```julia
begin
    using Plots
    using StatsBase  # for countmap
    using LaTeXStrings # for LaTeX support in L-strings L"..." for plots

    using Images  # for channelview
    using ImageShow  # to display large images
    using Colors
    using ImageIO  # needed together with FileIO
    using FileIO
    using FFTW  # for DCT implementation

    # using SparseArrays  # For a previous attempt at coding Exact Recompression in Julia
end
```

Histogram Method

```
ijg_dct (generic function with 1 method)
```

```julia
function ijg_dct(X)
    @assert size(X) == (8, 8)
    T = [
        8192 11363 10703 9633 8192 6437 4433 2260
        8192 9633 4433 -2259 -8192 -11362 -10704 -6436
        8192 6437 -4433 -11362 -8192 2261 10704 9633
        8192 2260 -10703 -6436 8192 9633 -4433 -11363
        8192 -2260 -10703 6436 8192 -9633 -4433 11363
        8192 -6437 -4433 11362 -8192 -2261 10704 -9633
        8192 -9633 4433 2259 -8192 11362 -10704 6436
        8192 -11363 10703 -9633 8192 -6437 4433 -2260
    ]
    V = [
        -2^23 2^10 2^10 2^10 0 2^10 2^10 2^10
        -2^23 2^10 2^10 2^10 0 2^10 2^10 2^10
        -2^23 2^10 2^10 2^10 0 2^10 2^10 2^10
        -2^23 2^10 2^10 2^10 0 2^10 2^10 2^10
        -2^23 2^10 2^10 2^10 0 2^10 2^10 2^10
        -2^23 2^10 2^10 2^10 0 2^10 2^10 2^10
        -2^23 2^10 2^10 2^10 0 2^10 2^10 2^10
        -2^23 2^10 2^10 2^10 0 2^10 2^10 2^10
    ]
    return (((T' * (((X * T) + V) .>> 11)) .+ (2^14)) .>> 15) .>> 3
    # return fld.(fld.(T' * fld.(X * T + V, 2^11) .+ 2^14, 2^15), 8)
    # clamp.(
    #   round.(Int, inv(T) * ((2^18 .* X .- 2^17) * inv(T') .* 2^11 .- 2^10)),
    #   -1023,
    #   1024
    # )
end
```

```
vec_rowmajor (generic function with 1 method)
```

```julia
function vec_rowmajor(X)
    # The default vec() method traverses the matrix X in column-major order.
    # permutedims(., (2, 1)) is equivalent to transpose(.) but works for non-numeric matrices too
    return vec(permutedims(X, (2, 1)))
end
```

rgb_to_ycbcr

Converts Tuple{UInt8, UInt8, UInt8} to Tuple{UInt8, UInt8, UInt8}

```julia
"""
Converts Tuple{UInt8, UInt8, UInt8} to Tuple{UInt8, UInt8, UInt8}
"""
function rgb_to_ycbcr((r, g, b))
    @assert typeof(r) == UInt8
    @assert typeof(g) == UInt8
    @assert typeof(b) == UInt8
    y = (19595 * r + 38470 * g + 7471 * b + 2^15) .>> 16
    cb = (-11058 * r - 21710 * g + 32768 * b + 2^23 + 2^15 - 1) .>> 16
    cr = (32768 * r - 27439 * g - 5329 * b + 2^23 + 2^15 - 1) .>> 16
    return (UInt8(y), UInt8(cb), UInt8(cr))
end
```

`split_into_blocks_and_perform_dct`

`image`: A single channel of pixels.

