

Almog Draw Library

Generated by Doxygen 1.9.1

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 Curve Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	6
3.1.2.1 capacity	6
3.1.2.2 color	6
3.1.2.3 elements	6
3.1.2.4 length	6
3.2 Curve_ada Struct Reference	7
3.2.1 Detailed Description	7
3.2.2 Member Data Documentation	7
3.2.2.1 capacity	7
3.2.2.2 elements	8
3.2.2.3 length	8
3.3 Figure Struct Reference	8
3.3.1 Detailed Description	9
3.3.2 Member Data Documentation	9
3.3.2.1 background_color	9
3.3.2.2 inv_z_buffer_mat	10
3.3.2.3 max_x	10
3.3.2.4 max_x_pixel	10
3.3.2.5 max_y	10
3.3.2.6 max_y_pixel	11
3.3.2.7 min_x	11
3.3.2.8 min_x_pixel	11
3.3.2.9 min_y	11
3.3.2.10 min_y_pixel	12
3.3.2.11 offset_zoom_param	12
3.3.2.12 pixels_mat	12
3.3.2.13 src_curve_array	12
3.3.2.14 to_draw_axis	13
3.3.2.15 to_draw_max_min_values	13
3.3.2.16 top_left_position	13
3.3.2.17 x_axis_head_size	13
3.3.2.18 y_axis_head_size	14
3.4 game_state_t Struct Reference	14
3.4.1 Detailed Description	15

3.4.2 Member Data Documentation	15
3.4.2.1 a_was_pressed	15
3.4.2.2 const_fps	15
3.4.2.3 d_was_pressed	15
3.4.2.4 delta_time	16
3.4.2.5 e_was_pressed	16
3.4.2.6 elapsed_time	16
3.4.2.7 font	16
3.4.2.8 fps	16
3.4.2.9 frame_target_time	17
3.4.2.10 game_is_running	17
3.4.2.11 inv_z_buffer_mat	17
3.4.2.12 left_button_pressed	17
3.4.2.13 offset_zoom_param	17
3.4.2.14 previous_frame_time	18
3.4.2.15 q_was_pressed	18
3.4.2.16 renderer	18
3.4.2.17 s_was_pressed	18
3.4.2.18 space_bar_was_pressed	18
3.4.2.19 to_clear_renderer	19
3.4.2.20 to_limit_fps	19
3.4.2.21 to_render	19
3.4.2.22 to_update	19
3.4.2.23 w_was_pressed	19
3.4.2.24 window	20
3.4.2.25 window_h	20
3.4.2.26 window_pixels_mat	20
3.4.2.27 window_surface	20
3.4.2.28 window_texture	20
3.4.2.29 window_w	21
3.5 Grid Struct Reference	21
3.5.1 Detailed Description	22
3.5.2 Member Data Documentation	22
3.5.2.1 curves	22
3.5.2.2 de1	22
3.5.2.3 de2	22
3.5.2.4 max_e1	23
3.5.2.5 max_e2	23
3.5.2.6 min_e1	23
3.5.2.7 min_e2	23
3.5.2.8 num_samples_e1	24
3.5.2.9 num_samples_e2	24

3.5.2.10 plane	24
3.6 Mat2D Struct Reference	24
3.6.1 Detailed Description	25
3.6.2 Member Data Documentation	25
3.6.2.1 cols	25
3.6.2.2 elements	25
3.6.2.3 rows	26
3.6.2.4 stride_r	26
3.7 Mat2D_Minor Struct Reference	26
3.7.1 Detailed Description	27
3.7.2 Member Data Documentation	27
3.7.2.1 cols	27
3.7.2.2 cols_list	27
3.7.2.3 ref_mat	28
3.7.2.4 rows	28
3.7.2.5 rows_list	28
3.7.2.6 stride_r	28
3.8 Mat2D_uint32 Struct Reference	28
3.8.1 Detailed Description	29
3.8.2 Member Data Documentation	29
3.8.2.1 cols	29
3.8.2.2 elements	29
3.8.2.3 rows	30
3.8.2.4 stride_r	30
3.9 Offset_zoom_param Struct Reference	30
3.9.1 Detailed Description	30
3.9.2 Member Data Documentation	31
3.9.2.1 mouse_x	31
3.9.2.2 mouse_y	31
3.9.2.3 offset_x	31
3.9.2.4 offset_y	31
3.9.2.5 zoom_multiplier	32
3.10 Point Struct Reference	32
3.10.1 Detailed Description	32
3.10.2 Member Data Documentation	32
3.10.2.1 w	32
3.10.2.2 x	33
3.10.2.3 y	33
3.10.2.4 z	33
3.11 Quad Struct Reference	34
3.11.1 Detailed Description	34
3.11.2 Member Data Documentation	34

3.11.2.1 colors	34
3.11.2.2 light_intensity	35
3.11.2.3 normals	35
3.11.2.4 points	35
3.11.2.5 to_draw	35
3.12 Quad_mesh Struct Reference	36
3.12.1 Detailed Description	36
3.12.2 Member Data Documentation	36
3.12.2.1 capacity	36
3.12.2.2 elements	37
3.12.2.3 length	37
3.13 Tri Struct Reference	37
3.13.1 Detailed Description	38
3.13.2 Member Data Documentation	38
3.13.2.1 colors	38
3.13.2.2 light_intensity	38
3.13.2.3 normals	38
3.13.2.4 points	39
3.13.2.5 tex_points	39
3.13.2.6 to_draw	39
3.14 Tri_mesh Struct Reference	40
3.14.1 Detailed Description	40
3.14.2 Member Data Documentation	40
3.14.2.1 capacity	41
3.14.2.2 elements	41
3.14.2.3 length	41
4 File Documentation	43
4.1 Almog_Draw_Library.h File Reference	43
4.1.1 Detailed Description	48
4.1.2 Macro Definition Documentation	48
4.1.2.1 ADL_ASSERT	49
4.1.2.2 adl_assert_point_is_valid	49
4.1.2.3 adl_assert_quad_is_valid	49
4.1.2.4 adl_assert_tri_is_valid	49
4.1.2.5 ADL_DEFAULT_OFFSET_ZOOM	50
4.1.2.6 ADL FIGURE_AXIS_COLOR	50
4.1.2.7 ADL FIGURE_HEAD_ANGLE_DEG	50
4.1.2.8 ADL FIGURE_PADDING_PERCENTAGE	50
4.1.2.9 ADL_MAX_CHARACTER_OFFSET	50
4.1.2.10 ADL_MAX FIGURE_PADDING	50
4.1.2.11 ADL_MAX_HEAD_SIZE	51

4.1.2.12 ADL_MAX_POINT_VAL	51
4.1.2.13 ADL_MAX_SENTENCE_LEN	51
4.1.2.14 ADL_MAX_ZOOM	51
4.1.2.15 ADL_MIN_CHARACTER_OFFSET	51
4.1.2.16 ADL_MIN FIGURE_PADDING	51
4.1.2.17 adl_offset2d	52
4.1.2.18 adl_offset_zoom_point	52
4.1.2.19 BLUE_hexARGB	52
4.1.2.20 CURVE	52
4.1.2.21 CURVE_ADA	52
4.1.2.22 CYAN_hexARGB	53
4.1.2.23 edge_cross_point	53
4.1.2.24 GREEN_hexARGB	53
4.1.2.25 HexARGB_RGB_VAR	53
4.1.2.26 HexARGB_RGBA	53
4.1.2.27 HexARGB_RGBA_VAR	54
4.1.2.28 is_left_edge	54
4.1.2.29 is_top_edge	54
4.1.2.30 is_top_left	54
4.1.2.31 POINT	54
4.1.2.32 PURPLE_hexARGB	55
4.1.2.33 QUAD	55
4.1.2.34 QUAD_MESH	55
4.1.2.35 RED_hexARGB	55
4.1.2.36 RGB_hexRGB	55
4.1.2.37 RGBA_hexARGB	55
4.1.2.38 TRI	56
4.1.2.39 TRI_MESH	56
4.1.2.40 YELLOW_hexARGB	56
4.1.3 Function Documentation	56
4.1.3.1 adl_2Dscalar_interp_on_figure()	56
4.1.3.2 adl_arrow_draw()	57
4.1.3.3 adl_axis_draw_on_figure()	58
4.1.3.4 adl_cartesian_grid_create()	58
4.1.3.5 adl_character_draw()	59
4.1.3.6 adl_circle_draw()	59
4.1.3.7 adl_circle_fill()	60
4.1.3.8 adl_curve_add_to_figure()	61
4.1.3.9 adl_curves_plot_on_figure()	61
4.1.3.10 adl_figure_alloc()	62
4.1.3.11 adl_figure_copy_to_screen()	62
4.1.3.12 adl_grid_draw()	63

4.1.3.13 adl_interpolate_ARGBcolor_on_okLch()	63
4.1.3.14 adl_line_draw()	64
4.1.3.15 adl_linear_map()	64
4.1.3.16 adl_linear_sRGB_to_okLab()	65
4.1.3.17 adl_linear_sRGB_to_okLch()	65
4.1.3.18 adl_lines_draw()	66
4.1.3.19 adl_lines_loop_draw()	67
4.1.3.20 adl_max_min_values_draw_on_figure()	67
4.1.3.21 adl_okLab_to_linear_sRGB()	68
4.1.3.22 adl_okLch_to_linear_sRGB()	68
4.1.3.23 adl_point_draw()	69
4.1.3.24 adl_quad2tris()	69
4.1.3.25 adl_quad_draw()	70
4.1.3.26 adl_quad_fill()	70
4.1.3.27 adl_quad_fill_interpolate_color_mean_value()	71
4.1.3.28 adl_quad_fill_interpolate_normal_mean_value()	72
4.1.3.29 adl_quad_mesh_draw()	72
4.1.3.30 adl_quad_mesh_fill()	73
4.1.3.31 adl_quad_mesh_fill_interpolate_color()	73
4.1.3.32 adl_quad_mesh_fill_interpolate_normal()	74
4.1.3.33 adl_rectangle_draw_min_max()	74
4.1.3.34 adl_rectangle_fill_min_max()	75
4.1.3.35 adl_sentence_draw()	75
4.1.3.36 adl_tan_half_angle()	76
4.1.3.37 adl_tri_draw()	77
4.1.3.38 adl_tri_fill_Pinedas_rasterizer()	77
4.1.3.39 adl_tri_fill_Pinedas_rasterizer_interpolate_color()	78
4.1.3.40 adl_tri_fill_Pinedas_rasterizer_interpolate_normal()	78
4.1.3.41 adl_tri_mesh_draw()	79
4.1.3.42 adl_tri_mesh_fill_Pinedas_rasterizer()	80
4.1.3.43 adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color()	80
4.1.3.44 adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal()	81
4.2 Almog_Draw_Library.h	81
4.3 Almog_Dynamic_Array.h File Reference	111
4.3.1 Detailed Description	113
4.3.2 Macro Definition Documentation	113
4.3.2.1 ada_appand	114
4.3.2.2 ADA_ASSERT	114
4.3.2.3 ada_init_array	114
4.3.2.4 ADA_INIT_CAPACITY	115
4.3.2.5 ada_insert	115
4.3.2.6 ada_insert_unordered	116

4.3.2.7 ADA_MALLOC	117
4.3.2.8 ADA_REALLOC	117
4.3.2.9 ada_remove	117
4.3.2.10 ada_remove_unordered	118
4.3.2.11 ada_resize	119
4.4 Almog_Dynamic_Array.h	120
4.5 display.c File Reference	121
4.5.1 Macro Definition Documentation	123
4.5.1.1 dprintCHAR	123
4.5.1.2 dprintD	123
4.5.1.3 dprintINT	123
4.5.1.4 dprintSIZE_T	123
4.5.1.5 dprintSTRING	123
4.5.1.6 FPS	124
4.5.1.7 FRAME_TARGET_TIME	124
4.5.1.8 RENDER	124
4.5.1.9 SETUP	124
4.5.1.10 UPDATE	124
4.5.1.11 WINDOW_HEIGHT	124
4.5.1.12 WINDOW_WIDTH	125
4.5.2 Function Documentation	125
4.5.2.1 check_window_mat_size()	125
4.5.2.2 copy_mat_to_surface_RGB()	125
4.5.2.3 destroy_window()	125
4.5.2.4 fix_framerate()	126
4.5.2.5 initialize_window()	126
4.5.2.6 main()	126
4.5.2.7 process_input_window()	126
4.5.2.8 render()	127
4.5.2.9 render_window()	127
4.5.2.10 setup()	127
4.5.2.11 setup_window()	127
4.5.2.12 update()	128
4.5.2.13 update_window()	128
4.6 display.c	128
4.7 example1.c File Reference	133
4.7.1 Macro Definition Documentation	133
4.7.1.1 ALMOG_DRAW_LIBRARY_IMPLEMENTATION	134
4.7.1.2 MATRIX2D_IMPLEMENTATION	134
4.7.1.3 RENDER	134
4.7.1.4 SETUP	134
4.7.1.5 UPDATE	134

4.7.2 Function Documentation	134
4.7.2.1 render()	135
4.7.2.2 setup()	135
4.7.2.3 update()	135
4.7.3 Variable Documentation	135
4.7.3.1 figure1	135
4.7.3.2 figure2	136
4.7.3.3 points	136
4.7.3.4 points1	136
4.8 example1.c	136
4.9 Matrix2D.h File Reference	138
4.9.1 Detailed Description	142
4.9.2 Macro Definition Documentation	143
4.9.2.1 __USE_MISC	143
4.9.2.2 MAT2D_AT	143
4.9.2.3 MAT2D_AT_UINT32	143
4.9.2.4 MAT2D_MINOR_AT	144
4.9.2.5 MAT2D_MINOR_PRINT	144
4.9.2.6 mat2D_normalize	144
4.9.2.7 MAT2D_PRINT	144
4.9.2.8 MAT2D_PRINT_AS_COL	145
4.9.2.9 MATRIX2D_ASSERT	145
4.9.2.10 MATRIX2D_MALLOC	145
4.9.2.11 PI	145
4.9.3 Function Documentation	145
4.9.3.1 mat2D_add()	145
4.9.3.2 mat2D_add_col_to_col()	146
4.9.3.3 mat2D_add_row_time_factor_to_row()	146
4.9.3.4 mat2D_add_row_to_row()	147
4.9.3.5 mat2D_alloc()	147
4.9.3.6 mat2D_alloc_uint32()	148
4.9.3.7 mat2D_calc_norma()	148
4.9.3.8 mat2D_col_is_all_digit()	149
4.9.3.9 mat2D_copy()	149
4.9.3.10 mat2D_copy_mat_to_mat_at_window()	150
4.9.3.11 mat2D_cross()	151
4.9.3.12 mat2D_det()	151
4.9.3.13 mat2D_det_2x2_mat()	152
4.9.3.14 mat2D_det_2x2_mat_minor()	152
4.9.3.15 mat2D_dot()	152
4.9.3.16 mat2D_dot_product()	153
4.9.3.17 mat2D_fill()	154

4.9.3.18 mat2D_fill_sequence()	154
4.9.3.19 mat2D_fill_uint32()	154
4.9.3.20 mat2D_free()	155
4.9.3.21 mat2D_free_uint32()	155
4.9.3.22 mat2D_get_col()	156
4.9.3.23 mat2D_get_row()	156
4.9.3.24 mat2D_invert()	157
4.9.3.25 mat2D_LUP_decomposition_with_swap()	157
4.9.3.26 mat2D_make_identity()	158
4.9.3.27 mat2D_mat_is_all_digit()	159
4.9.3.28 mat2D_minor_alloc_fill_from_mat()	159
4.9.3.29 mat2D_minor_alloc_fill_from_mat_minor()	160
4.9.3.30 mat2D_minor_det()	160
4.9.3.31 mat2D_minor_free()	161
4.9.3.32 mat2D_minor_print()	161
4.9.3.33 mat2D_mult()	162
4.9.3.34 mat2D_mult_row()	162
4.9.3.35 mat2D_offset2d()	162
4.9.3.36 mat2D_offset2d_uint32()	164
4.9.3.37 mat2D_print()	165
4.9.3.38 mat2D_print_as_col()	165
4.9.3.39 mat2D_rand()	165
4.9.3.40 mat2D_rand_double()	166
4.9.3.41 mat2D_row_is_all_digit()	166
4.9.3.42 mat2D_set_DCM_zyx()	167
4.9.3.43 mat2D_set_identity()	167
4.9.3.44 mat2D_set_rot_mat_x()	168
4.9.3.45 mat2D_set_rot_mat_y()	168
4.9.3.46 mat2D_set_rot_mat_z()	169
4.9.3.47 mat2D_solve_linear_sys_LUP_decomposition()	169
4.9.3.48 mat2D_sub()	170
4.9.3.49 mat2D_sub_col_to_col()	170
4.9.3.50 mat2D_sub_row_time_factor_to_row()	170
4.9.3.51 mat2D_sub_row_to_row()	171
4.9.3.52 mat2D_swap_rows()	171
4.9.3.53 mat2D_transpose()	172
4.9.3.54 mat2D_triangulate()	172
4.10 Matrix2D.h	173
4.11 temp.c File Reference	184
4.11.1 Macro Definition Documentation	185
4.11.1.1 ALMOG_DRAW_LIBRARY_IMPLEMENTATION	185
4.11.1.2 MATRIX2D_IMPLEMENTATION	185

4.11.1.3 RENDER	185
4.11.1.4 SETUP	185
4.11.1.5 UPDATE	185
4.11.2 Function Documentation	186
4.11.2.1 render()	186
4.11.2.2 setup()	186
4.11.2.3 update()	186
4.11.3 Variable Documentation	186
4.11.3.1 quad1	186
4.11.3.2 tri	187
4.12 temp.c	187
Index	189

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Curve	Polyline of points with a uniform color	5
Curve_ada	Dynamic array of curves (polyline container)	7
Figure	Plotting figure holding a pixel buffer, z-buffer and plot state	8
game_state_t	14
Grid	Grid definition (as lines) in a chosen plane	21
Mat2D	Dense row-major matrix of doubles	24
Mat2D_Minor	A minor "view" into a reference matrix	26
Mat2D_uint32	Dense row-major matrix of uint32_t	28
Offset_zoom_param	Pan/zoom parameters relative to screen center	30
Point	Homogeneous 2D/3D point with per-vertex depth (z) and w	32
Quad	Quad primitive with optional per-vertex attributes	34
Quad_mesh	Dynamic array of quads (quad mesh)	36
Tri	Triangle primitive with optional per-vertex attributes	37
Tri_mesh	Dynamic array of triangles (triangle mesh)	40

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Almog_Draw_Library.h	Immediate-mode 2D/3D raster helpers for drawing onto <code>Mat2D_uint32</code> pixel buffers	43
Almog_Dynamic_Array.h	Header-only C macros that implement a simple dynamic array	111
display.c		121
example1.c		133
Matrix2D.h	A single-header C library for simple 2D matrix operations on doubles and <code>uint32_t</code> , including allocation, basic arithmetic, linear algebra, and helpers (LUP, inverse, determinant, DCM, etc.)	138
temp.c		184

Chapter 3

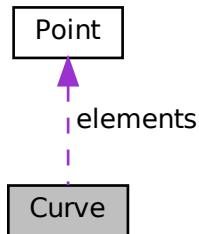
Class Documentation

3.1 Curve Struct Reference

Polyline of points with a uniform color.

```
#include <Almog_Draw_Library.h>
```

Collaboration diagram for Curve:



Public Attributes

- `uint32_t color`
- `size_t length`
- `size_t capacity`
- `Point * elements`

3.1.1 Detailed Description

Polyline of points with a uniform color.

Definition at line 94 of file [Almog_Draw_Library.h](#).

3.1.2 Member Data Documentation

3.1.2.1 capacity

```
size_t Curve::capacity
```

Allocated capacity.

Definition at line 97 of file [Almog_Draw_Library.h](#).

3.1.2.2 color

```
uint32_t Curve::color
```

ARGB color (0xAARRGGBB) for the entire curve.

Definition at line 95 of file [Almog_Draw_Library.h](#).

Referenced by [adl_curve_add_to_figure\(\)](#), and [adl_curves_plot_on_figure\(\)](#).

3.1.2.3 elements

```
Point* Curve::elements
```

[Point](#) array.

Definition at line 98 of file [Almog_Draw_Library.h](#).

Referenced by [adl_curves_plot_on_figure\(\)](#), [adl_grid_draw\(\)](#), and [setup\(\)](#).

3.1.2.4 length

```
size_t Curve::length
```

Number of points used.

Definition at line 96 of file [Almog_Draw_Library.h](#).

Referenced by [adl_curves_plot_on_figure\(\)](#), [adl_grid_draw\(\)](#), and [setup\(\)](#).

The documentation for this struct was generated from the following file:

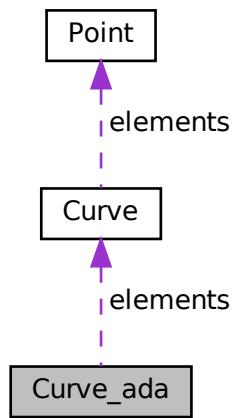
- [Almog_Draw_Library.h](#)

3.2 Curve_ada Struct Reference

Dynamic array of curves (polyline container).

```
#include <Almog_Draw_Library.h>
```

Collaboration diagram for Curve_ada:



Public Attributes

- size_t `length`
- size_t `capacity`
- `Curve * elements`

3.2.1 Detailed Description

Dynamic array of curves (polyline container).

Definition at line 107 of file [Almog_Draw_Library.h](#).

3.2.2 Member Data Documentation

3.2.2.1 capacity

```
size_t Curve_ada::capacity
```

Allocated capacity.

Definition at line 109 of file [Almog_Draw_Library.h](#).

3.2.2.2 elements

```
Curve* Curve_ada::elements
```

Curves array.

Definition at line 110 of file [Almog_Draw_Library.h](#).

Referenced by [adl_curves_plot_on_figure\(\)](#), and [adl_grid_draw\(\)](#).

3.2.2.3 length

```
size_t Curve_ada::length
```

Number of curves used.

Definition at line 108 of file [Almog_Draw_Library.h](#).

Referenced by [adl_curves_plot_on_figure\(\)](#), and [adl_grid_draw\(\)](#).

The documentation for this struct was generated from the following file:

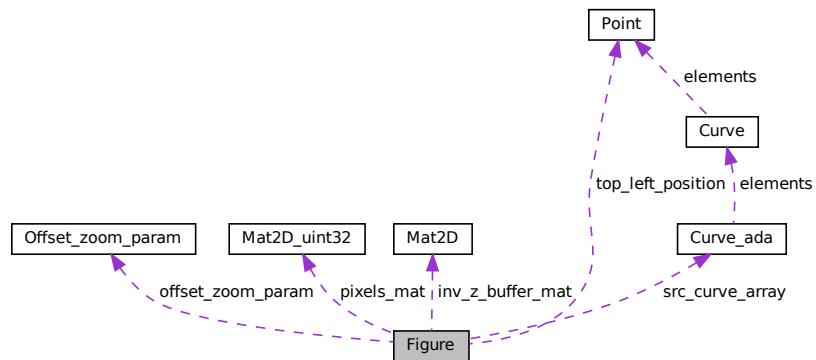
- [Almog_Draw_Library.h](#)

3.3 Figure Struct Reference

Plotting figure holding a pixel buffer, z-buffer and plot state.

```
#include <Almog_Draw_Library.h>
```

Collaboration diagram for Figure:



Public Attributes

- int `min_x_pixel`
- int `max_x_pixel`
- int `min_y_pixel`
- int `max_y_pixel`
- float `min_x`
- float `max_x`
- float `min_y`
- float `max_y`
- int `x_axis_head_size`
- int `y_axis_head_size`
- `Offset_zoom_param offset_zoom_param`
- `Curve_ada src_curve_array`
- `Point top_left_position`
- `Mat2D<uint32> pixels_mat`
- `Mat2D<float> inv_z_buffer_mat`
- `uint32_t background_color`
- bool `to_draw_axis`
- bool `to_draw_max_min_values`

3.3.1 Detailed Description

Plotting figure holding a pixel buffer, z-buffer and plot state.

A [Figure](#) owns an internal pixel buffer and an inverse-Z buffer used by the plotting utilities. It also stores axis extents, paddings and appearance flags.

Definition at line 174 of file [Almog_Draw_Library.h](#).

3.3.2 Member Data Documentation

3.3.2.1 `background_color`

```
uint32_t Figure::background_color
```

Clear color for figure.

Definition at line 195 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), and [setup\(\)](#).

3.3.2.2 inv_z_buffer_mat

`Mat2D Figure::inv_z_buffer_mat`

Owned inverse-Z buffer (double).

Definition at line 193 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), and [adl_figure_alloc\(\)](#).

3.3.2.3 max_x

`float Figure::max_x`

Max X value in source data.

Definition at line 181 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curve_add_to_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

3.3.2.4 max_x_pixel

`int Figure::max_x_pixel`

Right bound (pixel space).

Definition at line 176 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_axis_draw_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

3.3.2.5 max_y

`float Figure::max_y`

Max Y value in source data.

Definition at line 183 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curve_add_to_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

3.3.2.6 max_y_pixel

```
int Figure::max_y_pixel
```

Bottom bound (pixel space).

Definition at line 178 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_axis_draw_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

3.3.2.7 min_x

```
float Figure::min_x
```

Min X value in source data.

Definition at line 180 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curve_add_to_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

3.3.2.8 min_x_pixel

```
int Figure::min_x_pixel
```

Left padding (pixel space).

Definition at line 175 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_axis_draw_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

3.3.2.9 min_y

```
float Figure::min_y
```

Min Y value in source data.

Definition at line 182 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curve_add_to_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

3.3.2.10 min_y_pixel

```
int Figure::min_y_pixel
```

Top padding (pixel space).

Definition at line 177 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_axis_draw_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

3.3.2.11 offset_zoom_param

```
Offset_zoom_param Figure::offset_zoom_param
```

Pan/zoom parameters.

Definition at line 188 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_axis_draw_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

3.3.2.12 pixels_mat

```
Mat2D_uint32 Figure::pixels_mat
```

Owned ARGB pixel buffer.

Definition at line 192 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_axis_draw_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), [adl_figure_copy_to_screen\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

3.3.2.13 src_curve_array

```
Curve_ada Figure::src_curve_array
```

Curves to plot.

Definition at line 189 of file [Almog_Draw_Library.h](#).

Referenced by [adl_curve_add_to_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), and [adl_figure_alloc\(\)](#).

3.3.2.14 `to_draw_axis`

```
bool Figure::to_draw_axis
```

Draw axes when plotting.

Definition at line 196 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), and [setup\(\)](#).

3.3.2.15 `to_draw_max_min_values`

```
bool Figure::to_draw_max_min_values
```

Draw min/max labels.

Definition at line 197 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), and [setup\(\)](#).

3.3.2.16 `top_left_position`

```
Point Figure::top_left_position
```

On-screen copy position.

Definition at line 190 of file [Almog_Draw_Library.h](#).

Referenced by [adl_figure_alloc\(\)](#), and [adl_figure_copy_to_screen\(\)](#).

3.3.2.17 `x_axis_head_size`

```
int Figure::x_axis_head_size
```

Computed X-axis arrow head size (px).

Definition at line 185 of file [Almog_Draw_Library.h](#).

Referenced by [adl_axis_draw_on_figure\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

3.3.2.18 `y_axis_head_size`

```
int Figure::y_axis_head_size
```

Computed Y-axis arrow head size (px).

Definition at line 186 of file [Almog_Draw_Library.h](#).

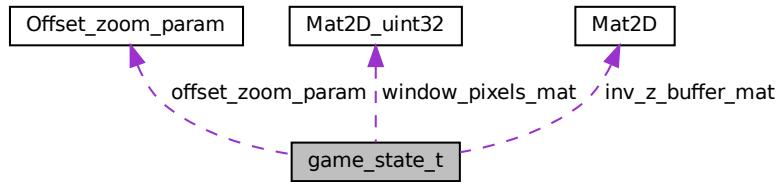
Referenced by [adl_axis_draw_on_figure\(\)](#), and [adl_max_min_values_draw_on_figure\(\)](#).

The documentation for this struct was generated from the following file:

- [Almog_Draw_Library.h](#)

3.4 game_state_t Struct Reference

Collaboration diagram for game_state_t:



Public Attributes

- int `game_is_running`
- float `delta_time`
- float `elapsed_time`
- float `const_fps`
- float `fps`
- float `frame_target_time`
- int `to_render`
- int `to_update`
- size_t `previous_frame_time`
- int `left_button_pressed`
- int `to_limit_fps`
- int `to_clear_renderer`
- int `space_bar_was_pressed`
- int `w_was_pressed`
- int `s_was_pressed`
- int `a_was_pressed`
- int `d_was_pressed`
- int `e_was_pressed`
- int `q_was_pressed`

- `SDL_Window * window`
- `int window_w`
- `int window_h`
- `SDL_Renderer * renderer`
- `TTF_Font * font`
- `SDL_Surface * window_surface`
- `SDL_Texture * window_texture`
- `Mat2D_uint32 window_pixels_mat`
- `Mat2D inv_z_buffer_mat`
- `Offset_zoom_param offset_zoom_param`

3.4.1 Detailed Description

Definition at line 38 of file [display.c](#).

3.4.2 Member Data Documentation

3.4.2.1 a_was_pressed

```
int game_state_t::a_was_pressed
```

Definition at line 55 of file [display.c](#).

Referenced by [main\(\)](#).

3.4.2.2 const_fps

```
float game_state_t::const_fps
```

Definition at line 42 of file [display.c](#).

Referenced by [main\(\)](#), [setup\(\)](#), and [update_window\(\)](#).

3.4.2.3 d_was_pressed

```
int game_state_t::d_was_pressed
```

Definition at line 56 of file [display.c](#).

Referenced by [main\(\)](#).

3.4.2.4 delta_time

```
float game_state_t::delta_time
```

Definition at line 40 of file [display.c](#).

Referenced by [fix_framerate\(\)](#), [main\(\)](#), and [update_window\(\)](#).

3.4.2.5 e_was_pressed

```
int game_state_t::e_was_pressed
```

Definition at line 57 of file [display.c](#).

Referenced by [main\(\)](#).

3.4.2.6 elapsed_time

```
float game_state_t::elapsed_time
```

Definition at line 41 of file [display.c](#).

Referenced by [main\(\)](#), and [update_window\(\)](#).

3.4.2.7 font

```
TTF_Font* game_state_t::font
```

Definition at line 64 of file [display.c](#).

Referenced by [main\(\)](#).

3.4.2.8 fps

```
float game_state_t::fps
```

Definition at line 43 of file [display.c](#).

Referenced by [main\(\)](#), and [update_window\(\)](#).

3.4.2.9 frame_target_time

```
float game_state_t::frame_target_time
```

Definition at line 44 of file [display.c](#).

Referenced by [fix_framerate\(\)](#), [main\(\)](#), and [update_window\(\)](#).

3.4.2.10 game_is_running

```
int game_state_t::game_is_running
```

Definition at line 39 of file [display.c](#).

Referenced by [main\(\)](#), and [process_input_window\(\)](#).

3.4.2.11 inv_z_buffer_mat

```
Mat2D game_state_t::inv_z_buffer_mat
```

Definition at line 70 of file [display.c](#).

Referenced by [check_window_mat_size\(\)](#), [render\(\)](#), [render_window\(\)](#), and [setup_window\(\)](#).

3.4.2.12 left_button_pressed

```
int game_state_t::left_button_pressed
```

Definition at line 48 of file [display.c](#).

Referenced by [main\(\)](#), and [process_input_window\(\)](#).

3.4.2.13 offset_zoom_param

```
OffsetZoomParam game_state_t::offset_zoom_param
```

Definition at line 72 of file [display.c](#).

Referenced by [main\(\)](#), [process_input_window\(\)](#), and [render\(\)](#).

3.4.2.14 previous_frame_time

```
size_t game_state_t::previous_frame_time
```

Definition at line 47 of file [display.c](#).

Referenced by [fix_framerate\(\)](#), [main\(\)](#), and [process_input_window\(\)](#).

3.4.2.15 q_was_pressed

```
int game_state_t::q_was_pressed
```

Definition at line 58 of file [display.c](#).

Referenced by [main\(\)](#).

3.4.2.16 renderer

```
SDL_Renderer* game_state_t::renderer
```

Definition at line 63 of file [display.c](#).

Referenced by [destroy_window\(\)](#), [initialize_window\(\)](#), and [main\(\)](#).

3.4.2.17 s_was_pressed

```
int game_state_t::s_was_pressed
```

Definition at line 54 of file [display.c](#).

Referenced by [main\(\)](#).

3.4.2.18 space_bar_was_pressed

```
int game_state_t::space_bar_was_pressed
```

Definition at line 52 of file [display.c](#).

Referenced by [main\(\)](#), and [process_input_window\(\)](#).

3.4.2.19 to_clear_renderer

```
int game_state_t::to_clear_renderer
```

Definition at line 50 of file [display.c](#).

Referenced by [main\(\)](#), and [render_window\(\)](#).

3.4.2.20 to_limit_fps

```
int game_state_t::to_limit_fps
```

Definition at line 49 of file [display.c](#).

Referenced by [fix_framerate\(\)](#), [main\(\)](#), [setup\(\)](#), and [update_window\(\)](#).

3.4.2.21 to_render

```
int game_state_t::to_render
```

Definition at line 45 of file [display.c](#).

Referenced by [main\(\)](#), and [process_input_window\(\)](#).

3.4.2.22 to_update

```
int game_state_t::to_update
```

Definition at line 46 of file [display.c](#).

Referenced by [main\(\)](#), and [process_input_window\(\)](#).

3.4.2.23 w_was_pressed

```
int game_state_t::w_was_pressed
```

Definition at line 53 of file [display.c](#).

Referenced by [main\(\)](#).

3.4.2.24 window

```
SDL_Window* game_state_t::window
```

Definition at line 60 of file [display.c](#).

Referenced by [check_window_mat_size\(\)](#), [destroy_window\(\)](#), [initialize_window\(\)](#), [main\(\)](#), [render_window\(\)](#), [setup_window\(\)](#), and [update_window\(\)](#).

3.4.2.25 window_h

```
int game_state_t::window_h
```

Definition at line 62 of file [display.c](#).

Referenced by [check_window_mat_size\(\)](#), [initialize_window\(\)](#), [main\(\)](#), [setup_window\(\)](#), and [update_window\(\)](#).

3.4.2.26 window_pixels_mat

```
Mat2D_uint32 game_state_t::window_pixels_mat
```

Definition at line 69 of file [display.c](#).

Referenced by [check_window_mat_size\(\)](#), [copy_mat_to_surface_RGB\(\)](#), [destroy_window\(\)](#), [render\(\)](#), [render_window\(\)](#), and [setup_window\(\)](#).

3.4.2.27 window_surface

```
SDL_Surface* game_state_t::window_surface
```

Definition at line 66 of file [display.c](#).

Referenced by [check_window_mat_size\(\)](#), [copy_mat_to_surface_RGB\(\)](#), [destroy_window\(\)](#), and [setup_window\(\)](#).

3.4.2.28 window_texture

```
SDL_Texture* game_state_t::window_texture
```

Definition at line 67 of file [display.c](#).

Referenced by [destroy_window\(\)](#).

3.4.2.29 window_w

```
int game_state_t::window_w
```

Definition at line 61 of file [display.c](#).

Referenced by [check_window_mat_size\(\)](#), [initialize_window\(\)](#), [main\(\)](#), [setup_window\(\)](#), and [update_window\(\)](#).

The documentation for this struct was generated from the following file:

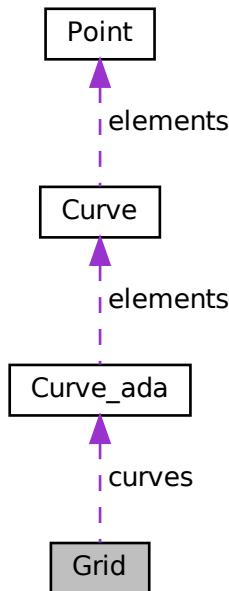
- [display.c](#)

3.5 Grid Struct Reference

[Grid](#) definition (as lines) in a chosen plane.

```
#include <Almog_Draw_Library.h>
```

Collaboration diagram for Grid:



Public Attributes

- [Curve_ada curves](#)
- float [min_e1](#)
- float [max_e1](#)
- float [min_e2](#)
- float [max_e2](#)
- int [num_samples_e1](#)
- int [num_samples_e2](#)
- float [de1](#)
- float [de2](#)
- char [plane](#) [3]

3.5.1 Detailed Description

`Grid` definition (as lines) in a chosen plane.

Definition at line 203 of file [Almog_Draw_Library.h](#).

3.5.2 Member Data Documentation

3.5.2.1 curves

`Curve_ada Grid::curves`

Line segments implementing the grid.

Definition at line 204 of file [Almog_Draw_Library.h](#).

Referenced by [adl_cartesian_grid_create\(\)](#), and [adl_grid_draw\(\)](#).

3.5.2.2 de1

`float Grid::de1`

Step size along axis 1.

Definition at line 213 of file [Almog_Draw_Library.h](#).

Referenced by [adl_cartesian_grid_create\(\)](#).

3.5.2.3 de2

`float Grid::de2`

Step size along axis 2.

Definition at line 214 of file [Almog_Draw_Library.h](#).

Referenced by [adl_cartesian_grid_create\(\)](#).

3.5.2.4 max_e1

```
float Grid::max_e1
```

Axis 1 max.

