EmojiCloud: a Tool for Emoji Cloud Visualization

Yunhe Feng[1], Cheng Guo[1], Bingbing Wen[1], Peng Sun[2], Yufei Yue[3], Dingwen Tao[4]

University of Washington[1]
The Chinese University of Hong Kong, Shenzhen[2]
Amazon[3]
Washington State University[4]

14th July, 2022

Representation Problems of Word Cloud of Emojis

Figure 1: Word cloud of emojis

- colors (⚽ ❤️)
- directionalities (😂 🌸)
- textures (🙌 💚)

Inaccurate Representations Lead to Misunderstanding

- When emojis  are upside down, they turn into  that conveys different sentiments and meanings.
- Miscolored emojis such as  may cause the problem of personal identity representations.

EmojiCloud Basic Idea

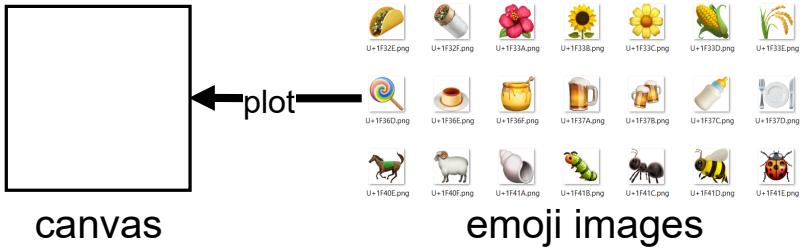


Figure 2: EmojiCloud basic idea

EmojiCloud Challenges

- Where and how to collect emoji images?
  - Emoji appearances are different across different vendors.
  - Many platform vendors exist (e.g., Twitter, Apple, Meta, and Google).
- How to determine emoji plotting sizes?
  - Emoji frequency and the canvas size.
- How to design a flexible canvas that supports various shapes?
  - Rectangle, square, ellipse, circle, and masked images.
- How to design the emoji layout?
  - Make EmojiCloud dense and beautiful.
  - Highlight the most important emojis.

Emoji Image Retrieval

Smileys & Emotion

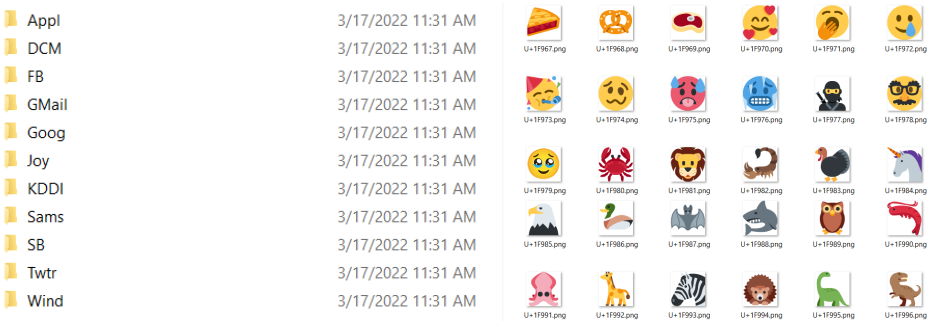
face-smiling

№	Code	Browser	Appl	Goog	FB	Wind	Twtr	Joy	Sams	GMail	SB	DCM	KDDI	CLDR Short Name
1	U+1F600										—	—	—	grinning face
2	U+1F603													grinning face with big eyes
3	U+1F604											—	—	grinning face with smiling eyes
4	U+1F601													beaming face with smiling eyes
5	U+1F606										—		—	grinning squinting face
6	U+1F605										—		—	grinning face with sweat
7	U+1F923									—	—	—	—	rolling on the floor laughing

Figure 3: Unicode full emoji list

Source: https://unicode.org/emoji/charts/full-emoji-list.html

Emoji Image Retrieval



(a) Emoji vendors (b) Emoji image examples

Figure 4: Emoji organized by vendors and emoji image examples

Emoji Image Preprocessing

(a) Raw image (b) Bounding box (c) Unoccupied pos.

