location */
                                action=7;
                                nexp=3;
                        }

        /* Perform expansion that occurs when ettringite is formed */
        /* xexp, yexp and zexp are the coordinates of the last ettringite */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extettr(xexp,yexp,zexp,0);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
                                                if(xexp>=SYSIZE){xexp=0;}
                                                break;
                                        case 3:
                                                yexp-=1;
                                                if(yexp<0){yexp=(SYSIZEM1);}
                                                break;
                                        case 4:
                                                yexp+=1;
                                                if(yexp>=SYSIZE){yexp=0;}
                                                break;
                                        case 5:
                                                zexp-=1;
                                                if(zexp<0){zexp=(SYSIZEM1);}
```

```
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                                 }
                        }

                  /* probabilistic-based expansion for last ettringite pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.6935){
                                newact=extettr(xexp,yexp,zexp,0);
                        }
                }
                /* examine reaction with diffusing CaCl2 to form FREIDEL */
                /* Only allow reaction with diffusing CaCl2 */
                else if(check==DIFFCACL2){
                        /* convert diffusing C3A to Freidel's salt */
                        mic[xcur][ycur][zcur]=FREIDEL;
                        count[FREIDEL]+=1;
                        /* decrement counts of diffusing C3A and CaCl2 */
                        count[DIFFC3A]-=1;
                        action=0;

/* convert diffusing CACL2 to solid FREIDEL or else leave as a diffusing CACL2 */
                        pexp=ran1(seed);
                        nexp=2;
                        if(pexp<=0.5793){
                                mic[xnew][ynew][znew]=FREIDEL;
                                count[FREIDEL]+=1;
                                count[DIFFCACL2]-=1;
                                nexp=1;
                        }
                        else{
                                nexp=2;
                        }

        /* Perform expansion that occurs when Freidel's salt is formed */
        /* xexp, yexp and zexp are the coordinates of the last FREIDEL */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extfreidel(xexp,yexp,zexp);
```

```
                /* update xexp, yexp and zexp */
                switch (newact){
                        case 1:
                                xexp-=1;
                                if(xexp<0){xexp=(SYSIZEM1);}
                                break;
                        case 2:
                                xexp+=1;
                                if(xexp>=SYSIZE){xexp=0;}
                                break;
                        case 3:
                                yexp-=1;
                                if(yexp<0){yexp=(SYSIZEM1);}
                                break;
                        case 4:
                                yexp+=1;
                                if(yexp>=SYSIZE){yexp=0;}
                                break;
                        case 5:
                                zexp-=1;
                                if(zexp<0){zexp=(SYSIZEM1);}
                                break;
                        case 6:
                                zexp+=1;
                                if(zexp>=SYSIZE){zexp=0;}
                                break;
                        default:
                                break;
                 }
        }

  /* probabilistic-based expansion for last FREIDEL pixel */
        pexp=ran1(seed);
        if(pexp<=0.3295){
                newact=extfreidel(xexp,yexp,zexp);
        }
}
/* examine reaction with diffusing CAS2 to form STRAT */
/* Only allow reaction with diffusing (not solid) CAS2 */
else if(check==DIFFCAS2){
        /* convert diffusing CAS2 to stratlingite */
        mic[xnew][ynew][znew]=STRAT;
        count[STRAT]+=1;
        /* decrement counts of diffusing C3A and CAS2 */
        count[DIFFCAS2]-=1;
        action=0;
```

```
/* convert diffusing C3A to solid STRAT or else leave as a diffusing C3A */
                        pexp=ran1(seed);
                        nexp=3;
                        if(pexp<=0.886){
                                mic[xcur][ycur][zcur]=STRAT;
                                count[STRAT]+=1;
                                count[DIFFC3A]-=1;
                                nexp=2;
                        }
                        else{
                                action=7;
                                nexp=3;
                        }

        /* Perform expansion that occurs when stratlingite is formed */
        /* xexp, yexp and zexp are the coordinates of the last STRAT */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extstrat(xexp,yexp,zexp);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
                                                if(xexp>=SYSIZE){xexp=0;}
                                                break;
                                        case 3:
                                                yexp-=1;
                                                if(yexp<0){yexp=(SYSIZEM1);}
                                                break;
                                        case 4:
                                                yexp+=1;
                                                if(yexp>=SYSIZE){yexp=0;}
                                                break;
                                        case 5:
                                                zexp-=1;
                                                if(zexp<0){zexp=(SYSIZEM1);}
                                                break;
                                        case 6:
                                                zexp+=1;
                                                if(zexp>=SYSIZE){zexp=0;}
```

```
                                        break;
                                default:
                                        break;
                                }
                        }

                  /* probabilistic-based expansion for last STRAT pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.286){
                                newact=extstrat(xexp,yexp,zexp);
                        }
                }
/* check for reaction with diffusing or solid ettringite to form AFm */
/* reaction at solid ettringite only possible if ettringite is soluble */
/* and even then on a limited bases to avoid a great formation of AFm */
/* when ettringite first becomes soluble */
                pgrow=ran1(seed);
   if((check==DIFFETTR)||((check==ETTR)&&(soluble[ETTR]==1)&&(pgrow<=C3AETTR))){
                /* convert diffusing or solid ettringite to AFm */
                mic[xnew][ynew][znew]=AFM;
                count[AFM]+=1;
                /* decrement count of ettringite */
                count[check]-=1;
                action=0;

                /* convert diffusing C3A to AFm or leave as diffusing C3A */
                pexp=ran1(seed);
                if(pexp<=0.2424){
                        mic[xcur][ycur][zcur]=AFM;
                        count[AFM]+=1;
                        count[DIFFC3A]-=1;
                        pafm=(-0.1);
                }
                else{
                        action=7;
                        pafm=0.04699;
                }

                /* probabilistic-based expansion for new AFm pixel */
                pexp=ran1(seed);
                if(pexp<=pafm){
                        extafm(xnew,ynew,znew);
                }
        }
        if((action!=0)&&(action!=7)){

                /* if diffusion is possible, execute it */
```

```
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFC3A;
                }
                else{
                        /* indicate that diffusing C3A remained */
                        /* at original location */
                        action=7;
                }
        }
        }
        return(action);
}
/* routine to move a diffusing C4A species */
/* from location (xcur,ycur,zcur) with nucleation probability of nucprob */
/* Called by hydrate */
/* Calls extc3ah6, moveone, extettr, and extafm */
int movec4a(xcur,ycur,zcur,finalstep,nucprob)
        int xcur,ycur,zcur,finalstep;
        float nucprob;
{
        int check,xnew,ynew,znew,plok,action,sumgarb,sumold;
        int xexp,yexp,zexp,nexp,iexp,newact;
        float pgen,pexp,pafm,pgrow,p2diff;

        /* First be sure that a diffusing C4A species is at (xcur,ycur,zcur) */
        if(mic[xcur][ycur][zcur]!=DIFFC4A){
                action=0;
                return(action);
        }

        /* Check for nucleation into solid C3AH6 */
        pgen=ran1(seed);
        p2diff=ran1(seed);

        if((nucprob>=pgen)||(finalstep==1)){
                action=0;
                mic[xcur][ycur][zcur]=C3AH6;
                count[C3AH6]+=1;
                /* decrement count of diffusing C3A species */
                count[DIFFC4A]-=1;
                /* allow for probabilistic-based expansion of C3AH6 */
                /* crystal to account for volume stoichiometry */
                pexp=ran1(seed);
                if(pexp<=0.69){
                        extc3ah6(xcur,ycur,zcur);
                }
```

```
        }
        else{
                /* determine new coordinates (using periodic boundaries) */
                xnew=xcur;
                ynew=ycur;
                znew=zcur;
                action=0;
                sumold=1;
                sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
                if(action==0){printf("Error in value of action in movec4a \n");}
                check=mic[xnew][ynew][znew];

                /* check for growth of C3AH6 crystal */
                if(check==C3AH6){
                pgrow=ran1(seed);
                        /* Try to slow down growth of C3AH6 crystals to */
                        /* promote ettringite and Afm formation */
                        if(pgrow<=C3AH6GROW){
                                mic[xcur][ycur][zcur]=C3AH6;
                                count[C3AH6]+=1;
                                count[DIFFC4A]-=1;
                                action=0;
                         /* allow for probabilistic-based expansion of C3AH6 */
                         /* crystal to account for volume stoichiometry */
                                pexp=ran1(seed);
                                if(pexp<=0.69){
                                        extc3ah6(xcur,ycur,zcur);
                                }
                        }
                }

                /* examine reaction with diffusing gypsum to form ettringite */
                /* Only allow reaction with diffusing gypsum */
                else if((check==DIFFGYP)&&(p2diff<0.01)){
                        /* convert diffusing gypsum to ettringite */
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                        /* decrement counts of diffusing gypsum */
                        count[DIFFGYP]-=1;
                        action=0;

/* convert diffusing C4A to solid ettringite or else leave as a diffusing C4A */
                        pexp=ran1(seed);
                        nexp=2;
                        if(pexp<=0.40){
                                mic[xcur][ycur][zcur]=ETTRC4AF;
                                count[ETTRC4AF]+=1;
```

```
                        count[DIFFC4A]-=1;
                        nexp=1;
                }
                else{
        /* indicate that diffusing species remains in current location */
                        action=7;
                        nexp=2;
                }

/* Perform expansion that occurs when ettringite is formed */
/* xexp, yexp and zexp are the coordinates of the last ettringite */
/* pixel to be added */
                xexp=xnew;
                yexp=ynew;
                zexp=znew;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,1);
                        /* update xexp, yexp and zexp */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                         }
                }
```

```
                  /* probabilistic-based expansion for last ettringite pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.30){
                                newact=extettr(xexp,yexp,zexp,1);
                        }
                }
                /* examine reaction with diffusing hemi to form ettringite */
                /* Only allow reaction with diffusing hemihydrate */
                else if((check==DIFFHEM)&&(p2diff<0.01)){
                        /* convert diffusing hemihydrate to ettringite */
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                        /* decrement counts of diffusing hemihydrate */
                        count[DIFFHEM]-=1;
                        action=0;

/* convert diffusing 43A to solid ettringite or else leave as a diffusing C4A */
                        pexp=ran1(seed);
                        nexp=3;
                        if(pexp<=0.5583){
                                mic[xcur][ycur][zcur]=ETTRC4AF;
                                count[ETTRC4AF]+=1;
                                count[DIFFC4A]-=1;
                                nexp=2;
                        }
                        else{
              /* indicate that diffusing species remains in current location */
                                action=7;
                                nexp=3;
                        }

        /* Perform expansion that occurs when ettringite is formed */
        /* xexp, yexp and zexp are the coordinates of the last ettringite */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extettr(xexp,yexp,zexp,1);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
```

```
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                                }
                        }

                  /* probabilistic-based expansion for last ettringite pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.6053){
                                newact=extettr(xexp,yexp,zexp,1);
                        }
                   }
                /* examine reaction with diffusing anhydrite to form ettringite */
                   /* Only allow reaction with diffusing anhydrite */
                   else if((check==DIFFANH)&&(p2diff<0.01)){
                        /* convert diffusing anhydrite to ettringite */
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                        /* decrement counts of diffusing anhydrite */
                        count[DIFFANH]-=1;
                        action=0;

/* convert diffusing C4A to solid ettringite or else leave as a diffusing C4A */
                        pexp=ran1(seed);
                        nexp=3;
                        if(pexp<=0.569){
                                mic[xcur][ycur][zcur]=ETTRC4AF;
                                count[ETTRC4AF]+=1;
                                count[DIFFC4A]-=1;
                                nexp=2;
```

```
                }
                else{
        /* indicate that diffusing species remains in current location */
                        action=7;
                        nexp=3;
                }

/* Perform expansion that occurs when ettringite is formed */
/* xexp, yexp and zexp are the coordinates of the last ettringite */
/* pixel to be added */
                xexp=xnew;
                yexp=ynew;
                zexp=znew;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,1);
                        /* update xexp, yexp and zexp */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                         }
                }

            /* probabilistic-based expansion for last ettringite pixel */
```

```
                        pexp=ran1(seed);
                        if(pexp<=0.6935){
                                newact=extettr(xexp,yexp,zexp,1);
                        }
                }
                /* examine reaction with diffusing CaCl2 to form FREIDEL */
                /* Only allow reaction with diffusing CaCl2 */
                else if(check==DIFFCACL2){
                        /* convert diffusing C4A to Freidel's salt */
                        mic[xcur][ycur][zcur]=FREIDEL;
                        count[FREIDEL]+=1;
                        /* decrement counts of diffusing C4A and CaCl2 */
                        count[DIFFC4A]-=1;
                        action=0;

/* convert diffusing CACL2 to solid FREIDEL or else leave as a diffusing CACL2 */
                        pexp=ran1(seed);
                        nexp=2;
                        if(pexp<=0.5793){
                                mic[xnew][ynew][znew]=FREIDEL;
                                count[FREIDEL]+=1;
                                count[DIFFCACL2]-=1;
                                nexp=1;
                        }
                        else{
                                nexp=2;
                        }

        /* Perform expansion that occurs when Freidel's salt is formed */
        /* xexp, yexp and zexp are the coordinates of the last FREIDEL */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extfreidel(xexp,yexp,zexp);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
                                                if(xexp>=SYSIZE){xexp=0;}
                                                break;
                                        case 3:
```

```
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                                 }
                        }

