Introduction

Inpainting refers to the process of replacing lost or corrupted parts of the image data. This idea was followed to attempt to remove a crease on the Abraham Lincoln picture that corrupted the pixels specifically along the hair. This work details the techniques used to remove the corrupted area, with emphasis on the usage of the Lagrange Interpolating Polynomial.

Corrupted Region

The picture contains a crease, which is a myriad of pixels of higher intensity values. The crease runs from the upper left corner of the image, through Lincoln’s hair, and finally to the mid-right of the image. The area of concern is mainly the corrupted region on the hair.

Modeling the crease by a function

Looking at the picture, we can see that the corrupted region closely resembles a logarithmic function. The function $y = \alpha \ln(x - \beta)$, where $\alpha$ and $\beta$ represent constants as determined by image size and quality, was used to model the position values of the corrupted region on the picture.
Using Minimum Values

This technique finds the minimum value in $\begin{bmatrix} \epsilon & \epsilon & \epsilon \\ \epsilon & \alpha & \epsilon \\ \epsilon & \epsilon & \epsilon \end{bmatrix}$, where $\epsilon$ is a bordering pixel and $\alpha$ is the center pixel of a $3 \times 3$ matrix. The minimum value is then applied to the center pixel and a Gaussian kernel is applied, with sigma of 3, to smooth the added pixels.
Using Median Values

This technique finds the median value of the pixels in the 3 x 3 matrix. The value is applied to the region in the 2 x 2 region around the center pixel. Then a Gaussian filter, with sigma of 5, was applied to smooth the adjusted pixels.

LaGrange Interpolating Polynomial

The techniques mentioned above do not adequately apply pixel values that closely model the intensity values needed to remove the crease. To provide appropriate intensities, the Lagrange Interpolating Polynomial uses supplied points to predict the most ideal value for each pixel.

By the interpolating polynomial, it is known that

\[ L(x) = \sum_{j=1}^{n} L_j(x) \quad \text{where} \]

\[ L_j(x) = y_j \prod_{k=1}^{n} \frac{x - x_k}{x_j - x_k} \]

The following formula simulates the polynomial:

\[ I(x, y) = \sum_{n=1}^{5} \frac{\alpha_n y_n}{b} \]

and \( \alpha_n = I(x \pm \beta, y \pm \beta) = I(x_n, y_n) \), where \( \beta \) is from the 5 x 5 surrounding.

It is also given that \( b = \text{det} \begin{bmatrix} x_1^1 & x_1^2 & y_1^1 & \cdots & y_1^1 \\ x_2^1 & x_2^2 & y_2^1 & \cdots & y_2^1 \\ x_3^1 & x_3^2 & y_3^1 & \cdots & y_3^1 \\ x_4^1 & x_4^2 & y_4^1 & \cdots & y_4^1 \\ x_5^1 & x_5^2 & y_5^1 & \cdots & y_5^1 \end{bmatrix} \)
\[

g_1 = \det \begin{bmatrix}
x^4 & x^3y & \cdots & y^4 \\
x^3 & x^2y_2 & \cdots & y_2^4 \\
x^2 & x^2y_3 & \cdots & y_3^4 \\
x & x^2y_4 & \cdots & y_4^4 \\
x & x^2y_5 & \cdots & y_5^4 \\
x & y_1 & \cdots & y_1^4 \\
x & y_1 & \cdots & y_1^4 \\
x & y_1 & \cdots & y_1^4 \\
x & y_1 & \cdots & y_1^4 \\
x & y_1 & \cdots & y_1^4 \\
x & y_1 & \cdots & y_1^4 \\
\end{bmatrix}
\]

\[

g_2 = \det \begin{bmatrix}
x^4 & x^3y & \cdots & y^4 \\
x^3 & x^2y & \cdots & y^4 \\
x^2 & x^2y & \cdots & y^4 \\
x & x^2y & \cdots & y^4 \\
x & x^2y & \cdots & y^4 \\
x & x^2y & \cdots & y^4 \\
x & x^2y & \cdots & y^4 \\
x & x^2y & \cdots & y^4 \\
x & x^2y & \cdots & y^4 \\
x & x^2y & \cdots & y^4 \\
\end{bmatrix}
\]

All relevant values of $\gamma$ are determined with the above principle in mind, which is dependent on the number of points used.