Figure 5: Preprocessing original emoji images by determining bounding boxes and marking unoccupied pixel positions (colored as black in Figure 5(c))

Emoji Size Calculation

We use a quintuple $e = (a, b, w, E, U)$ to represent an emoji, where a, b, w are the width, height, edge-level frequency weight of emoji e. $E_{x,y}$ and $U_{x,y} \in \{0, 1\}$ represent the pixel value (RGBA) and the pixel unoccupied status at the coordinate (x, y).

$$s \geq \sum_{i=1}^{|e|} w_i^2 * a_i * b_i * r^2 = \sum_{i=1}^{|e|} w_i^2 * c * r^2 \tag{1}$$

where s is the drawable canvas area and c is the constant area of preprocessed emojis.

- a *possible* maximum edge rescale ratio r can be $\sqrt{s/(c * \sum_{i=1}^{|e|} w_i^2)}$.
- the rescaled width and height: $a'_i = a_i * w_i * r$; $b'_i = b_i * w_i * r$.
- The edge rescale ratio r decays at a rate of 0.9 if there is not enough room to plot all emojis on the canvas.

Emoji Canvas Definition

- We use a quintuple (m, n, s, C, V) to represent a canvas, where $m * n$ defines a rectangle bounding box of the canvas; s is the drawable canvas size; C and V represent pixel values and the painting eligibility.
- $C_{x,y}$ represents canvas pixel values at the coordinate (x, y).
- $V_{x,y} \in \{0, 1\}$ indicates the painting eligibility of (x, y), where $x \in [1, m]$ and $y \in [1, n]$. The design of V controls the drawable shape (e.g., a circle or an ellipse) on the canvas.

EmojiCloud Layout

Algorithm 1: EmojiCloud Layout

Input: e: a list of emojis; (m, n, s, C, V): a canvas with width m, height n, drawable size s, pixel values C, and pixel painting eligibility V on the canvas; c: the standardized size of emoji images;
Output: C: an emoji cloud image;
$e \leftarrow$ sort the emoji list e by emoji weights $w = [w_1, w_2, .., w_{|e|}]$ in reverse order;
$r \leftarrow \sqrt{s/(c * \sum_{i=1}^{|e|} w_i^2)}$; // rescale ratio in Equation
for $x = 1 \rightarrow m$ **do** // x coordinate of canvas
  for $y = 1 \rightarrow n$ **do** // y coordinate of canvas
    if $V_{x,y} = 1$ **then** // canvas pixel is eligible for painting
      append (x, y) into the canvas pixel coordinate list p_c; // build p_c
$p_c \leftarrow$ sort p_c by the Euclidean distance between $(x, y) \in p_c$ and the canvas center $(m/2, n/2)$;
$count \leftarrow 0$; // count of plotted emojis
while $count < |e|$ **do** // not all emoji have been plotted
  $count \leftarrow 0$; // no emoji has been plotted
  for $i = 1 \rightarrow |e|$ **do** // iterate the emojis sorted by weights in reverse order
    $(a, b, w, E^i, U^i) \leftarrow e_i$; // parse e_i into its quintuple representation
    $a, b \leftarrow a_i * w_i * r$; $b_c' \leftarrow b_i * w_i * r$; // rescale width and height of e_i
    $E^i, U^i \leftarrow$ update E^i, U^i based on r ; // rescale E^i, U^i based on r
    for $(x, y) \in p_c$ **do** // (x, y) is where the center of e_i to be located
      $flag \leftarrow True$; // indicate the possibility of plotting e_i
      $p_t \leftarrow [\ ]$; // a list of canvas temporal coordinates to plot e_i
      for $x' = 1 \rightarrow a$ **do** // x coordinate of emoji image
        for $y' = 1 \rightarrow b$ **do** // y coordinate of emoji image
          if $U^i_{x',y'} = 0$ **then** // emoji pixel (x', y') is not unoccupied
            $x_o \leftarrow x' - a/2$; $y_o \leftarrow y' - b/2$; // the offsets to e_i center
            $x_t \leftarrow x + x_o$; $y_t \leftarrow y + y_o$; // canvas temporal coordinate for e_i
            append (x_t, y_t) to p_t;
            if $V_{x_t,y_t} = 0$ **then** // canvas pixel is not eligible for painting
              $flag \leftarrow False$; // no room to plot e_i at (x, y)
              break; // iterate the next $(x, y) \in p_c$
      if $flag = True$ **then** // the emoji e_i can be plot at (x, y)
        for $(x_t, y_t) \in p_t$ **do** // iterate temporal pixel coordinates
          $C_{x_t,y_t} \leftarrow E^i_{x_t - x + a/2, y_t - y + b/2}$; // plot emoji e_i
          $V_{x_t,y_t} \rightarrow 0$; // set painting eligibility as negative
          remove (x_t, y_t) from p_t; // delete (x_t, y_t) for computing efficiency
        $count \leftarrow count + 1$; // increase the number of plotted emoji by 1
        break;
  $r \leftarrow r * 0.9$; // decay the edge rescale ratio by 0.9
return C

