Photo histogram in CoffeeKup — *Smooth CoffeeScript*

This literate program is *interactive* in its HTML form. Edit a CoffeeScript segment to try it.

Photo histogram

It is easy to prototype an idea with CoffeeScript and CoffeeKup. This program shows separation of color channels as a starting point for some photo analysis.

HTML Rendering

In CoffeeKup you write a CoffeeScript function that is `render`ed to HTML. That function can contain the page elements, styling and CoffeeScript functions that are used on the web page. It can also refer to external files so you can modularize and separate the look and feel from the content. To load a script include a tag such as: `script src: 'underscore.js'`.

This `histogram` program can run embedded in a web page, where the program can be changed interactively or it can run standalone. In the browser scripts are loaded into the window environment, the standalone environment instead uses `global` and `require`.

An existential test on `exports` can be used to determine which environment the program is running in — another way is to look at whether the `window` environment is available.

```
kup = if exports? then require 'coffeekup' else window.CoffeeKup

webpage = kup.render ->
```

User Interface

Since the program will use a `canvas` and some Unicode characters, its `doctype` and `charset` are HTML5 and UTF–8. The CSS styling for the application is — so far — very simple so it is included in-line. That is convenient when experimenting, if you are reading this on the web page then you can change the styling and immediately see the effect on the output.

```
  doctype 5
  html ->
    head ->
      meta charset: 'utf-8'
      title 'Histogram'
      style 'body        {color: #FFFFFF;  background-color: #404040}
             #background {position: absolute; top:  40px; left: 20px}
             #image      {position: absolute; top:   0px; left: 80px}
             #picture    {position: absolute; top:   0px; left: 80px}
             #histogram  {position: absolute; top: 140px; left:  0px}'
```

The `img` tag is used to load a photo normally from a file, however in the interactive environment it is instead loaded from a global predefined variable[1]. The application does not need to display the original picture, instead it shows a `canvas` with the processed `picture`. That `canvas` is therefore set to the same size and placed via absolute positioning right on top of it.

```
    body ->
      div id: 'background', ->
        img    id: 'image', width: 90, height: 90, \
          src: window?.ostrich ? '../img/ostrich.jpg'
        canvas id: 'picture', width: 90, height: 90
        canvas id: 'histogram', width: 256, height: 100, \
          onClick: 'onChange()'
```

[1] This is a side effect of how the interactive environment is constructed . It uses an `iframe` with a URI encoded `data` attribute to display the embedded HTML output. The image is encoded in base64 and predefined to simplify the process. It is not something you would need to do when using CoffeeKup outside of this environment.

Functionality

The application responds to `click` events on the histogram by displaying either all color channels combined or one of the red, green, blue, or the transparency / alpha channel. First in the web application's CoffeeScript section is a definition of constants and a variable `view` that holds an index of what is currently displayed.

```
    coffeescript ->
      hues =
        red:       'rgba(255, 128, 128, 0.5)'
        green:     'rgba(128, 255, 128, 0.5)'
        blue:      'rgba(128, 128, 255, 0.5)'
        alpha:     'rgba(128, 128, 128, 0.5)'
      legend    =  ['', 'R', 'G', 'B',  '']
      textColor = '#F7C762'
      textFont  = '12pt Times'
      textPos   = x:230, y:-80
      view      = 0
```

The data in a photo is stored according to the definition given in HTML Canvas 2D Context by W3C:

> The CanvasPixelArray object provides ordered, indexed access to the color components of each pixel of the image data. The data must be represented in left-to-right order, row by row top to bottom, starting with the top left, with each pixel's red, green, blue, and alpha components being given in that order for each pixel. Each component of each device pixel represented in this array must be in the range 0..255, representing the 8 bit value for that component. The components must be assigned consecutive indices starting with 0 for the top left pixel's red component.

This standardized image format makes it easy to count the values in each of the channels with the `for ... in` statement's index feature and the modulus `%` operator.

```
      analyze = (data) ->
        bins = red: [], green: [], blue: [], alpha: []
        for name, bin of bins
          bin[i] = 0 for i in [0..255]
        for val, i in data
          switch i % 4
            when 0 then bins.red[val]++
            when 1 then bins.green[val]++
            when 2 then bins.blue[val]++
            when 3 then bins.alpha[val]++
        bins
```

