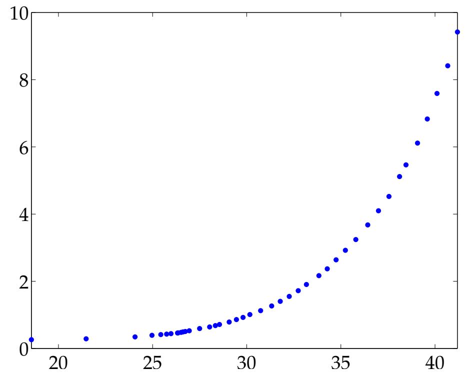
Supplemental Material: Encoding Cost Studies

Contents

1 Notes on kdu Compression

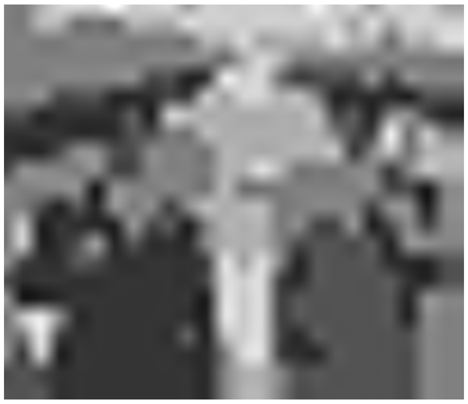
In these tests, we use the Kakadu JPEG2000 codec to create lossy compression results at various file sizes.[1] The Kakadu encoder (kdu) is arguably the most efficient JPEG2000 encoder available, it performed very well in the 2005 codec comparison studies performed at MSU. More recently, it proved to be the best of the various JPEG2000 codecs tested by Lee in his DLI performance studies. [2] [3]

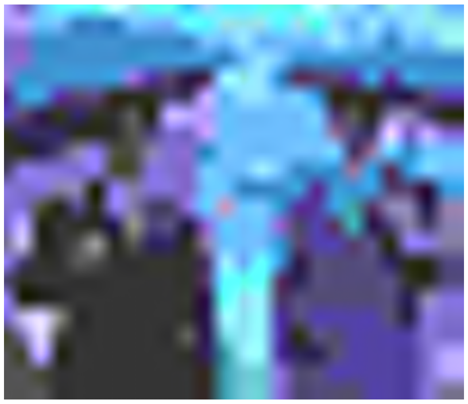
To evaluate each vectorization test case, we generate four different reference compression results. Unlike research codecs such as DLI, the Kakadu encoder is capable of quickly creating results at specific filesizes—making it straightforward to create comparison results at filesizes matching that of the vector data.

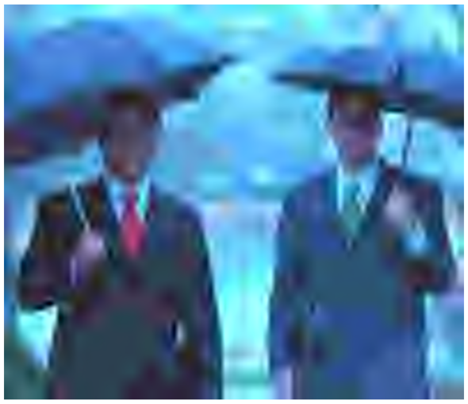


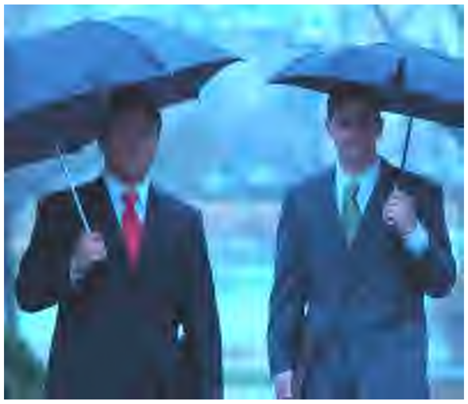
Filesize (kB) as a function of PSNR

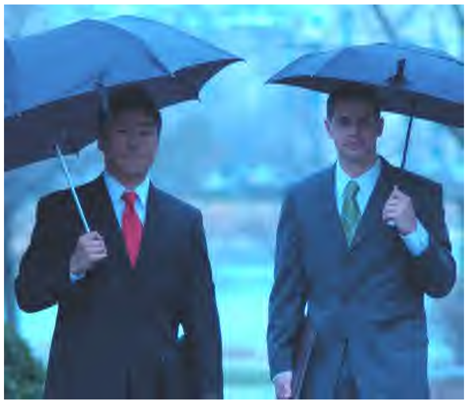
kdu at 0.24kB (psnr=11.06)

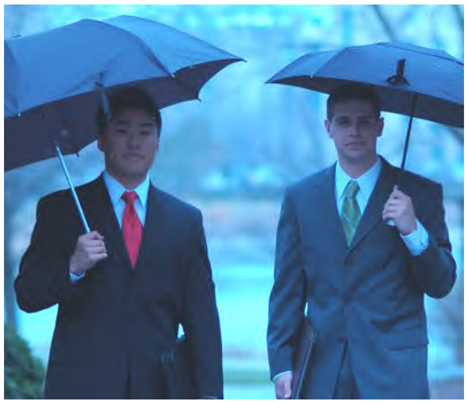

kdu at 0.33kB (psnr=14.04)

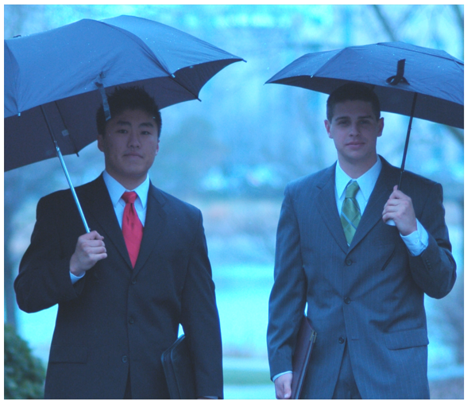

kdu at 0.44kB (psnr=18.11)


kdu at 0.92kB (psnr=23.36)


kdu at 2.07kB (psnr=26.68)


kdu at 4.44kB (psnr=29.82)


kdu at 9.44kB (psnr=33.00)

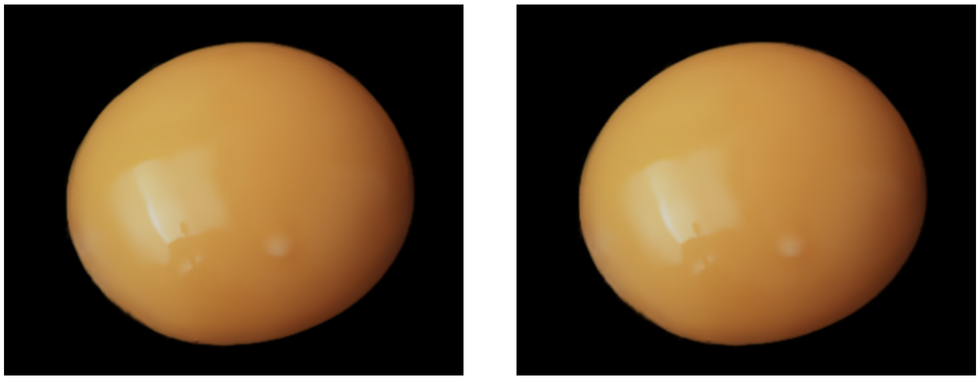

Source Image

[1] http://www.kakadusoftware.com/
[2] http://compression.ru/video/codec_comparison/pdf/jpeg2000_codec_comparison_en.pdf
[3] http://sites.google.com/site/dlimagecomp/test-methodology

Here we show a range of kdu compression results for one of the source images included in our test set. At very small filesizes, the codec appears to limit itself to encoding only luminance information, even when the input image contains colors. While the specific behavior varies for each input image, extreme compression artifacts reliably occur at filesizes near 1kB, though more minor distortions may remain at much larger filesizes.

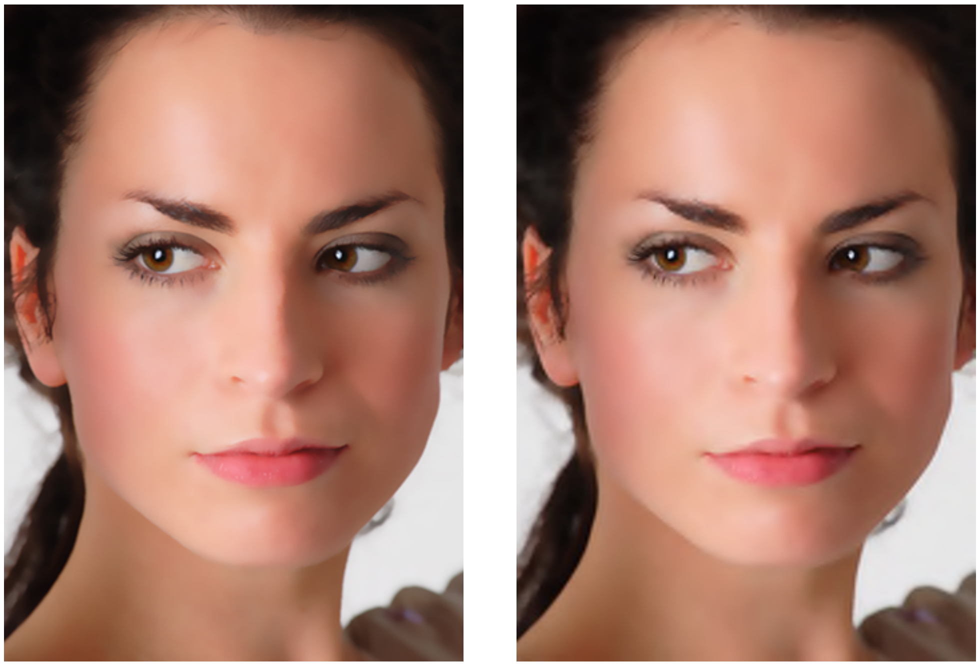
1.1 Gradient Meshes and kdu

In their 2007 paper on gradient meshes, Sun et al. suggested that compactness was one of the potential benefits of a gradient mesh image representation [1]. Though not the focus of their paper – they did reveal a that a nearly perfect gradient mesh representation of this yolk image had a filesize of 7.7kB, while storing the same image in JPEG format would require 37.5kB. The yolk example would have been much less compelling had they compared relative the JPEG2000 format – the kdu compression of the yolk image at 7.7kB shows few if any visible artifacts.



Source Image **kdu at 7.7kB**

While Sun et al. did not publish any filesize information beyond that of the yolk image, the single subject, smooth shaded images on which they test their algorithm are generally very amenable to low bitrate compression. For example, the following image requires a gradient mesh considerably more complex than that used in the yolk, and can also be represented well at by JPEG2000 at 7.7kB.



Source Image **kdu at 7.7kB**

More recent research on gradient meshes by Xia et al. has focused on generating meshes for more complex input images [2]. While their work is impressive, Xia et al. appear to have set aside the goal of compactness—perhaps because the very high efficiency of compression codecs such as Kakadu makes demonstrating any advantage relative to state of the art lossy compression difficult.

2 Example Results and Reference Compressions

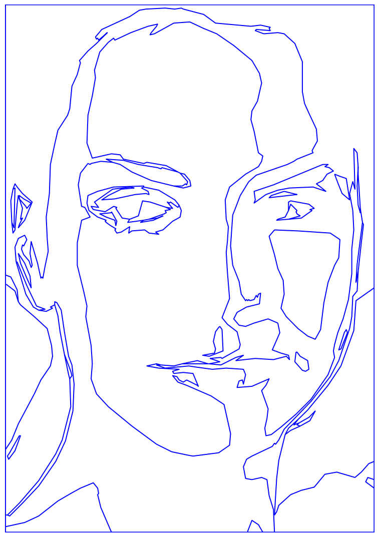
In this section, we show vectorization results for a large range of input photographs. The *vector reconstruction* is the end result of our system. The *image space stylization* shows the stylized image that the vector tracing is based on. The *input photograph* shows the photograph prior to stylization. If the input photograph is color, we also show a greyscale version of the input. For each of these four images, we also show the image as it appears given JPEG2000 compression to a filesize matching that of the vector reconstruction. Thus, when an image is labeled as "compressed", it can be assumed to be a kdu compression to a filesize that matches that of the vector encoding.

2.1 Female Face

Here we show results for one of the images that appears in Sun et al. [1]. Our vectorization results for this image are relatively poor – may significant details are lost or distorted a result of the stylization step, and important shading information is lost in the vector result. However, unlike a gradient mesh result, the vector data generated by our algorithm is sufficiently compact that JPEG2000 encoding produces many visible artifacts at a similar resolution.

In order to better handle the high overall lightness of this image, the bin values $\mathbf{b}_1$ have been changed from their default setting of $(.45, .75, .85)$ to $(.6, .8, .85)$.

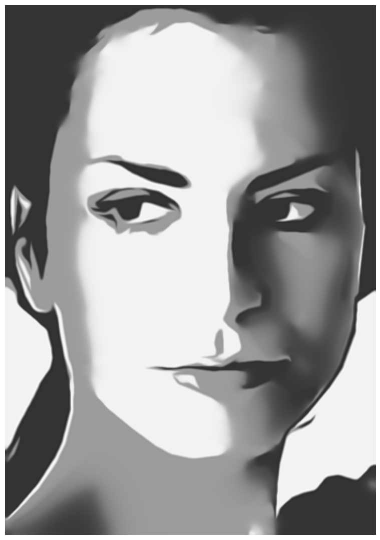
2.1.1 Vector Results

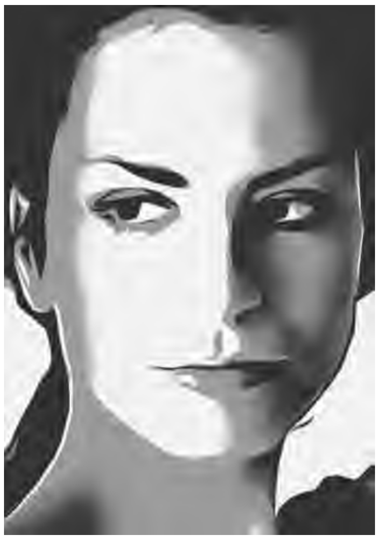

Edge Lines

Encoding Costs

Boundary Curves: 1.22kB (60%)
Edge Sharpness Data: 0.765kB (38%)
Luminance Data: 0.051kB (2.5%)

Total File Size: 2.04kB


Vector Reconstruction, 2.04kB


Compressed Reconstruction

2.1.2 Image Space Stylization Result

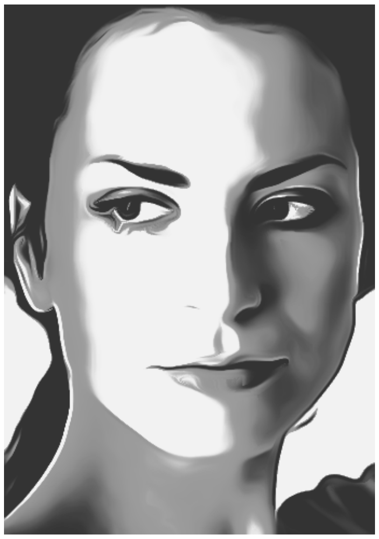
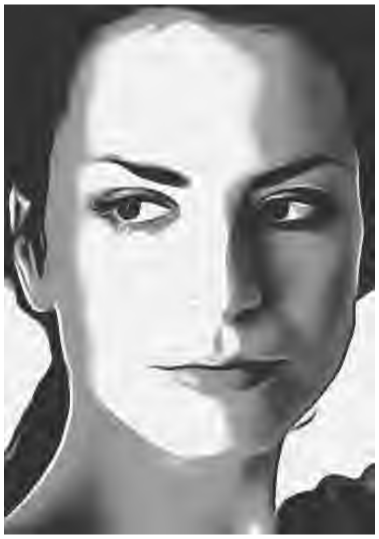
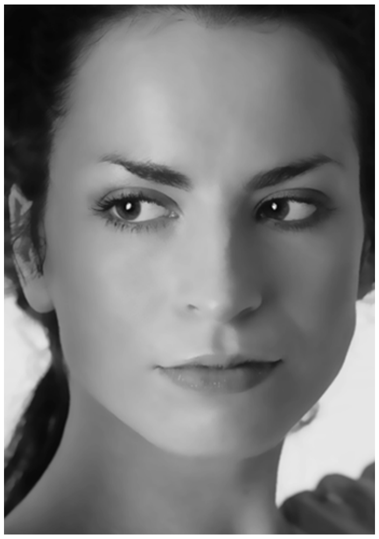
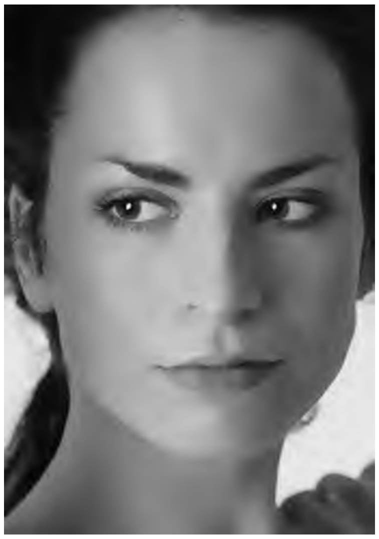


