Artificial Eye

New Mexico Supercomputing Challenge
Final Report
April 4, 2007

Team # 010
Albuquerque Academy

Team Members:

Nick Longenbaugh

Wesley Smalls

Sponsoring Teacher/Project Mentor:

Jim Mims

Table of Contents

Cover Page.....Page 1

Table of Contents.....Page 2

Executive Summary.....Page 3

Problem Statement.....Page 4

Description of Major Methods.....Page 5

Description of Major Routines.....Page 9

Results.....Page 12

Conclusions.....Page 15

Original Achievements.....Page 16

Acknowledgements.....Page 17

Citations and References.....Page 18

Appendices.....Page 19

- A) Simplified Flow Chart.....Page 20
- B) CConnection Code.....Page 21
- C) CSGridCell Code.....Page 23
- D) Beginning Code.....Page 28
- E) Background Code.....Page 30
- F) Filtering Code.....Page 31
- G) Smoothing Code.....Page 32
- H Scanning Code.....Page 36
- I) Analysis Code.....Page 40
- J) Form1 Code.....Page 42
- K) Settings Code.....Page 46

Executive Summary

Our team's goal for the 2006/2007 Supercomputing Challenge was to create a program that could successfully model the human eye. To do this, the program would need the ability to break down images given to it and identify objects within the images. After researching various methods used in the past, our team decided to focus on the two most important aspects that we found: boundaries and color.

Using embossing and engraving with respect to the left (arbitrary), the program finds differences between neighboring pixels. By paying attention to only the largest changes, it is able to determine the boundary between two color fields and filter out background noise. Using this technique, the program then uses one of two options to simplify the data into virtual lines: It either uses a swarm class to "trace" the boundaries, or it uses an original technique called "shrink wrapping" in order to find the outlines of objects in the picture.

The next step is to connect the lines and form polygons. Using these polygons as references, it then generates color field data. Using the lines, polygons and color fields, the program is then able to utilize "Template Matching" in order to identify objects within the picture.

Unfortunately, the final step that our team achieved in the development process was the generation of the boundary lines to be used in polygon generation. We simply could not make the jump from simple line data to a coherent list of relevant polygons, which was the next step in deciphering the picture. Our difficulties most likely arose from our approach to the problem, which was significantly different from previous attempts. Had more similarities between the techniques we used and the ones used in the past been present, we likely would have made the jump and created a successfully executing program.

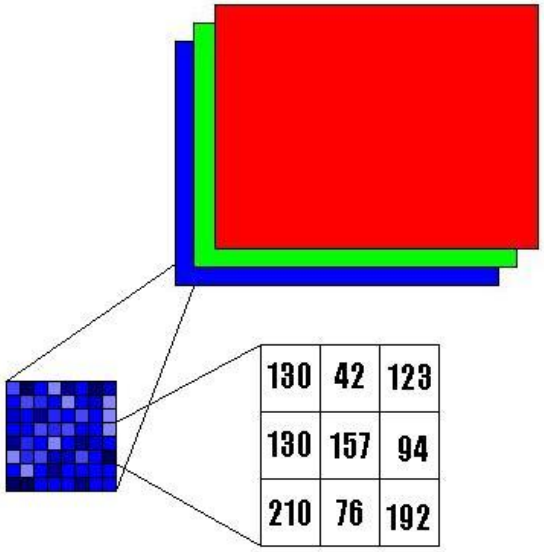
Problem Statement

The goal of our project was to create a program capable of mimicking the human eye. In order to do this, the program needed to collect data from an image that pertained to specific objects within the image. It needed to locate and address items including boundaries, shapes, color fields, and possibly textures. Also, for a fully developed understanding of the image to be obtained, the program would be required to take into account the image's context. This implied the use of a video stream and/or the usage of 3-Dimensional vision techniques.

Description of Major Methods

In order for the program to accurately mimic the human eye, it follows several simple steps: 1) break the image down into its basic elements, 2) filter out any “background noise”, 3) determine boundaries between color fields, 4) generate polygons and color fields from the boundaries, 5) use template matching to come up with the closest match in order to determine what objects are within the image.

In order to break an image down into its basic elements, a basic knowledge of computer images was required. A basic computer image, known as a bitmap, consists of a two dimensional array of pixels. In turn, each of the pixels in the array can be thought of as a stack of three dots. Each of these dots contains data about one of three colors: Red, Green, and Blue. The data is made up of a number which describes the amount of color in the dot. This number ranges from 0 (no color), to 255 (full color). In another sense, a bitmap can be thought of as three distinct planes of pixels, each plane corresponding to Red, Green or Blue.



The first step in deciphering an image is to break it up into its three color planes. This technique is called “Picture Parsing”. In order to do this, a pixel by pixel operation is performed that extracts amounts of Red, Green and Blue in a pixel and saves the data in three identical arrays that represent the three color planes of the

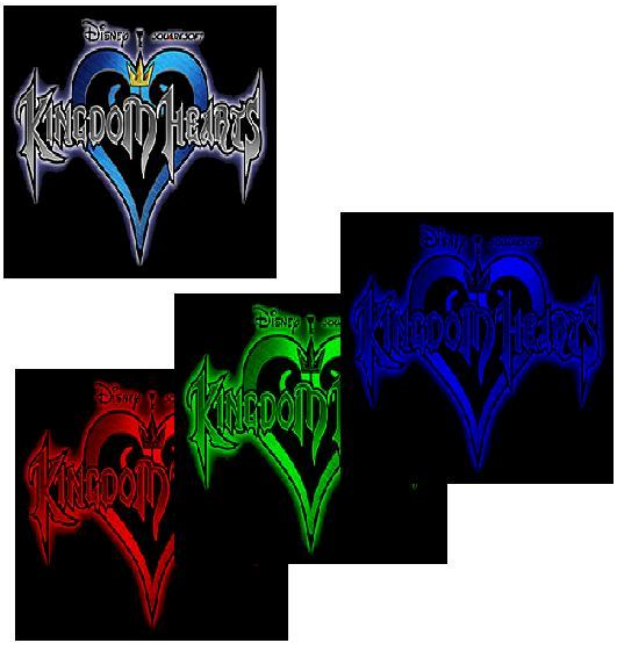
original image. This operation is given below. For convenience, the pixel value will be “Pixel”.

Red = Pixel And &HFF

Green = ((Pixel And &HFF00) / &H100) Mod &H100

Blue = ((Pixel And &HFF0000) / &H10000) Mod &H100

After the three color planes have been defined, a grayscale plane is generated by averaging each pixel in the other three planes.



The second and third steps in deciphering an image involve “smoothing” the image, or filtering out the “background noise”. Background Noise is any unimportant undulation or change in the coloring of an image. To do this, two very simple techniques called Embossing and Engraving are used. The only difference between the two is how the result appears. When viewed, embossing appears to cause the image to “pop out” of the picture, where as engraving appears to cause the image to “sink into” the picture. The process of both embossing and engraving involves nothing more than comparing a pixel to its neighbor and then assigning the pixel the value of their difference. By using a threshold, such as 50%, any insignificant changes, or background noise, can be filtered out, while saving the larger changes to be thought of as boundaries.

The second part of the third step is defining the definite boundaries between color fields in the image. To do this, one of two techniques is utilized. The first is to simply trace the boundaries using a swarm class, and using the data to generate virtual lines. The swarm class starts as "Seeds" which are placed on the boundaries of the image in a grid pattern. The seeds quickly develop into "worms" that continue to trace their boundary until they run into a grid line or hit a dead end. In the case of running into a grid line, the worm will terminate and create a new seed at its terminal point. Unfortunately, though this technique is very accurate, it is inefficient and consumes huge amounts of memory and processor power.

The second technique in the third step is an original technique designed specifically for use in this project: "Shrink-wrapping". It is a rather complicated technique that splits the image into virtual "fields" and in each field "wraps" any borders and/or objects in their entirety. In doing so, each boundary in a field is quickly estimated and fused with those nearest it. This technique, while considerably faster to execute and far less resource intensive than the Seeding method, is also far less accurate.

The fourth step in deciphering an image is to take the boundary data collected so far and generate simple polygons from it. Using these polygons, color fields can be extrapolated by averaging the color values of the pixels within the polygons.

The final step in the process of deciphering an image is to compile all of the data collected thus far and use it in a database search. This process, known as

“Template Matching”, uses the information in the database, which pertains to known objects, and compares it to the data gathered from the image. When matches (or near-matches) to the collected data are found, the object has been successfully identified. When all, or close to all, of the data in the image has been accounted for, the picture is deemed deciphered. However, in the case that no template matches or is acceptably close to the collected data, the program asks for a name to associate with the data and saves the data as a new template for future use.

Unfortunately, everything aforementioned beyond the third step is simply speculation. Our team never managed to move beyond the third step in the development stage of our project. We believe our problem stemmed from the fact that there were so many differences between our chosen methods and those that have worked successfully in the past. When we finally faced a problem that we couldn’t answer, we had no where to turn so to that would provide us with an answer.

Description of Major Routines

The CConnection Class

Almost all of the Routines in the CConnection class are trivial, and so will not be mentioned here. However, The FindNextPoint routine is very important. In fact it is the driving force behind the Seeding method of boundary recognition in our project.

The FindNextPoint routine acts as the "eyes" for the seeds/worms of the CConnection class. It decides where, if anywhere, the seed/worm will move next. To do this, it decides what points directly surrounding it 1) are parts of a boundary, and 2) have not been scanned by another seed/worm. The first clean (both of the aforementioned Booleans are true) point that the routine finds is automatically considered the seed/worm's next target, and the placement variables (its X and Y coordinates) are set to those of the new point.