Definition at line 207 of file [Almog_Draw_Library.h](#).

Referenced by [adl_cartesian_grid_create\(\)](#).

3.5.2.5 max_e2

```
float Grid::max_e2
```

Axis 2 max.

Definition at line 209 of file [Almog_Draw_Library.h](#).

Referenced by [adl_cartesian_grid_create\(\)](#).

3.5.2.6 min_e1

```
float Grid::min_e1
```

Axis 1 min.

Definition at line 206 of file [Almog_Draw_Library.h](#).

Referenced by [adl_cartesian_grid_create\(\)](#).

3.5.2.7 min_e2

```
float Grid::min_e2
```

Axis 2 min.

Definition at line 208 of file [Almog_Draw_Library.h](#).

Referenced by [adl_cartesian_grid_create\(\)](#).

3.5.2.8 num_samples_e1

```
int Grid::num_samples_e1
```

Number of divisions along axis 1.

Definition at line 211 of file [Almog_Draw_Library.h](#).

Referenced by [adl_cartesian_grid_create\(\)](#).

3.5.2.9 num_samples_e2

```
int Grid::num_samples_e2
```

Number of divisions along axis 2.

Definition at line 212 of file [Almog_Draw_Library.h](#).

Referenced by [adl_cartesian_grid_create\(\)](#).

3.5.2.10 plane

```
char Grid::plane[3]
```

Plane tag: "XY", "XZ", "YZ", "YX", "ZX", "ZY".

Definition at line 216 of file [Almog_Draw_Library.h](#).

Referenced by [adl_cartesian_grid_create\(\)](#).

The documentation for this struct was generated from the following file:

- [Almog_Draw_Library.h](#)

3.6 Mat2D Struct Reference

Dense row-major matrix of doubles.

```
#include <Matrix2D.h>
```

Public Attributes

- `size_t rows`
- `size_t cols`
- `size_t stride_r`
- `double * elements`

3.6.1 Detailed Description

Dense row-major matrix of doubles.

- rows: number of rows (height)
- cols: number of columns (width)
- stride_r: number of elements between successive rows in memory (for contiguous storage, stride_r == cols)
- elements: pointer to contiguous storage of size rows * cols

Definition at line 81 of file [Matrix2D.h](#).

3.6.2 Member Data Documentation

3.6.2.1 cols

```
size_t Mat2D::cols
```

Definition at line 83 of file [Matrix2D.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), [mat2D_add\(\)](#), [mat2D_add_col_to_col\(\)](#), [mat2D_add_row_time_factor_to_row\(\)](#), [mat2D_add_row_to_row\(\)](#), [mat2D_alloc\(\)](#), [mat2D_calc_norma\(\)](#), [mat2D_col_is_all_digit\(\)](#), [mat2D_copy\(\)](#), [mat2D_copy_mat_to_mat_at_window\(\)](#), [mat2D_cross\(\)](#), [mat2D_det\(\)](#), [mat2D_det_2x2_mat\(\)](#), [mat2D_dot\(\)](#), [mat2D_dot_product\(\)](#), [mat2D_fill\(\)](#), [mat2D_fill_sequence\(\)](#), [mat2D_get_col\(\)](#), [mat2D_get_row\(\)](#), [mat2D_invert\(\)](#), [mat2D_LUP_decomposition_with_swap\(\)](#), [mat2D_make_identity\(\)](#), [mat2D_mat_is_all_digit\(\)](#), [mat2D_minor_alloc_fill_from_mat\(\)](#), [mat2D_mult\(\)](#), [mat2D_mult_row\(\)](#), [mat2D_offset2d\(\)](#), [mat2D_print\(\)](#), [mat2D_print_as_col\(\)](#), [mat2D_rand\(\)](#), [mat2D_row_is_all_digit\(\)](#), [mat2D_set_identity\(\)](#), [mat2D_set_rot_mat_x\(\)](#), [mat2D_set_rot_mat_y\(\)](#), [mat2D_set_rot_mat_z\(\)](#), [mat2D_solve_linear_sys_LUP_decomposition\(\)](#), [mat2D_sub\(\)](#), [mat2D_sub_col_to_col\(\)](#), [mat2D_sub_row_time_factor_to_row\(\)](#), [mat2D_sub_row_to_row\(\)](#), [mat2D_swap_rows\(\)](#), [mat2D_transpose\(\)](#), [mat2D_triangulate\(\)](#), [render\(\)](#), and [render_window\(\)](#).

3.6.2.2 elements

```
double* Mat2D::elements
```

Definition at line 85 of file [Matrix2D.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), [mat2D_alloc\(\)](#), [mat2D_free\(\)](#), [mat2D_print_as_col\(\)](#), and [render_window\(\)](#).

3.6.2.3 rows

```
size_t Mat2D::rows
```

Definition at line 82 of file [Matrix2D.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_alloc\(\)](#), [mat2D_add\(\)](#), [mat2D_add_col_to_col\(\)](#), [mat2D_add_row_to_row\(\)](#), [mat2D_alloc\(\)](#), [mat2D_calc_norma\(\)](#), [mat2D_copy\(\)](#), [mat2D_copy_mat_to_mat_at_window\(\)](#), [mat2D_cross\(\)](#), [mat2D_det\(\)](#), [mat2D_det_2x2_mat\(\)](#), [mat2D_dot\(\)](#), [mat2D_dot_product\(\)](#), [mat2D_fill\(\)](#), [mat2D_fill_sequence\(\)](#), [mat2D_get_col\(\)](#), [mat2D_get_row\(\)](#), [mat2D_invert\(\)](#), [mat2D_LUP_decomposition_with_swap\(\)](#), [mat2D_make_identity\(\)](#), [mat2D_mat_is_all_digit\(\)](#), [mat2D_minor_alloc_fill_from_mat\(\)](#), [mat2D_mult\(\)](#), [mat2D_offset2d\(\)](#), [mat2D_print\(\)](#), [mat2D_print_as_col\(\)](#), [mat2D_rand\(\)](#), [mat2D_set_identity\(\)](#), [mat2D_set_rot_mat_x\(\)](#), [mat2D_set_rot_mat_y\(\)](#), [mat2D_set_rot_mat_z\(\)](#), [mat2D_solve_linear_sys_LUP_decomposition\(\)](#), [mat2D_sub\(\)](#), [mat2D_sub_col_to_col\(\)](#), [mat2D_sub_row_to_row\(\)](#), [mat2D_transpose\(\)](#), [mat2D_triangulate\(\)](#), [render\(\)](#), and [render_window\(\)](#).

3.6.2.4 stride_r

```
size_t Mat2D::stride_r
```

Definition at line 84 of file [Matrix2D.h](#).

Referenced by [mat2D_alloc\(\)](#), and [mat2D_offset2d\(\)](#).

The documentation for this struct was generated from the following file:

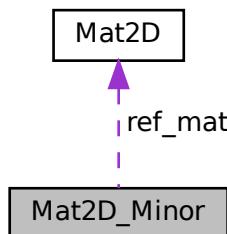
- [Matrix2D.h](#)

3.7 Mat2D_Minor Struct Reference

A minor "view" into a reference matrix.

```
#include <Matrix2D.h>
```

Collaboration diagram for Mat2D_Minor:



Public Attributes

- `size_t rows`
- `size_t cols`
- `size_t stride_r`
- `size_t * rows_list`
- `size_t * cols_list`
- `Mat2D ref_mat`

3.7.1 Detailed Description

A minor "view" into a reference matrix.

Represents a minor by excluding one row and one column of a reference matrix. It holds index lists mapping into the reference matrix, without owning the data of the reference matrix itself.

Memory ownership:

- `rows_list` and `cols_list` are heap-allocated by minor allocators and must be freed with `mat2D_minor_free`.
- The underlying matrix data (`ref_mat.elements`) is not owned by the minor and must not be freed by the minor functions.

Definition at line 119 of file [Matrix2D.h](#).

3.7.2 Member Data Documentation

3.7.2.1 cols

```
size_t Mat2D_Minor::cols
```

Definition at line 121 of file [Matrix2D.h](#).

Referenced by [mat2D_det_2x2_mat_minor\(\)](#), [mat2D_minor_alloc_fill_from_mat\(\)](#), [mat2D_minor_alloc_fill_from_mat_minor\(\)](#), [mat2D_minor_det\(\)](#), and [mat2D_minor_print\(\)](#).

3.7.2.2 cols_list

```
size_t* Mat2D_Minor::cols_list
```

Definition at line 124 of file [Matrix2D.h](#).

Referenced by [mat2D_minor_alloc_fill_from_mat\(\)](#), [mat2D_minor_alloc_fill_from_mat_minor\(\)](#), and [mat2D_minor_free\(\)](#).

3.7.2.3 ref_mat

```
Mat2D Mat2D_Minor::ref_mat
```

Definition at line 125 of file [Matrix2D.h](#).

Referenced by [mat2D_minor_alloc_fill_from_mat\(\)](#), and [mat2D_minor_alloc_fill_from_mat_minor\(\)](#).

3.7.2.4 rows

```
size_t Mat2D_Minor::rows
```

Definition at line 120 of file [Matrix2D.h](#).

Referenced by [mat2D_det_2x2_mat_minor\(\)](#), [mat2D_minor_alloc_fill_from_mat\(\)](#), [mat2D_minor_alloc_fill_from_mat_minor\(\)](#), [mat2D_minor_det\(\)](#), and [mat2D_minor_print\(\)](#).

3.7.2.5 rows_list

```
size_t* Mat2D_Minor::rows_list
```

Definition at line 123 of file [Matrix2D.h](#).

Referenced by [mat2D_minor_alloc_fill_from_mat\(\)](#), [mat2D_minor_alloc_fill_from_mat_minor\(\)](#), and [mat2D_minor_free\(\)](#).

3.7.2.6 stride_r

```
size_t Mat2D_Minor::stride_r
```

Definition at line 122 of file [Matrix2D.h](#).

Referenced by [mat2D_minor_alloc_fill_from_mat\(\)](#), and [mat2D_minor_alloc_fill_from_mat_minor\(\)](#).

The documentation for this struct was generated from the following file:

- [Matrix2D.h](#)

3.8 Mat2D_uint32 Struct Reference

Dense row-major matrix of uint32_t.

```
#include <Matrix2D.h>
```

Public Attributes

- `size_t rows`
- `size_t cols`
- `size_t stride_r`
- `uint32_t * elements`

3.8.1 Detailed Description

Dense row-major matrix of `uint32_t`.

- `rows`: number of rows (height)
- `cols`: number of columns (width)
- `stride_r`: number of elements between successive rows in memory (for contiguous storage, `stride_r == cols`)
- `elements`: pointer to contiguous storage of size `rows * cols`

Definition at line 98 of file [Matrix2D.h](#).

3.8.2 Member Data Documentation

3.8.2.1 cols

```
size_t Mat2D_uint32::cols
```

Definition at line 100 of file [Matrix2D.h](#).

Referenced by `adl_2Dscalar_interp_on_figure()`, `adl_axis_draw_on_figure()`, `adl_figure_alloc()`, `adl_figure_copy_to_screen()`, `adl_line_draw()`, `adl_point_draw()`, `adl_quad_fill()`, `adl_quad_fill_interpolate_color_mean_value()`, `adl_quad_fill_interpolate_normal_mean_value()`, `adl_tri_fill_Pinedas_rasterizer()`, `adl_tri_fill_Pinedas_rasterizer_interpolate_color()`, `adl_tri_fill_Pinedas_rasterizer_interpolate_normal_mean_value()`, `check_window_mat_size()`, `copy_mat_to_surface_RGB()`, `mat2D_alloc_uint32()`, `mat2D_fill_uint32()`, `mat2D_offset2d_uint32()`, and `render_window()`.

3.8.2.2 elements

```
uint32_t* Mat2D_uint32::elements
```

Definition at line 102 of file [Matrix2D.h](#).

Referenced by `copy_mat_to_surface_RGB()`, `mat2D_alloc_uint32()`, `mat2D_free_uint32()`, and `render_window()`.

3.8.2.3 rows

```
size_t Mat2D_uint32::rows
```

Definition at line 99 of file [Matrix2D.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_axis_draw_on_figure\(\)](#), [adl_figure_alloc\(\)](#), [adl_figure_copy_to_screen\(\)](#), [adl_line_draw\(\)](#), [adl_max_min_values_draw_on_figure\(\)](#), [adl_point_draw\(\)](#), [adl_quad_fill\(\)](#), [adl_quad_fill_interpolate_color_mean_value\(\)](#), [adl_quad_fill_interpolate_normal_mean_value\(\)](#), [adl_tri_fill_Pinedas_rasterizer\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_normal\(\)](#), [check_window_mat_size\(\)](#), [copy_mat_to_surface_RGB\(\)](#), [mat2D_alloc_uint32\(\)](#), [mat2D_fill_uint32\(\)](#), [mat2D_offset2d_uint32\(\)](#), and [render_window\(\)](#).

3.8.2.4 stride_r

```
size_t Mat2D_uint32::stride_r
```

Definition at line 101 of file [Matrix2D.h](#).

Referenced by [mat2D_alloc_uint32\(\)](#), and [mat2D_offset2d_uint32\(\)](#).

The documentation for this struct was generated from the following file:

- [Matrix2D.h](#)

3.9 Offset_zoom_param Struct Reference

Pan/zoom parameters relative to screen center.

```
#include <Almog_Draw_Library.h>
```

Public Attributes

- float [zoom_multiplier](#)
- float [offset_x](#)
- float [offset_y](#)
- int [mouse_x](#)
- int [mouse_y](#)

3.9.1 Detailed Description

Pan/zoom parameters relative to screen center.

The coordinates are shifted by (offset_x, offset_y) and scaled by zoom_multiplier about the screen center. The mouse fields are optional and can be used by UI code that updates the pan/zoom.

Definition at line 65 of file [Almog_Draw_Library.h](#).

3.9.2 Member Data Documentation

3.9.2.1 mouse_x

```
int Offset_zoom_param::mouse_x
```

Optional: last mouse x (pixels).

Definition at line 69 of file [Almog_Draw_Library.h](#).

3.9.2.2 mouse_y

```
int Offset_zoom_param::mouse_y
```

Optional: last mouse y (pixels).

Definition at line 70 of file [Almog_Draw_Library.h](#).

3.9.2.3 offset_x

```
float Offset_zoom_param::offset_x
```

Horizontal pan offset (pixels).

Definition at line 67 of file [Almog_Draw_Library.h](#).

Referenced by [adl_line_draw\(\)](#), [adl_point_draw\(\)](#), and [process_input_window\(\)](#).

3.9.2.4 offset_y

```
float Offset_zoom_param::offset_y
```

Vertical pan offset (pixels).

Definition at line 68 of file [Almog_Draw_Library.h](#).

Referenced by [adl_line_draw\(\)](#), [adl_point_draw\(\)](#), and [process_input_window\(\)](#).

3.9.2.5 zoom_multiplier

```
float Offset_zoom_param::zoom_multiplier
```

Zoom scale factor (>0).

Definition at line 66 of file [Almog_Draw_Library.h](#).

Referenced by [adl_line_draw\(\)](#), [adl_point_draw\(\)](#), [main\(\)](#), and [process_input_window\(\)](#).

The documentation for this struct was generated from the following file:

- [Almog_Draw_Library.h](#)

3.10 Point Struct Reference

Homogeneous 2D/3D point with per-vertex depth (z) and w.

```
#include <Almog_Draw_Library.h>
```

Public Attributes

- float [x](#)
- float [y](#)
- float [z](#)
- float [w](#)

3.10.1 Detailed Description

Homogeneous 2D/3D point with per-vertex depth (z) and w.

x,y are screen-space coordinates for rasterization. z,w are used for perspective-correct interpolation via inverse-Z buffering.

Definition at line 81 of file [Almog_Draw_Library.h](#).

3.10.2 Member Data Documentation

3.10.2.1 w

```
float Point::w
```

Homogeneous w.

Definition at line 85 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_cartesian_grid_create\(\)](#), [adl_quad_fill\(\)](#), [adl_quad_fill_interpolate_color_mean_w\(\)](#), [adl_quad_fill_interpolate_normal_mean_value\(\)](#), [adl_tri_fill_Pinedas_rasterizer\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), and [adl_tri_fill_Pinedas_rasterizer_interpolate_normal\(\)](#).

3.10.2.2 x

```
float Point::x
```

X coordinate (pixels).

Definition at line 82 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_cartesian_grid_create\(\)](#), [adl_curve_add_to_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_copy_to_screen\(\)](#), [adl_quad_fill\(\)](#), [adl_quad_fill_interpolate_color_mean_value\(\)](#), [adl_quad_fill_interpolate_normal_mean_value\(\)](#), [adl_tan_half_angle\(\)](#), [adl_tri_draw\(\)](#), [adl_tri_fill_Pinedas_rasterizer\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), and [adl_tri_fill_Pinedas_rasterizer_interpolate_normal\(\)](#).

3.10.2.3 y

```
float Point::y
```

Y coordinate (pixels).

Definition at line 83 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_cartesian_grid_create\(\)](#), [adl_curve_add_to_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_figure_copy_to_screen\(\)](#), [adl_quad_fill\(\)](#), [adl_quad_fill_interpolate_color_mean_value\(\)](#), [adl_quad_fill_interpolate_normal_mean_value\(\)](#), [adl_tan_half_angle\(\)](#), [adl_tri_draw\(\)](#), [adl_tri_fill_Pinedas_rasterizer\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), and [adl_tri_fill_Pinedas_rasterizer_interpolate_normal\(\)](#).

3.10.2.4 z

```
float Point::z
```

Depth value.

Definition at line 84 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_cartesian_grid_create\(\)](#), [adl_quad_fill\(\)](#), [adl_quad_fill_interpolate_color_mean_value\(\)](#), [adl_quad_fill_interpolate_normal_mean_value\(\)](#), [adl_tri_fill_Pinedas_rasterizer\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), and [adl_tri_fill_Pinedas_rasterizer_interpolate_normal\(\)](#).

The documentation for this struct was generated from the following file:

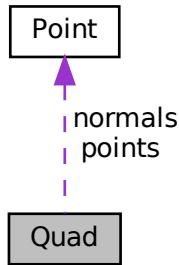
- [Almog_Draw_Library.h](#)

3.11 Quad Struct Reference

[Quad](#) primitive with optional per-vertex attributes.

```
#include <Almog_Draw_Library.h>
```

Collaboration diagram for Quad:



Public Attributes

- [Point points](#) [4]
- [Point normals](#) [4]
- [uint32_t colors](#) [4]
- [bool to_draw](#)
- [float light_intensity](#) [4]

3.11.1 Detailed Description

[Quad](#) primitive with optional per-vertex attributes.

Definition at line 134 of file [Almog_Draw_Library.h](#).

3.11.2 Member Data Documentation

3.11.2.1 colors

```
uint32_t Quad::colors[4]
```

Optional per-vertex ARGB colors.

Definition at line 137 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_quad2tris\(\)](#), [adl_quad_fill_interpolate_color_mean_value\(\)](#), and [setup\(\)](#).

3.11.2.2 light_intensity

```
float Quad::light_intensity[4]
```

Per-vertex light intensity multiplier.

Definition at line 139 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_quad2tris\(\)](#), [adl_quad_fill\(\)](#), [adl_quad_fill_interpolate_color_mean_value\(\)](#), [adl_quad_fill_interpolate_normal_mean_value\(\)](#), and [setup\(\)](#).

3.11.2.3 normals

```
Point Quad::normals[4]
```

Optional normals (unused here).

Definition at line 136 of file [Almog_Draw_Library.h](#).

3.11.2.4 points

```
Point Quad::points[4]
```

[Quad](#) vertices (0..3 order).

Definition at line 135 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_quad2tris\(\)](#), [adl_quad_draw\(\)](#), [adl_quad_fill\(\)](#), [adl_quad_fill_interpolate_color_mean_value\(\)](#), [adl_quad_fill_interpolate_normal_mean_value\(\)](#), and [setup\(\)](#).

3.11.2.5 to_draw

```
bool Quad::to_draw
```

Whether to include in rendering.

Definition at line 138 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_quad2tris\(\)](#), [adl_quad_mesh_draw\(\)](#), [adl_quad_mesh_fill\(\)](#), [adl_quad_mesh_fill_interpolate_color\(\)](#), [adl_quad_mesh_fill_interpolate_normal\(\)](#), and [setup\(\)](#).

The documentation for this struct was generated from the following file:

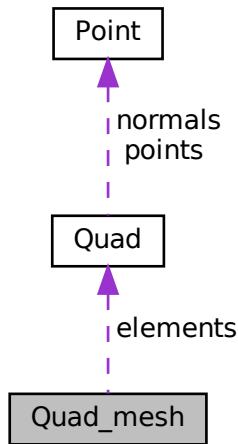
- [Almog_Draw_Library.h](#)

3.12 Quad_mesh Struct Reference

Dynamic array of quads (quad mesh).

```
#include <Almog_Draw_Library.h>
```

Collaboration diagram for Quad_mesh:



Public Attributes

- size_t `length`
- size_t `capacity`
- `Quad * elements`

3.12.1 Detailed Description

Dynamic array of quads (quad mesh).

Definition at line 160 of file [Almog_Draw_Library.h](#).

3.12.2 Member Data Documentation

3.12.2.1 capacity

```
size_t Quad_mesh::capacity
```

Allocated capacity.

Definition at line 162 of file [Almog_Draw_Library.h](#).

3.12.2.2 elements

```
Quad* Quad_mesh::elements
```

Quad array.

Definition at line 163 of file [Almog_Draw_Library.h](#).

Referenced by `adl_quad_mesh_draw()`, `adl_quad_mesh_fill()`, `adl_quad_mesh_fill_interpolate_color()`, and `adl_quad_mesh_fill_interpolate_normal()`.

3.12.2.3 length

```
size_t Quad_mesh::length
```

Number of quads used.

Definition at line 161 of file [Almog_Draw_Library.h](#).

Referenced by `adl_quad_mesh_draw()`, `adl_quad_mesh_fill()`, `adl_quad_mesh_fill_interpolate_color()`, and `adl_quad_mesh_fill_interpolate_normal()`.

The documentation for this struct was generated from the following file:

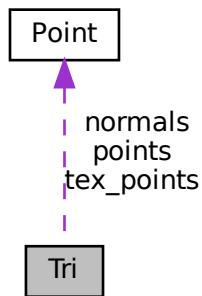
- [Almog_Draw_Library.h](#)

3.13 Tri Struct Reference

Triangle primitive with optional per-vertex attributes.

```
#include <Almog_Draw_Library.h>
```

Collaboration diagram for Tri:



Public Attributes

- `Point points` [3]
- `Point tex_points` [3]
- `Point normals` [3]
- `uint32_t colors` [3]
- `bool to_draw`
- `float light_intensity` [3]

3.13.1 Detailed Description

Triangle primitive with optional per-vertex attributes.

Definition at line 119 of file [Almog_Draw_Library.h](#).

3.13.2 Member Data Documentation

3.13.2.1 colors

```
uint32_t Tri::colors[3]
```

Optional per-vertex ARGB colors.

Definition at line 123 of file [Almog_Draw_Library.h](#).

Referenced by [adl_quad2tris\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), and [setup\(\)](#).

3.13.2.2 light_intensity

```
float Tri::light_intensity[3]
```

Per-vertex light intensity multiplier.

Definition at line 125 of file [Almog_Draw_Library.h](#).

Referenced by [adl_quad2tris\(\)](#), [adl_tri_fill_Pinedas_rasterizer\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_normal\(\)](#), and [setup\(\)](#).

3.13.2.3 normals

```
Point Tri::normals[3]
```

Optional normals (unused here).

Definition at line 122 of file [Almog_Draw_Library.h](#).

3.13.2.4 points

```
Point Tri::points[3]
```

Triangle vertices.

Definition at line 120 of file [Almog_Draw_Library.h](#).

Referenced by [adl_quad2tris\(\)](#), [adl_tri_draw\(\)](#), [adl_tri_fill_Pinedas_rasterizer\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_normal\(\)](#), and [setup\(\)](#).

3.13.2.5 tex_points

```
Point Tri::tex_points[3]
```

Optional texture coordinates (unused here).

Definition at line 121 of file [Almog_Draw_Library.h](#).

3.13.2.6 to_draw

```
bool Tri::to_draw
```

Whether to include in rendering.

Definition at line 124 of file [Almog_Draw_Library.h](#).

Referenced by [adl_quad2tris\(\)](#), [adl_tri_mesh_draw\(\)](#), [adl_tri_mesh_fill_Pinedas_rasterizer\(\)](#), [adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color\(\)](#), [adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal\(\)](#), and [setup\(\)](#).

The documentation for this struct was generated from the following file:

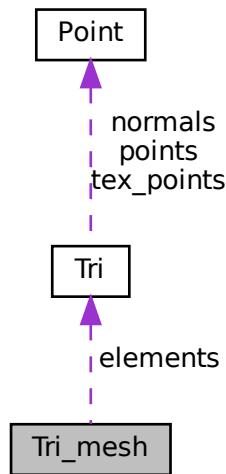
- [Almog_Draw_Library.h](#)

3.14 Tri_mesh Struct Reference

Dynamic array of triangles (triangle mesh).

```
#include <Almog_Draw_Library.h>
```

Collaboration diagram for Tri_mesh:



Public Attributes

- size_t `length`
- size_t `capacity`
- `Tri *` `elements`

3.14.1 Detailed Description

Dynamic array of triangles (triangle mesh).

Definition at line 148 of file [Almog_Draw_Library.h](#).

3.14.2 Member Data Documentation

3.14.2.1 capacity

```
size_t Tri_mesh::capacity
```

Allocated capacity.

Definition at line 150 of file [Almog_Draw_Library.h](#).

3.14.2.2 elements

```
Tri* Tri_mesh::elements
```

Triangle array.

Definition at line 151 of file [Almog_Draw_Library.h](#).

Referenced by [adl_tri_mesh_draw\(\)](#), [adl_tri_mesh_fill_Pinedas_rasterizer\(\)](#), [adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color\(\)](#), and [adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal\(\)](#).

3.14.2.3 length

```
size_t Tri_mesh::length
```

Number of triangles used.

Definition at line 149 of file [Almog_Draw_Library.h](#).

Referenced by [adl_tri_mesh_draw\(\)](#), [adl_tri_mesh_fill_Pinedas_rasterizer\(\)](#), [adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color\(\)](#), and [adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal\(\)](#).

The documentation for this struct was generated from the following file:

- [Almog_Draw_Library.h](#)

Chapter 4

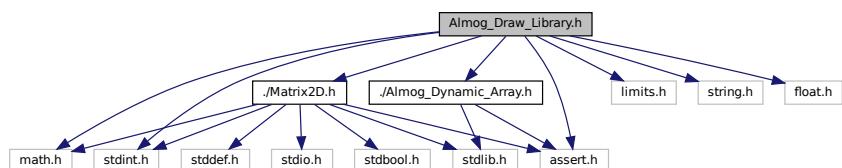
File Documentation

4.1 Almog_Draw_Library.h File Reference

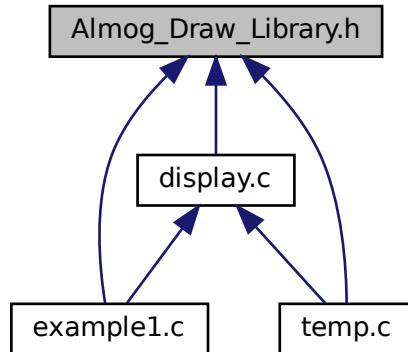
Immediate-mode 2D/3D raster helpers for drawing onto [Mat2D_uint32](#) pixel buffers.

```
#include <math.h>
#include <stdint.h>
#include <limits.h>
#include <string.h>
#include <float.h>
#include "./Matrix2D.h"
#include "./Almog_Dynamic_Array.h"
#include <assert.h>
```

Include dependency graph for Almog_Draw_Library.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [Offset_zoom_param](#)
Pan/zoom parameters relative to screen center.
- struct [Point](#)
Homogeneous 2D/3D point with per-vertex depth (z) and w.
- struct [Curve](#)
Polyline of points with a uniform color.
- struct [Curve_ada](#)
Dynamic array of curves (polyline container).
- struct [Tri](#)
Triangle primitive with optional per-vertex attributes.
- struct [Quad](#)
Quad primitive with optional per-vertex attributes.
- struct [Tri_mesh](#)
Dynamic array of triangles (triangle mesh).
- struct [Quad_mesh](#)
Dynamic array of quads (quad mesh).
- struct [Figure](#)
Plotting figure holding a pixel buffer, z-buffer and plot state.
- struct [Grid](#)
Grid definition (as lines) in a chosen plane.

Macros

- #define [ADL_ASSERT](#) assert
Assertion macro used by this header (defaults to assert).
- #define [POINT](#)
- #define [CURVE](#)
- #define [CURVE_ADA](#)

- #define TRI
- #define QUAD
- #define TRI_MESH
- #define QUAD_MESH
- #define HexARGB_RGBAlpha(x) ((x)>>(8*2)&0xFF), ((x)>>(8*1)&0xFF), ((x)>>(8*0)&0xFF), ((x)>>(8*3)&0xFF)
- #define HexARGB_RGB_VAR(x, r, g, b) r = ((x)>>(8*2)&0xFF); g = ((x)>>(8*1)&0xFF); b = ((x)>>(8*0)&0xFF);
- #define HexARGB_RGBA_VAR(x, r, g, b, a) r = ((x)>>(8*2)&0xFF); g = ((x)>>(8*1)&0xFF); b = ((x)>>(8*0)&0xFF); a = ((x)>>(8*3)&0xFF)
- #define RGB_hexRGB(r, g, b) (int)(0x010000*(int)(r) + 0x000100*(int)(g) + 0x000001*(int)(b))
- #define RGBA_hexARGB(r, g, b, a) (int)(0x01000000*(int)(fmin(a, 255)) + 0x010000*(int)(r) + 0x000100*(int)(g) + 0x000001*(int)(b))
- #define RED_hexARGB 0xFFFF0000
- #define GREEN_hexARGB 0xFF00FF00
- #define BLUE_hexARGB 0xFF0000FF
- #define PURPLE_hexARGB 0xFFFF00FF
- #define CYAN_hexARGB 0xFF00FFFF
- #define YELLOW_hexARGB 0xFFFFFFFF00
- #define edge_cross_point(a1, b, a2, p) (b.x-a1.x)*(p.y-a2.y)-(b.y-a1.y)*(p.x-a2.x)
- #define is_top_edge(x, y) (y == 0 && x > 0)
- #define is_left_edge(x, y) (y < 0)
- #define is_top_left(ps, pe) (is_top_edge(pe.x-ps.x, pe.y-ps.y) || is_left_edge(pe.x-ps.x, pe.y-ps.y))
- #define ADL_MAX_POINT_VAL 1e5
- #define adl_assert_point_is_valid(p) ADL_ASSERT(isfinite(p.x) && isfinite(p.y) && isfinite(p.z) && isfinite(p.w))
- #define adl_assert_tri_is_valid(tri)
- #define adl_assert_quad_is_valid(quad)
- #define ADL FIGURE_PADDING_PERCENTAGE 20
- #define ADL_MAX FIGURE_PADDING 70
- #define ADL_MIN FIGURE_PADDING 20
- #define ADL_MAX HEAD_SIZE 15
- #define ADL FIGURE_HEAD_ANGLE_DEG 30
- #define ADL FIGURE_AXIS_COLOR 0xff000000
- #define ADL_MAX_CHARACTER_OFFSET 10
- #define ADL_MIN_CHARACTER_OFFSET 5
- #define ADL_MAX_SENTENCE_LEN 256
- #define ADL_MAX_ZOOM 1e3
- #define ADL_DEFAULT_OFFSET_ZOOM (Offset_zoom_param){1,0,0,0,0}
- #define adl_offset_zoom_point(p, window_w, window_h, offset_zoom_param)
- #define adl_offset2d(i, j, ni) (j) * (ni) + (i)

Functions

- void **adl_point_draw** (Mat2D_uint32 screen_mat, int x, int y, uint32_t color, Offset_zoom_param offset_zoom_param)

Draw a single pixel with alpha blending.
- void **adl_line_draw** (Mat2D_uint32 screen_mat, const float x1_input, const float y1_input, const float x2_input, const float y2_input, uint32_t color, Offset_zoom_param offset_zoom_param)

Draw an anti-aliased-like line by vertical spans (integer grid).
- void **adl_lines_draw** (const Mat2D_uint32 screen_mat, const Point *points, const size_t len, const uint32_t color, Offset_zoom_param offset_zoom_param)

Draw a polyline connecting an array of points.

- void `adl_lines_loop_draw` (const `Mat2D_uint32` screen_mat, const `Point *points`, const `size_t len`, const `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Draw a closed polyline (loop).
- void `adl_arrow_draw` (`Mat2D_uint32` screen_mat, int xs, int ys, int xe, int ye, float head_size, float angle_deg, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Draw an arrow from start to end with a triangular head.
- void `adl_character_draw` (`Mat2D_uint32` screen_mat, char c, int width_pixel, int height_pixel, int x_top_left, int y_top_left, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Draw a vector glyph for a single ASCII character.
- void `adl_sentence_draw` (`Mat2D_uint32` screen_mat, const char sentence[], `size_t len`, const int x_top_left, const int y_top_left, const int height_pixel, const `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Draw a horizontal sentence using vector glyphs.
- void `adl_rectangle_draw_min_max` (`Mat2D_uint32` screen_mat, int min_x, int max_x, int min_y, int max_y, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Draw a rectangle outline defined by min/max corners (inclusive).
- void `adl_rectangle_fill_min_max` (`Mat2D_uint32` screen_mat, int min_x, int max_x, int min_y, int max_y, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Fill a rectangle defined by min/max corners (inclusive).
- void `adl_quad_draw` (`Mat2D_uint32` screen_mat, `Mat2D inv_z_buffer`, `Quad quad`, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Draw the outline of a quad (four points, looped).
- void `adl_quad_fill` (`Mat2D_uint32` screen_mat, `Mat2D inv_z_buffer`, `Quad quad`, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Fill a quad using mean-value (Barycentric) coordinates and flat base color.
- void `adl_quad_fill_interpolate_normal_mean_value` (`Mat2D_uint32` screen_mat, `Mat2D inv_z_buffer`, `Quad quad`, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Fill a quad with per-pixel light interpolation (mean value coords).
- void `adl_quad_fill_interpolate_color_mean_value` (`Mat2D_uint32` screen_mat, `Mat2D inv_z_buffer`, `Quad quad`, `Offset_zoom_param offset_zoom_param`)

Fill a quad with per-vertex colors (mean value coords).
- void `adl_quad_mesh_draw` (`Mat2D_uint32` screen_mat, `Mat2D inv_z_buffer_mat`, `Quad_mesh mesh`, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Draw outlines for all quads in a mesh.
- void `adl_quad_mesh_fill` (`Mat2D_uint32` screen_mat, `Mat2D inv_z_buffer_mat`, `Quad_mesh mesh`, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Fill all quads in a mesh with a uniform base color.
- void `adl_quad_mesh_fill_interpolate_normal` (`Mat2D_uint32` screen_mat, `Mat2D inv_z_buffer_mat`, `Quad_mesh mesh`, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Fill all quads in a mesh using interpolated lighting.
- void `adl_quad_mesh_fill_interpolate_color` (`Mat2D_uint32` screen_mat, `Mat2D inv_z_buffer_mat`, `Quad_mesh mesh`, `Offset_zoom_param offset_zoom_param`)

Fill all quads in a mesh using per-vertex colors.
- void `adl_circle_draw` (`Mat2D_uint32` screen_mat, float center_x, float center_y, float r, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Draw an approximate circle outline (1px thickness).
- void `adl_circle_fill` (`Mat2D_uint32` screen_mat, float center_x, float center_y, float r, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Fill a circle.
- void `adl_tri_draw` (`Mat2D_uint32` screen_mat, `Tri tri`, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Draw the outline of a triangle.
- void `adl_tri_fill_Pinedas_rasterizer` (`Mat2D_uint32` screen_mat, `Mat2D inv_z_buffer`, `Tri tri`, `uint32_t color`, `Offset_zoom_param offset_zoom_param`)

Fill a triangle using Pineda's rasterizer with flat base color.