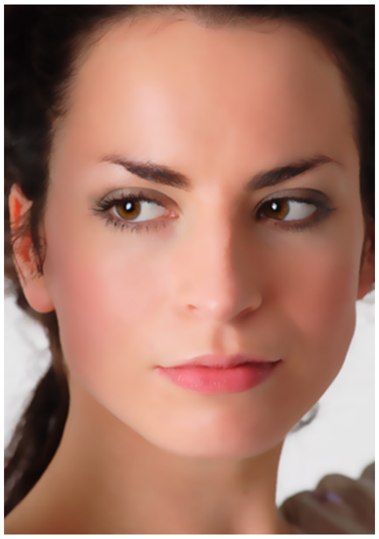
Image Space Stylization Result

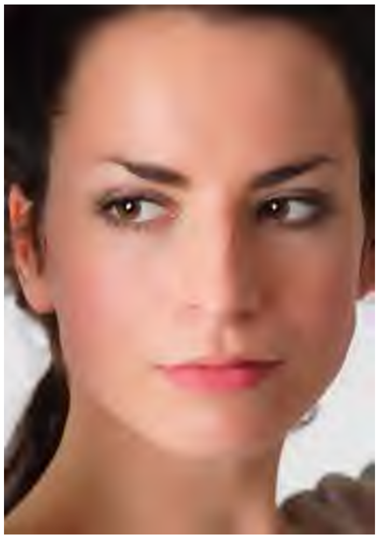

Compressed Stylization

2.1.3 Input Photograph


Greyscale Input Photograph


Compressed Greyscale Input

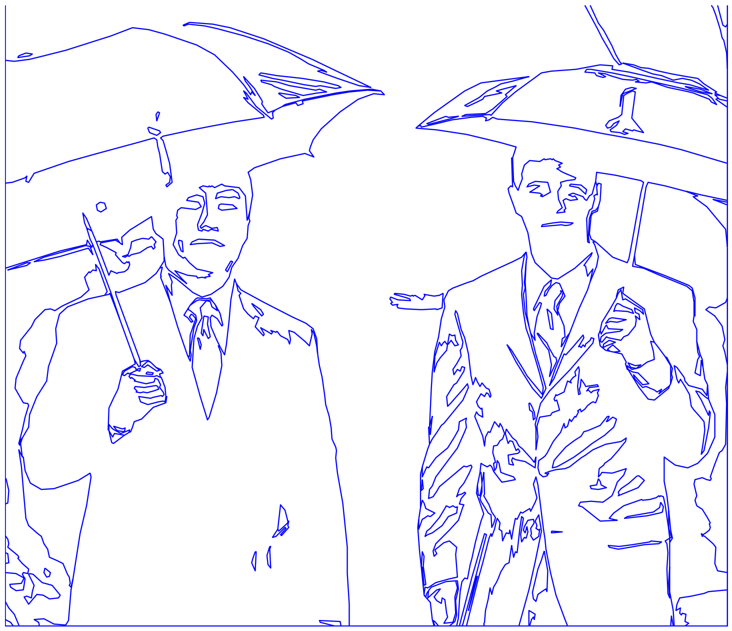

Input Photograph


Compressed Input Photograph

2.2 Two Student Example

For this example, the σ_c parameter has been changed to .3, from its default value of 1.6. This reduces the amount of smoothing in the stylized image, and thus causes the vector result to differ more significantly from the initial stylization. Reducing the amount of smoothing used in the stylization makes it much less likely that the reconstructed image will be capable of matching the stylized result—and particularly large differences between the vector reconstruction and input stylization are visible in this example.

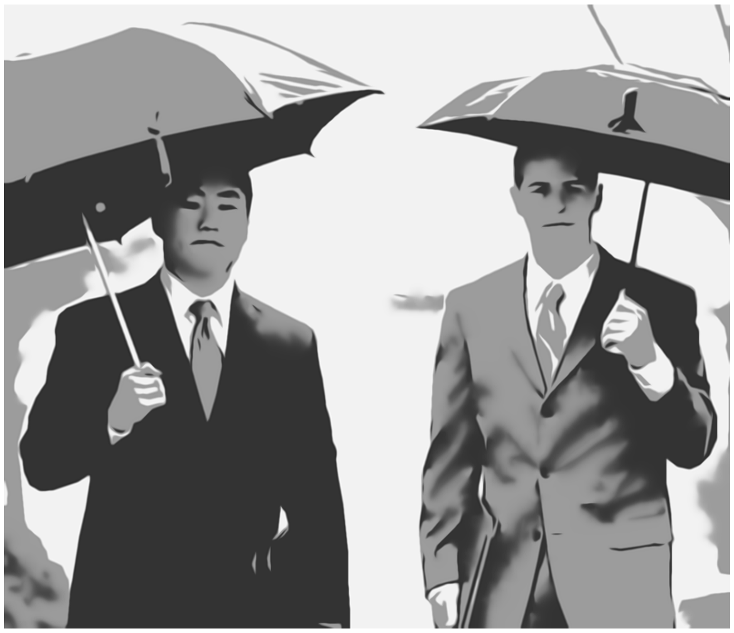
2.2.1 Vector Results

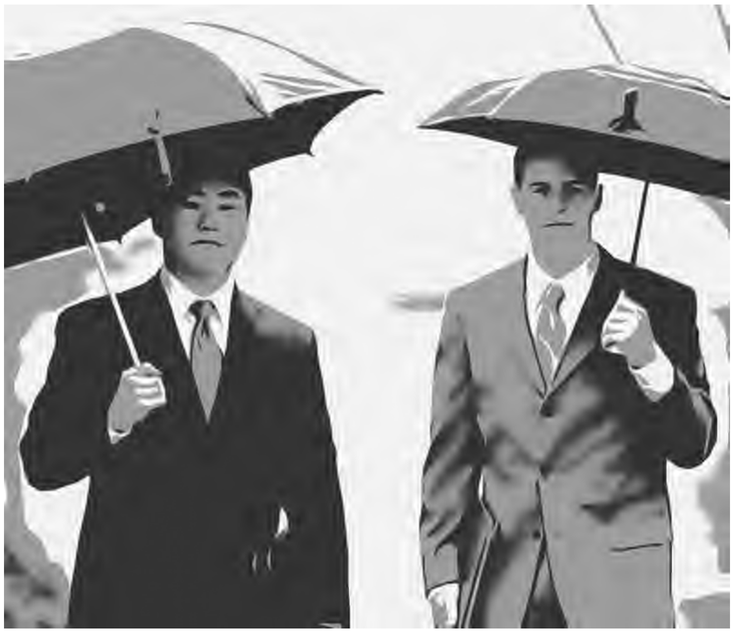

Edge Lines

Encoding Costs

Boundary Curves:	3.98kB (69%)
Edge Sharpness Data:	1.69kB (29%)
Luminance Data:	0.095kB (1.7%)

Total File Size: 5.76kB


Vector Reconstruction, 5.76kB


Compressed Reconstruction

2.2.2 Image Space Stylization Result

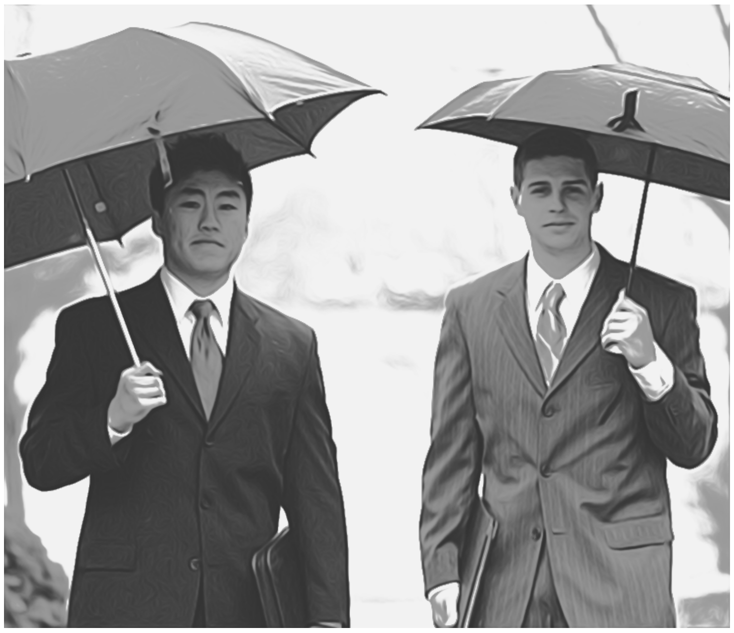
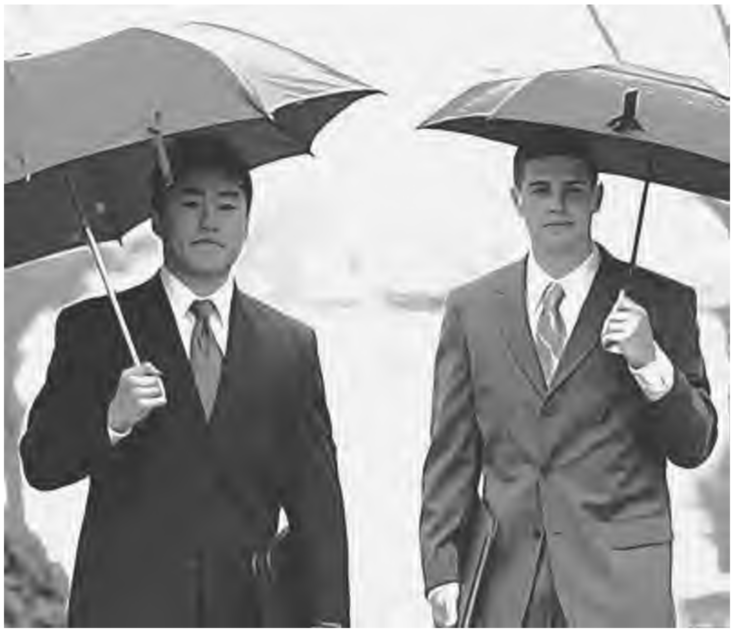
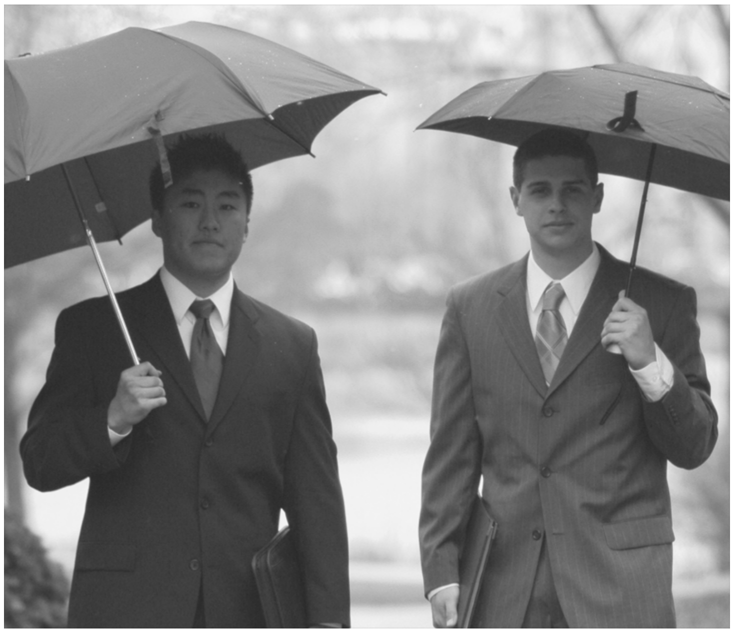
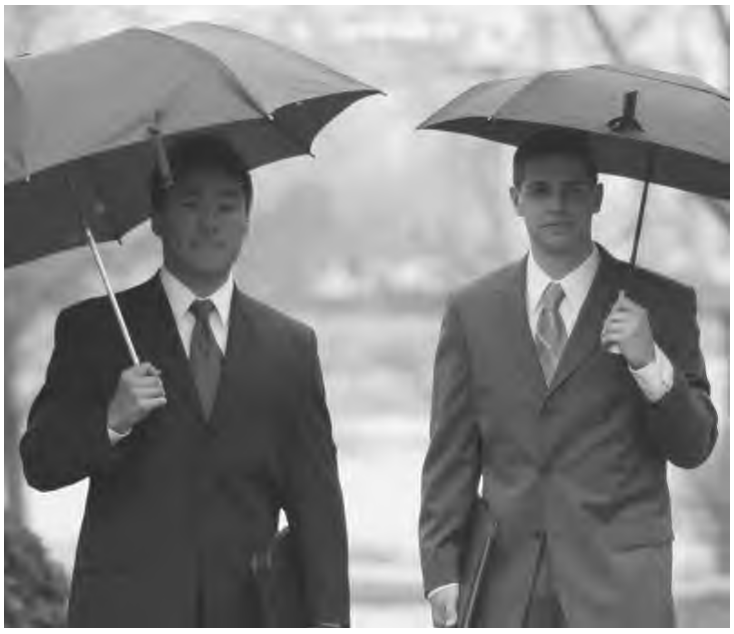


