EmojiCloud: a Tool for Emoji Cloud Visualization

Yunhe Feng¹, Cheng Guo¹, Bingbing Wen¹, Peng Sun², Yufei Yue³, Dingwen Tao⁴

University of Washington¹
The Chinese University of Hong Kong, Shenzhen²
Amazon³
Washington State University⁴

14th July, 2022
1 Motivation

2 EmojiCloud Design & Implementation

3 EmojiCloud Evaluation

4 Future Work
Representation Problems of Word Cloud of Emojis

• colors (⚽️❤️)
• directionalities (😭😢)
• textures (🙌❤️)

Figure 1: Word cloud of emojis
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.
Motivation

EmojiCloud Design & Implementation

EmojiCloud Evaluation

Future Work
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

### Figure 3: Unicode full emoji list

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

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 $(x, y)$.

$$s \geq \sum_{i=1}^{\lvert e \rvert} w_i^2 \cdot a_i \cdot b_i \cdot r^2 = \sum_{i=1}^{\lvert e \rvert} w_i^2 \cdot c \cdot r^2 \quad (1)$$

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 $\sqrt{s/(c \cdot \sum_{i=1}^{\lvert e \rvert} w_i^2)}$.
- the rescaled width and height: $a'_i = a_i \cdot w_i \cdot r; \quad b'_i = b_i \cdot w_i \cdot r$.
- 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 \(m \times n\) 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.
Motivation

EmojiCloud Design & Implementation

EmojiCloud Evaluation

Future Work

EmojiCloud Layout

Algorithm 1: EmojiCloud Layout

Input: e: a list of emojis; (m, n, c, V): a canvas with width m, height n, drawable size c, pixel values C, and pixel painting eligibility V on the canvas; w: the standardized size of emoji images;

Output: C: an emoji cloud image;

1. n - sort the emoji list by emoji weights \( w = [w_1, w_2, ..., w_n] \) in reverse order;
2. for \( x = 1 \) to \( m \) do
   // x coordinate of canvas
   for \( y = 1 \) to \( n \) 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 \( p_c \)
   // canvas pixel is eligible for painting
   count ← 0;
   // count of plotted emojis
   while count \(!=\) \# of emojis do
     // not all emojis have been plotted
     count ← 0;
     // count of plotted emojis
     for \( i = 1 \) to \# of emojis do
       // iterate the emojis sorted by weights in reverse order
       \( w_i \) - a weight of e\(_i\);
       // parse \( w_i \) into its quintuple representation
       \( e' \) - update \( E \) based on \( w_i \);
       // rescale \( E \) based on \( r \);
       \( \left[ x, y, E', E'' \right] \)
       for \((x, y)\) ∈ \( p_c \) do
         // (x, y) is where the center of \( e \) to be located
         flag ← True;
         // indicate the possibility of plotting \( e \)
         \( p_t \) ← \( \left[ x, y \right] \);
         // a list of canvas temporal coordinates to plot \( e \)
         for \( x' = 1 \) to \( x \) do
           // x coordinate of emoji image
           \( x_t \) ← \( x' \);
           for \( y' = 1 \) to \( y \) do
             // y coordinate of emoji image
             \( y_t \) ← \( y' \);
             if \( U_{x_t, y_t} = 0 \) then
               // emoji pixel \((x', y')\) is not unoccupied
               \( x_t \leftarrow x'_t = x + w, y_t \leftarrow y' = E'/2 \);
               // the offsets to \( e \) center
               \( x_t \leftarrow x + w; y_t \leftarrow y + p_t \);
               // canvas temporal coordinate for \( e \)
               append \((x_t, y_t)\) to \( p_t \);
             // canvas pixel is not eligible for painting
             if \( V_{x_t, y_t} = 0 \) then
               // no room to plot \( e \) at \((x_t, y_t)\)
               flag ← False;
             // iterate the next \((x_t, y_t)\)
           // no room to plot \( e \) at \((x_t, y_t)\)
           break;
         // the emoji \( e \) can be plot at \((x_t, y_t)\)
         if flag = True then
           for \((x_t, y_t)\) ∈ \( p_t \) do
             // iterate temporal pixel coordinates
             \( C_{x_t, y_t} \leftarrow C_{x_t, y_t} + 1 \);
             // plot emoji at \((x_t, y_t)\)
           // set painting eligibility as negative
           \( V_{x_t, y_t} \leftarrow 0 \);
           remove \((x_t, y_t)\) from \( p_t \);
         // delete \((x_t, y_t)\) for computing efficiency
         count ← count + 1;
         // increase the number of plotted emoji by 1
         break;
       // the number of plotted emoji by 1
     // the number of plotted emoji by 1
   // the number of plotted emoji by 1
   return \( C \);
   // decay the edge rescale ratio by 0.9
Yunhe Feng, Cheng Guo, Bingbing Wen, Peng Sun, Yufei Yue, Dingwen Tao

EmojiCloud: a Tool for Emoji Cloud Visualization
Default and Arbitrary Canvas

- We set the default canvas shape as an $m \times n$ 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 $m \times n$ 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.

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

(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.
• We determine a $m \times n$ bounding box of the image by removing the surrounding transparent pixels.

• We detect the image contour and draw a boundary accordingly (e.g., ⤢ → ⤣).
  - We scan the alpha values of pixels in the preprocessed image by row and by column respectively.
  - We identify pixels that cause an alpha value change greater than a threshold $\theta$ (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/](https://pypi.org/project/EmojiCloud/).
1 Motivation

2 EmojiCloud Design & Implementation

3 EmojiCloud Evaluation

4 Future Work
Figure 6: EmojiCloud for plotting different canvases

```
from EmojiCloud import EmojiCloud

# set emoji weights by a dict with key: emoji by codepoint, value: weight
dict_weight = \{"1f1e8-1f1e8": 1.1, "1f4a7": 1.2, "1f702": 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, "1f708": 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)
savedEmojiCloud_name = 'emoji_cloud_masked.png'
EmojiCloud.plot_masked_canvas(img_mask, thold_alpha_contour, contour_width, contour_color, emoji_vendor, dict_weight, savedEmojiCloud_name)

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

# ellipse canvas
canvas_w = 72*10
canvas_h = 72*4
savedEmojiCloud_name = 'emoji_cloud_ellipse.png'
EmojiCloud.plot_ellipse_canvas(canvas_w, canvas_h, emoji_vendor, dict_weight, savedEmojiCloud_name)
```
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+1F611': 2.2, 'U+1F612': 2.3, 'U+1F614': 2.4, 'U+1F615': 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+1F623': 3.2, 'U+1F625': 3.3, 'U+1F627': 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+1F928': 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

Figure 11: EmojiCloud for FIFA World Cup Trophy
Using Emoji Unicode as Input

```python
from EmojiCloud import EmojiCloud

# set emoji weights by a dict with key: emoji by unicode, value: weight
dict_weight = {'أشخاص': 1.1, ' абв ': 1.2, 'балон': 1.3, 'المستقبل': 1.4, 'الاسم': 1.5, 'الملحق': 1.6, 'العروس': 1.7, 'الماء': 1.8, 'الماء': 1.9, 'الماء': 2.0, 'الماء': 2.1, 'الماء': 2.2, 'الماء': 2.3, 'الماء': 2.4, 'الماء': 2.5, 'الماء': 2.6, 'الماء': 2.7, 'الماء': 2.8, 'الماء': 2.9, 'الماء': 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

![Graph showing running time of EmojiCloud](image)

Figure 13: Running time of EmojiCloud
1 Motivation

2 EmojiCloud Design & Implementation

3 EmojiCloud Evaluation

4 Future Work
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!