Default and Arbitrary Canvas

- We set the default canvas shape as an $m * n$ rectangle, and all pixel coordinates within the rectangle are eligible to draw emojis. The painting eligibility $V_{x,y}$ is set as 1 for all $x \in [1, m]$ and $y \in [1, n]$.
- Users are allowed to specify arbitrary canvas drawable shapes by configuring the painting eligibility $V_{x,y}$ for pixel coordinate (x, y) on the canvas.

Ellipse Canvas

Suppose we have a drawable ellipse area within an $m * n$ rectangle bounding box for plotting emojis. The semi-major and semi-minor axes' lengths are expressed as $m/2$ and $n/2$. The center pixel coordinate is expressed as $(m/2, n/2)$. If pixel coordinate (x, y) on canvas satisfies the following inequality, $V_{x,y}$ is set as 1.

$$(\frac{x - \frac{m}{2}}{\frac{m}{2}})^2 + (\frac{y - \frac{n}{2}}{\frac{n}{2}})^2 \leq 1 \tag{2}$$

Otherwise, the coordinate (x, y) is outside of the ellipse, and the corresponding $V_{x,y}$ is set as 0. When m equals n, a circle canvas is defined.

Masked Canvas

- We determine a $m * n$ bounding box of the image by removing the surrounding transparent pixels.
- We detect the image contour and draw a boundary accordingly (e.g.,  $\rightarrow$ ).
  - We scan the alpha values of pixels in the preprocessed image by row and by column respectively.
  - We identify pixels that cause a alpha value change greater than a threshold θ (by default $\theta = 10$) as boundary pixels.
  - After all boundary pixels are determined, they will be colored by specified colors.

EmojiCloud Inclusive Design

- EmojiCloud is flexible and inclusive to handle emoji images designed by seven vendors (i.e., Apple, Google, Meta, Windows, Twitter, JoyPixels, and Samsung).
- Users can customize and combine emojis based on their requirements.
  - a red apple emoji (U+1F34E)  can be replaced by  for marketing campaigns.
  - The sauropod (U+1F995)  and T-Rex emoji (U+1F996)  can be combined as  if it is not necessary to distinguish dinosaur species.

Implementation and Open Source

- EmojiCloud has been published through Python Package Index (PyPI).
- *pip install EmojiCloud* *from EmojiCloud import EmojiCloud*
- An EmojiCloud tutorial is available at `https://pypi.org/project/EmojiCloud/`.

Code for Different Canvases

```python
from EmojiCloud import EmojiCloud

# set emoji weights by a dict with key: emoji by codepoint, value: weight
dict_weight = {'1f1e6-1f1e8': 1.1, '1f4a7': 1.2, '1f602': 1.3, '1f6f4': 1.4, '1f6f5': 1.5, '1f6f6': 1.6, '1f6f7': 1.7, '1f6f8': 1.8, '1f6f9': 1.9,
'1f6fa': 2.0, '1f6fb': 2.1, '1f6fc': 2.2, '1f7e0': 2.3, '1f9a2': 2.4, '1f9a3': 2.5, '1f9a4': 2.6, '1f9a5': 2.7, '1f9a6': 2.8, '1f9a8': 2.9, '1f9a9': 3.0}

# emoji vendor
emoji_vendor = 'Twitter'

# masked canvas
img_mask = 'twitter-logo.png'
thold_alpha_contour = 10
contour_width = 5
contour_color = (0, 172, 238, 255)
saved_emoji_cloud_name = 'emoji_cloud_masked.png'
EmojiCloud.plot_masked_canvas(img_mask, thold_alpha_contour, contour_width, contour_color, emoji_vendor, dict_weight, saved_emoji_cloud_name)

# rectangle canvas
canvas_w = 72*10
canvas_h = 72*4
saved_emoji_cloud_name = 'emoji_cloud_rectangle.png'
EmojiCloud.plot_rectangle_canvas(canvas_w, canvas_h, emoji_vendor, dict_weight, saved_emoji_cloud_name)

# ellipse canvas
canvas_w = 72*10
canvas_h = 72*4
saved_emoji_cloud_name = 'emoji_cloud_ellipse.png'
EmojiCloud.plot_ellipse_canvas(canvas_w, canvas_h, emoji_vendor, dict_weight, saved_emoji_cloud_name)
```