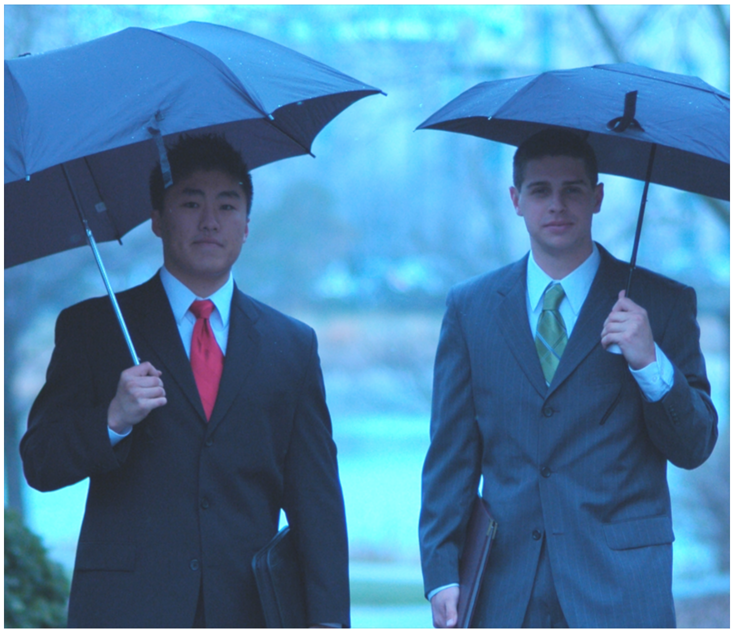
Image Space Stylization Result

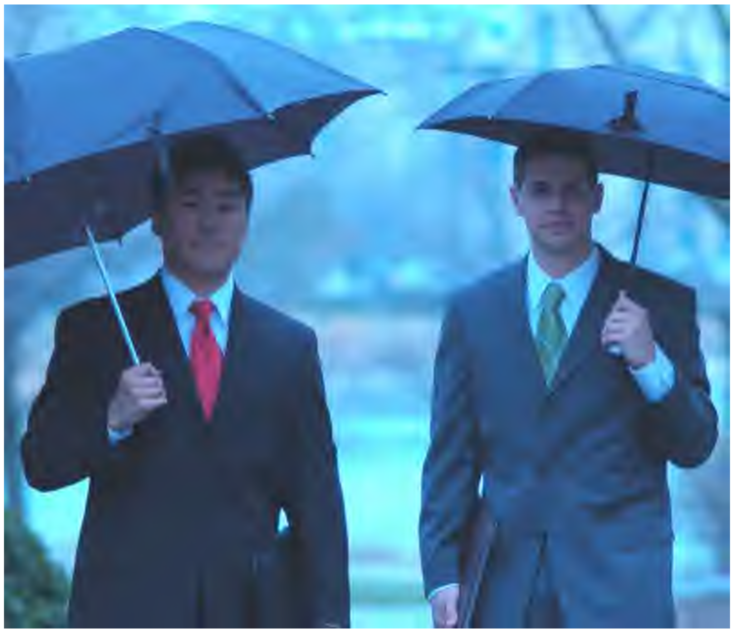

Compressed Stylization

2.2.3 Input Photograph


Greyscale Input Photograph

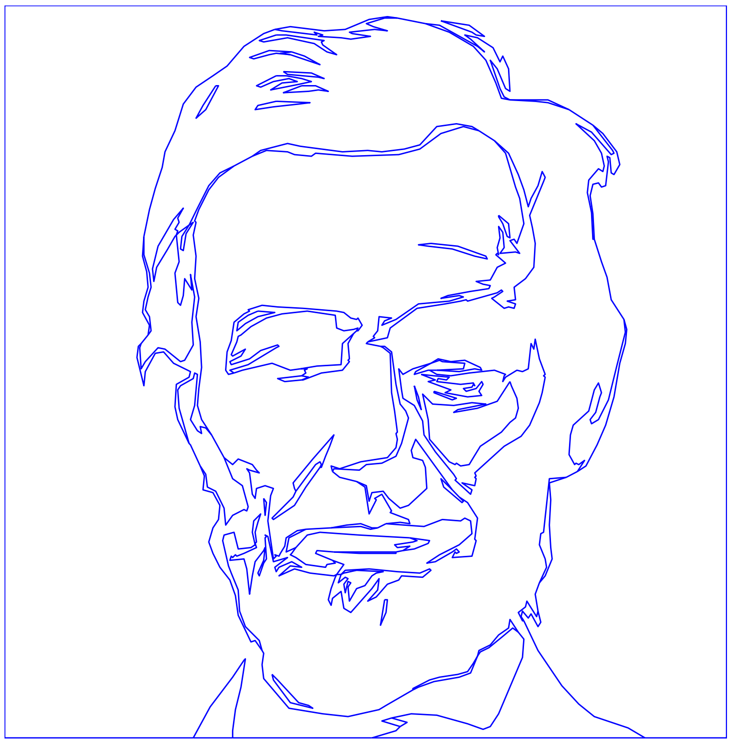

Compressed Greyscale Input


Input Photograph


Compressed Input Photograph

2.3 Historical Photograph #1

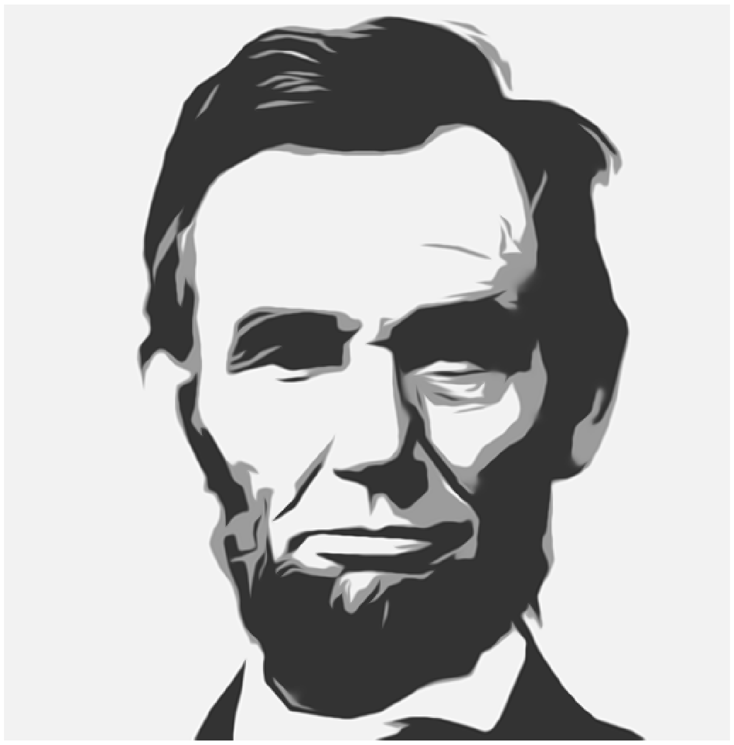
2.3.1 Vector Results

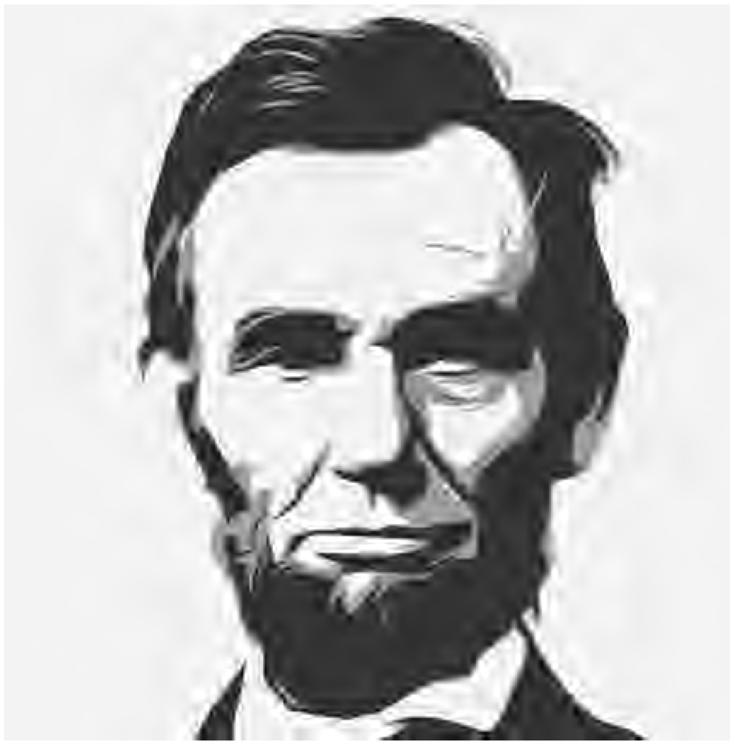

Edge Lines

Encoding Costs

Boundary Curves:	1.74kB (83%)
Edge Sharpness Data:	0.291kB (14%)
Luminance Data:	0.073kB (3.5%)

Total File Size: 2.11kB


Vector Reconstruction, 2.11kB


Compressed Reconstruction

2.3.2 Image Space Stylization Result

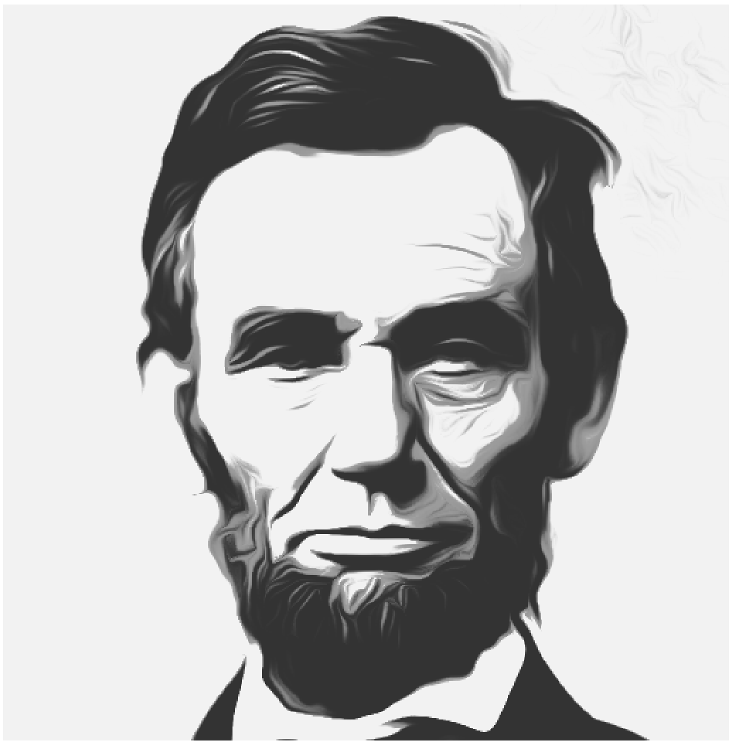
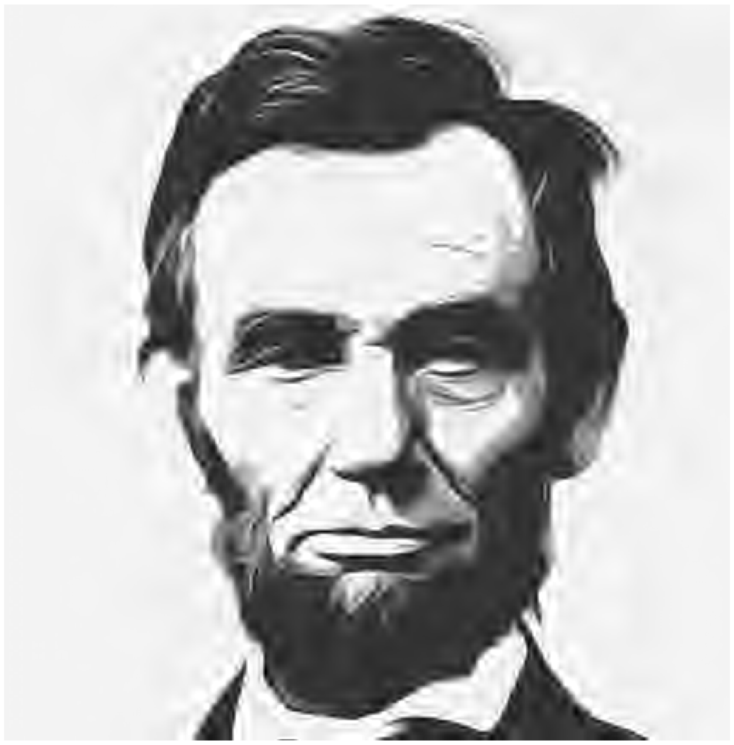
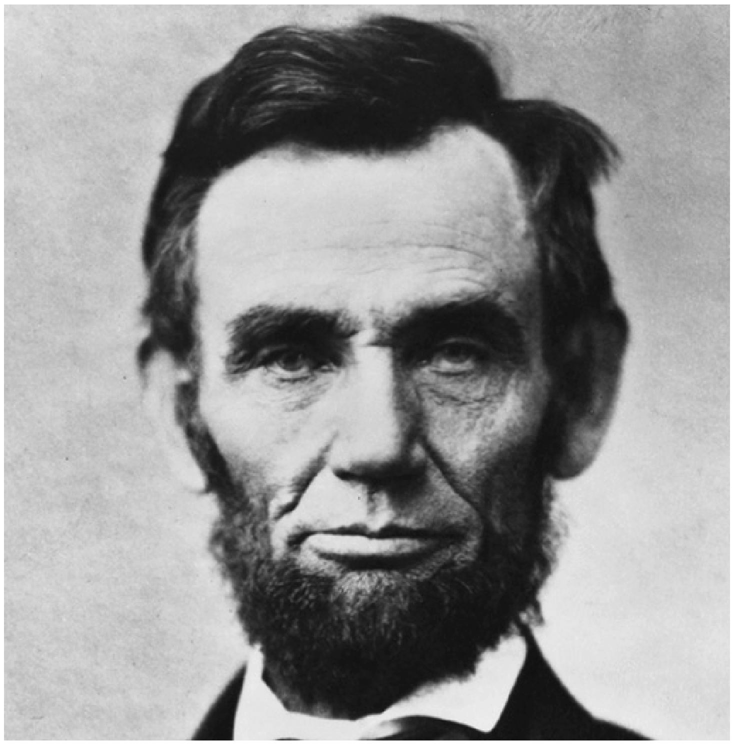


Image Space Stylization Result

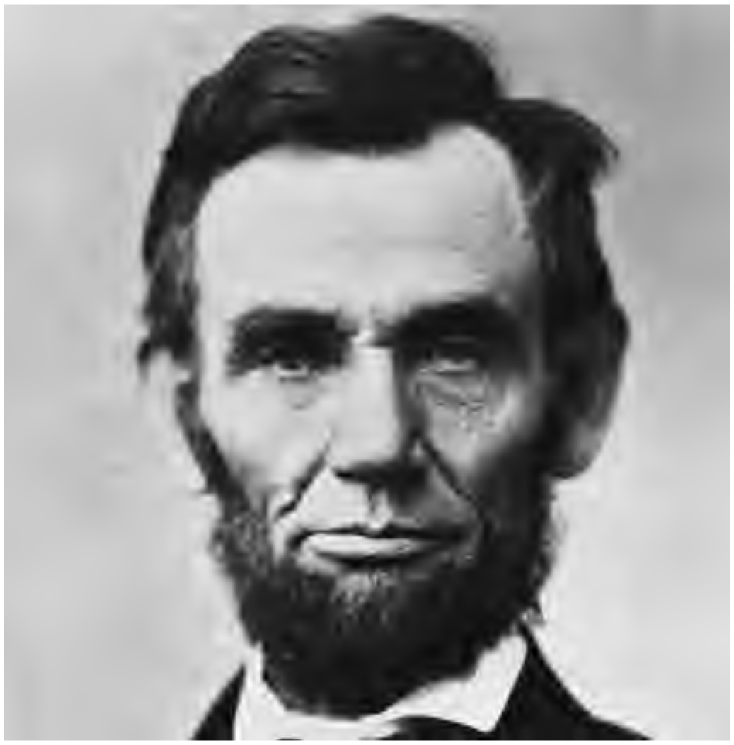


Compressed Stylization

2.3.3 Input Photograph



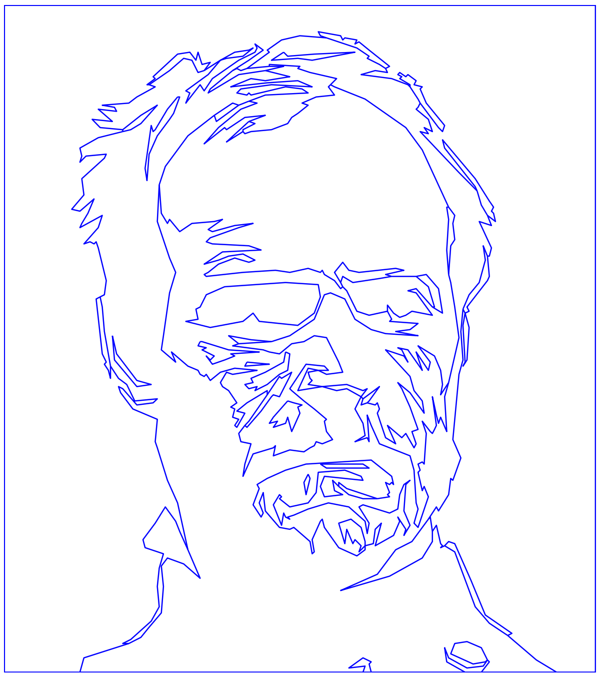
Input Photograph



Compressed Input Photograph

2.4 Historical Photograph #2

2.4.1 Vector Results

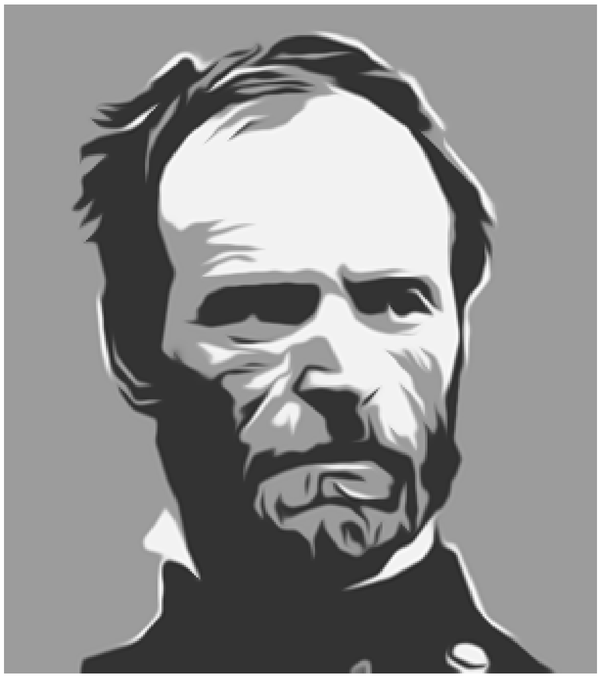


Edge Lines

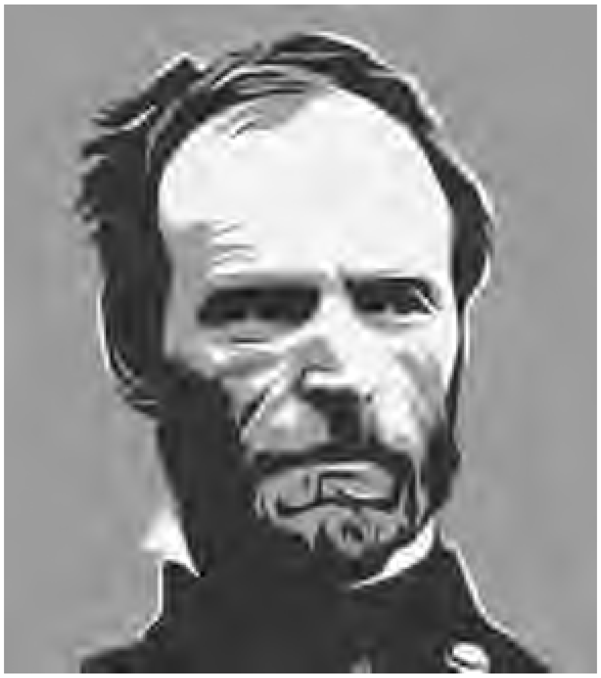
Encoding Costs

Boundary Curves:	1.34kB (83%)
Edge Sharpness Data:	0.212kB (13%)
Luminance Data:	0.065kB (4%)