Hence, using 5 pixels in the neighboring region, the pixels in the corrupted region adopt the value most likely to be the true value. As the crease on the hair was approximated to be 3 pixels wide and the most accurate results were required, this process was implemented for all the pixels of the width of the crease on the hair.

Code Implementation

This section details the key sections of the code created and run in GNU Octave and its relation to the formulas mentioned in previous sections. The intent and functionality of each section of code is also described to produce the picture above.

The position function is stated and iterated over in order to access each pixel value along the corrupted region.
Five pixels in the 5 x 5 surrounding region are chosen for the interpolation algorithm. Pixel values are chosen differently where the crease nears the bald part in order to ensure that the polynomial does not assume values not in the hair for approximating the correct intensity value.

```c
#define
for each point on z,i
for i = 120:240  #38 #118-270
  #Position function for the crease near the head
  #Adding/subtracting numbers to i will move the line up and down
  z = uint64(-1 .* 23 .* log1m(/(1-35)));

Points in 5 x 5 region
x1 = z;
x2 = z;
x3 = z+2;
x4 = z+2;
x5 = z+2;
y1 = 1+2;
y2 = 1+2;
y3 = 1+2;
y4 = 1+2;
y5 = 1+2;

#Altering which pixels looked at near bald part
if(1>= 182 && 1<=240)
    x1 = z;
x2 = z+2;
x3 = z+2;
x4 = z+1;
x5 = z+1;
y1 = 1+2;
y2 = 1+2;
y3 = 1+2;
y4 = 1+2;
y5 = 1+2;
endif

Constant matrix computed
constantMatrix = [ uint64(x1 ** 4), uint64((x1 ** 3) .* y1), uint64((x1 ** 2) .* (y1 ** 2)),
                  uint64((x1) .* (y1 ** 3)), uint64(y1 ** 4);

                  uint64(x2 ** 4), uint64((x2 ** 3) .* y2), uint64((x2 ** 2) .* (y2 ** 2)),
                  uint64((x2) .* (y2 ** 3)), uint64(y2 ** 4);

                  uint64(x3 ** 4), uint64((x3 ** 3) .* y3), uint64((x3 ** 2) .* (y3 ** 2)),
                  uint64((x3) .* (y3 ** 3)), uint64(y3 ** 4);

                  uint64(x4 ** 4), uint64((x4 ** 3) .* y4), uint64((x4 ** 2) .* (y4 ** 2)),
                  uint64((x4) .* (y4 ** 3)), uint64(y4 ** 4);

                  uint64(x5 ** 4), uint64((x5 ** 3) .* y5), uint64((x5 ** 2) .* (y5 ** 2)),
                  uint64((x5) .* (y5 ** 3)), uint64(y5 ** 4)
];
```

The value of the constant matrix, b, is determined using the surrounding pixel values chosen.

The matrix, \( \gamma \), is computed for each point to be used in the formula.
The appropriate intensity value is computed by the formula using determinants, saved to a variable, and later applied.

```c
if (constant == 0)
  #The formula
  answer = ((det(matrix1) / constant) * alpha1) + ((det(matrix2)/constant)*alpha2) + ((det(matrix3)/constant)*alpha3) + ((det(matrix4)/constant)*alpha4) + ((det(matrix5)/constant)*alpha5);
```

Applying a Gaussian filter, with a relatively small sigma, to allow mild blending of newly added intensities around region.

```c
origMod(z-1:i+1, l-1:i+1) = imfilter(newMatrixAbove2, h, 'symmetric');
originalMod(z-2:i, l-1:i+1) = imfilter(newMatrixAbove2, h, 'symmetric');
```

**Limitations of the polynomial**

Looking at the results, although the corrupted region is nearly completely removed, there is still some evidence of a crease. This is largely due to the direction of application of the new intensity values (a large limitation of this technique). Accordingly, new values are applied to the pixels laterally (iterated through a for loop from left to right), which causes slight evidence of being tampered.

To fix this problem, some other techniques can be used in conjunction with the polynomial, such as modeling heat flow and second order equations (Image inpainting using a TV-Stokes equation).

**Possible Applications**

Looking at the picture created, the algorithm can be used for images with unwanted features that either span throughout the picture or only a part of the image. Using different curve fitting algorithms, the position function should be relatively easy to determine and the Lagrange Interpolation can be applied to remove the corrupted region.