Drawing API

Complete drawing functions reference

Overview

All drawing functions must be called within the `draw()` event. Drawing happens after all `step()` events complete.

Sprite Drawing

`draw_self()`

Draws the instance's current sprite with all transformations.

Syntax: `draw_self.call(this)`

Arguments: None

Returns: `void`

Applies:

- Position (`x` , `y`)
- Sprite (`sprite_index` , `image_index`)
- Scale (`image_xscale` , `image_yscale`)
- Rotation (`image_angle`)
- Alpha (`image_alpha`)

Example:

```
draw(): void {
  draw_self.call(this);
}
```

Note: If `draw()` is not defined, the engine automatically calls `draw_self()`.

`draw_sprite()`

Draws a specific sprite at a position.

Syntax: `draw_sprite(sprite, subimg, x, y)`

Arguments:

- `sprite` (string) - Sprite name
- `subimg` (number) - Frame index
- `x` (number) - X position
- `y` (number) - Y position

Returns: `void`

Example:

```
draw(): void {
  // Draw health icons
  for (let i = 0; i < this.lives; i++) {
    draw_sprite('spr_heart', 0, 10 + (i * 32), 10);
  }

  // Draw different sprite
  draw_sprite('spr_shadow', 0, this.x, this.y + 10);

  // Draw sprite at specific frame
  draw_sprite('spr_explosion', 3, this.x, this.y);
}
```

Note: Does not apply instance transformations (scale, angle, alpha). Use manually if needed.

`draw_sprite_ext()`

Draws sprite with transformations (extended).

Syntax: `draw_sprite_ext(sprite, subimg, x, y, xscale, yscale, angle, color, alpha)`

Arguments:

- `sprite` (string) - Sprite name
- `subimg` (number) - Frame index
- `x`, `y` (number) - Position
- `xscale`, `yscale` (number) - Scale (1.0 = normal)
- `angle` (number) - Rotation in degrees
- `color` (string) - Tint color (use '#FFFFFF' for none)
- `alpha` (number) - Opacity (0.0-1.0)

Returns: `void`

Example:

```
draw(): void {
  // Draw scaled and rotated
  draw_sprite_ext(
    'spr_bullet',
    0,
    this.x, this.y,
    2.0, 2.0,      // Double size
    45,            // 45 degrees
    '#FFFFFF',     // No tint
    0.5            // 50% transparent
  );
}
```

Text Drawing

`draw_text()`

Draws text at a position.

Syntax: `draw_text(x, y, text)`

Arguments:

- `x`, `y` (number) - Position (top-left of text)
- `text` (string | number) - Text to draw

Returns: `void`

Example:

```
draw(): void {
  // Simple text
  draw_text(10, 10, 'Hello World');

  // Dynamic text
  draw_text(10, 30, `Score: ${this.score}`);
  draw_text(10, 50, `Health: ${this.health}/${this.maxHealth}`);

  // Numbers
  draw_text(10, 70, this.coins);
}
```

Font: Uses browser default. Customize via canvas context:

```
draw(): void {
  const ctx = (globalThis as any).gameContext;
  ctx.font = '20px Arial';
  draw_text(10, 10, 'Custom Font');
}
```

Shape Drawing

`draw_rectangle()`

Draws a rectangle.

Syntax: `draw_rectangle(x1, y1, x2, y2, outline)`

Arguments:

- `x1`, `y1` (number) - Top-left corner
- `x2`, `y2` (number) - Bottom-right corner

- `outline` (boolean) - Draw outline only? (false = filled)

Returns: `void`

Example:

```
draw(): void {
  // Filled rectangle
  draw_set_color('#FF0000');
  draw_rectangle(10, 10, 100, 50, false);

  // Outline rectangle
  draw_set_color('#00FF00');
  draw_rectangle(110, 10, 200, 50, true);

  // Health bar
  draw_set_color('#FF0000');
  draw_rectangle(this.x - 25, this.y - 35, this.x + 25, this.y - 30, false);

  const healthPercent = this.health / this.maxHealth;
  draw_set_color('#00FF00');
  draw_rectangle(
    this.x - 25,
    this.y - 35,
    this.x - 25 + (50 * healthPercent),
    this.y - 30,
    false
  );
}
```

`draw_circle()`

Draws a circle.

Syntax: `draw_circle(x, y, radius, outline)`

Arguments:

- `x`, `y` (number) - Center position
- `radius` (number) - Circle radius in pixels

- `outline` (boolean) - Draw outline only?

