Study of the graininess models using the Macintosh computer

Apiwat Nutavej

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation
Accessed from

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
STUDY OF THE GRAININESS MODELS USING THE MACINTOSH COMPUTER

by

Apiwat Nutavej

B.S. Chulalongkorn University

(1982)

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the Center for Imaging Science in the College of Graphic Arts and Photography of the Rochester Institute of Technology

August 1987

Signature of the Author  Apiwat Nutavej

Accepted by  Name Illegible  8-27-87

Coordinator, M.S. Degree Program
The M.S. Degree Thesis of Apiwat Nutavej has been examined and approved by the thesis committee as satisfactory for the thesis requirement for the Master of Science degree.

Dr. Rodney Shaw, Thesis Advisor.

Professor Guy Johnson.

Mr. C.R. Myers, Associate Professor.

24 August 1917
Date
THESIS RELEASE PERMISSION FORM

ROCHESTER INSTITUTE OF TECHNOLOGY
COLLEGE OF GRAPHIC ARTS AND PHOTOGRAPHY

Title of Thesis: Study of the graininess models using the Macintosh computer.

I, Apiwat Nutavej, hereby grant permission to the Wallace Memorial Library of R.I.T. to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Signature: Apiwat Nutavej
Date: 24 August 1987
STUDY OF THE GRAININESS MODELS USING THE MACINTOSH COMPUTER

by

Apiwat Nutavej

Submitted to the Center for Imaging Science in partial fulfillment of the requirements for the Master of Science degree at the Rochester Institute of Technology

ABSTRACT

This study attempts to develop a Macintosh program that is able to simulate the photographic graininess models, to perform a Fast Fourier Transform, to plot a three-dimensional graph, and to display a digitized image. Because of these four integrated functions, the program could become a useful tool in Imaging Science. The potential advantages include ease of use and the visual interaction between computer and user.

When comparing the simulation results among three density models and four granularity models of random dots used in this study, the answers are found to be in close agreement with the hypothetical equations, with no significant statistical error as a result. The simulation shows that the granularity increases with the density and the dot size.

This study also serves as an example of programming the Macintosh in Pascal language. The techniques to incorporate a Macintosh User Interface in the program are explained in details.
ACKNOWLEDGMENTS

First of all, I would like to express my gratitude towards Dr. Rodney Shaw, Prof. Guy Johnson, and Mr. C.R. Myers whose guidance made this project possible. Special thanks are also due to my home university - Chulalongkorn University - from which I got the scholarship to pursue my studies. Furthermore, I would like to thank Prof. Sakda Siripant, Head of the department of Photographic Science, Chulalongkorn University, for his support.

Last but not least, I would also like to express my appreciation to my parents in their everlasting support and encouragement.
# TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1

CHAPTER

I. INTRODUCTION

II. LITERATURE REVIEW

2.0 The Random Dot Model

2.1 Transmittance, Density, and Granularity

2.2 The Grain Size Distribution

2.3 The Prediction Of Granularity From Density

2.4 The Effect Of Light Scattering

III. MACINTOSH PROGRAMMING

3.0 The Macintosh User Interface

3.1 Windows

3.2 Menus

3.3 The Event Driven Program

3.4 Custom Event Data Structure

3.5 The Event Routines

IV. THE PROGRAM AND UNITS

4.0 The Main Program

4.1 The Model1 Unit

4.2 The FFT Unit

4.3 The Plot3D Unit

4.4 The Printing Unit

V. THE SIMULATIONS AND RESULTS

5.0 Overview

5.1 The Simulation of the Monosized Random Dot Model

5.2 The Simulation of the Grain Size Distribution Model

5.3 The Simulation of the Light Scattering Model
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS ( CONT. )</th>
</tr>
</thead>
<tbody>
<tr>
<td>VI. CONCLUSIONS</td>
</tr>
<tr>
<td>6.0  The Models</td>
</tr>
<tr>
<td>6.1  Similar Projects</td>
</tr>
<tr>
<td>6.2  Personal Observation</td>
</tr>
<tr>
<td>LIST OF REFERENCES</td>
</tr>
<tr>
<td>VITA</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>The density and granularity of the monosized random dot models.</td>
<td>42</td>
</tr>
<tr>
<td>Table 2</td>
<td>The density and granularity of the multiple-sized dot model.</td>
<td>44</td>
</tr>
<tr>
<td>Table 3</td>
<td>The density and granularity of the light scattering model.</td>
<td>45</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Fig. 1. The Random Dot. 4
Fig. 2. The Standard Document Window. 12
Fig. 3. The Main Segment. 19
Fig. 4. The Program and Units. 20
Fig. 5. The Image Drawing Window. 23
Fig. 6. The Analysis Dialog. 24
Fig. 7. The GrayMap Text Window. 25
Fig. 8. The FFT Menu. 28
Fig. 9. The Plot Menu. 31
Fig. 10. The 3D Plotting Window. 32
Fig. 11. The 3D Preview Window. 34
Fig. 12. The Editor Window. 35
Fig. 13. The File Menu. 38
Fig. 14. The Printing Preview Window. 39
Fig. 15. A comparison of the granularity from eq. (8). 47
Fig. 16. A comparison of the granularity from eq. (12). 47
Fig. 17. A comparison of the granularity from eq. (20). 48
Fig. 18. The comparison between eq. (12) and eq. (20). 48
Fig. 19. The granularity of the multiple-sized dots. 49
Fig. 20. A comparison of the monosized random dot at the same density level. 50
Fig. 21. A comparison of the monosized random dot at the different density levels. 51
Fig. 22. A comparison between the monosized dots and the multiplied-sized dots. 52

vi
LIST OF FIGURES (CONT.)

Fig. 23. The Random Dot Images. 53
Fig. 24. The Ordered-Array Dot Images. 54
Fig. 25. A Comparison of The Images. 55
I. INTRODUCTION

The study of spatial noise in the uniformly exposed and processed coating of silver halide emulsions is one of the major subjects in Photographic and Imaging Science. The noise comes from the nature of the image-forming elements that they are distributed at random, thus causing the spatial inhomogeneities across the images. Because photography relies heavily on the use of silver halide emulsions, the noise mechanisms associated with this type of images have been widely studied.