                  /* probabilistic-based expansion for last FREIDEL pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.3295){
                                newact=extfreidel(xexp,yexp,zexp);
                        }
                }
                /* examine reaction with diffusing CAS2 to form STRAT */
                /* Only allow reaction with diffusing (not solid) CAS2 */
                else if(check==DIFFCAS2){
                        /* convert diffusing CAS2 to stratlingite */
                        mic[xnew][ynew][znew]=STRAT;
                        count[STRAT]+=1;
                        /* decrement counts of diffusing CAS2 */
                        count[DIFFCAS2]-=1;
                        action=0;

/* convert diffusing C4A to solid STRAT or else leave as a diffusing C4A */
                        pexp=ran1(seed);
                        nexp=3;
                        if(pexp<=0.886){
                                mic[xcur][ycur][zcur]=STRAT;
                                count[STRAT]+=1;
                                count[DIFFC4A]-=1;
                                nexp=2;
                        }
                        else{
                                action=7;
```

```
                        nexp=3;
                }

        /* Perform expansion that occurs when stratlingite is formed */
        /* xexp, yexp and zexp are the coordinates of the last STRAT */
        /* pixel to be added */
                xexp=xnew;
                yexp=ynew;
                zexp=znew;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extstrat(xexp,yexp,zexp);
                        /* update xexp, yexp and zexp */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                         }
                }

           /* probabilistic-based expansion for last STRAT pixel */
                pexp=ran1(seed);
                if(pexp<=0.286){
                        newact=extstrat(xexp,yexp,zexp);
                }
```

```
                }
/* check for reaction with diffusing or solid ettringite to form AFm */
/* reaction at solid ettringite only possible if ettringite is soluble */
/* and even then on a limited bases to avoid a great formation of AFm */
/* when ettringite first becomes soluble */
                pgrow=ran1(seed);
   if((check==DIFFETTR)||((check==ETTR)&&(soluble[ETTR]==1)&&(pgrow<=C3AETTR))){
                /* convert diffusing or solid ettringite to AFm */
                mic[xnew][ynew][znew]=AFM;
                count[AFM]+=1;
                /* decrement count of ettringite */
                count[check]-=1;
                action=0;

                /* convert diffusing C4A to AFm or leave as diffusing C4A */
                pexp=ran1(seed);
                if(pexp<=0.2424){
                        mic[xcur][ycur][zcur]=AFM;
                        count[AFM]+=1;
                        count[DIFFC4A]-=1;
                        pafm=(-0.1);
                }
                else{
                        action=7;
                        pafm=0.04699;
                }

                /* probabilistic-based expansion for new AFm pixel */
                pexp=ran1(seed);
                if(pexp<=pafm){
                        extafm(xnew,ynew,znew);
                }
        }
        if((action!=0)&&(action!=7)){

                /* if diffusion is possible, execute it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFC4A;
                }
                else{
                        /* indicate that diffusing C4A remained */
                        /* at original location */
                        action=7;
                }
        }
        }
```

```
        return(action);
}
/* routine to oversee hydration by updating position of all */
/* remaining diffusing species */
/* Calls movech, movec3a, movefh3, moveettr, movecsh, and movegyp */
void hydrate(fincyc,stepmax,chpar1,chpar2,hgpar1,hgpar2,fhpar1,fhpar2,gypar1,gypar2)
        int fincyc,stepmax;
        float chpar1,chpar2,hgpar1,hgpar2,fhpar1,fhpar2,gypar1,gypar2;
{
        int xpl,ypl,zpl,phpl,agepl,xpnew,ypnew,zpnew;
        float chprob,c3ah6prob,fh3prob,gypprob;
        long int icnt,nleft,ntodo,ndale;
        int istep,termflag,reactf;
        float beterm;
        struct ants *curant,*antgone;

        ntodo=nmade;
        nleft=nmade;
        termflag=0;

/* Perform diffusion until all reacted or max. # of diffusion steps reached */
        for(istep=1;((istep<=stepmax)&&(nleft>0));istep++){
                if((fincyc==1)&&(istep==stepmax)){termflag=1;}

                nleft=0;
                ndale=0;

                /* determine probabilities for CH and C3AH6 nucleation */
                beterm=exp(-(double)(count[DIFFCH])/chpar2);
                chprob=chpar1*(1.-beterm);
                beterm=exp(-(double)(count[DIFFC3A])/hgpar2);
                c3ah6prob=hgpar1*(1.-beterm);
                beterm=exp(-(double)(count[DIFFFH3])/fhpar2);
                fh3prob=fhpar1*(1.-beterm);
                beterm=exp(-(double)(count[DIFFANH]+count[DIFFHEM])/gypar2);
                gypprob=gypar1*(1.-beterm);

                /* Process each diffusing species in turn */
                curant=headant->nextant;
                while(curant!=NULL){
                        ndale+=1;
                        xpl=curant->x;
                        ypl=curant->y;
                        zpl=curant->z;
                        phpl=curant->id;
                        agepl=curant->cycbirth;
```

```
/* based on ID, call appropriate routine to process diffusing species */
                  switch (phpl) {
                          case DIFFCSH:
                                  reactf=movecsh(xpl,ypl,zpl,termflag,agepl);
                                  break;
                          case DIFFANH:
                                  reactf=moveanh(xpl,ypl,zpl,termflag,gypprob);
                                  break;
                          case DIFFHEM:
                                  reactf=movehem(xpl,ypl,zpl,termflag,gypprob);
                                  break;
                          case DIFFCH:
                            reactf=movech(xpl,ypl,zpl,termflag,chprob);
                            break;
                          case DIFFFH3:
                            reactf=movefh3(xpl,ypl,zpl,termflag,fh3prob);
                            break;
                          case DIFFGYP:
                                  reactf=movegyp(xpl,ypl,zpl,termflag);
                                  break;
                          case DIFFC3A:
                           reactf=movec3a(xpl,ypl,zpl,termflag,c3ah6prob);
                           break;
                          case DIFFC4A:
                           reactf=movec4a(xpl,ypl,zpl,termflag,c3ah6prob);
                           break;
                          case DIFFETTR:
                                  reactf=moveettr(xpl,ypl,zpl,termflag);
                                  break;
                          case DIFFCACL2:
                                  reactf=movecacl2(xpl,ypl,zpl,termflag);
                                  break;
                          case DIFFCAS2:
                                  reactf=movecas2(xpl,ypl,zpl,termflag);
                                  break;
                          case DIFFAS:
                                  reactf=moveas(xpl,ypl,zpl,termflag);
                                  break;
                          default:
                                  printf("Error in ID of phase \n");
                                  break;
                  }

                  /* if no reaction */
                  if(reactf!=0){
                          nleft+=1;
                          xpnew=xpl;
```

```
                ypnew=ypl;
                zpnew=zpl;

                /* update location of diffusing species */
                switch (reactf) {
                        case 1:
                                xpnew-=1;
                                if(xpnew<0){xpnew=(SYSIZEM1);}
                                break;
                        case 2:
                                xpnew+=1;
                                if(xpnew>=SYSIZE){xpnew=0;}
                                break;
                        case 3:
                                ypnew-=1;
                                if(ypnew<0){ypnew=(SYSIZEM1);}
                                break;
                        case 4:
                                ypnew+=1;
                                if(ypnew>=SYSIZE){ypnew=0;}
                                break;
                        case 5:
                                zpnew-=1;
                                if(zpnew<0){zpnew=(SYSIZEM1);}
                                break;
                        case 6:
                                zpnew+=1;
                                if(zpnew>=SYSIZE){zpnew=0;}
                                break;
                        default:
                                break;
                }

                /* store new location of diffusing species */
                curant->x=xpnew;
                curant->y=ypnew;
                curant->z=zpnew;
                curant->id=phpl;
                curant=curant->nextant;
        } /* end of reactf!=0 loop */
        /* else remove ant from list */
        else{
                if(ndale==1){
                        headant->nextant=curant->nextant;
                }
                else{
                     (curant->prevant)->nextant=curant->nextant;
```

```
                                }
                                if(curant->nextant!=NULL){
                                     (curant->nextant)->prevant=curant->prevant;
                                }
                                else{
                                        tailant=curant->prevant;
                                }
                                antgone=curant;
                                curant=curant->nextant;
                                free(antgone);
                                ngoing-=1;
                        }
                } /* end of curant loop */
                ntodo=nleft;
        } /* end of istep loop */
}
```

C.6 Listing for disrealnew.c

```
/*************************************************************************/
/*                                                                       */
/*      Program disrealnew.c to hydrate three-dimensional cement         */
/*              particles in a 3-D box with periodic boundaries.         */
/*              Uses cellular automata techniques and preserves correct  */
/*              hydration volume stoichiometry.                          */
/*      Programmer: Dale P. Bentz                                        */
/*                  Building and Fire Research Laboratory                */
/*                  NIST                                                 */
/*                  100 Bureau Drive Mail Stop 8621                      */
/*                  Gaithersburg, MD  20899   USA                        */
/*                  (301) 975-5865      FAX: 301-990-6891                */
/*                  E-mail: dale.bentz@nist.gov                          */
/*                                                                       */
/*************************************************************************/
/* Heat of formation data added: 12/92 */
/* Three-dimensional version created 7/94 */
/* General C version created 8/94 */
/* Modified to have pseudo-continuous dissolution 11/94 */
/* In this program, dissolved silicates are located close */
/* to dissolution source within a 17*17*17 box centered at dissolution source */
/* Hydration under self-desiccating conditions included 9/95 */
/* Additions for adiabatic hydration conditions included 9/96 */
/* Adjustments for pozzolanic reaction included 11/96 */
/* Additions for calcium chloride to Freidel's salt added 4/97 */
/* Additions for stratlingite formation added 4/97 */
/* Additions for anhydrite to gypsum conversion added 5/97 */
/* Additions for calcium aluminodisilicate added 7/97 */
/* Temperature-variable C-S-H properties added 8/98 */
/* molar volumes and water consumption based on values */
/* given in Geiker thesis on chemical shrinkage */
/* Modelling of induction period based on an impermeable layer */
/* theory */
/* Ettringite unstable above 70 C added 9/98 */
/* Hemihydrate to gypsum conversion added 9/98 */
/* Decided to base alpha only on four major clinker phases 9/98 */
/* Updated dissolution to allow next nearest neighbors, etc. 9/98 */
/* Created stable iron-rich ettringite 3/99 */
/* Phases renumbered for simplification 1/00 */
#include <stdio.h>
#include <math.h>

#define MAXCYC 30000    /* Maximum number of cycles of hydration */
/* For hydration under sealed conditions: */
#define CUBEMAX 7      /* Maximum cube size for checking pore size */
#define CUBEMIN 3      /* Minimum cube size for checking pore size */
```

```
#define SYSIZE 100    /* System size in pixels per dimension */
#define SYSIZEM1 99    /* System size -1 */
#define DISBIAS 30.0  /* Dissolution bias- to change all dissolution rates */
#define DISMIN 0.001  /* Minimum dissolution for C3S dissolution */
#define DISMIN2 0.00025  /* Minimum dissolution for C2S dissolution */
#define DETTRMAX 1200 /* Maximum allowed # of ettringite diffusing species */
#define DGYPMAX 2000    /* Maximum allowed # of gypsum diffusing species */
#define DCACL2MAX 2000 /* Maximum allowed # of CaCl2 diffusing species */
#define DCAS2MAX 2000 /* Maximum allowed # of CAS2 diffusing species */
#define CHCRIT 50.0   /* Scale parameter to adjust CH dissolution probability */
#define C3AH6CRIT 10.0   /* Scale par. to adjust C3AH6 dissolution prob. */
#define C3AH6GROW 0.01  /* Probability for C3AH6 growth */
#define CHGROW 1.0      /* Probability for CH growth */
#define ETTRGROW 0.002   /* Probability for ettringite growth */
#define C3AETTR 0.001    /* Probability for reaction of diffusing C3A
                            with ettringite  */
#define PPOZZ 0.05      /* base probability for pozzolanic reaction */
#define PCSH2CSH 0.002  /* probability for CSH dissolution */
/* for conversion of C-S-H to pozz. C-S-H */
#define A0_CHSOL 1.325  /* Parameters for variation of CH solubility with */
#define A1_CHSOL 0.008162 /* temperature (data from Taylor- Cement Chemistry) */
#define CSHSCALE 70000.  /* scale factor for CSH controlling induction */
#define NEIGHBORS 18 /* number of neighbors to consider (6, 18, or 26) */
/* define IDs for each phase used in model */
/* To add a solid phase, insert new phase before ABSGYP */
/*  and increment all subsequent IDs */
/* To add a diffusing species, insert just before DIFFCACL2 */
/*  and increment all subsequent IDs */
#define POROSITY 0
#define C3S 1
#define C2S 2
#define C3A 3
#define C4AF 4
#define GYPSUM 5
#define HEMIHYD 6
#define ANHYDRITE 7
#define POZZ 8
#define INERT 9
#define ASG 10   /* aluminosilicate glass */
#define CAS2 11
#define CH 12
#define CSH 13
#define C3AH6 14
#define ETTR 15
#define ETTRC4AF 16     /* Iron-rich stable ettringite phase */
#define AFM 17
#define FH3 18
```