If none of the seed/worm's surrounding points are clean, it terminates. That is to say that it records its current (X,Y) coordinate pair as the ending point for a line segment, and sets its starting (X,Y) coordinate pair as the starting point for the same line segment. It then saves this line segment in a collection that stores it until the next step (polygon generation).

In the event that the seed/worm moves onto one of the lines of the seeding grid, it immediately terminates and creates a new seed to take over for it. In this way, the class creates boundaries that match the image's boundaries much more exactly.

The CSGridCell Class.

The name CSGridCell stands for Shrink-Grid Cell. Appropriately, members of this class are nothing but virtual cells of the grid created by the SetupSqueezeGrid routine in the Scanning module. Each cell is comprised of a 5x5 grid (arbitrary units), with each side of the grid containing 5 equally spaced points. These points slowly move toward the center until they find a boundary mark. If a point makes it all the

way to the center, it is disregarded. Once all of the points have ceased moving, the cell creates members of the CConnection class that connect the points sequentially, forming an outline of whatever is within the cell's grid.

The Squeeze routine is called to incrementally move the points toward the center of the cell's grid. It moves all twenty points, one pixel at a time. When a point hits a boundary mark, it sets a Boolean tied to it to equal false, and it is no longer moved. Finally, Squeeze checks to see if any of the points have reached the center of the cell's grid, if they have, it sets the Boolean tied to it to equal false.

The second and final noteworthy routine in the CSGridCell class is the CreateOutline routine. This routine sets up an array and moves sequentially through the points, finding the ones whose Boolean tie indicates they are still active. If they are, it adds them to the array. From this array, CreateOutline then creates a series of pre-terminated Worms (members of the CConnection class) that connect the active points, thereby creating the "Shrink-Wrapped" outline of the contents of the cell.

The Filtering .bas Module

The filtering .bas module's single noteworthy routine is nonetheless at the basis of the entire program's operation. The FilterPicture routine uses a pixel by pixel operation that splits the image into 5 separate color planes: Red, Green, Blue, Grayscale, and Black and White. All of the color planes are then loaded into identical arrays, and from then on are treated as separate pictures.

The Smoothing .bas Module

There are five major routines in the Smoothing module: Smooth, SmoothUpLeft, SmoothUpRight, SmoothDownLeft, SmoothDownRight.

Smooth, the main routine in this module, acts mainly as a crossroads, deciding which of the other four large routines to call. However, after it has called a combination of the other four routines, it applies error compensation in the form of correcting the solid lines around the edges of the image formed as a side effect of the four smoothing routines.

SmoothUpLeft is an embossing routine. Contained within it are two nested For Loops that work their way through the pixel data of the image from bottom right to upper left. For each pixel, the routine compares it with the neighboring pixel on its upper left side. It then plots the difference between the two pixels as a grayscale value in another array. It does this for all five of the color planes.

Each of the other three routines is either an embossing routine or an engraving routine. The names of the routines tell in which direction they are justified.

The Scanning .bas Module

The Scanning module contains quite a few major routines: LaySeeds, SetupSqueezeGrid, ScanningLoop, and several routines for creating the members of the CConnection, CSGridCell, and other classes.

The LaySeeds routine is the starting routine for the seeding method of finding boundaries. It is a fairly simple routine, yet is immensely important. It simply calculates where grid lines should be placed on the image. The number of lines on both the horizontal and vertical dimensions is defined by the user. Then, LaySeeds scans each pixel underneath the grid lines (in all of the color planes), and if it is a boundary mar, it lays a seed at that location. The grid method was chosen from trial and error after attempting to use many more approaches. It is very efficient in that it misses few, if any, of the image's details when set to a decently high level of detail.

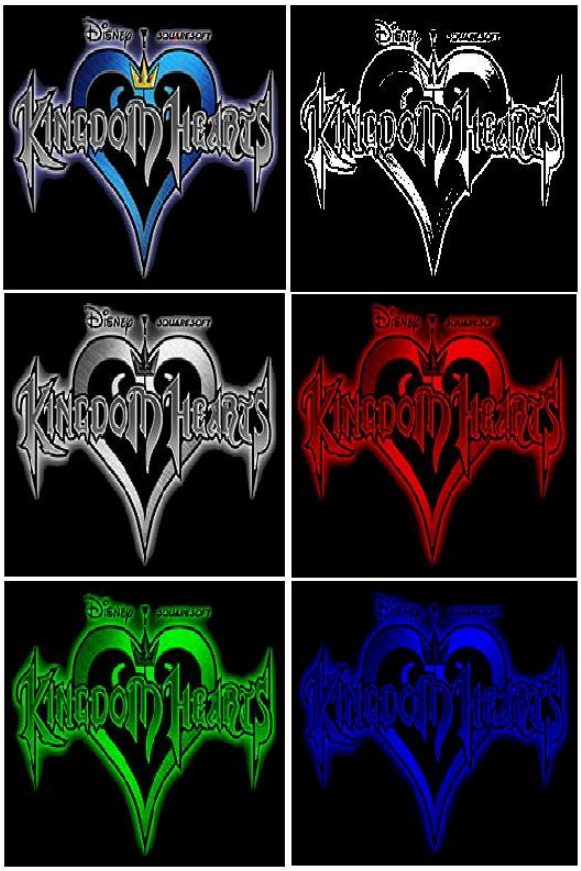
The SetupSqueezeGrid routine is the starting routine for the Shrink Wrap method of finding the boundaries. SetupSqueezeGrid, much like LaySeeds, simply defines the lines for a grid. Or, more accurately, it creates the individual cells of the grid. The number of cells on each side of the image (the number of cells to be squared) is set by the user, and the cells are then created sequentially.

ScanningLoop is arguably the most important routine in the program. It is comprised of a single loop that terminates when no further growing (Seeds/Worms) or shrinking (Shrink Wrap) can occur.

Results

Our results were fairly clear. Because we benchmarked our code, we were able to obtain results from all of the stages we completed throughout the entire program.

Step 1: Filtering

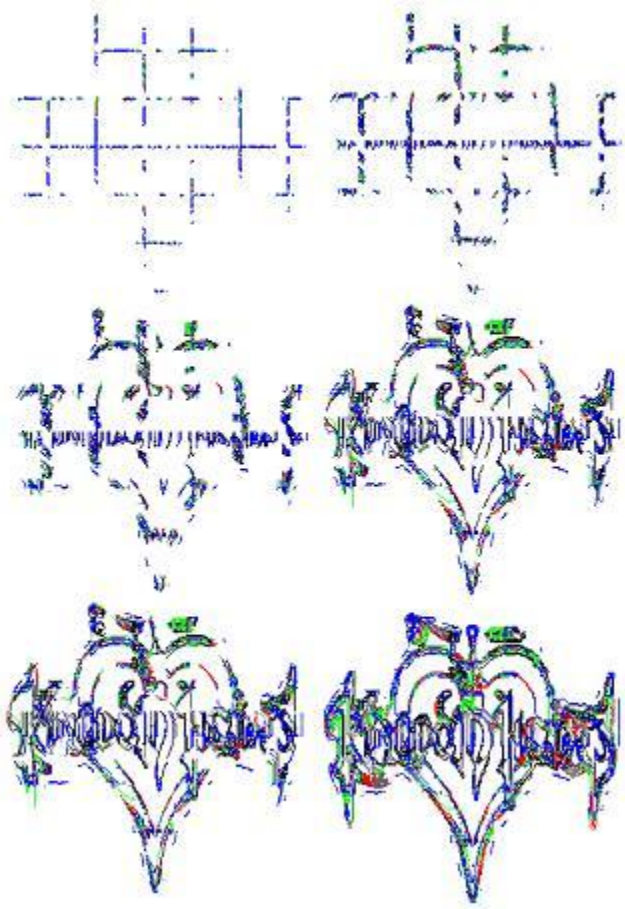


A picture of an image (top left), the three main color planes that make it up, the Grayscale color plane, and the Black and White color plane.

Step 2: Smoothing

A picture of the same image, but with the background noise filtered out and the borders between color fields accentuated.

Step3: Scanning



A time-lapse picture of the smoothed version of the image with the original seeding grid, and the Worms that developed from it.

Conclusions

Based on the data collected during the development and testing phase, I can draw the conclusion that although the project was not able to be completed, there is no reason that methods chosen are to blame; the project could very well have succeeded given more time. Upon closer inspection, I also came to the conclusion that the Shrink Wrapping method, under certain conditions, becomes unstable and unreliable. Also, after further testing of the Seeding method, we found that it is far more effective when used in a smaller, localized setting, with a limit placed on the seeds' propagation. In this way, we can confidently conclude that a blending of the principles behind the two scanning methods we used would be by far the most effective.

Original Achievements

By far, the most rewarding achievements of our team during the Challenge were the two scanning methods that we developed. Because they are both different from methods seen in the past, the fact that they worked (at least satisfactorily) shows that they could definitely be used in future routines and methods for modeling the human eye.

Acknowledgements

We would like to thank Jim Mims, our teacher and project mentor, for all of his time and effort. Although we chose a topic to study and work in that was leagues ahead of us, he never once tried to stop us, but rather simply encouraged us to do the best that we could. In the end, although we didn't manage to complete our project, he has taught us many new ways to approach problems and challenges that will undoubtedly help us to conquer future wall even taller than those we face now.

Finally, we would like to thank all of our many friends here at Albuquerque Academy and elsewhere that supported and encouraged us throughout the entire process. We would never have made it this far without them.