The width of the `histogram` canvas was chosen to be the same as the number of values in each of the color channel `bins`: 256. That simplifies the drawing of the plots, they only have to be scaled to match the height of the canvas. Since the y-coordinate on a canvas defaults to start at the top, the `scale` function is used to turn the coordinate system upside down — but only after the text `legend` has been printed in its predetermined location.

```
      drawGraphs = (ctx, graphs) ->
        drawPlot = (ctx, plot, color) ->
          ctx.fillStyle = color
          ctx.beginPath()
          ctx.moveTo 0, 0
          for y, x in plot
            ctx.lineTo x, y
          ctx.lineTo plot.length, 0
          ctx.closePath()
          ctx.fill()

        ctx.translate 0, ctx.canvas.height
        ctx.fillStyle = textColor
```

```
    ctx.font = textFont
    ctx.fillText  legend[view], textPos.x, textPos.y
    ctx.scale 1, -1                        # flip y-axis
    drawPlot ctx, graphs.red,   hues.red   if view in [0, 1]
    drawPlot ctx, graphs.green, hues.green if view in [0, 2]
    drawPlot ctx, graphs.blue,  hues.blue  if view in [0, 3]
    drawPlot ctx, graphs.alpha, hues.alpha if view in [0, 4]
```

Events

Shared variables, `canvas` contexts, and the display are initialized when the application has loaded.

```
window.onload = ->
  $ = (element) -> document.getElementById element
  @image = $ 'image'
  @canvas = $ 'picture'
  @histogram = $ 'histogram'

  @context = canvas.getContext '2d'
  @plot = histogram.getContext '2d'
  unless @context? or @plot?
    alert 'No canvas in this browser.'
    return
  window.onChange()
```

A complete redraw is performed when a touch or click triggers an update. If it is an RGB channel that is being displayed then the other channels are set to zero so only the contribution from the current channel is shown in the photo. The `view` index is updated and clamped.

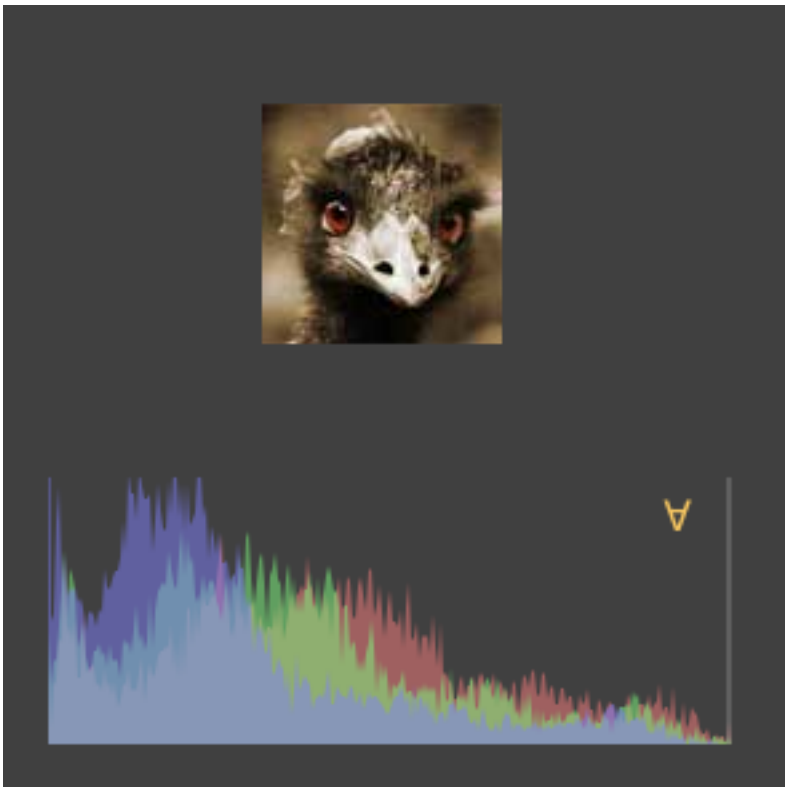
```
window.onChange = ->
  @histogram.width = @histogram.width     # reset
  @plot.clearRect 0, 0, @histogram.width, @histogram.height
  @context.drawImage @image, 0, 0

  picture = @context.getImageData 0, 0,
    @image.width, @image.height
  graphs = analyze picture.data
  drawGraphs @plot, graphs, view

  if 0 < view < 4
    picture = @context.getImageData 0, 0,
      @image.width, @image.height
    for i in [0...picture.data.length] by 1
      unless i%4 in [3, view-1] # unless alpha or current
        picture.data[i] = 0
    @context.putImageData picture, 0, 0

  if view++ is 4 then view = 0
  return
```

Wrap-up

This last bit do not relate to the web application but to the CoffeeKup `render`ing. The generated HTML is displayed either live in the interactive environment or as text — as nicely formatted as possible.

```
, format:on # Get formatted HTML
showDocument webpage
```