- void `adl_tri_fill_Pinedas_rasterizer_interpolate_color` (`Mat2D_uint32` screen_mat, `Mat2D` inv_z_buffer, `Tri` tri, `Offset_zoom_param` offset_zoom_param)

Fill a triangle using Pineda's rasterizer with per-vertex colors.
- void `adl_tri_fill_Pinedas_rasterizer_interpolate_normal` (`Mat2D_uint32` screen_mat, `Mat2D` inv_z_buffer, `Tri` tri, `uint32_t` color, `Offset_zoom_param` offset_zoom_param)

Fill a triangle with interpolated lighting over a uniform color.
- void `adl_tri_mesh_draw` (`Mat2D_uint32` screen_mat, `Tri_mesh` mesh, `uint32_t` color, `Offset_zoom_param` offset_zoom_param)

Draw outlines for all triangles in a mesh.
- void `adl_tri_mesh_fill_Pinedas_rasterizer` (`Mat2D_uint32` screen_mat, `Mat2D` inv_z_buffer_mat, `Tri_mesh` mesh, `uint32_t` color, `Offset_zoom_param` offset_zoom_param)

Fill all triangles in a mesh with a uniform base color.
- void `adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color` (`Mat2D_uint32` screen_mat, `Mat2D` inv_z_buffer_mat, `Tri_mesh` mesh, `Offset_zoom_param` offset_zoom_param)

Fill all triangles in a mesh with a uniform base color.
- void `adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal` (`Mat2D_uint32` screen_mat, `Mat2D` inv_z_buffer_mat, `Tri_mesh` mesh, `uint32_t` color, `Offset_zoom_param` offset_zoom_param)

Fill all triangles in a mesh with interpolated lighting.
- float `adl_tan_half_angle` (`Point` vi, `Point` vj, `Point` p, float li, float lj)

Compute tan(alpha/2) for the angle at point p between segments p->vi and p->vj.
- float `adl_linear_map` (float s, float min_in, float max_in, float min_out, float max_out)

Affine map from one scalar range to another (no clamping).
- void `adl_quad2tris` (`Quad` quad, `Tri`*tri1, `Tri`*tri2, char split_line[])

Split a quad into two triangles along a chosen diagonal.
- void `adl_linear_sRGB_to_okLab` (`uint32_t` hex_ARGB, float *L, float *a, float *b)

Convert a linear sRGB color (ARGB) to Oklab components.
- void `adl_okLab_to_linear_sRGB` (float L, float a, float b, `uint32_t`*hex_ARGB)

Convert Oklab components to a linear sRGB ARGB color.
- void `adl_linear_sRGB_to_okLch` (`uint32_t` hex_ARGB, float *L, float *c, float *h_deg)

Convert a linear sRGB color (ARGB) to OkLch components.
- void `adl_okLch_to_linear_sRGB` (float L, float c, float h_deg, `uint32_t`*hex_ARGB)

Convert OkLch components to a linear sRGB ARGB color.
- void `adl_interpolate_ARGBcolor_on_okLch` (`uint32_t` color1, `uint32_t` color2, float t, float num_of_rotations, `uint32_t`*color_out)

Interpolate between two ARGB colors in OkLch space.
- `Figure adl_figure_alloc` (`size_t` rows, `size_t` cols, `Point` top_left_position)

Allocate and initialize a `Figure` with an internal pixel buffer.
- void `adl_figure_copy_to_screen` (`Mat2D_uint32` screen_mat, `Figure` figure)

Blit a `Figure`'s pixels onto a destination screen buffer.
- void `adl_axis_draw_on_figure` (`Figure`*figure)

Draw X/Y axes with arrowheads into a `Figure`.
- void `adl_max_min_values_draw_on_figure` (`Figure` figure)

Draw min/max numeric labels for the current data range.
- void `adl_curve_add_to_figure` (`Figure`*figure, `Point`*src_points, `size_t` src_len, `uint32_t` color)

Add a curve (polyline) to a `Figure` and update its data bounds.
- void `adl_curves_plot_on_figure` (`Figure` figure)

Render all added curves into a `Figure`'s pixel buffer.
- void `adl_2Dscalar_interp_on_figure` (`Figure` figure, double*x_2Dmat, double*y_2Dmat, double*scalar_2Dmat, int ni, int nj, char color_scale[], float num_of_rotations)

Visualize a scalar field on a `Figure` by colored quads.
- `Grid adl_cartesian_grid_create` (float min_e1, float max_e1, float min_e2, float max_e2, int num_samples_e1, int num_samples_e2, char plane[], float third_direction_position)

Create a 2D grid for a Cartesian coordinate system.

Create a Cartesian grid (as curves) on one of the principal planes.

- void `adl_grid_draw (Mat2D_uint32 screen_mat, Grid grid, uint32_t color, Offset_zoom_param offset_zoom_param)`

Draw a previously created `Grid` as line segments.

4.1.1 Detailed Description

Immediate-mode 2D/3D raster helpers for drawing onto `Mat2D_uint32` pixel buffers.

This single-header library provides a minimal software rasterizer for drawing into a 32-bit ARGB pixel buffer (`Mat2D_uint32`). It supports:

- Points, lines, circles, triangles and quads (wire and filled)
- Z-buffered triangle/quad rasterization (inverse-Z convention)
- Per-vertex color and simple light-intensity interpolation
- Basic vector-text drawing (ASCII subset)
- Plotting helper types ([Figure](#)) and utilities for curve plots and 2D scalar-field visualization using perceptual color interpolation in the OKLab/OKLch color spaces
- Cartesian grid generation in common planes

All draw calls may accept an `Offset_zoom_param` that enables simple pan/zoom behavior around the screen center.

Types `Mat2D` and `Mat2D_uint32` are provided by [Matrix2D.h](#).

Usage:

- Include this header wherever you use the API.
- In exactly one translation unit (source file) define `ALMOG_DRAW_LIBRARY_IMPLEMENTATION` before including this header to compile the function definitions.

Note

- Colors are ARGB in 0xAARRGGBB packed 32-bit format.
- Z buffering uses an inverse-Z buffer (bigger is closer).
- The OKLab/OKLch conversions here assume linear sRGB channels.

Definition in file [Almog_Draw_Library.h](#).

4.1.2 Macro Definition Documentation

4.1.2.1 ADL_ASSERT

```
#define ADL_ASSERT assert
```

Assertion macro used by this header (defaults to assert).

Define ADL_ASSERT before including this file to override. When NDEBUG is defined, standard assert() is disabled.

Definition at line 55 of file [Almog_Draw_Library.h](#).

4.1.2.2 adl_assert_point_is_valid

```
#define adl_assert_point_is_valid(  
    p ) ADL_ASSERT(isfinite(p.x) && isfinite(p.y) && isfinite(p.z) && isfinite(p.w))
```

Definition at line 310 of file [Almog_Draw_Library.h](#).

4.1.2.3 adl_assert_quad_is_valid

```
#define adl_assert_quad_is_valid(  
    quad )
```

Value:

```
adl_assert_point_is_valid(quad.points[0]);  \  
adl_assert_point_is_valid(quad.points[1]);  \  
adl_assert_point_is_valid(quad.points[2]);  \  
adl_assert_point_is_valid(quad.points[3])  \  
\\
```

Definition at line 314 of file [Almog_Draw_Library.h](#).

4.1.2.4 adl_assert_tri_is_valid

```
#define adl_assert_tri_is_valid(  
    tri )
```

Value:

```
adl_assert_point_is_valid(tri.points[0]); \  
adl_assert_point_is_valid(tri.points[1]); \  
adl_assert_point_is_valid(tri.points[2])  \  
\\
```

Definition at line 311 of file [Almog_Draw_Library.h](#).

4.1.2.5 ADL_DEFAULT_OFFSET_ZOOM

```
#define ADL_DEFAULT_OFFSET_ZOOM (Offset_zoom_param) {1,0,0,0,0}
```

Definition at line 331 of file [Almog_Draw_Library.h](#).

4.1.2.6 ADL FIGURE AXIS COLOR

```
#define ADL FIGURE AXIS COLOR 0xff000000
```

Definition at line 324 of file [Almog_Draw_Library.h](#).

4.1.2.7 ADL FIGURE HEAD ANGLE DEG

```
#define ADL FIGURE HEAD ANGLE DEG 30
```

Definition at line 323 of file [Almog_Draw_Library.h](#).

4.1.2.8 ADL FIGURE PADDING PRECENTAGE

```
#define ADL FIGURE PADDING PRECENTAGE 20
```

Definition at line 319 of file [Almog_Draw_Library.h](#).

4.1.2.9 ADL MAX CHARACTER OFFSET

```
#define ADL MAX CHARACTER OFFSET 10
```

Definition at line 326 of file [Almog_Draw_Library.h](#).

4.1.2.10 ADL MAX FIGURE PADDING

```
#define ADL MAX FIGURE PADDING 70
```

Definition at line 320 of file [Almog_Draw_Library.h](#).

4.1.2.11 ADL_MAX_HEAD_SIZE

```
#define ADL_MAX_HEAD_SIZE 15
```

Definition at line 322 of file [Almog_Draw_Library.h](#).

4.1.2.12 ADL_MAX_POINT_VAL

```
#define ADL_MAX_POINT_VAL 1e5
```

Definition at line 309 of file [Almog_Draw_Library.h](#).

4.1.2.13 ADL_MAX_SENTENCE_LEN

```
#define ADL_MAX_SENTENCE_LEN 256
```

Definition at line 328 of file [Almog_Draw_Library.h](#).

4.1.2.14 ADL_MAX_ZOOM

```
#define ADL_MAX_ZOOM 1e3
```

Definition at line 329 of file [Almog_Draw_Library.h](#).

4.1.2.15 ADL_MIN_CHARACTER_OFFSET

```
#define ADL_MIN_CHARACTER_OFFSET 5
```

Definition at line 327 of file [Almog_Draw_Library.h](#).

4.1.2.16 ADL_MIN FIGURE_PADDING

```
#define ADL_MIN FIGURE_PADDING 20
```

Definition at line 321 of file [Almog_Draw_Library.h](#).

4.1.2.17 adl_offset2d

```
#define adl_offset2d(  
    i,  
    j,  
    ni ) (j) * (ni) + (i)
```

Definition at line 2294 of file [Almog_Draw_Library.h](#).

4.1.2.18 adl_offset_zoom_point

```
#define adl_offset_zoom_point(  
    p,  
    window_w,  
    window_h,  
    offset_zoom_param )
```

Value:

```
(p).x = ((p).x - (window_w)/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier +  
        (window_w)/2; \  
(p).y = ((p).y - (window_h)/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier +  
        (window_h)/2
```

Definition at line 332 of file [Almog_Draw_Library.h](#).

4.1.2.19 BLUE_hexARGB

```
#define BLUE_hexARGB 0xFF0000FF
```

Definition at line 299 of file [Almog_Draw_Library.h](#).

4.1.2.20 CURVE

```
#define CURVE
```

Definition at line 90 of file [Almog_Draw_Library.h](#).

4.1.2.21 CURVE_ADA

```
#define CURVE_ADA
```

Definition at line 103 of file [Almog_Draw_Library.h](#).

4.1.2.22 CYAN_hexARGB

```
#define CYAN_hexARGB 0xFF00FFFF
```

Definition at line 301 of file [Almog_Draw_Library.h](#).

4.1.2.23 edge_cross_point

```
#define edge_cross_point(
    a1,
    b,
    a2,
    p ) (b.x-a1.x)*(p.y-a2.y)-(b.y-a1.y)*(p.x-a2.x)
```

Definition at line 304 of file [Almog_Draw_Library.h](#).

4.1.2.24 GREEN_hexARGB

```
#define GREEN_hexARGB 0xFF00FF00
```

Definition at line 298 of file [Almog_Draw_Library.h](#).

4.1.2.25 HexARGB_RGB_VAR

```
#define HexARGB_RGB_VAR (
    x,
    r,
    g,
    b ) r = ((x)>>(8*2)&0xFF); g = ((x)>>(8*1)&0xFF); b = ((x)>>(8*0)&0xFF);
```

Definition at line 224 of file [Almog_Draw_Library.h](#).

4.1.2.26 HexARGB_RGBA

```
#define HexARGB_RGBA (
    x ) ((x)>>(8*2)&0xFF), ((x)>>(8*1)&0xFF), ((x)>>(8*0)&0xFF), ((x)>>(8*3)&0xFF)
```

Definition at line 221 of file [Almog_Draw_Library.h](#).

4.1.2.27 HexARGB_RGB4_VAR

```
#define HexARGB_RGB4_VAR(  
    x,  
    r,  
    g,  
    b,  
    a )  r = ((x)>>(8*2)&0xFF); g = ((x)>>(8*1)&0xFF); b = ((x)>>(8*0)&0xFF); a =  
((x)>>(8*3)&0xFF)
```

Definition at line 227 of file [Almog_Draw_Library.h](#).

4.1.2.28 is_left_edge

```
#define is_left_edge(  
    x,  
    y ) (y < 0)
```

Definition at line 306 of file [Almog_Draw_Library.h](#).

4.1.2.29 is_top_edge

```
#define is_top_edge(  
    x,  
    y ) (y == 0 && x > 0)
```

Definition at line 305 of file [Almog_Draw_Library.h](#).

4.1.2.30 is_top_left

```
#define is_top_left(  
    ps,  
    pe ) (is_top_edge(pe.x-ps.x, pe.y-ps.y) || is_left_edge(pe.x-ps.x, pe.y-ps.y))
```

Definition at line 307 of file [Almog_Draw_Library.h](#).

4.1.2.31 POINT

```
#define POINT
```

Definition at line 74 of file [Almog_Draw_Library.h](#).

4.1.2.32 PURPLE_hexARGB

```
#define PURPLE_hexARGB 0xFFFF00FF
```

Definition at line 300 of file [Almog_Draw_Library.h](#).

4.1.2.33 QUAD

```
#define QUAD
```

Definition at line 130 of file [Almog_Draw_Library.h](#).

4.1.2.34 QUAD_MESH

```
#define QUAD_MESH
```

Definition at line 156 of file [Almog_Draw_Library.h](#).

4.1.2.35 RED_hexARGB

```
#define RED_hexARGB 0xFFFF0000
```

Definition at line 297 of file [Almog_Draw_Library.h](#).

4.1.2.36 RGB_hexRGB

```
#define RGB_hexRGB (  
    r,  
    g,  
    b ) (int) (0x010000*(int)(r) + 0x000100*(int)(g) + 0x000001*(int)(b))
```

Definition at line 230 of file [Almog_Draw_Library.h](#).

4.1.2.37 RGBA_hexARGB

```
#define RGBA_hexARGB (  
    r,  
    g,  
    b,  
    a ) (int) (0x010000001*(int) (fminf(a, 255)) + 0x010000*(int)(r) + 0x000100*(int)(g)  
+ 0x000001*(int)(b))
```

Definition at line 233 of file [Almog_Draw_Library.h](#).

4.1.2.38 TRI

```
#define TRI
```

Definition at line 115 of file [Almog_Draw_Library.h](#).

4.1.2.39 TRI_MESH

```
#define TRI_MESH
```

Definition at line 144 of file [Almog_Draw_Library.h](#).

4.1.2.40 YELLOW_hexARGB

```
#define YELLOW_hexARGB 0xFFFFFFFF00
```

Definition at line 302 of file [Almog_Draw_Library.h](#).

4.1.3 Function Documentation

4.1.3.1 adl_2Dscalar_interp_on_figure()

```
void adl_2Dscalar_interp_on_figure (
    Figure figure,
    double * x_2Dmat,
    double * y_2Dmat,
    double * scalar_2Dmat,
    int ni,
    int nj,
    char color_scale[],
    float num_of_rotations )
```

Visualize a scalar field on a [Figure](#) by colored quads.

Treats `x_2Dmat` and `y_2Dmat` as a structured 2D grid of positions (column-major with stride `ni`) and colors each cell using `scalar_2Dmat` mapped through a two-color OkLch gradient. Also updates figure bounds from the provided data. Depth-tested inside the figure's buffers.

Parameters

<code>figure</code>	Figure to render into (uses its own pixel buffers).	
<code>x_2Dmat</code>	Grid X coordinates, size <code>ni*nj</code> .	
<code>y_2Dmat</code>	Grid Y coordinates, size <code>ni*nj</code> .	
<code>scalar_2Dmat</code>	Scalar values per grid node, size <code>ni*nj</code> .	
<code>ni</code>	Number of samples along the first index (rows).	Generated by Doxygen
<code>nj</code>	Number of samples along the second index (cols).	
<code>color_scale</code>	Two-letter code of endpoints ("b-c", "b-g", "b-r", "b-y", "g-y", "g-p", "g-r", "r-y").	
<code>num_of_rotations</code>	Hue turns for the OkLch interpolation (can be fractional/negative).	

Definition at line 2314 of file [Almog_Draw_Library.h](#).

References [adl_axis_draw_on_figure\(\)](#), [ADL_DEFAULT_OFFSET_ZOOM](#), [adl_interpolate_ARGBcolor_on_okLch\(\)](#), [adl_linear_map\(\)](#), [adl_max_min_values_draw_on_figure\(\)](#), [adl_offset2d](#), [adl_offset_zoom_point](#), [adl_quad_fill_interpolate_color_mean\(\)](#), [Figure::background_color](#), [BLUE_hexARGB](#), [Quad::colors](#), [Mat2D::cols](#), [Mat2D_uint32::cols](#), [CYAN_hexARGB](#), [Mat2D::elements](#), [GREEN_hexARGB](#), [Figure::inv_z_buffer_mat](#), [Quad::light_intensity](#), [mat2D_fill_uint32\(\)](#), [Figure::max_x](#), [Figure::max_x_pixel](#), [Figure::max_y](#), [Figure::max_y_pixel](#), [Figure::min_x](#), [Figure::min_x_pixel](#), [Figure::min_y](#), [Figure::min_y_pixel](#), [Figure::offset_zoom_param](#), [Figure::pixels_mat](#), [Quad::points](#), [PURPLE_hexARGB](#), [RED_hexARGB](#), [Mat2D::rows](#), [Mat2D_uint32::rows](#), [Quad::to_draw](#), [Figure::to_draw_axis](#), [Figure::to_draw_max_min_values](#), [Point::w](#), [Point::x](#), [Point::y](#), [YELLOW_hexARGB](#), and [Point::z](#).

4.1.3.2 adl_arrow_draw()

```
void adl_arrow_draw (
    Mat2D_uint32 screen_mat,
    int xs,
    int ys,
    int xe,
    int ye,
    float head_size,
    float angle_deg,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw an arrow from start to end with a triangular head.

The head is constructed by rotating around the arrow tip by +/- angle_deg and using head_size as a fraction of the shaft length.

Note

: This function is a bit complicated and expansive but this is what I could come up with

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>xs</i>	Start X (before pan/zoom).
<i>ys</i>	Start Y (before pan/zoom).
<i>xe</i>	End X (before pan/zoom), i.e., the arrow tip.
<i>ye</i>	End Y (before pan/zoom), i.e., the arrow tip.
<i>head_size</i>	Head size as a fraction of total length in [0,1].
<i>angle_deg</i>	Head wing rotation angle in degrees.
<i>color</i>	Arrow color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 518 of file [Almog_Draw_Library.h](#).

References [adl_line_draw\(\)](#), [mat2D_add\(\)](#), [mat2D_alloc\(\)](#), [MAT2D_AT](#), [mat2D_copy\(\)](#), [mat2D_dot\(\)](#), [mat2D_fill\(\)](#), [mat2D_free\(\)](#), [mat2D_set_rot_mat_z\(\)](#), and [mat2D_sub\(\)](#).

Referenced by [adl_axis_draw_on_figure\(\)](#).

4.1.3.3 adl_axis_draw_on_figure()

```
void adl_axis_draw_on_figure (
    Figure * figure )
```

Draw X/Y axes with arrowheads into a [Figure](#).

Uses the current figure's pixel extents and padding to place axes, and stores the computed head sizes for later label layout.

Parameters

<i>figure</i>	[in,out] Figure to draw onto.
---------------	---

Definition at line 2144 of file [Almog_Draw_Library.h](#).

References [adl_arrow_draw\(\)](#), [ADL_PICTURE_AXIS_COLOR](#), [ADL_PICTURE_HEAD_ANGLE_DEG](#), [ADL_PICTURE_PADDING_PREC](#), [ADL_MAX_PICTURE_PADDING](#), [ADL_MAX_HEAD_SIZE](#), [ADL_MIN_PICTURE_PADDING](#), [Mat2D::uint32::cols](#), [Figure::max_x_pixel](#), [Figure::max_y_pixel](#), [Figure::min_x_pixel](#), [Figure::min_y_pixel](#), [Figure::offset_zoom_param](#), [Figure::pixels_mat](#), [Mat2D::uint32::rows](#), [Figure::x_axis_head_size](#), and [Figure::y_axis_head_size](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), and [adl_curves_plot_on_figure\(\)](#).

4.1.3.4 adl_cartesian_grid_create()

```
Grid adl_cartesian_grid_create (
    float min_e1,
    float max_e1,
    float min_e2,
    float max_e2,
    int num_samples_e1,
    int num_samples_e2,
    char plane[],
    float third_direction_position )
```

Create a Cartesian grid (as curves) on one of the principal planes.

Supported planes (case-insensitive): "XY", "xy", "XZ", "xz", "YX", "yx", "YZ", "yz", "ZX", "zx", "ZY", "zy". The *third_direction_position* places the grid along the axis normal to the plane (e.g., Z for "XY").

Parameters

<i>min_e1</i>	Minimum coordinate along the first axis of the plane.
<i>max_e1</i>	Maximum coordinate along the first axis of the plane.
<i>min_e2</i>	Minimum coordinate along the second axis of the plane.
<i>max_e2</i>	Maximum coordinate along the second axis of the plane.
<i>num_samples_e1</i>	Number of segments along first axis.
<i>num_samples_e2</i>	Number of segments along second axis.
<i>plane</i>	Plane code string ("XY", "xy", "XZ", "xz", "YX", "yx", "YZ", "yz", "ZX", "zx", "ZY", "zy").
<i>third_direction_position</i>	Position along the axis normal to plane.

Returns

`Grid` structure containing the generated curves and spacing.

Definition at line 2513 of file [Almog_Draw_Library.h](#).

References `ada_appand`, `ada_init_array`, `Grid::curves`, `Grid::de1`, `Grid::de2`, `Grid::max_e1`, `Grid::max_e2`, `Grid::min_e1`, `Grid::min_e2`, `Grid::num_samples_e1`, `Grid::num_samples_e2`, `Grid::plane`, `Point::w`, `Point::x`, `Point::y`, and `Point::z`.

4.1.3.5 adl_character_draw()

```
void adl_character_draw (
    Mat2D_uint32 screen_mat,
    char c,
    int width_pixel,
    int height_pixel,
    int x_top_left,
    int y_top_left,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw a vector glyph for a single ASCII character.

Only a limited set of characters is supported (A–Z, a–z, 0–9, space, '.', ',', '-', '+'). Unsupported characters are rendered as a framed box with an 'X'. Coordinates are for the character's top-left corner.

Parameters

<code>screen_mat</code>	Destination ARGB pixel buffer.
<code>c</code>	The character to draw.
<code>width_pixel</code>	Character box width in pixels.
<code>height_pixel</code>	Character box height in pixels (spelled as in API).
<code>x_top_left</code>	X of top-left corner (before pan/zoom).
<code>y_top_left</code>	Y of top-left corner (before pan/zoom).
<code>color</code>	Stroke color (0xAARRGGBB).
<code>offset_zoom_param</code>	Pan/zoom transform. Use <code>ADL_DEFAULT_OFFSET_ZOOM</code> for identity.

Definition at line 586 of file [Almog_Draw_Library.h](#).

References `adl_line_draw()`, `adl_rectangle_draw_min_max()`, and `adl_rectangle_fill_min_max()`.

Referenced by `adl_sentence_draw()`, and `render()`.

4.1.3.6 adl_circle_draw()

```
void adl_circle_draw (
    Mat2D_uint32 screen_mat,
```

```

    float center_x,
    float center_y,
    float r,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )

```

Draw an approximate circle outline (1px thickness).

The outline is approximated on the integer grid by sampling a band around radius r.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>center_x</i>	Circle center X (before pan/zoom).
<i>center_y</i>	Circle center Y (before pan/zoom).
<i>r</i>	Circle radius in pixels.
<i>color</i>	Stroke color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1427 of file [Almog_Draw_Library.h](#).

References [adl_point_draw\(\)](#).

4.1.3.7 adl_circle_fill()

```

void adl_circle_fill (
    Mat2D_uint32 screen_mat,
    float center_x,
    float center_y,
    float r,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )

```

Fill a circle.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>center_x</i>	Circle center X (before pan/zoom).
<i>center_y</i>	Circle center Y (before pan/zoom).
<i>r</i>	Circle radius in pixels.
<i>color</i>	Fill color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1449 of file [Almog_Draw_Library.h](#).

References [adl_point_draw\(\)](#).

4.1.3.8 adl_curve_add_to_figure()

```
void adl_curve_add_to_figure (
    Figure * figure,
    Point * src_points,
    size_t src_len,
    uint32_t color )
```

Add a curve (polyline) to a [Figure](#) and update its data bounds.

The input points are copied into the figure's source curve array with the given color. [Figure](#) min/max bounds are updated to include them.

Parameters

<i>figure</i>	[in,out] Target figure.
<i>src_points</i>	Array of source points (in data space).
<i>src_len</i>	Number of points.
<i>color</i>	Curve color (0xAARRGGBB).

Definition at line 2230 of file [Almog_Draw_Library.h](#).

References [ada_appand](#), [ada_init_array](#), [Curve::color](#), [Figure::max_x](#), [Figure::max_y](#), [Figure::min_x](#), [Figure::min_y](#), [Figure::src_curve_array](#), [Point::x](#), and [Point::y](#).

Referenced by [setup\(\)](#).

4.1.3.9 adl_curves_plot_on_figure()

```
void adl_curves_plot_on_figure (
    Figure figure )
```

Render all added curves into a [Figure](#)'s pixel buffer.

Clears the pixel buffer to [background_color](#), draws axes if enabled, maps data-space points to pixel-space using current min/max bounds, draws the polylines, and optionally draws min/max labels.

Parameters

<i>figure</i>	Figure to render into (uses its own pixel buffer).
---------------	--

Definition at line 2265 of file [Almog_Draw_Library.h](#).

References [adl_axis_draw_on_figure\(\)](#), [adl_line_draw\(\)](#), [adl_linear_map\(\)](#), [adl_max_min_values_draw_on_figure\(\)](#), [Figure::background_color](#), [Curve::color](#), [Mat2D::cols](#), [Curve::elements](#), [Curve_ada::elements](#), [Mat2D::elements](#), [Figure::inv_z_buffer_mat](#), [Curve::length](#), [Curve_ada::length](#), [mat2D_fill_uint32\(\)](#), [Figure::max_x](#), [Figure::max_x_pixel](#), [Figure::max_y](#), [Figure::max_y_pixel](#), [Figure::min_x](#), [Figure::min_x_pixel](#), [Figure::min_y](#), [Figure::min_y_pixel](#), [Figure::offset_zoom_param](#), [Figure::pixels_mat](#), [Mat2D::rows](#), [Figure::src_curve_array](#), [Figure::to_draw_axis](#), [Figure::to_draw_max_min_values](#), [Point::x](#), and [Point::y](#).

Referenced by [render\(\)](#).

4.1.3.10 adl_figure_alloc()

```
Figure adl_figure_alloc (
    size_t rows,
    size_t cols,
    Point top_left_position )
```

Allocate and initialize a [Figure](#) with an internal pixel buffer.

Initializes the pixel buffer (rows x cols), an inverse-Z buffer (zeroed), an empty source curve array, and default padding/axes bounds. The `background_color`, `to_draw_axis`, and `to_draw_max_min_values` should be set by the caller before rendering.

Parameters

<code>rows</code>	Height of the figure in pixels.
<code>cols</code>	Width of the figure in pixels.
<code>top_left_position</code>	Target position when copying to a screen.

Returns

A new [Figure](#) with allocated buffers.

Definition at line 2081 of file [Almog_Draw_Library.h](#).

References `ada_init_array`, `ADL_ASSERT`, `adl_assert_point_is_valid`, `ADL_DEFAULT_OFFSET_ZOOM`, `ADL FIGURE_PADDING_PERCENTAGE`, `ADL_MAX FIGURE_PADDING`, `Mat2D::cols`, `Mat2D::uint32::cols`, `Mat2D::elements`, `Figure::inv_z_buffer_mat`, `mat2D_alloc()`, `mat2D_alloc_uint32()`, `Figure::max_x`, `Figure::max_x_pixel`, `Figure::max_y`, `Figure::max_y_pixel`, `Figure::min_x`, `Figure::min_x_pixel`, `Figure::min_y`, `Figure::min_y_pixel`, `Figure::offset_zoom_param`, `Figure::pixels_mat`, `Mat2D::rows`, `Mat2D::uint32::rows`, `Figure::src_curve_array`, and `Figure::top_left_position`.

Referenced by [setup\(\)](#).

4.1.3.11 adl_figure_copy_to_screen()

```
void adl_figure_copy_to_screen (
    Mat2D::uint32 screen_mat,
    Figure figure )
```

Blit a [Figure](#)'s pixels onto a destination screen buffer.

Performs per-pixel blending using `adl_point_draw` and the identity transform. The figure's `top_left_position` is used as the destination offset.

Parameters

<code>screen_mat</code>	Destination ARGB pixel buffer.
<code>figure</code>	Source figure to copy from.

Definition at line 2124 of file [Almog_Draw_Library.h](#).

References [adl_point_draw\(\)](#), [Mat2D::uint32::cols](#), [MAT2D_AT_UINT32](#), [Figure::pixels_mat](#), [Mat2D::uint32::rows](#), [Figure::top_left_position](#), [Point::x](#), and [Point::y](#).

Referenced by [render\(\)](#).

4.1.3.12 adl_grid_draw()

```
void adl_grid_draw (
    Mat2D::uint32 screen_mat,
    Grid grid,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw a previously created [Grid](#) as line segments.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>grid</i>	Grid to draw (curves are 2-point polylines).
<i>color</i>	Line color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 2791 of file [Almog_Draw_Library.h](#).

References [adl_lines_draw\(\)](#), [Grid::curves](#), [Curve::elements](#), [Curve_ada::elements](#), [Curve::length](#), and [Curve_ada::length](#).

4.1.3.13 adl_interpolate_ARGBcolor_on_okLch()

```
void adl_interpolate_ARGBcolor_on_okLch (
    uint32_t color1,
    uint32_t color2,
    float t,
    float num_of_rotations,
    uint32_t * color_out )
```

Interpolate between two ARGB colors in OkLch space.

Lightness and chroma are interpolated linearly. Hue is interpolated in degrees after adding $360 \times \text{num_of_rotations}$ to the second hue, allowing control over the winding direction.

Parameters

<i>color1</i>	Start color (0xAARRGGBB).
<i>color2</i>	End color (0xAARRGGBB).
<i>t</i>	Interpolation factor in [0,1].
<i>num_of_rotations</i>	Number of hue turns to add to color2 (can be fractional/negative).
<i>color_out</i>	[out] Interpolated ARGB color (A=255).

Definition at line 2053 of file [Almog_Draw_Library.h](#).

References [adl_linear_sRGB_to_okLch\(\)](#), and [adl_okLch_to_linear_sRGB\(\)](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#).

4.1.3.14 adl_line_draw()

```
void adl_line_draw (
    Mat2D_uint32 screen_mat,
    const float x1_input,
    const float y1_input,
    const float x2_input,
    const float y2_input,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw an anti-aliased-like line by vertical spans (integer grid).

The line is rasterized with a simple integer-span approach. Pan/zoom is applied about the screen center prior to rasterization.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>x1_input</i>	Line start X (before pan/zoom).
<i>y1_input</i>	Line start Y (before pan/zoom).
<i>x2_input</i>	Line end X (before pan/zoom).
<i>y2_input</i>	Line end Y (before pan/zoom).
<i>color</i>	Line color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 383 of file [Almog_Draw_Library.h](#).

References [ADL_ASSERT](#), [ADL_MAX_POINT_VAL](#), [adl_point_draw\(\)](#), [Mat2D_uint32::cols](#), [Offset_zoom_param::offset_x](#), [Offset_zoom_param::offset_y](#), [Mat2D_uint32::rows](#), and [Offset_zoom_param::zoom_multiplier](#).

Referenced by [adl_arrow_draw\(\)](#), [adl_character_draw\(\)](#), [adl_curves_plot_on_figure\(\)](#), [adl_lines_draw\(\)](#), [adl_lines_loop_draw\(\)](#), [adl_rectangle_draw_min_max\(\)](#), [adl_rectangle_fill_min_max\(\)](#), and [adl_tri_draw\(\)](#).

4.1.3.15 adl_linear_map()

```
float adl_linear_map (
    float s,
    float min_in,
    float max_in,
    float min_out,
    float max_out )
```

Affine map from one scalar range to another (no clamping).

Parameters

<i>s</i>	Input value.
<i>min_in</i>	Input range minimum.
<i>max_in</i>	Input range maximum.
<i>min_out</i>	Output range minimum.
<i>max_out</i>	Output range maximum.

Returns

Mapped value in the output range (may exceed if *s* is out-of-range).

Definition at line 1865 of file [Almog_Draw_Library.h](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_curves_plot_on_figure\(\)](#), and [render\(\)](#).

4.1.3.16 adl_linear_sRGB_to_okLab()

```
void adl_linear_sRGB_to_okLab (
    uint32_t hex_ARGB,
    float * L,
    float * a,
    float * b )
```

Convert a linear sRGB color (ARGB) to Oklab components.

Oklab components are returned in ranges: L in [0,1], a in [-0.5,0.5], b in [-0.5,0.5] (typical). Input is assumed to be linear sRGB.

Parameters

<i>hex_ARGB</i>	Input color (0xAARRGGBB). Alpha is ignored.
<i>L</i>	[out] Perceptual lightness.
<i>a</i>	[out] First opponent axis.
<i>b</i>	[out] Second opponent axis.

Definition at line 1945 of file [Almog_Draw_Library.h](#).

References [HexARGB_RGB_VAR](#).

Referenced by [adl_linear_sRGB_to_okLch\(\)](#).

4.1.3.17 adl_linear_sRGB_to_okLch()

```
void adl_linear_sRGB_to_okLch (
    uint32_t hex_ARGB,
```

```
    float * L,
    float * c,
    float * h_deg )
```

Convert a linear sRGB color (ARGB) to OkLch components.

Parameters

<i>hex_ARGB</i>	Input color (0xAARRGGBB). Alpha is ignored.
<i>L</i>	[out] Lightness in [0,1].
<i>c</i>	[out] Chroma (non-negative).
<i>h_deg</i>	[out] Hue angle in degrees [-180,180] from atan2.

Definition at line 2012 of file [Almog_Draw_Library.h](#).

References [adl_linear_sRGB_to_okLab\(\)](#), and [PI](#).

Referenced by [adl_interpolate_ARGBcolor_on_okLch\(\)](#).

4.1.3.18 adl_lines_draw()

```
void adl_lines_draw (
    const Mat2D_uint32 screen_mat,
    const Point * points,
    const size_t len,
    const uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw a polyline connecting an array of points.

Draws segments between consecutive points: p[0]-p[1]-...-p[len-1].

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>points</i>	Array of points in pixel space (before pan/zoom).
<i>len</i>	Number of points in the array (≥ 1).
<i>color</i>	Line color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 470 of file [Almog_Draw_Library.h](#).

References [adl_line_draw\(\)](#), and [points](#).

Referenced by [adl_grid_draw\(\)](#).

4.1.3.19 adl_lines_loop_draw()

```
void adl_lines_loop_draw (
    const Mat2D_uint32 screen_mat,
    const Point * points,
    const size_t len,
    const uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw a closed polyline (loop).

Same as adl_lines_draw, plus an extra segment from the last point back to the first point.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>points</i>	Array of points in pixel space (before pan/zoom).
<i>len</i>	Number of points in the array (≥ 1).
<i>color</i>	Line color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 490 of file [Almog_Draw_Library.h](#).

References [adl_line_draw\(\)](#), and [points](#).

Referenced by [adl_quad_draw\(\)](#).

4.1.3.20 adl_max_min_values_draw_on_figure()

```
void adl_max_min_values_draw_on_figure (
    Figure figure )
```

Draw min/max numeric labels for the current data range.

Renders textual min/max values for both axes inside the figure area. Assumes `figure.min_x/max_x/min_y/max_y` have been populated.

Parameters

<i>figure</i>	Figure whose labels are drawn into its own pixel buffer.
---------------	--

Definition at line 2170 of file [Almog_Draw_Library.h](#).

References [ADL_PICTURE_AXIS_COLOR](#), [ADL_MAX_CHARACTER_OFFSET](#), [ADL_MIN_CHARACTER_OFFSET](#), [adl_sentence_draw\(\)](#), [Figure::max_x](#), [Figure::max_x_pixel](#), [Figure::max_y](#), [Figure::max_y_pixel](#), [Figure::min_x](#), [Figure::min_x_pixel](#), [Figure::min_y](#), [Figure::min_y_pixel](#), [Figure::offset_zoom_param](#), [Figure::pixels_mat](#), [Mat2D_uint32::rows](#), [Figure::x_axis_head_size](#), and [Figure::y_axis_head_size](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), and [adl_curves_plot_on_figure\(\)](#).

4.1.3.21 adl_okLab_to_linear_sRGB()

```
void adl_okLab_to_linear_sRGB (
    float L,
    float a,
    float b,
    uint32_t * hex_ARGB )
```

Convert Oklab components to a linear sRGB ARGB color.

Output RGB components are clamped to [0,255], alpha is set to 255.

Parameters

<i>L</i>	Oklab lightness.
<i>a</i>	Oklab a component.
<i>b</i>	Oklab b component.
<i>hex_ARGB</i>	[out] Output color (0xAARRGGBB, A=255).

Definition at line 1980 of file [Almog_Draw_Library.h](#).

References [RGBA_hexARGB](#).

Referenced by [adl_okLch_to_linear_sRGB\(\)](#).

4.1.3.22 adl_okLch_to_linear_sRGB()

```
void adl_okLch_to_linear_sRGB (
    float L,
    float c,
    float h_deg,
    uint32_t * hex_ARGB )
```

Convert OkLch components to a linear sRGB ARGB color.

Hue is wrapped to [0,360). Output RGB is clamped to [0,255], alpha=255.

Parameters

<i>L</i>	Lightness.
<i>c</i>	Chroma.
<i>h_deg</i>	Hue angle in degrees.
<i>hex_ARGB</i>	[out] Output color (0xAARRGGBB, A=255).

Definition at line 2031 of file [Almog_Draw_Library.h](#).