Citations and References

Books and Written Sources:

ARTIFICIAL INTELLIGENCE
a revision of COMPUTERS THAT THINK?
Margaret O. Hyde

Artificial Intelligence
UNDERSTANDING COMPUTERS
The Editors of TIME-LIFE BOOKS

BRAIN MAKERS
David H. Freedman

MIND MATTERS
James P. Hogan

THE LOGIC OF INDUCTION
Halina Mortimer

Microsoft Visual Basic 5
The Comprehensive Guide
Richard Mansfield

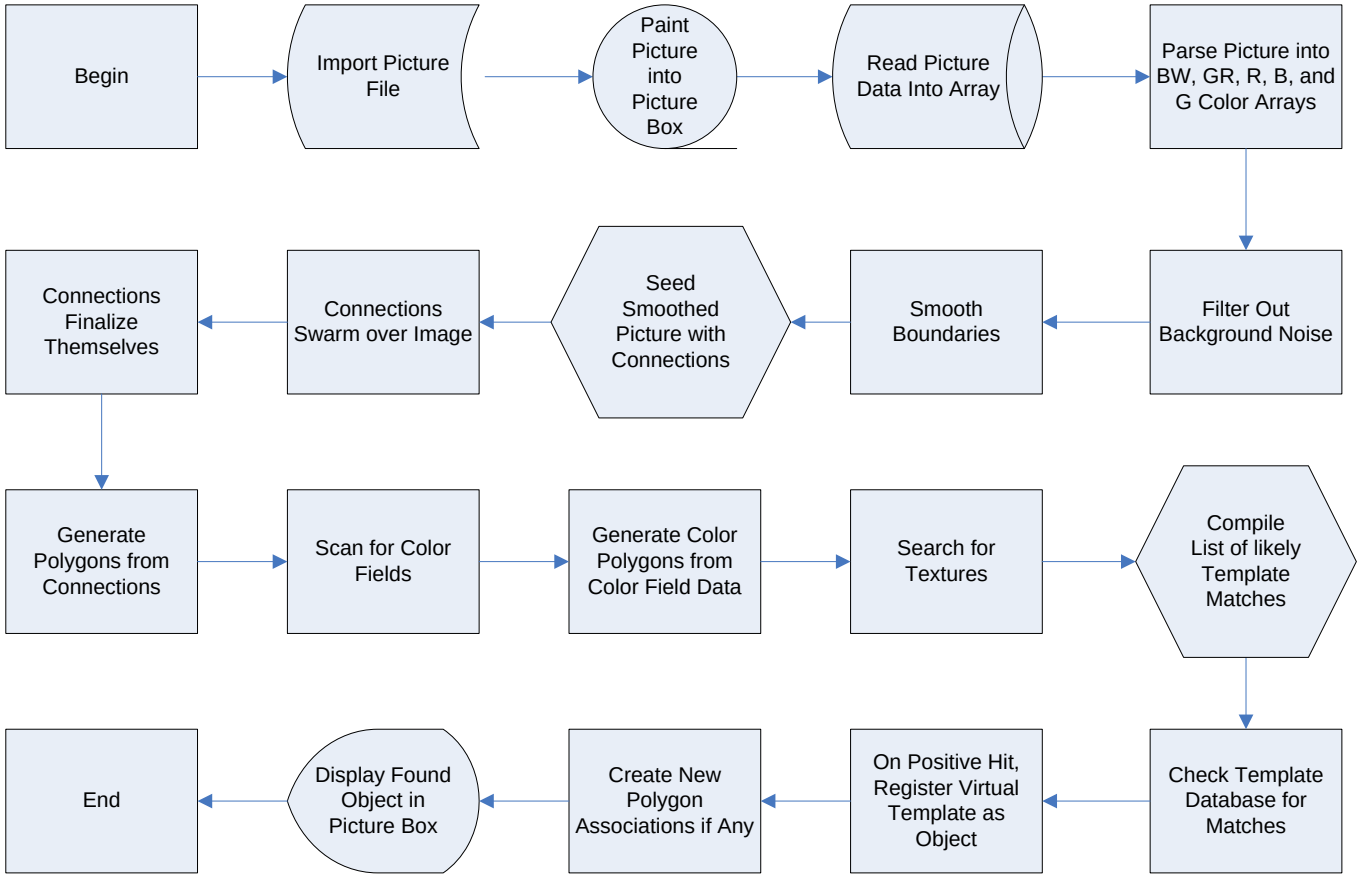
Microsoft Visual Basic 6.0
Programmer's Guide
Microsoft Corporation

Websites:

Appendices

Appendix A

Simplified Flow Chart



Appendix B
CConnection Code

```
'Code written in Microsoft Visual Basic 6.0, line comment character is '

'Property and Global variable Dimensioning

Public SX As Integer, SY As Integer 'Start X and Y
Public CX As Integer, CY As Integer 'Current X and Y
Public PX As Integer, PY As Integer 'Previous X and Y
Public EX As Integer, EY As Integer 'End X and Y
Public Following As Boolean
Public Color As String
Public ColorCode As Long
Public Remove As Boolean
Public Complete As Boolean


Public Sub Follow()
Call FindNextPoint
If (CX = SX And CY = SY) Or (EX = SX And EY = SY) Then
   Complete = True
   Remove = True
Else:
   If CX / Scanning.Detail = Round(CX / Scanning.Detail) Or CY / Scanning.Detail = Round(CY / Scanning.Detail) Then
      Complete = True
      Call EndConnection
      Call Scanning.CreateNewCorner(CX, CY, Color, ColorCode)
      Call Scanning.CreateNewConnection(CX, CY, Color, ColorCode)
   End If
End If
End Sub

Public Sub FindNextPoint()
Dim i As Integer, e As Integer, X As Integer, Y As Integer
Dim Good As Boolean
Good = False
i = CX - 1
For i = CX - 1 To CX + 1
   If i <= 0 Then i = 1
   If i > Background.PWidth Then Exit For
   e = CY - 1
   For e = CY - 1 To CY + 1
      If e <= 0 Then e = 1
      If e > Background.PHeight Then Exit For
      If i <> CX Or e <> CY Then
         Good2 = False
         Select Case Color
            Case "Full"
               If Background.Picture(i, e, 3) = 1 And Background.Picture(i, e, 4) = 0 Then
                  Background.Picture(i, e, 4) = 1
                  Good = True
               End If
            Case "Black"
               If Background.PictureBW(i, e, 2) And Background.PictureBW(i, e, 3) = False Then
                  Background.PictureBW(i, e, 3) = True
                  Good = True
               End If
            Case "Grey"
               If Background.PictureGR(i, e, 3) = 1 And Background.PictureGR(i, e, 4) = 0 Then
                  Background.PictureGR(i, e, 4) = 1
                  Good = True
               End If
            Case "Red"
               If Background.PictureR(i, e, 3) = 1 And Background.PictureR(i, e, 4) = 0 Then
                  Background.PictureR(i, e, 4) = 1
                  Good = True
               End If
```

```
            Case "Green"
               If Background.PictureG(i, e, 3) = 1 And Background.PictureG(i, e, 4) = 0 Then
                  Background.PictureG(i, e, 4) = 1
                  Good = True
               End If
            Case "Blue"
               If Background.PictureB(i, e, 3) = 1 And Background.PictureB(i, e, 4) = 0 Then
                  Background.PictureB(i, e, 4) = 1
                  Good = True
               End If
            Case "Simple"
               If Background.PictureS(i, e, 1) = 1 And Background.PictureS(i, e, 2) = 0 Then
                  Background.PictureS(i, e, 2) = 1
                  Good = True
               End If
         End Select
      End If
      If Good Then Exit For
   Next e
   If Good Then Exit For
Next i
If Good Then
   CX = i
   CY = e
Else:
   Call EndConnection
   Complete = True
   Remove = True
End If
End Sub

Public Sub EndConnection()
EX = CX
EY = CY
Following = False
End Sub

Public Sub DrawCurrent()
DoEvents
Form1.P1.PSet (CX, CY), ColorCode
End Sub

Public Sub EraseCurrent()
Form1.P1.Line (SX, SY)-(CX, CY), vbWhite
Form1.P1.PSet (CX, CY), ColorCode
End Sub

Public Sub DrawTotal()
Form1.P1.Line (SX, SY)-(CX, CY), ColorCode
End Sub
```

