DIP Homework 1

Qiuyi Zhang 12330402
joyeec9h3@gmail.com

October 19, 2014

Contents

1 Exercises

1.1 Storage

1. How many bit planes are there for this image?

   Answer: There are $\log_2 256 = 8$ bit planes.

2. Which plane is the most visually significant one?

   Answer: The one that contains the set of the most significant bit – the 8th plane.

3. How many bytes are required for storing this image? (Don't consider image headers and compression.)

   Answer:

$$\begin{aligned}\text{Bytes needed} &= 2048 \times 2048 \text{ bit per plane} \times 8 \text{ planes} \\ &= 2^{22} \times 8 \text{ bits} \\ &= 2^{22} \text{ bytes}\end{aligned}$$

1.2 Adjacency

1. There is no 4-path between p and q, since $N_4(q) = \emptyset$.

2. There is one shortest path between p and q with length of 4.

3. There is one shortest m-path between p and q with length of 5.

$$\begin{tikzcd}
3 & 4 & 1 & 2 & 0 \\
0 & 1 & 0 & 4 & 2(q) \\
2 & 2 \arrow[r] & 3 \arrow[r] & 1 \arrow[ur] & 4 \\
2(p) \arrow[ur] & 0 & 4 & 2 & 1 \\
1 & 2 & 0 & 3 & 4
\end{tikzcd}$$

(a) Shortest 8-path

$$\begin{tikzcd}
3 & 4 & 1 & 2 & 0 \\
0 & 1 & 0 & 4 & 2(q) \\
2 \arrow[r] & 2 \arrow[r] & 3 \arrow[r] & 1 \arrow[ur] & 4 \\
2(p) \arrow[u] & 0 & 4 & 2 & 1 \\
1 & 2 & 0 & 3 & 4
\end{tikzcd}$$

(b) Shortest m-path

Figure 1: Shortest paths

1.3 Logical Operations

1. $A \cap B \cap C$
2. $(A \cap B) \cup (B \cap C) \cup (A \cap C)$
3. $(\overline{A} \cap B \cap \overline{C}) \cup (A \cap \overline{B} \cap C)$

2 Programming Tasks

2.1 Scaling

2.1.1 Discussion

There are 5 common ways to scale a gray image [1]:

1. Sampling
2. Area mapping (or area averaging) and lowpass filtering
3. Mip-mapping
4. Min-max
5. Interpolation

For this project, I used interpolation since it is available in many third-party libraries and can produce a good result.

There are 3 common interpolation methods [3]:

1. Nearest-neighbor
2. Bilinear
3. Bicubic

I chose bicubic interpolation for this project because it is effcient enough, produces fewer interpolation artifacts, and is applicable for both down-scaling and up-scaling. The result are listed in section 2.1.2, and the algorithm is described in section 2.1.2. We can see that the false contouring starts to appear in the 32-level image.

Note

Although I have not implement it, there is a very impressive image scaling algorithm, called *seam carving* [2], which can scale an image to a new aspect ratio very natually.