```
#define POZZCSH 19
#define CACL2 20
#define FREIDEL 21   /* Freidel's salt */
#define STRAT 22   /* stratlingite (C2ASH8) */
#define GYPSUMS 23 /* Gypsum formed from hemihydrate and anhydrite */
#define INERTAGG 24
#define ABSGYP 25
#define DIFFCSH 26
#define DIFFCH 27
#define DIFFGYP 28
#define DIFFC3A 29
#define DIFFC4A 30
#define DIFFFH3 31
#define DIFFETTR 32
#define DIFFAS 33
#define DIFFANH 34
#define DIFFHEM 35
#define DIFFCAS2 36
#define DIFFCACL2 37
#define EMPTYP 38         /* Empty porosity due to self desiccation */
#define OFFSET 50         /* Offset for highlighted potentially soluble pixel */
/* define heat capacities for all components in J/g/C */
/* including free and bound water */
#define Cp_cement 0.75
#define Cp_pozz 0.75
#define Cp_agg 0.84
#define Cp_CH 0.75
#define Cp_h2o 4.18     /* Cp for free water */
#define Cp_bh2o 2.2     /* Cp for bound water */
#define WN 0.23        /* water bound per gram of cement during hydration */
#define WCHSH 0.06     /* water imbibed per gram of cement during chemical
                          shrinkage (estimate) */

/* data structure for diffusing species - to be dynamically allocated */
/* Use of a doubly linked list to allow for easy maintenance */
/* (i.e. insertion and deletion) */
/* Added 11/94 */
/* Note that if SYSIZE exceeds 256, need to change x, y, and z to */
/* int variables */
struct ants{
        char x,y,z,id;
        int cycbirth;
        struct ants *nextant;
        struct ants *prevant;
};

/* data structure for elements to remove to simulate self-desiccation */
```

```
/* once again a doubly linked list */
struct togo{
        int x,y,z,npore;
        struct togo *nexttogo;
        struct togo *prevtogo;
};

/* Global variables */
/* Microstructure stored in array mic of type char to minimize storage */
/* Initial particle IDs stored in array micpart (for assessing set point) */
static char mic [SYSIZE] [SYSIZE] [SYSIZE];
static char micorig [SYSIZE] [SYSIZE] [SYSIZE];
static short int micpart [SYSIZE] [SYSIZE] [SYSIZE];
static short int cshage [SYSIZE] [SYSIZE] [SYSIZE];
/* counts for dissolved and solid species */
long int discount[EMPTYP+1], count[EMPTYP+1];
/* Counts for pozzolan reacted, initial pozzolan, gypsum, ettringite,
initial porosity, and aluminosilicate reacted */
long int npr,nfill,ncsbar,netbar,porinit,nasr;
/* Initial clinker phase counts */
long int c3sinit,c2sinit,c3ainit,c4afinit,anhinit,heminit,chold,chnew;
long int nmade,ngoing,gypready,poregone,poretodo;
long int countkeep,water_left,water_off,pore_off;
int ncyc,cyccnt,cubesize,sealed;
int icyc,burnfreq,setfreq,setflag,porefl1,porefl2,porefl3;
int xoff[27]={1,0,0,-1,0,0,1,1,-1,-1,0,0,0,0,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,0};
int yoff[27]={0,1,0,0,-1,0,1,-1,1,-1,1,-1,1,-1,0,0,0,0,1,-1,1,-1,1,1,-1,-1,0};
int zoff[27]={0,0,1,0,0,-1,0,0,0,0,1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,0};
/* Parameters for kinetic modelling ---- maturity approach */
float ind_time,temp_0,temp_cur,time_cur,E_act,beta,heat_cf,w_to_c,s_to_c,krate;
float alpha_cur,heat_old,heat_new,cemmass,mass_agg,mass_water,mass_fill,Cp_now;
float CH_mass,mass_CH,mass_fill_pozz,E_act_pozz;
/* Arrays for variable CSH molar volume and water consumption */
float molarvcsh[MAXCYC],watercsh[MAXCYC],heatsum,molesh2o;
/* Arrays for dissolution probabilities for each phase */
float disprob[EMPTYP+1],disbase[EMPTYP+1],gypabsprob,ppozz;
/* Arrays for specific gravities, molar volumes, heats of formation, and */
/* molar water consumption for each phase */
float specgrav[EMPTYP+1],molarv[EMPTYP+1],heatf[EMPTYP+1],waterc[EMPTYP+1];
/* Solubility flags and diffusing species created for each phase */
/* Also flag for C1.7SH4.0 to C1.1SH3.9 conversion */
int soluble[EMPTYP+1],creates[EMPTYP+1],csh2flag;
int *seed;       /* Random number seed */
struct ants *headant,*tailant;

/* Supplementary programs */
#include "ran1.c" /* random number generation */
```

```
#include "burn3d.c" /* percolation of porosity assessment */
#include "burnset.c" /* set point assessment */
#include "parthyd.c" /* particle hydration assessment */
#include "hydrealnew.c" /* hydration execution */

/* routine to initialize values for solubilities, molar volumes, etc. */
/* Called by main program */
/* Calls no other routines */
void init()
{
        int i;

        for(i=0;i<=EMPTYP;i++){
                creates[i]=0;
                soluble[i]=0;
                disprob[i]=0.0;
                disbase[i]=0.0;
        }

        /* soluble [x] - flag indicating if phase x is soluble */
        /* disprob [x] - probability of dissolution (relative diss. rate) */
        gypabsprob=0.01; /* One sulfate absorbed per 100 CSH units */
/* Source is from Taylor's Cement Chemistry  for K and Na absorption */
        /* Note that for the first cycle, of the clinker phases only the */
        /* aluminates and gypsum are soluble */
        soluble[C4AF]=1;
        disprob[C4AF]=disbase[C4AF]=0.1/DISBIAS;
        creates[C4AF]=POROSITY;
        soluble[C3S]=0;
        disprob[C3S]=disbase[C3S]=0.8/DISBIAS;
        creates[C3S]=DIFFCSH;
        soluble[C2S]=0;
        disprob[C2S]=disbase[C2S]=0.15/DISBIAS;
        creates[C2S]=DIFFCSH;
        soluble[C3A]=1;
        /* increased back to 0.4 from 0.25 7/8/99 */
        disprob[C3A]=disbase[C3A]=0.4/DISBIAS;
        creates[C3A]=POROSITY;
        soluble[GYPSUM]=1;
        /* Changed from 0.05 to 0.015  9/29/98 */
        /* Changed to 0.040 10/15/98 */
        /* back to 0.05 from 0.10 7/8/99 */
        disprob[GYPSUM]=disbase[GYPSUM]=0.050;
        /* dissolved gypsum distributed at random throughout microstructure */
        creates[GYPSUM]=POROSITY;
        soluble[GYPSUMS]=1;
        disprob[GYPSUMS]=disbase[GYPSUMS]=disprob[GYPSUM];
```

```
   creates[GYPSUMS]=POROSITY;
   soluble[ANHYDRITE]=1;
   /* set anhydrite dissolution at 4/5ths of that of gypsum */
   /* Source: Uchikawa et al., CCR, 1984 */
   disprob[ANHYDRITE]=disbase[ANHYDRITE]=0.8*disprob[GYPSUM];
  /* dissolved anhydrite distributed at random throughout microstructure */
   creates[ANHYDRITE]=POROSITY;
   soluble[HEMIHYD]=1;
   /* set hemihydrate dissolution at 3 times that of gypsum */
   /* Source: Uchikawa et al., CCR, 1984 */
   disprob[HEMIHYD]=disbase[HEMIHYD]=3.0*disprob[GYPSUM];
/* dissolved hemihydrate distributed at random throughout microstructure */
   creates[HEMIHYD]=POROSITY;
   /* CH soluble to allow for Ostwald ripening of crystals */
   soluble[CH]=1;
   disprob[CH]=disbase[CH]=0.5/DISBIAS;
   creates[CH]=DIFFCH;
   soluble[C3AH6]=1;
   disprob[C3AH6]=disbase[C3AH6]=0.5/DISBIAS;
   creates[C3AH6]=POROSITY;
   /* Ettringite is initially insoluble */
   soluble[ETTR]=0;
   /* Changed to 0.008 from 0.020  3/11/99 */
   disprob[ETTR]=disbase[ETTR]=0.008/DISBIAS;
   creates[ETTR]=DIFFETTR;
   /* Iron-rich ettringite is always insoluble */
   soluble[ETTRC4AF]=0;
   disprob[ETTRC4AF]=disbase[ETTRC4AF]=0.0;
   creates[ETTRC4AF]=ETTRC4AF;
   /* calcium chloride is soluble */
   soluble[CACL2]=1;
   disprob[CACL2]=disbase[CACL2]=0.1/DISBIAS;
   creates[CACL2]=DIFFCACL2;
   /* aluminosilicate glass is soluble */
   soluble[ASG]=1;
   disprob[ASG]=disbase[ASG]=0.9/DISBIAS;
   creates[ASG]=DIFFAS;
   /* calcium aluminodisilicate is soluble */
   soluble[CAS2]=1;
   disprob[CAS2]=disbase[CAS2]=0.1/DISBIAS;
   creates[CAS2]=DIFFCAS2;

   /* establish molar volumes and heats of formation */
   /* molar volumes are in cm^3/mole */
   /* heats of formation are in kJ/mole */
   /* See paper by Fukuhara et al., Cem. & Conc. Res., 11, 407-14, 1981. */
   /* values for Porosity are those of water */
```

```
molarv[POROSITY]=18.0;
heatf[POROSITY]=(-285.83);
waterc[POROSITY]=1.0;
specgrav[POROSITY]=1.0;

molarv[C3S]=71.0;
heatf[C3S]=(-2927.82);
waterc[C3S]=0.0;
specgrav[C3S]=3.21;
/* For improvement in chemical shrinkage correspondence */
/* Changed molar volume of C(1.7)-S-H(4.0) to 108 5/24/95 */
molarv[CSH]=108.;
heatf[CSH]=(-3283.0);
waterc[CSH]=4.0;
specgrav[CSH]=2.12;

molarv[CH]=33.1;
heatf[CH]=(-986.1);
waterc[CH]=1.0;
specgrav[CH]=2.24;

molarv[C2S]=52.;
heatf[C2S]=(-2311.6);
waterc[C2S]=0.0;
specgrav[C2S]=3.28;

molarv[C3A]=89.1;
heatf[C3A]=(-3587.8);
waterc[C3A]=0.0;
specgrav[C3A]=3.03;

molarv[GYPSUM]=74.2;
heatf[GYPSUM]=(-2022.6);
waterc[GYPSUM]=0.0;
specgrav[GYPSUM]=2.32;
molarv[GYPSUMS]=74.2;
heatf[GYPSUMS]=(-2022.6);
waterc[GYPSUMS]=0.0;
specgrav[GYPSUMS]=2.32;

molarv[ANHYDRITE]=52.16;
heatf[ANHYDRITE]=(-1424.6);
waterc[ANHYDRITE]=0.0;
specgrav[ANHYDRITE]=2.61;

molarv[HEMIHYD]=53.17;
heatf[HEMIHYD]=(-1574.65);
```

```
        waterc[HEMIHYD]=0.0;
        specgrav[HEMIHYD]=2.73;

        molarv[C4AF]=128.;
        heatf[C4AF]=(-5090.3);
        waterc[C4AF]=0.0;
        specgrav[C4AF]=3.73;

        molarv[C3AH6]=150.;
        heatf[C3AH6]=(-5548.);
        waterc[C3AH6]=6.0;
        specgrav[C3AH6]=2.52;

/* Changed molar volume of FH3 to 69.8 (specific gravity of 3.0) 5/23/95 */
        molarv[FH3]=69.8;
        heatf[FH3]=(-823.9);
        waterc[FH3]=3.0;
        specgrav[FH3]=3.0;

/* Changed molar volue of ettringite to 735 (spec. gr.=1.7)  5/24/95 */
        molarv[ETTRC4AF]=735;
        heatf[ETTRC4AF]=(-17539.0);
        waterc[ETTRC4AF]=26.0;
        specgrav[ETTRC4AF]=1.7;
/* Changed molar volue of ettringite to 735 (spec. gr.=1.7)  5/24/95 */
        molarv[ETTR]=735;
        heatf[ETTR]=(-17539.0);
        waterc[ETTR]=26.0;
        specgrav[ETTR]=1.7;

        molarv[AFM]=313;
        heatf[AFM]=(-8778.0);
        /* Each mole of AFM which forms requires 12 moles of water, */
        /* two of which are supplied by gypsum in forming ETTR  */
        /* leaving 10 moles to be incorporated from free water */
        waterc[AFM]=10.0;
        specgrav[AFM]=1.99;

        molarv[CACL2]=51.62;
        heatf[CACL2]=(-795.8);
        waterc[CACL2]=0;
        specgrav[CACL2]=2.15;

        molarv[FREIDEL]=296.662;
        /* No data available for heat of formation */
        heatf[FREIDEL]=(0.0);
        /* 10 moles of H2O per mole of Freidel's salt */
```

```
        waterc[FREIDEL]=10.0;
        specgrav[FREIDEL]=1.892;

        /* Basic reaction is 2CH + ASG + 6H --> C2ASH8 (Stratlingite) */
        molarv[ASG]=49.9;
        /* No data available for heat of formation */
        heatf[ASG]=0.0;
        waterc[ASG]=0.0;
        specgrav[ASG]=3.247;

        molarv[CAS2]=100.62;
        /* No data available for heat of formation */
        heatf[CAS2]=0.0;
        waterc[CAS2]=0.0;
        specgrav[CAS2]=2.77;

        molarv[STRAT]=215.63;
        /* No data available for heat of formation */
        heatf[STRAT]=0.0;
        /* 8 moles of water per mole of stratlingite */
        waterc[STRAT]=8.0;
        specgrav[STRAT]=1.94;

        molarv[POZZ]=27.0;
        /* Use heat of formation of SiO2 (quartz) for unreacted pozzolan */
        /* Source- Handbook of Chemistry and Physics */
        heatf[POZZ]=-907.5;
        waterc[POZZ]=0.0;
        specgrav[POZZ]=2.2;

