User Guide for Program

combineall

Jeffrey W. Bullard
National Institute of Standards and Technology, Gaithersburg, MD
December 20, 2002

1 Introduction

1.1 Description

The program `combineall` segments a scanning electron micrograph of a flat, polished cement powder into its constituent phases. It accomplishes this by

1. reading a sequence of user-supplied X-ray element maps for the micrograph
2. applying user-defined threshold values for the X-ray intensity of each element that is relevant, to determine the presence or absence of that element at a given location
3. traversing a decision tree to assign a phase at each location, based on the combination of elements that are present at that location

A map for each of the following elements, formatted as a Sun[1] raster file (extension *.s)[2] are required by `combineall`:

- Ca, Si, Al, Fe, S, K, and Mg

By examining the combinations and relative intensities in which these elements occur, `combineall` is able to recognize the following cement phases/compounds[3]:

- C_3S, C_2S, C_3A, C_4AF, $CaSO_4 \cdot xH_2O$
- CaO (indistinguishable from $CaCO_3$)
- generic sulfate salts of potassium (K_2SO_4, aphthitalite, calcium langbeinite, syngenite, etc.)
- SiO_2, (generic) slag, periclase ($Mg_xCa_{1-x}O$)
- kaolin (AS_2H_2), CAS_2

[1] Commercial equipment, instruments, and software mentioned in this document are identified to foster understanding. Such identification does no imply recommendation or endorsement by the National Institute of Standards and Technology (NIST), nor does it imply that the materials or software identified are necessarily the best available for the purpose.

[2] The exact format of the file, for now, must conform to that generated by Princeton Gamma-Tech software. For other formats, please consult the VCCTL Consortium Manager.

[3] The usual convention is adopted for abbreviating cement compounds, e.g. C = CaO, etc.

1.2 Installation

To compile `combineall`, type the command `make combineall` at a command-line prompt. Please report any compile-time errors to the VCCTL Consortium Manager (*bullard@nist.gov*).

2 Using `combineall`

2.1 Command-line Syntax

To invoke `combineall`, simply type `combineall` at the command line. The program is best run when the user is in the directory that contains the X-ray element map files. If the message

```
bash: combineall: command not found
```

is displayed upon invoking combineall, it means that the resident directory for the `combineall` executable is not in your path. To resolve this either enter the fully-resolved filename or add the resident directory to the PATH environment variable.

2.2 Requested Input

Upon invoking `combineall` successfully, the user will be asked for several pieces of information. In the order they are requested, these are:

1. `Input x and y sizes`. The user must know the image size of each of the X-ray element maps. Respond by typing two integers separated by a single space. For example, if the element maps have a horizontal (x) size of 512 pixels and a vertical (y) size of 400 pixels, the user would enter `512 400` (followed by the ⟨Enter⟩ key). **NOTE:** Every element map for the image must have the *same* dimensions.

2. `Enter file root for processing`. Each element map file must be named according to the following convention:

   fileroot_element.ras

   where *element* is the designation of the element. This designation is the same as that used in the Periodic Table of the Elements, except that it must be **all lowercase letters**. The *fileroot* is everything that comes before the element designation, and that is what the user must enter. For example, if the element map file for silicon is named `mycement_si.ras`, then the user would enter `mycement_` (followed by the ⟨Enter⟩ key). **NOTE:** Every element map name must have the **same** file root.

3. `Enter number of pixels to skip at start`. Here is where a knowledge of the exact format used to store the image is needed. Every raster file has some amount of "header" material at the beginning of the file,

and some raster files also have buffered pixels at the beginning that are not part of the image. For example, the Princeton Gamma-Tech software used at NIST for image acquisition places 800 irrelevant pixels at the beginning of the image. In this instance, the user would enter the integer `800` (followed by the ⟨Enter⟩ key).

4. `Enter threshold value for element ca (0-255)`. The raster files store a single integer in the interval [0–255] at each pixel, namely the intensity of the X-ray signal collected at that pixel. The program needs to know how to distinguish a real signal from background noise in the image. The value entered here specifies that the computer is to ignore any signal for element Ca that is less than or equal to the value entered. A typical value for Ca is 100.

   The program proceeds to request a threshold value for every other element that it uses.

5. `Enter threshold value for free lime`. It is possible that, as the program steps through the decision tree (detailed later in this document), the combination and intensities of elements at a particular point do not meet the criteria for any of the recognized phases. If the level of Ca is above the threshold, then the program can choose to specify the point as CaO, but only if the Ca signal exceeds the threshold that is specified here for free lime. Typically, this value is set to 1.5× the value entered for Ca.

6. `Enter threshold value for silica`. The same considerations apply here as for free lime. If a given pixel cannot be assigned to any other phase, but it does have a real Si signal, then the program will assign it to be SiO_2 if the Si signal exceeds the threshold entered here.

7. `Enter critical ratio for C3S vs. C2S`. If a pixel has a real signal for both Ca and Si, but for no other elements, then it may be either C_3S or C_2S. The real-number value entered here for the ratio of the Ca/Si signal intensity determines which phase is assigned. A typical value is 1.10.

8. `Input any integer to continue`. The program is ready to begin its first pass through the image. After you enter an integer here, output about the computed phase fractions is printed to the screen. This output should be ignored.

9. `Input any integer to continue`. The program is ready to begin another pass through the image. After you enter an integer here, output about the computed phase fractions is once again printed to the screen. This output is meaningful and may be recorded.

10. `Enter binary filename to open for output`. The name you enter here will contain information that is required for other programs. Any name can be used here, but it is good practice to give the file the same file root

as for the element maps, and to use an extension, such as `bin` or `img` that will indicate the subsequent purpose of the file.

11. `Enter filename to open for phase ID image`. This file will contain the integer phase ID associated with every pixel in the image. It will be formatted as a Sun raster file, and so it should have the extension `.ras`.

12. `Enter filename to open for COLOR image`. This file will be a color-coded image of the microstructure. The colorscheme is identical to that used for other VCCTL images, and is suitable for immediate inclusion in the database if the user judges that the thresholding of each element is correct (see **Proper Thresholding** below).

13. `Input any integer to continue`. Entering any integer will exit the program and return control to the shell.

