/slack-gif-creator

Source: `~/.claude/skills/slack-gif-creator/SKILL.md`

---

name: slack-gif-creator description: Knowledge and utilities for creating animated GIFs optimized for Slack. Provides constraints, validation tools, and animation concepts. Use when users request animated GIFs for Slack like "make me a GIF of X doing Y for Slack." license: Complete terms in LICENSE.txt

Slack GIF Creator

A toolkit providing utilities and knowledge for creating animated GIFs optimized for Slack.

Slack Requirements

Dimensions:

- Emoji GIFs: 128x128 (recommended)
- Message GIFs: 480x480

Parameters:

- FPS: 10-30 (lower is smaller file size)
- Colors: 48-128 (fewer = smaller file size)
- Duration: Keep under 3 seconds for emoji GIFs

Core Workflow

```
from core.gif_builder import GIFBuilder
from PIL import Image, ImageDraw

# 1. Create builder
builder = GIFBuilder(width=128, height=128, fps=10)

# 2. Generate frames
for i in range(12):
    frame = Image.new('RGB', (128, 128), (240, 248, 255))
    draw = ImageDraw.Draw(frame)

    # Draw your animation using PIL primitives
    # (circles, polygons, lines, etc.)

    builder.add_frame(frame)

# 3. Save with optimization
builder.save('output.gif', num_colors=48, optimize_for_emoji=True)
```

Drawing Graphics

Working with User-Uploaded Images

If a user uploads an image, consider whether they want to:

- **Use it directly** (e.g., "animate this", "split this into frames")
- **Use it as inspiration** (e.g., "make something like this")

Load and work with images using PIL:

```
from PIL import Image

uploaded = Image.open('file.png')
# Use directly, or just as reference for colors/style
```

Drawing from Scratch

When drawing graphics from scratch, use PIL ImageDraw primitives:

```
from PIL import ImageDraw

draw = ImageDraw.Draw(frame)

# Circles/ovals
draw.ellipse([x1, y1, x2, y2], fill=(r, g, b), outline=(r, g, b), width=3)

# Stars, triangles, any polygon
points = [(x1, y1), (x2, y2), (x3, y3), ...]
draw.polygon(points, fill=(r, g, b), outline=(r, g, b), width=3)

# Lines
draw.line([(x1, y1), (x2, y2)], fill=(r, g, b), width=5)

# Rectangles
draw.rectangle([x1, y1, x2, y2], fill=(r, g, b), outline=(r, g, b), width=3)
```

Don't use: Emoji fonts (unreliable across platforms) or assume pre-packaged graphics exist in this skill.

Making Graphics Look Good

Graphics should look polished and creative, not basic. Here's how:

Use thicker lines - Always set `width=2` or higher for outlines and lines. Thin lines (width=1) look choppy and amateurish.

Add visual depth:

- Use gradients for backgrounds (`create_gradient_background`)

- Layer multiple shapes for complexity (e.g., a star with a smaller star inside)

Make shapes more interesting:

- Don't just draw a plain circle - add highlights, rings, or patterns
- Stars can have glows (draw larger, semi-transparent versions behind)
- Combine multiple shapes (stars + sparkles, circles + rings)

Pay attention to colors:

- Use vibrant, complementary colors
- Add contrast (dark outlines on light shapes, light outlines on dark shapes)
- Consider the overall composition

For complex shapes (hearts, snowflakes, etc.):

- Use combinations of polygons and ellipses
- Calculate points carefully for symmetry
- Add details (a heart can have a highlight curve, snowflakes have intricate branches)

Be creative and detailed! A good Slack GIF should look polished, not like placeholder graphics.