Output



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Histogram</title>
    <style>body        {color: #FFFFFF;  background-color: #404040}            #background {position: absolute; top:  40px; left: 20
  </head>
  <body>
    <div id="background">
      <img id="image" width="90" height="90" src="../img/ostrich.jpg" />
      <canvas id="picture" width="90" height="90"></canvas>
      <canvas id="histogram" width="256" height="100" onClick="onChange()"></canvas>
    </div>
  </body>
  <script>var __slice = Array.prototype.slice;var __hasProp = Object.prototype.hasOwnProperty;var __bind = function(fn, me){ return
        var analyze, drawGraphs, hues, legend, textColor, textFont, textPos, view;
        hues = {
          red: 'rgba(255, 128, 128, 0.5)',
          green: 'rgba(128, 255, 128, 0.5)',
          blue: 'rgba(128, 128, 255, 0.5)',
          alpha: 'rgba(128, 128, 128, 0.5)'
        };
        legend = [' ', 'R', 'G', 'B', ' '];
        textColor = '#F7C762';
        textFont = '12pt Times';
        textPos = {
          x: 230,
          y: -80
        };
        view = 0;
        analyze = function(data) {
          var bin, bins, i, name, val, _len;
          bins = {
            red: [],
            green: [],
            blue: [],
            alpha: []
```

```
    };
    for (name in bins) {
      bin = bins[name];
      for (i = 0; i <= 255; i++) {
        bin[i] = 0;
      }
    }
    for (i = 0, _len = data.length; i < _len; i++) {
      val = data[i];
      switch (i % 4) {
        case 0:
          bins.red[val]++;
          break;
        case 1:
          bins.green[val]++;
          break;
        case 2:
          bins.blue[val]++;
          break;
        case 3:
          bins.alpha[val]++;
      }
    }
    return bins;
  };
  drawGraphs = function(ctx, graphs) {
    var drawPlot;
    drawPlot = function(ctx, plot, color) {
      var x, y, _len;
      ctx.fillStyle = color;
      ctx.beginPath();
      ctx.moveTo(0, 0);
      for (x = 0, _len = plot.length; x < _len; x++) {
        y = plot[x];
        ctx.lineTo(x, y);
      }
      ctx.lineTo(plot.length, 0);
      ctx.closePath();
      return ctx.fill();
    };
    ctx.translate(0, ctx.canvas.height);
    ctx.fillStyle = textColor;
    ctx.font = textFont;
    ctx.fillText(legend[view], textPos.x, textPos.y);
    ctx.scale(1, -1);
    if (view === 0 || view === 1) drawPlot(ctx, graphs.red, hues.red);
    if (view === 0 || view === 2) drawPlot(ctx, graphs.green, hues.green);
    if (view === 0 || view === 3) drawPlot(ctx, graphs.blue, hues.blue);
    if (view === 0 || view === 4) {
      return drawPlot(ctx, graphs.alpha, hues.alpha);
    }
  };
  window.onload = function() {
    var $;
    $ = function(element) {
      return document.getElementById(element);
    };
    this.image = $('image');
    this.canvas = $('picture');
    this.histogram = $('histogram');
    this.context = canvas.getContext('2d');
    this.plot = histogram.getContext('2d');
    if (!((this.context != null) || (this.plot != null))) {
      alert('No canvas in this browser.');
      return;
    }
    return window.onChange();
  };
  return window.onChange = function() {
    var graphs, i, picture, _ref, _ref2;
    this.histogram.width = this.histogram.width;
    this.plot.clearRect(0, 0, this.histogram.width, this.histogram.height);
```

```
            this.context.drawImage(this.image, 0, 0);
            picture = this.context.getImageData(0, 0, this.image.width, this.image.height);
            graphs = analyze(picture.data);
            drawGraphs(this.plot, graphs, view);
            if ((0 < view && view < 4)) {
              picture = this.context.getImageData(0, 0, this.image.width, this.image.height);
              for (i = 0, _ref = picture.data.length; i < _ref; i += 1) {
                if ((_ref2 = i % 4) !== 3 && _ref2 !== (view - 1)) {
                  picture.data[i] = 0;
                }
              }
              this.context.putImageData(picture, 0, 0);
            }
            if (view++ === 4) view = 0;
          };
        }).call(this);</script>
</html>
```

JavaScript