/* Data for Pozzolanic CSH based on work of Atlassi, DeLarrard, */
/* and Jensen */
/* gives a chemical shrinkage of 0.2 g H2O/g CSF */
/* heat of formation estimated based on heat release of */
/* 780 J/g Condensed Silica Fume */
/* Changed stoichiometry to be C(1.1)SH(3.9) to see effect on */
/* results 1/22/97 */
/* MW is 191.8 g/mole */
        /* Changed molar volume to 101.81  3/10/97 */
        molarv[POZZCSH]=101.81;
        waterc[POZZCSH]=3.9;
        specgrav[POZZCSH]=1.884;
        heatf[POZZCSH]=(-2299.1);

/* Assume inert filler has same specific gravity and molar volume as SiO2 */
        molarv[INERT]=27.0;
        heatf[INERT]=0.0;
```

```
        waterc[INERT]=0.0;
        specgrav[INERT]=2.2;

        molarv[ABSGYP]=74.2;
        heatf[ABSGYP]=(-2022.6);
        waterc[ABSGYP]=0.0;
        specgrav[ABSGYP]=2.32;

        molarv[EMPTYP]=18.0;
        heatf[EMPTYP]=(-285.83);
        waterc[EMPTYP]=0.0;
        specgrav[EMPTYP]=1.0;
}

/* routine to check if a pixel located at (xck,yck,zck) is on an edge */
/* (in contact with pore space) in 3-D system */
/* Called by passone */
/* Calls no other routines */
int chckedge(xck,yck,zck)
        int xck,yck,zck;
{
        int edgeback,x2,y2,z2;
        int ip;

        edgeback=0;

        /* Check all six neighboring pixels */
/* with periodic boundary conditions */
        for(ip=0;((ip<NEIGHBORS)&&(edgeback==0));ip++){

                x2=xck+xoff[ip];
                y2=yck+yoff[ip];
                z2=zck+zoff[ip];
                if(x2>=SYSIZE){x2=0;}
                if(x2<0){x2=SYSIZEM1;}
                if(y2>=SYSIZE){y2=0;}
                if(y2<0){y2=SYSIZEM1;}
                if(z2>=SYSIZE){z2=0;}
                if(z2<0){z2=SYSIZEM1;}
                if(mic [x2] [y2] [z2]==POROSITY){
                        edgeback=1;
                }
        }
        return(edgeback);
}

/* routine for first pass through microstructure during dissolution */
```

```
/* low and high indicate phase ID range to check for surface sites */
/* Called by dissolve */
/* Calls chckedge */
void passone(low,high,cycid,cshexflag)
        int low,high,cycid,cshexflag;
{
        int i,xid,yid,zid,phid,iph,edgef,phread,cshcyc;

        /* gypready used to determine if any soluble gypsum remains */
        if((low<=GYPSUM)&&(GYPSUM<=high)){
                gypready=0;
        }
        /* Zero out count for the relevant phases */
        for(i=low;i<=high;i++){
                count[i]=0;
        }
        /* Scan the entire 3-D microstructure */
        for(xid=0;xid<SYSIZE;xid++){
        for(yid=0;yid<SYSIZE;yid++){
        for(zid=0;zid<SYSIZE;zid++){

        phread=mic[xid][yid][zid];
        /* Update heat data and water consumed for solid CSH */
        if((cshexflag==1)&&(phread==CSH)){
                cshcyc=cshage[xid][yid][zid];
                heatsum+=heatf[CSH]/molarvcsh[cshcyc];
                molesh2o+=watercsh[cshcyc]/molarvcsh[cshcyc];
        }
        /* Identify phase and update count */
        phid=60;
        for(i=low;((i<=high)&&(phid==60));i++){

                if(mic [xid][yid][zid]==i){
                        phid=i;
                        /* Update count for this phase */
                        count[i]+=1;
                        if((i==GYPSUM)||(i==GYPSUMS)){
                                gypready+=1;
                        }
                        /* If first cycle, then accumulate initial counts */
                        if((cycid==1)||((cycid==0)&&(ncyc==0))){
                                if(i==POROSITY){porinit+=1;}
                               /* Ordered in terms of likely volume fractions */
                               /* (largest to smallest) to speed execution */
                                else if(i==C3S){c3sinit+=1;}
                                else if(i==C2S){c2sinit+=1;}
                                else if(i==C3A){c3ainit+=1;}
```

```
                                else if(i==C4AF){c4afinit+=1;}
                                else if(i==GYPSUM){ncsbar+=1;}
                                else if(i==ANHYDRITE){anhinit+=1;}
                                else if(i==HEMIHYD){heminit+=1;}
                                else if(i==POZZ){nfill+=1;}
                                else if(i==ETTR){netbar+=1;}
                                else if(i==ETTRC4AF){netbar+=1;}
                        }
                }
        }

        if(phid!=60){
                /* If phase is soluble, see if it is in contact with porosity */
                if((cycid!=0)&&(soluble[phid]==1)){
                        edgef=chckedge(xid,yid,zid);
                        if(edgef==1){
/* Surface eligible species has an ID OFFSET greater than its original value */
                                mic [xid][yid][zid]+=OFFSET;
                        }
                }
        }
        }  /* end of zid */
        }  /* end of yid */
        }  /* end of xid */
}

/* routine to locate a diffusing CSH species near dissolution source */
/* at (xcur,ycur,zcur) */
/* Called by dissolve */
/* Calls no other routines */
int loccsh(xcur,ycur,zcur)
        int xcur,ycur,zcur;
{
        int xpmax,ypmax,effort,tries,xmod,ymod,zmod;
        struct ants *antnew;

        effort=0;    /* effort indicates if appropriate location found */
        tries=0;
        /* 500 tries in immediate vicinity */
        while((effort==0)&&(tries<500)){
                tries+=1;
                xmod=(-8)+(int)(17.*ran1(seed));
                ymod=(-8)+(int)(17.*ran1(seed));
                zmod=(-8)+(int)(17.*ran1(seed));
                if(xmod>8){xmod=8;}
                if(ymod>8){ymod=8;}
                if(zmod>8){zmod=8;}
```

```
                xmod+=xcur;
                ymod+=ycur;
                zmod+=zcur;
                /* Periodic boundaries */
                if(xmod>=SYSIZE){xmod-=SYSIZE;}
                else if(xmod<0){xmod+=SYSIZE;}
                if(zmod>=SYSIZE){zmod-=SYSIZE;}
                else if(zmod<0){zmod+=SYSIZE;}
                if(ymod<0){ymod+=SYSIZE;}
                else if(ymod>=SYSIZE){ymod-=SYSIZE;}

                if(mic[xmod][ymod][zmod]==POROSITY){
                        effort=1;
                        mic[xmod][ymod][zmod]=DIFFCSH;
                        nmade+=1;
                        ngoing+=1;
                        /* Add this diffusing species to the linked list */
                        antnew=(struct ants *)malloc(sizeof(struct ants));
                        antnew->x=xmod;
                        antnew->y=ymod;
                        antnew->z=zmod;
                        antnew->id=DIFFCSH;
                        antnew->cycbirth=cyccnt;
                     /* Now connect this ant structure to end of linked list */
                        antnew->prevant=tailant;
                        tailant->nextant=antnew;
                        antnew->nextant=NULL;
                        tailant=antnew;
                }
        }
        return(effort);
}

/* routine to count number of pore pixels in a cube of size boxsize */
/* centered at (qx,qy,qz) */
/* Called by makeinert */
/* Calls no other routines */
int countbox(boxsize,qx,qy,qz)
        int boxsize,qx,qy,qz;
{
        int nfound,ix,iy,iz,qxlo,qxhi,qylo,qyhi,qzlo,qzhi;
        int hx,hy,hz,boxhalf;

        boxhalf=boxsize/2;
        nfound=0;
        qxlo=qx-boxhalf;
        qxhi=qx+boxhalf;
```

```
        qylo=qy-boxhalf;
        qyhi=qy+boxhalf;
        qzlo=qz-boxhalf;
        qzhi=qz+boxhalf;
        /* Count the number of requisite pixels in the 3-D cube box */
        /* using periodic boundaries */
        for(ix=qxlo;ix<=qxhi;ix++){
                hx=ix;
                if(hx<0){hx+=SYSIZE;}
                else if(hx>=SYSIZE){hx-=SYSIZE;}
        for(iy=qylo;iy<=qyhi;iy++){
                hy=iy;
                if(hy<0){hy+=SYSIZE;}
                else if(hy>=SYSIZE){hy-=SYSIZE;}
        for(iz=qzlo;iz<=qzhi;iz++){
                hz=iz;
                if(hz<0){hz+=SYSIZE;}
                else if(hz>=SYSIZE){hz-=SYSIZE;}
                /* Count if porosity, diffusing species, or empty porosity */
                if((mic [hx] [hy] [hz]<C3S)||(mic[hx] [hy] [hz]>ABSGYP)){
                        nfound+=1;
                }
        }
        }
        }
        return(nfound);
}

/* routine to create ndesire pixels of empty pore space to simulate */
/* self-desiccation */
/* Called by dissolve */
/* Calls countbox */
void makeinert(ndesire)
        long int ndesire;
{
        struct togo *headtogo,*tailtogo,*newtogo,*lasttogo,*onetogo;
        long int idesire;
        int px,py,pz,placed,cntpore,cntmax;

        /* First allocate the first element of the linked list */
        headtogo=(struct togo *)malloc(sizeof(struct togo));
        headtogo->x=headtogo->y=headtogo->z=(-1);
        headtogo->npore=0;
        headtogo->nexttogo=NULL;
        headtogo->prevtogo=NULL;
        tailtogo=headtogo;
        cntmax=0;
```

```
printf("In makeinert with %ld needed elements \n",ndesire);
fflush(stdout);
/* Add needed number of elements to the end of the list */
for(idesire=2;idesire<=ndesire;idesire++){
        newtogo=(struct togo *)malloc(sizeof(struct togo));
        newtogo->npore=0;
        newtogo->x=newtogo->y=newtogo->z=(-1);
        tailtogo->nexttogo=newtogo;
        newtogo->prevtogo=tailtogo;
        tailtogo=newtogo;
}

/* Now scan the microstructure and rank the sites */
for(px=0;px<SYSIZE;px++){
for(py=0;py<SYSIZE;py++){
for(pz=0;pz<SYSIZE;pz++){
        if(mic[px][py][pz]==POROSITY){
                cntpore=countbox(cubesize,px,py,pz);
                if(cntpore>cntmax){cntmax=cntpore;}
                /* Store this site value at appropriate place in */
                /* sorted linked list */
                if(cntpore>(tailtogo->npore)){
                        placed=0;
                        lasttogo=tailtogo;
                        while(placed==0){
                                newtogo=lasttogo->prevtogo;
                                if(newtogo==NULL){
                                        placed=2;
                                }
                                else{
                                        if(cntpore<=(newtogo->npore)){
                                                placed=1;
                                        }
                                }
                                if(placed==0){
                                        lasttogo=newtogo;
                                }
                        }
                     onetogo=(struct togo *)malloc(sizeof(struct togo));
                        onetogo->x=px;
                        onetogo->y=py;
                        onetogo->z=pz;
                        onetogo->npore=cntpore;
                        /* Insertion at the head of the list */
                        if(placed==2){
                                onetogo->prevtogo=NULL;
                                onetogo->nexttogo=headtogo;
```

```
                                        headtogo->prevtogo=onetogo;
                                        headtogo=onetogo;
                                }
                                if(placed==1){
                                        onetogo->nexttogo=lasttogo;
                                        onetogo->prevtogo=newtogo;
                                        lasttogo->prevtogo=onetogo;
                                        newtogo->nexttogo=onetogo;
                                }
                                /* Eliminate the last element */
                                lasttogo=tailtogo;
                                tailtogo=tailtogo->prevtogo;
                                tailtogo->nexttogo=NULL;
                                free(lasttogo);
                        }
                }
        }
        }
        }

        /* Now remove the sites */
        /* starting at the head of the list */
        /* and deallocate all of the used memory */
        for(idesire=1;idesire<=ndesire;idesire++){
                px=headtogo->x;
                py=headtogo->y;
                pz=headtogo->z;
                if(px!=(-1)){
                        mic [px] [py] [pz]=EMPTYP;
                        count[POROSITY]-=1;
                        count[EMPTYP]+=1;
                }
                lasttogo=headtogo;
                headtogo=headtogo->nexttogo;
                free(lasttogo);
         }
        /* If only small cubes of porosity were found, then adjust */
        /* cubesize to have a more efficient search in the future */
        if(cubesize>CUBEMIN){
                if((2*cntmax)<(cubesize*cubesize*cubesize)){
                       cubesize-=2;
                }
        }
}

/* routine to implement a cycle of dissolution */
/* Called by main program */
/* Calls passone, loccsh, and makeinert */
```