3 Proper Thresholding

The first time a user invokes `combineall` for a given set of element maps, the threshold value chosen for each element is merely a guess, possibly based on past experience. However, correct segmentation of the image is critically dependent on the threshold values, and care must be taken to refine the threshold values to adjust the segmentation.

Two methods are available for ensuring proper thresholding, and a combination of both is usually used.

3.1 Histograms

Part of the output generated by `combineall` is a series of histogram files (`.hst` extension), one for each element. These histograms indicate the frequency of each intensity value (from 0–255). After exiting the program the first time, the `.hst` files reside in the working directory. They may be converted into PNG image files for viewing by using Gnuplot.

In the directory containing `combineall.c` there should be a file called `plothist`. Copy this file into the working directory. Open the file and change the name of all the `.hst` files to those that are actually present in your directory. Save the file and then enter the command `gnuplot plothist` from the command line. This will automatically create the PNG files (extension `.png`), which then may be viewed using any graphics display package that supports PNG formats. An example histogram for Ca is shown in Figure 1.

4 Code Documentation

⟨ *⟩≡

```
#include <stdio.h>
#include <stdlib.h>
```

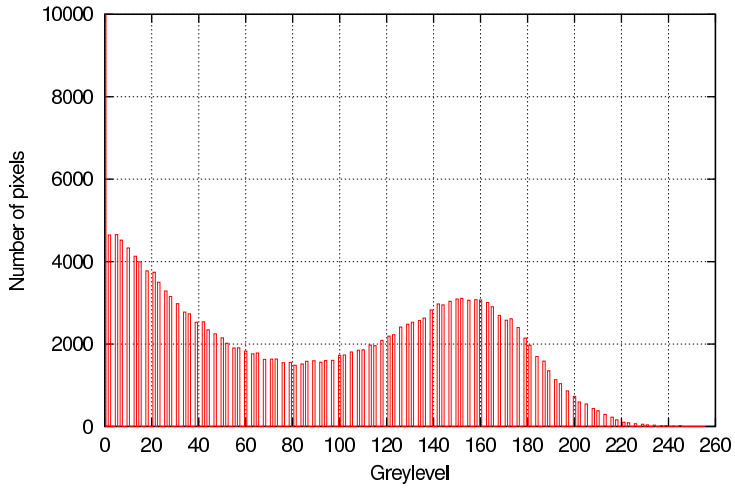

Figure 1: Example histogram for Ca output by combineall.

```
#include <malloc.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define CA 0
#define SI 1
#define AL 2
#define FE 3
#define S 4
#define K 5
#define MG 6
#define PORE 0
#define C3S 1
#define C2S 2
#define C3A 3
#define C4AF 4
```

```
#define GYP 5
#define FREELIME 6
#define K2SO4 7
#define MGCA 8
#define KAOLIN 9
#define SILICA 10
#define CAS 11
#define SLAG 12

static short int image[513][400][MG+1],imgact[513][400],imgproc[513][400];
long int phcount[SLAG+1];
int xsize,ysize;
int nmgca,nc3s,nc2s,nc3a,nc4af,ngyp,nlime,nalksulf,nkaolin,nsilica,ncas,nslag;

/* Routine to count neighboring pixels in an extent * extent square */
/* centered at each pixel in microstructure */
void segngh(ix,iy,extent)
        int ix,iy,extent;
{
        int ix1,iy1;
        int pin;

        for(ix1=(ix-extent);ix1<=(ix+extent);ix1++){
        for(iy1=(iy-extent);iy1<=(iy+extent);iy1++){

        if((ix1>=0)&&(ix1<xsize)&&(iy1>=0)&&(iy1<ysize)){
                pin=imgact[ix1][iy1];

                if(pin==C3S){
                        nc3s+=1;
                }
                else if(pin==C2S){
                        nc2s+=1;
                }
                else if(pin==C3A){
                        nc3a+=1;
                }
                else if(pin==C4AF){
                        nc4af+=1;
                }
                else if(pin==GYP){
                        ngyp+=1;
                }
                else if(pin==FREELIME){
                        nlime+=1;
                }
```

```
                else if(pin==K2SO4){
                        nalksulf+=1;
                }
                else if(pin==KAOLIN){
                        nkaolin+=1;
                }
                else if(pin==SILICA){
                        nsilica+=1;
                }
                else if(pin==CAS){
                        ncas+=1;
                }
                else if(pin==SLAG){
                        nslag+=1;
                }
                else if(pin==MGCA){
                        nmgca+=1;
                }
        }

        }
        }
}

/* Routine to execute a median filter on the 2-D image */
median()
{
        int xlo,xhi,ylo,yhi,i,j;
        int xlo1,xhi1,ylo1,yhi1,totngh;
        float maxfrac,pblack,pwhite,fc3s,fc2s,fc3a,fc4af,fgyp;
        float flime,fks,fmgca,fkaolin,fsilica,fcas,fslag;
        int color,pixin,colorm;
        long int cblack,cwhite,cc3s,cc2s,cc3a,cc4af,cgyp,ctot;
        long int clime,cks,cmgca,ckaolin,csilica,ccas,cslag;
        FILE *finfile,*foutfile,*fcolorfile;
        char file1[80],file2[80],file3[80];

        /* Initialize counters */
        cc3s=cc2s=cmgca=ckaolin=cc3a=0;
        cc4af=cgyp=cwhite=cblack=clime=cks=ccas=csilica=cslag=0;

        /* Now process each pixel in the 2-D microstructure */
        for(i=0;i<xsize;i++){
        for(j=0;j<ysize;j++){

                pixin=imgact[i][j];
```

```
        /* Determine the phase of the current pixel */
        color=PORE;
        if(pixin==C3S){
                color=C3S;
        }
        else if(pixin==C2S){
                color=C2S;
        }
        else if(pixin==C3A){
                color=C3A;
        }
        else if(pixin==K2SO4){
                color=K2SO4;
        }
        else if(pixin==MGCA){
                color=MGCA;
        }
        else if(pixin==KAOLIN){
                color=KAOLIN;
        }
        else if(pixin==FREELIME){
                color=FREELIME;
        }
        else if(pixin==C4AF){
                color=C4AF;
        }
        else if(pixin==SILICA){
