PyGraph Framework Documentation

This document provide, detailed documentation for the **PyGraph framework**, a custom 2D/3D graphics and multimedia application built upon a high-performance stack including **OpenCV**, **NumPy**, and **Pygame**.

1. Core Modules and Initialization

The framework combines multiple libraries for specialized tasks: **NumPy** for array-based data/images, **OpenCV (cv2)** for graphics and windowing, **Cython** for function acceleration, **Pygame (pygame.mixer)** for audio, and **Pynput** for non-blocking input handling.

Initialization and Global State Variables

Global Variable	Type	Description	Example/Usage
GLOBAL_VOLUME	List[float]	Master volume level, constantly monitored by a background thread.	pg.GLOBAL_VOLUME[0] = 0.5 (Sets volume to 50%)
KEYS	Dict[str, bool]	Tracks the state of currently held keyboard keys.	if pg.KEYS.get('w'): # 'w' is held
RUN	bool	Flag controlling the main application loop. Set to False to exit the program.	Set by default; set to False in on_release for Esc.

MOUSE_SCR_POS	Tuple[int, int]	Current mouse cursor position in **screen coordinates** (pixels from top-left).	(400, 300) for the center of an 800x600 window.
MOUSE_STATE	Dict	Detailed mouse input state.	pg.MOUSE_STATE['buttons'][0] checks if the Left Mouse Button (LMB) is held.
TS_STACK	List[Dict]	Stack for saving and restoring transformation states.	Used with pg.push_state(ts) and pg.pop_state(ts).
C	Dict[str, Tuple]	Pre-defined common RGB colors (see section 11).	pg.C['R'] is Red (255, 0, 0).

2. Asset Management (AssetManager)

The ASSETS global instance of AssetManager handles loading and caching of resources, making them available by a given name. Images are stored as NumPy arrays, and sounds as Pygame Sound objects.

Method	Parameters	Description	Example/Usage

load_image	name: str, path: str	Loads an image using cv2.imread(cv2.IMREAD_UNCHANGED) to preserve alpha channels.	pg.ASSETS.load_image('ship', 'assets/ship.png')
load_sound	name: str, path: str	Loads a sound using pygame.mixer.Sound.	pg.ASSETS.load_sound('shot', 'assets/laser.wav')
get_img	name: str	Retrieves a cached image (np.ndarray). Returns None if not found.	img = pg.ASSETS.get_img('ship')
get_snd	name: str	Retrieves a cached sound object. Returns None if not found.	snd = pg.ASSETS.get_snd('shot')

---

3. Transformation State Management (ts Dict)

The ts dictionary holds the state of the coordinate system, which is automatically applied to all drawing commands.

State Keys

Key	**Type**	**Description**	**Explanation**
rot	float	2D rotation angle in degrees.	The entire canvas rotates around the screen center/shift point.

size	float	Global scale factor (zoom).	Controls the zoom level; all dimensions are multiplied by this value.
shift	Tuple[float]	2D/3D translation offset.	Defines where the world origin (0, 0) maps to on the screen (in pixels).
cursor_pos	Tuple[float]	Current world drawing position.	Primitives start drawing here. Updated by functions like line() and move().
full_transform	np.ndarray	3x3 Homography matrix.	Used for non-linear 2D warping (e.g., projecting onto a trapezoid).
cam_pos_3d	Tuple[float, float, float]	3D camera world position (x, y, z).	Defines the viewer's location in 3D space.
cam_rot_3d	Tuple[float, float, float]	3D camera rotation (pitch, yaw, roll).	Defines the camera's orientation (view direction).
is_3d	bool	Toggles rendering between 2D (planar) and 3D (perspective) modes.	Enables the transform_3d pipeline.

Manipulation Functions

These functions update the ts dictionary.

Function	Parameters	Action	Example/Usage
push_state	ts: Dict	Saves a copy of the current state keys onto TS_STACK.	pg.push_state(ts) (Saves all transformations)
pop_state	ts: Dict	Restores the state from the top of TS_STACK.	pg.pop_state(ts) (Returns to previous state)
cycle_state	ts: Dict	Moves the top item of TS_STACK to the bottom.	pg.cycle_state(ts) (Cycles through saved states)
set_3d_mode	ts: Dict, is_3d: bool	Toggles 2D/3D mode.	pg.set_3d_mode(ts, True) (Enables 3D perspective)
move	ts: Dict, xy: Tuple	Moves ts['cursor_pos'] by a delta amount.	pg.move(ts, (10, 5)) (Moves cursor 10 right, 5 up/down)
set_pos	ts: Dict, xy: Tuple	Sets ts['cursor_pos'] to absolute coordinates.	pg.set_pos(ts, (50, -20)) (Sets cursor to world position 50, -20)