There is a second type of images, whose elements are highly structured. The television pictures offer a good example of these type of images. Since its image-forming elements are in perfect array, they are not the cause of spatial inhomogeneities. Nevertheless, noise could come from the distribution of the activated image pixels across the entire picture itself. These noise mechanisms associated with the ordered-array picture also have been widely studied since the beginning of the television era.

The noise can be measured by scanning the image with a microdensitometer. The random density fluctuations will arise from the fluctuations in a number of particles that pass through a measuring aperture. These fluctuations can be plotted as a density function, with the mean density $D$ and the variance of $\sigma^2$. The variance of the density fluctuations indicate the spatial inhomogenities of the images and can be used as an objective measurement of the noise. The underlying statistical process of the distribution of image elements controls this density variance as follows: in the random elements, the distribution is Poisson, in the ordered-array elements, the distribution is binomial.

Apart from using the microdensitometer for an objective measurement of the noise in the images, each person can see this unevenness of a density and associate it with the
impression called graininess. However, the human visual system can distinguish the density differences quite well at a low density level but not at a high density level. Therefore, the graininess is a function of the mean density level as well as the fluctuation in density.

In recent years, computers have became more advanced and affordable. Most personal computers offer, in some way or another, the graphic capability of displaying images on the screen. It seems likely that a computer can be programmed to simulate the differences between the random-dots and the ordered-array dots.

However, the variations of the uniformly distributed dots are not easily observed by human. Digitized images - images that have been quantized to a series of bits for the purpose of digital storage and display - are a better choice to use and allow viewers to note the different between the image that consists of random dots and the image that consists of ordered-array dots.

The goal of this thesis is to develop a Macintosh application program that could be a useful tool in Imaging Science. The Macintosh is a powerful machine and has a clear, high resolution graphics, so it is expected that the program will be able to display all these types of images and allow the user to easily select the different styles of images for comparison. Furthermore, the program includes a fast fourier transform operation, a three-dimensional plotting, and density and granularity modeling.

The programming of a Macintosh is a difficult task, largely because one program's actions are interconnected to the others. The programmers are also have to study and understand more than 500 commands provided by the Macintosh User Interface Toolbox before they can start writing a Macintosh program.

The potential advantages of the program present in this study is that it includes the
ease of use, the integrity among different functions, the ability to view the data graphically, and the ability to print out the images. The program also offers detailed examples of how to write a Macintosh application.
II. LITERATURE REVIEW

2.0 The Random Dot Model

Photographers use the term 'graininess' to describe the visual sensation of the non-uniformity in a developed photographic image. The objective measurement of this value is called the granularity, G, and is usually characterized by the standard deviation of the density fluctuations occurring in a processed, uniformly exposed emulsion when scanned by a microdensitometer. The extraction of information from the photographic image is basically a problem of signal detection in a background of noise. The granularity quantifies this noise, and together with the image contrast, determines the threshold below which no more detail can be resolved by the human eye.

The most widely discussed theory of the noise associated with the silver halide emulsions is based on the random-dot model, as illustrated in Fig. 1. An opaque circular dots, each of area a, are placed at random on a transparent background with no restriction on overlap. The extensive history concerning these models were discussed by Hamilton(1), Lawton and Trabka(2), by Silberstein and Trivelli(3), by Webb(4), and others. It will be referred to as The Random Dot Model.

Fig. 1.
The Random Dot
In this model, the random distributions of the dots follows a Poisson distribution, and the optical properties of the dots follow geometrical optics.

2.1 Transmittance, Density, and Granularity

Let the average number of dots per unit area be given by \( n \), then the average number of dots whose geometric centers fall within any area \( a \) is a random variable following a Poisson law. The mean number of dot centers in the area is given by \( na \), and the standard deviation is given by the square root of \( na \). Therefore, the probability that an area \( a \) contains no dot centers is given by:

\[
p = e^{-na}
\]  

(1)

Let a circular aperture be placed over a uniformly illuminated random dot pattern. Of the radiant flux incident within aperture, only a fraction \( T \) can be transmitted. This fraction is called the transmittance.

As the aperture is moved across the pattern, the transmittance varies at random according to the amount of area left uncovered by grains. In this model, the mean of the transmittance is denoted by \( T \) and the rms deviation of transmittance is denoted by \( \sigma_T \). The density \( D \) is defined as:

\[
D = -\log_{10}T
\]  

(2)

However, the mean and standard deviation of the density of the random dot pattern are infinite, because the logarithm of the transmittance is infinite whenever the scanning aperture is completely blackened by grains, and this event can occur with the random dot model regardless of the area of the aperture or the average concentration of grains. Therefore, to specify a central value about which the density fluctuates, the models have to
use the density $D$ corresponding to the mean transmittance. Since the average transmittance $T$ is equal to the probability $p$, the density can be written as:

$$D = -\log_{10} p$$

(3)

Similarly, to specify the granularity, or amount of variation in the density, the models have to use the fluctuation of the standard deviation of the transmittance. For small density fluctuations, the granularity parameter $G$ is given by

$$G = A\sigma^2$$

(4)

where $\sigma^2$ is the variance of the density fluctuation within the measurement aperture $A$. For a poisson distribution of dot centers, the probability $p$ that $k$ dot centers lie within $a$ is given by:

$$p = e^{-na} (na)^k / k!$$

(5)

where $n$ is the average number of dot centers per unit area, and it follows that

$$\sigma^2_n = n_A$$

(6)

where $n_A = nA$ is the average number of dot centers within the measuring aperture $A$. Substitution of eqs. (5) and (6) into eqs. (3) and (4) yields the well-known Nutting(5) and Seidentopf(6) formulas for density and granularity, respectively.

$$D = \log_{10} e^{na}$$

(7)

$$G = \log_{10} e^{aD}$$

(8)

Eqs. (7) and (8), or minor modification of them, have been found to agree well with experimental data; for example, the linear relationship between $G$ and $D$ in certain photographic emulsions is well established(7).
2.2 The Grain Size Distribution

The formulas described previously are dealing with the monosized dots distributed at random. However, there are some circumstances which make these assumptions invalid. The silver halide crystal and the developed silver particles in normal photographic emulsions generally have a different size. This is because the emulsion with a distribution of grain sizes of silver halide has better photographic quality than the emulsion that has only one grain size. Therefore, the effect of this must be considered.