References [adl_okLab_to_linear_sRGB\(\)](#), and [PI](#).

Referenced by [adl_interpolate_ARGBcolor_on_okLch\(\)](#).

4.1.3.23 adl_point_draw()

```
void adl_point_draw (
    Mat2D_uint32 screen_mat,
    int x,
    int y,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw a single pixel with alpha blending.

Applies the pan/zoom transform and writes the pixel if it falls inside the destination bounds. The source color is blended over the existing pixel using the source alpha; the stored alpha is set to 255.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>x</i>	X coordinate in pixels (before pan/zoom).
<i>y</i>	Y coordinate in pixels (before pan/zoom).
<i>color</i>	Source color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 350 of file [Almog_Draw_Library.h](#).

References [Mat2D::cols](#), [HexARGB_RGBA_VAR](#), [MAT2D_AT_UINT32](#), [Offset_zoom_param::offset_x](#), [Offset_zoom_param::offset_y](#), [RGBA_hexARGB](#), [Mat2D::rows](#), and [Offset_zoom_param::zoom_multiplier](#).

Referenced by [adl_circle_draw\(\)](#), [adl_circle_fill\(\)](#), [adl_figure_copy_to_screen\(\)](#), [adl_line_draw\(\)](#), [adl_quad_fill\(\)](#), [adl_quad_fill_interpolate_color_mean_value\(\)](#), [adl_quad_fill_interpolate_normal_mean_value\(\)](#), [adl_tri_fill_Pinedas_rasterizer\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), and [adl_tri_fill_Pinedas_rasterizer_interpolate_normal\(\)](#).

4.1.3.24 adl_quad2tris()

```
void adl_quad2tris (
    Quad quad,
    Tri * tri1,
    Tri * tri2,
    char split_line[] )
```

Split a quad into two triangles along a chosen diagonal.

The split is controlled by *split_line*:

- "02" splits along diagonal from vertex 0 to vertex 2.
- "13" splits along diagonal from vertex 1 to vertex 3.

The function copies positions, per-vertex colors, light_intensity, and the to_draw flag into the output triangles.

Parameters

<code>quad</code>	Input quad.
<code>tri1</code>	[out] First output triangle.
<code>tri2</code>	[out] Second output triangle.
<code>split_line</code>	Null-terminated code: "02" or "13".

Definition at line 1885 of file [Almog_Draw_Library.h](#).

References [Tri::colors](#), [Quad::colors](#), [Tri::light_intensity](#), [Quad::light_intensity](#), [Tri::points](#), [Quad::points](#), [Tri::to_draw](#), and [Quad::to_draw](#).

4.1.3.25 adl_quad_draw()

```
void adl_quad_draw (
    Mat2D<uint32> screen_mat,
    Mat2D inv_z_buffer,
    Quad quad,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw the outline of a quad (four points, looped).

Depth buffer is not used in this outline variant.

Parameters

<code>screen_mat</code>	Destination ARGB pixel buffer.
<code>inv_z_buffer</code>	Unused for outline; safe to pass a dummy Mat2D .
<code>quad</code>	Quad to draw in pixel space (before transform).
<code>color</code>	Stroke color (0xAARRGGBB).
<code>offset_zoom_param</code>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1010 of file [Almog_Draw_Library.h](#).

References [adl_lines_loop_draw\(\)](#), and [Quad::points](#).

Referenced by [adl_quad_mesh_draw\(\)](#), and [render\(\)](#).

4.1.3.26 adl_quad_fill()

```
void adl_quad_fill (
    Mat2D<uint32> screen_mat,
    Mat2D inv_z_buffer,
    Quad quad,
```

```
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Fill a quad using mean-value (Barycentric) coordinates and flat base color.

Performs a depth test against inv_z_buffer and modulates the base color with the average light_intensity of the quad's vertices.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>inv_z_buffer</i>	Inverse-Z buffer (larger is closer).
<i>quad</i>	Quad in pixel space; points carry z and w for depth.
<i>color</i>	Base color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1028 of file [Almog_Draw_Library.h](#).

References [adl_point_draw\(\)](#), [Mat2D::uint32::cols](#), [edge_cross_point](#), [HexARGB_RGB_ALPHA_VAR](#), [Quad::light_intensity](#), [MAT2D_AT](#), [Quad::points](#), [RGB_ALPHA_hexARGB](#), [Mat2D::uint32::rows](#), [Point::w](#), [Point::x](#), [Point::y](#), and [Point::z](#).

Referenced by [adl_quad_mesh_fill\(\)](#).

4.1.3.27 adl_quad_fill_interpolate_color_mean_value()

```
void adl_quad_fill_interpolate_color_mean_value (
    Mat2D<uint32> screen_mat,
    Mat2D<int> inv_z_buffer,
    Quad< > quad,
    Offset_zoom_param offset_zoom_param )
```

Fill a quad with per-vertex colors (mean value coords).

Interpolates ARGB vertex colors using mean-value coordinates, optionally modulated by the average light_intensity. Depth-tested.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>inv_z_buffer</i>	Inverse-Z buffer (larger is closer).
<i>quad</i>	Quad in pixel space with quad.colors[] set.
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1216 of file [Almog_Draw_Library.h](#).

References [adl_point_draw\(\)](#), [adl_tan_half_angle\(\)](#), [Quad::colors](#), [Mat2D::uint32::cols](#), [edge_cross_point](#), [HexARGB_RGB_ALPHA_VAR](#), [Quad::light_intensity](#), [MAT2D_AT](#), [Quad::points](#), [RGB_ALPHA_hexARGB](#), [Mat2D::uint32::rows](#), [Point::w](#), [Point::x](#), [Point::y](#), and [Point::z](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), [adl_quad_mesh_fill_interpolate_color\(\)](#), and [render\(\)](#).

4.1.3.28 adl_quad_fill_interpolate_normal_mean_value()

```
void adl_quad_fill_interpolate_normal_mean_value (
    Mat2D_uint32 screen_mat,
    Mat2D inv_z_buffer,
    Quad quad,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Fill a quad with per-pixel light interpolation (mean value coords).

Interpolates light_intensity across the quad using mean-value coordinates and modulates a uniform base color. Depth-tested.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>inv_z_buffer</i>	Inverse-Z buffer (larger is closer).
<i>quad</i>	Quad in pixel space; points carry z and w for depth.
<i>color</i>	Base color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1122 of file [Almog_Draw_Library.h](#).

References [adl_point_draw\(\)](#), [adl_tan_half_angle\(\)](#), [Mat2D::uint32::cols](#), [edge_cross_point](#), [HexARGB_RGBA_VAR](#), [Quad::light_intensity](#), [MAT2D_AT](#), [Quad::points](#), [RGBA_hexARGB](#), [Mat2D::uint32::rows](#), [Point::w](#), [Point::x](#), [Point::y](#), and [Point::z](#).

Referenced by [adl_quad_mesh_fill_interpolate_normal\(\)](#).

4.1.3.29 adl_quad_mesh_draw()

```
void adl_quad_mesh_draw (
    Mat2D_uint32 screen_mat,
    Mat2D inv_z_buffer_mat,
    Quad_mesh mesh,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw outlines for all quads in a mesh.

Skips elements with to_draw == false. Depth buffer is not used.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>inv_z_buffer_mat</i>	Unused for outline; safe to pass a dummy Mat2D.
<i>mesh</i>	Quad mesh (array + length).
<i>color</i>	Stroke color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1320 of file [Almog_Draw_Library.h](#).

References `adl_assert_quad_is_valid`, `adl_quad_draw()`, `Quad_mesh::elements`, `Quad_mesh::length`, and `Quad::to_draw`.

4.1.3.30 adl_quad_mesh_fill()

```
void adl_quad_mesh_fill (
    Mat2D<uint32> screen_mat,
    Mat2D<float> inv_z_buffer_mat,
    Quad_mesh mesh,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Fill all quads in a mesh with a uniform base color.

Applies per-quad average light_intensity. Depth-tested.

Parameters

<code>screen_mat</code>	Destination ARGB pixel buffer.
<code>inv_z_buffer_mat</code>	Inverse-Z buffer (larger is closer).
<code>mesh</code>	<code>Quad</code> mesh (array + length).
<code>color</code>	Base color (0xAARRGGBB).
<code>offset_zoom_param</code>	Pan/zoom transform. Use <code>ADL_DEFAULT_OFFSET_ZOOM</code> for identity.

Definition at line 1344 of file [Almog_Draw_Library.h](#).

References `adl_assert_quad_is_valid`, `adl_quad_fill()`, `Quad_mesh::elements`, `Quad_mesh::length`, and `Quad::to_draw`.

4.1.3.31 adl_quad_mesh_fill_interpolate_color()

```
void adl_quad_mesh_fill_interpolate_color (
    Mat2D<uint32> screen_mat,
    Mat2D<float> inv_z_buffer_mat,
    Quad_mesh mesh,
    Offset_zoom_param offset_zoom_param )
```

Fill all quads in a mesh using per-vertex colors.

Interpolates `quad.colors[]` across each quad with mean-value coordinates. Depth-tested.

Parameters

<code>screen_mat</code>	Destination ARGB pixel buffer.
<code>inv_z_buffer_mat</code>	Inverse-Z buffer (larger is closer).
<code>mesh</code>	<code>Quad</code> mesh (array + length).
<code>offset_zoom_param</code>	Pan/zoom transform. Use <code>ADL_DEFAULT_OFFSET_ZOOM</code> for identity.

Definition at line 1401 of file [Almog_Draw_Library.h](#).

References [adl_assert_quad_is_valid](#), [adl_quad_fill_interpolate_color_mean_value\(\)](#), [Quad_mesh::elements](#), [Quad_mesh::length](#), and [Quad::to_draw](#).

4.1.3.32 adl_quad_mesh_fill_interpolate_normal()

```
void adl_quad_mesh_fill_interpolate_normal (
    Mat2D<uint32> screen_mat,
    Mat2D<float> inv_z_buffer_mat,
    Quad_mesh mesh,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Fill all quads in a mesh using interpolated lighting.

Interpolates light_intensity across quads and modulates a uniform base color. Depth-tested.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>inv_z_buffer_mat</i>	Inverse-Z buffer (larger is closer).
<i>mesh</i>	Quad mesh (array + length).
<i>color</i>	Base color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1371 of file [Almog_Draw_Library.h](#).

References [adl_assert_quad_is_valid](#), [adl_quad_fill_interpolate_normal_mean_value\(\)](#), [Quad_mesh::elements](#), [HexARGB_RGBA_VAR](#), [Quad_mesh::length](#), and [Quad::to_draw](#).

4.1.3.33 adl_rectangle_draw_min_max()

```
void adl_rectangle_draw_min_max (
    Mat2D<uint32> screen_mat,
    int min_x,
    int max_x,
    int min_y,
    int max_y,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw a rectangle outline defined by min/max corners (inclusive).

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>min_x</i>	Minimum X (before pan/zoom).

Parameters

<i>max_x</i>	Maximum X (before pan/zoom).
<i>min_y</i>	Minimum Y (before pan/zoom).
<i>max_y</i>	Maximum Y (before pan/zoom).
<i>color</i>	Stroke color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 973 of file [Almog_Draw_Library.h](#).

References [adl_line_draw\(\)](#).

Referenced by [adl_character_draw\(\)](#).

4.1.3.34 adl_rectangle_fill_min_max()

```
void adl_rectangle_fill_min_max (
    Mat2D_uint32 screen_mat,
    int min_x,
    int max_x,
    int min_y,
    int max_y,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Fill a rectangle defined by min/max corners (inclusive).

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>min_x</i>	Minimum X (before pan/zoom).
<i>max_x</i>	Maximum X (before pan/zoom).
<i>min_y</i>	Minimum Y (before pan/zoom).
<i>max_y</i>	Maximum Y (before pan/zoom).
<i>color</i>	Fill color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 992 of file [Almog_Draw_Library.h](#).

References [adl_line_draw\(\)](#).

Referenced by [adl_character_draw\(\)](#).

4.1.3.35 adl_sentence_draw()

```
void adl_sentence_draw (
    Mat2D_uint32 screen_mat,
```

```

    const char sentence[],
    size_t len,
    const int x_top_left,
    const int y_top_left,
    const int hight_pixel,
    const uint32_t color,
    Offset_zoom_param offset_zoom_param )

```

Draw a horizontal sentence using vector glyphs.

Characters are laid out left-to-right with a spacing derived from the character height. All characters share the same height.

Parameters

<code>screen_mat</code>	Destination ARGB pixel buffer.
<code>sentence</code>	ASCII string buffer.
<code>len</code>	Number of characters to draw from sentence.
<code>x_top_left</code>	X of top-left of the first character (before transform).
<code>y_top_left</code>	Y of top-left of the first character (before transform).
<code>hight_pixel</code>	Character height in pixels (spelled as in API).
<code>color</code>	Stroke color (0xAARRGGBB).
<code>offset_zoom_param</code>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 949 of file [Almog_Draw_Library.h](#).

References [adl_character_draw\(\)](#), [ADL_MAX_CHARACTER_OFFSET](#), and [ADL_MIN_CHARACTER_OFFSET](#).

Referenced by [adl_max_min_values_draw_on_figure\(\)](#).

4.1.3.36 adl_tan_half_angle()

```

float adl_tan_half_angle (
    Point vi,
    Point vj,
    Point p,
    float li,
    float lj )

```

Compute $\tan(\alpha/2)$ for the angle at point p between segments p->vi and p->vj.

Uses the identity $\tan(\alpha/2) = |a \times b| / (|a||b| + a \cdot b)$, where $a = vi - p$ and $b = vj - p$. The lengths $li = |a|$ and $lj = |b|$ are passed in to avoid recomputation.

Parameters

<code>vi</code>	Vertex i.
<code>vj</code>	Vertex j.
<code>p</code>	Pivot point.
<code>li</code>	Precomputed $ vi - p $.
<code>lj</code>	Precomputed $ vj - p $.

Returns

$\tan(\alpha/2)$ (non-negative).

Definition at line 1845 of file [Almog_Draw_Library.h](#).

References [Point::x](#), and [Point::y](#).

Referenced by [adl_quad_fill_interpolate_color_mean_value\(\)](#), and [adl_quad_fill_interpolate_normal_mean_value\(\)](#).

4.1.3.37 adl_tri_draw()

```
void adl_tri_draw (
    Mat2D_uint32 screen_mat,
    Tri tri,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw the outline of a triangle.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>tri</i>	Triangle in pixel space (before transform).
<i>color</i>	Stroke color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1469 of file [Almog_Draw_Library.h](#).

References [adl_line_draw\(\)](#), [Tri::points](#), [tri](#), [Point::x](#), and [Point::y](#).

Referenced by [adl_tri_mesh_draw\(\)](#), and [render\(\)](#).

4.1.3.38 adl_tri_fill_Pinedas_rasterizer()

```
void adl_tri_fill_Pinedas_rasterizer (
    Mat2D_uint32 screen_mat,
    Mat2D_inv_z_buffer,
    Tri tri,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Fill a triangle using Pineda's rasterizer with flat base color.

Uses the top-left fill convention and performs a depth test using inverse-Z computed from per-vertex z and w.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>inv_z_buffer</i>	Inverse-Z buffer (larger is closer).
<i>tri</i>	Triangle in pixel space; points carry z and w for depth.
<i>color</i>	Base color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1492 of file [Almog_Draw_Library.h](#).

References [adl_point_draw\(\)](#), [Mat2D::uint32::cols](#), [edge_cross_point](#), [HexARGB_RGB_ALPHA_VAR](#), [is_top_left](#), [Tri::light_intensity](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), [Tri::points](#), [RGB_hexARGB](#), [Mat2D::uint32::rows](#), [tri](#), [Point::w](#), [Point::x](#), [Point::y](#), and [Point::z](#).

Referenced by [adl_tri_mesh_fill_Pinedas_rasterizer\(\)](#).

4.1.3.39 adl_tri_fill_Pinedas_rasterizer_interpolate_color()

```
void adl_tri_fill_Pinedas_rasterizer_interpolate_color (
    Mat2D_uint32 screen_mat,
    Mat2D inv_z_buffer,
    Tri tri,
    Offset_zoom_param offset_zoom_param )
```

Fill a triangle using Pineda's rasterizer with per-vertex colors.

Interpolates *tri.colors[]* and optionally modulates by average *light_intensity*. Depth-tested.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>inv_z_buffer</i>	Inverse-Z buffer (larger is closer).
<i>tri</i>	Triangle in pixel space with colors set.
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1573 of file [Almog_Draw_Library.h](#).

References [adl_point_draw\(\)](#), [Tri::colors](#), [Mat2D::uint32::cols](#), [edge_cross_point](#), [HexARGB_RGB_ALPHA_VAR](#), [is_top_left](#), [Tri::light_intensity](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), [Tri::points](#), [RGB_hexARGB](#), [Mat2D::uint32::rows](#), [tri](#), [Point::w](#), [Point::x](#), [Point::y](#), and [Point::z](#).

Referenced by [adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color\(\)](#), and [render\(\)](#).

4.1.3.40 adl_tri_fill_Pinedas_rasterizer_interpolate_normal()

```
void adl_tri_fill_Pinedas_rasterizer_interpolate_normal (
    Mat2D_uint32 screen_mat,
```

```
Mat2D inv_z_buffer,
Tri tri,
uint32_t color,
Offset_zoom_param offset_zoom_param )
```

Fill a triangle with interpolated lighting over a uniform color.

Interpolates light_intensity across the triangle and modulates a uniform base color. Depth-tested.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>inv_z_buffer</i>	Inverse-Z buffer (larger is closer).
<i>tri</i>	Triangle in pixel space; points carry z and w for depth.
<i>color</i>	Base color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1664 of file [Almog_Draw_Library.h](#).

References [adl_point_draw\(\)](#), [Mat2D::uint32::cols](#), [edge_cross_point](#), [HexARGB_RGB_ALPHA_VAR](#), [is_top_left](#), [Tri::light_intensity](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), [Tri::points](#), [RGBAlpha_hexARGB](#), [Mat2D::uint32::rows](#), [tri](#), [Point::w](#), [Point::x](#), [Point::y](#), and [Point::z](#).

Referenced by [adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal\(\)](#).

4.1.3.41 adl_tri_mesh_draw()

```
void adl_tri_mesh_draw (
    Mat2D_uint32 screen_mat,
    Tri_mesh mesh,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Draw outlines for all triangles in a mesh.

Skips elements with `to_draw == false`.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>mesh</i>	Triangle mesh (array + length).
<i>color</i>	Stroke color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1746 of file [Almog_Draw_Library.h](#).

References [adl_tri_draw\(\)](#), [Tri_mesh::elements](#), [Tri_mesh::length](#), [Tri::to_draw](#), and [tri](#).

4.1.3.42 adl_tri_mesh_fill_Pinedas_rasterizer()

```
void adl_tri_mesh_fill_Pinedas_rasterizer (
    Mat2D<uint32> screen_mat,
    Mat2D<float> inv_z_buffer_mat,
    Tri_mesh mesh,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Fill all triangles in a mesh with a uniform base color.

Applies average light_intensity per triangle. Depth-tested.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>inv_z_buffer_mat</i>	Inverse-Z buffer (larger is closer).
<i>mesh</i>	Triangle mesh (array + length).
<i>color</i>	Base color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1768 of file [Almog_Draw_Library.h](#).

References [adl_assert_tri_is_valid](#), [adl_tri_fill_Pinedas_rasterizer\(\)](#), [Tri_mesh::elements](#), [Tri_mesh::length](#), [Tri::to_draw](#), and [tri](#).

4.1.3.43 adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color()

```
void adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color (
    Mat2D<uint32> screen_mat,
    Mat2D<float> inv_z_buffer_mat,
    Tri_mesh mesh,
    Offset_zoom_param offset_zoom_param )
```

Fill all triangles in a mesh with a uniform base color.

Applies average light_intensity per triangle. Depth-tested.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>inv_z_buffer_mat</i>	Inverse-Z buffer (larger is closer).
<i>mesh</i>	Triangle mesh (array + length).
<i>color</i>	Base color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1792 of file [Almog_Draw_Library.h](#).

References [adl_assert_tri_is_valid](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), [Tri_mesh::elements](#), [Tri_mesh::length](#), [Tri::to_draw](#), and [tri](#).

4.1.3.44 adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal()

```
void adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal (
    Mat2D_uint32 screen_mat,
    Mat2D inv_z_buffer_mat,
    Tri_mesh mesh,
    uint32_t color,
    Offset_zoom_param offset_zoom_param )
```

Fill all triangles in a mesh with interpolated lighting.

Interpolates light_intensity across each triangle and modulates a uniform base color. Depth-tested.

Parameters

<i>screen_mat</i>	Destination ARGB pixel buffer.
<i>inv_z_buffer_mat</i>	Inverse-Z buffer (larger is closer).
<i>mesh</i>	Triangle mesh (array + length).
<i>color</i>	Base color (0xAARRGGBB).
<i>offset_zoom_param</i>	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1817 of file [Almog_Draw_Library.h](#).

References [adl_assert_tri_is_valid](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_normal\(\)](#), [Tri_mesh::elements](#), [Tri_mesh::length](#), [Tri::to_draw](#), and [tri](#).

4.2 Almog_Draw_Library.h

```
00001
00034 #ifndef ALMOG_DRAW_LIBRARY_H_
00035 #define ALMOG_DRAW_LIBRARY_H_
00036
00037 #include <math.h>
00038 #include <stdint.h>
00039 #include <limits.h>
00040 #include <string.h>
00041 #include <float.h>
00042
00043 #include "./Matrix2D.h"
00044 #include "./Almog_Dynamic_Array.h"
00045
00053 #ifndef ADL_ASSERT
00054 #include <assert.h>
00055 #define ADL_ASSERT assert
00056 #endif
00057
00065 typedef struct {
00066     float zoom_multiplier;
00067     float offset_x;
00068     float offset_y;
00069     int mouse_x;
00070     int mouse_y;
00071 } Offset_zoom_param;
00072
00073 #ifndef POINT
00074 #define POINT
00081 typedef struct {
00082     float x;
00083     float y;
00084     float z;
00085     float w;
00086 } Point;
```

```
00087 #endif
00088
00089 #ifndef CURVE
00090 #define CURVE
00094 typedef struct {
00095     uint32_t color;
00096     size_t length;
00097     size_t capacity;
00098     Point *elements;
00099 } Curve;
00100#endif
00101
00102 #ifndef CURVE_ADA
00103 #define CURVE_ADA
00107 typedef struct {
00108     size_t length;
00109     size_t capacity;
00110     Curve *elements;
00111 } Curve_ada;
00112#endif
00113
00114 #ifndef TRI
00115 #define TRI
00119 typedef struct {
00120     Point points[3];
00121     Point tex_points[3];
00122     Point normals[3];
00123     uint32_t colors[3];
00124     bool to_draw;
00125     float light_intensity[3];
00126 } Tri;
00127#endif
00128
00129 #ifndef QUAD
00130 #define QUAD
00134 typedef struct {
00135     Point points[4];
00136     Point normals[4];
00137     uint32_t colors[4];
00138     bool to_draw;
00139     float light_intensity[4];
00140 } Quad;
00141#endif
00142
00143 #ifndef TRI_MESH
00144 #define TRI_MESH
00148 typedef struct {
00149     size_t length;
00150     size_t capacity;
00151     Tri *elements;
00152 } Tri_mesh; /* Tri ada array */
00153#endif
00154
00155 #ifndef QUAD_MESH
00156 #define QUAD_MESH
00160 typedef struct {
00161     size_t length;
00162     size_t capacity;
00163     Quad *elements;
00164 } Quad_mesh; /* Quad ada array */
00165#endif
00166
00174 typedef struct {
00175     int min_x_pixel;
00176     int max_x_pixel;
00177     int min_y_pixel;
00178     int max_y_pixel;
00179     float min_x;
00180     float max_x;
00181     float min_y;
00182     float max_y;
00183     int x_axis_head_size;
00184     int y_axis_head_size;
00185     Offset_zoom_param offset_zoom_param;
00186     Curve_ada src_curve_array;
00187     Point top_left_position;
00188     Mat2D_uint32 pixels_mat;
00189     Mat2D inv_z_buffer_mat;
00190     uint32_t background_color;
00191     bool to_draw_axis;
00192     bool to_draw_max_min_values;
00193 } Figure;
00194
00203 typedef struct {
00204     Curve_ada curves;
00205     float min_el;
00206     float max_el;
```

```

00208     float min_e2;
00209     float max_e2;
00211     int num_samples_e1;
00212     int num_samples_e2;
00213     float de1;
00214     float de2;
00216     char plane[3];
00217 } Grid; /* direction: e1, e2 */
00218
00219
00220 #ifndef HexARGB_RGB
00221 #define HexARGB_RGB(x) ((x)»(8*2)&0xFF), ((x)»(8*1)&0xFF), ((x)»(8*0)&0xFF), ((x)»(8*3)&0xFF)
00222#endif
00223 #ifndef HexARGB_RGB_VAR
00224 #define HexARGB_RGB_VAR(x, r, g, b) r = ((x)»(8*2)&0xFF); g = ((x)»(8*1)&0xFF); b = ((x)»(8*0)&0xFF);
00225#endif
00226 #ifndef HexARGB_RGBA_VAR
00227 #define HexARGB_RGBA_VAR(x, r, g, b, a) r = ((x)»(8*2)&0xFF); g = ((x)»(8*1)&0xFF); b =
00228 ((x)»(8*0)&0xFF); a = ((x)»(8*3)&0xFF)
00229#endif
00229 #ifndef RGB_hexRGB
00230 #define RGB_hexRGB(r, g, b) (int)(0x010000*(int)(r) + 0x000100*(int)(g) + 0x000001*(int)(b))
00231#endif
00232 #ifndef RGBA_hexARGB
00233 #define RGBA_hexARGB(r, g, b, a) (int)(0x01000000*(int)(fminf(a, 255)) + 0x010000*(int)(r) +
0x000100*(int)(g) + 0x000001*(int)(b))
00234#endif
00235
00236
00237 void adl_point_draw(Mat2D_uint32 screen_mat, int x, int y, uint32_t color, Offset_zoom_param
offset_zoom_param);
00238 void adl_line_draw(Mat2D_uint32 screen_mat, const float x1_input, const float y1_input, const float
x2_input, const float y2_input, uint32_t color, Offset_zoom_param offset_zoom_param);
00239 void adl_lines_draw(const Mat2D_uint32 screen_mat, const Point *points, const size_t len, const
uint32_t color, Offset_zoom_param offset_zoom_param);
00240 void adl_lines_loop_draw(const Mat2D_uint32 screen_mat, const Point *points, const size_t len,
const uint32_t color, Offset_zoom_param offset_zoom_param);
00241 void adl_arrow_draw(Mat2D_uint32 screen_mat, int xs, int ys, int xe, int ye, float head_size, float
angle_deg, uint32_t color, Offset_zoom_param offset_zoom_param);
00242
00243 void adl_character_draw(Mat2D_uint32 screen_mat, char c, int width_pixel, int hight_pixel, int
x_top_left, int y_top_left, uint32_t color, Offset_zoom_param offset_zoom_param);
00244 void adl_sentence_draw(Mat2D_uint32 screen_mat, const char sentence[], size_t len, const int
x_top_left, const int y_top_left, const int hight_pixel, const uint32_t color, Offset_zoom_param
offset_zoom_param);
00245
00246 void adl_rectangle_draw_min_max(Mat2D_uint32 screen_mat, int min_x, int max_x, int min_y, int
max_y, uint32_t color, Offset_zoom_param offset_zoom_param);
00247 void adl_rectangle_fill_min_max(Mat2D_uint32 screen_mat, int min_x, int max_x, int min_y, int
max_y, uint32_t color, Offset_zoom_param offset_zoom_param);
00248
00249 void adl_quad_draw(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
Offset_zoom_param offset_zoom_param);
00250 void adl_quad_fill(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
Offset_zoom_param offset_zoom_param);
00251 void adl_quad_fill_interpolate_normal_mean_value(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad
quad, uint32_t color, Offset_zoom_param offset_zoom_param);
00252 void adl_quad_fill_interpolate_color_mean_value(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad
quad, Offset_zoom_param offset_zoom_param);
00253
00254 void adl_quad_mesh_draw(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Quad_mesh mesh, uint32_t
color, Offset_zoom_param offset_zoom_param);
00255 void adl_quad_mesh_fill(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Quad_mesh mesh, uint32_t
color, Offset_zoom_param offset_zoom_param);
00256 void adl_quad_mesh_fill_interpolate_normal(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat,
Quad_mesh mesh, uint32_t color, Offset_zoom_param offset_zoom_param);
00257 void adl_quad_mesh_fill_interpolate_color(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat,
Quad_mesh mesh, Offset_zoom_param offset_zoom_param);
00258
00259 void adl_circle_draw(Mat2D_uint32 screen_mat, float center_x, float center_y, float r, uint32_t
color, Offset_zoom_param offset_zoom_param);
00260 void adl_circle_fill(Mat2D_uint32 screen_mat, float center_x, float center_y, float r, uint32_t
color, Offset_zoom_param offset_zoom_param);
00261
00262 void adl_tri_draw(Mat2D_uint32 screen_mat, Tri tri, uint32_t color, Offset_zoom_param
offset_zoom_param);
00263 void adl_tri_fill_Pinedas_rasterizer(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Tri tri, uint32_t
color, Offset_zoom_param offset_zoom_param);
00264 void adl_tri_fill_Pinedas_rasterizer_interpolate_color(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer,
Tri tri, Offset_zoom_param offset_zoom_param);
00265 void adl_tri_fill_Pinedas_rasterizer_interpolate_normal(Mat2D_uint32 screen_mat, Mat2D
inv_z_buffer, Tri tri, uint32_t color, Offset_zoom_param offset_zoom_param);
00266
00267 void adl_tri_mesh_draw(Mat2D_uint32 screen_mat, Tri_mesh mesh, uint32_t color, Offset_zoom_param
offset_zoom_param);
00268 void adl_tri_mesh_fill_Pinedas_rasterizer(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Tri_mesh
mesh, uint32_t color, Offset_zoom_param offset_zoom_param);

```

```

00269 void      adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color(Mat2D_uint32 screen_mat, Mat2D
00270           inv_z_buffer_mat, Tri_mesh mesh, Offset_zoom_param offset_zoom_param);
00271 void      adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal(Mat2D_uint32 screen_mat, Mat2D
00272           inv_z_buffer_mat, Tri_mesh mesh, uint32_t color, Offset_zoom_param offset_zoom_param);
00273 float     adl_tan_half_angle(Point vi, Point vj, Point p, float li, float lj);
00274 float     adl_linear_map(float s, float min_in, float max_in, float min_out, float max_out);
00275 void      adl_quad2tris(Quad quad, Tri *tril, Tri *tri2, char split_line[]);
00276 void      adl_linear_sRGB_to_okLab(uint32_t hex_ARGB, float *L, float *a, float *b);
00277 void      adl_okLab_to_linear_sRGB(float L, float a, float b, uint32_t *hex_ARGB);
00278 void      adl_linear_sRGB_to_okLch(uint32_t hex_ARGB, float *L, float *c, float *h_deg);
00279 void      adl_okLch_to_linear_sRGB(float L, float c, float h_deg, uint32_t *hex_ARGB);
00280 void      adl_interpolate_ARGBcolor_on_okLch(uint32_t color1, uint32_t color2, float t, float
00281           num_of_rotations, uint32_t *color_out);
00282
00283 Figure   adl_figure_alloc(size_t rows, size_t cols, Point top_left_position);
00284 void      adl_figure_copy_to_screen(Mat2D_uint32 screen_mat, Figure figure);
00285 void      adl_axis_draw_on_figure(Figure *figure);
00286 void      adl_max_min_values_draw_on_figure(Figure figure);
00287 void      adl_curve_add_to_figure(Figure *figure, Point *src_points, size_t src_len, uint32_t color);
00288 void      adl_curves_plot_on_figure(Figure figure);
00289 void      adl_2Dscalar_interp_on_figure(Figure figure, double *x_2Dmat, double *y_2Dmat, double
00290           *scalar_2Dmat, int ni, int nj, char color_scale[], float num_of_rotations);
00291
00292 Grid      adl_cartesian_grid_create(float min_e1, float max_e1, float min_e2, float max_e2, int
00293           num_samples_e1, int num_samples_e2, char plane[], float third_direction_position);
00294 void      adl_grid_draw(Mat2D_uint32 screen_mat, Grid grid, uint32_t color, Offset_zoom_param
00295           offset_zoom_param);
00296
00297 #endif /*ALMOG_RENDER_SHAPES_H*/
00298
00299 #ifdef ALMOG_DRAW_LIBRARY_IMPLEMENTATION
00300 #undef ALMOG_DRAW_LIBRARY_IMPLEMENTATION
00301
00302 #define RED_hexARGB    0xFFFF0000
00303 #define GREEN_hexARGB  0xFF00FF00
00304 #define BLUE_hexARGB   0xFF0000FF
00305 #define PURPLE_hexARGB 0xFFFF00FF
00306 #define CYAN_hexARGB   0xFF00FFFF
00307 #define YELLOW_hexARGB 0xFFFFFFF0
00308
00309 #define edge_cross_point(a1, b, a2, p) (b.x-a1.x)*(p.y-a2.y)-(b.y-a1.y)*(p.x-a2.x)
00310 #define is_top_edge(x, y) (y == 0 && x > 0)
00311 #define is_left_edge(x, y) (y < 0)
00312 #define is_top_left_edge(ps, pe) (is_top_edge(pe.x-ps.x, pe.y-ps.y) || is_left_edge(pe.x-ps.x, pe.y-ps.y))
00313
00314 #define ADL_MAX_POINT_VAL 1e5
00315 #define adl_assert_point_is_valid(p) ADL_ASSERT(isfinite(p.x) && isfinite(p.y) && isfinite(p.z) &&
00316           isfinite(p.w))
00317 #define adl_assert_tri_is_valid(tri) adl_assert_point_is_valid(tri.points[0]); \
00318           adl_assert_point_is_valid(tri.points[1]); \
00319           adl_assert_point_is_valid(tri.points[2])
00320 #define adl_assert_quad_is_valid(quad) adl_assert_point_is_valid(quad.points[0]); \
00321           adl_assert_point_is_valid(quad.points[1]); \
00322           adl_assert_point_is_valid(quad.points[2]); \
00323           adl_assert_point_is_valid(quad.points[3])
00324
00325 #define ADL FIGURE_PADDING_PRECENTAGE 20
00326 #define ADL_MAX FIGURE_PADDING 70
00327 #define ADL_MIN FIGURE_PADDING 20
00328 #define ADL_MAX HEAD_SIZE 15
00329 #define ADL FIGURE_HEAD_ANGLE_DEG 30
00330 #define ADL FIGURE_AXIS_COLOR 0xff000000
00331 #define ADL_MAX_CHARACTER_OFFSET 10
00332 #define ADL_MIN_CHARACTER_OFFSET 5
00333 #define ADL_MAX_SENTENCE_LEN 256
00334 #define ADL_MAX_ZOOM 1e3
00335
00336 #define ADL_DEFAULT_OFFSET_ZOOM (Offset_zoom_param){1,0,0,0,0}
00337 #define adl_offset_zoom_point(p, window_w, window_h, offset_zoom_param)
00338           \
00339           (p).x = ((p).x - (window_w)/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier +
00340           (window_w)/2; \
00341           (p).y = ((p).y - (window_h)/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier +
00342           (window_h)/2
00343
00344 void      adl_point_draw(Mat2D_uint32 screen_mat, int x, int y, uint32_t color, Offset_zoom_param
00345           offset_zoom_param)
00346 {
00347     float window_w = (float)screen_mat.cols;
00348     float window_h = (float)screen_mat.rows;
00349
00350     x = (x - window_w/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier +
00351           window_w/2;
00352     y = (y - window_h/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier +
00353           window_h/2;

```