shift	ts: Dict, xy: Tuple	Updates the world-to-screen shift (pan the view) by a delta.	pg.shift(ts, (-1, 0)) (Pans the view one unit left)
set_shift	ts: Dict, xy: Tuple	Sets the world-to-screen shift to an absolute position.	pg.set_shift(ts, (400, 300)) (Centers the world origin in an 800x600 window)
scale	ts: Dict, ds: float	Increments/decrements ts['size'] (zoom) by a delta.	pg.scale(ts, 0.1) (Zooms in by 10%)
set_size	ts: Dict, ds: float	Sets ts['size'] (zoom) to an absolute value.	pg.set_size(ts, 2.5) (Sets zoom to 250%)
rotate	ts: Dict, dr: float	Increments/decrements ts['rot'] (view rotation) by a delta (degrees).	pg.rotate(ts, 5) (Rotates view 5 degrees)
set_rotate	ts: Dict, dr: float	Sets ts['rot'] (view rotation) to an absolute angle (degrees).	pg.set_rotate(ts, 45) (Sets view rotation to 45 degrees)
set_cam_pos	ts: Dict, p: Tuple	Sets 3D camera position.	pg.set_cam_pos(ts, (0, 0, -50)) (Moves camera back on the Z-axis)

set_cam_rot	ts: Dict, r: Tuple	Sets 3D camera rotation (pitch, yaw, roll).	pg.set_cam_rot(ts, (30, 0, 0)) (Tilts camera 30 degrees down)
set_homography	ts, src_pts, dst_pts	Calculates a 2D warping matrix (Homography) from 4 source points to 4 destination points.	pg.set_homography(ts, src, dst) (Applies complex warping to the view)

---

4. Core Transformations (Optimized)

Critical math functions are implemented in **Cython** for highly optimized performance.

Function	**Description**	**Example/Usage**
transform	Main **World $\rightarrow$ Screen** conversion. Selects 2D or 3D pipeline based on ts['is_3d'].	Used internally to convert all world coordinates to pixel positions.
inverse_transform	**Screen $\rightarrow$ World** conversion. Calculates the world coordinate of a given screen pixel.	Used by get_input("mouse", "pos") to find where the mouse is in world space.
transform_3d	The complete 3D pipeline: applies 2D rotation/scale, translates/rotates to camera	(screen_x, screen_y, depth) = pg.transform_3d((10, 5, 20), ts)

	space, and applies **perspective_projection**.	
perspective_projection	Converts 3D camera coordinates to 2D screen coordinates and returns depth.	Handles the visual "fading" and scaling required for depth perception.
full_transform	Calculates the 3x3 Homography matrix using Singular Value Decomposition (SVD).	Used internally by set_homography for advanced 2D perspective and projection.

5. Drawing Primitives

All drawing functions use **OpenCV (cv2)** to render onto the np.ndarray canvas. Thickness and dimensions are automatically scaled by ts['size'].

Function	Parameters	Description	Example/Usage
clear	arr, ts, color: Tuple	Fills the entire canvas with a solid color (OpenCV BGR format).	pg.clear(canvas, ts, C['DDK']) (Fills with Darkest Black)
line	arr, ts, d: Tuple, color, thickness, aa=False	Draws a line from cursor_pos to cursor_pos + d (delta). **Updates cursor_pos** to the end of the line.	pg.line(canvas, ts, (50, 0), C['W'], 2) (Draws a 50-unit horizontal line)

rect	arr, ts, wh: Tuple, color, thickness, fill=False, aa=False	Draws/fills a rectangle starting at cursor_pos with size wh.	pg.rect(canvas, ts, (20, 10), C['R'], 1, fill=True) (Draws a filled 20x10 red box)
circle	arr, ts, radius: float, color, thickness, fill=False, aa=False	Draws/fills a circle centered at cursor_pos.	pg.circle(canvas, ts, 5.0, C['BL'], 3) (Draws a blue circle outline with radius 5)
poly	arr, ts, *ds: Tuple, color, thickness, fill=False, aa=False	Draws/fills a polygon. Vertices are defined by deltas (ds) starting from cursor_pos.	pg.poly(canvas, ts, (10, 0), (0, 10), (-10, 0), C['G'], 1) (Draws a triangle)
tri	arr, ts, d1, d2, *a, **kw	Convenience wrapper for a triangle (3 vertices).	Same usage as poly with 2 delta arguments.
quad	arr, ts, d1, d2, d3, *a, **kw	Convenience wrapper for a quadrilateral (4 vertices).	Same usage as poly with 3 delta arguments.
blit	arr, ts, src_img, scale_factor: Tuple	Draws an image (src_img) at cursor_pos, applying transformations via cv2.warpPerspective. Handles alpha.	pg.blit(canvas, ts, my_array, (1.0, 1.0)) (Draws unscaled image)