                color=SILICA;
        }
        else if(pixin==CAS){
                color=CAS;
        }
        else if(pixin==SLAG){
                color=SLAG;
        }
        else if(pixin==GYP){
                color=GYP;
        }

    /* If the pixel is solid, may need to perform the median filter there */
    if(pixin!=PORE){
    /* Count the number of solid neighbors */
    nc3s=nc2s=nc3a=nc4af=ngyp=nmgca=nkaolin=nlime=nalksulf=nsilica=ncas=nslag=0;
    segngh(i,j,2);
    totngh=nc3s+nc2s+nc3a+nc4af+nmgca+ngyp+nkaolin+nlime+nalksulf+nsilica+ncas+nslag;
    /* If there are neighboring solid pixels, perform the median filter */
```

```
if(totngh>1){
        nc3s=nc2s=nc3a=nc4af=ngyp=nmgca=nlime=nkaolin=nalksulf=ncas=nsilica=nslag=
        segngh(i,j,3);
        totngh=nc3s+nc2s+nc3a+nc4af+nmgca+ngyp+nlime+nkaolin+nalksulf+nsilica+ncas
        if(totngh>0){
        /* Determine the most probable phase in the immediate neighborhood */
        maxfrac=(float)nc3s/(float)totngh;
        colorm=C3S;
        if(maxfrac<(float)nc2s/(float)totngh){
                maxfrac=(float)nc2s/(float)totngh;
                colorm=C2S;
        }
        if(maxfrac<(float)nc3a/(float)totngh){
                maxfrac=(float)nc3a/(float)totngh;
                colorm=C3A;
        }
        if(maxfrac<(float)nc4af/(float)totngh){
                maxfrac=(float)nc4af/(float)totngh;
                colorm=C4AF;
        }
        if(maxfrac<(float)ngyp/(float)totngh){
                maxfrac=(float)ngyp/(float)totngh;
                colorm=GYP;
        }
        if(maxfrac<(float)nlime/(float)totngh){
                maxfrac=(float)nlime/(float)totngh;
                colorm=FREELIME;
        }
        if(maxfrac<(float)nalksulf/(float)totngh){
                maxfrac=(float)nalksulf/(float)totngh;
                colorm=K2SO4;
        }
        if(maxfrac<(float)nkaolin/(float)totngh){
                maxfrac=(float)nkaolin/(float)totngh;
                colorm=KAOLIN;
        }
        if(maxfrac<(float)nsilica/(float)totngh){
                maxfrac=(float)nsilica/(float)totngh;
                colorm=SILICA;
        }
        if(maxfrac<(float)ncas/(float)totngh){
                maxfrac=(float)ncas/(float)totngh;
                colorm=CAS;
        }
        if(maxfrac<(float)nslag/(float)totngh){
                maxfrac=(float)nslag/(float)totngh;
```

```
        colorm=SLAG;
 }
 if(maxfrac<(float)nmgca/(float)totngh){
        maxfrac=(float)nmgca/(float)totngh;
        colorm=MGCA;
 }

/* Rules for updating the pixel being examined */
 if((color!=PORE)&&(maxfrac>=0.8)&&(totngh>=5)){
        color=colorm;
 }
 if((color!=PORE)&&(color!=K2SO4)&&(maxfrac>=0.6)&&(totngh>=5)){
        color=colorm;
 }
 if((color==C2S)&&(maxfrac>=0.5)&&(totngh>=5)){
        color=colorm;
 }
 if((color==FREELIME)&&(maxfrac>=0.25)&&(totngh>=4)){
        color=colorm;
 }
 if((color==MGCA)&&(maxfrac>=0.5)&&(totngh>=4)){
        color=colorm;
 }
 if((color==MGCA)&&(colorm==C3S)&&(maxfrac>=0.3)&&(totngh>=4)){
        color=colorm;
 }
 imgproc[i][j]=color;
 }

 /* Tabulate the new phase counts */
 if(color==PORE){
        cblack+=1;
 }
 else if(color==C3S){
        cc3s+=1;
 }
 else if(color==C2S){
        cc2s+=1;
 }
 else if(color==C3A){
        cc3a+=1;
 }
 else if(color==C4AF){
        cc4af+=1;
 }
 else if(color==GYP){
```

```
                cgyp+=1;
        }
        else if(color==FREELIME){
                clime+=1;
        }
        else if(color==K2SO4){
                cks+=1;
        }
        else if(color==KAOLIN){
                ckaolin+=1;
        }
        else if(color==SILICA){
                csilica+=1;
        }
        else if(color==CAS){
                ccas+=1;
        }
        else if(color==SLAG){
                cslag+=1;
        }
        else if(color==MGCA){
                cmgca+=1;
        }
}
}

}
}

/* Output the new phase fractions to the user */
ctot=cc3s+cc2s+cmgca+cc3a+cc4af+cgyp+cks+clime+ckaolin+ccas+csilica+cslag;
pblack=(float)cblack/(float)(ctot+cblack);
fc3s=(float)cc3s/(float)ctot;
fc2s=(float)cc2s/(float)ctot;
fks=(float)cks/(float)ctot;
fmgca=(float)cmgca/(float)ctot;
fc3a=(float)cc3a/(float)ctot;
fgyp=(float)cgyp/(float)ctot;
flime=(float)clime/(float)ctot;
fkaolin=(float)ckaolin/(float)ctot;
fcas=(float)ccas/(float)ctot;
fslag=(float)cslag/(float)ctot;
fsilica=(float)csilica/(float)ctot;
fc4af=(float)cc4af/(float)ctot;

printf("Fraction pore = %f \n",pblack);
```

```
        printf("Fraction C2S = %f \n",fc2s);
        printf("Fraction C3S = %f \n",fc3s);
        printf("Fraction C4AF = %f \n",fc4af);
        printf("Fraction C3A = %f \n",fc3a);
        printf("Fraction gypsum= %f \n",fgyp);
        printf("Fraction Free lime= %f \n",flime);
        printf("Fraction Kaolin= %f \n",fkaolin);
        printf("Fraction Alkali sulfates= %f \n",fks);
        printf("Fraction MgCa phase= %f\n",fmgca);
        printf("Fraction silica = %f\n",fsilica);
        printf("Fraction CAS = %f\n",fcas);
        printf("Fraction Slag = %f\n",fslag);
        for(j=0;j<ysize;j++){
        for(i=0;i<xsize;i++){
                imgact[i][j]=imgproc[i][j];