```

00357
00358     if ((x < (int)screen_mat.cols && y < (int)screen_mat.rows) && (x >= 0 && y >= 0)) { /* point is in
00359     screen */
00360         uint8_t r_new, g_new, b_new, a_new;
00361         uint8_t r_current, g_current, b_current, a_current;
00362         HexARGB_RGBA_VAR(MAT2D_AT_UINT32(screen_mat, y, x), r_current, g_current, b_current,
00363         a_current);
00363         HexARGB_RGBA_VAR(color, r_new, g_new, b_new, a_new);
00364         MAT2D_AT_UINT32(screen_mat, y, x) = RGB_A_hexARGB(r_current*(1-a_new/255.0f) +
00365         r_new*a_new/255.0f, g_current*(1-a_new/255.0f) + g_new*a_new/255.0f, b_current*(1-a_new/255.0f) +
00366         b_new*a_new/255.0f, 255);
00367         (void)a_current;
00368     }
00369 }
00370
00371 void adl_line_draw(Mat2D_uint32 screen_mat, const float x1_input, const float y1_input, const float
00372 x2_input, const float y2_input, uint32_t color, Offset_zoom_param offset_zoom_param)
00373 {
00374     /* This function is inspired by the Olive.c function developed by 'Tsoding' on his YouTube
00375     channel. You can find the video in this link:
00376     https://youtu.be/LmQKZmQhlZQ?list=PLpM-Dvs8t0Va-Gb0Dp4d9t8yvNFHaKH6N&t=4683. */
00377
00378     float window_w = (float)screen_mat.cols;
00379     float window_h = (float)screen_mat.rows;
00380
00381     int x1 = (x1_input - window_w/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier
00382     + window_w/2;
00383     int x2 = (x2_input - window_w/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier
00384     + window_w/2;
00385     int y1 = (y1_input - window_h/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier
00386     + window_h/2;
00387     int y2 = (y2_input - window_h/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier
00388     + window_h/2;
00389
00390     ADL_ASSERT((int)fabsf(fabsf((float)x2) - fabsf((float)x1)) < ADL_MAX_POINT_VAL);
00391     ADL_ASSERT((int)fabsf(fabsf((float)y2) - fabsf((float)y1)) < ADL_MAX_POINT_VAL);
00392
00393     int x = x1;
00394     int y = y1;
00395     int dx, dy;
00396
00397     adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00398
00399     dx = x2 - x1;
00400     dy = y2 - y1;
00401
00402     ADL_ASSERT(dy > INT_MIN && dy < INT_MAX);
00403     ADL_ASSERT(dx > INT_MIN && dx < INT_MAX);
00404
00405     if (0 == dx && 0 == dy) return;
00406     if (0 == dx) {
00407         while (x != x2 || y != y2) {
00408             if (dy > 0) {
00409                 y++;
00410             }
00411             if (dy < 0) {
00412                 y--;
00413             }
00414             adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00415         }
00416         return;
00417     }
00418     if (0 == dy) {
00419         while (x != x2 || y != y2) {
00420             if (dx > 0) {
00421                 x++;
00422             }
00423             if (dx < 0) {
00424                 x--;
00425             }
00426             adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00427         }
00428         return;
00429     }
00430
00431     /* float m = (float)dy / dx */
00432     int b = y1 - dy * x1 / dx;
00433
00434     if (x1 > x2) {
00435         int temp_x = x1;
00436         x1 = x2;
00437         x2 = temp_x;
00438     }
00439     for (x = x1; x < x2; x++) {
00440         int sy1 = dy * x / dx + b;
00441         int sy2 = dy * (x + 1) / dx + b;
00442         if (sy1 > sy2) {
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
```

```

00448         int temp_y = sy1;
00449         sy1 = sy2;
00450         sy2 = temp_y;
00451     }
00452     for (y = sy1; y <= sy2; y++) {
00453         adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00454     }
00455 }
00456
00457 }
00458
00459 void adl_lines_draw(const Mat2D_uint32 screen_mat, const Point *points, const size_t len, const
00460 uint32_t color, Offset_zoom_param offset_zoom_param)
00461 {
00462     if (len == 0) return;
00463     for (size_t i = 0; i < len-1; i++) {
00464         adl_line_draw(screen_mat, points[i].x, points[i].y, points[i+1].x, points[i+1].y, color,
00465         offset_zoom_param);
00466     }
00467 }
00468
00469 void adl_lines_loop_draw(const Mat2D_uint32 screen_mat, const Point *points, const size_t len, const
00470 uint32_t color, Offset_zoom_param offset_zoom_param)
00471 {
00472     if (len == 0) return;
00473     for (size_t i = 0; i < len-1; i++) {
00474         adl_line_draw(screen_mat, points[i].x, points[i].y, points[i+1].x, points[i+1].y, color,
00475         offset_zoom_param);
00476     }
00477     adl_line_draw(screen_mat, points[len-1].x, points[len-1].y, points[0].x, points[0].y, color,
00478         offset_zoom_param);
00479 }
00480
00481 void adl_arrow_draw(Mat2D_uint32 screen_mat, int xs, int ys, int xe, int ye, float head_size, float
00482 angle_deg, uint32_t color, Offset_zoom_param offset_zoom_param)
00483 {
00484     Mat2D pe = mat2D_alloc(3, 1);
00485     mat2D_fill(pe, 0);
00486     MAT2D_AT(pe, 0, 0) = xe;
00487     MAT2D_AT(pe, 1, 0) = ye;
00488     Mat2D v1 = mat2D_alloc(3, 1);
00489     mat2D_fill(v1, 0);
00490     Mat2D v2 = mat2D_alloc(3, 1);
00491     mat2D_fill(v2, 0);
00492     Mat2D temp_v = mat2D_alloc(3, 1);
00493     mat2D_fill(temp_v, 0);
00494     Mat2D DCM_p = mat2D_alloc(3, 3);
00495     mat2D_fill(DCM_p, 0);
00496     mat2D_set_rot_mat_z(DCM_p, angle_deg);
00497     Mat2D DCM_m = mat2D_alloc(3, 3);
00498     mat2D_fill(DCM_m, 0);
00499     mat2D_set_rot_mat_z(DCM_m, -angle_deg);
00500
00501     int x_center = xs*head_size + xe*(1-head_size);
00502     int y_center = ys*head_size + ye*(1-head_size);
00503
00504     MAT2D_AT(v1, 0, 0) = x_center;
00505     MAT2D_AT(v1, 1, 0) = y_center;
00506     mat2D_copy(v2, v1);
00507
00508     /* v1 */
00509     mat2D_copy(temp_v, v1);
00510     mat2D_sub(temp_v, pe);
00511     mat2D_fill(v1, 0);
00512     mat2D_dot(v1, DCM_p, temp_v);
00513     mat2D_add(v1, pe);
00514
00515     /* v2 */
00516     mat2D_copy(temp_v, v2);
00517     mat2D_sub(temp_v, pe);
00518     mat2D_fill(v2, 0);
00519     mat2D_dot(v2, DCM_m, temp_v);
00520     mat2D_add(v2, pe);
00521
00522     adl_line_draw(screen_mat, MAT2D_AT(v1, 0, 0), MAT2D_AT(v1, 1, 0), xe, ye, color,
00523     offset_zoom_param);
00524     adl_line_draw(screen_mat, MAT2D_AT(v2, 0, 0), MAT2D_AT(v2, 1, 0), xe, ye, color,
00525     offset_zoom_param);
00526     adl_line_draw(screen_mat, xs, ys, xe, ye, color, offset_zoom_param);
00527
00528     mat2D_free(pe);
00529     mat2D_free(v1);
00530     mat2D_free(v2);
00531     mat2D_free(temp_v);
00532     mat2D_free(DCM_p);
00533     mat2D_free(DCM_m);

```

```

00568 }
00569
00586 void adl_character_draw(Mat2D_uint32 screen_mat, char c, int width_pixel, int height_pixel, int
00587 { x_top_left, int y_top_left, uint32_t color, Offset_zoom_param offset_zoom_param)
00588     switch (c)
00589     {
00590         case 'a':
00591         case 'A':
00592             adl_line_draw(screen_mat, x_top_left, y_top_left+height_pixel, x_top_left+width_pixel/2,
00593                         y_top_left, color, offset_zoom_param);
00594             adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+width_pixel,
00595                         y_top_left+height_pixel, color, offset_zoom_param);
00596             adl_line_draw(screen_mat, x_top_left+width_pixel/6, y_top_left+2*height_pixel/3,
00597                         x_top_left+5*width_pixel/6, y_top_left+2*height_pixel/3, color, offset_zoom_param);
00598             break;
00599         case 'b':
00600         case 'B':
00601             adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+height_pixel, color,
00602                         offset_zoom_param);
00603             adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+2*width_pixel/3, y_top_left,
00604                         color, offset_zoom_param);
00605             adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel,
00606                         y_top_left+height_pixel/6, color, offset_zoom_param);
00607             adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+height_pixel/6,
00608                         x_top_left+width_pixel, y_top_left+height_pixel/3, color, offset_zoom_param);
00609             adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+height_pixel/3,
00610                         x_top_left+2*width_pixel/3, y_top_left+height_pixel/2, color, offset_zoom_param);
00611             adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+height_pixel/2,
00612                         x_top_left, y_top_left+height_pixel/2, color, offset_zoom_param);
00613             adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+height_pixel/2,
00614                         x_top_left+width_pixel, y_top_left+2*height_pixel/3, color, offset_zoom_param);
00615             adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+2*height_pixel/3,
00616                         x_top_left+width_pixel, y_top_left+5*height_pixel/6, color, offset_zoom_param);
00617             adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*height_pixel/6,
00618                         x_top_left+width_pixel/3, y_top_left+height_pixel, color, offset_zoom_param);
00619             break;
00620         case 'c':
00621         case 'C':
00622             adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left+width_pixel/3,
00623                         y_top_left, color, offset_zoom_param);
00624             adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
00625                         y_top_left+height_pixel/6, color, offset_zoom_param);
00626             adl_line_draw(screen_mat, x_top_left, y_top_left+height_pixel/6, x_top_left,
00627                         y_top_left+5*height_pixel/6, color, offset_zoom_param);
00628             adl_line_draw(screen_mat, x_top_left, y_top_left+5*height_pixel/6, x_top_left+width_pixel/3,
00629                         y_top_left+height_pixel, color, offset_zoom_param);
00630             adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+height_pixel,
00631                         x_top_left+width_pixel, y_top_left+height_pixel, color, offset_zoom_param);
00632             adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+height_pixel,
00633                         x_top_left+width_pixel, y_top_left+height_pixel, color, offset_zoom_param);
00634             adl_line_draw(screen_mat, x_top_left, y_top_left+height_pixel/2, x_top_left+width_pixel,
00635                         y_top_left+height_pixel/2, color, offset_zoom_param);
00636             break;
00637         case 'f':
00638         case 'F':
00639             adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left, y_top_left, color,
00640                         offset_zoom_param);
00641             adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+height_pixel, color,
00642                         offset_zoom_param);

```

```

00640
00641     adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
00642         y_top_left+hight_pixel/2, color, offset_zoom_param);
00643     break;
00644     case 'g':
00645     case 'G':
00646         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
00647             x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00648         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
00649             y_top_left, color, offset_zoom_param);
00650         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
00651             y_top_left+hight_pixel/6, color, offset_zoom_param);
00652         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left+width_pixel/3,
00653             y_top_left+hight_pixel, color, offset_zoom_param);
00654         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
00655             x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00656         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
00657             x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00658         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
00659             y_top_left+hight_pixel, color, offset_zoom_param);
00660         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel/2,
00661             x_top_left+width_pixel/2, y_top_left+hight_pixel/2, color, offset_zoom_param);
00662         break;
00663     case 'h':
00664     case 'H':
00665         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
00666             offset_zoom_param);
00667         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left+width_pixel,
00668             y_top_left+hight_pixel, color, offset_zoom_param);
00669         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
00670             y_top_left+hight_pixel/2, color, offset_zoom_param);
00671         break;
00672     case 'i':
00673     case 'I':
00674         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
00675             offset_zoom_param);
00676         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
00677             y_top_left+hight_pixel, color, offset_zoom_param);
00678         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+width_pixel/2,
00679             y_top_left+hight_pixel, color, offset_zoom_param);
00680         break;
00681     case 'j':
00682     case 'J':
00683         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
00684             offset_zoom_param);
00685         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
00686             y_top_left+hight_pixel, color, offset_zoom_param);
00687         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
00688             y_top_left, color, offset_zoom_param);
00689         break;
00690     case 'k':
00691     case 'K':
00692         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left, y_top_left, color,
00693             offset_zoom_param);
00694         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel/2,
00695             y_top_left+hight_pixel, color, offset_zoom_param);
00696         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left+hight_pixel,
00697             x_top_left+width_pixel, y_top_left, color, offset_zoom_param);
00698         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left+width_pixel,
00699             y_top_left+hight_pixel, color, offset_zoom_param);
00700         break;
00701     case 'l':
00702     case 'L':
00703         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
00704             offset_zoom_param);
00705         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
00706             y_top_left+hight_pixel, color, offset_zoom_param);
00707         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, color, offset_zoom_param);
00708         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left+width_pixel,
00709             y_top_left+hight_pixel, color, offset_zoom_param);
00710         break;
00711     case 'm':
00712     case 'M':
00713         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left, color,
00714             offset_zoom_param);
00715         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel/2,
00716             y_top_left+hight_pixel, color, offset_zoom_param);
00717         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left+hight_pixel,
00718             x_top_left+width_pixel, y_top_left, color, offset_zoom_param);
00719         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left+width_pixel,
00720             y_top_left+hight_pixel, color, offset_zoom_param);
00721         break;
00722     case 'n':
00723     case 'N':
00724         adl_line_draw(screen_mat, x_top_left, y_top_left, y_top_left+hight_pixel, x_top_left, y_top_left, color,
00725             offset_zoom_param);

```

```

00696     adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel,
00697     y_top_left+hight_pixel, color, offset_zoom_param);
00698     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel,
00699     x_top_left+width_pixel, y_top_left, color, offset_zoom_param);
00700     break;
00701     case 'o':
00702     case 'O':
00703         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
00704         y_top_left, color, offset_zoom_param);
00705         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
00706         y_top_left+hight_pixel/6, color, offset_zoom_param);
00707         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
00708         y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00709         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
00710         y_top_left+hight_pixel, color, offset_zoom_param);
00711         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
00712         x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00713         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
00714         x_top_left, y_top_left, color, offset_zoom_param);
00715         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel,
00716         y_top_left+hight_pixel/6, color, offset_zoom_param);
00717         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
00718         x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00719         break;
00720     case 'p':
00721     case 'P':
00722         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
00723         offset_zoom_param);
00724         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+2*width_pixel/3, y_top_left,
00725         color, offset_zoom_param);
00726         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel,
00727         y_top_left+hight_pixel/6, color, offset_zoom_param);
00728         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6, x_top_left,
00729         y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00730         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+width_pixel/3,
00731         y_top_left+hight_pixel, color, offset_zoom_param);
00732         break;
00733     case 'q':
00734     case 'Q':
00735         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
00736         y_top_left, color, offset_zoom_param);
00737         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left, y_top_left+hight_pixel,
00738         color, offset_zoom_param);
00739         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6, x_top_left,
00740         y_top_left+width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00741         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, x_top_left,
00742         y_top_left+hight_pixel/2, color, offset_zoom_param);
00743         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
00744         x_top_left+width_pixel, y_top_left+hight_pixel, color, offset_zoom_param);
00745         break;
00746     case 's':
00747     case 'S':
00748         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
00749         x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00750         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
00751         y_top_left, color, offset_zoom_param);

```

```

00749     adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
00750                 y_top_left+hight_pixel/6, color, offset_zoom_param);
00751     adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
00752                 y_top_left+hight_pixel/3, color, offset_zoom_param);
00753     adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/3, x_top_left+width_pixel/3,
00754                 y_top_left+hight_pixel/2, color, offset_zoom_param);
00755     adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel/2,
00756                 x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00757     adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
00758                 x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00759     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+2*hight_pixel/3,
00760                 x_top_left+width_pixel, y_top_left+2*hight_pixel/6, color, offset_zoom_param);
00761     break;
00762     case 't':
00763     case 'T':
00764         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
00765                     offset_zoom_param);
00766         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+width_pixel/2,
00767                     y_top_left+hight_pixel, color, offset_zoom_param);
00768         break;
00769     case 'u':
00770     case 'U':
00771         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel/6, color,
00772                     offset_zoom_param);
00773         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
00774                     y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00775         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
00776                     y_top_left+hight_pixel, color, offset_zoom_param);
00777         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
00778                     x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00779         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/6,
00780                     x_top_left+width_pixel, y_top_left+hight_pixel/6, color, offset_zoom_param);
00781         break;
00782     case 'v':
00783     case 'V':
00784         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel/2,
00785                     y_top_left+hight_pixel, color, offset_zoom_param);
00786         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left+hight_pixel,
00787                     x_top_left+width_pixel, y_top_left, color, offset_zoom_param);
00788         break;
00789     case 'w':
00790     case 'W':
00791         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel/3,
00792                     y_top_left+hight_pixel, color, offset_zoom_param);
00793         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
00794                     x_top_left+width_pixel/2, y_top_left, color, offset_zoom_param);
00795         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+2*width_pixel/3,
00796                     y_top_left+hight_pixel, color, offset_zoom_param);
00797         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
00798                     x_top_left+width_pixel, y_top_left, color, offset_zoom_param);
00799         break;
00800     case 'x':
00801     case 'X':
00802         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel,
00803                     y_top_left+hight_pixel, color, offset_zoom_param);
00804         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
00805                     y_top_left, color, offset_zoom_param);
00806         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left,
00807                     y_top_left+hight_pixel, color, offset_zoom_param);
00808         break;
00809     case '.':

```

```

00805     adl_rectangle_fill_min_max(screen_mat, x_top_left+width_pixel/6, x_top_left+width_pixel/3,
00806         y_top_left+5*hight_pixel/6, y_top_left+hight_pixel, color, offset_zoom_param);
00807     break;
00808     case ':':
00809         adl_rectangle_fill_min_max(screen_mat, x_top_left+width_pixel/6, x_top_left+width_pixel/3,
00810             y_top_left+5*hight_pixel/6, y_top_left+hight_pixel, color, offset_zoom_param);
00811         adl_rectangle_fill_min_max(screen_mat, x_top_left+width_pixel/6, x_top_left+width_pixel/3,
00812             y_top_left, y_top_left+hight_pixel/6, color, offset_zoom_param);
00813         break;
00814     case '0':
00815         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
00816             y_top_left, color, offset_zoom_param);
00817         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
00818             y_top_left+hight_pixel/6, color, offset_zoom_param);
00819         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
00820             y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00821         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
00822             y_top_left+hight_pixel, color, offset_zoom_param);
00823         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
00824             x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00825         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
00826             x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00827         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6, x_top_left,
00828             y_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00829         break;
00830     case '1':
00831         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left+width_pixel/2,
00832             y_top_left, color, offset_zoom_param);
00833         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+width_pixel/2,
00834             y_top_left+hight_pixel, color, offset_zoom_param);
00835         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
00836             y_top_left+hight_pixel, color, offset_zoom_param);
00837         break;
00838     case '2':
00839         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left+width_pixel/3,
00840             y_top_left, color, offset_zoom_param);
00841         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left+2*width_pixel/3,
00842             y_top_left, color, offset_zoom_param);
00843         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel,
00844             y_top_left+hight_pixel, color, offset_zoom_param);
00845         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/2,
00846             x_top_left+width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00847         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/2,
00848             x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00849         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
00850             x_top_left+width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00851         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel, x_top_left,
00852             y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00853         break;
00854     case '3':
00855         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
00856             x_top_left+width_pixel/3, y_top_left+hight_pixel/3, color, offset_zoom_param);
00857         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel/3,
00858             x_top_left+2*width_pixel/3, y_top_left+hight_pixel/3, color, offset_zoom_param);
00859         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/3,
00860             x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00861         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
00862             x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00863         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
00864             x_top_left+width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00865         break;
00866     case '4':
00867         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
00868             x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00869         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left,
00870             y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00871         adl_line_draw(screen_mat, x_top_left, y_top_left+2*hight_pixel/3, x_top_left+width_pixel,
00872             y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00873         break;
00874     case '5':
00875         break;

```

```

00857     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left, y_top_left, color,
00858     offset_zoom_param);
00859     adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel/2, color,
00860     offset_zoom_param);
00861     adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+2*width_pixel/3,
00862     y_top_left+hight_pixel/2, color, offset_zoom_param);
00863     adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
00864     x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00865     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+2*hight_pixel/3,
00866     x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00867     break;
00868     case '6':
00869     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
00870     x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00871     adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
00872     y_top_left, color, offset_zoom_param);
00873     adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
00874     y_top_left+hight_pixel/6, color, offset_zoom_param);
00875     adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel/6,
00876     x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00877     adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
00878     x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00879     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
00880     x_top_left+width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00881     adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel/2, x_top_left,
00882     y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00883     break;
00884     case '7':
00885     adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
00886     offset_zoom_param);
00887     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left+width_pixel/3,
00888     y_top_left+hight_pixel, color, offset_zoom_param);
00889     break;
00890     case '8':
00891     adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
00892     x_top_left+width_pixel, y_top_left+hight_pixel/3, color, offset_zoom_param);
00893     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/3,
00894     x_top_left+width_pixel, y_top_left+hight_pixel/6, color, offset_zoom_param);
00895     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
00896     x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00897     adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left,
00898     y_top_left+hight_pixel/6, color, offset_zoom_param);
00899     adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel/3,
00900     y_top_left+hight_pixel, color, offset_zoom_param);
00901     adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
00902     x_top_left+2*width_pixel/3, y_top_left+hight_pixel/3, color, offset_zoom_param);
00903     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel,
00904     x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00905     break;
00906     case '9':
00907     adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
00908     y_top_left+hight_pixel, color, offset_zoom_param);
00909     adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
00910     x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);

```

```

00907     adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
00908     x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00909     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6, color, offset_zoom_param);
00910     adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
00911     x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00912     adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
00913     y_top_left, color, offset_zoom_param);
00914     adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
00915     y_top_left+hight_pixel/6, color, offset_zoom_param);
00916     adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left+width_pixel/3,
00917     y_top_left+hight_pixel/3, color, offset_zoom_param);
00918     adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel/2,
00919     x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00920     adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
00921     x_top_left+width_pixel, y_top_left+hight_pixel/3, color, offset_zoom_param);
00922     break;
00923     case '-':
00924     adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
00925     y_top_left+hight_pixel/2, color, offset_zoom_param);
00926     break;
00927     case '+':
00928     adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
00929     y_top_left+hight_pixel/2, color, offset_zoom_param);
00930     adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+width_pixel/2,
00931     y_top_left+hight_pixel, color, offset_zoom_param);
00932     break;
00933 }
00934 void adl_sentence_draw(Mat2D_uint32 screen_mat, const char sentence[], size_t len, const int
00935 x_top_left, const int y_top_left, const int hight_pixel, const uint32_t color, Offset_zoom_param
00936 offset_zoom_param)
00937 {
00938     int character_width_pixel = hight_pixel/2;
00939     int current_x_top_left = x_top_left;
00940     int character_x_offset = (int)fmaxf(fminf(ADL_MAX_CHARACTER_OFFSET, character_width_pixel / 5),
00941     ADL_MIN_CHARACTER_OFFSET);
00942     for (size_t char_index = 0; char_index < len; char_index++) {
00943         adl_character_draw(screen_mat, sentence[char_index], character_width_pixel, hight_pixel,
00944         current_x_top_left, y_top_left, color, offset_zoom_param);
00945         current_x_top_left += character_width_pixel + character_x_offset;
00946     }
00947 }
00948 void adl_rectangle_draw_min_max(Mat2D_uint32 screen_mat, int min_x, int max_x, int min_y, int max_y,
00949     uint32_t color, Offset_zoom_param offset_zoom_param)
00950 {
00951     adl_line_draw(screen_mat, min_x, min_y, max_x, min_y, color, offset_zoom_param);
00952     adl_line_draw(screen_mat, min_x, max_y, max_x, max_y, color, offset_zoom_param);
00953     adl_line_draw(screen_mat, min_x, min_y, min_x, max_y, color, offset_zoom_param);
00954     adl_line_draw(screen_mat, max_x, min_y, max_x, max_y, color, offset_zoom_param);
00955 }
00956 void adl_rectangle_fill_min_max(Mat2D_uint32 screen_mat, int min_x, int max_x, int min_y, int max_y,
00957     uint32_t color, Offset_zoom_param offset_zoom_param)
00958 {
00959     for (int y = min_y; y <= max_y; y++) {
00960         adl_line_draw(screen_mat, min_x, y, max_x, y, color, offset_zoom_param);
00961     }
00962 }
00963 void adl_quad_draw(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
00964     Offset_zoom_param offset_zoom_param)
00965 {
00966     (void)inv_z_buffer;
00967     adl_lines_loop_draw(screen_mat, quad.points, 4, color, offset_zoom_param);
00968 }
00969 void adl_quad_fill(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
00970     Offset_zoom_param offset_zoom_param)
00971 {
00972     Point p0 = quad.points[0];

```

```

01031     Point p1 = quad.points[1];
01032     Point p2 = quad.points[2];
01033     Point p3 = quad.points[3];
01034
01035     int x_min = fminf(p0.x, fminf(p1.x, fminf(p2.x, p3.x)));
01036     int x_max = fmaxf(p0.x, fmaxf(p1.x, fmaxf(p2.x, p3.x)));
01037     int y_min = fminf(p0.y, fminf(p1.y, fminf(p2.y, p3.y)));
01038     int y_max = fmaxf(p0.y, fmaxf(p1.y, fmaxf(p2.y, p3.y)));
01039
01040     if (x_min < 0) x_min = 0;
01041     if (y_min < 0) y_min = 0;
01042     if (x_max >= (int)screen_mat.cols) x_max = (int)screen_mat.cols - 1;
01043     if (y_max >= (int)screen_mat.rows) y_max = (int)screen_mat.rows - 1;
01044
01045     float w = edge_cross_point(p0, p1, p1, p2) + edge_cross_point(p2, p3, p3, p0);
01046     if (fabs(w) < 1e-6) {
01047         // adl_quad_draw(screen_mat, inv_z_buffer, quad, quad.colors[0], offset_zoom_param);
01048         return;
01049     }
01050
01051     float size_p3_to_p0 = sqrt((p0.x - p3.x)*(p0.x - p3.x) + (p0.y - p3.y)*(p0.y - p3.y));
01052     float size_p0_to_p1 = sqrt((p1.x - p0.x)*(p1.x - p0.x) + (p1.y - p0.y)*(p1.y - p0.y));
01053     float size_p1_to_p2 = sqrt((p2.x - p1.x)*(p2.x - p1.x) + (p2.y - p1.y)*(p2.y - p1.y));
01054     float size_p2_to_p3 = sqrt((p3.x - p2.x)*(p3.x - p2.x) + (p3.y - p2.y)*(p3.y - p2.y));
01055
01056     int r, g, b, a;
01057     HexARGB_RGBA_VAR(color, r, g, b, a);
01058     float light_intensity = (quad.light_intensity[0] + quad.light_intensity[1] +
01059     quad.light_intensity[2] + quad.light_intensity[3]) / 4;
01060     uint8_t base_r = (uint8_t)fmaxf(0, fminf(255, r * light_intensity));
01061     uint8_t base_g = (uint8_t)fmaxf(0, fminf(255, g * light_intensity));
01062     uint8_t base_b = (uint8_t)fmaxf(0, fminf(255, b * light_intensity));
01063
01064     for (int y = y_min; y <= y_max; y++) {
01065         for (int x = x_min; x <= x_max; x++) {
01066             Point p = { .x = x, .y = y, .z = 0 };
01067             bool in_01, in_12, in_23, in_30;
01068
01069             in_01 = (edge_cross_point(p0, p1, p0, p) >= 0) != (w < 0);
01070             in_12 = (edge_cross_point(p1, p2, p1, p) >= 0) != (w < 0);
01071             in_23 = (edge_cross_point(p2, p3, p2, p) >= 0) != (w < 0);
01072             in_30 = (edge_cross_point(p3, p0, p3, p) >= 0) != (w < 0);
01073
01074             /* https://www.mn.uio.no/math/english/people/aca/michaelf/papers/mv3d.pdf. */
01075             float size_p_to_p0 = sqrt((p0.x - p.x)*(p0.x - p.x) + (p0.y - p.y)*(p0.y - p.y));
01076             float size_p_to_p1 = sqrt((p1.x - p.x)*(p1.x - p.x) + (p1.y - p.y)*(p1.y - p.y));
01077             float size_p_to_p2 = sqrt((p2.x - p.x)*(p2.x - p.x) + (p2.y - p.y)*(p2.y - p.y));
01078             float size_p_to_p3 = sqrt((p3.x - p.x)*(p3.x - p.x) + (p3.y - p.y)*(p3.y - p.y));
01079
01080             /* tangent of half the angle directly using vector math */
01081             float tan_theta_3_over_2 = size_p3_to_p0 / (size_p_to_p3 + size_p_to_p0);
01082             float tan_theta_0_over_2 = size_p0_to_p1 / (size_p_to_p0 + size_p_to_p1);
01083             float tan_theta_1_over_2 = size_p1_to_p2 / (size_p_to_p1 + size_p_to_p2);
01084             float tan_theta_2_over_2 = size_p2_to_p3 / (size_p_to_p2 + size_p_to_p3);
01085             float w0 = (tan_theta_3_over_2 + tan_theta_0_over_2) / size_p_to_p0;
01086             float w1 = (tan_theta_0_over_2 + tan_theta_1_over_2) / size_p_to_p1;
01087             float w2 = (tan_theta_1_over_2 + tan_theta_2_over_2) / size_p_to_p2;
01088             float w3 = (tan_theta_2_over_2 + tan_theta_3_over_2) / size_p_to_p3;
01089
01090             float inv_w_tot = 1.0f / (w0 + w1 + w2 + w3);
01091             float alpha = w0 * inv_w_tot;
01092             float beta = w1 * inv_w_tot;
01093             float gamma = w2 * inv_w_tot;
01094             float delta = w3 * inv_w_tot;
01095
01096             if (in_01 && in_12 && in_23 && in_30) {
01097
01098                 double inv_w = alpha * (1.0f / p0.w) + beta * (1.0f / p1.w) + gamma * (1.0f / p2.w) +
01099                 delta * (1.0f / p3.w);
01100                 double z_over_w = alpha * (p0.z / p0.w) + beta * (p1.z / p1.w) + gamma * (p2.z /
01101                 p2.w) + delta * (p3.z / p3.w);
01102                 double inv_z = inv_w / z_over_w;
01103
01104                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01105                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(base_r, base_g, base_b, a),
01106                     offset_zoom_param);
01107                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01108                 }
01109             }
01110
01111             void adl_quad_fill_interpolate_normal_mean_value(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad
01112             quad, uint32_t color, Offset_zoom_param offset_zoom_param)
01113             {
01114                 Point p0 = quad.points[0];

```

```

01125     Point p1 = quad.points[1];
01126     Point p2 = quad.points[2];
01127     Point p3 = quad.points[3];
01128
01129     int x_min = fminf(p0.x, fminf(p1.x, fminf(p2.x, p3.x)));
01130     int x_max = fmaxf(p0.x, fmaxf(p1.x, fmaxf(p2.x, p3.x)));
01131     int y_min = fminf(p0.y, fminf(p1.y, fminf(p2.y, p3.y)));
01132     int y_max = fmaxf(p0.y, fmaxf(p1.y, fmaxf(p2.y, p3.y)));
01133
01134     if (x_min < 0) x_min = 0;
01135     if (y_min < 0) y_min = 0;
01136     if (x_max >= (int)screen_mat.cols) x_max = (int)screen_mat.cols - 1;
01137     if (y_max >= (int)screen_mat.rows) y_max = (int)screen_mat.rows - 1;
01138
01139     float w = edge_cross_point(p0, p1, p1, p2) + edge_cross_point(p2, p3, p3, p0);
01140     if (fabs(w) < 1e-6) {
01141         // adl_quad_draw(screen_mat, inv_z_buffer, quad, quad.colors[0], offset_zoom_param);
01142         return;
01143     }
01144
01145     int r, g, b, a;
01146     HexARGB_RGBA_VAR(color, r, g, b, a);
01147
01148     for (int y = y_min; y <= y_max; y++) {
01149         for (int x = x_min; x <= x_max; x++) {
01150             Point p = {.x = x, .y = y, .z = 0};
01151             bool in_01, in_12, in_23, in_30;
01152
01153             in_01 = (edge_cross_point(p0, p1, p0, p) >= 0) != (w < 0);
01154             in_12 = (edge_cross_point(p1, p2, p1, p) >= 0) != (w < 0);
01155             in_23 = (edge_cross_point(p2, p3, p2, p) >= 0) != (w < 0);
01156             in_30 = (edge_cross_point(p3, p0, p3, p) >= 0) != (w < 0);
01157
01158             /* using 'mean value coordinates'
01159             * https://www.mn.uio.no/math/english/people/aca/michaelf/papers/mv3d.pdf. */
01160             float size_p_to_p0 = sqrt((p0.x - p.x)*(p0.x - p.x) + (p0.y - p.y)*(p0.y - p.y));
01161             float size_p_to_p1 = sqrt((p1.x - p.x)*(p1.x - p.x) + (p1.y - p.y)*(p1.y - p.y));
01162             float size_p_to_p2 = sqrt((p2.x - p.x)*(p2.x - p.x) + (p2.y - p.y)*(p2.y - p.y));
01163             float size_p_to_p3 = sqrt((p3.x - p.x)*(p3.x - p.x) + (p3.y - p.y)*(p3.y - p.y));
01164
01165             /* calculating the tangent of half the angle directly using vector math */
01166             float t0 = adl_tan_half_angle(p0, p1, p, size_p_to_p0, size_p_to_p1);
01167             float t1 = adl_tan_half_angle(p1, p2, p, size_p_to_p1, size_p_to_p2);
01168             float t2 = adl_tan_half_angle(p2, p3, p, size_p_to_p2, size_p_to_p3);
01169             float t3 = adl_tan_half_angle(p3, p0, p, size_p_to_p3, size_p_to_p0);
01170
01171             float w0 = (t3 + t0) / size_p_to_p0;
01172             float w1 = (t0 + t1) / size_p_to_p1;
01173             float w2 = (t1 + t2) / size_p_to_p2;
01174             float w3 = (t2 + t3) / size_p_to_p3;
01175
01176             float inv_w_tot = 1.0f / (w0 + w1 + w2 + w3);
01177             float alpha = w0 * inv_w_tot;
01178             float beta = w1 * inv_w_tot;
01179             float gamma = w2 * inv_w_tot;
01180             float delta = w3 * inv_w_tot;
01181
01182             if (in_01 && in_12 && in_23 && in_30) {
01183                 float light_intensity = quad.light_intensity[0]*alpha + quad.light_intensity[1]*beta +
01184                 quad.light_intensity[2]*gamma + quad.light_intensity[3]*delta;
01185
01186                 float rf = r * light_intensity;
01187                 float gf = g * light_intensity;
01188                 float bf = b * light_intensity;
01189                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01190                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01191                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01192
01193                 double inv_w = alpha * (1.0f / p0.w) + beta * (1.0f / p1.w) + gamma * (1.0f / p2.w) +
01194                 delta * (1.0f / p3.w);
01195                 double z_over_w = alpha * (p0.z / p0.w) + beta * (p1.z / p1.w) + gamma * (p2.z /
01196                 p2.w) + delta * (p3.z / p3.w);
01197                 double inv_z = inv_w / z_over_w;
01198
01199                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01200                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, a), offset_zoom_param);
01201                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01202                 }
01203             }
01204
01205             void adl_quad_fill_interpolate_color_mean_value(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad
01206             quad, Offset_zoom_param offset_zoom_param)
01207             {
01208                 Point p0 = quad.points[0];

```

```

01219     Point p1 = quad.points[1];
01220     Point p2 = quad.points[2];
01221     Point p3 = quad.points[3];
01222
01223     int x_min = fminf(p0.x, fminf(p1.x, fminf(p2.x, p3.x)));
01224     int x_max = fmaxf(p0.x, fmaxf(p1.x, fmaxf(p2.x, p3.x)));
01225     int y_min = fminf(p0.y, fminf(p1.y, fminf(p2.y, p3.y)));
01226     int y_max = fmaxf(p0.y, fmaxf(p1.y, fmaxf(p2.y, p3.y)));
01227
01228     if (x_min < 0) x_min = 0;
01229     if (y_min < 0) y_min = 0;
01230     if (x_max >= (int)screen_mat.cols) x_max = (int)screen_mat.cols - 1;
01231     if (y_max >= (int)screen_mat.rows) y_max = (int)screen_mat.rows - 1;
01232
01233     float w = edge_cross_point(p0, p1, p1, p2) + edge_cross_point(p2, p3, p3, p0);
01234     if (fabs(w) < 1e-6) {
01235         // adl_quad_draw(screen_mat, inv_z_buffer, quad, quad.colors[0], offset_zoom_param);
01236         return;
01237     }
01238
01239     for (int y = y_min; y <= y_max; y++) {
01240         for (int x = x_min; x <= x_max; x++) {
01241             Point p = { .x = x, .y = y, .z = 0 };
01242             bool in_01, in_12, in_23, in_30;
01243
01244             in_01 = (edge_cross_point(p0, p1, p0, p) >= 0) != (w < 0);
01245             in_12 = (edge_cross_point(p1, p2, p1, p) >= 0) != (w < 0);
01246             in_23 = (edge_cross_point(p2, p3, p2, p) >= 0) != (w < 0);
01247             in_30 = (edge_cross_point(p3, p0, p3, p) >= 0) != (w < 0);
01248
01249             /* using 'mean value coordinates'
01250             * https://www.mn.uio.no/math/english/people/aca/michaelf/papers/mv3d.pdf. */
01251             float size_p_to_p0 = sqrt((p0.x - p.x)*(p0.x - p.x) + (p0.y - p.y)*(p0.y - p.y));
01252             float size_p_to_p1 = sqrt((p1.x - p.x)*(p1.x - p.x) + (p1.y - p.y)*(p1.y - p.y));
01253             float size_p_to_p2 = sqrt((p2.x - p.x)*(p2.x - p.x) + (p2.y - p.y)*(p2.y - p.y));
01254             float size_p_to_p3 = sqrt((p3.x - p.x)*(p3.x - p.x) + (p3.y - p.y)*(p3.y - p.y));
01255
01256             /* calculating the tangent of half the angle directly using vector math */
01257             float t0 = adl_tan_half_angle(p0, p1, p, size_p_to_p0, size_p_to_p1);
01258             float t1 = adl_tan_half_angle(p1, p2, p, size_p_to_p1, size_p_to_p2);
01259             float t2 = adl_tan_half_angle(p2, p3, p, size_p_to_p2, size_p_to_p3);
01260             float t3 = adl_tan_half_angle(p3, p0, p, size_p_to_p3, size_p_to_p0);
01261
01262             float w0 = (t3 + t0) / size_p_to_p0;
01263             float w1 = (t0 + t1) / size_p_to_p1;
01264             float w2 = (t1 + t2) / size_p_to_p2;
01265             float w3 = (t2 + t3) / size_p_to_p3;
01266
01267             float inv_w_tot = 1.0f / (w0 + w1 + w2 + w3);
01268             float alpha = w0 * inv_w_tot;
01269             float beta = w1 * inv_w_tot;
01270             float gamma = w2 * inv_w_tot;
01271             float delta = w3 * inv_w_tot;
01272
01273             if (in_01 && in_12 && in_23 && in_30) {
01274                 int r0, g0, b0, a0;
01275                 int r1, g1, b1, a1;
01276                 int r2, g2, b2, a2;
01277                 int r3, g3, b3, a3;
01278                 HexRGB_RGBA_VAR(quad.colors[0], r0, g0, b0, a0);
01279                 HexRGB_RGBA_VAR(quad.colors[1], r1, g1, b1, a1);
01280                 HexRGB_RGBA_VAR(quad.colors[2], r2, g2, b2, a2);
01281                 HexRGB_RGBA_VAR(quad.colors[3], r3, g3, b3, a3);
01282
01283                 uint8_t current_r = r0*alpha + r1*beta + r2*gamma + r3*delta;
01284                 uint8_t current_g = g0*alpha + g1*beta + g2*gamma + g3*delta;
01285                 uint8_t current_b = b0*alpha + b1*beta + b2*gamma + b3*delta;
01286                 uint8_t current_a = a0*alpha + a1*beta + a2*gamma + a3*delta;
01287
01288                 float light_intensity = (quad.light_intensity[0] + quad.light_intensity[1] +
01289                     quad.light_intensity[2] + quad.light_intensity[3]) / 4;
01290                 float rf = current_r * light_intensity;
01291                 float gf = current_g * light_intensity;
01292                 float bf = current_b * light_intensity;
01293                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01294                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01295                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01296
01297                 double inv_w = alpha * (1.0f / p0.w) + beta * (1.0f / p1.w) + gamma * (1.0f / p2.w) +
01298                     delta * (1.0f / p3.w);
01299                 double z_over_w = alpha * (p0.z / p0.w) + beta * (p1.z / p1.w) + gamma * (p2.z /
01300                     p2.w) + delta * (p3.z / p3.w);
01301                 double inv_z = inv_w / z_over_w;
01302
01303                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01304                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, current_a),
01305                     offset_zoom_param);

```