Total File Size: 1.62kB



Vector Reconstruction, 1.62kB



Compressed Reconstruction

2.4.2 Image Space Stylization Result

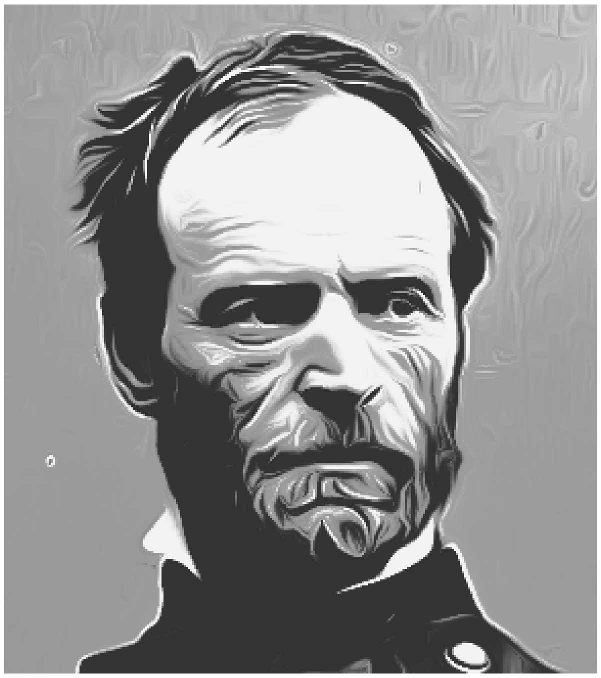
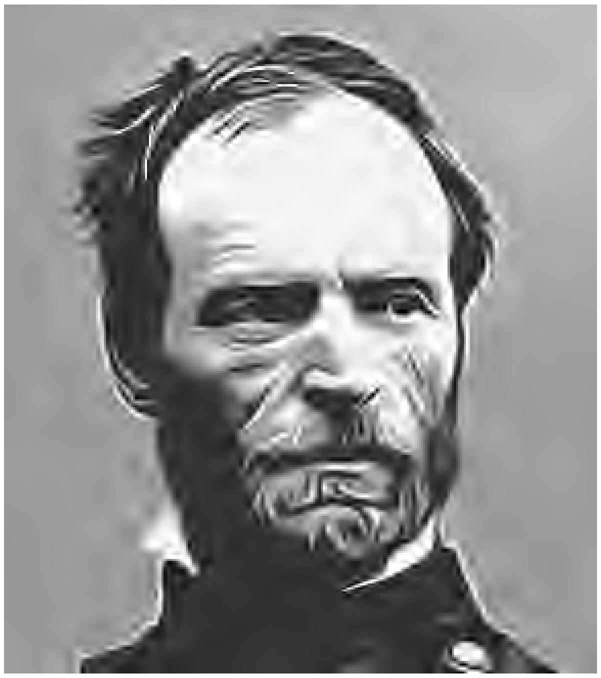
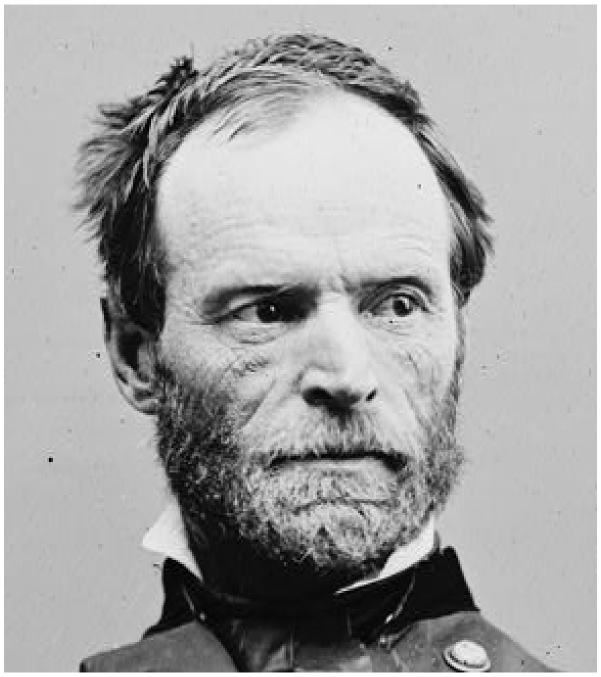
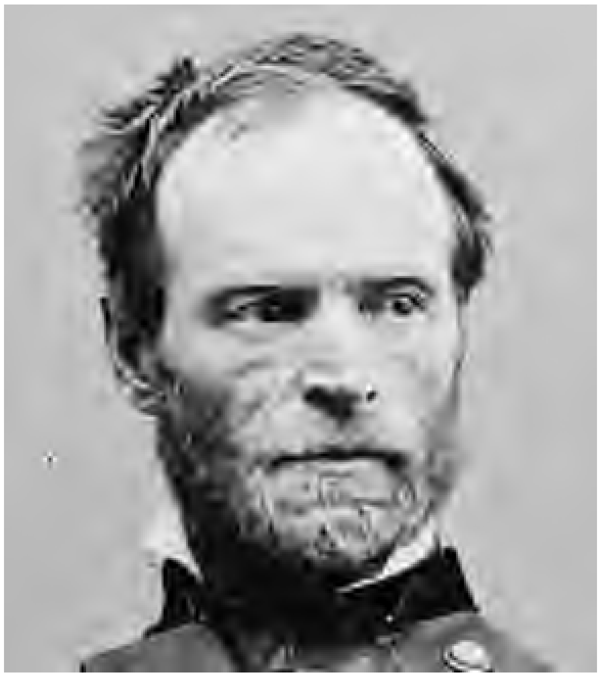


Image Space Stylization Result


Compressed Stylization

2.4.3 Input Photograph

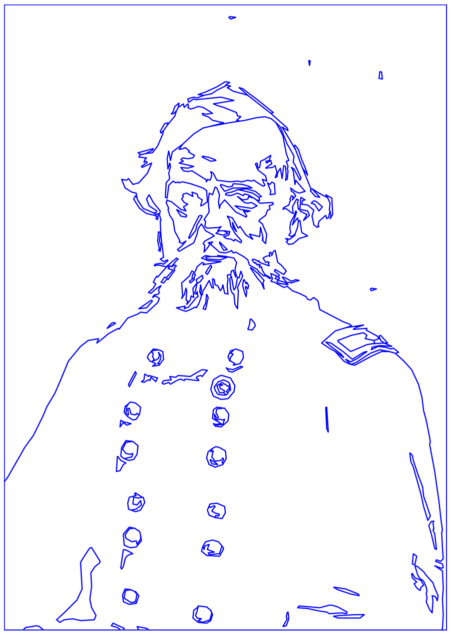

Input Photograph


Compressed Input Photograph

2.5 Historical Photograph #3

For this example, the bin values $\mathbf{b}_1$ have been changed from their default setting of $(.45, .75, .85)$ to $(.3, .65, .85)$, while the σ_c parameter has been changed to .3, from its default value of 1.6.

2.5.1 Vector Results

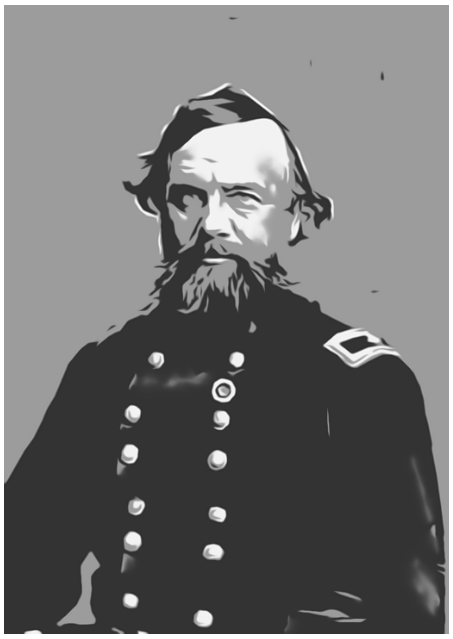


Edge Lines

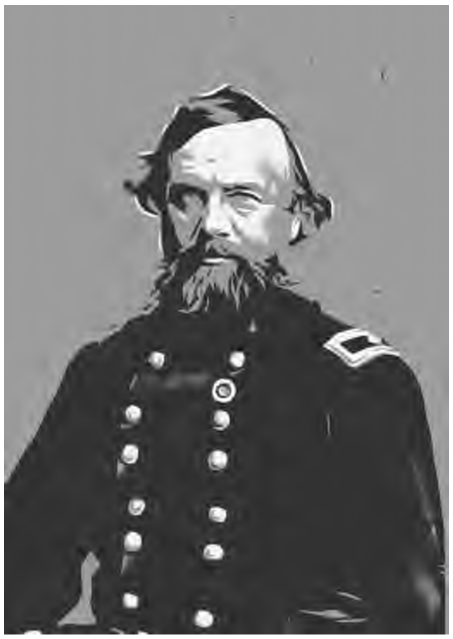
Encoding Costs

Boundary Curves:	2.34kB (79%)
Edge Sharpness Data:	0.536kB (18%)
Luminance Data:	0.08kB (2.7%)

Total File Size: 2.96kB



Vector Reconstruction, 2.96kB



Compressed Reconstruction

2.5.2 Image Space Stylization Result

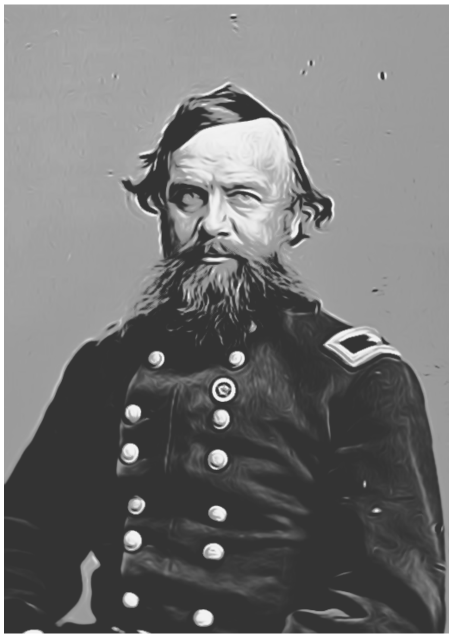
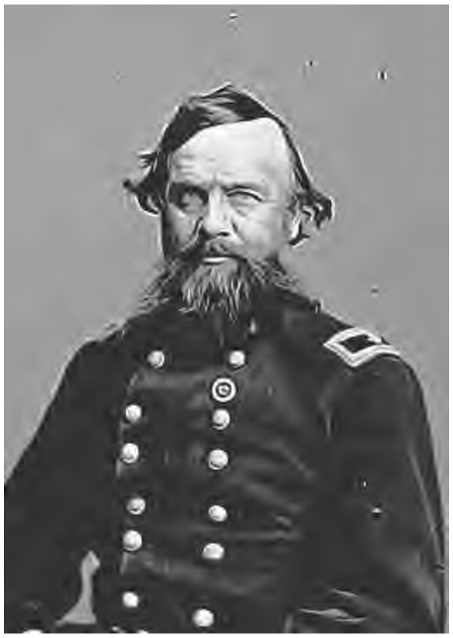
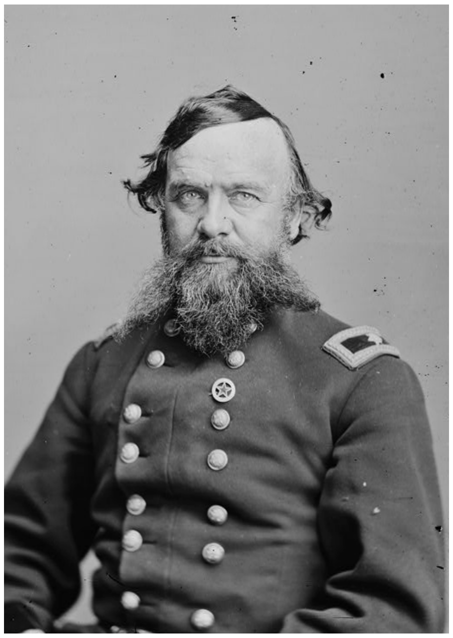


Image Space Stylization Result


Compressed Stylization

2.5.3 Input Photograph

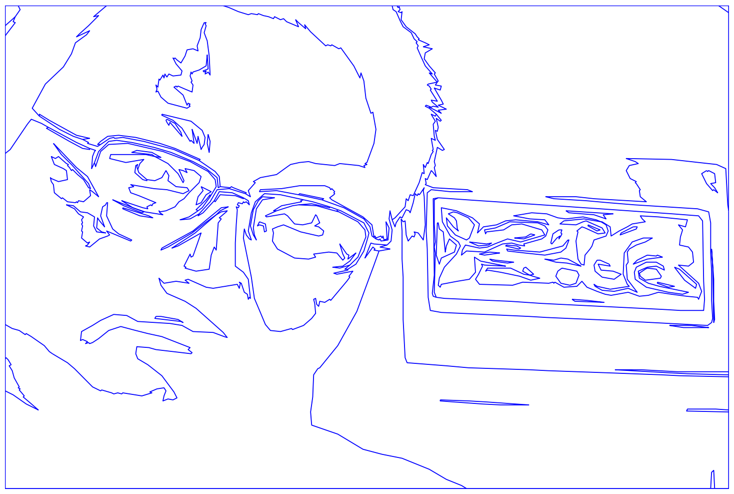

Input Photograph

Compressed Input Photograph

2.6 Gallery Example

This is an example that illustrates the difficulty of distinguishing between "success" and "failure" cases. At 3.45kB, kdu is capable of generating relatively faithful versions of the input photograph – though some significant artifacts are visible. The vector reconstruction, meanwhile, is not very faithful to the stylization result – significantly altering important features such as the mouth, glasses, and eyebrows. Despite this, the vector reconstruction is arguably an effective "abstraction" of the input photograph. Interestingly, the nonlinearities introduced by process of stylization and vectorization lead to a greyscale image that is significantly less amenable to compression than the source. Thus, while the greyscale source image shows relatively few compression artifacts, the greyscale abstraction is clearly more difficult to encode accurately.