Outputs a matrix where each column represents a block, and each column is a length-64 vector representing from top to bottom (F{0,0}, F{0,1}, ..., F{0,7}, F{1,0}, ..., F{7,7}) for that block

```julia
"""
`image`: A single channel of pixels.

Outputs a matrix where each column represents a block, and each column is a length-64 vector representing from top to bottom (F{0,0}, F{0,1}, ..., F{0,7}, F{1,0}, ..., F{7,7}) for that block
"""
function split_into_blocks_and_perform_dct(image)
    H, W = size(image)
    @assert H % 8 == 0 && W % 8 == 0
    count_blocks_vertical = div(H, 8)
    count_blocks_horizontal = div(W, 8)
    count_blocks_total = count_blocks_vertical * count_blocks_horizontal
    Fs = zeros((64, count_blocks_total))  # one column per block; from top to bottom of each column we have (F_{0,0}, F_{0,1}, ..., F_{0,7}, F_{1,0}, ..., F_{7,7}) for that block

    for block_i in range(0, count_blocks_vertical - 1)  # zero-indexed
        for block_j in range(0, count_blocks_horizontal - 1)  # zero-indexed
            # The 1+ in the indices is to convert from zero-indexed to one-indexed
            X = image[(1 + 8 * block_i):(8 + 8 * block_i), (1 + 8 * block_j):(8 + 8 * block_j)]

            F = ijg_dct(X)
            block_number = block_i * count_blocks_horizontal + block_j
            # To be compatible with plot()'s API later, we use row-major order traversal. This traverses elements of F in zero-index order {(0,0), (0,1), ..., (0,7), (1,0), ..., (7,7)}.
            Fs[:, 1 + block_number] = vec_rowmajor(F)
        end
    end

    return Fs
end
```

```
histogram_grid_titles =
1×64 Matrix{String}:
 "Y Histogram of \$\\mathbf{F}_{0, 0}\$"  …  "Y Histogram of \$\\mathbf{F}_{7, 7}\$"
```

```julia
histogram_grid_titles = reshape(
    vec_rowmajor(map(
        ij -> "Y Histogram of " * L"\mathbf{F}_{%$(ij[1] - 1), %$(ij[2] - 1)}",
        keys(zeros(8, 8))
    )),
    (1, :)
)  # a row matrix is expected by plot(), hence we reshape
```

```
load_greyscale_and_apply_dct (generic function with 1 method)
```

```
function load_greyscale_and_apply_dct(name)
    file = load(name)
    file = reinterpret(UInt8, file)  # size: H x W
    return split_into_blocks_and_perform_dct(file)
end
```

```
load_colour_and_apply_dct_Y (generic function with 1 method)
```

```
begin
    function rgb_to_tuple(rgb)
        return (
            reinterpret(UInt8, red(rgb)),
            reinterpret(UInt8, green(rgb)),
            reinterpret(UInt8, blue(rgb))
        )
    end

    function load_colour_and_apply_dct_Y(name)
        file = load(name)
        file = rgb_to_tuple.(file)  # each pixel location is now a (R, G, B) tuple
        file = rgb_to_ycbcr.(file)  # each pixel location is now a (Y, Cb, Cr) tuple
        file_Y = map(x -> x[1], file)
        # file_Cb = map(x -> x[2], file)
        # file_Cr = map(x -> x[3], file)
        return split_into_blocks_and_perform_dct(file_Y)
    end
end
```

```
▸ Dict("test1.png" ⇒ 64×190512 Matrix{Float64}:
                      -11.0  -16.0  -19.0  -2.0  -5.0  -10.0  …  -539.0  -530.0  -537.0
```

```
begin
    greyscale_names = ("test1.png", "test2.png", "test3.png")
    greyscale_name_to_Fs = Dict(name => load_greyscale_and_apply_dct(name) for name
    in greyscale_names)
end
```

The following cell produces the mega-plots of the greyscale images (Figure 4).

```julia
let
    ######################
    # Plotting histograms #
    ######################
    for name in greyscale_names
        Fs = greyscale_name_to_Fs[name]
        # In order: histograms of F_{0,0}, F_{0,1}, ..., F_{0,7}, F_{1,0}, ..., F_{7,7}
        histograms = [bar(countmap(Fs[i, :]); label=:none) for i in 1:64]
        plot(
            histograms...;
            titles=histogram_grid_titles,
            layout=(8, 8),
            size=(2400, 1800),
            tick_direction=:out,
            x_rotation=45,
            suptitle=L"\mathbf{F}_{i,j}" * " histograms of " * name
        )
        savefig("dct_" * name)

        # In order: log-histograms of F_{0,0}, F_{0,1}, ..., F_{0,7}, F_{1,0}, ..., F_{7,7}
        histograms_vlim = [bar(countmap(Fs[i, :]); label=:none, ylim=(0, 50)) for i in 1:64]
        plot(
            histograms_vlim...;
            titles=histogram_grid_titles,
            layout=(8, 8),
            size=(2400, 1800),
            tick_direction=:out,
            x_rotation=45,
            suptitle=L"\mathbf{F}_{i,j}" * " histograms of " * name
        )
        savefig("dct_ylim50_" * name)

        # In order: log-histograms of F_{0,0}, F_{0,1}, ..., F_{0,7}, F_{1,0}, ..., F_{7,7}
        histograms_log = [bar(countmap(Fs[i, :]); yscale=:log10, label=:none) for i in 1:64]
        plot(
            histograms_log...;
            titles=histogram_grid_titles,
            layout=(8, 8),
            size=(2400, 1800),
            tick_direction=:out,
            x_rotation=45,
            suptitle=L"\mathbf{F}_{i,j}" * " histograms (log counts) of " * name
        )
        savefig("dct_log_" * name)
    end
end
```