blit_cached	arr, ts, asset_name: str, scale_factor: Tuple	Same as blit, but loads the image from the ASSETS manager.	pg.blit_cached(canvas, ts, 'ship', (2.0, 2.0)) (Draws cached image scaled 2x)
text	arr, ts, content: str, font_info: str, size: float, color, thickness, aa=True	**Renders text at `cursor_pos`.** Uses **Pillow/PIL** for custom font rendering or **'CV2'** for the default OpenCV font. The effective font size is `size * ts['size']`.	`pg.text(canvas, ts, "Title", "Arial.ttf", 40, C['W'], 2)` (Draws text using a custom TTF font)

---

6. Command System (.tvf)

The system allows execution of drawing and state-management commands from text strings, typically stored in a **TVF (Text Vector Format)** file. This is useful for creating complex scenes declaratively.

Function	Parameters	Description	Example/Usage
draw_command_convert	arr, ts, cmd_str, info	Parses a command string into a function and properly typed arguments. **Crucial for security** as it strictly	pg.draw_command_convert(arr, ts, "set_size, 5.0", 1)

		validates syntax.	
draw_command	arr, ts, cmd_str, info	Converts and executes a single command string.	pg.draw_command(canvas, ts, "circle, 10.0, (255, 0, 0), 2, fill=True")
draw_tvf	arr, ts, file_name: str	Reads and executes commands line-by-line from a file_name.tvf file. Ignores lines starting with #.	pg.draw_tvf(canvas, ts, 'level1') (Executes commands from level1.tvf)

---

7. Input Handling

Input listeners run in separate daemon threads via **Pynput** to track keyboard and mouse state continuously, ensuring non-blocking input.

Function	**Method**	**Key**	**Dependency**	**Returns**	**Explanation**
get_input	"held"	key: str	N/A	bool	Checks if a key (e.g., 'w', '<Key.space>') is currently pressed down.

get_input	"press"	N/A	N/A	str or None	Gets a single, low-level key press from cv2.waitKey. Returns "esc" for Escape.
get_input	"mouse"	"pos"	ts: Dict	Tuple[float]	Returns the world coordinates of the mouse cursor, factoring in the current ts using inverse_transform.
get_input	"mouse"	"buttons"	N/A	Tuple[LMB, RMB, MMB, scroll_delta]	Returns the button states and the accumulated scroll wheel movement (scroll_delta is reset to 0.0 after reading).
simulate_key_press	Key: str	N/A	N/A	N/A	Simulates a key being pressed so if it simulates “W” when “get_input” is used for w it will return true.

---

8. Application Lifecycle and Audio

Function	Parameters	Description	Example/Usage
init	window_info, bg_color	Sets up the OpenCV window and canvas, starts input listeners, and initializes the default ts dictionary.	canvas, ts = pg.init(((800, 600), "My Window"), C['K'])
run	tick_function, window_info, bg_color, target_fps=60	The main game loop. Calls tick_function(canvas, ts) every frame, handles FPS timing, display, and exit conditions.	pg.run(tick, win_info, bg_color, 60)
play_sound	asset_name: str, start_time: float=0.0	**Blocking playback.** Pauses application execution until the sound finishes playing. Use for short, critical, synchronous sounds.	pg.play_sound('intro_music')
start_sound	asset_name: str, start_time: float=0.0	**Non-Blocking playback.** Plays sound in a new daemon thread and returns immediately. Use for music or non-critical sound effects.	pg.start_sound('laser_blast')

volume_monitor	N/A	Background thread that continuously sets Pygame master volume based on GLOBAL_VOLUME[0].	Starts automatically on framework initialization.