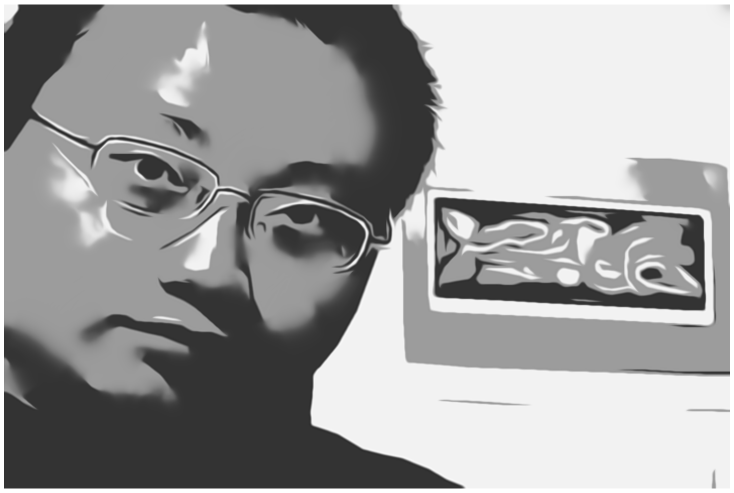
2.6.1 Vector Results

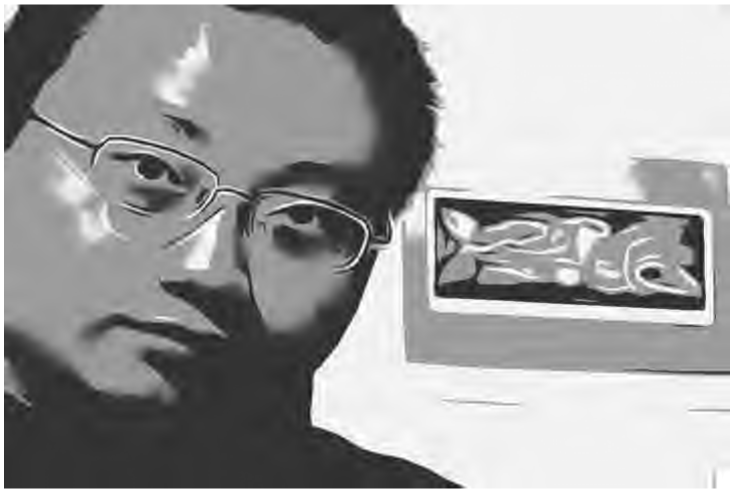

Edge Lines

Encoding Costs

Boundary Curves:	2.44kB (71%)
Edge Sharpness Data:	0.929kB (27%)
Luminance Data:	0.076kB (2.2%)

Total File Size: 3.45kB


Vector Reconstruction, 3.45kB


Compressed Reconstruction

2.6.2 Image Space Stylization Result

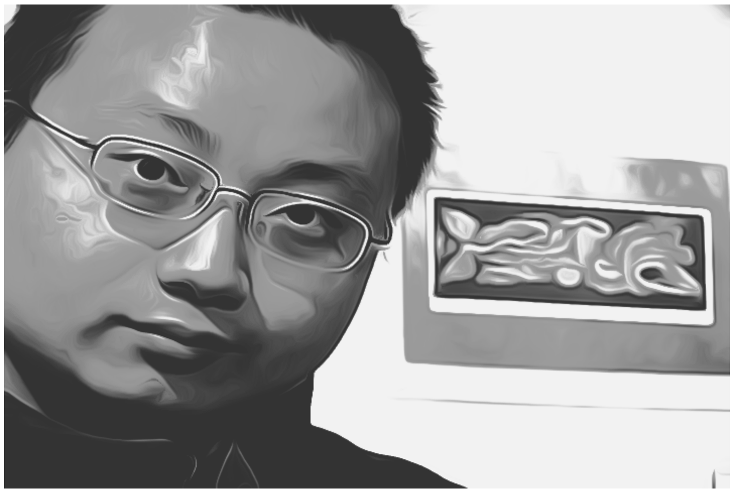
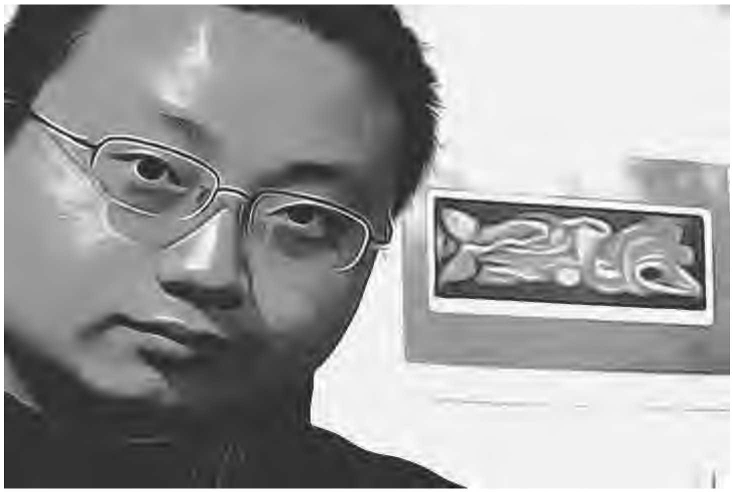


Image Space Stylization Result


Compressed Stylization

2.6.3 Input Photograph

Greyscale Input Photograph

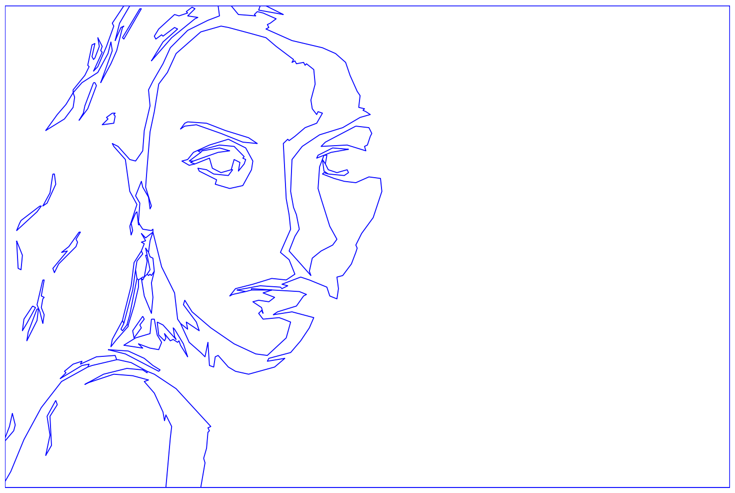
Compressed Greyscale Input

Input Photograph

Compressed Input Photograph

2.7 Girl in Shadow

2.7.1 Vector Results

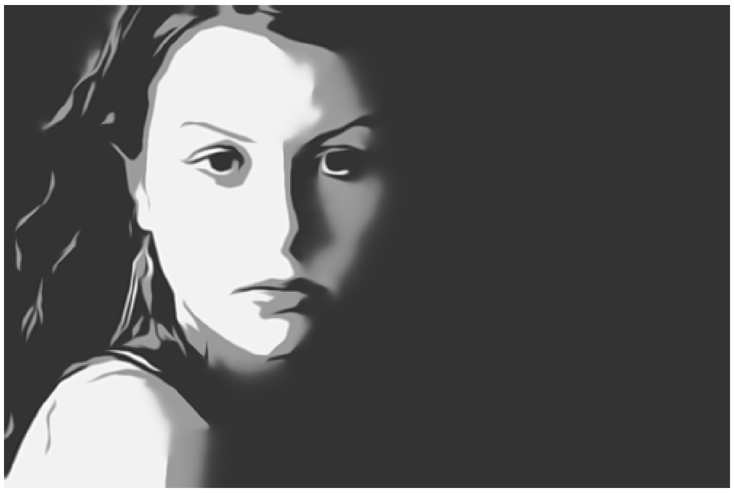


Edge Lines

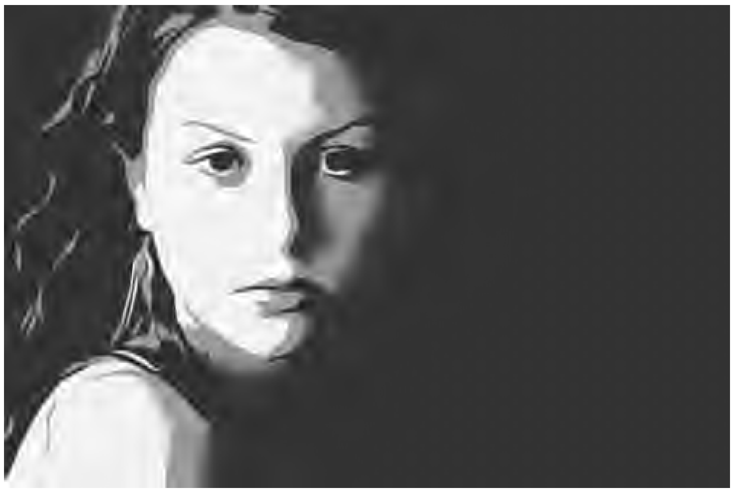
Encoding Costs

Boundary Curves:	1.06kB (69%)
Edge Sharpness Data:	0.417kB (27%)
Luminance Data:	0.054kB (3.5%)

Total File Size: 1.53kB



Vector Reconstruction, 1.53kB



Compressed Reconstruction

2.7.2 Image Space Stylization Result

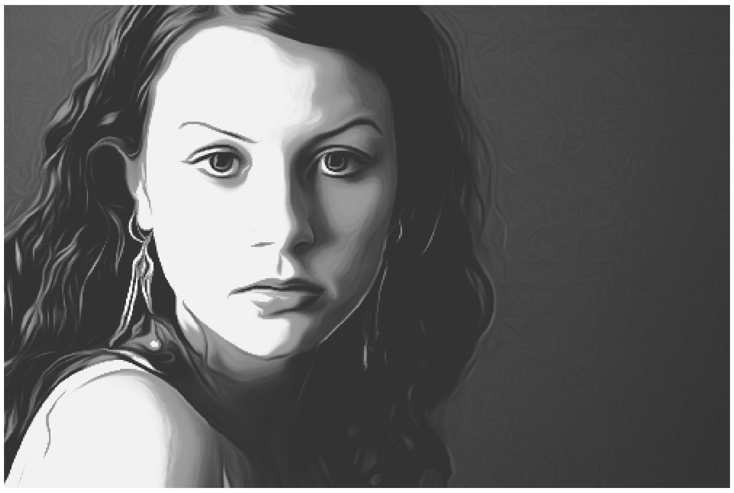
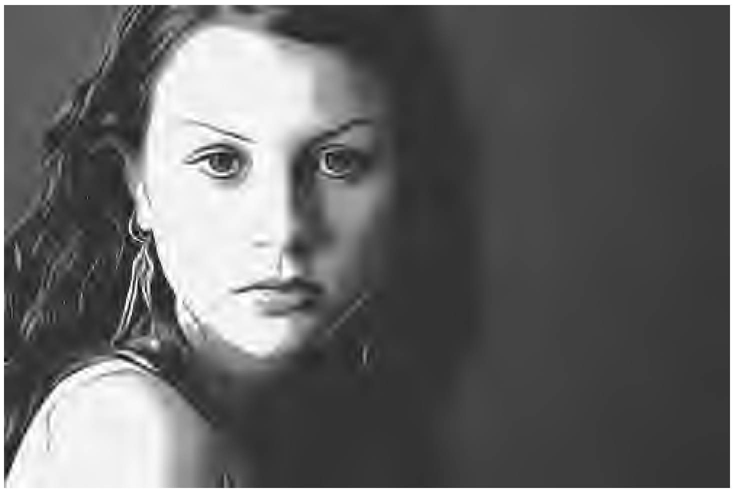
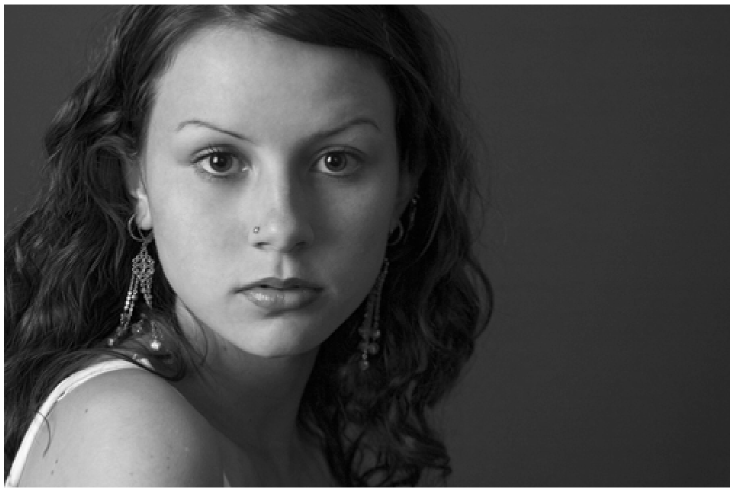


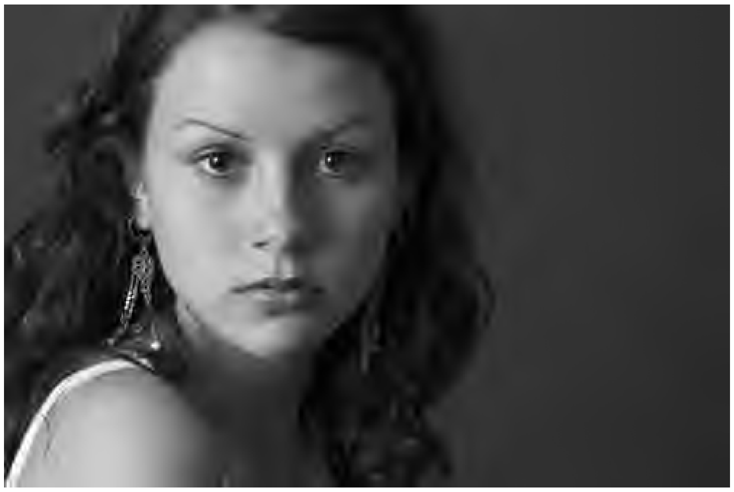
Image Space Stylization Result



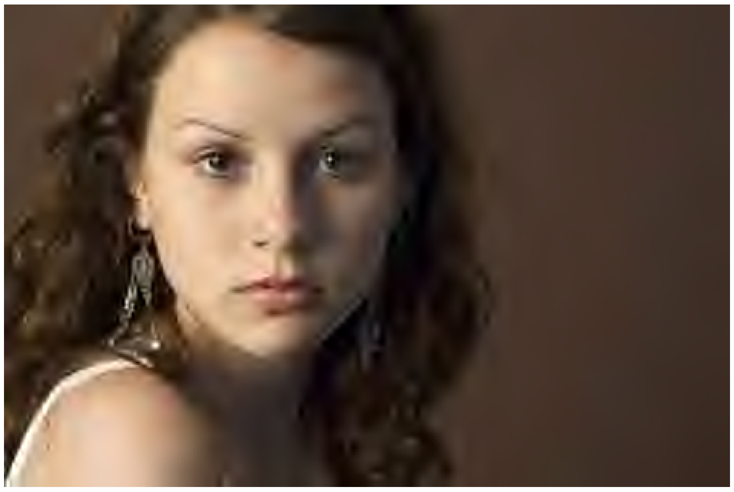
Compressed Stylization

2.7.3 Input Photograph


Greyscale Input Photograph

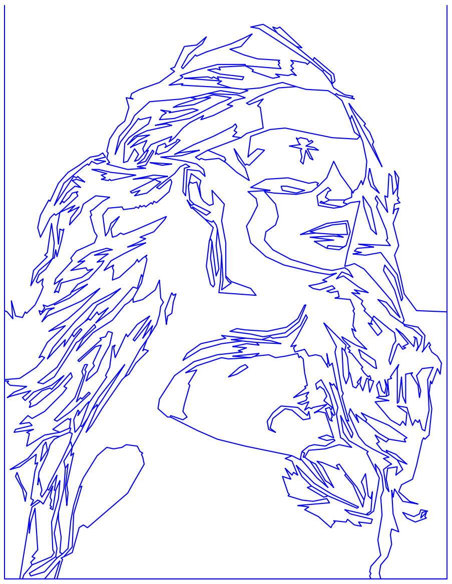

Compressed Greyscale Input

Input Photograph


Compressed Input Photograph

2.8 Sunglasses Example

2.8.1 Vector Results

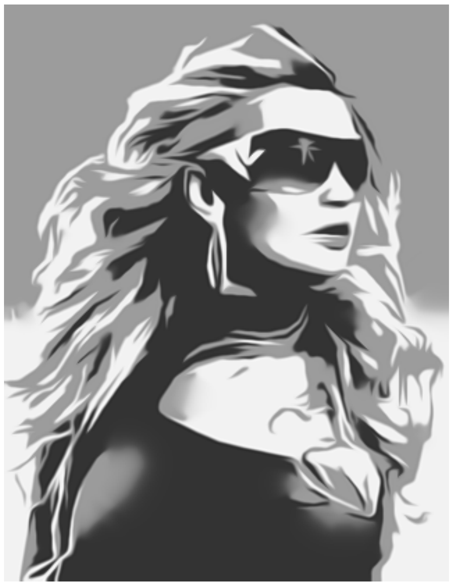


Edge Lines

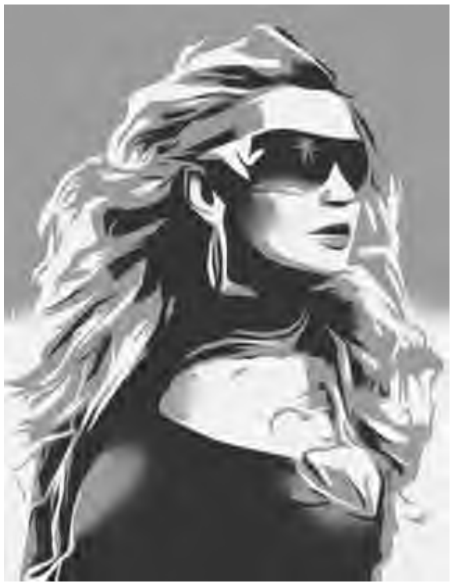
Encoding Costs

Boundary Curves: 2.73kB (80%)
Edge Sharpness Data: 0.595kB (17%)
Luminance Data: 0.108kB (3.1%)

