ALGORITHM LIST

Version 0.1
Computing
COMP120

Dr Michael Scott

Contents

1 This algorithm swaps two colour channels within an image

Algorithm 1 Swap Channel

Require: an image, *image* with channels, c in RGB format

```
procedure SWAP(image)
    for all c in image do
        t ← c_1
        c_1 ← c_0
        c_0 ← t
    end for
end procedure
```

2 This algorithm converts an image into a greyscale version

Algorithm 2 Greyscale

Require:

an image, *image*, with N channels, c

$0 \leq c_{0..N} \leq 255$

procedure GREYSCALE(image)
  for $x = 0$, width, $y = 0$,height **do**
    $c \leftarrow$ pixel(x, y)
    $c_{0..N} \leftarrow \frac{\sum_{i=0}^{N} c_i}{N}$
    pixel(x, y) $\leftarrow$ c
  end for
end procedure

3 This algorithm converts an image into a negative

Algorithm 3 Negative

Require:
an image, *image* with channels, c in RGB format
Ensure: The result, r, should be within the range
$0 \leq r \leq 255$

procedure NEGATIVE(image)
  for $x = 0$, width, $y = 0$,height **do**
    $c \leftarrow$ pixel(x, y)
    $c_{0..N} \leftarrow 255 - c_{0..N}$
    pixel(x, y) $\leftarrow c$
  end for
end procedure

4 This algorithm copies the top of an image to the bottom

Algorithm 4 Top-Copy

```
ht ← h/2
for y = 0, y < ht, x = 0, x < w do
    i ← ht + y
    pixel(x, i) ← pixel(x, y)
end for
```

5 This algorithm calculates the distance between two colours

Algorithm 5 Distance between colours

Require:

Two colours defined as a tuple of integers in 8-bit RGB format such that:
$0 \leq r_{0..1} \leq 255$
$0 \leq g_{0..1} \leq 255$
$0 \leq b_{0..1} \leq 255$

Ensure:

The distance between the two colours:
d

```
d ← √((r₁ − r₀)² + (g₁ − g₀)² + (b₁ − b₀)²)
return d
```

6 This algorithm checks if an existing pixel is close to another in colour

Algorithm 6 Colour Tolerance

Require:

a threshold value, $0 \leq t \leq 255$
a colour in RGB format, $0 \leq c_{0..2} \leq 255$
a pixel in RGB format, $0 \leq p_{0..2} \leq 255$

function TOLERANCE(color c, pixel p, threshold t)
  $d \leftarrow \sum_{i=0}^{2}(p_i - c_i)^2$
  if $r < t$ **then**
    return true
  else
    return false
  end if
end function

7 This algorithm reduces the colours within an image. Multiple conditions might be required.

Algorithm 7 Posterization

Require:
a channel value, $0 \leq c_{0..2} \leq 255$
a replacement value, $0 \leq r \leq 255$
a minimum threshold, $0 \leq t_{min} \leq 255$
a maximum threshold, $0 \leq t_{max} \leq 255$

procedure POSTERIZATION(t, c, r)
    for $x = 0$, width, $y = 0$,height **do**
        $c \leftarrow$ pixel(x, y)
        if $t_{min} \leq c_0 \leq t_{max}$ **then**
            $c_0 \leftarrow r$
            pixel(x, y) $\leftarrow c$
        end if
    end for
end procedure

8 This algorithm calculates the luminance of a pixel.

Algorithm 8 Luminance

Require:

A colour defined as a tuple of integers in 8-bit RGB format such that:
$0 \leq c_{0..2} \leq 255$

Ensure:

The luminance:
L

$L \leftarrow \frac{\sum_{i=0}^{2} c_i}{3}$

9 This algorithm mirrors an image around it's middle.

Algorithm 9 Mirroring

Require:
the height of the image, $0 \leq h$
the width of the image, $0 \leq w$

$ht \leftarrow \frac{h}{2}$
for $y = 0, y < ht, x = 0, x < w$ **do**
    $i \leftarrow h - y - 1$
    pixel(x, i) $\leftarrow$ pixel(x, y)
end for

10 This algorithm makes edges appear white.

Algorithm 10 Edge Detection

Require:
the height of the image, $0 \leq h$
the width of the image, $0 \leq w$
the source image, image

procedure EDGEDETECT(image)
    for all y in h **do**
        for all x in w **do**
            $p_h \leftarrow$ pixel(x, y)
            $t_h \leftarrow \sum_{i=0}^{3} p_{hi}$
            $p_r \leftarrow$ pixel(x+1, y)
            $t_r \leftarrow \sum_{i=0}^{3} d_{ri}$
            $p_d \leftarrow$ pixel(x, y+1)
            $t_d \leftarrow \sum_{i=0}^{3} d_{di}$

            if abs(t_h - t_d) > 20 and abs(t_h - t_r) > 20 **then**
                setPixel(x, y, 255, 255, 255)
            else
                setPixel(x, y, 0, 0, 0)
            end if
        end for
    end for
end procedure

11 This algorithm replaces one background with another.

Algorithm 11 Background Subtraction

```
Require:
    the height of the source image, 0 ≤ h
    the width of the source image, 0 ≤ w
    the source image, image
    the original background image, background
    the new background image, newBackground
procedure BACKGROUND(image, background, newBackground)
    for all y in h do
        for all x in w do
            p ← pixel(image, x, y)
            p_b ← pixel(background, x, y)
            if distance(p, p_b) < t then
                pixel(image, x, y) ← pixel(newBackground, x, y)
            end if
        end for
    end for
end procedure
```

12 This algorithm places one image inside another.

Algorithm 12 Collage

Require:
the source image, image
the height of the source image, $0 \le s_h$
the width of the source image, $0 \le s_w$
the destination image, canvas
the height of the canvas image, $0 \le c_h$
the width of the canvas image, $0 \le c_w$
the target location, $0 \le t_x < s_w$, $0 \le t_y < s_h$

procedure COLLAGE(source, canvas, t_x, t_y)
    for y=0, h; x=0, w **do**
        if $x \ge t_x$ and y < s_w+t_x **then**
            if ($y \ge t_y$ and y < s_h+t_y **then**
                Pixel(canvas, x, y) $\leftarrow$ Pixel(source, x - t_x, y - t_y)
            end if
        end if
    end for
end procedure