Appendix C
CSGridCell Code

```
'Property and Global variable Dimensioning

Public X As Integer, Y As Integer
Public Height As Integer, Width As Integer
Public Number As Integer
Public L1 As Integer, L2 As Integer, L3 As Integer, L4 As Integer, L5 As Integer
Public R1 As Integer, R2 As Integer, R3 As Integer, R4 As Integer, R5 As Integer
Public T1 As Integer, T2 As Integer, T3 As Integer, T4 As Integer, T5 As Integer
Public B1 As Integer, B2 As Integer, B3 As Integer, B4 As Integer, B5 As Integer

Public L1A As Boolean, L2A As Boolean, L3A As Boolean, L4A As Boolean, L5A As Boolean
Public R1A As Boolean, R2A As Boolean, R3A As Boolean, R4A As Boolean, R5A As Boolean
Public T1A As Boolean, T2A As Boolean, T3A As Boolean, T4A As Boolean, T5A As Boolean
Public B1A As Boolean, B2A As Boolean, B3A As Boolean, B4A As Boolean, B5A As Boolean

Public IW As Single, IH As Single
Public Squeezed As Boolean


Public Sub Squeeze()
Squeezed = True
If L1 < Int(X + (2 * IW)) And Background.PictureS(L1, Y, 1) = 0 Then
   L1 = L1 + 1
   Squeezed = False
End If
If L2 < Int(X + (2 * IW)) And Background.PictureS(L2, Int(Y + IW), 1) = 0 Then
   L2 = L2 + 1
   Squeezed = False
End If
If L3 < Int(X + (2 * IW)) And Background.PictureS(L3, Int(Y + (IW * 2)), 1) = 0 Then
   L3 = L3 + 1
   Squeezed = False
End If
If L4 < Int(X + (2 * IW)) And Background.PictureS(L4, Int(Y + (IW * 3)), 1) = 0 Then
   L4 = L4 + 1
   Squeezed = False
End If
If L5 < Int(X + (2 * IW)) And Background.PictureS(L5, Int(Y + (IW * 4)), 1) = 0 Then
   L5 = L5 + 1
   Squeezed = False
End If
If (X + (IW * 4)) + R1 > Int(X + (2 * IW)) And Background.PictureS((X + (IW * 4)) + R1, Y, 1) = 0 Then
   R1 = R1 - 1
   Squeezed = False
End If
If (X + (IW * 4)) + R2 > Int(X + (2 * IW)) And Background.PictureS((X + (IW * 4)) + R2, Int(Y + IW), 1) = 0 Then
   R2 = R2 - 1
   Squeezed = False
End If
If (X + (IW * 4)) + R3 > Int(X + (2 * IW)) And Background.PictureS((X + (IW * 4)) + R3, Int(Y + (IW * 2)), 1) = 0 Then
   R3 = R3 - 1
   Squeezed = False
End If
If (X + (IW * 4)) + R4 > Int(X + (2 * IW)) And Background.PictureS((X + (IW * 4)) + R4, Int(Y + (IW * 3)), 1) = 0 Then
   R4 = R4 - 1
   Squeezed = False
End If
If (X + (IW * 4)) + R5 > Int(X + (2 * IW)) And Background.PictureS((X + (IW * 4)) + R5, Int(Y + (IW * 4)), 1) = 0 Then
   R5 = R5 - 1
   Squeezed = False
End If
If T1 < Int(Y + (2 * IH)) And Background.PictureS(X, T1, 1) = 0 Then
   T1 = T1 + 1
   Squeezed = False
End If
```

```
If T2 < Int(Y + (2 * IH)) And Background.PictureS(Int(X + IH), T2, 1) = 0 Then
   T2 = T2 + 1
   Squeezed = False
End If
If T3 < Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IH * 2)), T3, 1) = 0 Then
   T3 = T3 + 1
   Squeezed = False
End If
If T4 < Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IH * 3)), T4, 1) = 0 Then
   T4 = T4 + 1
   Squeezed = False
End If
If T5 < Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IH * 4)), T5, 1) = 0 Then
   T5 = T5 + 1
   Squeezed = False
End If
If (Y + (IH * 4)) + B1 > Int(Y + (2 * IH)) And Background.PictureS(X, (Y + (IH * 4)) + B1, 1) = 0 Then
   B1 = B1 - 1
   Squeezed = False
End If
If (Y + (IH * 4)) + B2 > Int(Y + (2 * IH)) And Background.PictureS(Int(X + IH), (Y + (IH * 4)) + B2, 1) = 0 Then
   B2 = B2 - 1
   Squeezed = False
End If
If (Y + (IH * 4)) + B3 > Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IH * 2)), (Y + (IH * 4)) + B3, 1) = 0 Then
   B3 = B3 - 1
   Squeezed = False
End If
If (Y + (IH * 4)) + B4 > Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IH * 3)), (Y + (IH * 4)) + B4, 1) = 0 Then
   B4 = B4 - 1
   Squeezed = False
End If
If (Y + (IH * 4)) + B5 > Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IH * 4)), (Y + (IH * 4)) + B5, 1) = 0 Then
   B5 = B5 - 1
   Squeezed = False
End If
If L1 >= Int(X + (2 * IW)) And Background.PictureS(L1, Y, 1) = 1 Then L1A = False
If L2 >= Int(X + (2 * IW)) And Background.PictureS(L2, Int(Y + IH), 1) = 1 Then L2A = False
If L3 >= Int(X + (2 * IW)) And Background.PictureS(L3, Int(Y + (IH * 2)), 1) = 1 Then L3A = False
If L4 >= Int(X + (2 * IW)) And Background.PictureS(L4, Int(Y + (IH * 3)), 1) = 1 Then L4A = False
If L5 >= Int(X + (2 * IW)) And Background.PictureS(L5, Int(Y + (IH * 4)), 1) = 1 Then L5A = False
If (X + (IW * 4)) + R1 <= Int(X + (2 * IW)) And Background.PictureS((X + (IH * 4)) + R1, Y, 1) = 1 Then R1A = False
If (X + (IW * 4)) + R2 <= Int(X + (2 * IW)) And Background.PictureS((X + (IH * 4)) + R2, Int(Y + IH), 1) = 1 Then R2A = False
If (X + (IW * 4)) + R3 <= Int(X + (2 * IW)) And Background.PictureS((X + (IH * 4)) + R3, Int(Y + (IH * 2)), 1) = 1 Then R3A =
False
If (X + (IW * 4)) + R4 <= Int(X + (2 * IW)) And Background.PictureS((X + (IH * 4)) + R4, Int(Y + (IH * 3)), 1) = 1 Then R4A =
False
If (X + (IW * 4)) + R5 <= Int(X + (2 * IW)) And Background.PictureS((X + (IH * 4)) + R5, Int(Y + (IH * 4)), 1) = 1 Then R5A =
False
If T1 >= Int(Y + (2 * IH)) And Background.PictureS(X, T1, 1) = 1 Then T1A = False
If T2 >= Int(Y + (2 * IH)) And Background.PictureS(Int(X + IW), T2, 1) = 1 Then T2A = False
If T3 >= Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IW * 2)), T3, 1) = 1 Then T3A = False
If T4 >= Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IW * 3)), T4, 1) = 1 Then T4A = False
If T5 >= Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IW * 4)), T5, 1) = 1 Then T5A = False
If (Y + (IH * 4)) + B1 <= Int(Y + (2 * IH)) And Background.PictureS(X, (Y + (IW * 4)) + B1, 1) = 1 Then B1A = False
If (Y + (IH * 4)) + B2 <= Int(Y + (2 * IH)) And Background.PictureS(Int(X + IW), (Y + (IH * 4)) + B2, 1) = 1 Then B2A = False
If (Y + (IH * 4)) + B3 <= Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IW * 2)), (Y + (IH * 4)) + B3, 1) = 1 Then B3A = False
If (Y + (IH * 4)) + B4 <= Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IW * 3)), (Y + (IH * 4)) + B4, 1) = 1 Then B4A = False
If (Y + (IH * 4)) + B5 <= Int(Y + (2 * IH)) And Background.PictureS(Int(X + (IW * 4)), (Y + (IH * 4)) + B5, 1) = 1 Then B5A = False
If Squeezed Then Call CreateOutlines
End Sub

Public Sub CreateOutlines()
Dim Order() As Integer
Dim X1 As Integer, Y1 As Integer, X2 As Integer, Y2 As Integer
Dim n As Integer, n2 As Integer, i As Integer
If L1A Then n = n + 1
If L2A Then n = n + 1
If L3A Then n = n + 1
If L4A Then n = n + 1
If L5A Then n = n + 1
```

```
If R1A Then n = n + 1
If R2A Then n = n + 1
If R3A Then n = n + 1
If R4A Then n = n + 1
If R5A Then n = n + 1
If T1A Then n = n + 1
If T2A Then n = n + 1
If T3A Then n = n + 1
If T4A Then n = n + 1
If T5A Then n = n + 1
If B1A Then n = n + 1
If B2A Then n = n + 1
If B3A Then n = n + 1
If B4A Then n = n + 1
If B5A Then n = n + 1
ReDim Order(n, 2) As Integer


If T1A Then
   n2 = n2 + 1
   Order(n2, 1) = X
   Order(n2, 2) = Y + T1
End If
If T2A Then
   n2 = n2 + 1
   Order(n2, 1) = X + IW
   Order(n2, 2) = Y + T2
End If
If T3A Then
   n2 = n2 + 1
   Order(n2, 1) = X + (IW * 2)
   Order(n2, 2) = Y + T3
End If
If T4A Then
   n2 = n2 + 1
   Order(n2, 1) = X + (IW * 3)
   Order(n2, 2) = Y + T4
End If
If T5A Then
   n2 = n2 + 1
   Order(n2, 1) = X + (IW * 4)
   Order(n2, 2) = Y + T5
End If

If R1A Then
   n2 = n2 + 1
   Order(n2, 1) = X + Width + R1
   Order(n2, 2) = Y
End If
If R2A Then
   n2 = n2 + 1
   Order(n2, 1) = X + Width + R2
   Order(n2, 2) = Y + IH
End If
If R3A Then
   n2 = n2 + 1
   Order(n2, 1) = X + Width + R3
   Order(n2, 2) = Y + (IH * 2)
End If
If R4A Then
   n2 = n2 + 1
   Order(n2, 1) = X + Width + R4
   Order(n2, 2) = Y + (IH * 3)
End If
If R5A Then
   n2 = n2 + 1
   Order(n2, 1) = X + Width + R5
   Order(n2, 2) = Y + (IH * 4)
End If

If B5A Then
```

```
    n2 = n2 + 1
    Order(n2, 1) = X + (IW * 4)
    Order(n2, 2) = Y + Height + B5
End If
If B4A Then
    n2 = n2 + 1
    Order(n2, 1) = X + (IW * 3)
    Order(n2, 2) = Y + Height + B4
End If
If B3A Then
    n2 = n2 + 1
    Order(n2, 1) = X + (IW * 2)
    Order(n2, 2) = Y + Height + B3
End If
If B2A Then
    n2 = n2 + 1
    Order(n2, 1) = X + IW
    Order(n2, 2) = Y + Height + B2
End If
If B1A Then
    n2 = n2 + 1
    Order(n2, 1) = X
    Order(n2, 2) = Y + Height + B1
End If

If L5A Then
    n2 = n2 + 1
    Order(n2, 1) = X + L5
    Order(n2, 2) = Y + (IH * 4)
End If
If L4A Then
    n2 = n2 + 1
    Order(n2, 1) = X + L4
    Order(n2, 2) = Y + (IH * 3)
End If
If L3A Then
    n2 = n2 + 1
    Order(n2, 1) = X + L3
    Order(n2, 2) = Y + (IH * 2)
End If
If L2A Then
    n2 = n2 + 1
    Order(n2, 1) = X + L2
    Order(n2, 2) = Y + IH
End If
If L1A Then
    n2 = n2 + 1
    Order(n2, 1) = X + L1
    Order(n2, 2) = Y
End If



X1 = Order(1, 1)
Y1 = Order(1, 2)
For i = 1 To n
    X2 = X1
    Y2 = Y1
    X1 = Order(i, 1)
    Y1 = Order(i, 2)
    Call ConnectionCreation(X1, Y1, X2, Y2)
Next i
X2 = X1
Y2 = Y1
X1 = Order(1, 1)
Y1 = Order(1, 2)
Call ConnectionCreation(X1, Y1, X2, Y2)
End Sub

Public Sub CreatePolygons()
```

```
End Sub

Public Sub ConnectionCreation(SX As Integer, SY As Integer, CX As Integer, CY As Integer)
Dim C1 As New CConnection
C1.SX = SX
C1.SY = SY
C1.CX = CX
C1.CY = CY
C1.ColorCode = RGB(128, 128, 128)
C1.Complete = True
CompleteConnections.Add C1
End Sub
```