Total File Size: 3.43kB



Vector Reconstruction, 3.43kB



Compressed Reconstruction

2.8.2 Image Space Stylization Result

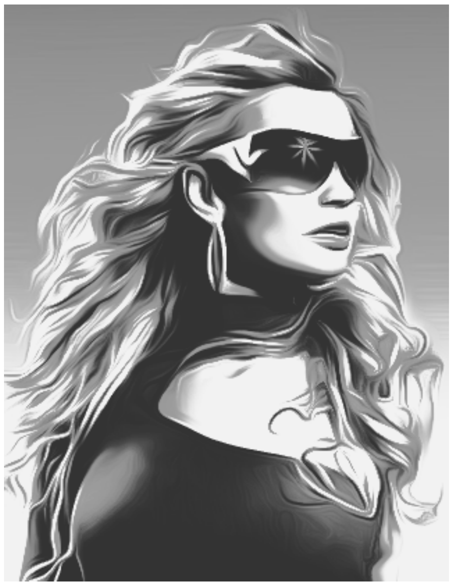
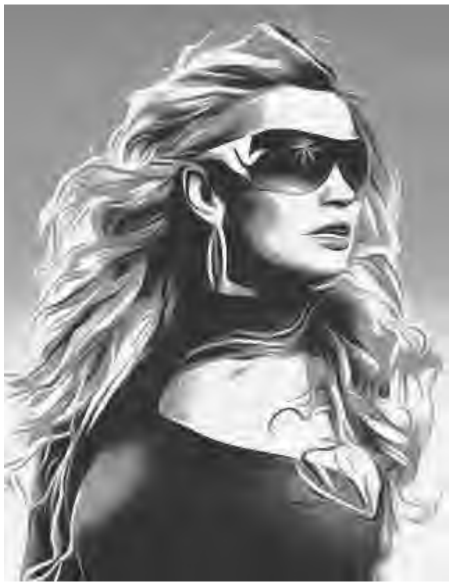
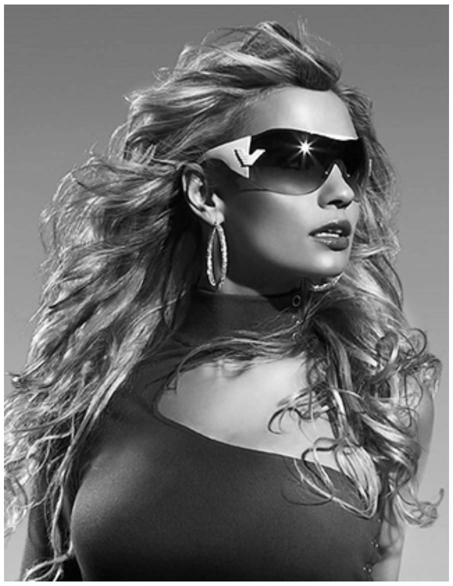
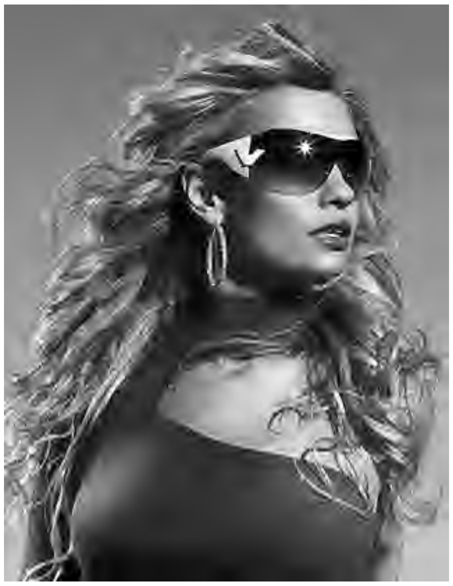


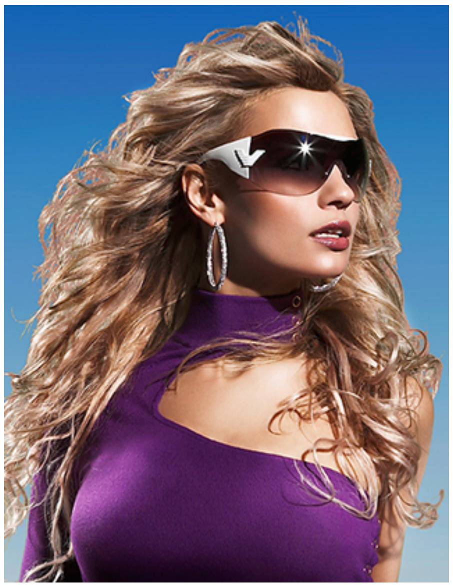
Image Space Stylization Result

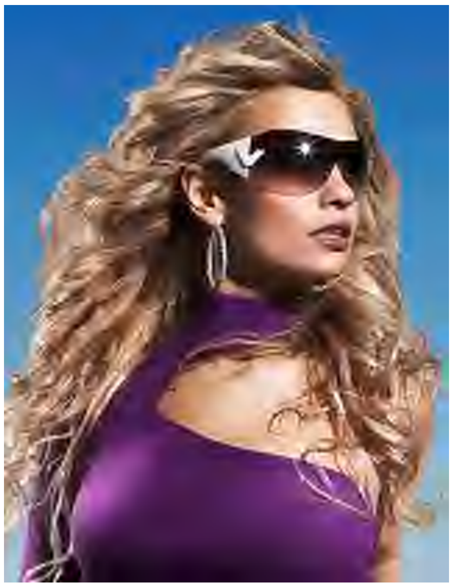

Compressed Stylization

2.8.3 Input Photograph


Greyscale Input Photograph


Compressed Greyscale Input

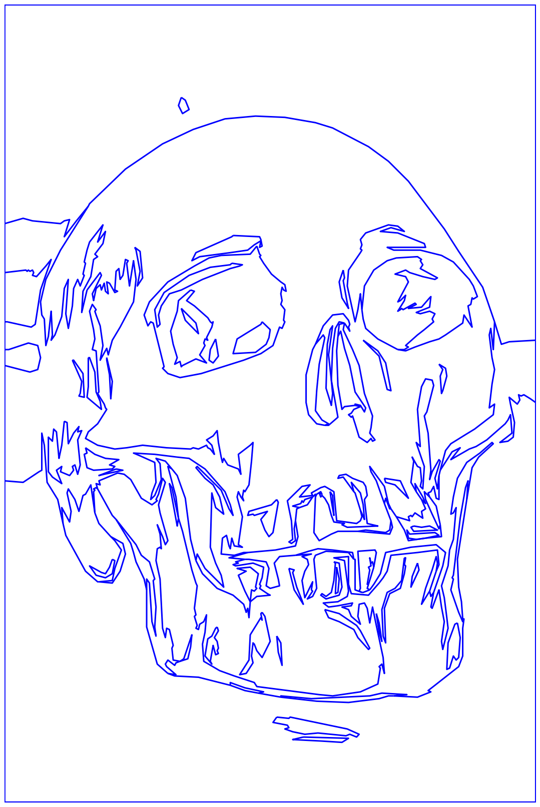

Input Photograph


Compressed Input Photograph

2.9 Skull Image

For this example, the bin values $\mathbf{b}_1$ have been changed from their default setting of $(.45, .75, .85)$ to $(.5, .85, .9)$.

2.9.1 Vector Results

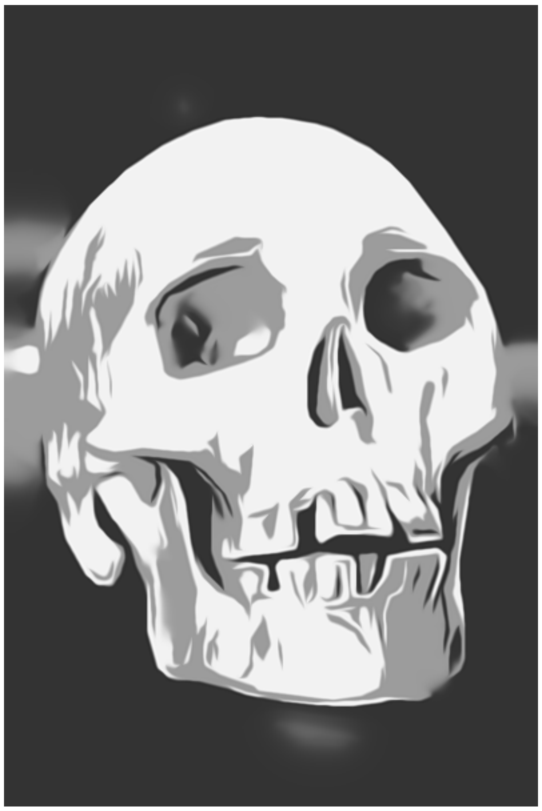


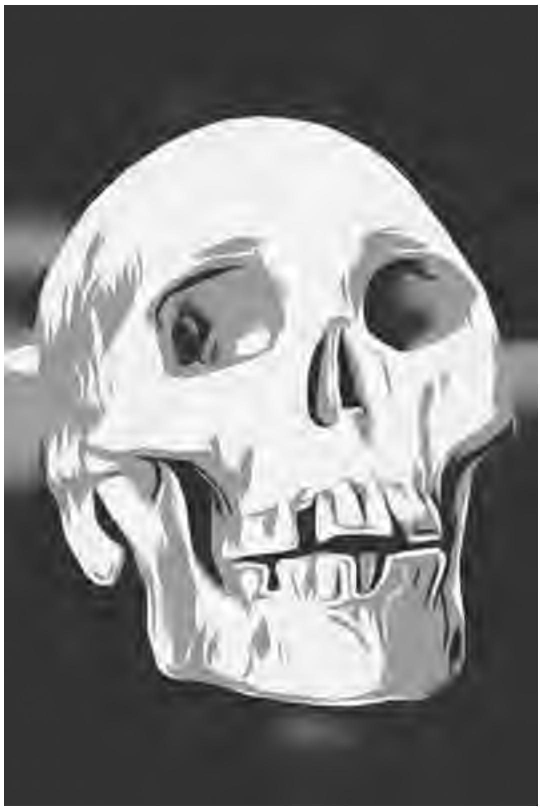
Edge Lines

Encoding Costs

Boundary Curves: 2.19kB (77%)
Edge Sharpness Data: 0.581kB (20%)
Luminance Data: 0.074kB (2.6%)

Total File Size: 2.84kB


Vector Reconstruction, 2.84kB


Compressed Reconstruction

2.9.2 Image Space Stylization Result

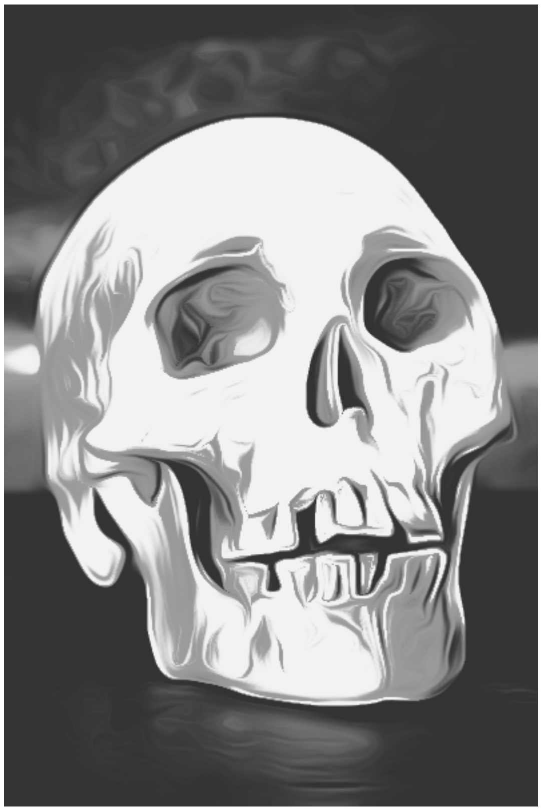
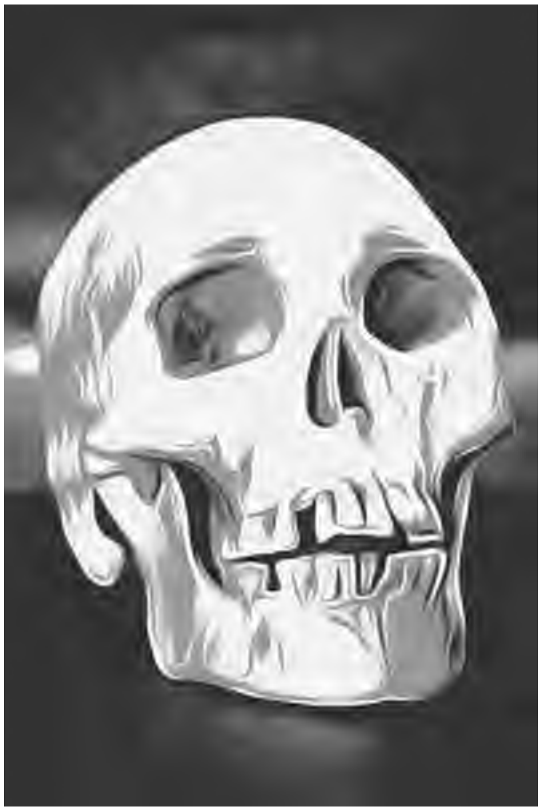
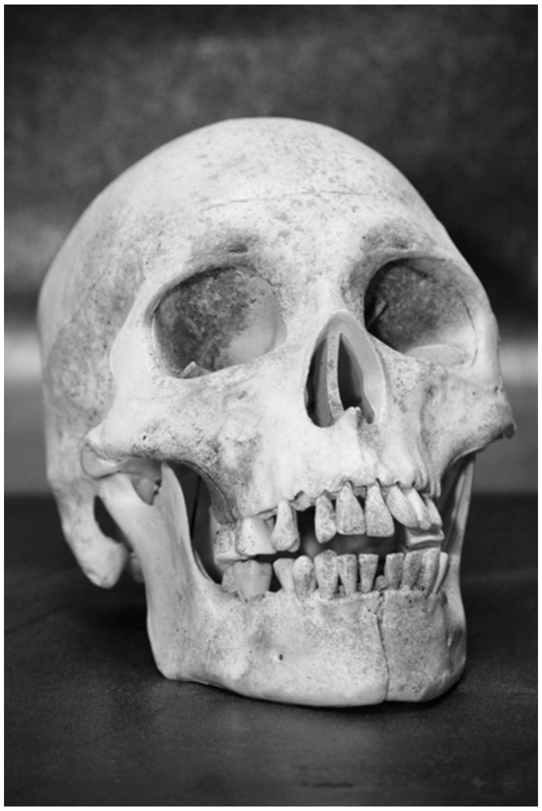
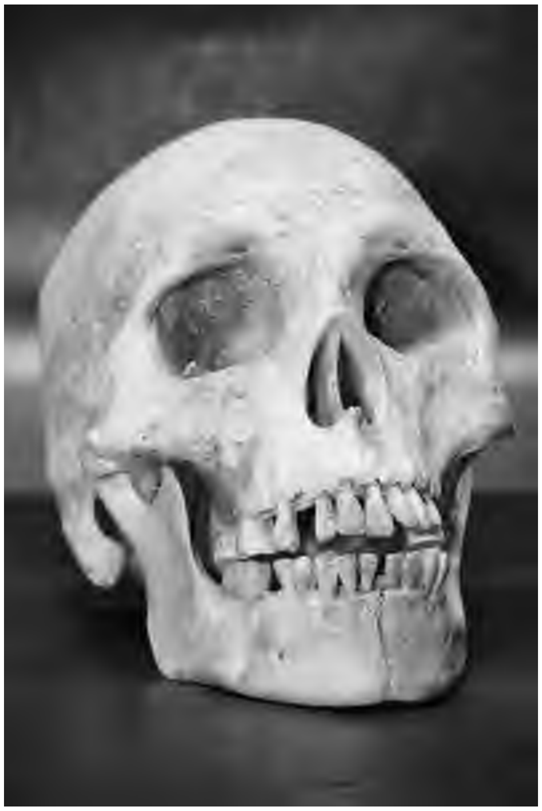