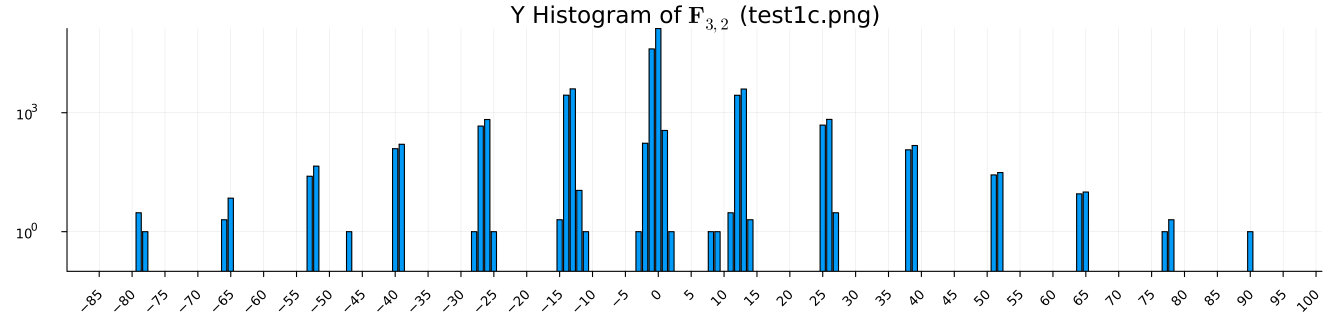
```
Dict("test3c.png" ⇒ 64×190512 Matrix{Float64}:
                     -811.0  -831.0  -660.0  -618.0  -678.0  …  -835.0  -837.0  -813.0
```

```
begin
    colour_names = ("test1c.png", "test2c.png", "test3c.png")
    colour_name_to_Fs = Dict(name => load_colour_and_apply_dct_Y(name) for name in
    colour_names)
end
```

The following cell produces the mega-plots of the colour images (Figure 4).