Figure 6: EmojiCloud for plotting different canvases

Visualization on Different Canvases

(a) Rectangle (b) Ellipse (c) Mask

Figure 7: EmojiCloud on different canvases

Code for Different Emoji Vendors

```python
from EmojiCloud import EmojiCloud

# set emoji weights by a dict with key: emoji by codepoint, value: weight
dict_weight = {'U+1F600': 1.1, 'U+1F601': 1.2, 'U+1F602': 1.3, 'U+1F603': 1.4, 'U+1F604': 1.5, 'U+1F605': 1.6, 'U+1F606': 1.7, 'U+1F607': 1.8, 'U+1F608': 1.9, 'U+1F609': 2.0, 'U+1F610': 2.1, 'U+1F612': 2.2, 'U+1F613': 2.3, 'U+1F614': 2.4, 'U+1F616': 2.5, 'U+1F617': 2.6, 'U+1F618': 2.7, 'U+1F619': 2.8, 'U+1F620': 2.9, 'U+1F621': 3.0, 'U+1F622': 3.1, 'U+1F624': 3.2, 'U+1F625': 3.3, 'U+1F628': 3.4, 'U+1F629': 3.5, 'U+1F630': 3.6, 'U+1F631': 3.7, 'U+1F632': 3.8, 'U+1F633': 3.9, 'U+1F634': 4.0, 'U+1F635': 4.1, 'U+1F637': 4.2, 'U+1F638': 4.3, 'U+1F639': 4.4, 'U+1F640': 4.5, 'U+1F641': 4.6, 'U+1F642': 4.7, 'U+1F643': 4.8, 'U+1F644': 4.9, 'U+1F910': 5.0, 'U+1F911': 5.1, 'U+1F912': 5.2, 'U+1F913': 5.3, 'U+1F914': 5.4, 'U+1F915': 5.5, 'U+1F917': 5.6, 'U+1F920': 5.7, 'U+1F921': 5.8, 'U+1F922': 5.9, 'U+1F923': 6.0, 'U+1F924': 6.1, 'U+1F925': 6.2, 'U+1F927': 6.3, 'U+1F929': 6.4, 'U+1F970': 6.5, 'U+1F971': 6.6, 'U+1F973': 6.7, 'U+1F974': 6.8, 'U+1F975': 6.9, 'U+1F976': 7.0, 'U+1FAE1': 7.1, 'U+1FAE2': 7.2, 'U+1FAE3': 7.3}

# emoji vendors
list_vendor = ['Google', 'Windows', 'Apple', 'Twitter', 'Meta', 'JoyPixels', 'Samsung']
for emoji_vendor in list_vendor:
    # circle canvas
    canvas_w = 72*10
    canvas_h = 72*10
    saved_emoji_cloud_name = 'emoji_cloud_circle_' + emoji_vendor + '.png'
    EmojiCloud.plot_ellipse_canvas(canvas_w, canvas_h, emoji_vendor, dict_weight, saved_emoji_cloud_name)
```

Figure 8: EmojiCloud for plotting different vendors

Visualization for Different Emoji Vendors

(a) Twitter

(b) Apple

(c) Google

(d) Windows

(e) JoyPixels

(f) Meta

(g) Samsung

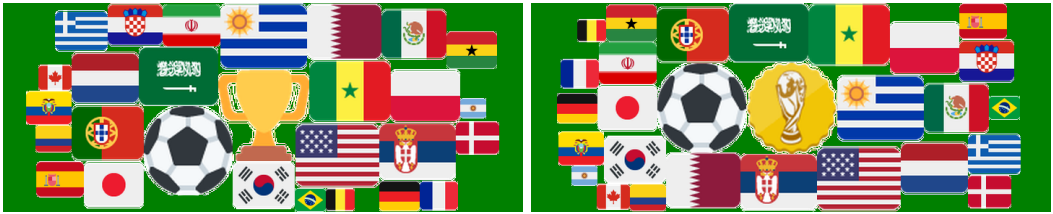
Figure 9: EmojiCloud for different vendors