Appendix D
Beginning Code

```
'Property and Global variable Dimensioning


Public Sub ClearPictureBuffers()
DoEvents
ReDim Background.Picture(PWidth, PHeight, 4) As Long
ReDim Background.PictureBW(PWidth, PHeight, 3) As Boolean
ReDim Background.PictureGR(PWidth, PHeight, 4) As Integer
ReDim Background.PictureR(PWidth, PHeight, 4) As Integer
ReDim Background.PictureG(PWidth, PHeight, 4) As Integer
ReDim Background.PictureB(PWidth, PHeight, 4) As Integer
ReDim Background.PictureS(PWidth, PHeight, 4) As Integer
Call ClearMainPictureBuffers
Call ClearSmoothedBuffers
Call ClearSimplifiedBuffers
Call ClearScanningPictureBuffers
End Sub

Public Sub ClearMainPictureBuffers()
Dim i As Integer, e As Integer
For i = 1 To PWidth
   e = 1
   For e = 1 To PHeight
      Background.Picture(i, e, 1) = 0
      Background.PictureBW(i, e, 1) = False
      Background.PictureGR(i, e, 1) = 0
      Background.PictureR(i, e, 1) = 0
      Background.PictureG(i, e, 1) = 0
      Background.PictureB(i, e, 1) = 0
   Next e
Next i
End Sub

Public Sub ClearSmoothedBuffers()
Dim i As Integer, e As Integer
For i = 1 To PWidth
   e = 1
   For e = 1 To PHeight
      Background.Picture(i, e, 2) = 0
      Background.PictureBW(i, e, 2) = False
      Background.PictureGR(i, e, 2) = 0
      Background.PictureR(i, e, 2) = 0
      Background.PictureG(i, e, 2) = 0
      Background.PictureB(i, e, 2) = 0
   Next e
Next i
End Sub

Public Sub ClearSimplifiedBuffers()
Dim i As Integer, e As Integer
For i = 1 To PWidth
   e = 1
   For e = 1 To PHeight
      Background.Picture(i, e, 3) = 0
      Background.PictureGR(i, e, 3) = 0
      Background.PictureR(i, e, 3) = 0
      Background.PictureG(i, e, 3) = 0
      Background.PictureB(i, e, 3) = 0
      Background.PictureS(i, e, 1) = 0
   Next e
Next i
End Sub
```

```
Public Sub ClearScanningPictureBuffers()
Dim i As Integer, e As Integer
For i = 1 To PWidth
   e = 1
   For e = 1 To PHeight
      Background.Picture(i, e, 4) = 0
      Background.PictureBW(i, e, 3) = False
      Background.PictureGR(i, e, 4) = 0
      Background.PictureR(i, e, 4) = 0
      Background.PictureG(i, e, 4) = 0
      Background.PictureB(i, e, 4) = 0
   Next e
Next i
End Sub


Public Sub LoadNewPicture()
Dim i As Integer, e As Integer
DoEvents
For i = 1 To PWidth
   e = 1
   For e = 1 To PHeight
      Background.Picture(i, e, 1) = Form1.P1.Point(i, e)
   Next e
Next i
End Sub
```

Appendix E
Background Code

```
'Property and Global variable Dimensioning


Public Picture() As Long
Public PictureBW() As Boolean
Public PictureGR() As Integer
Public PictureR() As Integer
Public PictureG() As Integer
Public PictureB() As Integer
Public PictureS() As Integer

Public PHeight As Integer, PWidth As Integer


Public Sub Start()
Dim i As Integer
Form1.LP.Enabled = False
'Form1.FR(0).Enabled = False
'Form1.FT.Enabled = False
'Form1.FO.Enabled = False
'Form1.IO.Enabled = False
Form1.Check1.Value = 0
Form1.Check1.Enabled = False
For i = 0 To 5
   Form1.MoveOn(i).Enabled = False
Next i
Form1.P1.Picture = Nothing
Form1.P1.Cls
End Sub
```

Appendix F
Filtering Code

```
'Property and Global variable Dimensioning


Public BWThreshold As Integer


Public Sub FilterPicture()
Dim i As Integer, e As Integer
Dim Red As Integer, Green As Integer, Blue As Integer, Grey As Integer
Dim BW As Boolean
DoEvents
For i = 1 To PWidth
   e = 1
   For e = 1 To PHeight
      Red = Picture(i, e, 1) And &HFF
      Green = ((Picture(i, e, 1) And &HFF00) / &H100) Mod &H100
      Blue = ((Picture(i, e, 1) And &HFF0000) / &H10000) Mod &H100
      Grey = (Red + Green + Blue) / 3
      BW = False
      If Grey <= BWThreshold Then BW = True
      If Grey < 0 Then Grey = 0
      If Red < 0 Then Red = 0
      If Green < 0 Then Green = 0
      If Blue < 0 Then Blue = 0
      PictureBW(i, e, 1) = BW
      PictureGR(i, e, 1) = Grey
      PictureR(i, e, 1) = Red
      PictureG(i, e, 1) = Green
      PictureB(i, e, 1) = Blue
   Next e
Next i
End Sub

Public Sub LoadFullPictureF()
Dim i As Integer, e As Integer
DoEvents
For i = 1 To PWidth
   e = 1
   For e = 1 To PHeight
      Form1.P1.PSet (i, e), Picture(i, e, 1)
   Next e
Next i
End Sub
```