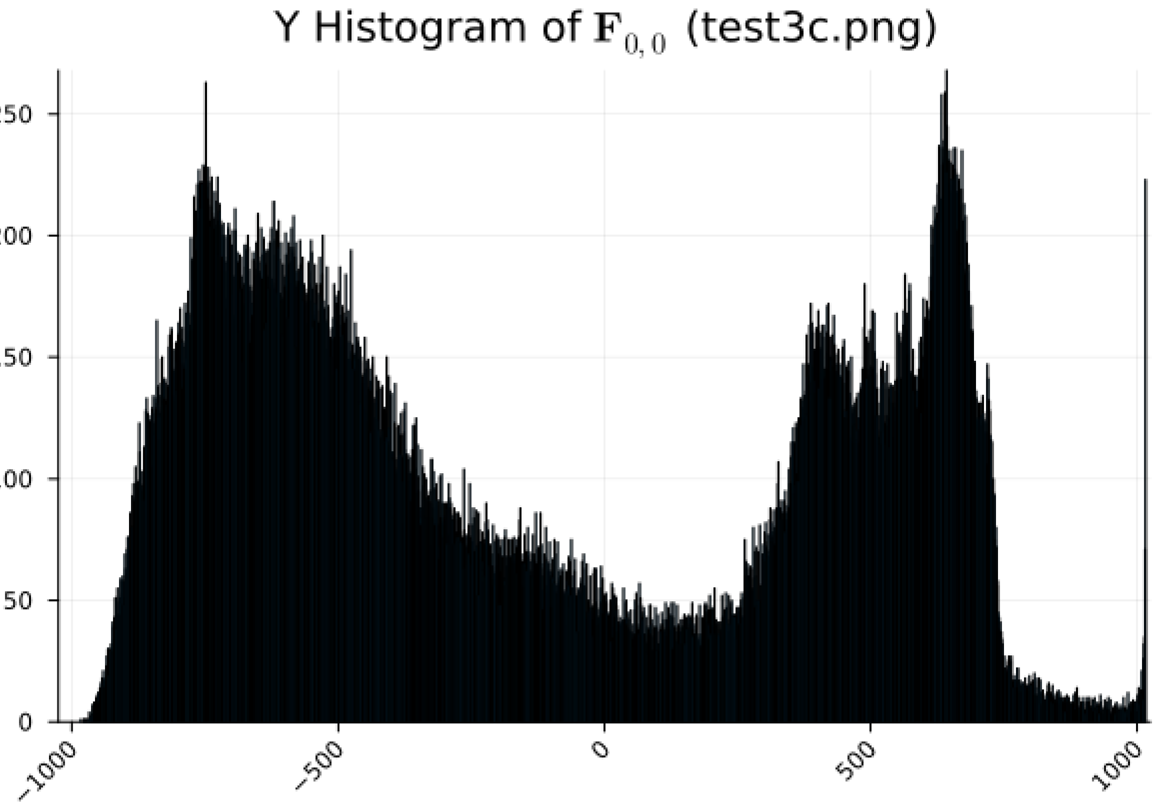
```julia
let
    ########################
    # Plotting histograms #
    ########################
    for name in colour_names
        Fs = colour_name_to_Fs[name]
        # In order: histograms of F_{0,0}, F_{0,1}, ..., F_{0,7}, F_{1,0}, ..., F_{7,7}
        histograms = [bar(countmap(Fs[i, :]); label=:none) for i in 1:64]
        plot(
            histograms...;
            titles=histogram_grid_titles,
            layout=(8, 8),
            size=(2400, 1800),
            tick_direction=:out,
            x_rotation=45,
            suptitle=L"\mathbf{F}_{i,j}" * " histograms of " * name
        )
        savefig("dct_Y_" * name)

        # In order: log-histograms of F_{0,0}, F_{0,1}, ..., F_{0,7}, F_{1,0}, ..., F_{7,7}
        histograms_vlim = [bar(countmap(Fs[i, :]); label=:none, ylim=(0, 50)) for i in 1:64]
        plot(
            histograms_vlim...;
            titles=histogram_grid_titles,
            layout=(8, 8),
            size=(2400, 1800),
            tick_direction=:out,
            x_rotation=45,
            suptitle=L"\mathbf{F}_{i,j}" * " histograms of " * name
        )
        savefig("dct_ylim50_Y_" * name)

        # In order: log-histograms of F_{0,0}, F_{0,1}, ..., F_{0,7}, F_{1,0}, ..., F_{7,7}
        histograms_log = [bar(countmap(Fs[i, :]); yscale=:log10, label=:none) for i in 1:64]
        plot(
            histograms_log...;
            titles=histogram_grid_titles,
            layout=(8, 8),
            size=(2400, 1800),
            tick_direction=:out,
            x_rotation=45,
            suptitle=L"\mathbf{F}_{i,j}" * " histograms (log counts) of " * name
        )
        savefig("dct_log_Y_" * name)
    end
end
```

The following cell is used for exploring the histograms.



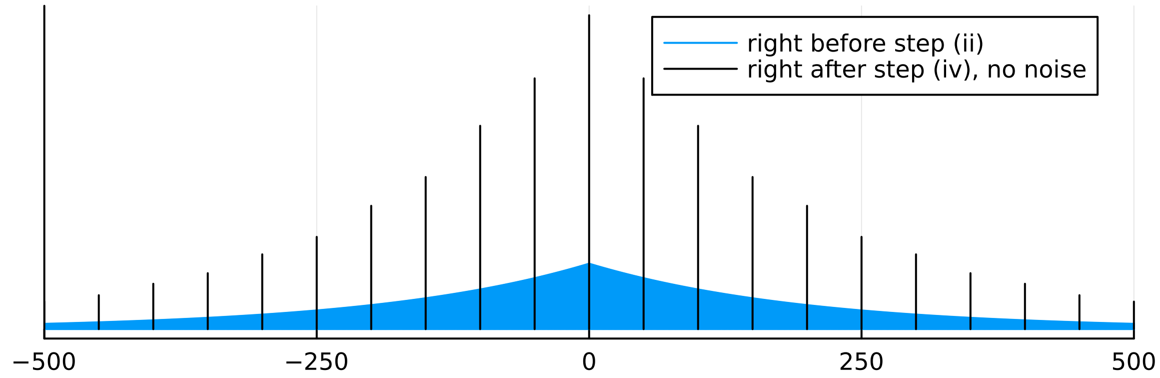

```julia
begin
    name = "test1c.png"
    is_colour = name in colour_names
    histogram_i = 27
    x_bound = 150
    log_scale = true
    # y_max = 50
    bar(
        countmap(filter(
            f -> abs(f) < x_bound,
            (is_colour ? colour_name_to_Fs : greyscale_name_to_Fs)[name][histogram_i, :]
        ));
        label=:none,
        y_scale=log_scale ? :log10 : :identity,
        title="$(histogram_grid_titles[histogram_i]) ($(name))",
        size=(1200,300),
        left_margin= 5 * Plots.PlotMeasures.mm,
        bottom_margin= 10 * Plots.PlotMeasures.mm,
        tick_direction=:out,
        xticks=-x_bound:5:x_bound,
        xrotation=45,
        # ylim=(0, y_max)
    )
end
```

