Pointillism via Linear Programming

Robert Bosch* and Adrianne Herman
Dept. of Mathematics, Oberlin College, Oberlin, Ohio, 44074
(bobb@cs.oberlin.edu, aherman@cs.oberlin.edu)

May 24, 2004

Abstract

Pointillism is a painting technique in which the painter places dots of paint on the canvas in such a way that they blend together into desired forms when viewed from a distance. In this brief note, we describe how to use linear programming to construct a pointillist portrait.

1 Introduction

Pointillism is a painting technique in which the painter places dots of paint on the canvas in such a way that they blend together into desired forms when viewed from a distance. Pointillism was invented by the French Neo-Impressionist Georges Seurat (1859-1891).

In this brief note, we describe how to use linear programming to construct a pointillist portrait. In the next section, we discuss how we prepare and process the photograph of the subject of our portrait. In the following section, we present the linear program we solve to construct our portrait. In the final section, we display some of our artwork. For other examples of "Opt Art" (art produced using mathematical optimization techniques), see [1], [2], and [3].

2 The Photograph

We begin the construction of our portrait by converting a $r_{\text{photo}} \times c_{\text{photo}}$ digital photograph of the subject of our portrait into PGM (portable graymap) format. By doing this, we are converting the photograph into black-and-white, treating each of its $r_{\text{photo}}c_{\text{photo}}$ pixels as an integer between 0 and 255. These integers are grayscale values. A grayscale value of 0 tells us that the pixel is completely black, while a grayscale value of 255 tells us that the pixel is completely white. A grayscale value strictly between 0 and 255 indicates that the pixel is some shade of gray other than completely black or completely white. The larger

*corresponding author

the grayscale value, the lighter the shade of gray. Note that at this point, our photograph can be thought of as a list of $r_{\text{photo}} c_{\text{photo}}$ integers, each one between 0 and 255.

Next, we compress our photograph. (We reduce both the *number* of integers in our list and the *largest* integer in our list.) First, we find positive integers r_{portrait} and c_{portrait} and a small positive integer k such that $r_{\text{photo}} = k r_{\text{portrait}}$ and $c_{\text{photo}} = k c_{\text{portrait}}$. (If necessary, we crop the photograph.) Next, we divide the canvas into r_{portrait} rows and c_{portrait} columns of unit squares, and we partition the pixels of our photograph into r_{portrait} rows and c_{portrait} columns of $k \times k$ squares of pixels. Then, for each row $1 \leq i \leq r_{\text{portrait}}$ and column $1 \leq j \leq c_{\text{portrait}}$ of our photograph, we compute the mean grayscale value $\mu_{i,j}$ of the pixels in square (i, j) and set

$$g_{i,j} = \gamma - \lfloor \gamma \mu_{i,j} / 256 \rfloor,$$

where γ is a positive integer. By doing this, we are defining $g_{i,j}$ to be the average darkness of square (i, j) of our photograph on a 0 (completely white) to γ (completely black) gray scale. Note that at this point, our photograph can be thought of as a list of $r_{\text{portrait}} c_{\text{portrait}}$ integers, each one between 0 and γ.

Figure 1 illustrates this "photo processing." Figure 1(a) is a portion of a photograph of DaVinci's Mona Lisa (her right eye). Figure 1(b) is a table of the resulting $g_{i,j}$'s. Here $r_{\text{photo}} = 40$, $c_{\text{photo}} = 32$, $r_{\text{portrait}} = 20$, $c_{\text{portrait}} = 16$, $k = 2$, and $\gamma = 9$.