Appendix G
Smoothing Code

```
'Property and Global variable Dimensioning


Public SMThreshold As Integer


Public Sub Smooth()
Dim i As Integer, e As Integer
Call ClearSimplifiedBuffers
If Form1.Option10.Value Then
   Call SmoothUpLeft
   Call SmoothUpRight
   Call SmoothDownLeft
   Call SmoothDownRight
ElseIf Form1.Option9.Value Then
   Call SmoothUpLeft
   Call SmoothUpRight
   Call SmoothDownLeft
ElseIf Form1.Option8.Value Then
   Call SmoothUpLeft
   Call SmoothDownLeft
Else:
   Call SmoothUpLeft
End If
For i = 1 To PWidth
   e = 1
   For e = 1 To PHeight
      If Picture(i, e, 2) < SMThreshold Then
         Picture(i, e, 3) = 1
         PictureS(i, e, 1) = 1
      End If
      If PictureGR(i, e, 2) < SMThreshold Then
         PictureGR(i, e, 3) = 1
         PictureS(i, e, 1) = 1
      End If
      If PictureR(i, e, 2) < SMThreshold Then
         PictureR(i, e, 3) = 1
         PictureS(i, e, 1) = 1
      End If
      If PictureG(i, e, 2) < SMThreshold Then
         PictureG(i, e, 3) = 1
         PictureS(i, e, 1) = 1
      End If
      If PictureB(i, e, 2) < SMThreshold Then
         PictureB(i, e, 3) = 1
         PictureS(i, e, 1) = 1
      End If
   Next e
Next i
i = 0
e = 0
For i = 1 To PWidth
   Picture(i, 1, 2) = Picture(i, 2, 2)
   PictureBW(i, 1, 2) = PictureBW(i, 2, 2)
   PictureGR(i, 1, 2) = PictureGR(i, 2, 2)
   PictureR(i, 1, 2) = PictureR(i, 2, 2)
   PictureG(i, 1, 2) = PictureG(i, 2, 2)
   PictureB(i, 1, 2) = PictureB(i, 2, 2)
   Picture(i, 1, 3) = Picture(i, 2, 3)
   PictureGR(i, 1, 3) = PictureGR(i, 2, 3)
   PictureR(i, 1, 3) = PictureR(i, 2, 3)
```

```
    PictureG(i, 1, 3) = PictureG(i, 2, 3)
    PictureB(i, 1, 3) = PictureB(i, 2, 3)
Next i
For e = 1 To PHeight
    Picture(1, e, 2) = Picture(2, e, 2)
    PictureBW(1, e, 2) = PictureBW(2, e, 2)
    PictureGR(1, e, 2) = PictureGR(2, e, 2)
    PictureR(1, e, 2) = PictureR(2, e, 2)
    PictureG(1, e, 2) = PictureG(2, e, 2)
    PictureB(1, e, 2) = PictureB(2, e, 2)
    Picture(1, e, 3) = Picture(2, e, 3)
    PictureGR(1, e, 3) = PictureGR(2, e, 3)
    PictureR(1, e, 3) = PictureR(2, e, 3)
    PictureG(1, e, 3) = PictureG(2, e, 3)
    PictureB(1, e, 3) = PictureB(2, e, 3)
Next e
End Sub

Public Sub SmoothUpLeft()
Dim i As Integer, e As Integer
i = PWidth
DoEvents
For i = PWidth To 2 Step -1
    e = PHeight
    For e = PHeight To 2 Step -1
        Picture(i, e, 2) = 255 - Abs(((PictureR(i - 1, e - 1, 1) + PictureG(i - 1, e - 1, 1) + PictureB(i - 1, e - 1, 1)) / 3) - ((PictureR(i, e, 1)
+ PictureG(i, e, 1) + PictureB(i, e, 1)) / 3))
        If (PictureBW(i - 1, e - 1, 1) And Not PictureBW(i, e, 1)) Or (Not PictureBW(i - 1, e - 1, 1) And PictureBW(i, e, 1)) Then
PictureBW(i, e, 2) = True
        PictureGR(i, e, 2) = 255 - Abs(PictureGR(i - 1, e - 1, 1) - PictureGR(i, e, 1))
        PictureR(i, e, 2) = 255 - Abs(PictureR(i - 1, e - 1, 1) - PictureR(i, e, 1))
        PictureG(i, e, 2) = 255 - Abs(PictureG(i - 1, e - 1, 1) - PictureG(i, e, 1))
        PictureB(i, e, 2) = 255 - Abs(PictureB(i - 1, e - 1, 1) - PictureB(i, e, 1))
    Next e
Next i
End Sub

Public Sub SmoothUpRight()
Dim i As Integer, e As Integer
DoEvents
For i = 1 To PWidth - 1
    e = PHeight
    For e = PHeight To 2 Step -1
        t = Picture(i, e, 2)
        Picture(i, e, 2) = 255 - Abs(((PictureR(i + 1, e - 1, 1) + PictureG(i + 1, e - 1, 1) + PictureB(i + 1, e - 1, 1)) / 3) - ((PictureR(i, e,
1) + PictureG(i, e, 1) + PictureB(i, e, 1)) / 3))
        If t < Picture(i, e, 2) Then Picture(i, e, 2) = t
        t = PictureGR(i, e, 2)
        PictureGR(i, e, 2) = 255 - Abs(PictureGR(i + 1, e - 1, 1) - PictureGR(i, e, 1))
        If t < PictureGR(i, e, 2) Then PictureGR(i, e, 2) = t
        t = PictureR(i, e, 2)
        PictureR(i, e, 2) = 255 - Abs(PictureR(i + 1, e - 1, 1) - PictureR(i, e, 1))
        If t < PictureR(i, e, 2) Then PictureR(i, e, 2) = t
        t = PictureG(i, e, 2)
        PictureG(i, e, 2) = 255 - Abs(PictureG(i + 1, e - 1, 1) - PictureG(i, e, 1))
        If t < PictureG(i, e, 2) Then PictureG(i + 1, e - 1, 2) = t
        t = PictureB(i, e, 2)
        PictureB(i, e, 2) = 255 - Abs(PictureB(i + 1, e - 1, 1) - PictureB(i, e, 1))
        If t < PictureB(i, e, 2) Then PictureB(i, e, 2) = t
        If (PictureBW(i + 1, e - 1, 2) And Not PictureBW(i, e, 1)) Or (Not PictureBW(i + 1, e - 1, 1) And PictureBW(i, e, 1)) Then
PictureBW(i, e, 2) = True
    Next e
Next i
End Sub

Public Sub SmoothDownLeft()
Dim i As Integer, e As Integer, t As Integer
i = PWidth
DoEvents
For i = PWidth To 2 Step -1
    e = 1
```

```
   For e = 1 To PHeight - 1
      t = Picture(i, e, 2)
      Picture(i, e, 2) = 255 - Abs(((PictureR(i - 1, e + 1, 1) + PictureG(i - 1, e + 1, 1) + PictureB(i - 1, e + 1, 1)) / 3) - ((PictureR(i, e,
1) + PictureG(i, e, 1) + PictureB(i, e, 1)) / 3))
      If t < Picture(i, e, 2) Then Picture(i, e, 2) = t
      t = PictureGR(i, e, 2)
      PictureGR(i, e, 2) = 255 - Abs(PictureGR(i - 1, e + 1, 1) - PictureGR(i, e, 1))
      If t < PictureGR(i, e, 2) Then PictureGR(i, e, 2) = t
      t = PictureR(i, e, 2)
      PictureR(i, e, 2) = 255 - Abs(PictureR(i - 1, e + 1, 1) - PictureR(i, e, 1))
      If t < PictureR(i, e, 2) Then PictureR(i, e, 2) = t
      t = PictureG(i, e, 2)
      PictureG(i, e, 2) = 255 - Abs(PictureG(i - 1, e + 1, 1) - PictureG(i, e, 1))
      If t < PictureG(i, e, 2) Then PictureG(i, e, 2) = t
      t = PictureB(i, e, 2)
      PictureB(i, e, 2) = 255 - Abs(PictureB(i - 1, e + 1, 1) - PictureB(i, e, 1))
      If t < PictureB(i, e, 2) Then PictureB(i, e, 2) = t
      If (PictureBW(i - 1, e + 1, 1) And Not PictureBW(i, e, 1)) Or (Not PictureBW(i - 1, e + 1, 1) And PictureBW(i, e, 1)) Then
PictureBW(i, e, 2) = True
   Next e
Next i
End Sub

Public Sub SmoothDownRight()
Dim i As Integer, e As Integer
i = PWidth
DoEvents
For i = 1 To PWidth - 1
   e = 1
   For e = 1 To PHeight - 1
      t = Picture(i, e, 2)
      Picture(i, e, 2) = 255 - Abs(((PictureR(i + 1, e + 1, 1) + PictureG(i + 1, e + 1, 1) + PictureB(i + 1, e + 1, 1)) / 3) - ((PictureR(i,
e, 1) + PictureG(i, e, 1) + PictureB(i, e, 1)) / 3))
      If t < Picture(i, e, 2) Then Picture(i, e, 2) = t
      t = PictureGR(i, e, 2)
      PictureGR(i, e, 2) = 255 - Abs(PictureGR(i + 1, e + 1, 1) - PictureGR(i, e, 1))
      If t < PictureGR(i, e, 2) Then PictureGR(i, e, 2) = t
      t = PictureR(i, e, 2)
      PictureR(i, e, 2) = 255 - Abs(PictureR(i + 1, e + 1, 1) - PictureR(i, e, 1))
      If t < PictureR(i, e, 2) Then PictureR(i, e, 2) = t
      t = PictureG(i, e, 2)
      PictureG(i, e, 2) = 255 - Abs(PictureG(i + 1, e + 1, 1) - PictureG(i, e, 1))
      If t < PictureG(i, e, 2) Then PictureG(i, e, 2) = t
      t = PictureB(i, e, 2)
      PictureB(i, e, 2) = 255 - Abs(PictureB(i + 1, e + 1, 1) - PictureB(i, e, 1))
      If t < PictureB(i, e, 2) Then PictureB(i, e, 2) = t
      If (PictureBW(i + 1, e + 1, 1) And Not PictureBW(i, e, 1)) Or (Not PictureBW(i + 1, e + 1, 1) And PictureBW(i, e, 1)) Then
PictureBW(i, e, 2) = True
   Next e
Next i
End Sub

Public Sub LoadFullPictureS(Optional All As Boolean)
Dim i As Integer, e As Integer
DoEvents
If Form1.Check1.Value = 0 Then
   For i = 1 To PWidth
      e = 1
      For e = 1 To PHeight
         Form1.P1.PSet (i, e), RGB(Picture(i, e, 2), Picture(i, e, 2), Picture(i, e, 2))
      Next e
   Next i
Else:
   For i = 1 To PWidth
      e = 1
      For e = 1 To PHeight
         If Picture(i, e, 3) = 0 Then
            Form1.P1.PSet (i, e), vbWhite
         Else:
            Form1.P1.PSet (i, e), 0
         End If
```

```
            If All Then
               If PictureS(i, e, 1) <> 0 Then Form1.P1.PSet (i, e), 0
            End If
         Next e
      Next i
   End If
   End Sub
```