        }
        }
}

/* Routine to perform a median filter and output the resultant */
/* microstructure to two image files (one for further analysis and */
/* one for viewing) */
median1()
{
        int xlo,xhi,ylo,yhi,i,j;
        int xlo1,xhi1,ylo1,yhi1,totngh;
        float maxfrac,pblack,pwhite,fc3s,fc2s,fc3a,fc4af,fgyp;
        float flime,fks,fmgca,fkaolin,fcas,fsilica,fslag;
        int color,pixin,colorm;
        long int cblack,cwhite,cc3s,cc2s,cc3a,cc4af,cgyp,ctot,cslag;
        long int clime,cks,cmgca,ckaolin,ccas,csilica;
        FILE *finfile,*foutfile,*fcolorfile;
        char file1[80],file2[80],file3[80];

        cc3s=cc2s=cmgca=ckaolin=cc3a=cslag=0;
        cc4af=cgyp=cwhite=cblack=clime=cks=csilica=ccas=0;

        for(i=0;i<xsize;i++){
        for(j=0;j<ysize;j++){

                pixin=imgact[i][j];
                color=PORE;
                if(pixin==C3S){
                        color=C3S;
                }
                else if(pixin==C2S){
```

```
                color=C2S;
        }
        else if(pixin==C3A){
                color=C3A;
        }
        else if(pixin==K2SO4){
                color=K2SO4;
        }
        else if(pixin==MGCA){
                color=MGCA;
        }
        else if(pixin==KAOLIN){
                color=KAOLIN;
        }
        else if(pixin==SILICA){
                color=SILICA;
        }
        else if(pixin==CAS){
                color=CAS;
        }
        else if(pixin==SLAG){
                color=SLAG;
        }
        else if(pixin==FREELIME){
                color=FREELIME;
        }
        else if(pixin==C4AF){
                color=C4AF;
        }
        else if(pixin==GYP){
                color=GYP;
        }

if(pixin!=PORE){
nc3s=nc2s=nc3a=nc4af=ngyp=nmgca=nkaolin=nlime=nalksulf=nsilica=ncas=nslag=0;
segngh(i,j,1);
totngh=nc3s+nc2s+nc3a+nc4af+nmgca+ngyp+nkaolin+nlime+nalksulf+ncas+nsilica+nslag;
if(totngh>1){
        nc3s=nc2s=nc3a=nc4af=ngyp=nmgca=nlime=nkaolin=nalksulf=ncas=nsilica=nslag=
        segngh(i,j,2);
        totngh=nc3s+nc2s+nc3a+nc4af+nmgca+ngyp+nlime+nkaolin+nalksulf+ncas+nsilica
        if(totngh>0){
        maxfrac=(float)nc3s/(float)totngh;
        colorm=C3S;
        if(maxfrac<(float)nc2s/(float)totngh){
                maxfrac=(float)nc2s/(float)totngh;
```

```
        colorm=C2S;
}
if(maxfrac<(float)nc3a/(float)totngh){
        maxfrac=(float)nc3a/(float)totngh;
        colorm=C3A;
}
if(maxfrac<(float)nc4af/(float)totngh){
        maxfrac=(float)nc4af/(float)totngh;
        colorm=C4AF;
}
if(maxfrac<(float)ngyp/(float)totngh){
        maxfrac=(float)ngyp/(float)totngh;
        colorm=GYP;
}
if(maxfrac<(float)nlime/(float)totngh){
        maxfrac=(float)nlime/(float)totngh;
        colorm=FREELIME;
}
if(maxfrac<(float)nalksulf/(float)totngh){
        maxfrac=(float)nalksulf/(float)totngh;
        colorm=K2SO4;
}
if(maxfrac<(float)nkaolin/(float)totngh){
        maxfrac=(float)nkaolin/(float)totngh;
        colorm=KAOLIN;
}
if(maxfrac<(float)nsilica/(float)totngh){
        maxfrac=(float)nsilica/(float)totngh;
        colorm=SILICA;
}
if(maxfrac<(float)ncas/(float)totngh){
        maxfrac=(float)ncas/(float)totngh;
        colorm=CAS;
}
if(maxfrac<(float)nslag/(float)totngh){
        maxfrac=(float)nslag/(float)totngh;
        colorm=SLAG;
}
if(maxfrac<(float)nmgca/(float)totngh){
        maxfrac=(float)nmgca/(float)totngh;
        colorm=MGCA;
}

if((color!=PORE)&&(maxfrac>=0.8)&&(totngh>=5)){
        color=colorm;
}
```

```
if((color!=PORE)&&(color!=K2SO4)&&(maxfrac>=0.6)&&(totngh>=5)){
        color=colorm;
}
if((color==C2S)&&(maxfrac>=0.5)&&(totngh>=5)){
        color=colorm;
}
if((color==FREELIME)&&(maxfrac>=0.25)&&(totngh>=4)){
        color=colorm;
}
if((color==MGCA)&&(maxfrac>=0.5)&&(totngh>=4)){
        color=colorm;
}
if((color==MGCA)&&(colorm==C3S)&&(maxfrac>=0.3)&&(totngh>=4)){
        color=colorm;
}
imgproc[i][j]=color;
}

if(color==PORE){
        cblack+=1;
}
else if(color==C3S){
        cc3s+=1;
}
else if(color==C2S){
        cc2s+=1;
}
else if(color==C3A){
        cc3a+=1;
}
else if(color==C4AF){
        cc4af+=1;
}
else if(color==GYP){
        cgyp+=1;
}
else if(color==FREELIME){
        clime+=1;
}
else if(color==K2SO4){
        cks+=1;
}
else if(color==KAOLIN){
        ckaolin+=1;
}
else if(color==SILICA){
```

```
                csilica+=1;
        }
        else if(color==CAS){
                ccas+=1;
        }
        else if(color==SLAG){
                cslag+=1;
        }
        else if(color==MGCA){
                cmgca+=1;
        }
}
}

}
}

ctot=cc3s+cc2s+cmgca+cc3a+cc4af+cgyp+cks+clime+ckaolin+ccas+csilica+cslag;
pblack=(float)cblack/(float)(ctot+cblack);
fc3s=(float)cc3s/(float)ctot;
fc2s=(float)cc2s/(float)ctot;
fks=(float)cks/(float)ctot;
fmgca=(float)cmgca/(float)ctot;
fc3a=(float)cc3a/(float)ctot;
fgyp=(float)cgyp/(float)ctot;
flime=(float)clime/(float)ctot;
fkaolin=(float)ckaolin/(float)ctot;
fsilica=(float)csilica/(float)ctot;
fcas=(float)ccas/(float)ctot;
fslag=(float)cslag/(float)ctot;
fc4af=(float)cc4af/(float)ctot;

printf("Fraction pore = %f \n",pblack);