Available Utilities

GIFBuilder (`core.gif_builder`)

Assembles frames and optimizes for Slack:

```
builder = GIFBuilder(width=128, height=128, fps=10)
builder.add_frame(frame)  # Add PIL Image
builder.add_frames(frames)  # Add list of frames
builder.save('out.gif', num_colors=48, optimize_for_emoji=True, remove_duplicates=True)
```

Validators (`core.validators`)

Check if GIF meets Slack requirements:

```
from core.validators import validate_gif, is_slack_ready

# Detailed validation
passes, info = validate_gif('my.gif', is_emoji=True, verbose=True)
```

```
# Quick check
if is_slack_ready('my.gif'):
    print("Ready!")
```

Easing Functions (`core.easing`)

Smooth motion instead of linear:

```
from core.easing import interpolate

# Progress from 0.0 to 1.0
t = i / (num_frames - 1)

# Apply easing
y = interpolate(start=0, end=400, t=t, easing='ease_out')

# Available: linear, ease_in, ease_out, ease_in_out,
#           bounce_out, elastic_out, back_out
```

Frame Helpers (`core.frame_composer`)

Convenience functions for common needs:

```
from core.frame_composer import (
    create_blank_frame,         # Solid color background
    create_gradient_background,  # Vertical gradient
    draw_circle,                # Helper for circles
    draw_text,                  # Simple text rendering
    draw_star                   # 5-pointed star
)
```

Animation Concepts

Shake/Vibrate

Offset object position with oscillation:

- Use `math.sin()` or `math.cos()` with frame index
- Add small random variations for natural feel
- Apply to x and/or y position

Pulse/Heartbeat

Scale object size rhythmically:

- Use `math.sin(t * frequency * 2 * math.pi)` for smooth pulse
- For heartbeat: two quick pulses then pause (adjust sine wave)
- Scale between 0.8 and 1.2 of base size

Bounce

Object falls and bounces:

- Use `interpolate()` with `easing='bounce_out'` for landing
- Use `easing='ease_in'` for falling (accelerating)
- Apply gravity by increasing y velocity each frame

Spin/Rotate

Rotate object around center:

- PIL: `image.rotate(angle, resample=Image.BICUBIC)`
- For wobble: use sine wave for angle instead of linear

Fade In/Out

Gradually appear or disappear:

- Create RGBA image, adjust alpha channel
- Or use `Image.blend(image1, image2, alpha)`
- Fade in: alpha from 0 to 1
- Fade out: alpha from 1 to 0

Slide

Move object from off-screen to position:

- Start position: outside frame bounds
- End position: target location
- Use `interpolate()` with `easing='ease_out'` for smooth stop

- For overshoot: use `easing='back_out'`

Zoom

Scale and position for zoom effect:

- Zoom in: scale from 0.1 to 2.0, crop center
- Zoom out: scale from 2.0 to 1.0
- Can add motion blur for drama (PIL filter)

Explode/Particle Burst

Create particles radiating outward:

- Generate particles with random angles and velocities
- Update each particle: `x += vx`, `y += vy`
- Add gravity: `vy += gravity_constant`
- Fade out particles over time (reduce alpha)

Optimization Strategies

Only when asked to make the file size smaller, implement a few of the following methods:

1. **Fewer frames** - Lower FPS (10 instead of 20) or shorter duration
2. **Fewer colors** - `num_colors=48` instead of 128
3. **Smaller dimensions** - 128x128 instead of 480x480
4. **Remove duplicates** - `remove_duplicates=True` in save()
5. **Emoji mode** - `optimize_for_emoji=True` auto-optimizes

```
# Maximum optimization for emoji
builder.save(
    'emoji.gif',
    num_colors=48,
    optimize_for_emoji=True,
    remove_duplicates=True
)
```

Philosophy

This skill provides:

- **Knowledge**: Slack's requirements and animation concepts
- **Utilities**: GIFBuilder, validators, easing functions
- **Flexibility**: Create the animation logic using PIL primitives

It does NOT provide:

- Rigid animation templates or pre-made functions
- Emoji font rendering (unreliable across platforms)
- A library of pre-packaged graphics built into the skill

Note on user uploads: This skill doesn't include pre-built graphics, but if a user uploads an image, use PIL to load and work with it - interpret based on their request whether they want it used directly or just as inspiration.

Be creative! Combine concepts (bouncing + rotating, pulsing + sliding, etc.) and use PIL's full capabilities.

Dependencies

```
pip install pillow imageio numpy
```

Revision #4
Created 2026-02-18 08:42:05 UTC by John
Updated 2026-05-31 20:01:55 UTC by John