Appendix H
Scanning Code

```
'Property and Global variable Dimensioning


Public Corners As New Collection
Public ActiveConnections As New Collection
Public CompleteConnections As New Collection
Public SqueezeGrid As New Collection

Public Detail As Integer
Public ScanSpeed As String
Public ScanMode As Integer


Public Sub ThinLines()

End Sub


Public Sub LaySeeds()
Dim i As Integer, e As Integer, s As Integer
For i = 1 To PWidth
   e = 1
   If i / Detail = Round(i / Detail) Then
      s = 1
   Else:
      s = Detail
   End If
   For e = 1 To PHeight Step s
      If Picture(i, e, 3) = 1 Then
         Call CreateNewCorner(i, e, "Full", Picture(i, e, 1))
         Call CreateNewConnection(i, e, "Full", Picture(i, e, 1))
      End If
      If PictureBW(i, e, 2) Then
         Call CreateNewCorner(i, e, "Black", 0)
         Call CreateNewConnection(i, e, "Black", 0)
      End If
      If PictureGR(i, e, 3) = 1 Then
         Call CreateNewCorner(i, e, "Grey", RGB(128, 128, 128))
         Call CreateNewConnection(i, e, "Grey", RGB(128, 128, 128))
      End If
      If PictureR(i, e, 3) = 1 Then
         Call CreateNewCorner(i, e, "Red", RGB(255, 0, 0))
         Call CreateNewConnection(i, e, "Red", RGB(255, 0, 0))
      End If
      If PictureG(i, e, 3) = 1 Then
         Call CreateNewCorner(i, e, "Green", RGB(0, 255, 0))
         Call CreateNewConnection(i, e, "Green", RGB(0, 255, 0))
      End If
      If PictureB(i, e, 3) = 1 Then
         Call CreateNewCorner(i, e, "Blue", RGB(0, 0, 255))
         Call CreateNewConnection(i, e, "Blue", RGB(0, 0, 255))
      End If
      If PictureS(i, e, 1) = 1 Then
         Call CreateNewCorner(i, e, "Simple", RGB(128, 128, 128))
         Call CreateNewConnection(i, e, "Simple", RGB(128, 128, 128))
      End If
   Next e
Next i
End Sub

Public Sub SetupSqueezeGrid()
Dim i As Integer, e As Integer, n As Integer, CX As Integer, CY As Integer
```

```
Dim IntervalW As Single, IntervalH As Single
IntervalW = PWidth / Detail
IntervalH = PHeight / Detail
CX = 1 - IntervalW
CY = 1 - IntervalH
For i = 0 To Detail - 1
   e = 0
   CX = 1 - IntervalW
   CY = CY + IntervalH
   For e = 0 To Detail - 1
      n = n + 1
      CX = CX + IntervalW
      Call CreateSGField(CX, CY, n, IntervalW, IntervalH)
   Next e
Next i
End Sub

Public Sub CreateSGField(X As Integer, Y As Integer, Number As Integer, IW As Single, IH As Single)
Dim F1 As New CSGridCell
F1.Width = IW
F1.Height = IH
F1.IW = Int(IW / 4)
F1.IH = Int(IH / 4)
F1.Number = Number
F1.X = X
F1.Y = Y
F1.L1A = True
F1.L2A = True
F1.L3A = True
F1.L4A = True
F1.L5A = True
F1.R1A = True
F1.R2A = True
F1.R3A = True
F1.R4A = True
F1.R5A = True
F1.T1A = True
F1.T2A = True
F1.T3A = True
F1.T4A = True
F1.T5A = True
F1.B1A = True
F1.B2A = True
F1.B3A = True
F1.B4A = True
F1.B5A = True
SqueezeGrid.Add F1
End Sub

Public Sub ScanSub3()

End Sub

Public Sub ScanSub4()

End Sub

Public Sub ScanningLoop()
Dim i As Long
Dim Done As Boolean
DoEvents
Do While Done = False
   Select Case ScanMode
      Case 1
         If ActiveConnections.Count > 0 Then
            Done = True
            For i = 1 To ActiveConnections.Count
               If i > ActiveConnections.Count Then Exit For
               If ActiveConnections(i).Following Then
                  'Call ActiveConnections(i).DrawCurrent
                  Call ActiveConnections(i).Follow
```

```
                If ActiveConnections(i).Remove Then Call RemoveConnection(i)
                Done = False
              End If
            Next i
          End If
          If Done Then
            Form1.ScanningTimer.Enabled = False
            Form1.Frame1(3).Enabled = True
          End If
        Case 2
          Done = True
          For i = 1 To SqueezeGrid.Count
            If SqueezeGrid(i).Squeezed = False Then
              Call SqueezeGrid(i).Squeeze
              Done = False
            End If
          Next i
          If Done Then
            Form1.ScanningTimer.Enabled = False
            Form1.Frame1(3).Enabled = True
          End If
        Case 3

        Case 4

    End Select
Loop
End Sub

Public Sub RemoveConnection(i As Long)
Dim C1 As New CConnection
C1.SX = ActiveConnections(i).SX
C1.SY = ActiveConnections(i).SY
C1.PX = ActiveConnections(i).PX
C1.PY = ActiveConnections(i).PY
C1.CX = ActiveConnections(i).CX
C1.CY = ActiveConnections(i).CY
C1.EX = ActiveConnections(i).EX
C1.EY = ActiveConnections(i).EY
C1.Following = ActiveConnections(i).Following
C1.Color = ActiveConnections(i).Color
C1.ColorCode = ActiveConnections(i).ColorCode
C1.Remove = ActiveConnections(i).Remove
C1.Complete = ActiveConnections(i).Complete
If C1.Complete Then CompleteConnections.Add C1
ActiveConnections.Remove (i)
End Sub

Public Sub DrawConnections(Color As String)
Dim i As Integer
Form1.P1.Cls
If CompleteConnections.Count > 0 Then
    For i = 1 To CompleteConnections.Count
        If CompleteConnections(i).Color = Color Or Color = "Full" Then Call CompleteConnections(i).DrawTotal
    Next i
End If
End Sub

Public Sub CreateNewCorner(X As Integer, Y As Integer, c As String, CC As Long)
Dim C1 As New CCorner
C1.X = X
C1.Y = Y
C1.Color = c
C1.ColorCode = CC
Corners.Add C1
End Sub

Public Sub CreateNewConnection(X As Integer, Y As Integer, c As String, CC As Long)
Dim C1 As New CConnection
C1.Following = True
C1.Color = c
```

```
C1.ColorCode = CC
C1.SX = X
C1.SY = Y
C1.CX = X
C1.CY = Y
ActiveConnections.Add C1
End Sub
```

Appendix I
Analyses Code

```
' Attention, the code within this module has not been used by the program as of yet.

'Property and Global variable Dimensioning



Public FieldBW() As Integer
Public FieldGR() As Integer
Public FieldR() As Integer
Public FieldG() As Integer
Public FieldB() As Integer

Public PixelationFactor As Integer
Public EComp  As Integer


Public Sub GeneratePolygons()

End Sub

Public Sub Pixelate()
Dim i As Integer, e As Integer, c As Integer, f As Integer, g As Integer
Dim Sum(6) As Long
ReDim FieldBW((PWidth / PixelationFactor) + 1, (PHeight / PixelationFactor) + 1, 2) As Integer
ReDim FieldGR((PWidth / PixelationFactor) + 1, (PHeight / PixelationFactor) + 1, 2) As Integer
ReDim FieldR((PWidth / PixelationFactor) + 1, (PHeight / PixelationFactor) + 1, 2) As Integer
ReDim FieldG((PWidth / PixelationFactor) + 1, (PHeight / PixelationFactor) + 1, 2) As Integer
ReDim FieldB((PWidth / PixelationFactor) + 1, (PHeight / PixelationFactor) + 1, 2) As Integer

For i = 1 To PWidth Step PixelationFactor
   e = 1
   For e = 1 To PHeight Step PixelationFactor
      f = i
      c = 0
      Sum(1) = 0
      Sum(2) = 0
      Sum(3) = 0
      Sum(4) = 0
      Sum(5) = 0
      For f = i To i + PixelationFactor
         g = e
         For g = e To e + PixelationFactor
            If PictureBW(f, g, 1) Then Sum(1) = Sum(1) + 1
            Sum(2) = Sum(2) + PictureGR(f, g, 1)
            Sum(3) = Sum(3) + PictureR(f, g, 1)
            Sum(4) = Sum(4) + PictureG(f, g, 1)
            Sum(1) = Sum(1) + PictureB(f, g, 1)
            c = c + 1
            If g = PHeight Then Exit For
         Next g
         If f = PWidth Then Exit For
      Next f
      If i > 1 Then
         If e > 1 Then
            FieldBW(i / PixelationFactor, e / PixelationFactor, 1) = Sum(1) / c
            FieldGR(i / PixelationFactor, e / PixelationFactor, 1) = Sum(2) / c
            FieldR(i / PixelationFactor, e / PixelationFactor, 1) = Sum(3) / c
            FieldG(i / PixelationFactor, e / PixelationFactor, 1) = Sum(4) / c
            FieldB(i / PixelationFactor, e / PixelationFactor, 1) = Sum(5) / c
         Else:
            FieldBW(i / PixelationFactor, 1, 1) = Sum(1) / c
            FieldGR(i / PixelationFactor, 1, 1) = Sum(2) / c
            FieldR(i / PixelationFactor, 1, 1) = Sum(3) / c
            FieldG(i / PixelationFactor, 1, 1) = Sum(4) / c
```

```
            FieldB(i / PixelationFactor, 1, 1) = Sum(5) / c
        End If
    Else:
        If e > 1 Then
            FieldBW(1, e / PixelationFactor, 1) = Sum(1) / c
            FieldGR(1, e / PixelationFactor, 1) = Sum(2) / c
            FieldR(1, e / PixelationFactor, 1) = Sum(3) / c
            FieldG(1, e / PixelationFactor, 1) = Sum(4) / c
            FieldB(1, e / PixelationFactor, 1) = Sum(5) / c
        Else:
            FieldBW(1, 1, 1) = Sum(1) / c
            FieldGR(1, 1, 1) = Sum(2) / c
            FieldR(1, 1, 1) = Sum(3) / c
            FieldG(1, 1, 1) = Sum(4) / c
            FieldB(1, 1, 1) = Sum(5) / c
        End If
    End If



  Next e
Next i



End Sub

Public Sub FindFieldCorners()

End Sub
```