```

01302             MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01303         }
01304     }
01305 }
01306 }
01307 }
01308
01320 void adl_quad_mesh_draw(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Quad_mesh mesh, uint32_t
01321 color, Offset_zoom_param offset_zoom_param)
01322 {
01323     for (size_t i = 0; i < mesh.length; i++) {
01324         Quad quad = mesh.elements[i];
01325         /* Reject invalid quad */
01326         adl_assert_quad_is_valid(quad);
01327
01328         if (!quad.to_draw) continue;
01329         adl_quad_draw(screen_mat, inv_z_buffer_mat, quad, color, offset_zoom_param);
01330     }
01331 }
01332
01344 void adl_quad_mesh_fill(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Quad_mesh mesh, uint32_t
01345 color, Offset_zoom_param offset_zoom_param)
01346 {
01347     for (size_t i = 0; i < mesh.length; i++) {
01348         Quad quad = mesh.elements[i];
01349         /* Reject invalid quad */
01350         adl_assert_quad_is_valid(quad);
01351
01352         if (!quad.to_draw) continue;
01353
01354         // color = rand_double() * 0xFFFFFFFF;
01355         adl_quad_fill(screen_mat, inv_z_buffer_mat, quad, color, offset_zoom_param);
01356     }
01357 }
01358
01371 void adl_quad_mesh_fill_interpolate_normal(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Quad_mesh
01372 mesh, uint32_t color, Offset_zoom_param offset_zoom_param)
01373 {
01374     for (size_t i = 0; i < mesh.length; i++) {
01375         Quad quad = mesh.elements[i];
01376         /* Reject invalid quad */
01377         adl_assert_quad_is_valid(quad);
01378
01379         uint8_t a, r, g, b;
01380         HexARGB_RGBA_VAR(color, a, r, g, b);
01381         (void)r;
01382         (void)g;
01383         (void)b;
01384
01385         if (!quad.to_draw && a == 255) continue;
01386
01387         adl_quad_fill_interpolate_normal_mean_value(screen_mat, inv_z_buffer_mat, quad, color,
01388             offset_zoom_param);
01389     }
01401 void adl_quad_mesh_fill_interpolate_color(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Quad_mesh
01402 mesh, Offset_zoom_param offset_zoom_param)
01403 {
01404     for (size_t i = 0; i < mesh.length; i++) {
01405         Quad quad = mesh.elements[i];
01406         /* Reject invalid quad */
01407         adl_assert_quad_is_valid(quad);
01408
01409         if (!quad.to_draw) continue;
01410
01411         adl_quad_fill_interpolate_color_mean_value(screen_mat, inv_z_buffer_mat, quad,
01412             offset_zoom_param);
01413     }
01427 void adl_circle_draw(Mat2D_uint32 screen_mat, float center_x, float center_y, float r, uint32_t color,
01428 Offset_zoom_param offset_zoom_param)
01429 {
01430     for (int dy = -r; dy <= r; dy++) {
01431         for (int dx = -r; dx <= r; dx++) {
01432             float diff = dx * dx + dy * dy - r * r;
01433             if (diff < 0 && diff > -r*2) {
01434                 adl_point_draw(screen_mat, center_x + dx, center_y + dy, color, offset_zoom_param);
01435             }
01436         }
01437     }
01438
01449 void adl_circle_fill(Mat2D_uint32 screen_mat, float center_x, float center_y, float r, uint32_t color,

```

```

    Offset_zoom_param offset_zoom_param)
01450 {
01451     for (int dy = -r; dy <= r; dy++) {
01452         for (int dx = -r; dx <= r; dx++) {
01453             float diff = dx * dx + dy * dy - r * r;
01454             if (diff < 0) {
01455                 adl_point_draw(screen_mat, center_x + dx, center_y + dy, color, offset_zoom_param);
01456             }
01457         }
01458     }
01459 }
01460
01461 void adl_tri_draw(Mat2D_uint32 screen_mat, Tri tri, uint32_t color, Offset_zoom_param
01462 offset_zoom_param)
01463 {
01464     adl_line_draw(screen_mat, tri.points[0].x, tri.points[0].y, tri.points[1].x, tri.points[1].y,
01465 color, offset_zoom_param);
01466     adl_line_draw(screen_mat, tri.points[1].x, tri.points[1].y, tri.points[2].x, tri.points[2].y,
01467 color, offset_zoom_param);
01468     adl_line_draw(screen_mat, tri.points[2].x, tri.points[2].y, tri.points[0].x, tri.points[0].y,
01469 color, offset_zoom_param);
01470
01471     // adl_draw_arrow(screen_mat, tri.points[0].x, tri.points[0].y, tri.points[1].x, tri.points[1].y,
01472 0.3, 22, color);
01473     // adl_draw_arrow(screen_mat, tri.points[1].x, tri.points[1].y, tri.points[2].x, tri.points[2].y,
01474 0.3, 22, color);
01475     // adl_draw_arrow(screen_mat, tri.points[2].x, tri.points[2].y, tri.points[0].x, tri.points[0].y,
01476 0.3, 22, color);
01477 }
01478
01479 void adl_tri_fill_Pinedas_rasterizer(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Tri tri, uint32_t
01480 color, Offset_zoom_param offset_zoom_param)
01481 {
01482     /* This function follows the rasterizer of 'Pikuma' shown in his YouTube video. You can fine the
01483 video in this link: https://youtu.be/k5wtuKWmV48. */
01484
01485     Point p0, p1, p2;
01486     p0 = tri.points[0];
01487     p1 = tri.points[1];
01488     p2 = tri.points[2];
01489
01490     /* finding bounding box */
01491     int x_min = fmin(p0.x, fmin(p1.x, p2.x));
01492     int x_max = fmax(p0.x, fmax(p1.x, p2.x));
01493     int y_min = fmin(p0.y, fmin(p1.y, p2.y));
01494     int y_max = fmax(p0.y, fmax(p1.y, p2.y));
01495
01496     /* Clamp to screen bounds */
01497     if (x_min < 0) x_min = 0;
01498     if (y_min < 0) y_min = 0;
01499     if (x_max >= (int)screen_mat.cols) x_max = screen_mat.cols - 1;
01500     if (y_max >= (int)screen_mat.rows) y_max = screen_mat.rows - 1;
01501
01502     /* draw only outline of the tri if there is no area */
01503     float w = edge_cross_point(p0, p1, p1, p2);
01504     if (fabsf(w) < 1e-6) {
01505         // adl_tri_draw(screen_mat, tri, tri.colors[0], offset_zoom_param);
01506         return;
01507     }
01508     MATRIX2D_ASSERT(fabsf(w) > 1e-6 && "triangle must have area");
01509
01510     /* fill conventions */
01511     int bias0 = is_top_left(p0, p1) ? 0 : -1;
01512     int bias1 = is_top_left(p1, p2) ? 0 : -1;
01513     int bias2 = is_top_left(p2, p0) ? 0 : -1;
01514
01515     for (int y = y_min; y <= y_max; y++) {
01516         for (int x = x_min; x <= x_max; x++) {
01517             Point p = { .x = x, .y = y, .z = 0 };
01518
01519             float w0 = edge_cross_point(p0, p1, p0, p) + bias0;
01520             float w1 = edge_cross_point(p1, p2, p1, p) + bias1;
01521             float w2 = edge_cross_point(p2, p0, p2, p) + bias2;
01522
01523             float alpha = fabs(w1 / w);
01524             float beta = fabs(w2 / w);
01525             float gamma = fabs(w0 / w);
01526
01527             if (w0 * w >= 0 && w1 * w >= 0 && w2 * w >= 0) {
01528                 int r, b, g, a;
01529                 HexARGB_RGBA_VAR(color, r, g, b, a);
01530                 float light_intensity = (tri.light_intensity[0] + tri.light_intensity[1] +
01531 tri.light_intensity[2]) / 3;
01532                 float rf = r * light_intensity;
01533                 float gf = g * light_intensity;
01534                 float bf = b * light_intensity;
01535                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01536             }
01537         }
01538     }
01539 }
```

```

01546     uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01547     uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01548
01549     double inv_w = alpha * (1.0 / p0.w) + beta * (1.0 / p1.w) + gamma * (1.0 / p2.w);
01550     double z_over_w = alpha * (p0.z / p0.w) + beta * (p1.z / p1.w) + gamma * (p2.z /
01551     p2.w);
01552     double inv_z = inv_w / z_over_w;
01553
01554     if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01555         adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, a), offset_zoom_param);
01556         MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01557     }
01558 }
01559 }
01560 }
01561
01562 void adl_tri_fill_Pinedas_rasterizer_interpolate_color(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer,
01563     Tri tri, Offset_zoom_param offset_zoom_param)
01564 {
01565     /* This function follows the rasterizer of 'Pikuma' shown in his YouTube video. You can fine the
01566     video in this link: https://youtu.be/k5wtuKWmV48. */
01567     Point p0, p1, p2;
01568     p0 = tri.points[0];
01569     p1 = tri.points[1];
01570     p2 = tri.points[2];
01571
01572     float w = edge_cross_point(p0, p1, p1, p2);
01573     if (fabsf(w) < le-6) {
01574         // adl_tri_draw(screen_mat, tri, tri.colors[0], offset_zoom_param);
01575         return;
01576     }
01577     MATRIX2D_ASSERT(w != 0 && "triangle has area");
01578
01579     /* fill conventions */
01580     int bias0 = is_top_left(p0, p1) ? 0 : -1;
01581     int bias1 = is_top_left(p1, p2) ? 0 : -1;
01582     int bias2 = is_top_left(p2, p0) ? 0 : -1;
01583
01584     /* finding bounding box */
01585     int x_min = fmin(p0.x, fmin(p1.x, p2.x));
01586     int x_max = fmax(p0.x, fmax(p1.x, p2.x));
01587     int y_min = fmin(p0.y, fmin(p1.y, p2.y));
01588     int y_max = fmax(p0.y, fmax(p1.y, p2.y));
01589     // printf("xmin: %d, xmax: %d || ymin: %d, ymax: %d\n", x_min, x_max, y_min, y_max);
01590
01591     /* Clamp to screen bounds */
01592     if (x_min < 0) x_min = 0;
01593     if (y_min < 0) y_min = 0;
01594     if (x_max >= (int)screen_mat.cols) x_max = screen_mat.cols - 1;
01595     if (y_max >= (int)screen_mat.rows) y_max = screen_mat.rows - 1;
01596
01597     for (int y = y_min; y <= y_max; y++) {
01598         for (int x = x_min; x <= x_max; x++) {
01599             Point p = { .x = x, .y = y, .z = 0 };
01600
01601             float w0 = edge_cross_point(p0, p1, p0, p) + bias0;
01602             float w1 = edge_cross_point(p1, p2, p1, p) + bias1;
01603             float w2 = edge_cross_point(p2, p0, p2, p) + bias2;
01604
01605             float alpha = fabs(w1 / w);
01606             float beta = fabs(w2 / w);
01607             float gamma = fabs(w0 / w);
01608
01609             if (w0 * w >= 0 && w1 * w >= 0 && w2 * w >= 0) {
01610                 int r0, b0, g0, a0;
01611                 int r1, b1, g1, a1;
01612                 int r2, b2, g2, a2;
01613                 HexARGB_RGB4_VAR(tri.colors[0], r0, g0, b0, a0);
01614                 HexARGB_RGB4_VAR(tri.colors[1], r1, g1, b1, a1);
01615                 HexARGB_RGB4_VAR(tri.colors[2], r2, g2, b2, a2);
01616
01617                 uint8_t current_r = r0 * alpha + r1 * beta + r2 * gamma;
01618                 uint8_t current_g = g0 * alpha + g1 * beta + g2 * gamma;
01619                 uint8_t current_b = b0 * alpha + b1 * beta + b2 * gamma;
01620                 uint8_t current_a = a0 * alpha + a1 * beta + a2 * gamma;
01621
01622                 float light_intensity = (tri.light_intensity[0] + tri.light_intensity[1] +
01623                 tri.light_intensity[2]) / 3;
01624                 float rf = current_r * light_intensity;
01625                 float gf = current_g * light_intensity;
01626                 float bf = current_b * light_intensity;
01627                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01628                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01629                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01630
01631                 double inv_w = alpha * (1.0 / p0.w) + beta * (1.0 / p1.w) + gamma * (1.0 / p2.w);
01632             }
01633         }
01634     }
01635 }
```

```

01640     double z_over_w = alpha * (p0.z / p0.w) + beta * (p1.z / p1.w) + gamma * (p2.z /
01641     p2.w);
01642     double inv_z = inv_w / z_over_w;
01643     if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01644         adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, current_a),
01645         offset_zoom_param);
01646         MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01647     }
01648 }
01649 }
01650 }
01651
01652 void adl_tri_fill_Pinedas_rasterizer_interpolate_normal(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer,
01653   Tri tri, uint32_t color, Offset_zoom_param offset_zoom_param)
01654 {
01655   /* This function follows the rasterizer of 'Pikuma' shown in his YouTube video. You can fine the
01656   video in this link: https://youtu.be/k5wtuKWMv48. */
01657   Point p0, p1, p2;
01658   p0 = tri.points[0];
01659   p1 = tri.points[1];
01660   p2 = tri.points[2];
01661
01662   float w = edge_cross_point(p0, p1, p1, p2);
01663   if (fabsf(w) < 1e-6) {
01664     // adl_tri_draw(screen_mat, tri, tri.colors[0], offset_zoom_param);
01665     return;
01666   }
01667   MATRIX2D_ASSERT(w != 0 && "triangle has area");
01668
01669   /* fill conventions */
01670   int bias0 = is_top_left(p0, p1) ? 0 : -1;
01671   int bias1 = is_top_left(p1, p2) ? 0 : -1;
01672   int bias2 = is_top_left(p2, p0) ? 0 : -1;
01673
01674   /* finding bounding box */
01675   int x_min = fmin(p0.x, fmin(p1.x, p2.x));
01676   int x_max = fmax(p0.x, fmax(p1.x, p2.x));
01677   int y_min = fmin(p0.y, fmin(p1.y, p2.y));
01678   int y_max = fmax(p0.y, fmax(p1.y, p2.y));
01679   // printf("xmin: %d, xmax: %d || ymin: %d, ymax: %d\n", x_min, x_max, y_min, y_max);
01680
01681   /* Clamp to screen bounds */
01682   if (x_min < 0) x_min = 0;
01683   if (y_min < 0) y_min = 0;
01684   if (x_max >= (int)screen_mat.cols) x_max = screen_mat.cols - 1;
01685   if (y_max >= (int)screen_mat.rows) y_max = screen_mat.rows - 1;
01686
01687   int r, b, g, a;
01688   HexARGB_RGBA_VAR(color, r, g, b, a);
01689
01690   for (int y = y_min; y <= y_max; y++) {
01691     for (int x = x_min; x <= x_max; x++) {
01692       Point p = { .x = x, .y = y, .z = 0 };
01693
01694       float w0 = edge_cross_point(p0, p1, p0, p) + bias0;
01695       float w1 = edge_cross_point(p1, p2, p1, p) + bias1;
01696       float w2 = edge_cross_point(p2, p0, p2, p) + bias2;
01697
01698       float alpha = fabs(w1 / w);
01699       float beta = fabs(w2 / w);
01700       float gamma = fabs(w0 / w);
01701
01702       if (w0 * w >= 0 && w1 * w >= 0 && w2 * w >= 0) {
01703
01704         float light_intensity = tri.light_intensity[0]*alpha + tri.light_intensity[1]*beta +
01705         tri.light_intensity[2]*gamma;
01706
01707         float rf = r * light_intensity;
01708         float gf = g * light_intensity;
01709         float bf = b * light_intensity;
01710         uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01711         uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01712         uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01713
01714         double inv_w = alpha * (1.0 / p0.w) + beta * (1.0 / p1.w) + gamma * (1.0 / p2.w);
01715         double z_over_w = alpha * (p0.z / p0.w) + beta * (p1.z / p1.w) + gamma * (p2.z /
01716         p2.w);
01717         double inv_z = inv_w / z_over_w;
01718
01719         if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01720             adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, a), offset_zoom_param);
01721             MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01722         }
01723     }
01724   }
01725 }
```

```

01733     }
01734 }
01735
01746 void adl_tri_mesh_draw(Mat2D_uint32 screen_mat, Tri_mesh mesh, uint32_t color, Offset_zoom_param
01747 offset_zoom_param)
01748 {
01749     for (size_t i = 0; i < mesh.length; i++) {
01750         Tri tri = mesh.elements[i];
01751         if (tri.to_draw) {
01752             // color = rand_double() * 0xFFFFFFFF;
01753             adl_tri_draw(screen_mat, tri, color, offset_zoom_param);
01754         }
01755     }
01756 }
01757
01768 void adl_tri_mesh_fill_Pinedas_rasterizer(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Tri_mesh
01769 mesh, uint32_t color, Offset_zoom_param offset_zoom_param)
01770 {
01771     for (size_t i = 0; i < mesh.length; i++) {
01772         Tri tri = mesh.elements[i];
01773         /* Reject invalid triangles */
01774         adl_assert_tri_is_valid(tri);
01775
01776         if (!tri.to_draw) continue;
01777         adl_tri_fill_Pinedas_rasterizer(screen_mat, inv_z_buffer_mat, tri, color, offset_zoom_param);
01778     }
01779 }
01780
01792 void adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color(Mat2D_uint32 screen_mat, Mat2D
01793 inv_z_buffer_mat, Tri_mesh mesh, Offset_zoom_param offset_zoom_param)
01794 {
01795     for (size_t i = 0; i < mesh.length; i++) {
01796         Tri tri = mesh.elements[i];
01797         /* Reject invalid triangles */
01798         adl_assert_tri_is_valid(tri);
01799
01800         if (!tri.to_draw) continue;
01801         adl_tri_fill_Pinedas_rasterizer_interpolate_color(screen_mat, inv_z_buffer_mat, tri,
01802 offset_zoom_param);
01803     }
01804 }
01805
01817 void adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal(Mat2D_uint32 screen_mat, Mat2D
01818 inv_z_buffer_mat, Tri_mesh mesh, uint32_t color, Offset_zoom_param offset_zoom_param)
01819 {
01820     for (size_t i = 0; i < mesh.length; i++) {
01821         Tri tri = mesh.elements[i];
01822         /* Reject invalid triangles */
01823         adl_assert_tri_is_valid(tri);
01824
01825         if (!tri.to_draw) continue;
01826         adl_tri_fill_Pinedas_rasterizer_interpolate_normal(screen_mat, inv_z_buffer_mat, tri, color,
01827 offset_zoom_param);
01828     }
01829 }
01830
01845 float adl_tan_half_angle(Point vi, Point vj, Point p, float li, float lj)
01846 {
01847     float ax = vi.x - p.x, ay = vi.y - p.y;
01848     float bx = vj.x - p.x, by = vj.y - p.y;
01849     float dot = ax * bx + ay * by;
01850     float cross = ax * by - ay * bx;           // signed 2D cross (scalar)
01851     float denom = dot + li * lj;               // = |a||b|(1 + cos(alpha))
01852     return fabsf(cross) / fmaxf(1e-20f, denom); // tan(alpha/2)
01853 }
01854
01865 float adl_linear_map(float s, float min_in, float max_in, float min_out, float max_out)
01866 {
01867     return (min_out + ((s-min_in)*(max_out-min_out))/(max_in-min_in));
01868 }
01869
01885 void adl_quad2tris(Quad quad, Tri *tril, Tri *tri2, char split_line[])
01886 {
01887     if (!strcmp(split_line, "02", 2)) {
01888         tril->points[0] = quad.points[0];
01889         tril->points[1] = quad.points[1];
01890         tril->points[2] = quad.points[2];
01891         tril->to_draw = quad.to_draw;
01892         tril->light_intensity[0] = quad.light_intensity[0];
01893         tril->light_intensity[1] = quad.light_intensity[1];
01894         tril->light_intensity[2] = quad.light_intensity[2];
01895         tril->colors[0] = quad.colors[0];
01896         tril->colors[1] = quad.colors[1];
01897         tril->colors[2] = quad.colors[2];

```

```

01898     tri2->points[0] = quad.points[2];
01999     tri2->points[1] = quad.points[3];
02000     tri2->points[2] = quad.points[0];
02001     tri2->to_draw = quad.to_draw;
02002     tri2->light_intensity[0] = quad.light_intensity[2];
02003     tri2->light_intensity[1] = quad.light_intensity[3];
02004     tri2->light_intensity[2] = quad.light_intensity[0];
02005     tri2->colors[0] = quad.colors[2];
02006     tri2->colors[1] = quad.colors[3];
02007     tri2->colors[2] = quad.colors[0];
02008 } else if (!strcmp(split_line, "13", 2)) {
02009     tri1->points[0] = quad.points[1];
02010     tri1->points[1] = quad.points[2];
02011     tri1->points[2] = quad.points[3];
02012     tri1->to_draw = quad.to_draw;
02013     tri1->light_intensity[0] = quad.light_intensity[1];
02014     tri1->light_intensity[1] = quad.light_intensity[2];
02015     tri1->light_intensity[2] = quad.light_intensity[3];
02016     tri1->colors[0] = quad.colors[1];
02017     tri1->colors[1] = quad.colors[2];
02018     tri1->colors[2] = quad.colors[3];
02019
02020     tri2->points[0] = quad.points[3];
02021     tri2->points[1] = quad.points[0];
02022     tri2->points[2] = quad.points[1];
02023     tri2->to_draw = quad.to_draw;
02024     tri1->light_intensity[0] = quad.light_intensity[3];
02025     tri1->light_intensity[1] = quad.light_intensity[0];
02026     tri1->light_intensity[2] = quad.light_intensity[1];
02027     tri2->colors[0] = quad.colors[3];
02028     tri2->colors[1] = quad.colors[0];
02029     tri2->colors[2] = quad.colors[1];
02030 }
02031 }
02032 }
02033
02045 void adl_linear_sRGB_to_okLab(uint32_t hex_ARGB, float *L, float *a, float *b)
02046 {
02047     /* https://bottosson.github.io/posts/oklab/
02048        https://en.wikipedia.org/wiki/Oklab_color_space */
02049     int R_255, G_255, B_255;
02050     HexARGB_RGB_VAR(hex_ARGB, R_255, G_255, B_255);
02051
02052     float R = R_255;
02053     float G = G_255;
02054     float B = B_255;
02055
02056     float l = 0.4122214705f * R + 0.5363325363f * G + 0.0514459929f * B;
02057     float m = 0.2119034982f * R + 0.6806995451f * G + 0.1073969566f * B;
02058     float s = 0.0883024619f * R + 0.2817188376f * G + 0.6299787005f * B;
02059
02060     float l_ = cbrtf(l);
02061     float m_ = cbrtf(m);
02062     float s_ = cbrtf(s);
02063
02064     *L = 0.2104542553f * l_ + 0.7936177850f * m_ - 0.0040720468f * s_;
02065     *a = 1.9779984951f * l_ - 2.4285922050f * m_ + 0.4505937099f * s_;
02066     *b = 0.0259040371f * l_ + 0.7827717662f * m_ - 0.8086757660f * s_;
02067
02068 }
02069
02080 void adl_okLab_to_linear_sRGB(float L, float a, float b, uint32_t *hex_ARGB)
02081 {
02082     /* https://bottosson.github.io/posts/oklab/
02083        https://en.wikipedia.org/wiki/Oklab_color_space */
02084
02085     float l_ = L + 0.396337774f * a + 0.2158037573f * b;
02086     float m_ = L - 0.1055613458f * a - 0.0638541728f * b;
02087     float s_ = L - 0.0894841775f * a - 1.2914855480f * b;
02088
02089     float l = l_ * l_ * l_;
02090     float m = m_ * m_ * m_;
02091     float s = s_ * s_ * s_;
02092
02093     float R = + 4.0767416621f * l - 3.3077115913f * m + 0.2309699292f * s;
02094     float G = - 1.2684380046f * l + 2.6097574011f * m - 0.3413193965f * s;
02095     float B = - 0.0041960863f * l - 0.7034186147f * m + 1.7076147010f * s;
02096
02097     R = fmaxf(fminf(R, 255), 0);
02098     G = fmaxf(fminf(G, 255), 0);
02099     B = fmaxf(fminf(B, 255), 0);
02000
02001     *hex_ARGB = RGBA_hexARGB(R, G, B, 0xFF);
02002 }
02003
02012 void adl_linear_sRGB_to_okLch(uint32_t hex_ARGB, float *L, float *c, float *h_deg)
02013 {

```

```

02014     float a, b;
02015     adl_linear_sRGB_to_okLab(hex_ARGB, L, &a, &b);
02016
02017     *c = sqrtf(a * a + b * b);
02018     *h_deg = atan2f(b, a) * 180 / PI;
02019 }
02020
02031 void adl_okLch_to_linear_sRGB(float L, float c, float h_deg, uint32_t *hex_ARGB)
02032 {
02033     h_deg = fmodf((h_deg + 360), 360);
02034     float a = c * cosf(h_deg * PI / 180);
02035     float b = c * sinf(h_deg * PI / 180);
02036     adl_okLab_to_linear_sRGB(L, a, b, hex_ARGB);
02037 }
02038
02039 void adl_interpolate_ARGBcolor_on_okLch(uint32_t color1, uint32_t color2, float t, float
02040     num_of_rotations, uint32_t *color_out)
02041 {
02042     float L_1, c_1, h_1;
02043     float L_2, c_2, h_2;
02044     adl_linear_sRGB_to_okLch(color1, &L_1, &c_1, &h_1);
02045     adl_linear_sRGB_to_okLch(color2, &L_2, &c_2, &h_2);
02046     h_2 = h_2 + 360 * num_of_rotations;
02047
02048     float L, c, h;
02049     L = L_1 * (1 - t) + L_2 * (t);
02050     c = c_1 * (1 - t) + c_2 * (t);
02051     h = h_1 * (1 - t) + h_2 * (t);
02052     adl_okLch_to_linear_sRGB(L, c, h, color_out);
02053 }
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081 Figure adl_figure_alloc(size_t rows, size_t cols, Point top_left_position)
02082 {
02083     ADL_ASSERT(rows && cols);
02084     adl_assert_point_is_valid(top_left_position);
02085
02086     Figure figure = {0};
02087     figure.pixels_mat = mat2D_alloc_uint32(rows, cols);
02088     figure.inv_z_buffer_mat = mat2D_alloc(rows, cols);
02089     memset(figure.inv_z_buffer_mat.elements, 0x0, sizeof(double) * figure.inv_z_buffer_mat.rows *
02090         figure.inv_z_buffer_mat.cols);
02091     ada_init_array(Curve, figure.src_curve_array);
02092
02093     figure.top_left_position = top_left_position;
02094
02095     int max_i      = (int)(figure.pixels_mat.rows);
02096     int max_j      = (int)(figure.pixels_mat.cols);
02097     int offset_i   = (int)fminf(figure.pixels_mat.rows * ADL FIGURE_PADDING_PERCENTAGE / 100.0f,
02098         ADL_MAX FIGURE_PADDING);
02099     int offset_j   = (int)fminf(figure.pixels_mat.cols * ADL FIGURE_PADDING_PERCENTAGE / 100.0f,
02100         ADL_MAX FIGURE_PADDING);
02101
02102     figure.min_x_pixel = offset_j;
02103     figure.max_x_pixel = max_j - offset_j;
02104     figure.min_y_pixel = offset_i;
02105     figure.max_y_pixel = max_i - offset_i;
02106
02107     figure.min_x = + FLT_MAX;
02108     figure.max_x = - FLT_MAX;
02109     figure.min_y = + FLT_MAX;
02110     figure.max_y = - FLT_MAX;
02111
02112     figure.offset_zoom_param = ADL_DEFAULT_OFFSET_ZOOM;
02113
02114     return figure;
02115 }
02116
02117 void adl_figure_copy_to_screen(Mat2D_uint32 screen_mat, Figure figure)
02118 {
02119     for (size_t i = 0; i < figure.pixels_mat.rows; i++) {
02120         for (size_t j = 0; j < figure.pixels_mat.cols; j++) {
02121             int offset_i = figure.top_left_position.y;
02122             int offset_j = figure.top_left_position.x;
02123
02124             adl_point_draw(screen_mat, offset_j+j, offset_i+i, MAT2D_AT_UINT32(figure.pixels_mat, i,
02125                 j), (Offset_zoom_param){1,0,0,0,0});
02126         }
02127     }
02128 }
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02140
02141
02142
02143
02144 void adl_axis_draw_on_figure(Figure *figure)
02145 {
02146     int max_i      = (int)(figure->pixels_mat.rows);
02147     int max_j      = (int)(figure->pixels_mat.cols);
02148     int offset_i   = (int)fmaxf(fminf(figure->pixels_mat.rows * ADL FIGURE_PADDING_PERCENTAGE / 100.0f,
02149         ADL_MAX FIGURE_PADDING), ADL_MIN FIGURE_PADDING);
02150     int offset_j   = (int)fmaxf(fminf(figure->pixels_mat.cols * ADL FIGURE_PADDING_PERCENTAGE / 100.0f,

```

```

ADL_MAX FIGURE PADDING), ADL_MIN FIGURE PADDING);

02150     int arrow_head_size_x = (int)fminf(ADL_MAX_HEAD_SIZE, ADL FIGURE PADDING PRECENTAGE / 100.0f *
02151     (max_j - 2 * offset_j));
02152     int arrow_head_size_y = (int)fminf(ADL_MAX_HEAD_SIZE, ADL FIGURE PADDING PRECENTAGE / 100.0f *
02153     (max_i - 2 * offset_i));
02154     adl_arrow_draw(figure->pixels_mat, figure->min_x_pixel, figure->max_y_pixel, figure->max_x_pixel,
02155     figure->max_y_pixel, (float)arrow_head_size_x / (max_j-2*offset_j), ADL FIGURE HEAD ANGLE DEG,
02156     ADL FIGURE AXIS COLOR, figure->offset_zoom_param);
02157     adl_arrow_draw(figure->pixels_mat, figure->min_x_pixel, figure->max_y_pixel, figure->min_x_pixel,
02158     figure->min_y_pixel, (float)arrow_head_size_y / (max_i-2*offset_i), ADL FIGURE HEAD ANGLE DEG,
02159     ADL FIGURE AXIS COLOR, figure->offset_zoom_param);
02160     // adl_draw_rectangle_min_max(figure->pixels_mat, figure->min_x_pixel, figure->max_x_pixel,
02161     // figure->min_y_pixel, figure->max_y_pixel, 0);
02162
02163     figure->x_axis_head_size = arrow_head_size_x;
02164     figure->y_axis_head_size = arrow_head_size_y;
02165 }

02166 void adl_max_min_values_draw_on_figure(FIGURE figure)
02167 {
02168     char x_min_sentence[256];
02169     char x_max_sentence[256];
02170     snprintf(x_min_sentence, 256, "%g", figure.min_x);
02171     snprintf(x_max_sentence, 256, "%g", figure.max_x);
02172
02173     int x_sentence_hight_pixel = (figure.pixels_mat.rows - figure.max_y_pixel -
02174     ADL_MIN_CHARACTER_OFFSET * 3);
02175     int x_min_char_width_pixel = x_sentence_hight_pixel / 2;
02176     int x_max_char_width_pixel = x_sentence_hight_pixel / 2;
02177
02178     int x_min_sentence_width_pixel = (int)fminf((figure.max_x_pixel - figure.min_x_pixel)/2,
02179     (x_min_char_width_pixel + ADL_MAX_CHARACTER_OFFSET)*strlen(x_min_sentence));
02180     x_min_char_width_pixel = x_min_sentence_width_pixel / strlen(x_min_sentence) -
02181     ADL_MIN_CHARACTER_OFFSET;
02182
02183     int x_max_sentence_width_pixel = (int)fminf((figure.max_x_pixel - figure.min_x_pixel)/2,
02184     (x_max_char_width_pixel + ADL_MAX_CHARACTER_OFFSET)*strlen(x_max_sentence)) -
02185     figure.x_axis_head_size;
02186     x_max_char_width_pixel = (x_max_sentence_width_pixel + figure.x_axis_head_size) /
02187     strlen(x_max_sentence) - ADL_MIN_CHARACTER_OFFSET;
02188
02189     int x_min_sentence_hight_pixel = (int)fminf(x_min_char_width_pixel * 2, x_sentence_hight_pixel);
02190     int x_max_sentence_hight_pixel = (int)fminf(x_max_char_width_pixel * 2, x_sentence_hight_pixel);
02191
02192     x_min_sentence_hight_pixel = (int)fminf(x_min_sentence_hight_pixel, x_max_sentence_hight_pixel);
02193     x_max_sentence_hight_pixel = x_min_sentence_hight_pixel;
02194
02195     int x_max_x_top_left = figure.max_x_pixel - strlen(x_max_sentence) * (x_max_sentence_hight_pixel /
02196     2 + ADL_MIN_CHARACTER_OFFSET) - figure.x_axis_head_size;
02197
02198     adl_sentence_draw(figure.pixels_mat, x_min_sentence, strlen(x_min_sentence), figure.min_x_pixel,
02199     figure.max_y_pixel+ADL_MIN_CHARACTER_OFFSET*2, x_min_sentence_hight_pixel, ADL FIGURE AXIS COLOR,
02200     figure.offset_zoom_param);
02201     adl_sentence_draw(figure.pixels_mat, x_max_sentence, strlen(x_max_sentence), x_max_x_top_left,
02202     figure.max_y_pixel+ADL_MIN_CHARACTER_OFFSET*2, x_max_sentence_hight_pixel, ADL FIGURE AXIS COLOR,
02203     figure.offset_zoom_param);
02204
02205     char y_min_sentence[256];
02206     char y_max_sentence[256];
02207     snprintf(y_min_sentence, 256, "%g", figure.min_y);
02208     snprintf(y_max_sentence, 256, "%g", figure.max_y);
02209
02210     int y_sentence_width_pixel = figure.min_x_pixel - ADL_MAX_CHARACTER_OFFSET -
02211     figure.y_axis_head_size;
02212     int y_max_char_width_pixel = y_sentence_width_pixel;
02213     y_max_char_width_pixel /= strlen(y_max_sentence);
02214     int y_max_sentence_hight_pixel = y_max_char_width_pixel * 2;
02215
02216     int y_min_char_width_pixel = y_sentence_width_pixel;
02217     y_min_char_width_pixel /= strlen(y_min_sentence);
02218     int y_min_sentence_hight_pixel = y_min_char_width_pixel * 2;
02219
02220     y_min_sentence_hight_pixel = (int)fmaxf(fminf(y_min_sentence_hight_pixel,
02221     y_max_sentence_hight_pixel), 1);
02222     y_max_sentence_hight_pixel = y_min_sentence_hight_pixel;
02223
02224     adl_sentence_draw(figure.pixels_mat, y_max_sentence, strlen(y_max_sentence),
02225     ADL_MAX_CHARACTER_OFFSET/2, figure.min_y_pixel, y_max_sentence_hight_pixel, ADL FIGURE AXIS COLOR,
02226     figure.offset_zoom_param);
02227     adl_sentence_draw(figure.pixels_mat, y_min_sentence, strlen(y_min_sentence),
02228     ADL_MAX_CHARACTER_OFFSET/2, figure.max_y_pixel-y_min_sentence_hight_pixel,
02229     y_min_sentence_hight_pixel, ADL FIGURE AXIS COLOR, figure.offset_zoom_param);
02230 }

