AlgorithmicTypography — Cheat Sheet

SETUP & LIFECYCLE

Minimal sketch — auto-render mode

```
import algorithmic.typography.*;

AlgorithmicTypography at;

void setup() {
  size(800, 800);
  at = new AlgorithmicTypography(this);
  at.loadConfiguration("config.json");
  at.initialize();
}

void draw() {
  // auto-renders — nothing to add
}
```

Programmatic config (no JSON file)

```
Configuration cfg = new Configuration.Builder()
  .initialTiles(12, 12)
  .changedTiles(6, 6)
  .changeTime(5000)
  .character("@")
  .build();

at.setConfiguration(cfg);
at.initialize();
```

Restart & keyboard

```
void keyPressed() {
  if (key == 'r') at.restart();
  if (key == 's') at.toggleFrameSaving();
}
```

CORE API

Method	Returns	Description
loadConfiguration(file)	this	Load JSON config
setConfiguration(cfg)	this	Set Configuration object
getConfiguration()	Configuration	Get current config
initialize()	this	Build grid & start
render()	void	Draw full-window grid
renderAt(x,y,w,h)	void	Draw into rectangle
restart()	void	Reset animation timer
setAutoRender(bool)	this	Enable/disable auto draw
isAutoRender()	boolean	Check auto-render state
isRunning()	boolean	Animation still playing?
getProgress()	float	0.0 → 1.0 progress
getFrameCount()	int	Current frame index
getTotalFrames()	int	Total frames in anim
toggleFrameSaving()	this	Toggle frame export
setFrameSaving(bool)	this	Set frame export on/off
getFramesDirectory()	String	Output folder path
dispose()	void	Clean up resources

CONFIG.JSON SCHEMA

```
{
  "canvas": {
    "width": 1080,
    "height": 1080
  },
  "animation": {
    "duration": 18,
    "fps": 30,
    "changeTime": 6000,
    "secondChangeTime": 12000,
    "fadeDuration": 2000,
    "character": "A",
    "textScale": 0.8,
    "waveSpeed": 1.0,
    "waveAngle": 45,
    "waveMultiplierMin": 0.0,
    "waveMultiplierMax": 2.0,
    "saveFrames": false
  },
  "grid": {
    "initialTilesX": 16,
    "initialTilesY": 16,
    "changedTilesX": 8,
    "changedTilesY": 8,
    "finalTilesX": 4,
    "finalTilesY": 4
  },
  "colors": {
    "hueMin": 0,
    "hueMax": 0,
    "saturationMin": 0,
    "saturationMax": 0,
    "brightnessMin": 50,
    "brightnessMax": 255,
    "waveAmplitudeMin": -200,
    "waveAmplitudeMax": 200
  }
}
```

WAVE & STYLE

Wave presets

```
at.setWaveType(WavePresets.Type.SINE);
// SINE · SAWTOOTH · SQUARE · TRIANGLE
// BOUNCE · ELASTIC · NOISE · PULSE
```

Custom wave function (lambda)

```
at.setWaveFunction(
  (x, y, time, angle) -> {
    return (float)Math.sin(
      x * 0.1 + time
    );
  }
);
```

Design system presets

```
at.useDesignSystem(
  DesignSystem.SystemType.SWISS
);
// SWISS · BAUHAUS · JAPANESE
// BRUTALIST · MINIMAL
```

Vibe coding — natural language

```
at.setVibe("calm ocean waves");
at.setVibe("chaotic energy");
at.setVibe("geometric minimal");
```

GRID STAGES & TIMING

Three-stage grid progression

Stage	Grid Size	Trigger	Default
1 Initial	initialTilesX × Y	0 ms	16 × 16
2 Changed	changedTilesX × Y	changeTime	8 × 8 @ 6 s
3 Final	finalTilesX × Y	secondChangeTime	4 × 4 @ 12 s

`fadeDuration` Cross-fade duration between stages (default 2000 ms)

`secondChangeTime` Set to 0 to disable the third stage

Reading grid state at runtime

```
Configuration cfg = at.getConfiguration();

cfg.getInitialTilesX();   // stage 1 cols
cfg.getInitialTilesY();   // stage 1 rows
cfg.getChangedTilesX();   // stage 2 cols
cfg.getChangedTilesY();   // stage 2 rows
cfg.getFinalTilesX();     // stage 3 cols
cfg.getFinalTilesY();     // stage 3 rows

cfg.getChangeTime();      // ms → stage 2
cfg.getSecondChangeTime();// ms → stage 3
cfg.getFadeDuration();    // crossfade ms
```

Determining current stage manually

```
int elapsed = millis() - startTime;
int cols, rows;

if (elapsed > cfg.getSecondChangeTime()) {
  cols = cfg.getFinalTilesX();
  rows = cfg.getFinalTilesY();
} else if (elapsed > cfg.getChangeTime()) {
  cols = cfg.getChangedTilesX();
  rows = cfg.getChangedTilesY();
} else {
  cols = cfg.getInitialTilesX();
  rows = cfg.getInitialTilesY();
}
```

CONFIGURATION GETTERS

Method	Type	Default
getCanvasWidth()	int	1080
getCanvasHeight()	int	1080
getAnimationDuration()	int	18
getAnimationFPS()	int	30
getChangeTime()	int	6000
getSecondChangeTime()	int	12000
getFadeDuration()	int	2000
getCharacter()	String	"A"
getTextScale()	float	0.8
getWaveSpeed()	float	1.0
getWaveAngle()	float	45.0
getWaveMultiplierMin()	float	0.0
getWaveMultiplierMax()	float	2.0
getInitialTilesX/Y()	int	16
getChangedTilesX/Y()	int	8
getFinalTilesX/Y()	int	4
getHueMin/Max()	float	0 / 0
getSaturationMin/Max()	float	0 / 0
getBrightnessMin/Max()	float	50 / 255
getWaveAmplitudeMin/Max()	float	−200 / 200
isSaveFrames()	boolean	false