```
void dissolve(cycle)
        int cycle;
{
        int nc3aext,ncshext,nchext,nfh3ext,ngypext,nanhext,plok,edgef;
        int nsum5,nsum4,nsum3,nsum2,nhemext,nsum6,nc4aext;
        int xpmax,ypmax,phid,phnew,plnew,cread;
        int i,xloop,yloop,zloop,ngood,ix1,iy1,xc,yc,valid,xc1,yc1;
        int iz1,zc,zc1,cycnew;
        long int ctest;
        int placed,cshrand,ntrycsh,maxsulfate;
        long int ncshgo,nsurf,suminit;
        long int xext,nhgd,npchext;
        float pdis,plfh3,fchext,fc3aext,fanhext,mass_now,tot_mass,heatfill;
        float dfact,molesdh2o,h2oinit,alpha,heat2,heat3,heat4,fhemext,fc4aext;
        float pconvert,pc3scsh,pc2scsh,calcx,calcy,calcz,tdisfact;
        FILE *heatfile,*phfile;
        struct ants *antadd;

        /* Initialize variables */
        nmade=0;
        /* counter for number of csh diffusing species to */
        /* be located at random locations in microstructure */
        npchext=ncshgo=cshrand=0;
        heat_old=heat_new; /* new and old values for heat released */

        /* Initialize dissolution and phase counters */
        nsurf=0;
        for(i=0;i<=EMPTYP;i++){
                discount[i]=0;
                count[i]=0;
        }

        /* Pass one- highlight all edge points which are soluble */
        soluble[C3AH6]=0;
        heatsum=molesh2o=0.0;
        passone(0,EMPTYP,cycle,1);

        /* If first cycle, then determine all mixture proportions based */
        /* on user input and original microstructure */
        if(cycle==1){
                /* Mass of cement in system */
                cemmass=(specgrav[C3S]*(float)count[C3S]+specgrav[C2S]*
                (float)count[C2S]+specgrav[C3A]*(float)count[C3A]+
                specgrav[C4AF]*(float)count[C4AF]);
                CH_mass=specgrav[CH]*(float)count[CH];
                /* Total mass in system neglecting single aggregate */
                tot_mass=cemmass+(float)count[POROSITY]+specgrav[INERT]*
```

```
                (float)count[INERT]+specgrav[CACL2]*(float)count[CACL2]+
                specgrav[ASG]*(float)count[ASG]+
                specgrav[HEMIHYD]*(float)count[HEMIHYD]+
                specgrav[ANHYDRITE]*(float)count[ANHYDRITE]+
                specgrav[CAS2]*(float)count[CAS2]+
                specgrav[GYPSUM]*(float)count[GYPSUM]+
                specgrav[ANHYDRITE]*(float)count[ANHYDRITE]+
                specgrav[HEMIHYD]*(float)count[HEMIHYD]+
                specgrav[GYPSUMS]*(float)count[GYPSUMS]+
                specgrav[POZZ]*(float)count[POZZ]+CH_mass;
                /* water-to-cement ratio */
                if(cemmass!=0.0){
                        w_to_c=(float)count[POROSITY]/(cemmass+
                        specgrav[GYPSUM]*(float)count[GYPSUM]+
                        specgrav[ANHYDRITE]*(float)count[ANHYDRITE]+
                        specgrav[HEMIHYD]*(float)count[HEMIHYD]);
                }
                else{
                        w_to_c=0.0;
                }
/* Adjust masses for presence of aggregates in concrete */
                mass_water=(1.-mass_agg)*(float)count[POROSITY]/tot_mass;
                mass_CH=(1.-mass_agg)*CH_mass/tot_mass;
                /* pozzolan-to-cement ratio */
                if(cemmass!=0.0){
                        s_to_c=(float)(count[INERT]*specgrav[INERT]+
                        count[CACL2]*specgrav[CACL2]+count[ASG]*specgrav[ASG]+
                        count[CAS2]*specgrav[CAS2]+
                        count[POZZ]*specgrav[POZZ])/cemmass;
                }
                else{
                        s_to_c=0.0;
                }
                /* Conversion factor to kJ/kg for heat produced */
                if(cemmass!=0.0){
                       heatfill=(float)(count[INERT]+count[POZZ]+count[CACL2]+
                         count[ASG]+count[CAS2])/cemmass;
                }
                else{
                       heatfill=0.0;
                }
                if(w_to_c>0.01){
                       heat_cf=0.001*(0.3125+w_to_c+heatfill);
                }
                else{
                        /* Need volume per 1 gram of silica fume */
                        heat_cf=0.001*((1./specgrav[POZZ])+(float)
```

```
(count[POROSITY]+count[CH]+count[INERT])/(specgrav[POZZ]*(float)count[POZZ]));
              }
   mass_fill_pozz=(1.-mass_agg)*(float)(count[POZZ]*specgrav[POZZ])/tot_mass;
              mass_fill=(1.-mass_agg)*(float)(count[INERT]*specgrav[INERT]+
              count[ASG]*specgrav[ASG]+
              count[CAS2]*specgrav[CAS2]+count[POZZ]*specgrav[POZZ]+
              count[CACL2]*specgrav[CACL2])/tot_mass;
              printf("Calculated w/c is %.4f\n",w_to_c);
              printf("Calculated s/c is %.4f \n",s_to_c);
              printf("Calculated heat conversion factor is %f \n",heat_cf);
printf("Calculated mass fractions of water and filler are %.4f  and %.4f \n",
              mass_water,mass_fill);
      }

      molesdh2o=0.0;
      alpha=0.0;      /* degree of hydration */
      /* heat2 is heat of hydration based on heats of hydration of pure */
      /* compounds (c3s,c2s,c3a,c4af) from Neville and Brooks */
      /* page 14, Table 2.4 */
      /* Originally from Lerch and Bogue,  */
      /* See book (The Chemistry of Cement) by Lea for complete ref. */
      /* heat3 is heat of hydration based on heats of hydration of pure */
      /* compounds (c3s,c2s,c3a,c4af) from Cement Chemistry by Taylor */
      /* page 232, Table 7.5 */
      /* heat4 contains measured heat release for C4AF hydration from  */
      /* Fukuhara et al., Cem. and Conc. Res. article */
      heat2=heat3=heat4=0.0;
      mass_now=0.0;    /* total cement mass corrected for hydration */
      suminit=c3sinit+c2sinit+c3ainit+c4afinit;
      /* ctest is number of gypsum likely to form ettringite */
      /* 1 unit of C3A can react with 2.5 units of Gypsum */
      ctest=count[DIFFGYP];
      if((float)ctest>(2.5*(float)(count[DIFFC3A]+count[DIFFC4A]))){
               ctest=2.5*(float)(count[DIFFC3A]+count[DIFFC4A]);
      }
      for(i=0;i<=EMPTYP;i++){
              if((i!=0)&&(i<=ABSGYP)&&(i!=INERTAGG)&&(i!=CSH)){
                      heatsum+=(float)count[i]*heatf[i]/molarv[i];
              /* Tabulate moles of H2O consumed by reactions so far */
                      molesh2o+=(float)count[i]*waterc[i]/molarv[i];
              }
              /* assume that all C3A which can, does form ettringite */
              if(i==DIFFC3A){
         heatsum+=((float)count[DIFFC3A]-(float)ctest/2.5)*heatf[C3A]/molarv[C3A];
              }
              /* assume that all C4A which can, does form ettringite */
              if(i==DIFFC4A){
```

```
heatsum+=((float)count[DIFFC4A]-(float)ctest/2.5)*heatf[C4AF]/molarv[C4AF];
     }
     /* assume all gypsum which can, does form ettringite */
     /* rest will remain as gypsum */
     if(i==DIFFGYP){
      heatsum+=(float)(count[DIFFGYP]-ctest)*heatf[GYPSUM]/molarv[GYPSUM];
             /* 3.3 is the molar expansion from GYPSUM to ETTR */
          heatsum+=(float)ctest*3.30*heatf[ETTR]/molarv[ETTR];
          molesdh2o+=(float)ctest*3.30*waterc[ETTR]/molarv[ETTR];
     }
     else if(i==DIFFCH){
          heatsum+=(float)count[DIFFCH]*heatf[CH]/molarv[CH];
          molesdh2o+=(float)count[DIFFCH]*waterc[CH]/molarv[CH];
     }
     else if(i==DIFFFH3){
           heatsum+=(float)count[DIFFFH3]*heatf[FH3]/molarv[FH3];
           molesdh2o+=(float)count[DIFFFH3]*waterc[FH3]/molarv[FH3];
     }
     else if(i==DIFFCSH){
             /* use current CSH properties */
        heatsum+=(float)count[DIFFCSH]*heatf[CSH]/molarvcsh[cycle];
   molesdh2o+=(float)count[DIFFCSH]*watercsh[cycle]/molarvcsh[cycle];
     }
     else if(i==DIFFETTR){
         heatsum+=(float)count[DIFFETTR]*heatf[ETTR]/molarv[ETTR];
         molesdh2o+=(float)count[DIFFETTR]*waterc[ETTR]/molarv[ETTR];
     }
     else if(i==DIFFCACL2){
       heatsum+=(float)count[DIFFCACL2]*heatf[CACL2]/molarv[CACL2];
       molesdh2o+=(float)count[DIFFCACL2]*waterc[CACL2]/molarv[CACL2];
     }
     else if(i==DIFFAS){
          heatsum+=(float)count[DIFFAS]*heatf[ASG]/molarv[ASG];
          molesdh2o+=(float)count[DIFFAS]*waterc[ASG]/molarv[ASG];
     }
     else if(i==DIFFCAS2){
          heatsum+=(float)count[DIFFCAS2]*heatf[CAS2]/molarv[CAS2];
          molesdh2o+=(float)count[DIFFCAS2]*waterc[CAS2]/molarv[CAS2];
     }
 /* assume that all diffusing anhydrite leads to gypsum formation */
     else if(i==DIFFANH){
        heatsum+=(float)count[DIFFANH]*heatf[GYPSUMS]/molarv[GYPSUMS];
        /* 2 moles of water per mole of gypsum formed */
        molesdh2o+=(float)count[DIFFANH]*2.0/molarv[GYPSUMS];
     }
/* assume that all diffusing hemihydrate leads to gypsum formation */
     else if(i==DIFFHEM){
```

```
      heatsum+=(float)count[DIFFHEM]*heatf[GYPSUMS]/molarv[GYPSUMS];
            /* 1.5 moles of water per mole of gypsum formed */
            molesdh2o+=(float)count[DIFFHEM]*1.5/molarv[GYPSUMS];
    }
    else if(i==C3S){
            alpha+=(float)(c3sinit-count[C3S]);
            mass_now+=specgrav[C3S]*(float)count[C3S];
            heat2+=.502*(float)(c3sinit-count[C3S])*specgrav[C3S];
            heat3+=.517*(float)(c3sinit-count[C3S])*specgrav[C3S];
            heat4+=.517*(float)(c3sinit-count[C3S])*specgrav[C3S];
    }
    else if(i==C2S){
            alpha+=(float)(c2sinit-count[C2S]);
            mass_now+=specgrav[C2S]*(float)count[C2S];
            heat2+=.260*(float)(c2sinit-count[C2S])*specgrav[C2S];
            heat3+=.262*(float)(c2sinit-count[C2S])*specgrav[C2S];
            heat4+=.262*(float)(c2sinit-count[C2S])*specgrav[C2S];
    }
    else if(i==C3A){
            alpha+=(float)(c3ainit-count[C3A]);
            mass_now+=specgrav[C3A]*(float)count[C3A];
            heat2+=.867*(float)(c3ainit-count[C3A])*specgrav[C3A];
            heat3+=1.144*(float)(c3ainit-count[C3A])*specgrav[C3A];
            heat4+=1.144*(float)(c3ainit-count[C3A])*specgrav[C3A];
    }
    else if(i==C4AF){
            alpha+=(float)(c4afinit-count[C4AF]);
            mass_now+=specgrav[C4AF]*(float)count[C4AF];
           heat2+=.419*(float)(c4afinit-count[C4AF])*specgrav[C4AF];
           heat3+=.418*(float)(c4afinit-count[C4AF])*specgrav[C4AF];
           heat4+=.725*(float)(c4afinit-count[C4AF])*specgrav[C4AF];
    }
    /* 0.187 kJ/g anhydrite for anhydrite --> gypsum conversion */
    else if(i==ANHYDRITE){
         /*  alpha+=(float)(anhinit-count[ANHYDRITE]);
           mass_now+=specgrav[ANHYDRITE]*(float)count[ANHYDRITE]; */
  heat2+=.187*(float)(anhinit-count[ANHYDRITE])*specgrav[ANHYDRITE];
  heat3+=.187*(float)(anhinit-count[ANHYDRITE])*specgrav[ANHYDRITE];
  heat4+=.187*(float)(anhinit-count[ANHYDRITE])*specgrav[ANHYDRITE];
       /* 2 moles of water consumed per mole of anhydrite reacted */
  molesh2o+=(float)(anhinit-count[ANHYDRITE])*2.0/molarv[ANHYDRITE];
    }
    /* 0.132 kJ/g hemihydrate for hemihydrate-->gypsum conversion */
    else if(i==HEMIHYD){
      heat2+=.132*(float)(heminit-count[HEMIHYD])*specgrav[HEMIHYD];
      heat3+=.132*(float)(heminit-count[HEMIHYD])*specgrav[HEMIHYD];
      heat4+=.132*(float)(heminit-count[HEMIHYD])*specgrav[HEMIHYD];
```