```
(function() {
  var kup, webpage;

  kup = typeof exports !== "undefined" && exports !== null ? require('coffeekup') : window.CoffeeKup;

  webpage = kup.render(function() {
    doctype(5);
    return html(function() {
      head(function() {
        meta({
          charset: 'utf-8'
        });
        title('Histogram');
        return style('body        {color: #FFFFFF;  background-color: #404040}\
             #background {position: absolute; top:  40px; left: 20px}\
             #image      {position: absolute; top:   0px; left: 80px}\
             #picture    {position: absolute; top:   0px; left: 80px}\
             #histogram  {position: absolute; top: 140px; left:  0px}');
      });
      body(function() {
        return div({
          id: 'background'
        }, function() {
          var _ref;
          img({
            id: 'image',
            width: 90,
            height: 90,
            src: (_ref = typeof window !== "undefined" && window !== null ? window.ostrich : void 0) != null ? _ref : '../img/ostrich
          });
          canvas({
            id: 'picture',
            width: 90,
            height: 90
          });
          return canvas({
            id: 'histogram',
            width: 256,
            height: 100,
            onClick: 'onChange()'
          });
        });
      });
      return coffeescript(function() {
        var analyze, drawGraphs, hues, legend, textColor, textFont, textPos, view;
        hues = {
          red: 'rgba(255, 128, 128, 0.5)',
          green: 'rgba(128, 255, 128, 0.5)',
```

```
    blue: 'rgba(128, 128, 255, 0.5)',
    alpha: 'rgba(128, 128, 128, 0.5)'
  };
  legend = [' ', 'R', 'G', 'B', ' '];
  textColor = '#F7C762';
  textFont = '12pt Times';
  textPos = {
    x: 230,
    y: -80
  };
  view = 0;
  analyze = function(data) {
    var bin, bins, i, name, val, _len;
    bins = {
      red: [],
      green: [],
      blue: [],
      alpha: []
    };
    for (name in bins) {
      bin = bins[name];
      for (i = 0; i <= 255; i++) {
        bin[i] = 0;
      }
    }
    for (i = 0, _len = data.length; i < _len; i++) {
      val = data[i];
      switch (i % 4) {
        case 0:
          bins.red[val]++;
          break;
        case 1:
          bins.green[val]++;
          break;
        case 2:
          bins.blue[val]++;
          break;
        case 3:
          bins.alpha[val]++;
      }
    }
    return bins;
  };
  drawGraphs = function(ctx, graphs) {
    var drawPlot;
    drawPlot = function(ctx, plot, color) {
      var x, y, _len;
      ctx.fillStyle = color;
      ctx.beginPath();
      ctx.moveTo(0, 0);
      for (x = 0, _len = plot.length; x < _len; x++) {
        y = plot[x];
        ctx.lineTo(x, y);
      }
      ctx.lineTo(plot.length, 0);
      ctx.closePath();
      return ctx.fill();
    };
    ctx.translate(0, ctx.canvas.height);
    ctx.fillStyle = textColor;
    ctx.font = textFont;
    ctx.fillText(legend[view], textPos.x, textPos.y);
    ctx.scale(1, -1);
    if (view === 0 || view === 1) drawPlot(ctx, graphs.red, hues.red);
    if (view === 0 || view === 2) drawPlot(ctx, graphs.green, hues.green);
    if (view === 0 || view === 3) drawPlot(ctx, graphs.blue, hues.blue);
    if (view === 0 || view === 4) {
      return drawPlot(ctx, graphs.alpha, hues.alpha);
    }
  };
  window.onload = function() {
    var $;
```

```
121             $ = function(element) {
122               return document.getElementById(element);
123             };
124             this.image = $('image');
125             this.canvas = $('picture');
126             this.histogram = $('histogram');
127             this.context = canvas.getContext('2d');
128             this.plot = histogram.getContext('2d');
129             if (!((this.context != null) || (this.plot != null))) {
130               alert('No canvas in this browser.');
131               return;
132             }
133             return window.onChange();
134           };
135           return window.onChange = function() {
136             var graphs, i, picture, _ref, _ref2;
137             this.histogram.width = this.histogram.width;
138             this.plot.clearRect(0, 0, this.histogram.width, this.histogram.height);
139             this.context.drawImage(this.image, 0, 0);
140             picture = this.context.getImageData(0, 0, this.image.width, this.image.height);
141             graphs = analyze(picture.data);
142             drawGraphs(this.plot, graphs, view);
143             if ((0 < view && view < 4)) {
144               picture = this.context.getImageData(0, 0, this.image.width, this.image.height);
145               for (i = 0, _ref = picture.data.length; i < _ref; i += 1) {
146                 if ((_ref2 = i % 4) !== 3 && _ref2 !== (view - 1)) {
147                   picture.data[i] = 0;
148                 }
149               }
150               this.context.putImageData(picture, 0, 0);
151             }
152             if (view++ === 4) view = 0;
153           };
154         });
155       });
156     }, {
157       format: true
158     });
159
160     showDocument(webpage);
161
162 }).call(this);
```

Formats Standalone CoffeeScript Markdown PDF HTML