The following cell is used for generating Figure 1.



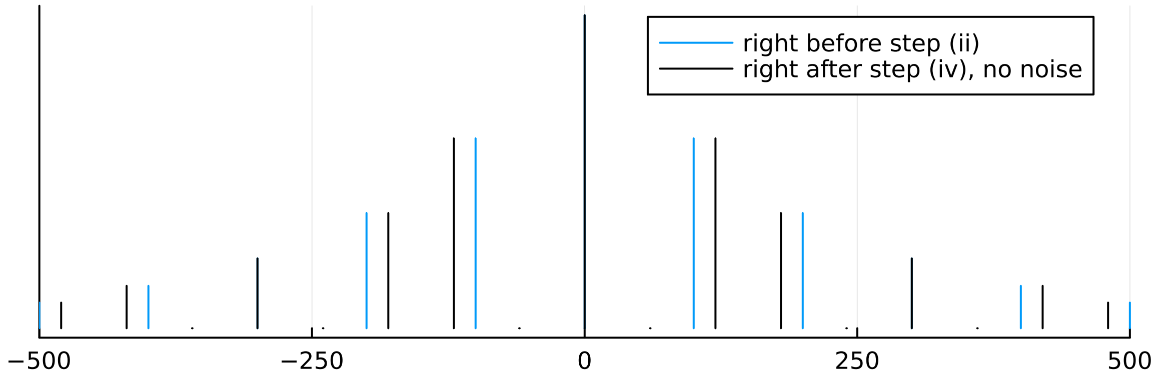

```julia
let
    name = "test3c.png"
    is_colour = name in colour_names
    histogram_i = 1
    x_bound = 9999
    log_scale = false
    # y_max = 50
    bar(
        countmap(filter(
            f -> abs(f) < x_bound,
            (is_colour ? colour_name_to_Fs : greyscale_name_to_Fs)[name][histogram_i, :]
        ));
        label=:none,
        y_scale=log_scale ? :log10 : :identity,
        title="$(histogram_grid_titles[histogram_i]) ($(name))",
        size=(600,400),
        xlim=(-1025,1025),
        # left_margin= 5 * Plots.PlotMeasures.mm,
        # bottom_margin= 10 * Plots.PlotMeasures.mm,
        tick_direction=:out,
        # xticks=-x_bound:1:x_bound,
        xrotation=45,
        # ylim=(0, y_max),
        fmt=:png,
    )
end
```

The following cell is used for generating Figure 2.