printf("Fraction C2S = %f \n",fc2s);
printf("Fraction C3S = %f \n",fc3s);
printf("Fraction C4AF = %f \n",fc4af);
printf("Fraction C3A = %f \n",fc3a);
printf("Fraction gypsum= %f \n",fgyp);
printf("Fraction Free lime= %f \n",flime);
printf("Fraction Kaolin= %f \n",fkaolin);
printf("Fraction Alkali sulfates= %f \n",fks);
printf("Fraction MgCa phase= %f\n",fmgca);
printf("Fraction silica phase= %f\n",fsilica);
printf("Fraction CAS phase= %f\n",fcas);
printf("Fraction Slag phase= %f\n",fslag);
printf("Total count is %ld \n",ctot);
```

```
printf("Enter binary filename to open for output \n");
scanf("%s",file1);
printf("%s\n",file1);
printf("Enter filename to open for phase ID image \n");
scanf("%s",file2);
printf("%s\n",file2);
printf("Enter filename to open for COLOR image \n");
scanf("%s",file3);
printf("%s\n",file3);
finfile=fopen(file1,"w");
foutfile=fopen(file2,"w");
fcolorfile=fopen(file3,"w");
fprintf(foutfile,"P2\n");
fprintf(foutfile,"%d %d\n",xsize,ysize);
fprintf(foutfile,"255\n");
fprintf(fcolorfile,"P3\n");
fprintf(fcolorfile,"%d %d\n",xsize,ysize);
fprintf(fcolorfile,"255\n");
for(i=0;i<xsize;i++){
for(j=0;j<ysize;j++){
        imgact[i][j]=imgproc[i][j];
        fprintf(finfile,"%c",(char)imgact[i][j]);
}
}
for(j=0;j<ysize;j++){
for(i=0;i<xsize;i++){
        fprintf(foutfile,"%d\n",imgact[i][j]);
        switch (imgact[i][j]) {
                case PORE:
                        fprintf(fcolorfile,"0 0 0\n");
                        break;
                case C3S:
                        fprintf(fcolorfile,"128 75 51\n");
                        break;
                case C2S:
                        fprintf(fcolorfile,"0 128 255\n");
                        break;
                case C3A:
                        fprintf(fcolorfile,"165 165 165\n");
                        break;
                case C4AF:
                        fprintf(fcolorfile,"253 253 253\n");
                        break;
                case GYP:
                        fprintf(fcolorfile,"255 255 0\n");
```

```
                        break;
                    case FREELIME:
                        fprintf(fcolorfile,"51 205 51\n");
                        break;
                    case K2SO4:
                        fprintf(fcolorfile,"255 0 0\n");
                        break;
                    case MGCA:
                        fprintf(fcolorfile,"255 105 180\n");
                        break;
                    case KAOLIN:
                        fprintf(fcolorfile,"255 165 0\n");
                        break;
                    case SILICA:
                        fprintf(fcolorfile,"0 255 255\n");
                        break;
                    case CAS:
                        fprintf(fcolorfile,"0 0 128\n");
                        break;
                    case SLAG:
                        fprintf(fcolorfile,"0 100 0\n");
                        break;
                    default:
                }
        }
        }
        fclose(finfile);
        fclose(foutfile);
        fclose(fcolorfile);
}

/* Routine to fill in one pixel void pixels */
/* surrounded by seven or eight solid neighbors */
void bkfill()
{
        int xlo,xhi,ylo,yhi,i,j;
        int xlo1,xhi1,ylo1,yhi1,totngh;
        float maxfrac,pblack,pwhite,fc3s,fc2s,fc3a,fc4af,fgyp;
        float flime,fks,fmgca,fkaolin,fsilica,fcas,fslag;
        int pixin,color,colorm;
        long int cblack,cmgca,cwhite,cc3s,cc2s,cc3a,cc4af,cgyp,ctot;
        long int nfill,clime,cks,ckaolin,ccas,csilica,cslag;

        cc3s=cc2s=cc3a=cc4af=cmgca=cgyp=cwhite=cblack=clime=cks=cslag=0;
        ckaolin=ccas=csilica=nfill=0;
```

```
for(i=0;i<xsize;i++){
for(j=0;j<ysize;j++){

        pixin=color=imgact[i][j];

/* If pixel is a void pixel totally surrounded by solids, */
/* convert it to the most prevalent phase in the surrounding pixels */
if(color==PORE){
nc3s=nc2s=nc3a=nc4af=ngyp=nmgca=nlime=nalksulf=nkaolin=nsilica=ncas=nslag=0;
segngh(i,j,1);
totngh=nc3s+nc2s+nc3a+nc4af+ngyp+nmgca+nlime+nalksulf+nkaolin+nsilica+ncas+nslag;
if(totngh>=7){
maxfrac=(float)nc3s/(float)totngh;
colorm=C3S;
if(maxfrac<(float)nc2s/(float)totngh){
        maxfrac=(float)nc2s/(float)totngh;
        colorm=C2S;
}
if(maxfrac<(float)nc3a/(float)totngh){
        maxfrac=(float)nc3a/(float)totngh;
        colorm=C3A;
}
if(maxfrac<(float)nc4af/(float)totngh){
        maxfrac=(float)nc4af/(float)totngh;
        colorm=C4AF;
}
if(maxfrac<(float)ngyp/(float)totngh){
        maxfrac=(float)ngyp/(float)totngh;
        colorm=GYP;
}
if(maxfrac<(float)nlime/(float)totngh){
        maxfrac=(float)nlime/(float)totngh;
        colorm=FREELIME;
}
if(maxfrac<(float)nalksulf/(float)totngh){
        maxfrac=(float)nalksulf/(float)totngh;
        colorm=K2SO4;
}
if(maxfrac<(float)nkaolin/(float)totngh){
        maxfrac=(float)nkaolin/(float)totngh;
        colorm=KAOLIN;
}
if(maxfrac<(float)nsilica/(float)totngh){
        maxfrac=(float)nsilica/(float)totngh;
        colorm=SILICA;
}
```

```
if(maxfrac<(float)ncas/(float)totngh){
        maxfrac=(float)ncas/(float)totngh;
        colorm=CAS;
}
if(maxfrac<(float)nslag/(float)totngh){