2.1.2 Results

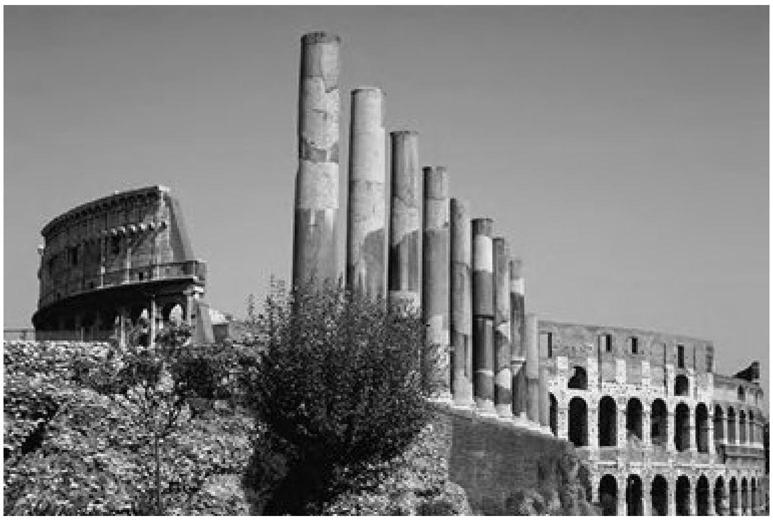

Figure 2: The original image

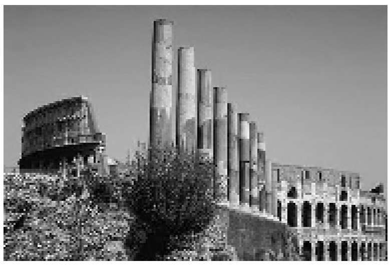

Figure 3: Down-scale to 192×128

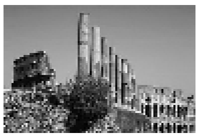

Figure 4: Down-scale to 96×64

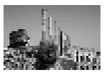

Figure 5: Down-scale to 48×32

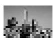

Figure 6: Down-scale to 24×16

Figure 7: Down-scale to 12×8

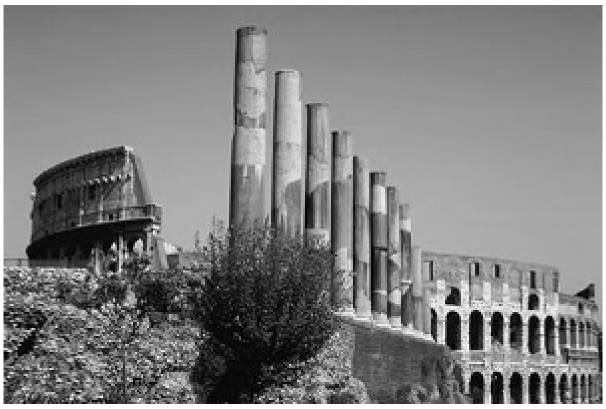
Figure 8: Down-scale to 300 × 200

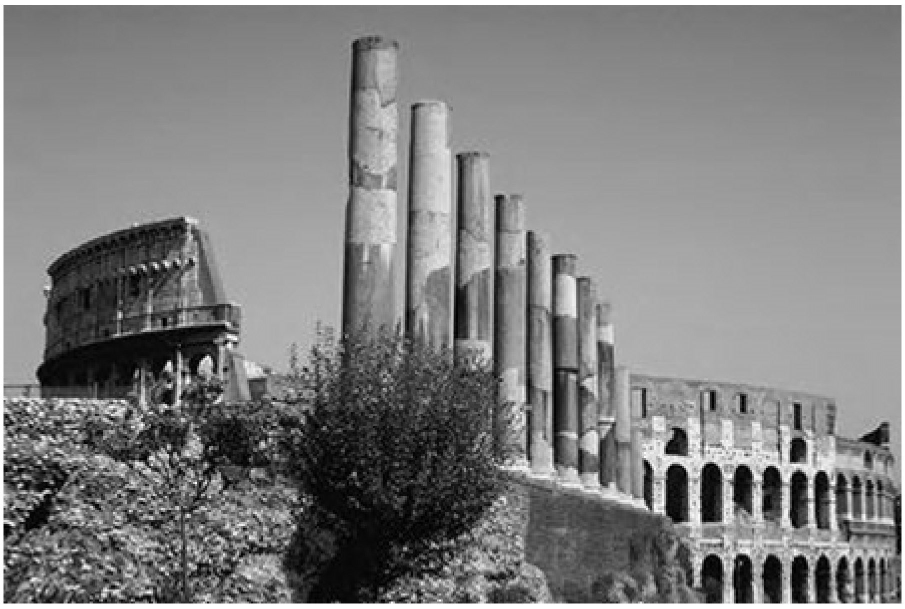
Figure 9: Up-scale to 450 × 300

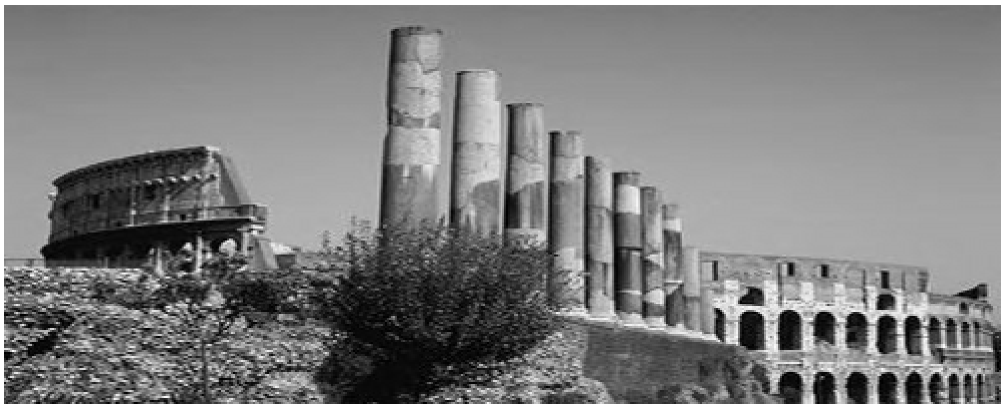
Figure 10: Scale to 500 × 200

2.1.3 Algorithm

Algorithm 1 Scaling gray image

```
function SCALE($input\_img$, $size$)
    Create an empty image $output\_img$ with $size$
    if $input\_img.size == size$ then
        Copy $input\_img$ to $output\_img$
        return $output\_img$
    end if
    INTERPOLATE ← GETINTERPOLATION($input\_img$)
    for each pixel $(x, y)$ in $output\_img$ do
        $(relx, rely)$ ← GETRELATIVEPOSITION($x$, $y$, $size$, $input\_img.size$)
        $new\_value$ ← INTERPOLATE($relx$, $rely$)
        Put $new\_value$ back into $(x, y)$ in $output\_img$
    end for
    return $output\_img$