RENDER MODES

Auto Default — library clears & draws each frame

```
at.initialize();
// draw() is empty — auto-render handles it
```

Manual You control what goes on the canvas

```
at.setAutoRender(false);
at.initialize();

void draw() {
  background(0);
  // … your drawing code …
  at.renderAt(0, 0, width, height);
}
```

Multi Several instances side by side

```
AlgorithmicTypography a, b;
void setup() {
  size(1600, 800);
  a = new AlgorithmicTypography(this);
  b = new AlgorithmicTypography(this);
  a.setAutoRender(false); b.setAutoRender(false);
  a.loadConfiguration("a.json"); b.loadConfiguration("b.json");
  a.initialize(); b.initialize();
}
void draw() {
  a.renderAt(0,0,width/2,height);
  b.renderAt(width/2,0,width/2,height);
}
```

EXPORT & SVG

Save frames as PNG sequence

```
// Enable in config.json
"saveFrames": true

// Or toggle at runtime
at.toggleFrameSaving();
at.setFrameSaving(true);

// Check output directory
println(at.getFramesDirectory());
```

Save vector SVG

```
import processing.svg.*;

void keyPressed() {
  if (key == 's') {
    beginRecord(SVG, "output.svg");
    at.render();
    endRecord();
    println("SVG saved!");
  }
}
```

Common keyboard shortcuts

Key	Action
R	Restart animation
S	Toggle frame saving / save SVG

BACKGROUND IMAGE OVERLAY

Render typography on top of a photograph — place the image in the sketch's `data/` folder.

```
AlgorithmicTypography at;
PImage bg;

void setup() {
  size(800, 800);
  bg = loadImage("background.png");
  bg.resize(width, height);

  at = new AlgorithmicTypography(this);
  at.loadConfiguration("config.json");
  at.setAutoRender(false);
  at.initialize();
}

void draw() {
  image(bg, 0, 0);          // draw photo first
  at.renderAt(0, 0, width, height); // overlay grid
}
```

PER-CELL PHOTOGRAPHS (MULTIPHOTO)

Different photo behind each grid cell. Place photo-01.png … photo-16.png in `data/`

```
// setup(): load photos into PImage[] photos
for (int i = 1; i <= 16; i++) {
  PImage img = loadImage("photo-"+nf(i,2)+".png");
  if (img != null && img.width > 0)
    photos[photoCount++] = img;
}
```

```
// draw(): tile photos, then overlay glyphs
float cw = (float)width/cols, ch = (float)height/rows;
for (int gx=0; gx<cols; gx++)
 for (int gy=0; gy<rows; gy++) {
   int idx = (gy*cols+gx) % photoCount;
   image(photos[idx], gx*cw, gy*ch, cw, ch);
 }
at.renderAt(0, 0, width, height);
```

See `MultiPhoto` example for centre-crop & 3-stage tracking.

EXAMPLES GALLERY — 19 SKETCHES

AudioReactive
Audio input drives wave parameters in real time.

BackgroundImage
Typography overlay on a single background photograph.

BasicGrid
Minimal grid with JSON config — the "hello world" sketch.

CulturalStyles
Switch between built-in design systems (Swiss, Bauhaus …).

CustomFont
Load and render a custom .ttf / .otf font.

CustomWave
Provide a lambda for a custom wave function.

GlyphPath
Access individual glyph outlines as PShape paths.

GlyphPhysicsExample
Apply physics simulation to glyph shapes.

LiveControls
Adjust parameters live via WebSocket UI.

MultiPhoto
Different photograph behind each grid cell with stage tracking.

MultipleSystems
Run several AlgorithmicTypography instances side by side.

PerformanceMode
Optimised rendering path for high-resolution grids.

ProgrammaticConfig
Build Configuration in code with the Builder API.

RandomASCII
Random ASCII characters in each cell.

SaveSVG
Export the current frame as a vector SVG file.

TextDrivenAnimation
Typography driven by external text file content.

TrailEffect
Motion trail / ghosting effect on the grid.

VibeCoding
Natural-language vibe strings control the aesthetic.

WaveAngle
Explore wave propagation angle with JSON config.

PACKAGE STRUCTURE

Package	Purpose
algorithmic.typography	Core — AlgorithmicTypography, Configuration, HeadlessRenderer
…typography.core	Grid engine, tile management
…typography.audio	Audio-reactive input
…typography.ml	ML-based vibe interpretation
…typography.net	WebSocket server for live controls
…typography.render	Renderers (screen, SVG, headless)
…typography.style	Design systems & cultural presets
…typography.system	Vibe presets & system utilities
…typography.text	Text / glyph processing
…typography.ui	UI components for live controls

QUICK TIPS

#	Tip
1	Always call initialize() after loading or setting a configuration.
2	Use setAutoRender(false) whenever you draw backgrounds or composite.
3	renderAt(x, y, w, h) places the grid anywhere — split layouts, insets, etc.
4	Track your own startTime = millis() — the library's start time is private.
5	Call restart() + reset startTime together when restarting.
6	Set secondChangeTime to 0 in JSON to use only two grid stages.
7	Colours are HSB: hue 0–360, saturation 0–255, brightness 0–255.
8	Place images and JSON files in the sketch's data/ folder.
9	Builder pattern chains: new Configuration.Builder().initialTiles(8,8).build()
10	Javadoc is in reference/ — open index.html in a browser.