02231 void adl_curve_add_to_figure(FIGURE *figure, Point *src_points, size_t src_len, uint32_t color)

```

```

02231 {
02232     Curve src_points_ada;
02233     ada_init_array(Point, src_points_ada);
02234     src_points_ada.color = color;
02235
02236     for (size_t i = 0; i < src_len; i++) {
02237         Point current_point = src_points[i];
02238         if (current_point.x > figure->max_x) {
02239             figure->max_x = current_point.x;
02240         }
02241         if (current_point.y > figure->max_y) {
02242             figure->max_y = current_point.y;
02243         }
02244         if (current_point.x < figure->min_x) {
02245             figure->min_x = current_point.x;
02246         }
02247         if (current_point.y < figure->min_y) {
02248             figure->min_y = current_point.y;
02249         }
02250         ada_append(Point, src_points_ada, current_point);
02251     }
02252
02253     ada_append(Curve, figure->src_curve_array, src_points_ada);
02254 }
02255
02256 void adl_curves_plot_on_figure(Figure figure)
02257 {
02258     mat2D_fill_uint32(figure.pixels_mat, figure.background_color);
02259     memset(figure.inv_z_buffer_mat.elements, 0x0, sizeof(double) * figure.inv_z_buffer_mat.rows *
02260     figure.inv_z_buffer_mat.cols);
02261     if (figure.to_draw_axis) adl_axis_draw_on_figure(&figure);
02262
02263     for (size_t curve_index = 0; curve_index < figure.src_curve_array.length; curve_index++) {
02264         size_t src_len = figure.src_curve_array.elements[curve_index].length;
02265         Point *src_points = figure.src_curve_array.elements[curve_index].elements;
02266         for (size_t i = 0; i < src_len-1; i++) {
02267             Point src_start = src_points[i];
02268             Point src_end = src_points[i+1];
02269             Point des_start = {0};
02270             Point des_end = {0};
02271
02272             des_start.x = adl_linear_map(src_start.x, figure.min_x, figure.max_x, figure.min_x_pixel,
02273                 figure.max_x_pixel);
02274             des_start.y = ((figure.max_y_pixel + figure.min_y_pixel) - adl_linear_map(src_start.y,
02275                 figure.min_y, figure.max_y, figure.min_y_pixel, figure.max_y_pixel));
02276
02277             des_end.x = adl_linear_map(src_end.x, figure.min_x, figure.max_x, figure.min_x_pixel,
02278                 figure.max_x_pixel);
02279             des_end.y = ((figure.max_y_pixel + figure.min_y_pixel) - adl_linear_map(src_end.y,
02280                 figure.min_y, figure.max_y, figure.min_y_pixel, figure.max_y_pixel));
02281
02282             adl_line_draw(figure.pixels_mat, des_start.x, des_start.y, des_end.x, des_end.y,
02283                 figure.src_curve_array.elements[curve_index].color, figure.offset_zoom_param);
02284         }
02285     }
02286
02287     if (figure.to_draw_max_min_values) adl_max_min_values_draw_on_figure(figure);
02288 }
02289
02290 /* check offset2D. might convert it to a Mat2D */
02291 #define adl_offset2d(i, j, ni) (j) * (ni) + (i)
02292
02293 /* check offset2D. might convert it to a Mat2D */
02294 #define adl_offset2d(i, j, ni) (j) * (ni) + (i)
02295
02296 void adl_2Dscalar_interp_on_figure(Figure figure, double *x_2Dmat, double *y_2Dmat, double
02297     *scalar_2Dmat, int ni, int nj, char color_scale[], float num_of_rotations)
02298 {
02299     mat2D_fill_uint32(figure.pixels_mat, figure.background_color);
02300     memset(figure.inv_z_buffer_mat.elements, 0x0, sizeof(double) * figure.inv_z_buffer_mat.rows *
02301     figure.inv_z_buffer_mat.cols);
02302     if (figure.to_draw_axis) adl_axis_draw_on_figure(&figure);
02303
02304     float min_scalar = FLT_MAX;
02305     float max_scalar = FLT_MIN;
02306     for (int i = 0; i < ni; i++) {
02307         for (int j = 0; j < nj; j++) {
02308             float val = scalar_2Dmat[adl_offset2d(i, j, ni)];
02309             if (val > max_scalar) max_scalar = val;
02310             if (val < min_scalar) min_scalar = val;
02311             float current_x = x_2Dmat[adl_offset2d(i, j, ni)];
02312             float current_y = y_2Dmat[adl_offset2d(i, j, ni)];
02313             if (current_x > figure.max_x) {
02314                 figure.max_x = current_x;
02315             }
02316             if (current_y > figure.max_y) {
02317                 figure.max_y = current_y;
02318             }
02319             if (current_x < figure.min_x) {
02320                 figure.min_x = current_x;
02321             }
02322         }
02323     }
02324 }
```

```

02338         if (current_y < figure.min_y) {
02339             figure.min_y = current_y;
02340         }
02341     }
02342 }
02343
02344 float window_w = (float)figure.pixels_mat.cols;
02345 float window_h = (float)figure.pixels_mat.rows;
02346
02347 for (int i = 0; i < ni-1; i++) {
02348     for (int j = 0; j < nj-1; j++) {
02349         Quad quad = {0};
02350         quad.light_intensity[0] = 1;
02351         quad.light_intensity[1] = 1;
02352         quad.light_intensity[2] = 1;
02353         quad.light_intensity[3] = 1;
02354         quad.to_draw = 1;
02355
02356         quad.points[3].x = x_2Dmat[adl_offset2d(i , j , ni)];
02357         quad.points[3].y = y_2Dmat[adl_offset2d(i , j , ni)];
02358         quad.points[2].x = x_2Dmat[adl_offset2d(i+1, j , ni)];
02359         quad.points[2].y = y_2Dmat[adl_offset2d(i+1, j , ni)];
02360         quad.points[1].x = x_2Dmat[adl_offset2d(i+1, j+1, ni)];
02361         quad.points[1].y = y_2Dmat[adl_offset2d(i+1, j+1, ni)];
02362         quad.points[0].x = x_2Dmat[adl_offset2d(i , j+1, ni)];
02363         quad.points[0].y = y_2Dmat[adl_offset2d(i , j+1, ni)];
02364
02365         for (int p_index = 0; p_index < 4; p_index++) {
02366             quad.points[p_index].z = 1;
02367             quad.points[p_index].w = 1;
02368             quad.points[p_index].x = adl_linear_map(quad.points[p_index].x, figure.min_x,
02369             figure.max_x, figure.min_x_pixel, figure.max_x_pixel);
02370             quad.points[p_index].y = ((figure.max_y_pixel + figure.min_y_pixel) -
02371             adl_linear_map(quad.points[p_index].y, figure.min_y, figure.max_y, figure.min_y_pixel,
02372             figure.max_y_pixel));
02373
02374         adl_offset_zoom_point(quad.points[p_index], window_w, window_h,
02375         figure.offset_zoom_param);
02376     }
02377
02378     float t3 = adl_linear_map(scalar_2Dmat[adl_offset2d(i , j , ni)], min_scalar,
02379     max_scalar, 0, 1);
02380     float t2 = adl_linear_map(scalar_2Dmat[adl_offset2d(i+1, j , ni)], min_scalar,
02381     max_scalar, 0, 1);
02382     float t1 = adl_linear_map(scalar_2Dmat[adl_offset2d(i+1, j+1, ni)], min_scalar,
02383     max_scalar, 0, 1);
02384     float t0 = adl_linear_map(scalar_2Dmat[adl_offset2d(i , j+1, ni)], min_scalar,
02385     max_scalar, 0, 1);
02386
02387     /* https://en.wikipedia.org/wiki/Oklab_color_space */
02388     if (!strcmp(color_scale, "b-c")) {
02389         uint32_t color = 0, color1 = BLUE_hexARGB, color2 = CYAN_hexARGB;
02390         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02391         quad.colors[0] = color;
02392
02393         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02394         quad.colors[1] = color;
02395
02396         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02397         quad.colors[2] = color;
02398
02399         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02400         quad.colors[3] = color;
02401
02402     } else if (!strcmp(color_scale, "b-g")) {
02403         uint32_t color = 0, color1 = BLUE_hexARGB, color2 = GREEN_hexARGB;
02404         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02405         quad.colors[0] = color;
02406
02407         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02408         quad.colors[1] = color;
02409
02410         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02411         quad.colors[2] = color;
02412
02413     } else if (!strcmp(color_scale, "b-r")) {
02414         uint32_t color = 0, color1 = BLUE_hexARGB, color2 = RED_hexARGB;
02415         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02416         quad.colors[0] = color;
02417
02418         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02419         quad.colors[1] = color;
02420
02421         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02422         quad.colors[2] = color;
02423
02424     }
02425 }
```

```

02417         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02418         quad.colors[3] = color;
02419     } else if (!strcmp(color_scale, "b-y")) {
02420         uint32_t color = 0, color1 = BLUE_hexARGB, color2 = YELLOW_hexARGB;
02421         adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02422         quad.colors[0] = color;
02423
02424     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02425     quad.colors[1] = color;
02426
02427     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02428     quad.colors[2] = color;
02429
02430     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02431     quad.colors[3] = color;
02432 } else if (!strcmp(color_scale, "g-y")) {
02433     uint32_t color = 0, color1 = GREEN_hexARGB, color2 = YELLOW_hexARGB;
02434     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02435     quad.colors[0] = color;
02436
02437     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02438     quad.colors[1] = color;
02439
02440     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02441     quad.colors[2] = color;
02442
02443     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02444     quad.colors[3] = color;
02445 } else if (!strcmp(color_scale, "g-p")) {
02446     uint32_t color = 0, color1 = GREEN_hexARGB, color2 = PURPLE_hexARGB;
02447     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02448     quad.colors[0] = color;
02449
02450     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02451     quad.colors[1] = color;
02452
02453     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02454     quad.colors[2] = color;
02455
02456     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02457     quad.colors[3] = color;
02458 } else if (!strcmp(color_scale, "g-r")) {
02459     uint32_t color = 0, color1 = GREEN_hexARGB, color2 = RED_hexARGB;
02460     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02461     quad.colors[0] = color;
02462
02463     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02464     quad.colors[1] = color;
02465
02466     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02467     quad.colors[2] = color;
02468
02469     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02470     quad.colors[3] = color;
02471 } else if (!strcmp(color_scale, "r-y")) {
02472     uint32_t color = 0, color1 = RED_hexARGB, color2 = YELLOW_hexARGB;
02473     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02474     quad.colors[0] = color;
02475
02476     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02477     quad.colors[1] = color;
02478
02479     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02480     quad.colors[2] = color;
02481
02482     adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02483     quad.colors[3] = color;
02484 }
02485
02486     adl_quad_fill_interpolate_color_mean_value(figure.pixels_mat, figure.inv_z_buffer_mat,
02487     quad, ADL_DEFAULT_OFFSET_ZOOM);
02488 }
02489
02490 if (figure.to_draw_max_min_values) {
02491     adl_max_min_values_draw_on_figure(figure);
02492 }
02493
02494 }
02495
02513 Grid adl_cartesian_grid_create(float min_e1, float max_e1, float min_e2, float max_e2, int
02514     num_samples_e1, int num_samples_e2, char plane[], float third_direction_position)
02515 {
02516     Grid grid;
02517     ada_init_array(Curve, grid.curves);
02518     grid.min_e1 = min_e1;

```

```

02519     grid.max_e1 = max_e1;
02520     grid.min_e2 = min_e2;
02521     grid.max_e2 = max_e2;
02522     grid.num_samples_e1 = num_samples_e1;
02523     grid.num_samples_e2 = num_samples_e2;
02524     strncpy(grid.plane, plane, 2);
02525
02526     float del_e1 = (max_e1 - min_e1) / num_samples_e1;
02527     float del_e2 = (max_e2 - min_e2) / num_samples_e2;
02528
02529     grid.del1 = del_e1;
02530     grid.del2 = del_e2;
02531
02532     if (!strcmp(plane, "XY", 3) || !strcmp(plane, "xy", 3)) {
02533         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02534             Curve curve = {0};
02535             ada_init_array(Point, curve);
02536             Point point_max = {0}, point_min = {0};
02537
02538             point_min.x = min_e1 + e1_index * del_e1;
02539             point_min.y = min_e2;
02540             point_min.z = third_direction_position;
02541             point_min.w = 1;
02542
02543             point_max.x = min_e1 + e1_index * del_e1;
02544             point_max.y = max_e2;
02545             point_max.z = third_direction_position;
02546             point_max.w = 1;
02547
02548             ada_appand(Point, curve, point_min);
02549             ada_appand(Point, curve, point_max);
02550
02551             ada_appand(Curve, grid.curves, curve);
02552         }
02553         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02554             Curve curve = {0};
02555             ada_init_array(Point, curve);
02556             Point point_max = {0}, point_min = {0};
02557
02558             point_min.x = min_e1;
02559             point_min.y = min_e2 + e2_index * del_e2;
02560             point_min.z = third_direction_position;
02561             point_min.w = 1;
02562
02563             point_max.x = max_e1;
02564             point_max.y = min_e2 + e2_index * del_e2;
02565             point_max.z = third_direction_position;
02566             point_max.w = 1;
02567
02568             ada_appand(Point, curve, point_min);
02569             ada_appand(Point, curve, point_max);
02570
02571             ada_appand(Curve, grid.curves, curve);
02572         }
02573     } else if (!strcmp(plane, "XZ", 3) || !strcmp(plane, "xz", 3)) {
02574         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02575             Curve curve = {0};
02576             ada_init_array(Point, curve);
02577             Point point_max = {0}, point_min = {0};
02578
02579             point_min.x = min_e1 + e1_index * del_e1;
02580             point_min.y = third_direction_position;
02581             point_min.z = min_e2;
02582             point_min.w = 1;
02583
02584             point_max.x = min_e1 + e1_index * del_e1;
02585             point_max.y = third_direction_position;
02586             point_max.z = max_e2;
02587             point_max.w = 1;
02588
02589             ada_appand(Point, curve, point_min);
02590             ada_appand(Point, curve, point_max);
02591
02592             ada_appand(Curve, grid.curves, curve);
02593         }
02594         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02595             Curve curve = {0};
02596             ada_init_array(Point, curve);
02597             Point point_max = {0}, point_min = {0};
02598
02599             point_min.x = min_e1;
02600             point_min.y = third_direction_position;
02601             point_min.z = min_e2 + e2_index * del_e2;
02602             point_min.w = 1;
02603
02604             point_max.x = max_e1;
02605             point_max.y = third_direction_position;

```

```
02606     point_max.z = min_e2 + e2_index * del_e2;
02607     point_max.w = 1;
02608
02609     ada_append(Point, curve, point_min);
02610     ada_append(Point, curve, point_max);
02611
02612     ada_append(Curve, grid.curves, curve);
02613 }
02614 } else if (!strcmp(plane, "YX", 3) || !strcmp(plane, "yx", 3)) {
02615     for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02616         Curve curve = {0};
02617         ada_init_array(Point, curve);
02618         Point point_max = {0}, point_min = {0};
02619
02620         point_min.x = min_e2;
02621         point_min.y = min_e1 + e1_index * del_e1;
02622         point_min.z = third_direction_position;
02623         point_min.w = 1;
02624
02625         point_max.x = max_e2;
02626         point_max.y = min_e1 + e1_index * del_e1;
02627         point_max.z = third_direction_position;
02628         point_max.w = 1;
02629
02630         ada_append(Point, curve, point_min);
02631         ada_append(Point, curve, point_max);
02632
02633         ada_append(Curve, grid.curves, curve);
02634     }
02635     for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02636         Curve curve = {0};
02637         ada_init_array(Point, curve);
02638         Point point_max = {0}, point_min = {0};
02639
02640         point_min.x = min_e2 + e2_index * del_e2;
02641         point_min.y = min_e1;
02642         point_min.z = third_direction_position;
02643         point_min.w = 1;
02644
02645         point_max.x = min_e2 + e2_index * del_e2;
02646         point_max.y = max_e1;
02647         point_max.z = third_direction_position;
02648         point_max.w = 1;
02649
02650         ada_append(Point, curve, point_min);
02651         ada_append(Point, curve, point_max);
02652
02653         ada_append(Curve, grid.curves, curve);
02654     }
02655 } else if (!strcmp(plane, "YZ", 3) || !strcmp(plane, "yz", 3)) {
02656     for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02657         Curve curve = {0};
02658         ada_init_array(Point, curve);
02659         Point point_max = {0}, point_min = {0};
02660
02661         point_min.x = third_direction_position;
02662         point_min.y = min_e1 + e1_index * del_e1;
02663         point_min.z = min_e2;
02664         point_min.w = 1;
02665
02666         point_max.x = third_direction_position;
02667         point_max.y = min_e1 + e1_index * del_e1;
02668         point_max.z = max_e2;
02669         point_max.w = 1;
02670
02671         ada_append(Point, curve, point_min);
02672         ada_append(Point, curve, point_max);
02673
02674         ada_append(Curve, grid.curves, curve);
02675     }
02676     for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02677         Curve curve = {0};
02678         ada_init_array(Point, curve);
02679         Point point_max = {0}, point_min = {0};
02680
02681         point_min.x = third_direction_position;
02682         point_min.y = min_e1;
02683         point_min.z = min_e2 + e2_index * del_e2;
02684         point_min.w = 1;
02685
02686         point_max.x = third_direction_position;
02687         point_max.y = max_e1;
02688         point_max.z = min_e2 + e2_index * del_e2;
02689         point_max.w = 1;
02690
02691         ada_append(Point, curve, point_min);
02692         ada_append(Point, curve, point_max);
```

```

02693
02694     ada_appand(Curve, grid.curves, curve);
02695 }
02696 } else if (!strcmp(plane, "ZX", 3) || !strcmp(plane, "zx", 3)) {
02697     for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02698         Curve curve = {0};
02699         ada_init_array(Point, curve);
02700         Point point_max = {0}, point_min = {0};
02701
02702         point_min.x = min_e2;
02703         point_min.y = third_direction_position;
02704         point_min.z = min_e1 + e1_index * del_e1;
02705         point_min.w = 1;
02706
02707         point_max.x = max_e2;
02708         point_max.y = third_direction_position;
02709         point_max.z = min_e1 + e1_index * del_e1;
02710         point_max.w = 1;
02711
02712         ada_appand(Point, curve, point_min);
02713         ada_appand(Point, curve, point_max);
02714
02715         ada_appand(Curve, grid.curves, curve);
02716     }
02717     for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02718         Curve curve = {0};
02719         ada_init_array(Point, curve);
02720         Point point_max = {0}, point_min = {0};
02721
02722         point_min.x = min_e2 + e2_index * del_e2;
02723         point_min.y = third_direction_position;
02724         point_min.z = min_e1;
02725         point_min.w = 1;
02726
02727         point_max.x = min_e2 + e2_index * del_e2;
02728         point_max.y = third_direction_position;
02729         point_max.z = max_e1;
02730         point_max.w = 1;
02731
02732         ada_appand(Point, curve, point_min);
02733         ada_appand(Point, curve, point_max);
02734
02735         ada_appand(Curve, grid.curves, curve);
02736     }
02737 } else if (!strcmp(plane, "ZY", 3) || !strcmp(plane, "zy", 3)) {
02738     for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02739         Curve curve = {0};
02740         ada_init_array(Point, curve);
02741         Point point_max = {0}, point_min = {0};
02742
02743         point_min.x = third_direction_position;
02744         point_min.y = min_e2;
02745         point_min.z = min_e1 + e1_index * del_e1;
02746         point_min.w = 1;
02747
02748         point_max.x = third_direction_position;
02749         point_max.y = max_e2;
02750         point_max.z = min_e1 + e1_index * del_e1;
02751         point_max.w = 1;
02752
02753         ada_appand(Point, curve, point_min);
02754         ada_appand(Point, curve, point_max);
02755
02756         ada_appand(Curve, grid.curves, curve);
02757     }
02758     for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02759         Curve curve = {0};
02760         ada_init_array(Point, curve);
02761         Point point_max = {0}, point_min = {0};
02762
02763         point_min.x = third_direction_position;
02764         point_min.y = min_e2 + e2_index * del_e2;
02765         point_min.z = min_e1;
02766         point_min.w = 1;
02767
02768         point_max.x = third_direction_position;
02769         point_max.y = min_e2 + e2_index * del_e2;
02770         point_max.z = max_e1;
02771         point_max.w = 1;
02772
02773         ada_appand(Point, curve, point_min);
02774         ada_appand(Point, curve, point_max);
02775
02776         ada_appand(Curve, grid.curves, curve);
02777     }
02778 }
02779

```

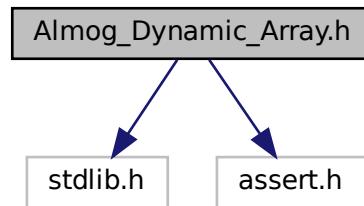
```
02780     return grid;
02781 }
02782
02791 void adl_grid_draw(Mat2D_uint32 screen_mat, Grid grid, uint32_t color, Offset_zoom_param
02792 offset_zoom_param)
02792 {
02793     for (size_t curve_index = 0; curve_index < grid.curves.length; curve_index++) {
02794         adl_lines_draw(screen_mat, grid.curves.elements[curve_index].elements,
02795 grid.curves.elements[curve_index].length, color, offset_zoom_param);
02796     }
02797
02798 #endif /*ALMOG_DRAW_LIBRARY_IMPLEMENTATION*/
```

4.3 Almog_Dynamic_Array.h File Reference

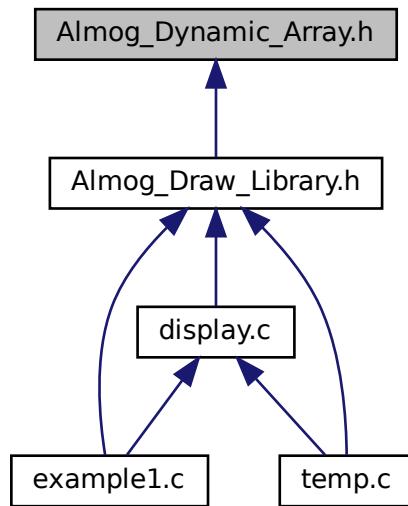
Header-only C macros that implement a simple dynamic array.

```
#include <stdlib.h>
#include <assert.h>
```

Include dependency graph for Almog_Dynamic_Array.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define ADA_INIT_CAPACITY 10`
Default initial capacity used by ada_init_array.
- `#define ADA_MALLOC malloc`
Allocation function used by this header (defaults to malloc).
- `#define ADA_REALLOC realloc`
Reallocation function used by this header (defaults to realloc).
- `#define ADA_ASSERT assert`
Assertion macro used by this header (defaults to assert).
- `#define ada_init_array(type, header)`
Initialize an array header and allocate its initial storage.
- `#define ada_resize(type, header, new_capacity)`
Resize the underlying storage to hold new_capacity elements.
- `#define ada_append(type, header, value)`
Append a value to the end of the array, growing if necessary.
- `#define ada_insert(type, header, value, index)`
Insert value at position index, preserving order ($O(n)$).
- `#define ada_insert_unordered(type, header, value, index)`
Insert value at index without preserving order ($O(1)$ amortized).
- `#define ada_remove(type, header, index)`
Remove element at index, preserving order ($O(n)$).
- `#define ada_remove_unordered(type, header, index)`
Remove element at index by moving the last element into its place ($O(1)$); order is not preserved.

4.3.1 Detailed Description

Header-only C macros that implement a simple dynamic array.

This header provides a minimal, macro-based dynamic array for POD-like types. The array "header" is a user-defined struct with three fields:

- `size_t length`; current number of elements
- `size_t capacity`; allocated capacity (in elements)
- `T* elements`; pointer to contiguous storage of elements (type T)

How to use: 1) Define a header struct with length/capacity/elements fields. 2) Initialize it with `ada_init_array(T, header)`. 3) Modify it with `ada_appand` (append), `ada_insert`, `remove` variants, etc. 4) When done, `free(header.elements)` (or your custom deallocator).

Customization:

- Define `ADA_MALLOC`, `ADA_REALLOC`, and `ADA_ASSERT` before including this header to override allocation and assertion behavior.

Complexity (n = number of elements):

- Append: amortized O(1)
- Ordered insert/remove: O(n)
- Unordered insert/remove: O(1)

Notes and limitations:

- These are macros; arguments may be evaluated multiple times. Pass only simple lvalues (no side effects).
- Index checks rely on `ADA_ASSERT`; with `NDEBUG` they may be compiled out.
- `ada_resize` exits the process (`exit(1)`) if reallocation fails.
- `ada_insert` reads `header.elements[header.length - 1]` internally; inserting into an empty array via `ada_insert` is undefined behavior. Use `ada_appand` or `ada_insert_unordered` for that case.
- No automatic shrinking; you may call `ada_resize` manually.

Example: `typedef struct { size_t length; size_t capacity; int* elements; } ada_int_array;`

```
ada_int_array arr; ada\_init\_array\(int, arr\); ada\_appand\(int, arr, 42\); ada\_insert\(int, arr, 7, 0\); // requires arr.length > 0  
ada\_remove\(int, arr, 1\); free\(arr.elements\);
```

Definition in file [Almog_Dynamic_Array.h](#).

4.3.2 Macro Definition Documentation

4.3.2.1 ada_append

```
#define ada_append(
    type,
    header,
    value )
```

Value:

```
do { \
    if (header.length >= header.capacity) { \
        ada_resize(type, header, (int)(header.capacity*1.5)); \
    } \
    header.elements[header.length] = value; \
    header.length++; \
} while (0)
```

Append a value to the end of the array, growing if necessary.

Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>value</i>	Value to append.

Postcondition

header.length is incremented by 1; the last element equals *value*.

Note

Growth factor is $(\text{int})(\text{header.capacity} * 1.5)$. Because of truncation, very small capacities may not grow (e.g., from 1 to 1). With the default INIT_CAPACITY=10 this is typically not an issue unless you manually shrink capacity. Ensure growth always increases capacity by at least 1 if you customize this macro.

Definition at line 169 of file [Almog_Dynamic_Array.h](#).

4.3.2.2 ADA_ASSERT

```
#define ADA_ASSERT assert
```

Assertion macro used by this header (defaults to assert).

Define ADA_ASSERT before including this file to override. When NDEBUG is defined, standard assert() is disabled.

Definition at line 96 of file [Almog_Dynamic_Array.h](#).

4.3.2.3 ada_init_array

```
#define ada_init_array(
    type,
    header )
```

Value:

```
do { \
    header.capacity = ADA_INIT_CAPACITY; \
    header.length = 0; \
    header.elements = (type *)ADA_MALLOC(sizeof(type) * header.capacity); \
    ADA_ASSERT(header.elements != NULL); \
} while (0)
```

Initialize an array header and allocate its initial storage.

Parameters

<i>type</i>	Element type stored in the array (e.g., int).
<i>header</i>	Lvalue of the header struct containing fields: length, capacity, and elements.

Precondition

header is a modifiable lvalue; *header.elements* is uninitialized or ignored and will be overwritten.

Postcondition

header.length == 0, *header.capacity* == INIT_CAPACITY, *header.elements* != NULL (or ADA_ASSERT fails).

Note

Allocation uses ADA_MALLOC and is checked via ADA_ASSERT.

Definition at line 120 of file [Almog_Dynamic_Array.h](#).

4.3.2.4 ADA_INIT_CAPACITY

```
#define ADA_INIT_CAPACITY 10
```

Default initial capacity used by ada_init_array.

You may override this by defining INIT_CAPACITY before including this file.

Definition at line 64 of file [Almog_Dynamic_Array.h](#).

4.3.2.5 ada_insert

```
#define ada_insert(
    type,
    header,
    value,
    index )
```

Value:

```
do {
    ADA_ASSERT((int)(index) >= 0);
    ADA_ASSERT((float)(index) - (int)(index) == 0);
    ada_append(type, header, header.elements[header.length-1]);
    for (size_t ada_for_loop_index = header.length-2; ada_for_loop_index > (index); ada_for_loop_index--) {
        header.elements[ada_for_loop_index] = header.elements[ada_for_loop_index-1];
    }
    header.elements[(index)] = value;
} while (0)
```

Insert value at position index, preserving order (O(n)).

Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>value</i>	Value to insert.
<i>index</i>	Destination index in the range [0, header.length].

Precondition

$0 \leq \text{index} \leq \text{header.length}$.

$\text{header.length} > 0$ if $\text{index} == \text{header.length}$ (this macro reads the last element internally). For inserting into an empty array, use `ada_appand` or `ada_insert_unordered`.

Postcondition

Element is inserted at index; subsequent elements are shifted right; `header.length` is incremented by 1.

Note

This macro asserts index is non-negative and an integer value using `ADA_ASSERT`. No explicit upper-bound assert is performed.

Definition at line 196 of file [Almog_Dynamic_Array.h](#).

4.3.2.6 ada_insert_unordered

```
#define ada_insert_unordered(
```

`type,`
`header,`
`value,`
`index)`
Value:

```
do { \
    ADA_ASSERT((int)(index) >= 0); \
    ADA_ASSERT((float)(index) - (int)(index) == 0); \
    if ((size_t)(index) == header.length) { \
        ada_appand(type, header, value); \
    } else { \
        ada_appand(type, header, header.elements[(index)]); \
        header.elements[(index)] = value; \
    } \
} while (0)
```

Insert value at index without preserving order ($O(1)$ amortized).

If $\text{index} == \text{header.length}$, this behaves like an append. Otherwise, the current element at index is moved to the end, and value is written at index.

Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>value</i>	Value to insert.
<i>index</i>	Index in the range [0, header.length].

Precondition

$0 \leq index \leq header.length$.

Postcondition

$header.length$ is incremented by 1; array order is not preserved.

Definition at line 222 of file [Almog_Dynamic_Array.h](#).

4.3.2.7 ADA_MALLOC

```
#define ADA_MALLOC malloc
```

Allocation function used by this header (defaults to malloc).

Define ADA_MALLOC to a compatible allocator before including this file to override the default.

Definition at line 74 of file [Almog_Dynamic_Array.h](#).

4.3.2.8 ADA_REALLOC

```
#define ADA_REALLOC realloc
```

Reallocation function used by this header (defaults to realloc).

Define ADA_REALLOC to a compatible reallocator before including this file to override the default.

Definition at line 85 of file [Almog_Dynamic_Array.h](#).

4.3.2.9 ada_remove

```
#define ada_remove(
    type,
    header,
    index )
```

Value:

```
do {
    ADA_ASSERT((int)(index) >= 0);
    ADA_ASSERT((float)(index) - (int)(index) == 0);
    for (size_t ada_for_loop_index = (index); ada_for_loop_index < header.length-1; ada_for_loop_index++) {
        header.elements[ada_for_loop_index] = header.elements[ada_for_loop_index+1];
    }
    header.length--;
} while (0)
```

Remove element at index, preserving order ($O(n)$).

Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>index</i>	Index in the range [0, header.length - 1].

Precondition

$0 \leq index < header.length$.

Postcondition

header.length is decremented by 1; subsequent elements are shifted left by one position. The element beyond the new length is left uninitialized.

Definition at line 246 of file [Almog_Dynamic_Array.h](#).

4.3.2.10 ada_remove_unordered

```
#define ada_remove_unordered(
    type,
    header,
    index )
```

Value:

```
do {           \
    ADA_ASSERT((int)(index) >= 0);          \
    ADA_ASSERT((float)(index) - (int)(index) == 0); \
    header.elements[index] = header.elements[header.length-1]; \
    header.length--;                         \
} while (0)
```

Remove element at index by moving the last element into its place ($O(1)$); order is not preserved.

Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>index</i>	Index in the range [0, header.length - 1].

Precondition

$0 \leq index < header.length$ and $header.length > 0$.

Postcondition

header.length is decremented by 1; array order is not preserved.

Definition at line 267 of file [Almog_Dynamic_Array.h](#).

4.3.2.11 ada_resize

```
#define ada_resize(
    type,
    header,
    new_capacity )
```

Value:

```
do {
    type *ada_temp_pointer = (type *)ADA_REALLOC((void *)(header.elements), new_capacity*sizeof(type));
    \
    if (ada_temp_pointer == NULL) {
        \
        exit(1);
        \
    }
    \
    header.elements = ada_temp_pointer;
    \
    ADA_ASSERT(header.elements != NULL);
    \
    header.capacity = new_capacity;
} while (0)
```

Resize the underlying storage to hold new_capacity elements.

Parameters

<i>type</i>	Element type stored in the array.
<i>header</i>	Lvalue of the header struct.
<i>new_capacity</i>	New capacity in number of elements.

Precondition

new_capacity \geq *header.length* (otherwise elements beyond *new_capacity* are lost and *length* will not be adjusted).

Postcondition

header.capacity == *new_capacity* and *header.elements* points to a block large enough for *new_capacity* elements.

Warning

On allocation failure, this macro calls `exit(1)`.

Note

Reallocation uses `ADA_REALLOC` and is also checked via `ADA_ASSERT`.

Definition at line 143 of file [Almog_Dynamic_Array.h](#).

4.4 Almog_Dynamic_Array.h

```

00001
00051 #ifndef ALMOG_DYNAMIC_ARRAY_H_
00052 #define ALMOG_DYNAMIC_ARRAY_H_
00053
00054 #include <stdlib.h>
00055 #include <assert.h>
00056
00057
00064 #define ADA_INIT_CAPACITY 10
00065
00073 #ifndef ADA_MALLOC
00074 #define ADA_MALLOC malloc
00075 #endif /*ADA_MALLOC*/
00076
00084 #ifndef ADA_REALLOC
00085 #define ADA_REALLOC realloc
00086 #endif /*ADA_REALLOC*/
00087
00095 #ifndef ADA_ASSERT
00096 #define ADA_ASSERT assert
00097 #endif /*ADA_ASSERT*/
00098
00099 /* typedef struct {
00100     size_t length;
00101     size_t capacity;
00102     int* elements;
00103 } ada_int_array; */
00104
00120 #define ada_init_array(type, header) do {
00121     header.capacity = ADA_INIT_CAPACITY;
00122     header.length = 0;
00123     header.elements = (type *)ADA_MALLOC(sizeof(type) * header.capacity);
00124     ADA_ASSERT(header.elements != NULL);
00125 } while (0)
00126
00143 #define ada_resize(type, header, new_capacity) do {
00144     type *ada_temp_pointer = (type *)ADA_REALLOC((void *)header.elements,
00145     new_capacity*sizeof(type)); \
00146     if (ada_temp_pointer == NULL) {
00147         \
00148         exit(1);
00149     \
00150     header.elements = ada_temp_pointer;
00151     \
00152     ADA_ASSERT(header.elements != NULL);
00153     \
00154     header.capacity = new_capacity;
00155 } while (0)
00156
00169 #define ada_append(type, header, value) do {
00170     if (header.length >= header.capacity) {
00171         ada_resize(type, header, (int)(header.capacity*1.5));
00172     }
00173     header.elements[header.length] = value;
00174     header.length++;
00175 } while (0)
00176
00196 #define ada_insert(type, header, value, index) do {
00197     \
00198     ADA_ASSERT((int)(index) >= 0);
00199     \
00200     ADA_ASSERT((float)(index) - (int)(index) == 0);
00201     \
00202     ada_append(type, header, header.elements[header.length-1]);
00203     \
00204     for (size_t ada_for_loop_index = header.length-2; ada_for_loop_index > (index);
00205         ada_for_loop_index--) {
00206         \
00207         header.elements[ada_for_loop_index] = header.elements [ada_for_loop_index-1];
00208     }
00209     \
00210     header.elements[(index)] = value;
00211 } while (0)
00212
00222 #define ada_insert_unordered(type, header, value, index) do {
00223     \
00224     ADA_ASSERT((int)(index) >= 0);
00225     \
00226     ADA_ASSERT((float)(index) - (int)(index) == 0);
00227     \
00228     if ((size_t)(index) == header.length) {
00229         \
00230         ada_append(type, header, value);
00231     }
00232 }
```

```

00227     } else {
00228         ada_append(type, header, header.elements[(index)]);
00229         header.elements[(index)] = value;
00230     }
00231 } while (0)
00232
00246 #define ada_remove(type, header, index) do {
00247     \
00248     ADA_ASSERT((int)(index) >= 0);
00249     ADA_ASSERT((float)(index) - (int)(index) == 0);
00250     for (size_t ada_for_loop_index = (index); ada_for_loop_index < header.length-1;
00251         ada_for_loop_index++) {
00252         header.elements[ada_for_loop_index] = header.elements[ada_for_loop_index+1];
00253     }
00254     header.length--;
00255 } while (0)
00256
00267 #define ada_remove_unordered(type, header, index) do {
00268     ADA_ASSERT((int)(index) >= 0);
00269     ADA_ASSERT((float)(index) - (int)(index) == 0);
00270     header.elements[index] = header.elements[header.length-1];
00271     header.length--;
00272 } while (0)
00273
00274
00275 #endif /*ALMOG_DYNAMIC_ARRAY_H*/