In the scanning area A, suppose that the grains have a projection area rA, where A is the projected area of the original silver halide grain and rA is the effective projected area after development. Nutting's equation (7) can be rewritten as

\[ D = \log_{10} e \frac{n rA}{A} \]  \hspace{1cm} (9)

the standard deviation \( \sigma \) satisfies the equation

\[ \sigma^2_D = (\log_{10} e n \frac{rA}{A})^2 \sigma^2_n \]  \hspace{1cm} (10)

The Selwyn granularity coefficient \( G \) is

\[ G = \sqrt{2A\sigma_D} \]  \hspace{1cm} (11)

therefore

\[ G = (2 \log_{10} e aD)^{1/2} \]  \hspace{1cm} (12)

Let the original silver halide grains be divided into size classes in such way that the \( i \)-th size class has the projected area \( A_i \) where \( i = 1, 2, 3 \ldots \). Let \( n \) be the number of developed grains of the \( i \)-th size class present in a particular area A of the film. The generalization of Nutting's equation (7) is given by

\[ D = \log_{10} e n \frac{rA}{A} \sum n_i A_i \]  \hspace{1cm} (13)
where it had been assumed that \( r \) has the same value of all grains. The average value of the optical density is given by

\[
D = \log_{10} e \frac{r}{r} A \sum n_i A_i
\]  

(14)

where \( n \) is the average value of \( n \). The area-weighted average grain area \(<A>\) is defined as:

\[
<A> = \frac{\sum n_i A_i^2}{\sum n_i A_i}
\]  

(15)

Substituting this into eq. (12) then gives

\[
G = \left( 2 \log_{10} e <A> D \right)^{1/2}
\]  

(16)

Note that the values of the quantities \( G \), \(<A>\), and \( D \) are all evaluated at the same given value of the exposure.

This expression for the granularity coefficient is the same as that obtained earlier as eq. (12), except that the value \( A \) is now replaced by the average grain area \(<A>\).

### 2.3 The Prediction Of Granularity From Density

Recall from the Nutting and Seidentopf formulas that

\[
D = \log_{10} e n a
\]  

(7)

\[
G = \log_{10} e a D
\]  

(8)

In one of the few studies in which \( a \) was measured experimentally, Romer and Morawski(8) found that density values calculated from eq. (7) were only about one-half of the experimental ones. Farnell and Solman(9) confirmed the need for this additional constant and suggested that the constant accounts for light scattering within the film. They also found that they could replace \( a \) by 1.3\( a \), where \( a \) is the mean projective area of the undeveloped silver halide grain. Thus they suggested that eq. (7) be replaced by
\[ D = \log_{10}e \text{kna} \quad (17) \]

where \( k \) is constant whose value is about 2.6. Farnell, Sanders, and Solman(10) later suggested a similar modification to equation \( A^{1/2} \sigma = \log_{10}e \text{an}^{1/2} \).

\[ A^{1/2} \sigma = \log_{10}e \text{kan}^{1/2} \quad (18) \]

The resulting equation after rearranging slightly is:

\[ \sigma = D/(\text{na})^{1/2} \left[ 1 + \frac{gD}{4\log_{10}e} \right]^{1/2} \quad (19) \]

For the purpose of curve fitting, eq. (19) should be easily used by adjustment of values of \( g \) until agreement with experimental values is achieved. However, this formula could be simplified even more. Trabka and Dorner(11) found that the best fit of their data required the value \( g = 0.41 \). For simplicity, \( g = \log_{10}e = 0.43 \) can be used. This assumption leads to eq. (20):

\[ \sigma = D/(\text{na})^{1/2} \left[ 1 + \frac{D}{4} \right]^{1/2} \quad (20) \]

### 2.4 The Effect Of Light Scattering

The dimensions of typical photographic grains are comparable to the wavelength of light. This implies that the consideration of the theory of wave optics should lead to a greater insight into the extinction properties of the grains. The model therefore replaces the area \( a \) in eq. (7) by the extinction cross-section, \( C_{\text{ext}} \), and arrives at a new formulation of the density model:

\[ D = \log_{10}e \text{n Cext/A} \quad (21) \]

This is similar in form to a model suggested by Salib, Depalma, and Gasper(12). The cross section for extinction, \( C_{\text{ext}} \), is associated with an equivalent sphere corresponding to a grain of a particular size and shape.
The efficiency at which particles scatter or absorb light is given by their efficiency factor \( Q \). The eq. (21) can be written in terms of the extinction efficiency, \( Q_{\text{ext}} \), and in addition, \( \bar{n} \) is the mean value of \( n \), then the mean density is given by:

\[
D = \log_{10} e \frac{n Q_{\text{ext}}}{A} \tag{22}
\]

Finally, eq. (22) can be written in terms of \( n \) as

\[
D = \log_{10} e n a Q_{\text{ext}} \tag{23}
\]

\[
G = \log_{10} e a n^{1/2} Q_{\text{ext}} \tag{24}
\]

Eq. (23) expresses the optical density as a function of the grain cross-section area, concentration, and extinction efficiency. Eq. (24) for the granularity in addition includes the effect of scanning aperture area \( A \), which correlates with variations in image magnification. The extinction efficiency accounts for both the absorption and scattering characteristics of grain and is a function of grain size, relative refractive index, and the wavelength of light.
III. MACINTOSH PROGRAMMING

3.0 The Macintosh User Interface

Prior to the development of the Macintosh, most programs directed the behavior of the user by passing through a series of modes which limited the number of valid actions. Examples of this style of programs include those with a screenful of menus in which only the options on the current menu or sub-menu can be selected. The user must constantly maneuver within the program to reach the point where they can issue a command or initiate an action. Programs of this type are an improvement over the older command-line style interface in terms of ease of use. Still, they require a memorization of menus and locations of commands.

In contrast, the Macintosh User Interface often strives to make available as many options as possible. Skill and careful design are required so that the many alternatives do not confuse the user.

The Macintosh program should have three qualities: responsiveness, permissiveness, and consistency. Responsiveness means that the user's actions tend to have a direct result. The user should be able to reach the desired command directly and instantaneously. Permissiveness means that the applications tend to allow the user to perform an action that is reasonable. Consistency is the most important among these three principles; the program should not force the user to learn a new interface for each application. Consistency is easier to achieve on the Macintosh than on many other computers because the Macintosh provides many of the routines used to implement the user interface in the Operating System and User Interface Toolbox.
The Macintosh displays information on the screen graphically because it has no text mode. Nevertheless, a program can make a distinction between text and graphic, only the images on the screen are treated as graphics. For example, while word processors might consist of nothing but text, others such as graphic-oriented applications use text almost incidentally. Graphics, the pictures drawn either by the user or by the application, are used extensively on the Macintosh even in places where other applications use text.