Appendix J
Form1 Code

```
'Property and Global variable Dimensioning



Private Sub FFP_Click()

FSP.Enabled = True
FSP.SetFocus
End Sub

Private Sub FSP_Click()

MoveOn(3).Enabled = True
MoveOn(3).SetFocus
End Sub

Private Sub MAdvanced_Click()
Settings.Show
End Sub

Private Sub AO_Click()
IO.Enabled = True
IO.SetFocus
End Sub

Private Sub Check1_Click()
P1.Cls
If Check1.Value = 1 Then
   Call LoadFullPictureS(True)
Else:
   Call LoadFullPictureS
End If
End Sub

Private Sub FC_Click()
Do While ActiveConnections.Count > 0
   ActiveConnections.Remove (1)
Loop
Do While CompleteConnections.Count > 0
   CompleteConnections.Remove (1)
Loop
Call ClearScanningPictureBuffers
Frame1(3).Enabled = False
P1.Cls
Select Case ScanMode
   Case 1
      If Option1.Value Then Detail = 40
      If Option2.Value Then Detail = 25
      If Option3.Value Then Detail = 18
      If Option4.Value Then Detail = 10
      If Option5.Value Then Detail = 5
      If Option6.Value Then Detail = 2
      Call LaySeeds
   Case 2
      If Option1.Value Then Detail = 3
      If Option2.Value Then Detail = 5
      If Option3.Value Then Detail = 7
      If Option4.Value Then Detail = 10
      If Option5.Value Then Detail = 14
      If Option6.Value Then Detail = 20
      Call SetupSqueezeGrid
   Case 3
      Call ScanSub3
```

```
    Case 4
        Call ScanSub4
End Select
Select Case ScanSpeed
    Case "I"
        Call Scanning.ScanningLoop
    Case "VF"
        Call Scanning.ScanningLoop
    Case "F"
        ScanningTimer.Interval = 1
        ScanningTimer.Enabled = True
    Case "S"
        ScanningTimer.Interval = 10
        ScanningTimer.Enabled = True
    Case "VS"
        ScanningTimer.Interval = 20
        ScanningTimer.Enabled = True
End Select
NOL.Caption = "Number of Lines: " & CompleteConnections.Count
Call DrawConnections("Full")
FFP.Enabled = True
FFP.SetFocus
End Sub

Private Sub FL_Click()
FR(0).Enabled = True
End Sub

Private Sub FO_Click()
MoveOn(4).Enabled = True
MoveOn(4).SetFocus
End Sub

Private Sub Form_Load()
'If Dir("C:\Program Files\Artificial Eye\InSat") = "" Then
'    MsgBox "Please Install Artificial Eye", , "ERROR [>_<]"
'    End
'End If
P1.BackColor = vbWhite
BWThreshold = 127
SMThreshold = 215
ScanSpeed = "VF"
ScanMode = 2
Detail = 15
Settings.W = 204
Settings.H = 204
Call Start
End Sub

Private Sub Form_Terminate()
End
End Sub

Private Sub FP_Click()
Call FilterPicture
P1.Picture = Nothing
Call LoadFullPictureF
MoveOn(1).Enabled = True
MoveOn(1).SetFocus
End Sub

Private Sub FR_Click(Index As Integer)
PixelationFactor = Detail / 2
Call Pixelate


FT.Enabled = True
FT.SetFocus
End Sub

Private Sub FT_Click()
```

```
FO.Enabled = True
FO.SetFocus
End Sub

Private Sub GP_Click()
EComp = Slider1.Value
Call GeneratePolygons
MoveOn(3).Enabled = True
MoveOn(3).SetFocus
End Sub

Private Sub IO_Click()
MoveOn(5).Enabled = True
MoveOn(5).SetFocus
End Sub

Private Sub IP_Click()
CD1.ShowOpen
If CD1.FileName = "" Then Exit Sub
P1.Picture = LoadPicture(CD1.FileName)
P1.PaintPicture P1.Picture, 0, 0, P1.ScaleWidth, P1.ScaleHeight
PHeight = P1.ScaleHeight
PWidth = P1.ScaleWidth
Call ClearPictureBuffers
LP.Enabled = True
LP.SetFocus
End Sub

Private Sub LP_Click()
Call LoadNewPicture
MoveOn(0).Enabled = True
MoveOn(0).SetFocus
End Sub

Private Sub MExit_Click()
End
End Sub

Private Sub MoveOn_Click(Index As Integer)
If Index < 5 Then
   Frame1(Index).Visible = False
   Frame1(Index + 1).Visible = True
Else:
   Frame1(5).Visible = False
   Frame1(0).Visible = True
   Call Start
End If
Select Case Index
   Case 0
      FP.SetFocus
   Case 1
      SP.SetFocus
   Case 2
      FC.Enabled = True
      FC.SetFocus
   Case 3
      'FR(0).Enabled = True
      'FR(0).SetFocus
   Case 4
      AO.SetFocus
   Case 5
      IP.SetFocus
End Select
End Sub

Private Sub ScanningTimer_Timer()
Dim i As Long
Dim Done As Boolean
If ActiveConnections.Count > 0 Then
```

```
    Done = True
    For i = 1 To ActiveConnections.Count
      If i > ActiveConnections.Count Then Exit For
      If ActiveConnections(i).Following Then
        Call ActiveConnections(i).DrawCurrent
        Call ActiveConnections(i).Follow
        If ActiveConnections(i).Remove Then ActiveConnections.Remove (i)
        Done = False
      End If
    Next i
  End If
  If Done Then
    If Done Then
      ScanningTimer.Enabled = False
      Call DrawConnections("Black")
      Call DrawConnections("Grey")
      Call DrawConnections("Red")
      Call DrawConnections("Green")
      Call DrawConnections("Blue")
      Frame1(3).Enabled = True
    End If
  End If
  End Sub

  Private Sub SP_Click()
  Call Smooth
  Call LoadFullPictureS
  Check1.Enabled = True
  MoveOn(2).Enabled = True
  MoveOn(2).SetFocus
  End Sub

  Private Sub TL_Click()
  Call ThinLines
  End Sub
```

Appendix K
Settings_Code

```
'Property and Global variable Dimensioning


Public W As Integer, H As Integer

Private Sub Apply_Click()
Form1.ScaleWidth = W + 200
Form1.P1.Width = W
Form1.P1.Height = H
If Option9.Value Then
   If Val(Text1.Text) < 75 Or Val(Text1.Text) > 7125 Then
      MsgBox "Please enter a reasonable width. (75-7125)", , "Invalid value"
      Text1.SetFocus
      Exit Sub
   End If
   If Val(Text2.Text) < 75 Or Val(Text2.Text) > 12790 Then
      MsgBox "Please enter a reasonable width. (75-12790)", , "Invalid value"
      Text2.SetFocus
      Exit Sub
   End If
   W = Val(Text2.Text)
   H = Val(Text1.Text)
   Form1.ScaleWidth = W + 200
End If
Form1.P1.Width = W
Form1.P1.Height = H
If Option10.Value Then ScanMode = 1
If Option11.Value Then ScanMode = 2
If Option12.Value Then ScanMode = 3
If Option13.Value Then ScanMode = 4
BWThreshold = Slider1.Value
SMThreshold = Slider2.Value
If Option1.Value Then ScanSpeed = "I"
If Option2.Value Then ScanSpeed = "VF"
If Option3.Value Then ScanSpeed = "F"
If Option4.Value Then ScanSpeed = "S"
If Option5.Value Then ScanSpeed = "VS"
Hide
End Sub

Private Sub Form_Load()
Slider1.Value = BWThreshold
Slider2.Value = SMThreshold
Select Case ScanSpeed
   Case "I"
      Option1.Value = True
   Case "VF"
      Option2.Value = True
   Case "F"
      Option3.Value = True
   Case "S"
      Option4.Value = True
   Case "VS"
      Option5.Value = True
End Select
Select Case ScanMode
   Case 1
      Option10.Value = True
   Case 2
      Option11.Value = True
```

```
    Case 3
        Option12.Value = True
    Case 4
        Option13.Value = True
End Select
End Sub

Private Sub Option6_Click()
Label3.Enabled = False
Label4.Enabled = False
Text1.Enabled = False
Text2.Enabled = False
W = 104
H = 104
Text1.Text = "100"
Text2.Text = "100"
End Sub

Private Sub Option7_Click()
Label3.Enabled = False
Label4.Enabled = False
Text1.Enabled = False
Text2.Enabled = False
W = 154
H = 154
Text1.Text = "150"
Text2.Text = "150"
End Sub

Private Sub Option8_Click()
Label3.Enabled = False
Label4.Enabled = False
Text1.Enabled = False
Text2.Enabled = False
W = 204
H = 204
Text1.Text = "200"
Text2.Text = "200"
End Sub

Private Sub Option9_Click()
Label3.Enabled = True
Label4.Enabled = True
Text1.Enabled = True
Text2.Enabled = True
End Sub
```