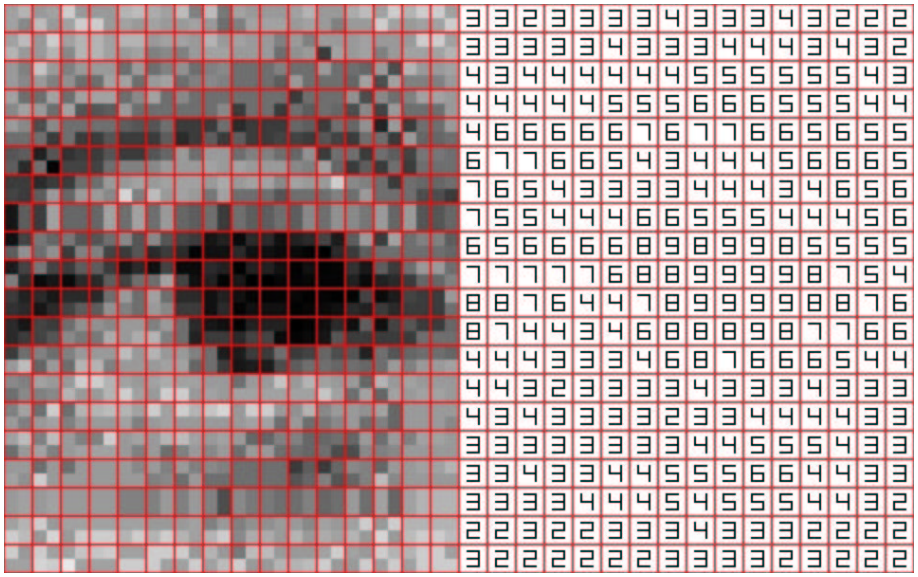

Figure 1: (a) A portion of DaVinci's Mona Lisa, (b) the resulting $g_{i,j}$'s

3 The Linear Program

The pointillist painter paints a portrait by placing dots of paint on the canvas, placing the dots wherever he or she chooses, making each dot as dark as he or she pleases.

We construct a pointillist portrait by placing disks of tinted glass on the canvas. We assume that the canvas is completely white. We do not allow ourselves to place disks wherever we wish; instead, we force ourselves to place one disk at the center of each square of the canvas. All of our disks have radius ρ, and all are the same thickness. All of our disks have constant opacity, but we allow the opacity to vary from disk to disk. To do this, we let $x_{i,j}$ equal the opacity of the disk centered at square (i, j). The opacity of a disk is the fraction of light the disk absorbs. So if $x_{i,j} = 0$, the disk centered at (i, j) is completely transparent. If $x_{i,j} = 1$, the disk is completely opaque.

Once we have assigned values to the $x_{i,j}$'s, it is easy to compute the darkness of any point on the canvas. We define the darkness of a point to be the combined opacity of the disks that cover it. Suppose, for example, that a beam of light is aimed at the center of square $(2, 4)$ in figure 2. This point is covered by two nontransparent disks, each having opacity $1/3$. Two thirds of the light passes through the first disk. And two thirds of the light that passes through the first disk passes through the second disk. So four ninths of the light passes through both disks, which means that five ninths of the light is absorbed. Consequently, the darkness of the point is $5/9$. The general rule is that the darkness equals one minus the product of the complements of the opacities of the disks that cover the point (e.g. $\frac{5}{9} = 1 - (1 - \frac{1}{3})(1 - \frac{1}{3})$).

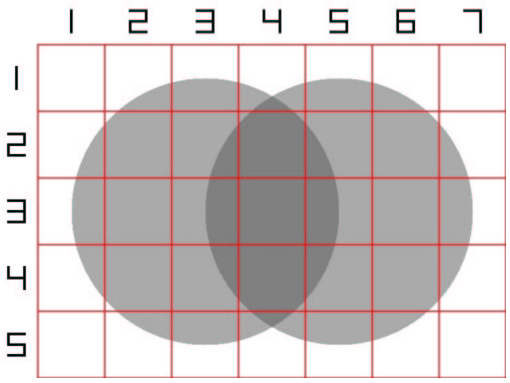

Figure 2: $\rho = 2$, $x_{3,3} = x_{3,5} = \frac{1}{3}$, $x_{i,j} = 0$ for all other (i, j).

Now that we know how to compute the darkness of any point on the canvas, we can talk about how to compute the darkness $d_{i,j}$ of any square (i, j) of the canvas. One possibility is to construct a darkness function that takes as input the coordinates of a point and returns as output the darkness of that point. If

we had such a function, we could integrate it over all points in square (i, j) to get the darkness of square (i, j). Unfortunately, if we do this, $d_{i,j}$ will end up being a very complicated nonlinear function of the $x_{i,j}$'s. Another possibility is to define

$$C_{i,j} = \{(i', j') : 1 \leq i' \leq r, 1 \leq j' \leq c, (i' - i)^2 + (j' - j)^2 \leq \rho^2\},$$

the set of all squares (i', j') within ρ of square (i, j). Note that we are measuring from center-of-square to center-of-square. Then we could set