3.1 Windows

The applications can display an information on the screen by using the window, and the user can respond by clicking or typing characters at some specific points inside the window. Fig. 2. shows the structure of the most commonly encountered type of the Macintosh window - a standard document window.

![The Standard Document Window](image)

**Fig. 2.**
The Standard Document Window.
Some applications may be able to keep several windows on the screen at the same time. Each window is in a different plane and windows can be moved around on the Macintosh's screen much like pieces of paper can be moved around on a desktop. Each window can overlap those behind it, and can be overlapped by those in front of it.

The document windows have a close box that, when clicked, makes the window go away. The application in control of the window determines what is done with the window visually and logically when the close box is clicked. Visually, the window can be made to disappear from the screen. Logically, the information in the window is either retained or discarded. If an application does not support closing a window with a close box, it should not include a close box on the window.

Of all the windows that are displayed on the screen, the user can work in only one window at a time. This window is called the active window. To make a window active, the user clicks in it. Making a window active has two immediate consequences:

- The window changes its appearance: Its title bar is highlighted and the scroll bars and a size box are shown.
- The window is moved to the frontmost plane, so that it is shown in front of any windows that it overlaps.

The user can move a window to a new location on the screen by dragging it by its title bar. If a window has a size box in its bottom right corner, the user can change the size of the window by dragging this size box.
3.2 Menus

The menu bar is displayed at the top of the screen. It contains a number of words and phrases: These are the titles of the menus associated with the current application. Each application has its own menu bar.

Only menu titles appear in the menu bar. If all the commands in a menu are currently disabled (that is, when the user can not choose them), the menu title will be drawn in gray.

To choose a command, the user positions the cursor over the menu title and presses the mouse button. The application highlights the title and displays the menu items. While holding down the mouse button, the user moves the cursor down the menu. As the cursor moves to each command, the command is highlighted. The command that is highlighted when the user releases the mouse button is chosen. As soon as the mouse button is released, the command blinks briefly, the menu disappears, and the command is executed.

3.3 The Event Driven Program

At the heart of every well-written Macintosh application there is a Main Event Loop, where the program spends most of its time waiting for the user to do something. When the user pushes the mouse button, types a character, or in some other way initiates an action, an event is generated so that the program can interpret and correctly respond to. Most events are kept in an event queue, where they are stored in order of priorities by the Operating System Manager.

The important types of events record user's actions, which can be in the
following categories:

**mouse events** - occur when the user presses or releases a mouse button.

**keyboard events** occur when the user presses or releases a key on the keyboard.

**disk events** occur when the user inserts a disk into a disk drive.

**window events** - occur when the user deactivates a window, activates a window, or expose the hidden part of a window.

**null events** - occur when there is no event left in the event queue.

### 3.4 Custom Event Data Structure

This program keeps the events in the custom data structure `EventStuff`, which contains most of the events that are important:

```pascal
type
EventStuff = Record
  WhichWindow : WindowPtr;
  WindowPart : Integer;
  WhichDialog : DialogPtr;
  DialogItem : Integer;
  WhichControl : ControlHandle;
  ControlPart : Integer;
  MenuH : MenuHandle;
  MenuNum : Integer;
  ItemNum : Integer;
  TEH : TEHandle;
  Ch : Char;
  NewMouseDown : EventRecord;
  LastMouseDown : EventRecord;
  DoubleClick : Boolean;
  DialogEvent : Boolean;
  WindowIsNew : Boolean;
End;
```
By examining the different fields of the EventStuff record, the programmer can find out what event has happened at any given moment. The meaning of each field is explained as follows:

**WhichWindow**
This field returns a pointer to the specific window, if there is an event involving that window.

**WindowPart**
- This field returns a window part code from the toolbox FindWindow() function, as explained in the Window Manager chapter of the Inside Macintosh(13).

**WhichDialog**
This field returns a pointer to a modeless dialog.

**DialogEvent**
This field returns TRUE, if the IsDialogEvent() toolbox function is true, meaning that there is an event involving the dialog box.

**DialogItem**
- This field returns an item number in the modeless dialog which has been selected.

**WhichControl**
This field is initially set to nil, unless there is an event involving a mouse click inside some control (e.g., a pushButton, a scrollBar, etc.), then it will return with a handle point to that control.

**ControlPart**
This field returns a control part code from the toolbox FindControl() function, as explained in the Control Manager chapter of the Inside Macintosh(14).

**MenuH**
This field is set to nil unless the user selects an item from a pull-down menu. If the menu item is actually selected, the MenuH field returns a handle to that menu, the MenuNum field returns the value of menuID and the ItemNum field also returns an item number.

**Ch**
This field returns the character typed in from the keyboard. If no key has been pressed, this field will have a value of char(0).

**TEH**
This field contains a handle point to the Text Edit record associated with the window. If there is no text record in that window TEH field is set to nil.

The most important procedure in this project is the HandleEvent() routine. HandleEvent() is a high-level procedure that is called whenever an event has taken place. These events are in turn handled by the subsidiary routines in the EventMaster Unit Library. They deal with the regular processing tasks such as redrawing newly exposed
parts of a window's content, scrolling a graphic, or zooming a window, etc.

3.5 The Event Routines

PROCEDURE SystemTask;

The SystemTask procedure causes the desk accessory to perform the periodic action defined for it, if any such action has been defined and if the proper time period has passed since the action was last performed. The application should call this procedure at least every sixtieth of a second.

FUNCTION GetNextEvent ( eventMask : Integer;
    var theEvent : EventRecord ) : Boolean;

GetNextEventO returns a boolean result TRUE, if the application program has to respond to the event. The event is returned in the parameter theEvent.

• FUNCTION XTGetNextEvent ( eventMask : Integer;
    var theEvent : EventRecord ) : Boolean;

The function XTGetNextEvent() calls the Toolbox function GetNextEvent() to see if any events have occurred since the last time it was called. If more than one event has occurred, they will be stored in order of occurrence in an event queue.

Unlike GetNextEvent(), when there is no event waiting to be processed (i.e., GetNextEvent() return FALSE), XTGetNextEvent() provides the Dialog Manager with an opportunity to blink the insertion point in edit fields and other housekeeping chores.