```
                /* 1.5 moles of water consumed per mole of anhydrite reacted */
                 molesh2o+=(float)(heminit-count[HEMIHYD])*1.5/molarv[HEMIHYD];
               }
        }
        if(suminit!=0){
                alpha=alpha/(float)suminit;
        }
        else{
                alpha=0.0;
        }
        /* Current degree of hydration on a mass basis */
        if(cemmass!=0.0){
                alpha_cur=1.0-(mass_now/cemmass);
        }
        else{
                alpha_cur=0.0;
        }
        h2oinit=(float)porinit/molarv[POROSITY];
        /* Assume 780 J/g S for pozzolanic reaction */
        /* Each unit of silica fume consumes 1.35 units of CH, */
        /* so divide npr by 1.35 to get silca fume which has reacted */
        heat2+=0.78*((float)npr/1.35)*specgrav[POZZ];
        heat3+=0.78*((float)npr/1.35)*specgrav[POZZ];
        heat4+=0.78*((float)npr/1.35)*specgrav[POZZ];
        /* Assume 800 J/g AS for stratlingite formation (DeLarrard) */
        /* Each unit of AS consumes 1.3267 units of CH, */
        /* so divide nasr by 1.3267 to get ASG which has reacted */
        heat2+=0.80*((float)nasr/1.3267)*specgrav[ASG];
        heat3+=0.80*((float)nasr/1.3267)*specgrav[ASG];
        heat4+=0.80*((float)nasr/1.3267)*specgrav[ASG];

        /* Should be additional code here for heat release due to CAS2 to */
        /* stratlingite conversion, but data unavailable at this time */

        /* Adjust heat sum for water left in system */
        water_left=(long int)((h2oinit-molesh2o)*molarv[POROSITY]+0.5);
        countkeep=count[POROSITY];
        heatsum+=(h2oinit-molesh2o-molesdh2o)*heatf[POROSITY];
        heatfile=fopen("heat.out","a");
        if(cyccnt==0){
fprintf(heatfile,"Cycle  alpha_vol  alpha_mass  heat1 heat2  heat3  heat4\n");
        }
        fprintf(heatfile,"%d %f %f %f %f %f %f\n",
         cyccnt,alpha,alpha_cur,heatsum,heat2,heat3,heat4);
        heat_new=heat4;     /* use heat4 for all adiabatic calculations */
                            /* due to best agreement with calorimetry data */
        fclose(heatfile);
```

```
if(((water_left+water_off)<0)&&(sealed==1)){
        printf("All water consumed at cycle %d \n",cyccnt);
        fflush(stdout);
        exit();
}
/* Attempt to create empty porosity to account for self-desiccation */
if((sealed==1)&&((count[POROSITY]-water_left)>0)){
        poretodo=(count[POROSITY]-pore_off)-(water_left-water_off);
        if(poretodo>0){
                makeinert(poretodo);
                poregone+=poretodo;
        }
}
/* Output phase counts */
/* phfile for reactant and product phases */
phfile=fopen("phases.out","a");
fprintf(phfile,"%d ",cyccnt);
for(i=0;i<=EMPTYP;i++){
        if((i<DIFFCSH)||(i>DIFFCACL2)){
                   fprintf(phfile,"%ld ",count[i]);
        }
printf("%ld ",count[i]);
}
printf("\n");
fprintf(phfile,"%ld\n",water_left);
fclose(phfile);

cyccnt+=1;
/* Update current volume count for CH */
chold=chnew;
chnew=count[CH];

if(cycle==0){
        return;
}
/* See if ettringite is soluble yet */
/* Gypsum 90% consumed */
/* or system temperature exceeds 70 C */
if(((ncsbar+anhinit+heminit)!=0.0)||(temp_cur>=70.0)){
/* Account for all sulfate sources and forms */
if((soluble[ETTR]==0)&&((temp_cur>=70.0)||(count[AFM]!=0)||
((((float)count[GYPSUM]+1.42*(float)count[ANHYDRITE]+1.4*
(float)count[HEMIHYD]+(float)count[GYPSUMS])/((float)ncsbar+
1.42*(float)anhinit+1.4*(float)heminit+((float)netbar/3.30)))<0.10))){
        soluble[ETTR]=1;
        printf("Ettringite is soluble beginning at cycle %d \n",cycle);
        /* identify all new soluble ettringite */
```

```
        passone(ETTR,ETTR,2,0);
}
} /* end of soluble ettringite test */

/* Adjust gypsum solubility */
/* if too many diffusing gypsums already in solution */
if((count[DIFFGYP]+count[DIFFANH]+count[DIFFHEM])>DGYPMAX){
        disprob[GYPSUM]=disprob[GYPSUMS]=0.0;
        disprob[HEMIHYD]=disprob[ANHYDRITE]=0.0;
}
else{
        disprob[GYPSUM]=disbase[GYPSUM];
        disprob[GYPSUMS]=disbase[GYPSUMS];
        disprob[ANHYDRITE]=disbase[ANHYDRITE];
        disprob[HEMIHYD]=disbase[HEMIHYD];
}
/* Adjust ettringite solubility */
/* if too many ettringites already in solution */
if(count[DIFFETTR]>DETTRMAX){
        disprob[ETTR]=0.0;
}
else{
        disprob[ETTR]=disbase[ETTR];
}
/* Adjust CAS2 solubility */
/* if too many CAS2 already in solution */
if(count[DIFFCAS2]>DCAS2MAX){
        disprob[CAS2]=0.0;
}
else{
        disprob[CAS2]=disbase[CAS2];
}
/* Adjust CaCl2 solubility */
/* if too many CaCl2 already in solution */
if(count[DIFFCACL2]>DCACL2MAX){
        disprob[CACL2]=0.0;
}
else{
        disprob[CACL2]=disbase[CACL2];
}
/* Adjust solubility of CH */
/* based on amount of CH currently diffusing */
/* Note that CH is always soluble to allow some */
/* Ostwald ripening of the CH crystals */
if((float)count[DIFFCH]>=CHCRIT){
        disprob[CH]=disbase[CH]*CHCRIT/(float)count[DIFFCH];
}
```

```
        else{
                disprob[CH]=disbase[CH];
        }
        /* Adjust solubility of CH for temperature */
        /* Fit to data provided in Taylor, Cement Chemistry */
        /* Scale to a reference temperature of 25 C */
        /* and adjust based on availability of pozzolan */
        printf("CH dissolution probability goes from %f ",disprob[CH]);
        disprob[CH]*=((A0_CHSOL-A1_CHSOL*temp_cur)/(A0_CHSOL-A1_CHSOL*25.0));
        if(ppozz>0.0){
                disprob[CH]*=ppozz/PPOZZ;
        }
        printf("to %f \n",disprob[CH]);

        /* Adjust solubility of ASG and CAS2 phases */
        /* based on pH rise during hydration */
        /* To be added at a later date */

        /* Address solubility of C3AH6 */
        /* If lots of gypsum or reactive ettringite, allow C3AH6 to dissolve */
        /* to generate diffusing C3A species */
        if(((count[GYPSUM]+count[GYPSUMS])>(int)(((float)ncsbar+
        1.42*(float)anhinit+1.4*(float)heminit)*0.05))
        ||(count[ETTR]>500)){
                soluble[C3AH6]=1;
                passone(C3AH6,C3AH6,2,0);
                /* Base C3AH6 solubility on maximum sulfate in solution */
                /* from gypsum or ettringite available for dissolution */
                /* The more the sulfate, the higher this solubility should be */
                maxsulfate=count[DIFFGYP];
                if((maxsulfate<count[DIFFETTR])&&(soluble[ETTR]==1)){
                        maxsulfate=count[DIFFETTR];
                }
/* Adjust C3AH6 solubility based on potential gypsum which will dissolve */
                if(maxsulfate<(int)((float)gypready*disprob[GYPSUM]*
                (float)count[POROSITY]/1000000.)){
                        maxsulfate=(int)((float)gypready*disprob[GYPSUM]*
                        (float)count[POROSITY]/1000000.);
                }
                if(maxsulfate>0){
                      disprob[C3AH6]=disbase[C3AH6]*(float)maxsulfate/C3AH6CRIT;
                      if(disprob[C3AH6]>0.5){disprob[C3AH6]=0.5;}
                }
                else{
                      disprob[C3AH6]=disbase[C3AH6];
                }
        }
```

```
else{
        soluble[C3AH6]=0;
}

/* See if silicates are soluble yet */
if((soluble[C3S]==0)&&((cycle>1)||(count[ETTR]>0)||(count[AFM]>0)||
 (count[ETTRC4AF]>0))){
        soluble[C2S]=1;
        soluble[C3S]=1;
        /* identify all new soluble silicates */
        passone(C3S,C2S,2,0);
} /* end of soluble silicate test */
/* Adjust solubility of C3S and C2S with CSH concentration */
/* for simulation of induction period */
tdisfact=A0_CHSOL-temp_cur*A1_CHSOL;
dfact=tdisfact*(float)count[CSH]*(float)count[CSH]/CSHSCALE/CSHSCALE;
disprob[C3S]=DISMIN+dfact*disbase[C3S];
disprob[C2S]=DISMIN2+dfact*disbase[C2S];
if(disprob[C3S]>disbase[C3S]){disprob[C3S]=disbase[C3S];}
if(disprob[C2S]>disbase[C2S]){disprob[C2S]=disbase[C2S];}
/* Assume that aluminate dissolution also controlled by formation */
/* of impermeable layer proportional to CSH concentration */
disprob[C3A]=DISMIN+dfact*disbase[C3A];
disprob[C4AF]=DISMIN2+dfact*disbase[C4AF];
if(disprob[C3A]>disbase[C3A]){disprob[C3A]=disbase[C3A];}
if(disprob[C4AF]>disbase[C4AF]){disprob[C4AF]=disbase[C4AF];}
printf("Silicate and aluminate probabilities: %f %f %f %f\n",
   disprob[C3S],disprob[C2S],disprob[C3A],disprob[C4AF]);

/* Pass two- perform the dissolution of species */
nhgd=0;
/* Update molar volume ratios for CSH formation */
pc3scsh=molarvcsh[cyccnt]/molarv[C3S]-1.0;
pc2scsh=molarvcsh[cyccnt]/molarv[C2S]-1.0;
/* Once again, scan all pixels in microstructure */
for(xloop=0;xloop<SYSIZE;xloop++){
for(yloop=0;yloop<SYSIZE;yloop++){
for(zloop=0;zloop<SYSIZE;zloop++){
        if(mic[xloop][yloop][zloop]>OFFSET){
                phid=mic[xloop][yloop][zloop]-OFFSET;
                /* attempt a one-step random walk to dissolve */
                plnew=(int)((float)NEIGHBORS*ran1(seed));
                if((plnew<0)||(plnew>=NEIGHBORS)){ plnew=NEIGHBORS-1;}
                xc=xloop+xoff[plnew];
                yc=yloop+yoff[plnew];
                zc=zloop+zoff[plnew];
                if(xc<0){xc=(SYSIZEM1);}
```

```
 if(yc<0){yc=(SYSIZEM1);}
 if(xc>=SYSIZE){xc=0;}
 if(yc>=SYSIZE){yc=0;}
 if(zc<0){zc=(SYSIZEM1);}
 if(zc>=SYSIZE){zc=0;}

/* Generate probability for dissolution */
pdis=ran1(seed);

if((pdis<=disprob[phid])&&(mic[xc][yc][zc]==POROSITY)){
         discount[phid]+=1;
         cread=creates[phid];
         count[phid]-=1;
         mic[xloop][yloop][zloop]=POROSITY;
         if(phid==C3AH6){nhgd+=1;}
         /* Special dissolution for C4AF */
         if(phid==C4AF){
                plfh3=ran1(seed);
                if((plfh3<0.0)||(plfh3>1.0)){
                        plfh3=1.0;
                }
           /* For every C4AF that dissolves, 0.5453 */
           /* diffusing FH3 species should be created */
                if(plfh3<=0.5453){
                        cread=DIFFFH3;
                }
          }
          if(cread==POROSITY){
                count[POROSITY]+=1;
          }
          if(cread!=POROSITY){
                nmade+=1;
                ngoing+=1;
                phnew=cread;
                count[phnew]+=1;
                mic[xc][yc][zc]=phnew;
     antadd=(struct ants *)malloc(sizeof(struct ants));
                antadd->x=xc;
                antadd->y=yc;
                antadd->z=zc;
                antadd->id=phnew;
                antadd->cycbirth=cyccnt;
/* Now connect this ant structure to end of linked list */
                antadd->prevant=tailant;
                tailant->nextant=antadd;
                antadd->nextant=NULL;
                tailant=antadd;
```

```
                        }
                /* Extra CSH diffusing species based on current temperature */
                        if((phid==C3S)||(phid==C2S)){
                                plfh3=ran1(seed);
                          if(((phid==C2S)&&(plfh3<=pc2scsh))||(plfh3<=pc3scsh)){
                                        placed=loccsh(xc,yc,zc);
                                        if(placed!=0){
                                                count[DIFFCSH]+=1;
                                                count[POROSITY]-=1;
                                        }
                                        else{
                                                cshrand+=1;
                                        }
                                }
                        }

                        if((phid==C2S)&&(pc2scsh>1.0)){
                                plfh3=ran1(seed);
                                if(plfh3<=(pc2scsh-1.0)){
                                        placed=loccsh(xc,yc,zc);
                                        if(placed!=0){
                                                count[DIFFCSH]+=1;
                                                count[POROSITY]-=1;
                                        }
                                        else{
                                                cshrand+=1;
                                        }
                                }
                        }
                }
                else{
                        mic[xloop][yloop][zloop]-=OFFSET;
                }