$$d_{i,j} \;=\; 1 - \prod_{(i',j') \in C_{i,j}} (1 - x_{i',j'}). \tag{1}$$

But here, too, $d_{i,j}$ is a nonlinear function of the $x_{i,j}$'s. A third approach—the one we adopted—is to use a linear approximation of equation (1):

$$d_{i,j} \;= \sum_{(i',j') \in C_{i,j}} x_{i',j'}.$$

Now we are finally ready to present the linear program we solve to construct our portrait. Clearly our goal, in plain English, is to place our tinted disks in such a way that the canvas ends up resembling our photograph as much as possible. In mathematical terms, our goal is to assign values to the $x_{i,j}$'s that make each $\gamma d_{i,j}$ term (the darkness of square (i, j) of the canvas on a 0-to-γ scale) as close as possible to the corresponding $g_{i,j}$ (the darkness of square (i, j) of the photograph). One way to do this is to solve the following nonlinear programming problem:

$$\begin{array}{lll}
\text{(NLP)} \quad \text{minimize} & \displaystyle\sum_{i,j} |\gamma d_{i,j} - g_{i,j}| & \\
\text{subject to} & d_{i,j} \;= \displaystyle\sum_{(i',j') \in C_{i,j}} x_{i',j'} & \forall i, j, \\
& 0 \leq d_{i,j} \leq 1 & \forall i, j, \\
& 0 \leq x_{i,j} \leq 1 & \forall i, j.
\end{array}$$

The problem (NLP) can be "linearized" using a standard trick described in most textbooks on linear programming (see page 222 of [4], for example). For each absolute value term, $|\gamma d_{i,j} - g_{i,j}|$, we introduce a variable $z_{i,j}$ and two constraints, $z_{i,j} \geq \gamma d_{i,j} - g_{i,j}$ and $z_{i,j} \geq -\gamma d_{i,j} + g_{i,j}$. We end up with the following linear program:

$$\begin{array}{lll}
\text{(LP)} \quad \text{minimize} & \displaystyle\sum_{i,j} z_{i,j} & \\
\text{subject to} & z_{i,j} \geq \gamma d_{i,j} - g_{i,j} & \forall i, j, \\
& z_{i,j} \geq -\gamma d_{i,j} + g_{i,j} & \forall i, j, \\
& d_{i,j} \;= \displaystyle\sum_{(i',j') \in C_{i,j}} x_{i',j'} & \forall i, j,
\end{array}$$

$$0 \leq d_{i,j} \leq 1 \qquad \forall i, j,$$
$$0 \leq x_{i,j} \leq 1 \qquad \forall i, j.$$

Note that at optimality, $z_{i,j}$ will equal $|\gamma d_{i,j} - g_{i,j}|$. Also note that linear program (LP) has $3r_{\text{portrait}}c_{\text{portrait}}$ variables and $3r_{\text{portrait}}c_{\text{portrait}}$ constraints.

4 Results

Figure 3 displays two examples of our pointillist artwork. We constructed them by solving the linear program (LP), running version 6.6 of CPLEX on an 800 Mz Pentium III PC. Figure 3(a) is based on a photograph of a portion of DaVinci's Mona Lisa, and Figure 3(b) is based on a photograph of a portion of Vermeer's Girl with a Pearl Earring. For each piece, we used $r = 90$, $c = 70$, $k = 2$, $\rho = 2$, and $\gamma = 9$. For figure 3(a), the solution of (LP) required 12158 seconds. For figure 3(b), 15586 seconds were needed. For figure 3(a), the optimal objective value was approximately 905.81; for figure 3(b), it was 782.93. Consequently, for figure 3(a), the "average error" (i.e. the average of the $|\gamma d_{i,j} - g_{i,j}|$ terms) was 0.14378. For figure 3(b), the average error was 0.12428.

Figure 3: (a) Mona Lisa, (b) Girl with a Pearl Earring

References

[1] Bosch, R. 2002. Domino Artwork. http://www.dominoartwork.com.

[2] Bosch, R. 2003. Dominizing Venus. *OR/MS Today.* April 20-21.

[3] Bosch, R., A. Herman. 2004. Continuous line drawings via the traveling salesman problem. *Oper. Res. Lett.* **32** 302-303.

[4] Chvátal, V. 1983. *Linear Programming.* WH Freeman, New York.