Returns: `void`

Example:

```
draw(): void {
  // Filled circle
  draw_set_color('#0000FF');
  draw_circle(this.x, this.y, 20, false);

  // Outline circle
  draw_set_color('#FFFF00');
  draw_circle(this.x, this.y, 50, true);

  // Detection radius visualization
  draw_set_alpha(0.2);
  draw_circle(this.x, this.y, 100, false);
  draw_set_alpha(1.0);
}
```

`draw_line()`

Draws a line between two points.

Syntax: `draw_line(x1, y1, x2, y2)`

Arguments:

- `x1`, `y1` (number) - Start point
- `x2`, `y2` (number) - End point

Returns: `void`

Example:

```
draw(): void {
  // Line to target
  draw_set_color('#FF0000');
  draw_line(this.x, this.y, this.targetX, this.targetY);

  // Crosshair
  draw_set_color('#FFFFFF');
  draw_line(mouse_x - 10, mouse_y, mouse_x + 10, mouse_y);
  draw_line(mouse_x, mouse_y - 10, mouse_x, mouse_y + 10);
}
```

`draw_point()`

Draws a single pixel.

Syntax: `draw_point(x, y)`

Arguments:

- `x`, `y` (number) - Position

Returns: `void`

Example:

```
draw(): void {
  // Trail effect
  for (const pos of this.trail) {
    draw_set_color('#FFFF00');
    draw_point(pos.x, pos.y);
  }
}
```

Color and Alpha

`draw_set_color()`

Sets the drawing color for shapes and text.

Syntax: `draw_set_color(color)`

Arguments:

- `color` (string) - Hex color code (e.g., "#FF0000")

Returns: `void`

Example:

```
draw(): void {
  draw_set_color('#FF0000'); // Red
  draw_rectangle(10, 10, 50, 50, false);

  draw_set_color('#00FF00'); // Green
  draw_rectangle(60, 10, 100, 50, false);

  draw_set_color('#0000FF'); // Blue
  draw_rectangle(110, 10, 150, 50, false);

  draw_set_color('#FFFFFF'); // Reset to white
}
```

Common colors:

- `'#FFFFFF'` - White
- `'#000000'` - Black
- `'#FF0000'` - Red
- `'#00FF00'` - Green
- `'#0000FF'` - Blue
- `'#FFFF00'` - Yellow
- `'#FF00FF'` - Magenta
- `'#00FFFF'` - Cyan
- `'#808080'` - Gray

RGB Format: `'#RRGGBB'` where each component is 00-FF hex

`draw_set_alpha()`

Sets the drawing transparency.

Syntax: `draw_set_alpha(alpha)`

Arguments:

- `alpha` (number) - Alpha value (0.0 = transparent, 1.0 = opaque)

Returns: `void`

Example:

```
draw(): void {
  // Semi-transparent overlay
  draw_set_alpha(0.5);
  draw_set_color('#000000');
  draw_rectangle(0, 0, room_width, room_height, false);
  draw_set_alpha(1.0); // Always reset!

  // Fading text
  draw_set_alpha(this.fadeAmount);
  draw_text(100, 100, 'Fading...');
  draw_set_alpha(1.0);

  // Ghost effect
  draw_set_alpha(0.3);
  draw_self.call(this);
  draw_set_alpha(1.0);
}
```

Important: Always reset alpha to 1.0 after use!

Drawing State

Resetting State

Always reset drawing state after changing it:

```
draw(): void {
  // GOOD: Reset after use
  draw_set_color('#FF0000');
  draw_rectangle(10, 10, 50, 50, false);
  draw_set_color('#FFFFFF'); // Reset

  draw_set_alpha(0.5);
  draw_circle(100, 100, 20, false);
  draw_set_alpha(1.0); // Reset

  // BAD: Forget to reset
  draw_set_color('#FF0000');
  draw_rectangle(10, 10, 50, 50, false);
  // Everything after this will be red!
}
```

Advanced Drawing

Layered Drawing

Use depth to control draw order:

```
// obj_background
create(): void {
  this.depth = 100; // Behind everything
}

// obj_player
create(): void {
  this.depth = 0; // Middle
}

// obj_ui
create(): void {
  this.depth = -100; // In front of everything
}
```

Custom Drawing Order

Skip auto-draw and control order manually:

```
draw(): void {
  // Draw shadow first (behind)
  draw_sprite('spr_shadow', 0, this.x, this.y + 5);

  // Draw sprite
  draw_self.call(this);

  // Draw effects on top
  if (this.powered) {
    draw_set_alpha(0.5);
    draw_set_color('#FFFF00');
    draw_circle(this.x, this.y, 25, false);
    draw_set_alpha(1.0);
  }
}
```