        maxfrac=(float)nslag/(float)totngh;
        colorm=SLAG;
}
if(maxfrac<(float)nmgca/(float)totngh){
        maxfrac=(float)nmgca/(float)totngh;
        colorm=MGCA;
}

if((color==PORE)&&(maxfrac>=0.3)){
        color=colorm;
}

}
}

/* If changed, update the pixel on the screen */
if(color!=pixin){
        nfill+=1;
}
imgproc[i][j]=color;
/* Tabulate and report the new phase counts */
if(color==PORE){
        cblack+=1;
}
if(color==C3S){
        cc3s+=1;
}
if(color==C2S){
        cc2s+=1;
}
if(color==C3A){
        cc3a+=1;
}
if(color==C4AF){
        cc4af+=1;
}
if(color==GYP){
        cgyp+=1;
}
if(color==FREELIME){
        clime+=1;
```

```
}
if(color==K2SO4){
        cks+=1;
}
if(color==MGCA){
        cmgca+=1;
}
if(color==KAOLIN){
        ckaolin+=1;
}
if(color==SILICA){
        csilica+=1;
}
if(color==CAS){
        ccas+=1;
}
if(color==SLAG){
        cslag+=1;
}

}
}
ctot=cmgca+cc3s+cc2s+cc3a+cc4af+cgyp+cks+clime+ckaolin+ccas+csilica+cslag;
pblack=(float)cblack/(float)(ctot+cblack);
fc3s=(float)cc3s/(float)ctot;
flime=(float)clime/(float)ctot;
fmgca=(float)cmgca/(float)ctot;
fkaolin=(float)ckaolin/(float)ctot;
fcas=(float)ccas/(float)ctot;
fslag=(float)cslag/(float)ctot;
fsilica=(float)csilica/(float)ctot;
fks=(float)cks/(float)ctot;
fc2s=(float)cc2s/(float)ctot;
fc3a=(float)cc3a/(float)ctot;
fgyp=(float)cgyp/(float)ctot;
fc4af=(float)cc4af/(float)ctot;

printf("Updated %ld pixels \n",nfill);
printf("Fraction pore = %f \n",pblack);
printf("Fraction C2S = %f \n",fc2s);
printf("Fraction C3S = %f \n",fc3s);
printf("Fraction C4AF = %f \n",fc4af);
printf("Fraction C3A = %f \n",fc3a);
printf("Fraction gypsum= %f \n",fgyp);
printf("Fraction Free lime= %f \n",flime);
printf("Fraction Alkali sulfates= %f \n",fks);
```

```
        printf("Fraction Kaolin phase= %f\n",fkaolin);
        printf("Fraction MgCA phase= %f\n",fmgca);
        printf("Fraction Silica phase= %f\n",fsilica);
        printf("Fraction CAS phase= %f\n",fcas);
        printf("Fraction Slag phase= %f\n",fslag);

        for(i=0;i<xsize;i++){
        for(j=0;j<ysize;j++){
                imgact[i][j]=imgproc[i][j];
        }
        }

}

/* Routine to fill in one pixel void pixels */
/* surrounded by eight solid neighbors */
void bkfill1()
{
        int xlo,xhi,ylo,yhi,i,j;
        int xlo1,xhi1,ylo1,yhi1,totngh;
        float maxfrac,pblack,pwhite,fc3s,fc2s,fc3a,fc4af,fgyp;
        float flime,fks,fmgca,fkaolin,fcas,fsilica,fslag;
        int pixin,color,colorm;
        long int cblack,cmgca,cwhite,cc3s,cc2s,cc3a,cc4af,cgyp,ctot;
        long int nfill,clime,cks,ckaolin,ccas,csilica,cslag;

        cc3s=cc2s=cc3a=cc4af=cmgca=cgyp=cwhite=cblack=clime=cks=0;
        ckaolin=csilica=ccas=nfill=cslag=0;

        for(i=0;i<xsize;i++){
        for(j=0;j<ysize;j++){

                pixin=color=imgact[i][j];

        if(color==PORE){
        nc3s=nc2s=nc3a=nc4af=ngyp=nmgca=nlime=nalksulf=nkaolin=nsilica=ncas=nslag=0;
        segngh(i,j,1);
        totngh=nc3s+nc2s+nc3a+nc4af+ngyp+nmgca+nlime+nalksulf+nkaolin+nsilica+ncas+nslag;
        if(totngh==8){
        maxfrac=(float)nc3s/(float)totngh;
        colorm=C3S;
        if(maxfrac<(float)nc2s/(float)totngh){
                maxfrac=(float)nc2s/(float)totngh;
                colorm=C2S;
        }
        if(maxfrac<(float)nc3a/(float)totngh){
```

```
        maxfrac=(float)nc3a/(float)totngh;
        colorm=C3A;
}
if(maxfrac<(float)nc4af/(float)totngh){
        maxfrac=(float)nc4af/(float)totngh;
        colorm=C4AF;
}
if(maxfrac<(float)ngyp/(float)totngh){
        maxfrac=(float)ngyp/(float)totngh;
        colorm=GYP;
}
if(maxfrac<(float)nlime/(float)totngh){
        maxfrac=(float)nlime/(float)totngh;
        colorm=FREELIME;
}
if(maxfrac<(float)nalksulf/(float)totngh){
        maxfrac=(float)nalksulf/(float)totngh;
        colorm=K2SO4;
}
if(maxfrac<(float)nkaolin/(float)totngh){
        maxfrac=(float)nkaolin/(float)totngh;
        colorm=KAOLIN;
}
if(maxfrac<(float)nsilica/(float)totngh){
        maxfrac=(float)nsilica/(float)totngh;
        colorm=SILICA;
}
if(maxfrac<(float)ncas/(float)totngh){
        maxfrac=(float)ncas/(float)totngh;
        colorm=CAS;
}
if(maxfrac<(float)nslag/(float)totngh){
        maxfrac=(float)nslag/(float)totngh;
        colorm=SLAG;
}
if(maxfrac<(float)nmgca/(float)totngh){
        maxfrac=(float)nmgca/(float)totngh;
        colorm=MGCA;
}

if((color==PORE)&&(maxfrac>=0.3)){
        color=colorm;
}

}
}
```

```
if(color!=pixin){
        nfill+=1;
}
imgproc[i][j]=color;
if(color==PORE){
        cblack+=1;
}
if(color==C3S){
        cc3s+=1;
}
if(color==C2S){
        cc2s+=1;
}
if(color==C3A){
        cc3a+=1;
}
if(color==C4AF){
        cc4af+=1;
}
if(color==GYP){
        cgyp+=1;
}
if(color==FREELIME){
        clime+=1;
}
if(color==K2SO4){
        cks+=1;
}
if(color==MGCA){
        cmgca+=1;
}
if(color==KAOLIN){
        ckaolin+=1;
}
if(color==SILICA){
        csilica+=1;
}
if(color==CAS){
        ccas+=1;
}
if(color==SLAG){
        cslag+=1;
}

}
```

```
        }
        ctot=cmgca+cc3s+cc2s+cc3a+cc4af+cgyp+cks+clime+ckaolin+ccas+csilica+cslag;
        pblack=(float)cblack/(float)(ctot+cblack);