• PROCEDURE MaintainCursor (    systemCursor : Cursor;
    textCursor : Cursor;
    graphicCursor : Cursor    );

MaintainCursor() gives the program an ability to switch between several cursors. That cursor specified by the systemCursor parameter is displayed whenever the cursor is over the menu bar, the desktop, or a window without text or graphics. If the cursor position is over the viewing rectangle of an active editable text, the textCursor cursor will be displayed. If the cursor is positioned over a graphics window, the graphicCursor cursor will be displayed. The predefined cursor, arrow, will usually be used for the systemCursor and often for the graphicCursor as well. Whenever an editable text record is visible and active on the screen, the iBeam cursor is generally used as the textCursor.
• PROCEDURE HandleEvent ( theEvent : EventRecord;
  var whatHappened : EventStuff );

The HandleEvent() procedure is responsible for all of the normal event processing which occurred in the application. The program will call this routine when the XTGetNextEvent() returns TRUE. HandleEvent() passes an EventRecord and a pointer to an EventStuff record. Copied inside the EventStuff record is a complete account of the event, what it affected, and what has been done about it.
IV. THE PROGRAM AND UNITS

4.0 The Main Program

The main segment of the program consists of three routines, which initialize a Toolbox, set up the windows, and enter the main event loop, as illustrated in fig. 3.

![Diagram of program segments](image)

Fig. 3.
The Main Segment.

The main program also opens a resource file that contains several resources, such as cursors or dialog templates. These resources are created from the Apple's ResEdit program, which generates a resource directly; therefore, no resource source code is provided. They are saved in the file called NoiseResource.

The main segment incorporates several units into a program code, some are created as a new segment, some are not. Fig. 4. shows the units that this program is using. The more important codes that get executed frequently are kept in the main segment where it stays locked and always present (All the libraries and the unit that linked by the $U$ directive). The other units are in the separate segments because they are not needed all the time and will only be loaded when necessary (the unit that linked into a main segment by $U<$ directive).
Fig. 4. The Program and Units.
All of the actions take place in the main event loop. This main loop can essentially be simplified into several steps as shown in listing 1.

```pascal
Procedure MainEventLoop;
Begin
  Repeat
    SystemTask;
    MaintainCursor();
    If XTGetNextEvent() then begin
      HandleEvent()
      with whatHappened do begin
        { responding to the events }
      end;
    end;
    UnloadSeg();
  Until ExitRequest();
End;
```

Listing 1.
The Main Event Loop

The detailed description of all the routines use in this main event loop are explained in chapter III of this thesis. First, the program calls SystemTask to let a system driver, such as a desk accessory, have a chance to handle its jobs. After that, the program calls MaintainCursor() routine, which changes the cursor to an appropriate one when it passes over desktop, graphic, or text area.

The XTGetNextEvent() function is called to check whether there are any events waiting to be processed. If it returns TRUE, events have occurred and must be handled accordingly.

The most important routine of all is the HandleEvent() procedure which should be called to take care of the events. This procedure can handle virtually all events and process most of its parts. The rest of the events that require a special attention from the programmer are kept in the EventStuff record and the programmer can handle this
himself from inside the `with whatHappened` statement (`whatHappened` is a local variable of the type `EventStuff`).

The last routine that is called in this main loop (before the loop repeats itself once more), is the Toolbox `UnloadSeg()` routine. This routine unlocks the program code that may be getting loaded, either from the `HandleEvent()` or from inside the `with whatHappened` statement, and make it purgable. If next time through the loop the Macintosh needs more memory space, it will be able to purge these units.

The loop continues as long as the user does not select menu item 'Quit' from the file menu. If the user selects this 'Quit' menu item or type an equivalent command-Q, the `ExitRequest()` function returns TRUE and the loop is terminated. The Operating System Manager then returns a control to the Finder.

4.1 The Model1 Unit

This unit contains a program code responsible for drawing the images. The nature of these images can be separated into two groups: one is the randomly distributed dots, and the other is the digitized image. The models that can be used are the following:

The random dots models:

- The dots are having the same size.
- The dot are having different sizes.
- The light is scattered around the dots.

The ordered-array models:

- The elements are having the same size.
- The elements are having different sizes.
Fig. 5. The Image Drawing Window.
Fig. 6. The Analysis Dialog.
Fig. 7. The GrayMap Text Window.
In order to construct an image from a digitized picture, a gray-map informations of that pictures must be loaded into a GrayMap Text window. The user can easily makes his own gray map file from any commercially available software that let him save a digitized image as a text-only file.

For example, a ThunderScane digitizer can save a file in this format. The user selects a portion of the image that he wants to use, holds down the option key and selects the menu command Halftone. Instead of producing a halftone image, the ThunderScane software sends that portion of the image to a disk as a stream of characters, each with its ASCII value corresponding to a gray level of the original image.

The random dot models, on the other hand, do not need a gray-map information. The models generate their own random numbers and use this information to place the dots on the screen.

All of the drawings are taken place inside Image Drawing window, as shown in Fig. 5. In this window the user can adjust three parameters that affect the models in different ways. These values are the following:

%na parameter

This is the percentage of a coverage area. Its value is calculated from the number of dots in the display window (n) times the area of a dot (a). Since the dots are positioned at random, the dots concentration %na value of 100 percents does not completely cover the measuring aperture. This parameter has a meaning only in the random dots models.
Size parameter

This variable controls the size of a dot because the program sets the parameter \( a \) in the variable \( \%na \) to this value. In the models, the larger the dots the smaller the number of the dots presented, in order to keep the percentage of the coverage area constant.

Gray parameter

This variable is the threshold gray level. It has a meaning only in the models that are using the gray-map information. The gray levels that are equal to or higher than this setting are treated as it is capable of turning on the detector elements. The gray levels below the setting are not able to activate the detectors and therefore have no effect on the resulting image.

4.2 The FFT Unit

A fast fourier transform (FFT.) has many uses in a digital image processing. This operation transforms an image from a spatial domain to a frequency domain, thus essentially reducing the image into a periodical sine wave. Because the frequency is easier to manipulate than the space, this operation is very useful to correct the imperfection of the image. Usually, the image is transformed into a frequency domain, manipulates the spectrum, and is transformed back into a space domain.