            } /* end of if edge loop */
            /* Now check if CSH to pozzolanic CSH conversion is possible */
            /* Only if CH is less than 15% in volume */
            /* Only if CSH is in contact with at least one porosity */
            /* and user wishes to use this option */
if((count[POZZ]>=13000)&&(chnew<(0.15*SYSIZE*SYSIZE*SYSIZE))&&(csh2flag==1)){
                if(mic[xloop][yloop][zloop]==CSH){
                if((countbox(3,xloop,yloop,zloop))>=1){
                        pconvert=ran1(seed);
                        if(pconvert<PCSH2CSH){
                                count[CSH]-=1;
                                plfh3=ran1(seed);
                                /* molarvcsh units of C1.7SHx goes to */
```

```
                                        /* 101.81 units of C1.1SH3.9 */
                                        /* with 19.86 units of CH */
                                        /* so p=calcy */
                                        calcz=0.0;
                                        cycnew=cshage[xloop][yloop][zloop];
                                        calcy=molarv[POZZCSH]/molarvcsh[cycnew];
                                        if(calcy>1.0){
                                                calcz=calcy-1.0;
                                                calcy=1.0;
printf("Problem of not creating enough pozz. CSH during CSH conversion \n");
                                printf("Current temperature is %f C\n",temp_cur);
                                        }

                                        if(plfh3<=calcy){
                                               mic[xloop][yloop][zloop]=POZZCSH;
                                               count[POZZCSH]+=1;
                                        }
                                        else{
                                                mic[xloop][yloop][zloop]=DIFFCH;
                                                nmade+=1;
                                                ncshgo+=1;
                                                ngoing+=1;
                                                count[DIFFCH]+=1;
                                antadd=(struct ants *)malloc(sizeof(struct ants));
                                                antadd->x=xloop;
                                                antadd->y=yloop;
                                                antadd->z=zloop;
                                                antadd->id=DIFFCH;
                                                antadd->cycbirth=cyccnt;
                        /* Now connect this ant structure to end of linked list */
                                                antadd->prevant=tailant;
                                                tailant->nextant=antadd;
                                                antadd->nextant=NULL;
                                                tailant=antadd;
                                         }

                                        plfh3=ran1(seed);
                                      calcx=(19.86/molarvcsh[cycnew])-(1.-calcy);
                                      /* Ex. 0.12658=(19.86/108.)-(1.-0.94269) */
                                        if(plfh3<calcx){
                                                npchext+=1;
                                        }
                                   }
                             }
                             }
                    }
    } /* end of zloop */
```

```
} /* end of yloop */
} /* end of xloop */

if(ncshgo!=0){printf("CSH dissolved is %ld \n",ncshgo);}

if(npchext>0){printf("npchext is %ld at cycle %d \n",npchext,cycle);}
/* Now add in the extra diffusing species for dissolution */
/* Expansion factors from Young and Hansen and */
/* Mindess and Young (Concrete) */
ncshext=cshrand;
if(cshrand!=0){
          printf("cshrand is %d \n",cshrand);
}
/* CH, Gypsum, and diffusing C3A are added at totally random */
/* locations as opposed to at the dissolution site */
fchext=0.61*(float)discount[C3S]+0.191*(float)discount[C2S]+
0.2584*(float)discount[C4AF];
nchext=fchext;
if(fchext>(float)nchext){
        pdis=ran1(seed);
        if((fchext-(float)nchext)>pdis){
                nchext+=1;
        }
}
nchext+=npchext;
fc3aext=discount[C3A]+0.5917*(float)discount[C3AH6];
nc3aext=fc3aext;
if(fc3aext>(float)nc3aext){
        pdis=ran1(seed);
        if((fc3aext-(float)nc3aext)>pdis){
                nc3aext+=1;
        }
}
fc4aext=0.696*(float)discount[C4AF];
nc4aext=fc4aext;
if(fc4aext>(float)nc4aext){
        pdis=ran1(seed);
        if((fc4aext-(float)nc4aext)>pdis){
             nc4aext+=1;
        }
}
/* both forms of GYPSUM form same DIFFGYP species */
ngypext=discount[GYPSUM]+discount[GYPSUMS];
/* Convert to diffusing anhydrite at volume necessary for final */
/* gypsum formation  (1 anhydrite --> 1.423 gypsum) */
/* Since hemihydrate can now react with C3A, etc., can't */
/* do expansion here any longer  7/99 */
```

```
/* fanhext=1.423*(float)discount[ANHYDRITE]; */
        fanhext=(float)discount[ANHYDRITE];
        nanhext=fanhext;
        if(fanhext>(float)nanhext){
                pdis=ran1(seed);
                if((fanhext-(float)nanhext)>pdis){
                        nanhext+=1;
                }
        }
        /* Convert to diffusing hemiydrate at volume necessary for final */
        /* gypsum formation  (1 hemihydrate --> 1.4 gypsum) */
        /* Since hemihydrate can now react with C3A, etc., can't */
        /* do expansion here any longer  7/99 */
/* fhemext=1.3955*(float)discount[HEMIHYD];  */
        fhemext=(float)discount[HEMIHYD];

        nhemext=fhemext;
        if(fhemext>(float)nhemext){
                pdis=ran1(seed);
                if((fhemext-(float)nhemext)>pdis){
                        nhemext+=1;
                }
        }
        count[DIFFGYP]+=ngypext;
        count[DIFFANH]+=nanhext;
        count[DIFFHEM]+=nhemext;
        count[DIFFCH]+=nchext;
        count[DIFFCSH]+=ncshext;
        count[DIFFC3A]+=nc3aext;
        count[DIFFC4A]+=nc4aext;

        nsum2=nchext+ncshext;
        nsum3=nsum2+nc3aext;
        nsum4=nsum3+nc4aext;
        nsum5=nsum4+ngypext;
        nsum6=nsum5+nhemext;
        fflush(stdout);
        for(xext=1;xext<=(nsum6+nanhext);xext++){
        plok=0;
        do{
                xc=(int)((float)SYSIZE*ran1(seed));
                yc=(int)((float)SYSIZE*ran1(seed));
                zc=(int)((float)SYSIZE*ran1(seed));
                if(xc>=SYSIZE){xc=0;}
                if(yc>=SYSIZE){yc=0;}
                if(zc>=SYSIZE){zc=0;}
```

```
                if(mic[xc][yc][zc]==POROSITY){
                        plok=1;
                        phid=DIFFCH;
                        count[POROSITY]-=1;
                        if(xext>nsum6){phid=DIFFANH;}
                        else if(xext>nsum5){phid=DIFFHEM;}
                        else if(xext>nsum4){phid=DIFFGYP;}
                        else if(xext>nsum3){phid=DIFFC4A;}
                        else if(xext>nsum2){phid=DIFFC3A;}
                        else if(xext>nchext){phid=DIFFCSH;}
                        mic[xc][yc][zc]=phid;
                        nmade+=1;
                        ngoing+=1;
                        antadd=(struct ants *)malloc(sizeof(struct ants));
                        antadd->x=xc;
                        antadd->y=yc;
                        antadd->z=zc;
                        antadd->id=phid;
                        antadd->cycbirth=cyccnt;
                     /* Now connect this ant structure to end of linked list */
                        antadd->prevant=tailant;
                        tailant->nextant=antadd;
                        antadd->nextant=NULL;
                        tailant=antadd;
                }
        } while (plok==0);

        } /* end of xext for extra species generation */

        printf("Dissolved- %ld %ld %ld %ld %ld %ld %ld %ld %ld %ld %ld\n",count[DIFFCSH]
        count[DIFFCH],count[DIFFGYP],count[DIFFC3A],count[DIFFFH3],
        count[DIFFETTR],count[DIFFAS],count[DIFFANH],count[DIFFHEM],
        count[DIFFCAS2],count[DIFFCACL2]);
        printf("C3AH6 dissolved- %ld with prob. of %f \n",nhgd,disprob[C3AH6]);
        fflush(stdout);
}

/* routine to add nneed one pixel elements of phase randid at random */
/* locations in microstructure */
/* Called by main program */
/* Calls no other routines */
void addrand(randid,nneed)
        int randid;
        long int nneed;
{
        int ix,iy,iz;
        long int ic;
        int success;
```

```
        /* Add number of requested phase pixels at random pore locations */
        for(ic=1;ic<=nneed;ic++){
                success=0;
                while(success==0){
                        ix=(int)((float)SYSIZE*ran1(seed));
                        iy=(int)((float)SYSIZE*ran1(seed));
                        iz=(int)((float)SYSIZE*ran1(seed));
                        if(ix==SYSIZE){ix=0;}
                        if(iy==SYSIZE){iy=0;}
                        if(iz==SYSIZE){iz=0;}
                        if(mic[ix][iy][iz]==POROSITY){
                                mic[ix][iy][iz]=randid;
                                micorig[ix][iy][iz]=randid;
                                success=1;
                        }
                }
       }
}

/* Calls dissolve and addrand */
main()
{
        int ntimes,valin;
        int cycflag,ix,iy,iz,phtodo;
        int iseed,phydfreq,oflag,adiaflag;
        long int nadd;
        int xpl,xph,ypl,yph,fidc3s,fidc2s,fidc3a,fidc4af,fidgyp,fidagg,ffac3a;
        int fidhem,fidanh;
        float pnucch,pscalech,pnuchg,pscalehg,pnucfh3,pscalefh3;
        float pnucgyp,pscalegyp;
        float thtimelo,thtimehi,thtemplo,thtemphi;
        float mass_cement,mass_cem_now,mass_cur,kpozz;
        FILE *infile,*outfile,*adiafile,*thfile;
        char filei[80],fileo[80];

        ngoing=0;
        porefl1=porefl2=porefl3=1;
        pore_off=water_off=0;
        cycflag=0;
        heat_old=heat_new=0.0;
        chold=chnew=0;     /* Current and previous cycle CH counts */
        time_cur=0.0;      /* Elapsed time according to maturity principles */
        cubesize=CUBEMAX;
        ppozz=PPOZZ;
        poregone=poretodo=0;
        /* Get random number seed */
        printf("Enter random number seed \n");
```

```
    scanf("%d",&iseed);
    printf("%d\n",iseed);
    seed=(&iseed);

    printf("Dissolution bias is set at %f \n",DISBIAS);
printf("Do you wish to output final microstructure to file (0-No 1-Yes)\n");
    scanf("%d",&oflag);
    printf("%d\n",oflag);
    if(oflag==1){
            printf("Enter name of file to create \n");
            scanf("%s",fileo);
            printf("%s\n",fileo);
    }
    /* Open file and read in original cement particle microstructure */
    printf("Enter name of file to read initial microstructure from \n");
    scanf("%s",filei);
    printf("%s\n",filei);
    /* Get phase assignments for original microstructure */
    /* to transform to needed ID values */
    printf("Enter IDs in file for C3S, C2S, C3A, C4AF, Gypsum, Anhydrite,");
    printf(" Hemihydrate, Aggregate\n");
    scanf("%d %d %d %d %d %d %d %d",&fidc3s,&fidc2s,&fidc3a,&fidc4af,
    &fidgyp,&fidanh,&fidhem,&fidagg);
    printf("%d %d %d %d %d %d %d %d\n",fidc3s,fidc2s,fidc3a,fidc4af,
    fidgyp,fidanh,fidhem,fidagg);
    printf("Enter ID in file for C3A in fly ash  (default=35)\n");
    scanf("%d",&ffac3a);
    printf("%d\n",ffac3a);
    fflush(stdout);

    infile=fopen(filei,"r");

    for(ix=0;ix<SYSIZE;ix++){
    for(iy=0;iy<SYSIZE;iy++){
    for(iz=0;iz<SYSIZE;iz++){
            cshage[ix][iy][iz]=0;
            fscanf(infile,"%d",&valin);
            mic[ix][iy][iz]=valin;
            if(valin==fidc3s){
                    mic[ix][iy][iz]=C3S;
            }
            else if(valin==fidc2s){
                    mic[ix][iy][iz]=C2S;
            }
            else if((valin==fidc3a)||(valin==ffac3a)){
                    mic[ix][iy][iz]=C3A;
            }
```

```
        else if(valin==fidc4af){
                mic[ix][iy][iz]=C4AF;
        }
        else if(valin==fidgyp){
                mic[ix][iy][iz]=GYPSUM;
        }
        else if(valin==fidanh){
                mic[ix][iy][iz]=ANHYDRITE;
        }
        else if(valin==fidhem){
                mic[ix][iy][iz]=HEMIHYD;
        }
        else if(valin==fidagg){
                mic[ix][iy][iz]=INERTAGG;
        }
        micorig[ix][iy][iz]=mic[ix][iy][iz];
}
}
}
fclose(infile);
fflush(stdout);

/* Now read in particle IDs from file */
printf("Enter name of file to read particle IDs from \n");
scanf("%s",filei);
printf("%s\n",filei);
infile=fopen(filei,"r");

for(ix=0;ix<SYSIZE;ix++){
for(iy=0;iy<SYSIZE;iy++){
for(iz=0;iz<SYSIZE;iz++){
        fscanf(infile,"%d",&valin);
        micpart[ix][iy][iz]=valin;
}
}
}

fclose(infile);
fflush(stdout);