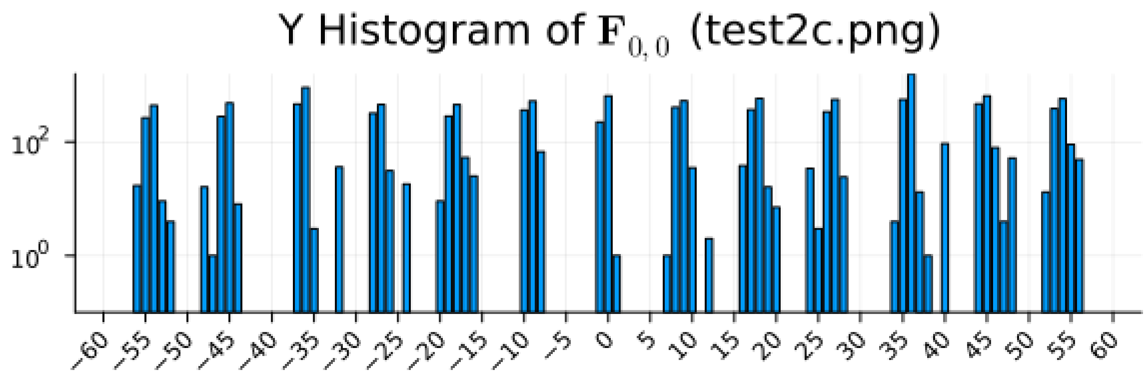

```
let
    x = -1023:1:1024
    b = 200
    q = 50
    max_k = floor(1024 / q) # maximum multiple_number of q that fits within -1023:1024
    pdf = exp.(- abs.(x) / b) ./ b
    plot(
        x,
        pdf * 10,  # artificially scale the blue pdf, otherwise it can't be seen
        lines=:stem,
        label="right before step (ii)",
        size=(600, 200),
        xlim=(-500,500),
        yticks=:none
    )

    function sum_if_same_multiple(multiple_number, list_i, list)
        sum = 0
        for (i, pdf) in zip(list_i, list)
            if round(i / q) == multiple_number
                sum += pdf
            end
        end
        return sum
    end
    plot!(
        -max_k * q:q:max_k * q,
        [sum_if_same_multiple(k, x, pdf) for k in -max_k:max_k],
        lines=:stem,
        color=:black,
        label="right after step (iv), no noise",
        left_margin= 5 * Plots.PlotMeasures.mm,
        right_margin= 5 * Plots.PlotMeasures.mm,
    )
end
```

The following cell is used for generating Figure 3.




```julia
let
    x = -1000:100:1000
    b = 200
    q = 60
    max_k = floor(1024 / q) # maximum multiple_number of q that fits within
    -1023:1024
    pdf = exp.(- abs.(x) / b) ./ b
    plot(
        x,
        pdf,
        lines=:stem,
        label="right before step (ii)",
        size=(600, 200),
        xlim=(-500,500),
        yticks=:none
    )

    function sum_if_same_multiple(multiple_number, list_i, list)
        sum = 0
        for (i, pdf) in zip(list_i, list)
            if round(i / q) == multiple_number
                sum += pdf
            end
        end
        return sum
    end
    plot!(
        -max_k * q:q:max_k * q,
        [sum_if_same_multiple(k, x, pdf) for k in -max_k:max_k],
        lines=:stem,
        color=:black,
        label="right after step (iv), no noise",
        left_margin= 5 * Plots.PlotMeasures.mm,
        right_margin= 5 * Plots.PlotMeasures.mm,
    )
end
```

The following cell was used to generate Figures 5-8, tweaking `name`, `x_bound`, `x_ticks` and `size` accordingly.




```
let
    name = "test2c.png"
    is_colour = name in colour_names
    histogram_i = 1  # 5, 27, 29
    x_bound = 60
    log_scale = true
    # y_max = 50
    bar(
        countmap(filter(
            f -> abs(f) < x_bound,
            (is_colour ? colour_name_to_Fs : greyscale_name_to_Fs)[name]
            [histogram_i, :]
        ));
        label=:none,
        y_scale=log_scale ? :log10 : :identity,
        title="$(histogram_grid_titles[histogram_i]) ($(name))",
        size=(600,200),
        left_margin= 5 * Plots.PlotMeasures.mm,
        top_margin= 3 * Plots.PlotMeasures.mm,
        bottom_margin= 7 * Plots.PlotMeasures.mm,
        tick_direction=:out,
        xticks=-x_bound:5:x_bound,
        xrotation=45,
        # ylim=(0, y_max),
        fmt=:png,
    )
end
```