end function

function GETINTERPOLATION($input\_img$)
    $x \leftarrow [0, 1, ..., input\_img.width - 1]$
    $y \leftarrow [0, 1, ..., input\_img.height - 1]$
    for each $j$ in $y$ do
        for each $i$ in $x$ do
            $row$ ← an empty list
            Append $input\_img.getpixel(i, j)$ to $row$
        end for
        Append $row$ to $z$
    end for
    return GETBICUBICINTERPOLATION($x$, $y$, $z$)
end function

function GETRALATIVEPOSITION($x$, $y$, $new\_size$, $old\_size$)
    $new\_width, new\_height = new\_size$
    $old\_width, old\_height = old\_size$
    $new\_x \leftarrow x \times \frac{old\_width}{new\_width}$
    $new\_y \leftarrow y \times \frac{old\_height}{new\_height}$
    return $new\_x, new\_y$
end function
```

2.2 Quantization

2.2.1 Discussion

Although color quantization for a color image is complex, it is relatively simple to implement quantization for a gray image. For this project, I first compute the new palette for the given gray level, then replace each pixel with its nearest neighbor in this palette. This algorithm is somewhat similar to the median-cut algorithm applied in one dimension.

The results are listed in section 2.2.2, and the algorithm is described in section 2.2.3.