The algorithm used in this study creates the data storage area in memory, thus speeds up the calculation but requires a lot of free memory space. Fig. 8. shows the FFT menu and its effect:
Two-dimensional arrays holding data for FFT operation.

Fig. 8. The FFT Menu.
The user can select data from 8 to 1024 points, but a large number of points require long calculation times. This problem becomes prominent for the two-dimensional array, where the calculations are increased by the power of two. The program limits the maximum number of points in two-dimensional FFT to 64x64.

The FFT unit code is relatively independent from the rest of the program. It does not use any window and the only interaction between the user is through the standard file dialog.

When the user selects menu item 2 Dimension from the FFT menu, the program calls the procedure DoFFT. When invoked, this procedure displays a dialog box asking for a file name for loading the data, as shown in Fig. 8. If the user dismisses this dialog box (clicking on 'Cancel' button), the program terminates and returns to the main event loop.

On the other hand, if the user clicks 'Open', the program loads data, performs FFT operation, and displays another dialog box asking for a file name to be saved. The user has a choice to accept or dismiss the file saving operation.

The required data are in standard text-only format, each value must be separated by a carriage return. The pascal procedures, ReadLn() and WriteLn(), are used for loading and saving the data on the disk. These operations can convert a text buffer into a numeric value, therefore the text-only file can be used. The advantage is that the user can edit this data by any word processing program.
4.3 The Plot3D Unit

This unit implements the 3D QuickDraw routines. The main purpose is to plot a three-dimensional graph from the data stored in a two-dimensional array, such as the resulting array received from the fast fourier transform operation. However, the FFT operation yields a complex number - a number that contains a real part and an imaginary part. It is commonly known that, in order to plot this graph, the data have to be calculated as an amplitude which has a maximum and a minimum value normalized to 1.0 and 0.0, respectively.

The program can use this amplitude for plotting in a Z-axis. The values along X and Y axes are taken from the array's indexing number. Therefore, of all the 3 coordinates in space, the user needs to supply only the Z-axis variables.

These data can be read from the data file that the FFT unit has generated. The file is saved in the text format, each value is kept in sequential order separated by a carriage return. Pascal procedures, WriteLn() and ReadLn(), are used to save and restore data, respectively.

The ReadLn() operation can read a text from a file buffer and convert it into a numeric value. The WriteLn() operation is doing the opposite, it takes a numeric data and writes it down to a disk file as a text. Fig. 9 shows the Plot menu and its effect:
Fig. 9. The Plot Menu.
Fig. 10. The 3DPlotting Window.
The menu item **New Graph** calls the procedure **DoWindow3D**. This procedure initializes a three-dimensional graphic window, sets data arrays to the size set by Points value in the **FFT** menu, and puts up a dialog box asking the name of the data file that the user wants to load. The user has two choices; he can accept or dismiss this dialog box. The values are loaded only if the user confirms the choice.

The program does not draw a graph immediately after the data have been loaded. The menu item **Draw Graph** in the **Plot** menu must be selected in order to execute the procedure **UpdatePlot3D** which actually draws a picture. The graph can be in three different forms: a wire-frame drawing, a slice-surface drawing, or a solid-surface drawing.

The wire-frame drawing is the easiest to draw. The program has to connect the lines through all the data points. No other operations are needed.

The slice-surface drawing is a little more complicated. The program has to draw a line through the data points first, and then sets up a region under the line. This region is filled with solid 'ink', which effectively removes anything underneath the line.

The solid-looking surface is the most complicated. It is in fact the combination of the above two drawings. The program sets up a region from adjacent points and fills the region to remove any line behind or below it.

Before the program draws a graph, it reads an information of scale, rotation, elevation, etc. from the scrollbars in a preview window. The user can select the menu item **Preference...** which calls a procedure **PrefPlot3D** and makes this window visible.

The preview window shows a small 3D image that is the simplified picture of the image from the **3D Plotting** window. When the program changes an entire picture in the **3D Plotting** window, the calculations of many data points make the process very slow.
PrefPlot3D;
MouseInPrefPlot3D;

Perspective

Pitch

Roll

Yaw

Elevation

Zoom lens

O Back

O Floor

O Line

O Plot

O Grid

Fig. 11. The 3D Preview Window.
The cursor changes from an arrow to a cross-hair.

The box shows the value of data at that point.

Fig. 12. The Editor Window.
However, for this scale down image, the calculations are much faster. When the user clicks on 'Okay' button, the program calls a procedure `UpdatePlot3D` to perform the massive calculations of the data set and to redraw the entire image.

The last menu item, the Editor On option, turns on the build-in editing function of a program. This editor selects one part of the data set and assigns a region on each of the data points. When the user click the mouse button in this region, which appears on a screen as series of dots along a curve, the program draws an (X, Y) indexes and a value of that particular data point. The editor window is shown in Fig. 12.

4.4 The Printing Unit

Some commercial software allows the user to preview a graphic before printing. The program displays a reduced view of the image on an area that is a print page. The user can move an image freely around the area of printing, and then print the image at the selected position. This approach greatly emphasizes a direct control from the user and visual feedback from Macintosh. The program in this thesis also handled the printing this way.

Fig. 13 shows the preview printing window that has a reduced view of image relative to a print page and buttons label 'Print' and 'Cancel'. After hitting the 'Print' button printing starts immediately, while hitting the 'Cancel' button the printing stops. The program can drag the image around by calling the `DragGrayRgn()` function.

Text printing does not have a preview window, it just prints the text according to a normal practice. The text starts to print at the top of a page and continues to the end. Unlike graphics, the text can be displayed in various ways - it can be made to wrap around when it reaches a right margin, or it can be made to break to a new line only at returns. In this
application, the text is likely to be a gray map file - a continuous line of text separated by return, so the second method was chosen.

Before printing the text the program has to calculate many of the text attributes such as ascent, descent, line width, line height, etc. One font family differs from other font families, also in one family there are many font sizes, which must be considered.

Most important, however, are the features necessary for printing loop. There are two ways of displaying the text on the screen from the Toolbox routine the TextBox() and the DrawText() procedure.

The TextBox() routine supplied by TextEdit is considerably more versatile than its QuickDraw equivalent. For example, it allows the programmer to justify the text and automatically warp the text in a rectangular box.

The biggest drawback of the TextBox() routine is that each time it draws, redraws, or creates a new area it must call EraseRect(). It takes a lot of time to erase the rectangle for each box in case a lot of text is displayed at once. TextBox() has been known to be very slow when printing on the LaserWriter because of the EraseRect() operation.