        fc3s=(float)cc3s/(float)ctot;
        flime=(float)clime/(float)ctot;
        fmgca=(float)cmgca/(float)ctot;
        fkaolin=(float)ckaolin/(float)ctot;
        fcas=(float)ccas/(float)ctot;
        fslag=(float)cslag/(float)ctot;
        fsilica=(float)csilica/(float)ctot;
        fks=(float)cks/(float)ctot;
        fc2s=(float)cc2s/(float)ctot;
        fc3a=(float)cc3a/(float)ctot;
        fgyp=(float)cgyp/(float)ctot;
        fc4af=(float)cc4af/(float)ctot;

        printf("Updated %ld pixels \n",nfill);
        printf("Fraction pore = %f \n",pblack);
        printf("Fraction C2S = %f \n",fc2s);
        printf("Fraction C3S = %f \n",fc3s);
        printf("Fraction C4AF = %f \n",fc4af);
        printf("Fraction C3A = %f \n",fc3a);
        printf("Fraction gypsum= %f \n",fgyp);
        printf("Fraction Free lime= %f \n",flime);
        printf("Fraction Alkali sulfates= %f \n",fks);
        printf("Fraction Kaolin phase= %f\n",fkaolin);
        printf("Fraction MgCA phase= %f\n",fmgca);
        printf("Fraction Silica phase= %f\n",fsilica);
        printf("Fraction CAS phase= %f\n",fcas);
        printf("Fraction Slag phase= %f\n",fslag);

        for(i=0;i<xsize;i++){
        for(j=0;j<ysize;j++){
                imgact[i][j]=imgproc[i][j];
        }
        }

}

/* Routine to count the neighboring pixels that are pores */
int countngh(ix,iy)
        int ix,iy;
{
        int ix1,iy1;
        int nfound;
```

```
        nfound=0;
        for(ix1=(ix-1);ix1<=(ix+1);ix1++){
        for(iy1=(iy-1);iy1<=(iy+1);iy1++){

        if((ix1>=0)&&(ix1<xsize)&&(iy1>=0)&&(iy1<ysize)){

                if(imgact[ix1][iy1]==0){
                        nfound+=1;
                }
        }

        }
        }
        return(nfound);
}

/* Routine to remove the one-pixel regions of solids */
void onegone()
{
        int i,j;
        int nc3s,color,colorm;
        long int ndone;

        ndone=0;
        for(i=0;i<xsize;i++){
        for(j=0;j<ysize;j++){

        color=imgact[i][j];
        imgproc[i][j]=imgact[i][j];
        if(imgact[i][j]!=0){

                nc3s=0;
                nc3s=countngh(i,j);
                if(nc3s==8){
                        imgproc[i][j]=0;
                        ndone+=1;
                }
        }

        }
        }
        for(i=0;i<xsize;i++){
        for(j=0;j<ysize;j++){
                imgact[i][j]=imgproc[i][j];
        }
```

```
        }

        printf("Updated %ld pixels \n",ndone);
}

main(){
        char *junk1;
        int nskip,i,ix,iy,ival;
        int thresh;
        int thr[MG+1],thrcao,thrsio;
        int fval;
        float valsi,valca,ratiocasi,fval1;
        long int totcnt,hist[256],finaltot=0;
        FILE *infile,*outfile;
        char filen[80],valin,filert[80],elem[MG+1][10],filenow[80],fileout[80];

        totcnt=0;
        for(i=0;i<=SLAG;i++){
                phcount[i]=0;
        }

        /* Obtain user input */
        printf("Input x and y sizes \n");
        scanf("%d %d",&xsize,&ysize);
        printf("%d  %d \n",xsize,ysize);

        sprintf(elem[0],"ca");
        sprintf(elem[1],"si");
        sprintf(elem[2],"al");
        sprintf(elem[3],"fe");
        sprintf(elem[4],"s");
        sprintf(elem[5],"k");
        sprintf(elem[6],"mg");
        printf("Enter file root for processing \n");
        scanf("%s",filert);
        printf("%s\n",filert);
        printf("Enter number of pixels to skip at start \n");
        scanf("%d",&nskip);
        printf("%d\n",nskip);
        /* Read in all starting images */
        for(i=0;i<=MG;i++){
                printf("Enter threshold value for element %s (0-255) \n",elem[i]);
                scanf("%d",&thresh);
                printf("%d\n",thresh);
                thr[i]=thresh;
                sprintf(filenow,"%s%s.ras",filert,elem[i]);
```

```
        sprintf(fileout,"%s%s.hst",filert,elem[i]);
        printf("%s\n",filenow);
        infile=fopen(filenow,"rb");
        outfile=fopen(fileout,"w");
        for(ix=0;ix<256;ix++){
                hist[ix]=0;
        }
        for(ix=0;ix<nskip;ix++){
                fscanf(infile,"%c",&valin);
        }
        for(iy=0;iy<ysize;iy++){
        for(ix=0;ix<xsize;ix++){
                fscanf(infile,"%c",&valin);
                ival=valin;
                if(ival<0){ival+=256;}
                hist[ival]+=1;
                image[ix][iy][i]=ival;
        }
        }
        for(ix=0;ix<256;ix++){
                fprintf(outfile,"%d %ld \n",ix,hist[ix]);
        }
        fclose(infile);
        fclose(outfile);
}
/* Now create the histogram file for the Ca/Si ratio */
/* cementcasi.hst for example */
        for(ix=0;ix<256;ix++){
                hist[ix]=0;
        }
        for(iy=0;iy<ysize;iy++){
        for(ix=0;ix<xsize;ix++){
                if((image[ix][iy][CA]>=thr[CA])&&(image[ix][iy][SI]>=thr[SI])&&(im
                        fval1=50.*(float)image[ix][iy][CA]/(float)image[ix][iy][SI
                        ival=(int)fval1;
                        if(ival>255){ival=255;}
                        hist[ival]+=1;
                }
        }
        }