Code for Customized Emojis

```python
from EmojiCloud import EmojiCloud

# set emoji weights by a dict with key: emoji by codepoint, value: weight
dict_weight = {'1F1E6-1F1F7': 1.1, '1F1E7-1F1EA': 1.2, '1F1E7-1F1F7': 1.3, '1F1E8-1F1E6': 1.4,
'1F1E8-1F1F4': 1.5, '1F1E8-1F1F5': 1.6, '1F1E9-1F1EA': 1.7, '1F1E9-1F1F0': 1.8, '1F1EA-1F1E8': 1.9,
'1F1EA-1F1F8': 2.0, '1F1EC-1F1ED': 2.1, '1F1EC-1F1F7': 2.2, '1F1ED-1F1F7': 2.3, '1F1EE-1F1F7': 2.4,
'1F1EF-1F1F5': 2.5, '1F1F0-1F1F7': 2.6, '1F1F2-1F1FD': 2.7, '1F1F3-1F1F1': 2.8, '1F1F5-1F1F1': 2.9,
'1F1F5-1F1F9': 3.0, '1F1F6-1F1E6': 3.1, '1F1F7-1F1F8': 3.2, '1F1F8-1F1E6': 3.3, '1F1F8-1F1F3': 3.4,
'1F1FA-1F1F8': 3.5, '1F1FA-1F1FE': 3.6, '26BD': 3.7, '1F3C6': 3.8}
dict_customized = {'1F3C6':'./trophy_emoji.png'}

# emoji vendor
emoji_vendor = 'Twitter'

# rectangle canvas
canvas_w = 72*10
canvas_h = 72*4
canvas_color = 'green'
saved_emoji_cloud_name = 'emoji_cloud_customized.png'
EmojiCloud.plot_rectangle_canvas(canvas_w, canvas_h, emoji_vendor, dict_weight, saved_emoji_cloud_name,
dict_customized, canvas_color)
```

Figure 10: EmojiCloud for plotting customized emojis

Visualization of Customized Emojis



(a) Original (b) Customized

Figure 11: EmojiCloud for FIFA World Cup Trophy

Using Emoji Unicode as Input

```
from EmojiCloud import EmojiCloud

# set emoji weights by a dict with key: emoji by unicode, value: weight
dict_weight = {'AC': 1.1, '💧': 1.2, '😂': 1.3, '🦵': 1.4, '🏊': 1.5, '🗡': 1.6,
'🚛': 1.7, '🍲': 1.8, '💉': 1.9, '🥼': 2.0, '😜': 2.1, '🍉': 2.2, '🟠': 2.3, '🦢':
2.4, '🦄': 2.5, '🦃': 2.6, '🐌': 2.7, '🦡': 2.8, '🦈': 2.9, '🦌': 3.0}

# emoji vendor
emoji_vendor = 'Google'

# circle canvas
canvas_w = 72*5
canvas_h = 72*5
saved_emoji_cloud_name = 'emoji_cloud_circle.png'
EmojiCloud.plot_ellipse_canvas(canvas_w, canvas_h, emoji_vendor, dict_weight,
saved_emoji_cloud_name)
```

Figure 12: EmojiCloud for plotting customized emojis

Running Time Evaluation

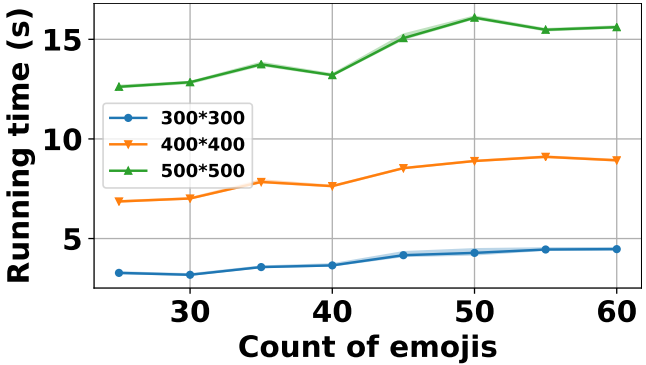


Figure 13: Running time of EmojiCloud

Future Work

- Keep updating the open-source EmojiCloud based on the users' feedback, such as adding new functions and covering more emoji vendors.
- Provide an online EmojiCloud service via `www.emojicloud.org`.
- Explore the possibility of merging words and emojis in a unified word-emoji cloud.

Q & A
Thank you!