Screen-Space Drawing

Draw relative to view (for UI):

```
draw(): void {
  // World-space (scrolls with camera)
  draw_self.call(this);

  // Screen-space (fixed to view)
  const screenX = view_xview[0] + 10;
  const screenY = view_yview[0] + 10;
  draw_set_color('#FFFFFF');
  draw_text(screenX, screenY, `Score: ${(window as any).score}`);
}
```

Common Patterns

Health Bar Above Character

```
draw(): void {
  // Draw character
  draw_self.call(this);

  // Health bar
  const barWidth = 50;
  const barHeight = 5;
  const barX = this.x - barWidth / 2;
  const barY = this.y - 35;
  const healthPercent = this.health / this.maxHealth;

  // Background (dark gray)
  draw_set_color('#333333');
  draw_rectangle(barX, barY, barX + barWidth, barY + barHeight, false);

  // Health (red to green gradient)
  if (healthPercent > 0.5) {
    draw_set_color('#00FF00'); // Green
  } else if (healthPercent > 0.25) {
    draw_set_color('#FFFF00'); // Yellow
  } else {
    draw_set_color('#FF0000'); // Red
  }
  draw_rectangle(
    barX,
    barY,
    barX + (barWidth * healthPercent),
    barY + barHeight,
    false
  );

  // Border (white)
  draw_set_color('#FFFFFF');
  draw_rectangle(barX, barY, barX + barWidth, barY + barHeight, true);
}
```

Progress Bar

```
draw(): void {
  const x = 100;
  const y = 400;
  const width = 200;
  const height = 20;
  const progress = this.loadProgress; // 0.0 to 1.0

  // Background
  draw_set_color('#333333');
  draw_rectangle(x, y, x + width, y + height, false);

  // Progress
  draw_set_color('#00FF00');
  draw_rectangle(x, y, x + (width * progress), y + height, false);

  // Border
  draw_set_color('#FFFFFF');
  draw_rectangle(x, y, x + width, y + height, true);

  // Text
  const percent = Math.floor(progress * 100);
  draw_text(x + width / 2 - 10, y + height / 2 - 5, `${percent}%`);
}
```

Minimap

```
draw(): void {
  const mapX = view_xview[0] + view_wview[0] - 110;
  const mapY = view_yview[0] + 10;
  const mapSize = 100;
  const scale = mapSize / room_width;

  // Background
  draw_set_alpha(0.5);
  draw_set_color('#000000');
  draw_rectangle(mapX, mapY, mapX + mapSize, mapY + mapSize, false);
  draw_set_alpha(1.0);

  // Player dot
  const playerX = mapX + (this.x * scale);
  const playerY = mapY + (this.y * scale);
  draw_set_color('#00FF00');
  draw_circle(playerX, playerY, 2, false);

  // Enemy dots
  const enemyCount = instance_number('obj_enemy');
  for (let i = 0; i < enemyCount; i++) {
    const enemy = instance_find('obj_enemy', i);
    if (enemy) {
      const ex = mapX + (enemy.x * scale);
      const ey = mapY + (enemy.y * scale);
      draw_set_color('#FF0000');
      draw_circle(ex, ey, 2, false);
    }
  }

  // Border
  draw_set_color('#FFFFFF');
  draw_rectangle(mapX, mapY, mapX + mapSize, mapY + mapSize, true);
}
```

Performance Tips

1. **Minimize state changes**: Group draws by color/alpha
2. **Skip off-screen draws**: Check if visible before drawing
3. **Cache calculations**: Calculate positions in step(), use in draw()
4. **Batch similar draws**: Draw all of same type together

```
// GOOD: Batched by color
draw(): void {
  draw_set_color('#FF0000');
  draw_rectangle(10, 10, 50, 50, false);
  draw_rectangle(60, 10, 100, 50, false);

  draw_set_color('#00FF00');
  draw_rectangle(110, 10, 150, 50, false);
  draw_rectangle(160, 10, 200, 50, false);
}

// BAD: Too many state changes
draw(): void {
  draw_set_color('#FF0000');
  draw_rectangle(10, 10, 50, 50, false);
  draw_set_color('#00FF00');
  draw_rectangle(60, 10, 100, 50, false);
  draw_set_color('#FF0000'); // Redundant!
  draw_rectangle(110, 10, 150, 50, false);
}
```

Next Steps

- **09-drawing.md** - Drawing guide
- **20-api-gameobject.md** - GameObject API
- **40-common-patterns.md** - Drawing patterns

← Back to Index