DrawText() is a QuickDraw routine which doesn't need EraseRect() so it is considerably faster. In this application, when the text has only one size, style, and no wrap around, DrawText() is naturally a better choice. The program calls DrawText() for each line of text instead of the TextBox().
Fig. 13. The File Menu.
DoPrinting;

GetWData()

if WD.TEH <> nil
    DoPrintText()

if WD.windowPic <> nil
    DoShowGraphicWindow()

Alert()

end;

mouseinPrinting()

DragGrayRgn()

KillWindow()

DoPrintGraphic()

Print

Cancel

Fig. 14. The Printing Preview Window.
V. THE SIMULATIONS AND RESULTS

5.0 Overview

For the production of a graininess sample, a system is needed to vary the parameter in eqs. (7) through (24). The eqs. (7), (8), (12), and (20) need to vary the number of dots, \( n \), and the area of dot, \( a \). The eqs. (14) and (16) need to vary the dot size, \( a_i \), and the number of dots of each size, \( n_i \). The eqs. (23) and (24) need to vary the light scattering coefficient, \( Q_{\text{ext}} \). The starting of this process is a program that generates the random dots. The number of parameters such as \( a \), \( n \), and \( Q_{\text{ext}} \) could be varied according to the monosized, the multiple-sized, or the light scattering models.

5.1 The Simulation of A Monosized Random Dot Model

On a Macintosh screen, the location of a pixel is represented by \( X,Y \) coordinates. The coordinate origin (0,0) is located at the top-left corner of the screen. Horizontal coordinates increase as it moves from left to right, and vertical coordinates increase as it moves from top to bottom. In order to generate random dots, the program must create a set of random numbers and uses these numbers as a \((X,Y)\) location.

Most computer languages provide a programmer with a function that can generate random numbers. In this thesis, that function is called \texttt{Random}. The random function generates a pseudo-random integer, which is uniformly distributed in the range -32767 to 32767.
The values in the sequence may appear at random, but actually they are derived from a 'seed' number. The 'seed' number is a global variable and has a name RandSeed, which program initializes to 1. If a Random function receives the same 'seed' number, the values that it generates will have the same sequence.

The program can display a dot at random on the screen by generating a random number and use this number as the X,Y coordinates. However, the numbers are between the range -32767 to 32767. The program must scale the values down to a suitable range before they can be used.

The Random function, as noted earlier, generates the same number sequence from the same 'seed' number. The simulation will not be able to achieve the variance encountered in the real measurement if the program uses the same number sequence. Therefore, the seed must be changed every time the program generates a new image.

Fortunately, Macintosh has a battery-powered clock build inside. The purpose of this clock is to keep track of time even when the machine is turned off. The program can set the variable RandSeed equal to this clock value. The 'seed' number then changes every second and the random numbers are not repeated.

The program calculates a number of dots ( n ) in such a relationship that the number of dots inside the measuring aperture times the area of dot ( a ) is equal to the coverage area ( na ). However, only the dots that appear inside the displaying rectangle are registered. This allows the number of dots to fluctuate.

The density can be calculated from eq. ( 7 ), and the granularity can be calculated from eqs. ( 8 ), ( 12 ), and ( 20 ). The experimental data are listed in table 1.
Table 1.
The density and granularity of the monosized random dot models.

The experimental data in table 1 show the effect of increasing the dot size. The dots area coverage is the same in all the experiments, therefore, the density level is the same. Only the number of dots and the area of dots are different.

It was also shown in table 1 that increasing the dot size results in increasing the granularity. The same conclusion can be reached with the prediction from eqs. (8), (12), and (20); The granularity is a direct proportion to the value of the dot size. Therefore, at the same density level, granularity can be reduced by decreasing the dot size. Figure 15 17 shows the granularity values from three different equations.

5.2 The Simulation of The Grain Size Distribution Model

The most noticeable fact about silver halide emulsion is that it has different grain sizes in a single layer. This is different from the monosized random dot model. The size distribution has the advantages of improving the photographic quality such as speed, contrast, etc.
Most of commercial silver halide emulsions do have a grain size distribution, only in a special types of emulsion that the manufacturers take great care of making the monosized silver halide particles. The monosized emulsion has an undesirable effect of low speed, unsuitable for pictorial photography. But it is used in graphic arts where high resolution and high contrast are important.

The program has to generate dots of different sizes. The position of a dot on the screen still follows a random number as it does in the monosized random dot. The question remains what kinds of dot the program should generate and what the size range it should take.

In this study, the distribution of dot size varies in a narrow range. The simulation contains a dot that is one pixel smaller than the setting value, a dot that is equal to the setting value, and a dot that is one pixel bigger than the setting value. This method is selected because it is more convenient to program. For example, if the selected dot size is four pixels wide, the program generates the dots of size three, four, and five pixels.

The dots are generated separately and the program draws the smallest sized dots first, the positions of dots are at random. The program then draws the middle sized dots, followed by the biggest sized dots.

The number of dots are calculated from the relationship between the area coverage and the size of a dot, as in the monosized random dots model. This number is then divided by three and the value is used for the dots of each size.

After the program generates dots of each size, it calculates the value of density and granularity from eqs. (14) and (16), respectively.

The result of one such experiment is listed in table 2.
average numbers from 10 experiments

<table>
<thead>
<tr>
<th>dot size</th>
<th>&lt; A &gt;</th>
<th>density eq.(14)</th>
<th>granularity eq.(16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>19.19</td>
<td>0.230</td>
<td>3.835</td>
</tr>
<tr>
<td>8</td>
<td>67.373</td>
<td>0.233</td>
<td>13.647</td>
</tr>
<tr>
<td>12</td>
<td>147.78</td>
<td>0.241</td>
<td>30.99</td>
</tr>
</tbody>
</table>

Table 2.
The density and granularity of the multiple-sized dot model.

The data listed in table 2, as compared to the data listed in table 1, shows that the granularity of the multiple-sized random dot is always higher than the granularity of the monosized random dot at the same density level.

A plot of the granularity as a function of dot size is shown in Fig. 18. It is confirmed that the granularity value is a direct proportion to the dot size. The slope of a graph in fig. 18 is higher that the slope of a graph in Fig. 15, which indicates that the granularity of multiple-sized random dots is increasing at a higher rate than the granularity of monosized random dots.