2.2.2 Results

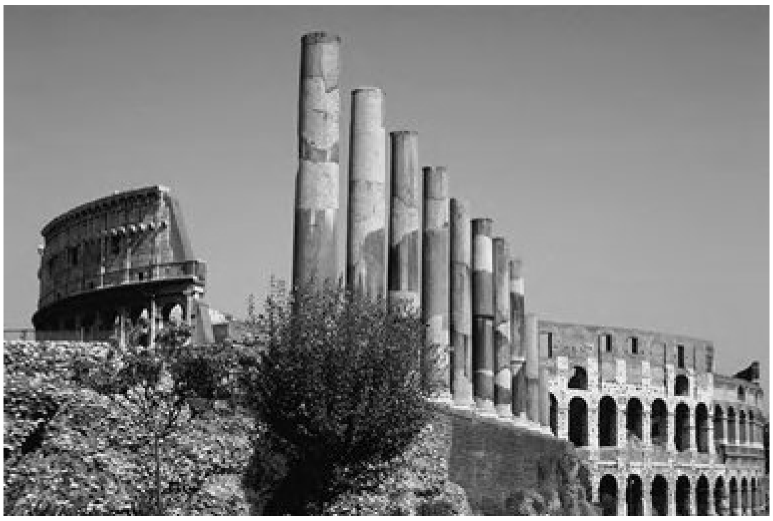

Figure 11: The original image

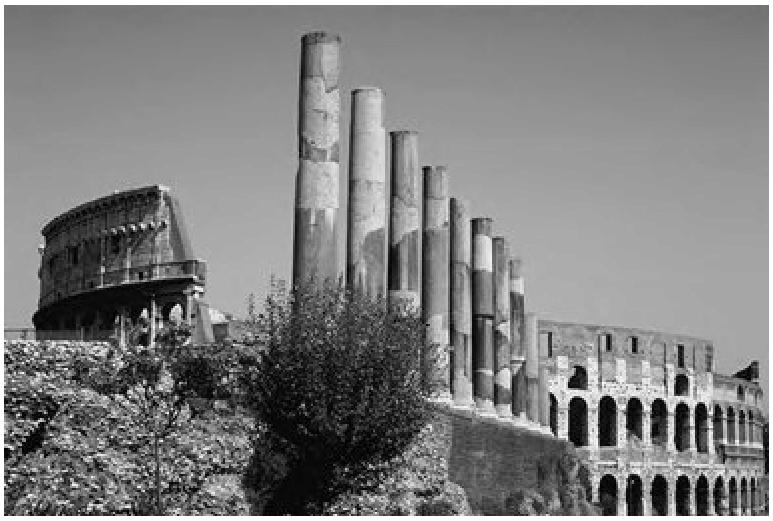

Figure 12: 128 gray levels

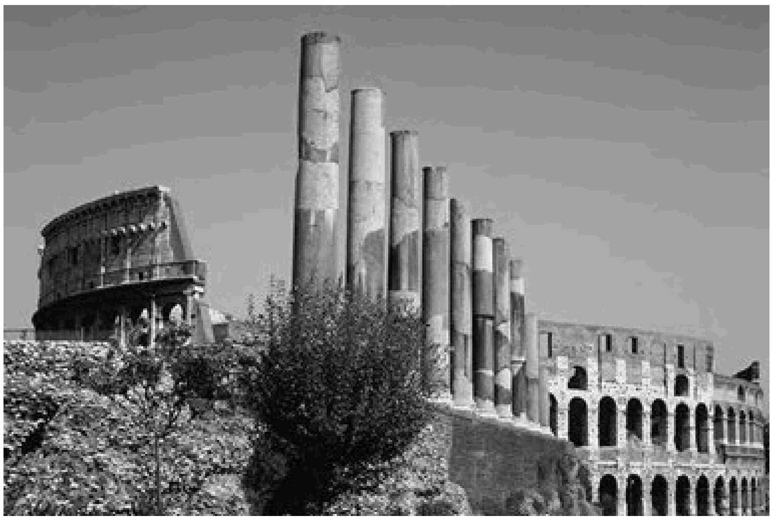
Figure 13: 32 gray levels

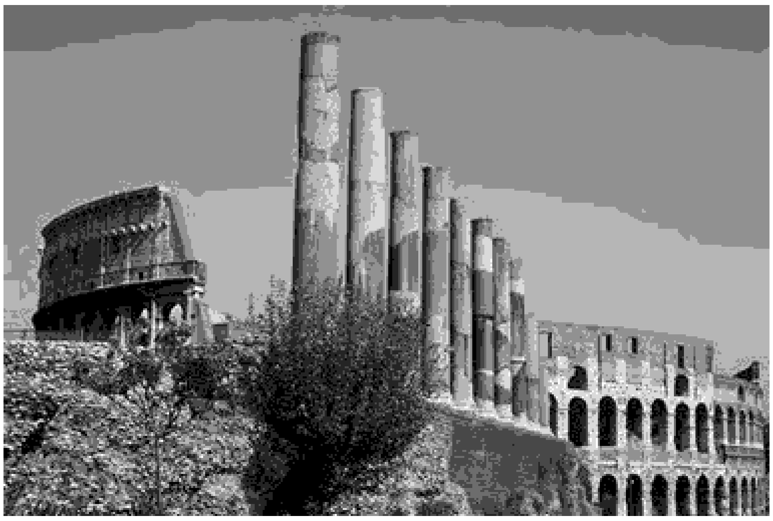
Figure 14: 8 gray levels

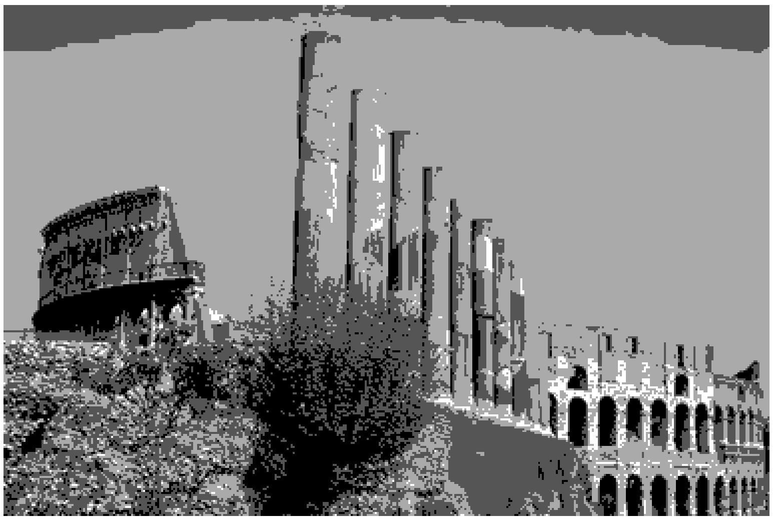
Figure 15: 4 gray levels

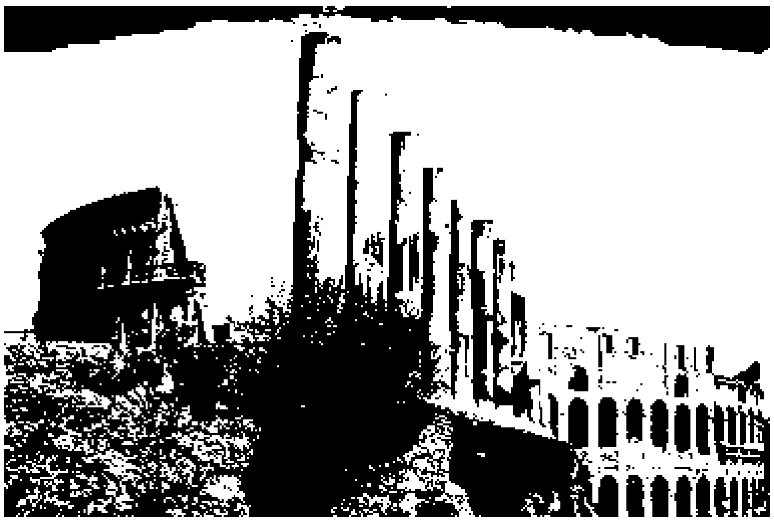

Figure 16: 2 gray levels

2.2.3 Algorithm

Algorithm 2 Quantize gray image

```
1: function QUANTIZE($input\_img, level$)
2:     Create an empty image $output\_img$ with $input\_img.size$
3:     if $input\_img.level == level$ then
4:         Copy $input\_img$ to $output\_img$
5:         return $output\_img$
6:     end if
7:     $new\_pallete \leftarrow [\lfloor 255 \times \frac{i}{level-1} \rfloor$ for $i$ in $[0, 1, ..., level]]$
8:     for each pixel $(x, y)$ in $input_img$ do
9:         $new\_color \leftarrow$ the nearest neighbor of the pixel color in $new\_pallete$
10:        Put $new\_color$ into $(x, y)$ in the $output\_img$
11:    end for
12:    return $output\_img$
13: end function
```

References

[1] General scaling - leptonica. `http://www.leptonica.com/scaling.html`. Accessed: 2014-10-19.

[2] Avidan, S., and Shamir, A. Seam carving for content-aware image resizing. In *ACM SIGGRAPH 2007 Papers* (New York, NY, USA, 2007), SIGGRAPH '07, ACM.

[3] Gonzalez, R., and Woods, R. *Digital Image Processing*. Pearson/Prentice Hall, 2008.