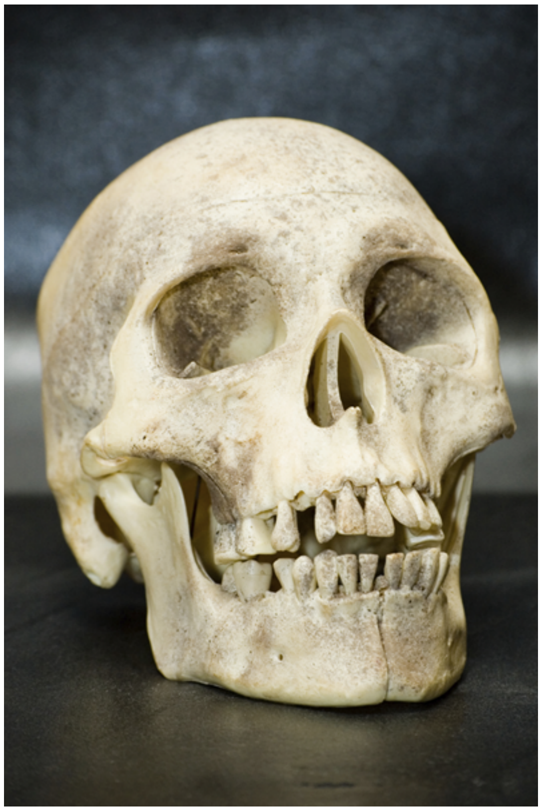
Image Space Stylization Result

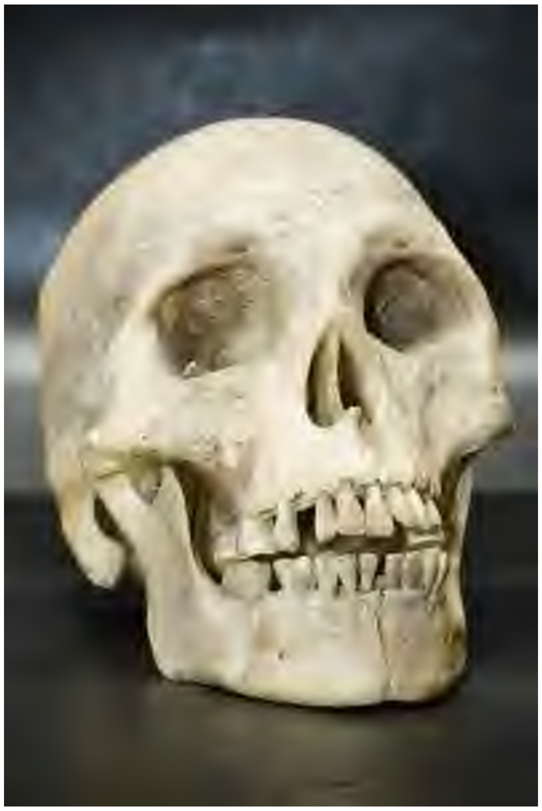

Compressed Stylization

2.9.3 Input Photograph


Greyscale Input Photograph


Compressed Greyscale Input

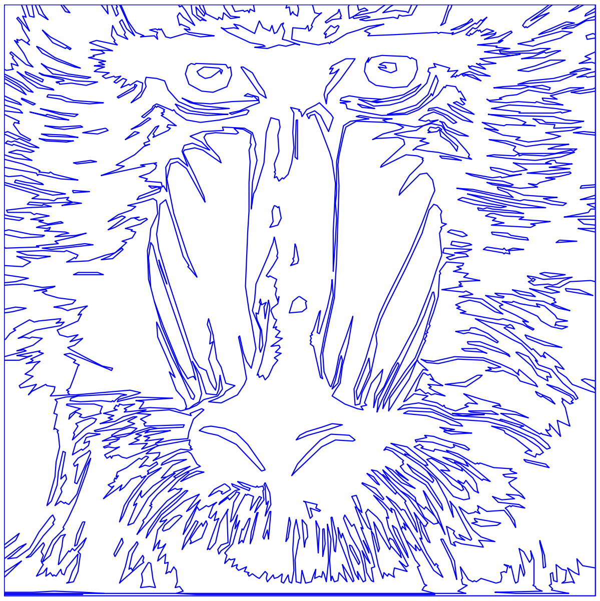

Input Photograph


Compressed Input Photograph

2.10 Mandrill Test Image

For this example, the bin values $\mathbf{b}_1$ have been changed from their default setting of $(.45, .75, .85)$ to $(.6, .8, .85)$.

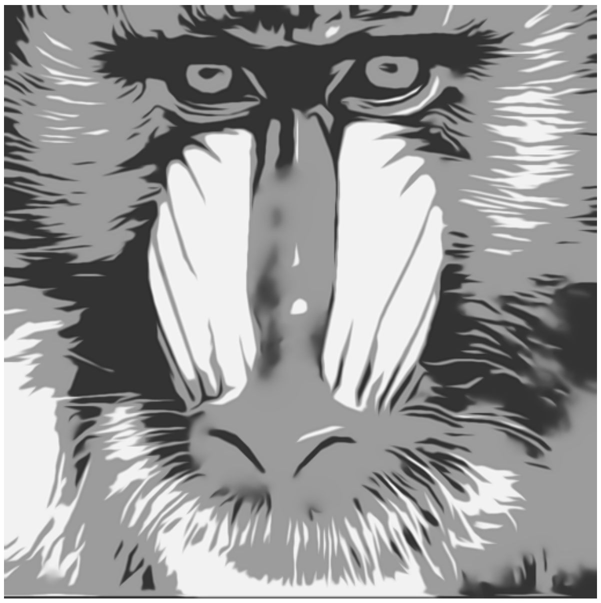
2.10.1 Vector Results

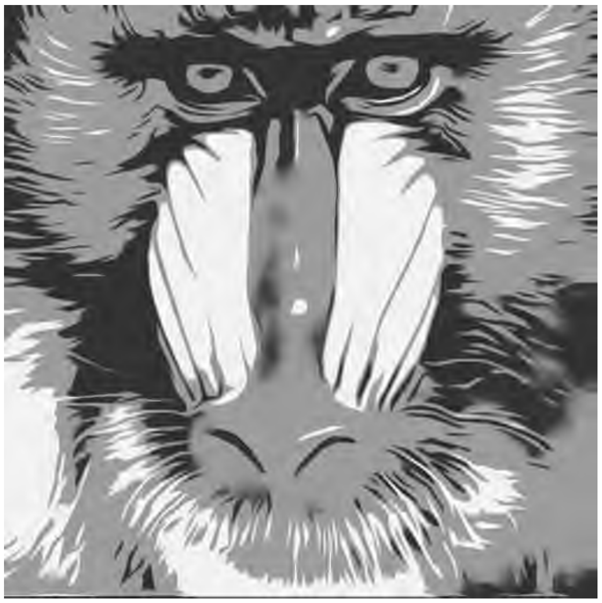

Edge Lines

Encoding Costs

Boundary Curves: 5.92kB (79%)
Edge Sharpness Data: 1.41kB (19%)
Luminance Data: 0.196kB (2.6%)

Total File Size: 7.53kB


Vector Reconstruction, 7.53kB


Compressed Reconstruction

2.10.2 Image Space Stylization Result

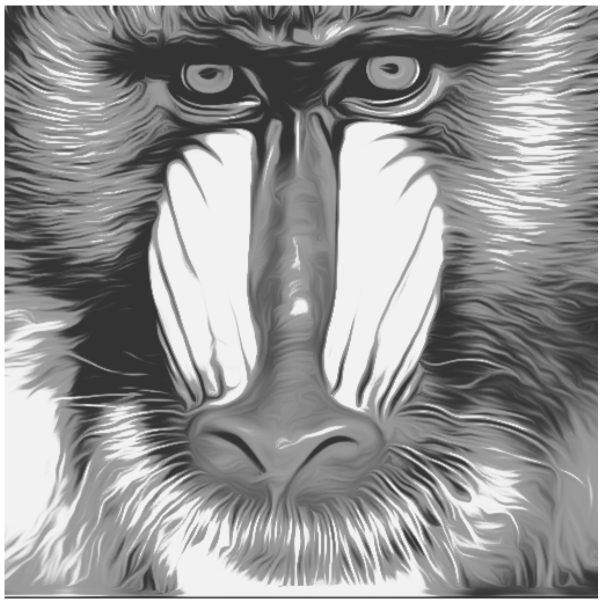
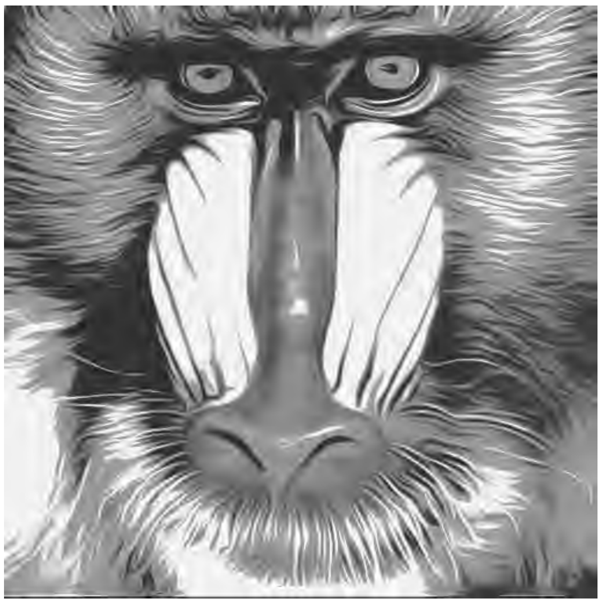
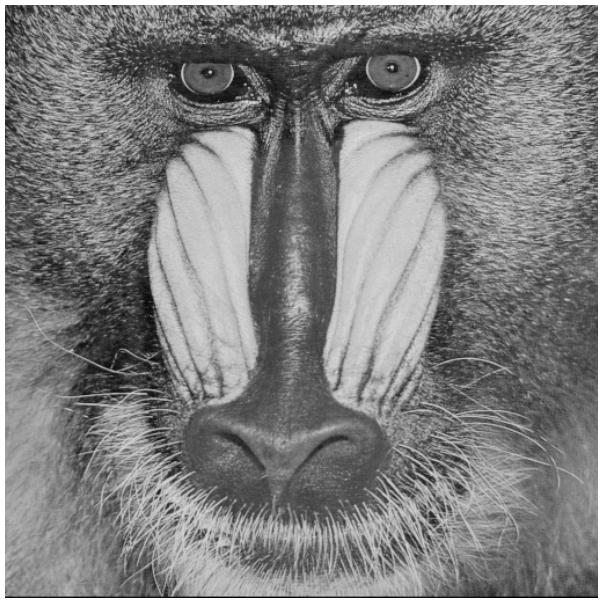
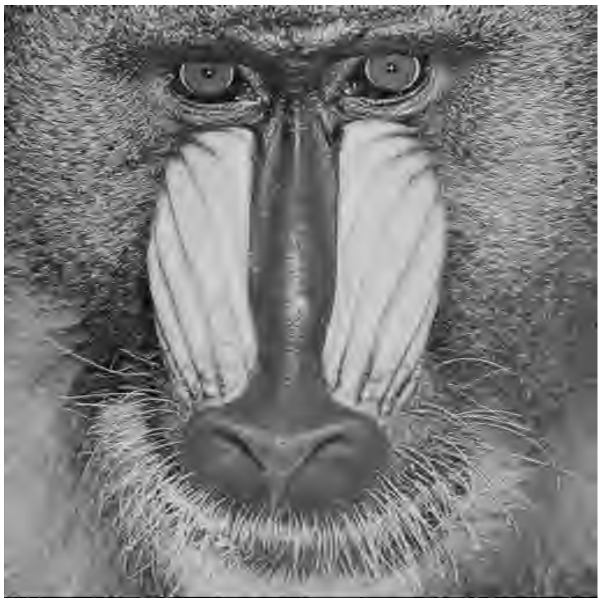


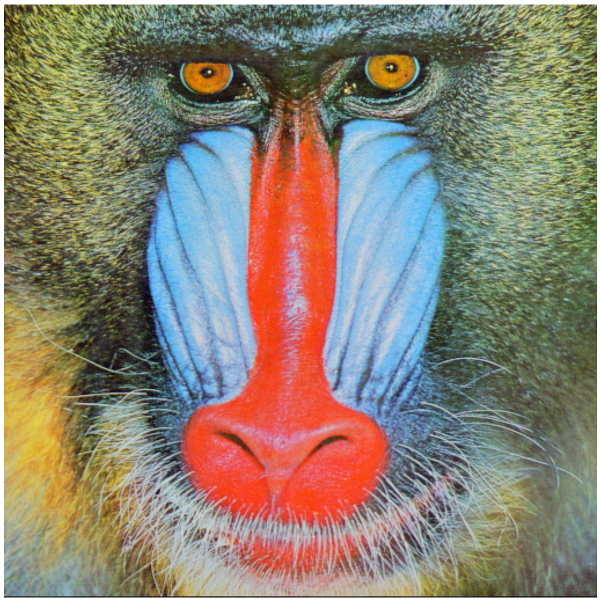
Image Space Stylization Result

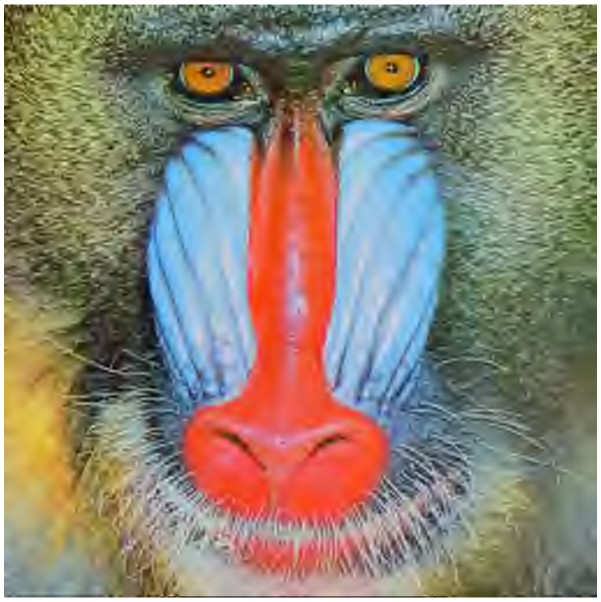

Compressed Stylization

2.10.3 Input Photograph


Greyscale Input Photograph


Compressed Greyscale Input

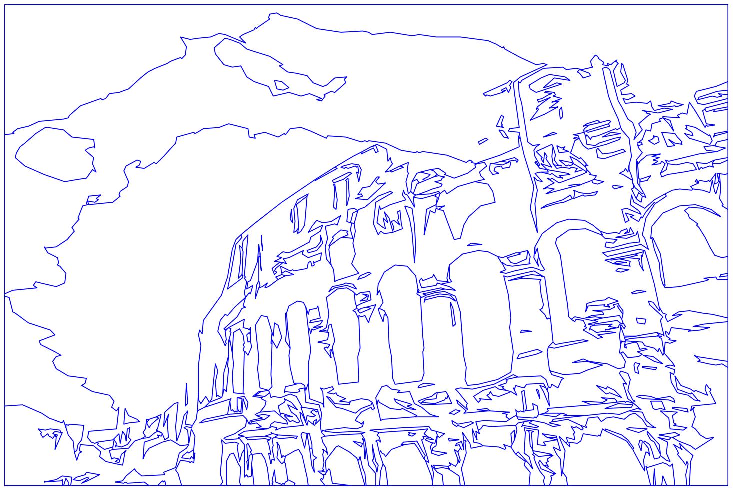

Input Photograph


Compressed Input Photograph

2.11 Architecture Example

For this example, the bin values $\mathbf{b}_1$ have been changed from their default setting of $(.45, .75, .85)$ to $(.6, .8, .9)$, while the σ_c parameter has been changed to .8, from its default value of 1.6.