5.3 The Simulation of The Light scattering model

For an investigation of the light-scattering effect on the granularity, a new factor, $Q_{ext}$, is introduced. The $Q_{ext}$ is the extinction coefficient factor of the radiation. The dot size multiplied by this factor gives an effective projection area of a dot. The $Q_{ext}$ value larger than 1.0 increases the dot size, while the $Q_{ext}$ value less than 1.0 decreases the dot size.
The comparison between eqs. (8) and (24) reveals that the granularity of a light scattering model is equal to the granularity of a monosized random dots model multiplied by $Q_{ext}$. Therefore, the program can use the monosized random dots model simulation and multiply the granularity and density by $Q_{ext}$.

The $Q_{ext}$ values of silver halide emulsion are found to be a maximum of 2.5, accounted for two and a half times increasing in projection area of the dot.\(^{(15)}\)

The computer program used in the simulation is the same as the monosized random dot model, only the area of a dot is multiplied by the $Q_{ext}$ value which is supplied by users through a dialog box. The dots that display on the screen are of the same size as the setting value. Although the $Q_{ext}$ effects the projection area of a dot, it is not possible to display a scaling dot on a Macintosh screen.

The result from one such experiment is listed in table 3.

<table>
<thead>
<tr>
<th>dot size</th>
<th>density eq. (23)</th>
<th>granularity eq. (24)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.223</td>
<td>1.559</td>
</tr>
</tbody>
</table>

Table 3.
The density and granularity of the light scattering model.

The data listed in table 3 are obtained from the simulation that set the $Q_{ext}$ value to 1.0, which means that the projection area of the dot is the same as the geometrical area. When comparing these data with the data listed in table 1, it shows that the density and granularity of the light scattering model of $Q_{ext}$ value equal 1.0 is the same as the density and granularity of the monosized random dot model. For example, changing $Q_{ext}$ value by
a factor of 2 equals the double of the dot size. This situation is illustrated by the granularity vs. dot area curves shown in Fig. 19.

Therefore, It is confirmed that the $Q_{\text{ext}}$ value scales the density and granularity of the monosized random dot as if the dot size has been change.
Fig. 15. A comparison of the granularity from eq. (8).

Fig. 16. A comparison of the granularity from eq. (12).
Fig. 17. A comparison of the granularity from eq. (20).

Fig. 18. The comparison between eq. (12) and eq. (20).
Fig. 19. The granularity of the multiple-sized dots.
Dot size = 4 pixels
D = 0.218
G =
1.514   1.740   1.263

Dot size = 8 pixels
D = 0.213
G =
5.916   3.440   2.495

Dot size = 12 pixels
D = 0.215
G =
13.437  5.184   3.76

Fig. 20. A comparison of the monosized random dot at the same density level.
Fig. 21. A comparison of the monosized random dots at different density levels.
The monosized random dots dot size = 4 pixels $D = 0.217$ $G = 1.740$

The multiple-sized random dots dot sizes = 3, 4, and 5 pixels $D = 0.228$ $G = 3.793$

Fig. 22. A comparison between the monosized dots and the multiple-sized dots.
Dot size = 4 pixels
coverage area = 150%

Dot size = 8 pixels
coverage area = 300%

Dot size = 12 pixels
coverage area = 450%

Fig. 23. The Random Dot Images.
Element size = 4 pixels
Threshold gray level = 15

Element size = 8 pixels
Threshold gray level = 15

Element size = 12 pixels
Threshold gray level = 15

Fig. 24. The Ordered-Array Dot Images.
Fig. 25. A Comparison of The Images.
VI. CONCLUSION

6.0 The Models

The study has demonstrated that, in the uniformly distributed random dots, granularity is in direct proportion to the dot size. The models show a good prediction power for the granularity value.

The monosized random dots simulation gives a granularity value that agrees with the theoretical equation. When the dot size increases, the granularity from the simulation will also increase.

The multiple-sized random dots model, at the same density level, gives higher granularity value than the monosized random dots model. This is because the different sized dots when displayed together, add more spatial inhomogeneity to the image and therefore raise the granularity value.

Due to the effect of light scattering, which makes the projection area of the dot bigger than the geometrical area of the dot, the light scattering model is the same as the monosized random dots model that has enlarged dots. If the light scattering coefficient equals 1.0, the formulas in calculating the density and granularity in the light scattering model are reduced to that of the monosized dots.

Eqs. (9), (13), and (21) give the linear relationship between granularity and dot size. But inspection of the eq. (9) revealed that the granularity value is calculated as a function of the dot area, while in the eqs. (13) and (21), the granularity is calculated as a function of the square root of the dot area. In order to compare these three equations, one should take the square root of the answer in eq. (9).
6.1 Similar Projects

There are a number of ideas which one could come up with for projects that are similar to, or an extension of, the program presented in this thesis. For instance, two such ideas are the following:

Even though the Macintosh Toolbox routines are written in Pascal, the programmer does not have to limit himself to that language. Many compilers of different languages are now available on the Mac, such as Forth, Fortran, Modular-2, C, LISP, etc. If the programmer uses another compiler other than Pascal, he should be able to write this program in another language. This can be done by alterations to the existing package. All the source lines would have to be changed and the program skeleton would have to be altered to allow for a new language. But, the basic logic of the program can be kept similar.

Another possibility for a project similar to this one could be an extended version of a program. Apart from the functions which we discussed in our project, the programmer could also design his own windows and menus, add more features to the existing code, or build new modules with specific purposes.

6.2 Personal Observations

There are only few things which I could have done in a different way while writing this program. They seemed to operate in a very satisfactory way as they were, and at the same time they were written in such a way that easily allowed for changes in the way it worked. Perhaps I could have tightened the code so that the program would be smaller, but, in general I would not change them at all.
I have gained a lot of valuable knowledge while doing this project, especially my understanding about the granularity and density models has increased substantially.

I have also gained insight in the way the Macintosh program works. I hope that my program can be of use in the Imaging Science and Computer Science programs, either for general purposes or as an example to similar programming problems that were encountered.
LIST OF REFERENCES

2. ibid., pp 817.
7. Ref. 3., pp 58 - 61.
Apiwat Nutavej was born in Bangkok, Thailand on July 31, 1960. He is a citizen of Thailand. He attended the Rochester Institute of Technology from 1984 to 1987 and received the Master of Science degree in 1987.