Determining the parameters used in previous compressions

Implementation of Exact JPEG recompression. An attempt was made to program this in Julia, but it was way too slow (both in terms of performance and programming speed). I have switched to Python for this part.

```
# let
#   function quantised_interval(w_interval, qfcij=50)
#       print(w_interval)
#       w_bot, w_top = w_interval
#       if w_bot < 0 || w_bot % qfcij == 0
#           result_bot = div(w_bot, qfcij)
#       else
#           result_bot = div(w_bot, qfcij) + 1
#       end
#       if w_top < 0 || w_top % qfcij == 0
#           result_top = div(w_bot, qfcij) + 1
#       else
#           result_top = div(w_bot, qfcij)
#       end
#       print(" after quantisation gives ")
#       if result_bot > result_top
#           println("<INVALID>")
#       else
#           println((result_bot, result_top))
#       end
#   end
#   quantised_interval((1,2))
#   quantised_interval((-49, 49))
#   quantised_interval((-99, -51))
#   quantised_interval((51,99))
#   quantised_interval((50,99))
#   quantised_interval((49,99))
#   quantised_interval((49,100))
#   quantised_interval((50,100))
#   quantised_interval((51,100))
#   quantised_interval((51,199))
# end
```

```
# begin
#   # Everything to do with intervals

#   emptyinterval = ()

#   function in_interval(y, bar_x)
#       if bar_x == emptyinterval
#           return false
#       end
#       bar_x_bot, bar_x_top = bar_x
#       return y >= bar_x_bot && y <= bar_x_top
#   end

#   function union_intervals(bar_x, bar_y)
#       if bar_x == emptyinterval
#           return bar_y
#       elseif bar_y == emptyinterval
#           return bar_x
#       end
#       bar_x_bot, bar_x_top = bar_x
#       bar_y_bot, bar_y_top = bar_y
#       return (min(bar_x_bot, bar_y_bot), max(bar_x_top, bar_y_top))
#   end

#   function intersect_intervals(bar_x, bar_y)
#       if bar_x == emptyinterval || bar_y == emptyinterval
#           return emptyinterval
#       end
#       bar_x_bot, bar_x_top = bar_x
#       bar_y_bot, bar_y_top = bar_y
#       if bar_x_bot > bar_y_top || bar_y_bot > bar_x_top
#           return emptyinterval
#       else
#           return (max(bar_x_bot, bar_y_bot), min(bar_x_top, bar_y_top))
#       end
#   end
# end
```

```
# begin
#   # Everything to do with colour-space conversion

#     # The rgb_to_ycbcr((r, g, b)) function is defined before.
#   # It converts Tuple{UInt8, UInt8, UInt8} to Tuple{UInt8, UInt8, UInt8} (RGB to YCbCr).
#   # We want to map each RGB value u_xy to the set of all YCbCr values ddot_v_xy that maps to u_xy via rgb_to_ycbcr.
#   """
#   Converts Tuple{UInt8, UInt8, UInt8} to Tuple{UInt8, UInt8, UInt8}
#   """
#   function ycbcr_to_rgb((y, cb, cr))
#       @assert typeof(y) == UInt8
#       @assert typeof(cb) == UInt8
#       @assert typeof(cr) == UInt8
#       r = y + div(91881 * (cr - 128) + 2^15, 2^16)
#       g = y + div(-22553 * (cb - 128) - 46802 * (cr - 128) + 2^15, 2^16)
#       b = y + div(116130 * (cb - 128) + 2^15, 2^16)
#       r = clamp(r, 0, 255)
#       g = clamp(g, 0, 255)
#       b = clamp(b, 0, 255)
#       return (UInt8(r), UInt8(g), UInt8(b))
#   end
# end
```

```
# begin
#   function interpret_int_as_tup(n)
#       return (UInt8(n >> 16), UInt8(n >> 8 & 0xff), UInt8(n & 0xff))
#   end

#   function interpret_tup_as_int((x, y, z))
#       return Int(x << 16 + y << 8 + z)
#   end

#   uint8r = range(UInt8(0), UInt8(255))
#   results = spzeros(Bool, 2^24, 2^24)  # columns: RGB values; rows: YCrCb values that map to the RGB value represented by that column

#   # rgb_to_set_of_ycbcr = Dict((r, g, b) => Set() for r in uint8r, g in uint8r, b in uint8r)
#   for y in uint8r
#   for cb in uint8r
#   for cr in uint8r
#       (r, g, b) = ycbcr_to_rgb((y, cr, cb))
#       ycbcr_int = interpret_tup_as_int((y, cb, cr))  # range: 0..(2^24 - 1)
#       rgb_int = interpret_tup_as_int((r, g, b))  # range: 0..(2^24 - 1)
#       results[1 + rgb_int, 1 + ycbcr_int] = true  # 1+ to deal with one-indexed Julia. Indices are ranged 1..2^24
#   end
#       break
#   end
#       break
#   end

# end
```