/* Initialize counters, etc. */
npr=nasr=0;
nfill=0;
ncsbar=0;
netbar=0;
porinit=0;
cyccnt=0;
```

```
  setflag=0;
  c3sinit=c2sinit=c3ainit=c4afinit=anhinit=heminit=0;

  /* Initialize structure for ants */
  headant=(struct ants *)malloc(sizeof(struct ants));
  headant->prevant=NULL;
  headant->nextant=NULL;
  headant->x=0;
  headant->y=0;
  headant->z=0;
  headant->id=100;       /* special ID indicating first ant in list */
  headant->cycbirth=0;
  tailant=headant;

/* Allow user to iteratively add one pixel particles of various phases */
/* Typical application would be for addition of silica fume */
  printf("Enter number of one pixel particles to add (0 to quit) \n");
  scanf("%ld",&nadd);
  printf("%ld\n",nadd);
  while(nadd>0){
          phtodo=25;
          while((phtodo<0)||(phtodo>GYPSUMS)){
                  printf("Enter phase to add \n");
                  printf(" C3S 1 \n");
                  printf(" C2S 2 \n");
                  printf(" C3A 3 \n");
                  printf(" C4AF 4 \n");
                  printf(" GYPSUM 5 \n");
                  printf(" HEMIHYD 6 \n");
                  printf(" ANHYDRITE 7 \n");
                  printf(" POZZ 8 \n");
                  printf(" INERT 9 \n");
                  printf(" ASG 10 \n");
                  printf(" CAS2 11 \n");
                  printf(" CH 12 \n");
                  printf(" CSH 13 \n");
                  printf(" C3AH6 14 \n");
                  printf(" Ettringite 15 \n");
                  printf(" Stable Ettringite from C4AF 16 \n");
                  printf(" AFM 17 \n");
                  printf(" FH3 18 \n");
                  printf(" POZZCSH 19 \n");
                  printf(" CACL2 20 \n");
                  printf(" Friedels salt 21 \n");
                  printf(" Stratlingite 22 \n");
                  scanf("%d",&phtodo);
                  printf("%d \n",phtodo);
```

```
                }
                addrand(phtodo,nadd);
printf("Enter number of one pixel particles to add (0 to quit) \n");
                scanf("%ld",&nadd);
                printf("%ld\n",nadd);
        }
        fflush(stdout);

        init();
        printf("After init routine \n");
        printf("Enter number of cycles to execute \n");
        scanf("%d",&ncyc);
        printf("%d \n",ncyc);
  printf("Do you wish hydration under 0) saturated or 1) sealed conditions \n");
        scanf("%d",&sealed);
        printf("%d \n",sealed);
        printf("Enter max. # of diffusion steps per cycle (500) \n");
        scanf("%d",&ntimes);
        printf("%d \n",ntimes);
        printf("Enter nuc. prob. and scale factor for CH nucleation \n");
        scanf("%f %f",&pnucch,&pscalech);
        printf("%f %f \n",pnucch,pscalech);
        printf("Enter nuc. prob. and scale factor for gypsum nucleation \n");
        scanf("%f %f",&pnucgyp,&pscalegyp);
        printf("%f %f \n",pnucgyp,pscalegyp);
        printf("Enter nuc. prob. and scale factor for C3AH6 nucleation \n");
        scanf("%f %f",&pnuchg,&pscalehg);
        printf("%f %f \n",pnuchg,pscalehg);
        printf("Enter nuc. prob. and scale factor for FH3 nucleation \n");
        scanf("%f %f",&pnucfh3,&pscalefh3);
        printf("%f %f \n",pnucfh3,pscalefh3);
        printf("Enter cycle frequency for checking pore space percolation \n");
        scanf("%d",&burnfreq);
        printf("%d\n",burnfreq);
  printf("Enter cycle frequency for checking percolation of solids (set) \n");
        scanf("%d",&setfreq);
        printf("%d\n",setfreq);
  printf("Enter cycle frequency for checking hydration of particles \n");
        scanf("%d",&phydfreq);
        printf("%d\n",phydfreq);
        /* Parameters for adiabatic temperature rise calculation */
        printf("Enter the induction time in hours \n");
        scanf("%f",&ind_time);
        printf("%f \n",ind_time);
        time_cur+=ind_time;
        printf("Enter the initial temperature in degrees Celsius \n");
        scanf("%f",&temp_0);
```

```
      printf("%f \n",temp_0);
      temp_cur=temp_0;
      printf("Enter apparent activation energy for hydration in kJ/mole \n");
      scanf("%f",&E_act);
      printf("%f \n",E_act);
printf("Enter apparent activation energy for pozz. reactions in kJ/mole \n");
      scanf("%f",&E_act_pozz);
      printf("%f \n",E_act_pozz);
      printf("Enter kinetic factor to convert cycles to time for 25 C \n");
      scanf("%f",&beta);
      printf("%f \n",beta);
      printf("Enter mass fraction of aggregate in concrete \n");
      scanf("%f",&mass_agg);
      printf("%f \n",mass_agg);
      printf("Hydration under 0) isothermal, 1) adiabatic");
      printf(" or 2) programmed temperature history conditions \n");
      scanf("%d",&adiaflag);
      printf("%d \n",adiaflag);
      if(adiaflag==2){
              thfile=fopen("temphist.dat","r");
         fscanf(thfile,"%f %f %f %f",&thtimelo,&thtimehi,&thtemplo,&thtemphi);
              printf("%f %f %f %f\n",thtimelo,thtimehi,thtemplo,thtemphi);
      }
      printf("CSH to pozzolanic CSH 0) prohibited or 1) allowed \n");
      scanf("%d",&csh2flag);
      printf("%d \n",csh2flag);
      fflush(stdout);
      krate=exp(-(1000.*E_act/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
      /* Determine pozzolanic reaction rate constant */
      kpozz=exp(-(1000.*E_act_pozz/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
     /* Update probability of pozzolanic reaction */
     /* based on ratio of pozzolanic reaction rate to hydration rate */
     ppozz=PPOZZ*kpozz/krate;
     disprob[ASG]=disbase[ASG]*kpozz/krate;
     disprob[CAS2]=disbase[CAS2]*kpozz/krate;

      adiafile=fopen("adiabatic.out","w");
      fprintf(adiafile,"Time(h) Temperature  Alpha  Krate");
      fprintf(adiafile,"     Cp_now  Mass_cem kpozz/khyd\n");
      /* Set initial properties of CSH */
      molarvcsh[0]=molarv[CSH];
      watercsh[0]=waterc[CSH];
      for(icyc=1;icyc<=ncyc;icyc++){
              molarvcsh[icyc]=molarv[CSH]-8.0*((temp_cur-20.)/(80.-20.));
              watercsh[icyc]=waterc[CSH]-1.3*((temp_cur-20.)/(80.-20.));
              if(icyc==ncyc){cycflag=1;}
              dissolve(icyc);
```

```
printf("Number dissolved this pass- %ld total diffusing- %ld \n",nmade,ngoing);
                fflush(stdout);
                if(icyc==1){
                     printf("ncsbar is %ld   netbar is %ld \n",ncsbar,netbar);
                }
      hydrate(cycflag,ntimes,pnucch,pscalech,pnuchg,pscalehg,pnucfh3,
              pscalefh3,pnucgyp,pscalegyp);
                temp_0=temp_cur;
                /* Handle adiabatic case first */
                /* Cement + aggregate +water + filler=1;  that's all there is */
                mass_cement=1.-mass_agg-mass_fill-mass_water-mass_CH;
                mass_cem_now=mass_cement;
                if(adiaflag==1){
                        /* determine heat capacity of current mixture, */
                        /* accounting for imbibed water if necessary */
                        if(sealed==1){
                                Cp_now=mass_agg*Cp_agg;
                                Cp_now+=Cp_pozz*mass_fill;
                                Cp_now+=Cp_cement*mass_cement;
                                Cp_now+=Cp_CH*mass_CH;
          Cp_now+=(Cp_h2o*mass_water-alpha_cur*WN*mass_cement*(Cp_h2o-Cp_bh2o));
                                mass_cem_now=mass_cement;
                        }
                /* Else need to account for extra capillary water drawn in */
        /* Basis is WCHSH(0.06) g H2O per gram cement for chemical shrinkage */
                /* Need to adjust mass basis to account for extra imbibed H2O */
                        else{
                                mass_cur=1.+WCHSH*mass_cement*alpha_cur;
                                Cp_now=mass_agg*Cp_agg/mass_cur;
                                Cp_now+=Cp_pozz*mass_fill/mass_cur;
                                Cp_now+=Cp_cement*mass_cement/mass_cur;
                                Cp_now+=Cp_CH*mass_CH/mass_cur;
        Cp_now+=(Cp_h2o*mass_water-alpha_cur*WN*mass_cement*(Cp_h2o-Cp_bh2o));
        Cp_now+=(WCHSH*Cp_h2o*alpha_cur*mass_cement);
                                mass_cem_now=mass_cement/mass_cur;
                        }
                /* Determine rate constant based on Arrhenius expression */
                /* Recall that temp_cur is in degrees Celsius */
         krate=exp(-(1000.*E_act/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
                /* Determine pozzolanic reaction rate constant */
      kpozz=exp(-(1000.*E_act_pozz/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
        /* Update probability of pozzolanic reaction */
        /* based on ratio of pozzolanic reaction rate to hydration rate */
                        ppozz=PPOZZ*kpozz/krate;
                        disprob[ASG]=disbase[ASG]*kpozz/krate;
                        disprob[CAS2]=disbase[CAS2]*kpozz/krate;
```

```
              /* Update temperature based on heat generated and current Cp */
                     if(mass_cem_now>0.01){
              temp_cur=temp_0+mass_cem_now*heat_cf*(heat_new-heat_old)/Cp_now;
                     }
                     else{
            temp_cur=temp_0+mass_fill_pozz*heat_cf*(heat_new-heat_old)/Cp_now;

                     }
              }
              else if(adiaflag==2){
                      /* Update system temperature based on current time */
                      /* and requested temperature history */
                      while((time_cur>thtimehi)&&(!feof(thfile))){
         fscanf(thfile,"%f %f %f %f",&thtimelo,&thtimehi,&thtemplo,&thtemphi);
                              printf("New temperature history values : \n");
                   printf("%f %f %f %f\n",thtimelo,thtimehi,thtemplo,thtemphi);
                      }
temp_cur=thtemplo+(thtemphi-thtemplo)*(time_cur-thtimelo)/(thtimehi-thtimelo);
          krate=exp(-(1000.*E_act/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
      kpozz=exp(-(1000.*E_act_pozz/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
                      ppozz=PPOZZ*kpozz/krate;
                      disprob[ASG]=disbase[ASG]*kpozz/krate;
                      disprob[CAS2]=disbase[CAS2]*kpozz/krate;
              }
              /* Update time based on simple numerical integration */
              /* simulating maturity approach */
              /* with parabolic kinetics (Knudsen model) */
              if(cyccnt>1){time_cur+=(2.*(float)(cyccnt-1)-1.0)*beta/krate;}
              fprintf(adiafile,"%f %f %f %f %f %f %f\n",time_cur,temp_cur,
               alpha_cur,krate,Cp_now,mass_cem_now,kpozz/krate);
              fflush(adiafile);
        /* Check percolation of pore space */
        /* Note that first variable passed corresponds to phase to check */
        /* Could easily add calls to check for percolation of CH, CSH, etc. */
        if(((icyc%burnfreq)==0)&&((porefl1+porefl2+porefl3)!=0)){
               porefl1=burn3d(0,1,0,0);
               porefl2=burn3d(0,0,1,0);
               porefl3=burn3d(0,0,0,1);
               /* Switch to self-desiccating conditions when porosity */
               /* disconnects */
               if(((porefl1+porefl2+porefl3)==0)&&(sealed==0)){
                       water_off=water_left;
                       pore_off=countkeep;
                       sealed=1;
                  printf("Switching to self-desiccating at cycle %d \n",cyccnt);
                       fflush(stdout);
               }
```

```
        }
        /* Check percolation of solids (set point) */
        if((icyc%setfreq)==0){
                setflag=burnset(1,0,0);
                setflag+=burnset(0,1,0);
                setflag+=burnset(0,0,1);
        }


        /* Check hydration of particles */
        if((icyc%phydfreq)==0){
               parthyd();
        }

        }
        fclose(adiafile);
        dissolve(0);
        /* Check percolation of pore space */
        /* Note that first variable passed corresponds to phase to check */
        /* Could easily add calls to check for percolation of CH, CSH, etc. */
        if((burnfreq!=0)&&(burnfreq<=ncyc)&&((porefl1+porefl2+porefl3)!=0)){
               porefl1=burn3d(0,1,0,0);
               porefl2=burn3d(0,0,1,0);
               porefl3=burn3d(0,0,0,1);
        }
        /* Check percolation of solids (set point) */
        if((setfreq!=0)&&(setfreq<=ncyc)){
                setflag=burnset(1,0,0);
                setflag+=burnset(0,1,0);
                setflag+=burnset(0,0,1);
        }


        /* Output final microstructure if desired */
        if(oflag==1){
                outfile=fopen(fileo,"w");

                for(ix=0;ix<SYSIZE;ix++){
                for(iy=0;iy<SYSIZE;iy++){
                for(iz=0;iz<SYSIZE;iz++){
                        fprintf(outfile,"%d\n",(int)mic[ix][iy][iz]);
                }
                }
                }
        fclose(outfile);
        }
}
```