/* Now output the Ca/Si historgram file */
        sprintf(fileout,"%scasi.hst",filert);
        outfile=fopen(fileout,"w");
        for(ix=0;ix<256;ix++){
                fprintf(outfile,"%f %ld \n",(float)ix/50.,hist[ix]);
```

```
        }
        fclose(outfile);

printf("Enter threshold value for element free lime \n");
scanf("%d",&thresh);
printf("%d\n",thresh);
thrcao=thresh;
printf("Enter threshold value for silica \n");
scanf("%d",&thresh);
printf("%d\n",thresh);
thrsio=thresh;
printf("Enter critical ratio for C3S vs. C2S \n");
scanf("%f",&ratiocasi);
printf("%f\n",ratiocasi);

/* Set the initial phase assignments in the resultant image */
/* Process each pixel in the 2-D image in turn */
for(iy=0;iy<ysize;iy++){
for(ix=0;ix<xsize;ix++){
        totcnt+=1;
        fval=(-100);
        /* traverse the decision tree */
        if(image[ix][iy][CA]>thr[CA]){
                if(image[ix][iy][AL]>thr[AL]){
                        if(image[ix][iy][FE]>thr[FE]){
                                fval=C4AF;
                                phcount[C4AF]+=1;
                        }
                        else{
                                if(image[ix][iy][SI]>thr[SI]){
                                        if(image[ix][iy][MG]>thr[MG]){
                                                fval=SLAG;
                                                phcount[SLAG]+=1;
                                        }
                                        else{
                                                fval=CAS;
                                                phcount[CAS]+=1;
                                        }
                                }
                                else{
                                        fval=C3A;
                                        phcount[C3A]+=1;
                                }
                        }
                }
                else if(image[ix][iy][SI]>thr[SI]){
```

```
                        if(image[ix][iy][MG]>thr[MG]){
                                fval=SLAG;
                                phcount[SLAG]+=1;
                        }
                        else{
                                valsi=(float)(image[ix][iy][SI]);
                                valca=(float)(image[ix][iy][CA]);
                                if((valca/valsi)>ratiocasi){
                                        fval=C3S;
                                        phcount[C3S]+=1;
                                }
                                else{
                                        fval=C2S;
                                        phcount[C2S]+=1;
                                }
                        }
                }
                else if(image[ix][iy][S]>thr[S]){
                        fval=GYP;
                        phcount[GYP]+=1;
                }
                else if(image[ix][iy][MG]>thr[MG]){
                        fval=MGCA;
                        phcount[MGCA]+=1;
                }
                else if(image[ix][iy][CA]>thrcao){
                        fval=FREELIME;
                        phcount[FREELIME]+=1;
                }
        }
        else if(image[ix][iy][MG]>thr[MG]){
                if((image[ix][iy][SI]>thr[SI])&&(image[ix][iy][AL]>thr[AL])){
                        fval=SLAG;
                        phcount[SLAG]+=1;
                }
                else{
                        fval=MGCA;
                        phcount[MGCA]+=1;
                }
        }
        else if(image[ix][iy][S]>thr[S]){
                if(image[ix][iy][K]>thr[K]){
                        fval=K2SO4;
                        phcount[K2SO4]+=1;
                }
                else{
```

```
                                fval=GYP;
                                phcount[GYP]+=1;
                        }
                }
                else if(image[ix][iy][SI]>thr[SI]){
                        if(image[ix][iy][AL]>((int)(1.5*(float)thr[AL]))){
                                fval=KAOLIN;
                                phcount[KAOLIN]+=1;
                        }
                        else if(image[ix][iy][SI]>thrsio){
                                fval=SILICA;
                                phcount[SILICA]+=1;
                        }
                        else if(image[ix][iy][AL]>thr[AL]){
                                fval=KAOLIN;
                                phcount[KAOLIN]+=1;
                        }
                }

                imgact[ix][iy]=fval;
                if(fval<0){imgact[ix][iy]=0;}
        }
        }

        printf("Input any integer to continue \n");
        scanf("%d",&i);
        /* Remove the one-pixel particles */
        onegone();
        /* Fill in the one-pixel voids in two passes of the filling algorithm */
        bkfill();
        bkfill1();
        printf("Input any integer to continue \n");
        scanf("%d",&i);
        /* Perform two iterations of the median filtering algorthm */
        median();
        median1();
        printf("Input any integer to continue \n");
        scanf("%d",&i);
}
```