2.11.1 Vector Results

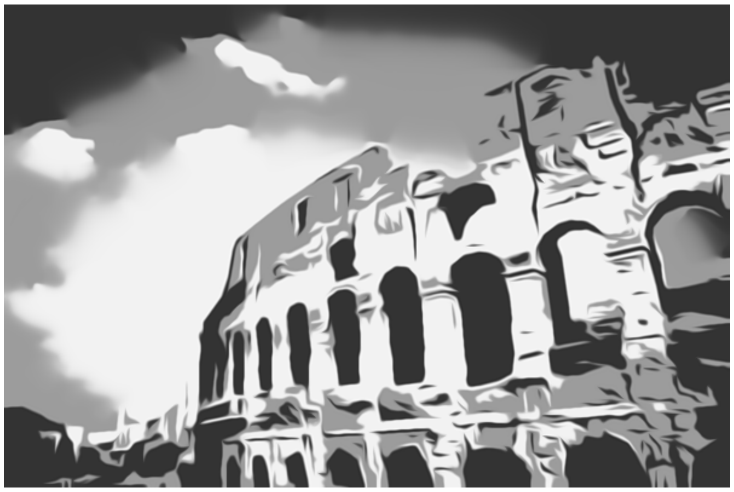


Edge Lines

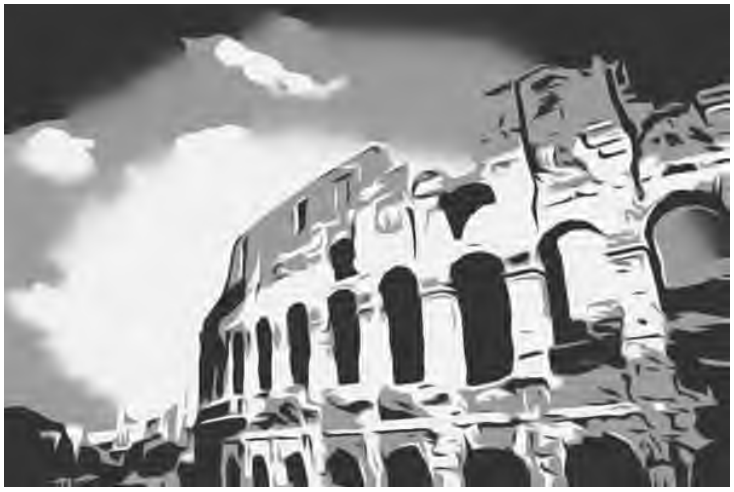
Encoding Costs

Boundary Curves:	5.54kB (89%)
Edge Sharpness Data:	0.532kB (8.6%)
Luminance Data:	0.138kB (2.2%)

Total File Size: 6.21kB



Vector Reconstruction, 6.21kB



Compressed Reconstruction

2.11.2 Image Space Stylization Result

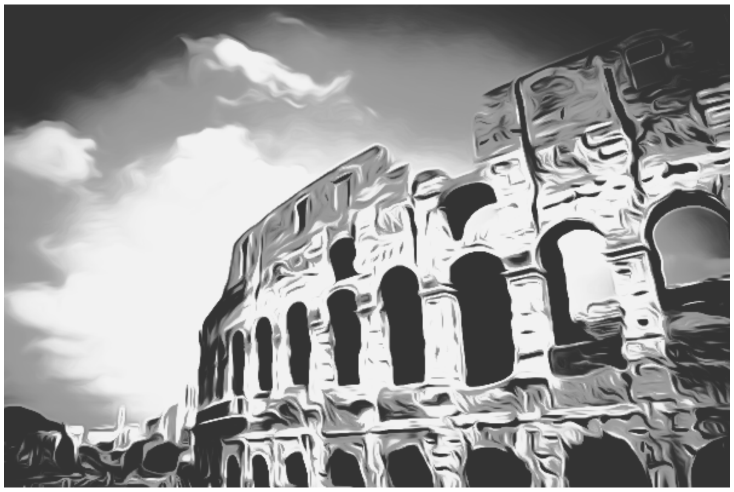


Image Space Stylization Result

Compressed Stylization

2.11.3 Input Photograph

Greyscale Input Photograph

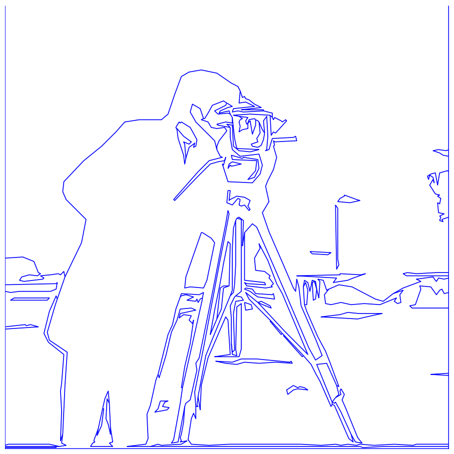
Compressed Greyscale Input

Input Photograph

Compressed Input Photograph

2.12 Cameraman Example

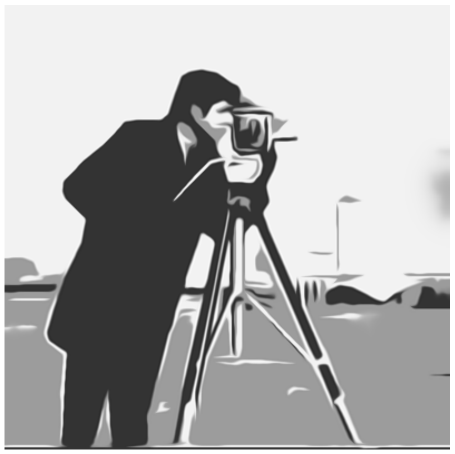
2.12.1 Vector Results

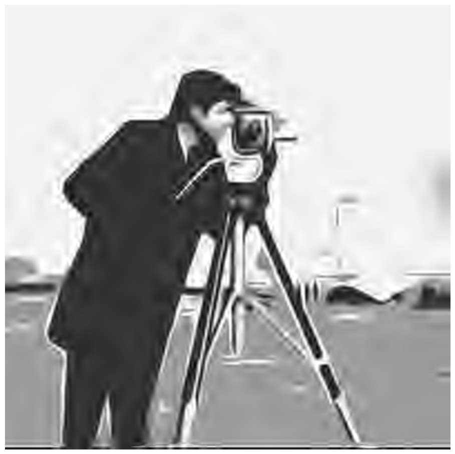

Edge Lines

Encoding Costs

Boundary Curves: 1.51kB (83%)
Edge Sharpness Data: 0.257kB (14%)
Luminance Data: 0.057kB (3.1%)

Total File Size: 1.82kB


Vector Reconstruction, 1.82kB


Compressed Reconstruction

2.12.2 Image Space Stylization Result

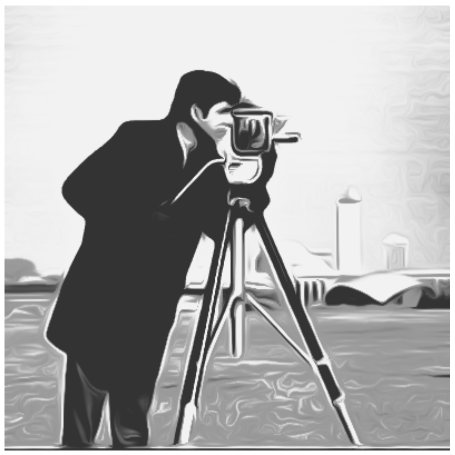
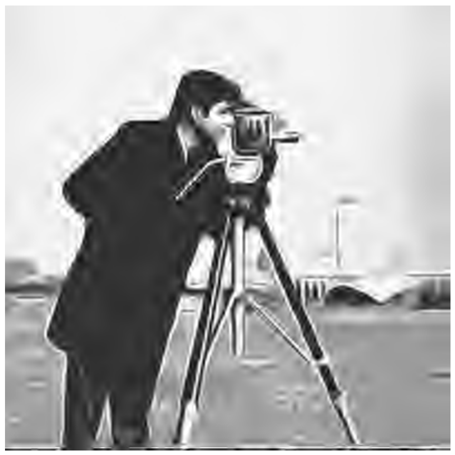
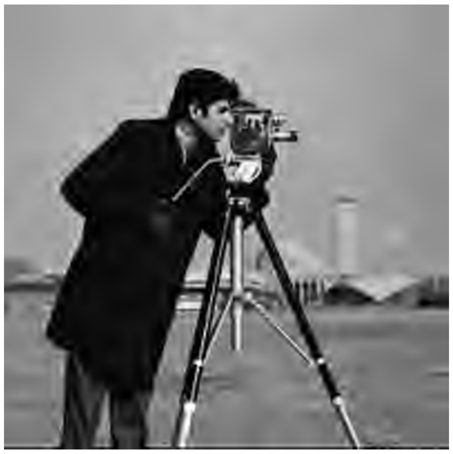


Image Space Stylization Result


Compressed Stylization

2.12.3 Input Photograph

Input Photograph


Compressed Input Photograph

3 Vector Format Details

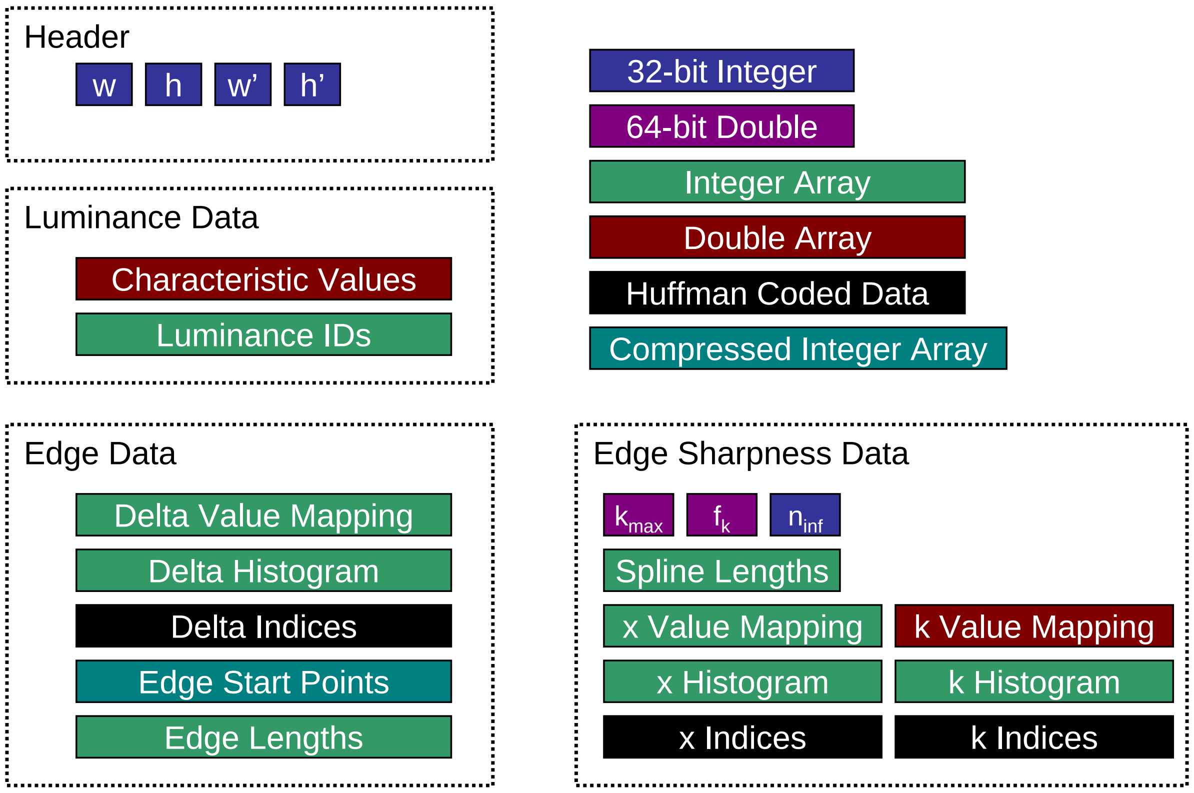


Figure 1: Encoding Format. The header includes the resolution of the source image, as well as the resolution of the label image used when calculating connected components. The edge sharpness data includes the sampling frequency of the position data, f_k, along with the maximum sharpness bound k_{max}. The integer n_{inf} records the number of edge lines for which the sharpness is uniformly infinite, n_{inf}. The histogram calculated for edge delta values is slightly compressed; in that the edge delta histogram is forced to be symmetric about the origin. For example, if there are three delta values equal to -4 and one value equal to 4, the recorded histogram will consider the frequency of both 4 and -4 to be two.

The details of the encoding format are summarized in Figure 1. The format is most simply described in terms of the following primitives:

Integer Arrays A list of unsigned integer values, $L = (i_0, ..., i_n)$. Integer arrays are encoded by first storing the number of elements, followed by the number of bits b needed to encode any element of array, calculated as $b = \lceil \log_2(\max(L)) \rceil$. Both n and b are stored as 32-bit integers. Finally, the elements $(i_0, ..., i_n)$ themselves are stored using b bits each.

Compressed Integer Array A slightly more memory efficient variant on an integer array. For a list of unsigned integer values, $(i_0, ..., i_n)$, the number of elements n is recorded, followed by the the maximum integer $m = \max(L)$. Then a huffman encoding for the integers between 0 and m is generated by assuming that all integers in that range are equally probable. If m is a power of two, a compressed integer array encoding will be nearly identical to the uncompressed encoding. In other cases, it may be slightly more memory efficient.

Double Arrays A list of 64 bit floating point values, $(d_0, ..., d_n)$. Double arrays are encoded by first storing the number of elements, followed by the 64-bit representation of each element of the array.

Huffman Coded Data Store a list of unsigned integer values $L = (i_0, ..., i_n)$, using a matching list of probabilities $p_0, ..., p_m$. The probability data is generated by calculating the frequency of each integer value, and then setting p_i equal to the number of times i occurs in L. The probabilities are used to generate Huffman coding strings for each element of L, which are then used to store each element of L. In order to store data in this format, the probability data (i.e., the data histogram) must be stored in a separate integer array.

Typically, the Huffman encoded integer list will contain indices into a set of either floating point or integer values. Thus a list of source double values $(d_0, ..., d_n)$ might be coded by first finding the set of all unique d_j values. Represented as a double array, that set defines a mapping between the double values d and integers i. A histogram for the implied integer list $(i_0, ..., i_n)$ would then be calculated. As written to disk, the source double array $(d_0, ..., d_n)$ would represented as the combination of its mapping data, histogram data, and the Huffman coded indices $(i_0, ..., i_n)$.

The edge and sharpness data are both stored using flattened lists. Position data for all edges is concatenated into a single list, while the number of points in each edge is stored as a separate integer array.

It is often the case that all points on an edge line will be "infinitely sharp". No sharpness data is stored for such edge lines. To facilitate this, edges are arranged into a list having the property that the first n_{inf} edges fall into the "infinitely sharp" case. Thus, the number of sharpness splines stored is equal to the total number of edge lines, minus n_{inf}.

References

[1] Jian Sun, Lin Liang, Fang Wen, and Heung-Yeung Shum. Image vectorization using optimized gradient meshes. *ACM Trans. Graph.*, 26(3):11, 2007.

[2] Tian Xia, Binbin Liao, and Yizhou Yu. Patch-based image vectorization with automatic curvilinear feature alignment. *ACM Trans. Graph.*, 28(5):1–10, 2009.