9. Compilation System (Experimental Deployment)

Function	**Parameters**	**Description**	**Example/Usage**
compile_project	main_file_path: str	Gathers the PyGraph source, the main application source, and all project assets/scripts. **Bundles, compresses, and base64-encodes them into a single executable Python script (_c.py).** This creates a simple, single-file distribution for the project.	pg.compile_project('my_game.py') (Creates my_game_c.py)

---

10. Graphing Utility (experimental)

Function	Parameters	Description	Example/Usage
Graph	arr, ts, fn_str: str, grid: bool, range: tuple, gy: float, In_C, Grid_C, xy_size: Tuple	Plots a mathematical function defined by fn_str (e.g., "math.sin(x)") within a given range of values. Can draw a grid and uses set_pos/line to render the curve.	pg.Graph(canvas, ts, "x**2 * 0.1", True, (-100, 100), 1.0, C['R'], C['S'], True)

11. Color Dictionary (C) Table

The C dictionary contains pre-defined RGB color tuples, categorized by primary and secondary variations (Dark, Light, etc.). All colors are provided as **(R, G, B)** but are converted to **(B, G, R)** internally for OpenCV drawing.

Code	Base Color	Example RGB (255)	Code	Base Color	Example RGB (255)
W	White	(255, 255, 255)	**R**	Red	(255, 0, 0)
K	Black	(0, 0, 0)	**G**	Green	(0, 255, 0)
G	Gray	(128, 128, 128)	**BL**	Blue	(0, 0, 255)
Y	Yellow	(255, 255, 0)	**C**	Cyan	(0, 255, 255)

M	Magenta	(255, 0, 255)	**O**	Orange	(255, 165, 0)
V	Violet/Indigo	(148, 0, 211)	**S**	Silver	(192, 192, 192)
PU	Purple	(128, 0, 118)	**BR**	Brown	(139, 69, 19)
OL	Olive	(128, 128, 0)	**N**	Navy Blue	(16, 0, 128)
P	Pink	(255, 182, 193)	**TU**	Turquoise	(64, 224, 208)

Variations:

For each base color (e.g., R), there are light (LR), dark (DR), extra light (LLR), and extra dark (DDR) variations available in the dictionary.

---

12. Font and Text Utilities

The framework has utilities for advanced font handling using Pillow (PIL), including a global cache for efficiency.

Function	**Parameters**	**Description**	**Example/Usage**

text_size	text_content: str, font_info: str, size: float	**Calculates the width and height** a text string will occupy (in pixels) for a given font and size. **Does not apply ts['size'] scaling.**	w, h = pg.text_size("Hello", "CV2", 30)
load_font	font_path_or_name: str, size: int	**(Internal, cached)** Loads a font file (.ttf) or system font name at a specific point size. Uses the **global FONT_CACHE** to avoid reloading the same font object multiple times.	Used internally by text and text_size.

13. PyGraph Example Project

```
import PyGraph as pg
from PyGraph import C

win_info = ((800, 800), "base")
bg_color = C['DDK'] # Darkest Black background

def tick(canvas, ts):
    # 1. World Scale/Zoom
    pg.set_size(ts, 3) # Sets global scale to 3x

    # 2. Cursor Positioning (relative to the world origin)
    pg.set_pos(ts, (0, 0)) # Sets cursor_pos to the world origin (0, 0)

    # 3. View Panning (Shift) Logic
    # The world origin (0, 0) is shifted to follow the logic below.
    cx, cy = ts["shift"]
    if cx > 800 or cy > 800:
        pg.set_shift(ts, (0, 0)) # Reset pan when hitting boundaries
    pg.shift(ts, (5, 5)) # Increment shift, causing the view to pan diagonally

    # 4. View Rotation
    pg.rotate(ts, 10) # Rotates the entire world view by +10 degrees every frame

    # 5. Drawing Primitives (using line() implicitly updates cursor_pos)
    # The 'sword' is drawn starting from the current cursor_pos (0, 0 in world space)
    pg.line(canvas, ts, (20.0, 0.0), C['DG'], 3, aa=True) # Handle
    pg.move(ts, (0, -10)) # Move cursor up 10 units for the next segment
    pg.line(canvas, ts, (0.0, 20.0), C['G'], 3, aa=True) # Brace

    # The sword blade is drawn using sequential line calls, where the end point
    # of one line becomes the start point (cursor_pos) of the next.
    pg.line(canvas, ts, (50.0, 0.0), C['G'], 10, aa=True)
    pg.line(canvas, ts, (25.0, 0.0), C['G'], 8, aa=True)
    # ... more lines to create a tapered blade

    # Check for user input
    if pg.get_input("held", 'q'):
        return False # Exit the loop if 'q' is pressed

    return True # Keep the application running

if __name__ == "__main__":
    canvas, ts = pg.init(win_info, bg_color)
    pg.set_shift(ts, (400, 400)) # Start with the world origin centered

    pg.run(tick, win_info, bg_color, 60)
```