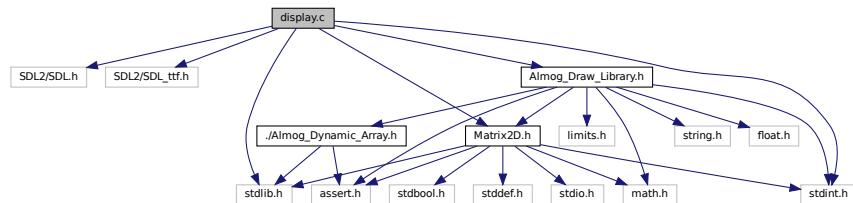
```

4.5 display.c File Reference

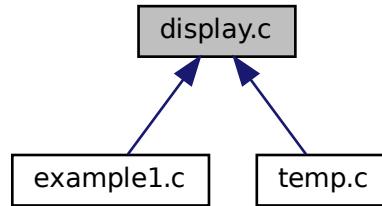
```

#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include "Matrix2D.h"
#include <stdlib.h>
#include <stdint.h>
#include "Almog_Draw_Library.h"
Include dependency graph for display.c:

```



This graph shows which files directly or indirectly include this file:



Classes

- struct `game_state_t`

Macros

- `#define WINDOW_WIDTH (16 * 80)`
- `#define WINDOW_HEIGHT (9 * 80)`
- `#define FPS 100`
- `#define FRAME_TARGET_TIME (1000 / FPS)`
- `#define dprintSTRING(expr) printf(#expr " = %s\n", expr)`
- `#define dprintCHAR(expr) printf(#expr " = %c\n", expr)`
- `#define dprintINT(expr) printf(#expr " = %d\n", expr)`
- `#define dprintD(expr) printf(#expr " = %g\n", expr)`
- `#define dprintSIZE_T(expr) printf(#expr " = %zu\n", expr)`
- `#define SETUP`
- `#define UPDATE`
- `#define RENDER`

Functions

- `int initialize_window (game_state_t *game_state)`
- `void setup_window (game_state_t *game_state)`
- `void process_input_window (game_state_t *game_state)`
- `void update_window (game_state_t *game_state)`
- `void render_window (game_state_t *game_state)`
- `void destroy_window (game_state_t *game_state)`
- `void fix_framerate (game_state_t *game_state)`
- `void setup (game_state_t *game_state)`
- `void update (game_state_t *game_state)`
- `void render (game_state_t *game_state)`
- `void check_window_mat_size (game_state_t *game_state)`
- `void copy_mat_to_surface_RGB (game_state_t *game_state)`
- `int main ()`

4.5.1 Macro Definition Documentation

4.5.1.1 dprintCHAR

```
#define dprintCHAR(  
    expr ) printf(#expr " = %c\n", expr)
```

Definition at line 25 of file [display.c](#).

4.5.1.2 dprintD

```
#define dprintD(  
    expr ) printf(#expr " = %g\n", expr)
```

Definition at line 27 of file [display.c](#).

4.5.1.3 dprintINT

```
#define dprintINT(  
    expr ) printf(#expr " = %d\n", expr)
```

Definition at line 26 of file [display.c](#).

4.5.1.4 dprintSIZE_T

```
#define dprintSIZE_T(  
    expr ) printf(#expr " = %zu\n", expr)
```

Definition at line 28 of file [display.c](#).

4.5.1.5 dprintSTRING

```
#define dprintSTRING(  
    expr ) printf(#expr " = %s\n", expr)
```

Definition at line 24 of file [display.c](#).

4.5.1.6 FPS

```
#define FPS 100
```

Definition at line 17 of file [display.c](#).

4.5.1.7 FRAME_TARGET_TIME

```
#define FRAME_TARGET_TIME (1000 / FPS)
```

Definition at line 21 of file [display.c](#).

4.5.1.8 RENDER

```
#define RENDER
```

Definition at line 351 of file [display.c](#).

4.5.1.9 SETUP

```
#define SETUP
```

Definition at line 341 of file [display.c](#).

4.5.1.10 UPDATE

```
#define UPDATE
```

Definition at line 346 of file [display.c](#).

4.5.1.11 WINDOW_HEIGHT

```
#define WINDOW_HEIGHT (9 * 80)
```

Definition at line 13 of file [display.c](#).

4.5.1.12 WINDOW_WIDTH

```
#define WINDOW_WIDTH (16 * 80)
```

Definition at line 9 of file [display.c](#).

4.5.2 Function Documentation

4.5.2.1 check_window_mat_size()

```
void check_window_mat_size (
    game_state_t * game_state )
```

Definition at line 355 of file [display.c](#).

References [Mat2D::uint32::cols](#), [game_state_t::inv_z_buffer_mat](#), [mat2D_alloc\(\)](#), [mat2D_alloc_uint32\(\)](#), [mat2D_free\(\)](#), [mat2D_free_uint32\(\)](#), [Mat2D::uint32::rows](#), [game_state_t::window](#), [game_state_t::window_h](#), [game_state_t::window_pixels_mat](#), [game_state_t::window_surface](#), and [game_state_t::window_w](#).

Referenced by [update_window\(\)](#).

4.5.2.2 copy_mat_to_surface_RGB()

```
void copy_mat_to_surface_RGB (
    game_state_t * game_state )
```

Definition at line 369 of file [display.c](#).

References [Mat2D::uint32::cols](#), [Mat2D::uint32::elements](#), [Mat2D::uint32::rows](#), [game_state_t::window_pixels_mat](#), and [game_state_t::window_surface](#).

Referenced by [render_window\(\)](#).

4.5.2.3 destroy_window()

```
void destroy_window (
    game_state_t * game_state )
```

Definition at line 312 of file [display.c](#).

References [mat2D_free_uint32\(\)](#), [game_state_t::renderer](#), [game_state_t::window](#), [game_state_t::window_pixels_mat](#), [game_state_t::window_surface](#), and [game_state_t::window_texture](#).

Referenced by [main\(\)](#).

4.5.2.4 fix_framerate()

```
void fix_framerate (
    game_state_t * game_state )
```

Definition at line 327 of file [display.c](#).

References [game_state_t::delta_time](#), [game_state_t::frame_target_time](#), [game_state_t::previous_frame_time](#), and [game_state_t::to_limit_fps](#).

Referenced by [update_window\(\)](#).

4.5.2.5 initialize_window()

```
int initialize_window (
    game_state_t * game_state )
```

Definition at line 141 of file [display.c](#).

References [game_state_t::renderer](#), [game_state_t::window](#), [game_state_t::window_h](#), and [game_state_t::window_w](#).

Referenced by [main\(\)](#).

4.5.2.6 main()

```
int main ( )
```

Definition at line 89 of file [display.c](#).

References [game_state_t::a_was_pressed](#), [game_state_t::const_fps](#), [game_state_t::d_was_pressed](#), [game_state_t::delta_time](#), [destroy_window\(\)](#), [game_state_t::e_was_pressed](#), [game_state_t::elapsed_time](#), [game_state_t::font](#), [FPS](#), [game_state_t::fps](#), [FRAME_TARGET_TIME](#), [game_state_t::frame_target_time](#), [game_state_t::game_is_running](#), [initialize_window\(\)](#), [game_state_t::left_button_pressed](#), [game_state_t::offset_zoom_param](#), [game_state_t::previous_frame_time](#), [process_input_window\(\)](#), [game_state_t::q_was_pressed](#), [render_window\(\)](#), [game_state_t::renderer](#), [game_state_t::s_was_pressed](#), [setup_window\(\)](#), [game_state_t::space_bar_was_pressed](#), [game_state_t::to_clear_renderer](#), [game_state_t::to_limit_fps](#), [game_state_t::to_render](#), [game_state_t::to_update](#), [update_window\(\)](#), [game_state_t::w_was_pressed](#), [game_state_t::window](#), [game_state_t::window_h](#), [WINDOW_HEIGHT](#), [game_state_t::window_w](#), [WINDOW_WIDTH](#), and [Offset_zoom_param::zoom_multiplier](#).

4.5.2.7 process_input_window()

```
void process_input_window (
    game_state_t * game_state )
```

Definition at line 196 of file [display.c](#).

References [ADL_MAX_ZOOM](#), [game_state_t::game_is_running](#), [game_state_t::left_button_pressed](#), [Offset_zoom_param::offset_x](#), [Offset_zoom_param::offset_y](#), [game_state_t::offset_zoom_param](#), [game_state_t::previous_frame_time](#), [game_state_t::space_bar_w](#), [game_state_t::to_render](#), [game_state_t::to_update](#), and [Offset_zoom_param::zoom_multiplier](#).

Referenced by [main\(\)](#).

4.5.2.8 render()

```
void render (
    game_state_t * game_state )
```

Definition at line 352 of file [display.c](#).

Referenced by [render_window\(\)](#).

4.5.2.9 render_window()

```
void render_window (
    game_state_t * game_state )
```

Definition at line 291 of file [display.c](#).

References [Mat2D::cols](#), [Mat2D_uint32::cols](#), [copy_mat_to_surface_RGB\(\)](#), [Mat2D::elements](#), [Mat2D_uint32::elements](#), [game_state_t::inv_z_buffer_mat](#), [render\(\)](#), [Mat2D::rows](#), [Mat2D_uint32::rows](#), [game_state_t::to_clear_renderer](#), [game_state_t::window](#), and [game_state_t::window_pixels_mat](#).

Referenced by [main\(\)](#).

4.5.2.10 setup()

```
void setup (
    game_state_t * game_state )
```

Definition at line 342 of file [display.c](#).

Referenced by [setup_window\(\)](#).

4.5.2.11 setup_window()

```
void setup_window (
    game_state_t * game_state )
```

Definition at line 182 of file [display.c](#).

References [game_state_t::inv_z_buffer_mat](#), [mat2D_alloc\(\)](#), [mat2D_alloc_uint32\(\)](#), [setup\(\)](#), [game_state_t::window](#), [game_state_t::window_h](#), [game_state_t::window_pixels_mat](#), [game_state_t::window_surface](#), and [game_state_t::window_w](#).

Referenced by [main\(\)](#).

4.5.2.12 update()

```
void update (
    game_state_t * game_state )
```

Definition at line 347 of file [display.c](#).

Referenced by [update_window\(\)](#).

4.5.2.13 update_window()

```
void update_window (
    game_state_t * game_state )
```

Definition at line 263 of file [display.c](#).

References [check_window_mat_size\(\)](#), [game_state_t::const_fps](#), [game_state_t::delta_time](#), [game_state_t::elapsed_time](#), [fix_framerate\(\)](#), [game_state_t::fps](#), [game_state_t::frame_target_time](#), [game_state_t::to_limit_fps](#), [update\(\)](#), [game_state_t::window](#), [game_state_t::window_h](#), and [game_state_t::window_w](#).

Referenced by [main\(\)](#).

4.6 display.c

```
00001 #include <SDL2/SDL.h>
00002 #include <SDL2/SDL_ttf.h>
00003 #include "Matrix2D.h"
00004 #include <stdlib.h>
00005 #include <stdint.h>
00006 #include "Almog_Draw_Library.h"
00007
00008 #ifndef WINDOW_WIDTH
00009 #define WINDOW_WIDTH (16 * 80)
00010#endif
00011
00012 #ifndef WINDOW_HEIGHT
00013 #define WINDOW_HEIGHT (9 * 80)
00014#endif
00015
00016 #ifndef FPS
00017 #define FPS 100
00018#endif
00019
00020 #ifndef FRAME_TARGET_TIME
00021 #define FRAME_TARGET_TIME (1000 / FPS)
00022#endif
00023
00024 #define dprintSTRING(expr) printf(#expr " = %s\n", expr)
00025 #define dprintCHAR(expr) printf(#expr " = %c\n", expr)
00026 #define dprintINT(expr) printf(#expr " = %d\n", expr)
00027 #define dprintD(expr) printf(#expr " = %g\n", expr)
00028 #define dprintSIZE_T(expr) printf(#expr " = %zu\n", expr)
00029
00030 #ifndef PI
00031     #ifndef __USE_MISC
00032         #define __USE_MISC
00033     #endif
00034     #include <math.h>
00035     #define PI M_PI
00036 #endif
00037
00038 typedef struct {
00039     int game_is_running;
00040     float delta_time;
00041     float elapsed_time;
00042     float const_fps;
00043     float fps;
```

```
00044     float frame_target_time;
00045     int to_render;
00046     int to_update;
00047     size_t previous_frame_time;
00048     int left_button_pressed;
00049     int to_limit_fps;
00050     int to_clear_renderer;
00051
00052     int space_bar_was_pressed;
00053     int w_was_pressed;
00054     int s_was_pressed;
00055     int a_was_pressed;
00056     int d_was_pressed;
00057     int e_was_pressed;
00058     int q_was_pressed;
00059
00060     SDL_Window *window;
00061     int window_w;
00062     int window_h;
00063     SDL_Renderer *renderer;
00064     TTF_Font *font;
00065
00066     SDL_Surface *window_surface;
00067     SDL_Texture *window_texture;
00068
00069     Mat2D_uint32 window_pixels_mat;
00070     Mat2D inv_z_buffer_mat;
00071
00072     Offset_zoom_param offset_zoom_param;
00073 } game_state_t;
00074
00075 int initialize_window(game_state_t *game_state);
00076 void setup_window(game_state_t *game_state);
00077 void process_input_window(game_state_t *game_state);
00078 void update_window(game_state_t *game_state);
00079 void render_window(game_state_t *game_state);
00080 void destroy_window(game_state_t *game_state);
00081 void fix_framerate(game_state_t *game_state);
00082 void setup(game_state_t *game_state);
00083 void update(game_state_t *game_state);
00084 void render(game_state_t *game_state);
00085
00086 void check_window_mat_size(game_state_t *game_state);
00087 void copy_mat_to_surface_RGB(game_state_t *game_state);
00088
00089 int main()
00090 {
00091     game_state_t game_state = {0};
00092
00093     game_state.game_is_running = 0;
00094     game_state.delta_time = 0;
00095     game_state.elapsed_time = 0;
00096     game_state.const_fps = FPS;
00097     game_state.fps = 0;
00098     game_state.frame_target_time = FRAME_TARGET_TIME;
00099
00100    game_state.space_bar_was_pressed = 0;
00101    game_state.w_was_pressed = 0;
00102    game_state.s_was_pressed = 0;
00103    game_state.a_was_pressed = 0;
00104    game_state.d_was_pressed = 0;
00105    game_state.e_was_pressed = 0;
00106    game_state.q_was_pressed = 0;
00107
00108    game_state.to_render = 1;
00109    game_state.to_update = 1;
00110    game_state.previous_frame_time = 0;
00111    game_state.left_button_pressed = 0;
00112    game_state.to_limit_fps = 1;
00113    game_state.to_clear_renderer = 1;
00114    game_state.window = NULL;
00115    game_state.window_w = WINDOW_WIDTH;
00116    game_state.window_h = WINDOW_HEIGHT;
00117    game_state.renderer = NULL;
00118    game_state.font = NULL;
00119
00120    game_state.offset_zoom_param.zoom_multiplier = 1;
00121
00122    game_state.game_is_running = !initialize_window(&game_state);
00123
00124    setup_window(&game_state);
00125
00126    while (game_state.game_is_running) {
00127        process_input_window(&game_state);
00128        if (game_state.to_update) {
00129            update_window(&game_state);
00130        }
00131    }
00132}
```

```

00131     if (game_state.to_render) {
00132         render_window(&game_state);
00133     }
00134 }
00135 destroy_window(&game_state);
00136
00137 return 0;
00138 }
00139 }
00140
00141 int initialize_window(game_state_t *game_state)
00142 {
00143     if (SDL_Init(SDL_INIT_EVERYTHING) != 0) {
00144         fprintf(stderr, "%s:%d: [Error] initializing SDL.\n", __FILE__, __LINE__);
00145         return -1;
00146     }
00147
00148     game_state->window = SDL_CreateWindow(NULL,
00149                                         SDL_WINDOWPOS_CENTERED,
00150                                         SDL_WINDOWPOS_CENTERED,
00151                                         game_state->window_w,
00152                                         game_state->window_h,
00153                                         SDL_WINDOW_RESIZABLE
00154                                         );
00155     if (!game_state->window) {
00156         fprintf(stderr, "%s:%d: [Error] creating SDL window.\n", __FILE__, __LINE__);
00157         return -1;
00158     }
00159
00160     game_state->renderer = SDL_CreateRenderer(game_state->window, -1, 0);
00161     if (!game_state->renderer) {
00162         fprintf(stderr, "%s:%d: [Error] creating SDL renderer.\n", __FILE__, __LINE__);
00163         return -1;
00164     }
00165
00166     if (TTF_Init() == -1) {
00167         fprintf(stderr, "%s:%d: [Error] initializing SDL_ttf.\n", __FILE__, __LINE__);
00168         return -1;
00169     }
00170
00171 // game_state->font = TTF_OpenFont("./font/Gabriely Black.ttf",32);
00172 // if (!game_state->font) {
00173 //     fprintf(stderr, "%s:%d: [Error] loading font.\n", __FILE__, __LINE__);
00174 //     return -1;
00175 // }
00176
00177 (void)game_state;
00178
00179 return 0;
00180 }
00181
00182 void setup_window(game_state_t *game_state)
00183 {
00184
00185     game_state->window_surface = SDL_GetWindowSurface(game_state->window);
00186
00187     game_state->window_pixels_mat = mat2D_alloc_uint32(game_state->window_h, game_state->window_w);
00188     game_state->inv_z_buffer_mat = mat2D_alloc(game_state->window_h, game_state->window_w);
00189
00190 /*-----*/
00191
00192     setup(game_state);
00193
00194 }
00195
00196 void process_input_window(game_state_t *game_state)
00197 {
00198     SDL_Event event;
00199     while (SDL_PollEvent(&event)) {
00200         switch (event.type) {
00201             case SDL_QUIT:
00202                 game_state->game_is_running = 0;
00203                 break;
00204             case SDL_KEYDOWN:
00205                 if (event.key.keysym.sym == SDLK_ESCAPE) {
00206                     game_state->game_is_running = 0;
00207                 }
00208                 if (event.key.keysym.sym == SDLK_SPACE) {
00209                     if (!game_state->space_bar_was_pressed) {
00210                         game_state->to_render = 0;
00211                         game_state->to_update = 0;
00212                         game_state->space_bar_was_pressed = 1;
00213                         break;
00214                     }
00215                     if (game_state->space_bar_was_pressed) {
00216                         game_state->to_render = 1;
00217                         game_state->to_update = 1;
00218                     }
00219                 }
00220             }
00221         }
00222     }
00223 }
```

```

00218             game_state->previous_frame_time = SDL_GetTicks();
00219             game_state->space_bar_was_pressed = 0;
00220             break;
00221         }
00222     }
00223     if (event.key.keysym.sym == SDLK_w) {
00224         game_state->offset_zoom_param.offset_y += 5/game_state->offset_zoom_param.zoom_multiplier;
00225     }
00226     if (event.key.keysym.sym == SDLK_s) {
00227         game_state->offset_zoom_param.offset_y -= 5/game_state->offset_zoom_param.zoom_multiplier;
00228     }
00229     if (event.key.keysym.sym == SDLK_a) {
00230         game_state->offset_zoom_param.offset_x += 5/game_state->offset_zoom_param.zoom_multiplier;
00231     }
00232     if (event.key.keysym.sym == SDLK_d) {
00233         game_state->offset_zoom_param.offset_x -= 5/game_state->offset_zoom_param.zoom_multiplier;
00234     }
00235     if (event.key.keysym.sym == SDLK_e) {
00236         game_state->offset_zoom_param.zoom_multiplier += 0.1*game_state->offset_zoom_param.zoom_multiplier;
00237         game_state->offset_zoom_param.zoom_multiplier =
00238             fminf(game_state->offset_zoom_param.zoom_multiplier, ADL_MAX_ZOOM);
00239     }
00240     if (event.key.keysym.sym == SDLK_q) {
00241         game_state->offset_zoom_param.zoom_multiplier -= 0.1*game_state->offset_zoom_param.zoom_multiplier;
00242         game_state->offset_zoom_param.zoom_multiplier =
00243             fminf(game_state->offset_zoom_param.zoom_multiplier, ADL_MAX_ZOOM);
00244     }
00245     if (event.key.keysym.sym == SDLK_r) {
00246         game_state->offset_zoom_param.zoom_multiplier = 1;
00247         game_state->offset_zoom_param.offset_x = 0;
00248         game_state->offset_zoom_param.offset_y = 0;
00249     }
00250     break;
00251 case SDL_MOUSEBUTTONDOWN:
00252     if (event.button.button == SDL_BUTTON_LEFT) {
00253         game_state->left_button_pressed = 1;
00254     }
00255     break;
00256 case SDL_MOUSEBUTTONUP:
00257     if (event.button.button == SDL_BUTTON_LEFT) {
00258         game_state->left_button_pressed = 0;
00259     }
00260 }
00261 }
00262
00263 void update_window(game_state_t *game_state)
00264 {
00265     SDL_GetWindowSize(game_state->window, &(game_state->>window_w), &(game_state->>window_h));
00266
00267     fix_framerate(game_state);
00268     game_state->elapsed_time += game_state->delta_time;
00269     game_state->fps = 1.0f / game_state->delta_time;
00270     game_state->frame_target_time = 1000/game_state->const_fps;
00271
00272     char fps_count[100];
00273     if (!game_state->to_limit_fps) {
00274         sprintf(fps_count, "dt = %5.02f [ms]", game_state->delta_time*1000);
00275     } else {
00276         sprintf(fps_count, "FPS = %5.2f", game_state->fps);
00277     }
00278
00279     if (game_state->elapsed_time*10-(int)(game_state->elapsed_time*10) < 0.1) {
00280         SDL_SetWindowTitle(game_state->window, fps_count);
00281     }
00282
00283     check_window_mat_size(game_state);
00284
00285     /*-----*/
00286
00287     update(game_state);
00288
00289 }
00290
00291 void render_window(game_state_t *game_state)
00292 {
00293     if (game_state->to_clear_renderer) {
00294         // SDL_SetRenderDrawColor(game_state->renderer, HexARGB_RGBA(0xFF181818));
00295         // SDL_RenderClear(game_state->renderer);
00296         // mat2D_fill(game_state->window_pixels_mat, 0x181818);

```

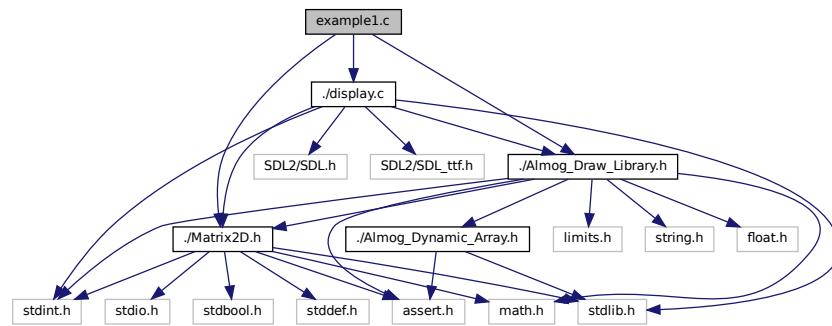
```

00297     memset(game_state->window_pixels_mat.elements, 0x20, sizeof(uint32_t) *
00298     game_state->window_pixels_mat.rows * game_state->window_pixels_mat.cols);
00299     /* not using mat2D_fill but using memset because it is way faster, so the buffer needs to be
00300     of 1/z */
00301     memset(game_state->inv_z_buffer_mat.elements, 0x0, sizeof(double) *
00302     game_state->inv_z_buffer_mat.rows * game_state->inv_z_buffer_mat.cols);
00303     }
00304     /*-----*/
00305     render(game_state);
00306     /*-----*/
00307     copy_mat_to_surface_RGB(game_state);
00308     SDL_UpdateWindowSurface(game_state->window);
00309
00310 }
00311
00312 void destroy_window(game_state_t *game_state)
00313 {
00314     mat2D_free_uint32(game_state->window_pixels_mat);
00315
00316     if (!game_state->window_surface) SDL_FreeSurface(game_state->window_surface);
00317     if (!game_state->window_texture) SDL_DestroyTexture(game_state->window_texture);
00318
00319     SDL_DestroyRenderer(game_state->renderer);
00320     SDL_DestroyWindow(game_state->window);
00321
00322     SDL_Quit();
00323
00324     (void)game_state;
00325 }
00326
00327 void fix_framerate(game_state_t *game_state)
00328 {
00329     int time_ellapsed = SDL_GetTicks() - game_state->previous_frame_time;
00330     int time_to_wait = game_state->frame_target_time - time_ellapsed;
00331     if (time_to_wait > 0 && time_to_wait < game_state->frame_target_time) {
00332         if (game_state->to_limit_fps) {
00333             SDL_Delay(time_to_wait);
00334         }
00335     }
00336     game_state->delta_time = (SDL_GetTicks() - game_state->previous_frame_time) / 1000.0f;
00337     game_state->previous_frame_time = SDL_GetTicks();
00338 }
00339
00340 #ifndef SETUP
00341 #define SETUP
00342 void setup(game_state_t *game_state) { (void)game_state; }
00343 #endif
00344
00345 #ifndef UPDATE
00346 #define UPDATE
00347 void update(game_state_t *game_state) { (void)game_state; }
00348 #endif
00349
00350 #ifndef RENDER
00351 #define RENDER
00352 void render(game_state_t *game_state) { (void)game_state; }
00353 #endif
00354
00355 void check_window_mat_size(game_state_t *game_state)
00356 {
00357     if (game_state->window_h != (int)game_state->window_pixels_mat.rows || game_state->window_w !=
00358     (int)game_state->window_pixels_mat.cols) {
00359         mat2D_free_uint32(game_state->window_pixels_mat);
00360         mat2D_free(game_state->inv_z_buffer_mat);
00361         SDL_FreeSurface(game_state->window_surface);
00362
00363         game_state->window_pixels_mat = mat2D_alloc_uint32(game_state->window_h,
00364         game_state->window_w);
00365         game_state->inv_z_buffer_mat = mat2D_alloc(game_state->window_h, game_state->window_w);
00366
00367         game_state->window_surface = SDL_GetWindowSurface(game_state->window);
00368     }
00369 }
00370
00371 void copy_mat_to_surface_RGB(game_state_t *game_state)
00372 {
00373     SDL_LockSurface(game_state->window_surface);
00374
00375     memcpy(game_state->window_surface->pixels, game_state->window_pixels_mat.elements,
00376     sizeof(uint32_t) * game_state->window_pixels_mat.rows * game_state->window_pixels_mat.cols);
00377
00378     SDL_UnlockSurface(game_state->window_surface);
00379 }

```

4.7 example1.c File Reference

```
#include "./Almog_Draw_Library.h"
#include "./display.c"
#include "./Matrix2D.h"
Include dependency graph for example1.c:
```



Macros

- #define SETUP
- #define UPDATE
- #define RENDER
- #define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
- #define MATRIX2D_IMPLEMENTATION

Functions

- void setup (game_state_t *game_state)
- void update (game_state_t *game_state)
- void render (game_state_t *game_state)

Variables

- Figure figure1
- Figure figure2
- Curve points
- Curve points1

4.7.1 Macro Definition Documentation

4.7.1.1 ALMOG_DRAW_LIBRARY_IMPLEMENTATION

```
#define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
```

Definition at line [4](#) of file [example1.c](#).

4.7.1.2 MATRIX2D_IMPLEMENTATION

```
#define MATRIX2D_IMPLEMENTATION
```

Definition at line [7](#) of file [example1.c](#).

4.7.1.3 RENDER

```
#define RENDER
```

Definition at line [3](#) of file [example1.c](#).

4.7.1.4 SETUP

```
#define SETUP
```

Definition at line [1](#) of file [example1.c](#).

4.7.1.5 UPDATE

```
#define UPDATE
```

Definition at line [2](#) of file [example1.c](#).

4.7.2 Function Documentation

4.7.2.1 render()

```
void render (
    game_state_t * game_state )
```

Definition at line 64 of file [example1.c](#).

References [adl_character_draw\(\)](#), [adl_curves_plot_on_figure\(\)](#), [ADL_DEFAULT_OFFSET_ZOOM](#), [adl_figure_copy_to_screen\(\)](#), [figure1](#), [figure2](#), and [game_state_t::window_pixels_mat](#).

4.7.2.2 setup()

```
void setup (
    game_state_t * game_state )
```

Definition at line 15 of file [example1.c](#).

References [ada_appand](#), [ada_init_array](#), [adl_curve_add_to_figure\(\)](#), [adl_figure_alloc\(\)](#), [Figure::background_color](#), [game_state_t::const_fps](#), [Curve::elements](#), [figure1](#), [figure2](#), [Curve::length](#), [points](#), [points1](#), [Figure::to_draw_axis](#), and [Figure::to_draw_max_min_values](#).

4.7.2.3 update()

```
void update (
    game_state_t * game_state )
```

Definition at line 60 of file [example1.c](#).

4.7.3 Variable Documentation

4.7.3.1 figure1

`Figure figure1`

Definition at line 11 of file [example1.c](#).

Referenced by [render\(\)](#), and [setup\(\)](#).

4.7.3.2 figure2

`Figure figure2`

Definition at line 12 of file [example1.c](#).

Referenced by [render\(\)](#), and [setup\(\)](#).

4.7.3.3 points

`Curve points`

Definition at line 13 of file [example1.c](#).

Referenced by [adl_lines_draw\(\)](#), [adl_lines_loop_draw\(\)](#), and [setup\(\)](#).

4.7.3.4 points1

`Curve points1`

Definition at line 14 of file [example1.c](#).

Referenced by [setup\(\)](#).

4.8 example1.c

```

00001 #define SETUP
00002 #define UPDATE
00003 #define RENDER
00004 #define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
00005 #include "./Almog_Draw_Library.h"
00006 #include "./display.c"
00007 #define MATRIX2D_IMPLEMENTATION
00008 #include "./Matrix2D.h"
00009
00010
00011 Figure figure1;
00012 Figure figure2;
00013 Curve points;
00014 Curve points1;
00015 void setup(game_state_t *game_state)
00016 {
00017     game_state->const_fps = 30;
00018     // game_state->to_limit_fps = 0;
00019
00020     figure1 = adl_figure_alloc(100, 70, (Point){100, 100, 0, 0});
00021     figure2 = adl_figure_alloc(600, 500, (Point){190, 100, 0, 0});
00022
00023     ada_init_array(Point, points);
00024     ada_init_array(Point, points1);
00025     Point temp_point = (Point){1,1,0,0};
00026     ada_appand(Point, points, temp_point);
00027     ada_appand(Point, points1, temp_point);
00028     temp_point = (Point){2,2,0,0};
00029     ada_appand(Point, points, temp_point);
00030     ada_appand(Point, points1, temp_point);
00031     temp_point = (Point){3,1,0,0};
00032     ada_appand(Point, points, temp_point);
00033     ada_appand(Point, points1, temp_point);
00034     temp_point = (Point){4,10,0,0};
00035     ada_appand(Point, points, temp_point);

```

```
00036     temp_point = (Point){5,-10,0,0};
00037     ada_appand(Point, points, temp_point);
00038     temp_point = (Point){3,-20,0,0};
00039     ada_appand(Point, points, temp_point);
00040
00041     temp_point = (Point){3.5,-10,0,0};
00042     ada_appand(Point, points1, temp_point);
00043
00044     figure1.background_color = 0xFFFFFFFF;
00045     figure1.to_draw_axis = true;
00046     figure1.to_draw_max_min_values = true;
00047
00048     figure2.background_color = 0xFFFFFFFF;
00049     figure2.to_draw_axis = true;
00050     figure2.to_draw_max_min_values = true;
00051
00052     adl_curve_add_to_figure(&figure1, points.elements, points.length, 0xFFFF0000);
00053     adl_curve_add_to_figure(&figure2, points.elements, points.length, 0xFFFF0000);
00054
00055     adl_curve_add_to_figure(&figure1, points1.elements, points1.length, 0xFF0000FF);
00056     adl_curve_add_to_figure(&figure2, points1.elements, points1.length, 0xFF0000FF);
00057
00058 }
00059
00060 void update(game_state_t *game_state)
00061 {
00062 }
00063
00064 void render(game_state_t *game_state)
00065 {
00066     adl_curves_plot_on_figure(figure1);
00067     adl_curves_plot_on_figure(figure2);
00068
00069     adl_figure_copy_to_screen(game_state->window_pixels_mat, figure1);
00070     adl_figure_copy_to_screen(game_state->window_pixels_mat, figure2);
00071
00072     adl_character_draw(game_state->window_pixels_mat, 'A', 50, 100, 700, 200, 0xFFFFFFFF,
00073 ADL_DEFAULT_OFFSET_ZOOM);
00074     adl_character_draw(game_state->window_pixels_mat, 'B', 50, 100, 755, 200, 0xFFFFFFFF,
00075 ADL_DEFAULT_OFFSET_ZOOM);
00076     adl_character_draw(game_state->window_pixels_mat, 'C', 50, 100, 810, 200, 0xFFFFFFFF,
00077 ADL_DEFAULT_OFFSET_ZOOM);
00078     adl_character_draw(game_state->window_pixels_mat, 'D', 50, 100, 865, 200, 0xFFFFFFFF,
00079 ADL_DEFAULT_OFFSET_ZOOM);
00080     adl_character_draw(game_state->window_pixels_mat, 'E', 50, 100, 920, 200, 0xFFFFFFFF,
00081 ADL_DEFAULT_OFFSET_ZOOM);
00082     adl_character_draw(game_state->window_pixels_mat, 'F', 50, 100, 975, 200, 0xFFFFFFFF,
00083 ADL_DEFAULT_OFFSET_ZOOM);
00084     adl_character_draw(game_state->window_pixels_mat, 'G', 50, 100, 1030, 200, 0xFFFFFFFF,
00085 ADL_DEFAULT_OFFSET_ZOOM);
00086     adl_character_draw(game_state->window_pixels_mat, 'H', 50, 100, 1085, 200, 0xFFFFFFFF,
00087 ADL_DEFAULT_OFFSET_ZOOM);
00088     adl_character_draw(game_state->window_pixels_mat, 'I', 50, 100, 1140, 200, 0xFFFFFFFF,
00089 ADL_DEFAULT_OFFSET_ZOOM);
00090     adl_character_draw(game_state->window_pixels_mat, 'J', 50, 100, 1195, 200, 0xFFFFFFFF,
00091 ADL_DEFAULT_OFFSET_ZOOM);
00092     adl_character_draw(game_state->window_pixels_mat, 'K', 50, 100, 700, 305, 0xFFFFFFFF,
00093 ADL_DEFAULT_OFFSET_ZOOM);
00094     adl_character_draw(game_state->window_pixels_mat, 'L', 50, 100, 755, 305, 0xFFFFFFFF,
00095 ADL_DEFAULT_OFFSET_ZOOM);
00096     adl_character_draw(game_state->window_pixels_mat, 'M', 50, 100, 810, 305, 0xFFFFFFFF,
00097 ADL_DEFAULT_OFFSET_ZOOM);
00098     adl_character_draw(game_state->window_pixels_mat, 'N', 50, 100, 865, 305, 0xFFFFFFFF,
00099 ADL_DEFAULT_OFFSET_ZOOM);
00100     adl_character_draw(game_state->window_pixels_mat, 'O', 50, 100, 920, 305, 0xFFFFFFFF,
00101 ADL_DEFAULT_OFFSET_ZOOM);
00102     adl_character_draw(game_state->window_pixels_mat, 'P', 50, 100, 975, 305, 0xFFFFFFFF,
00103 ADL_DEFAULT_OFFSET_ZOOM);
00104     adl_character_draw(game_state->window_pixels_mat, 'Q', 50, 100, 1030, 305, 0xFFFFFFFF,
00105 ADL_DEFAULT_OFFSET_ZOOM);
00106     adl_character_draw(game_state->window_pixels_mat, 'R', 50, 100, 1085, 305, 0xFFFFFFFF,
00107 ADL_DEFAULT_OFFSET_ZOOM);
00108     adl_character_draw(game_state->window_pixels_mat, 'S', 50, 100, 1140, 305, 0xFFFFFFFF,
00109 ADL_DEFAULT_OFFSET_ZOOM);
00110     adl_character_draw(game_state->window_pixels_mat, 'T', 50, 100, 1195, 305, 0xFFFFFFFF,
00111 ADL_DEFAULT_OFFSET_ZOOM);
00112     adl_character_draw(game_state->window_pixels_mat, 'U', 50, 100, 700, 410, 0xFFFFFFFF,
00113 ADL_DEFAULT_OFFSET_ZOOM);
00114     adl_character_draw(game_state->window_pixels_mat, 'V', 50, 100, 755, 410, 0xFFFFFFFF,
00115 ADL_DEFAULT_OFFSET_ZOOM);
00116     adl_character_draw(game_state->window_pixels_mat, 'W', 50, 100, 810, 410, 0xFFFFFFFF,
00117 ADL_DEFAULT_OFFSET_ZOOM);
00118     adl_character_draw(game_state->window_pixels_mat, 'X', 50, 100, 865, 410, 0xFFFFFFFF,
00119 ADL_DEFAULT_OFFSET_ZOOM);
00120     adl_character_draw(game_state->window_pixels_mat, 'Y', 50, 100, 920, 410, 0xFFFFFFFF,
00121 ADL_DEFAULT_OFFSET_ZOOM);
```

```

00098     adl_character_draw(game_state->window_pixels_mat, 'Z', 50, 100, 975 , 410, 0xFFFFFFFF,
00099         ADL_DEFAULT_OFFSET_ZOOM);
00100     adl_character_draw(game_state->window_pixels_mat, '.', 50, 100, 1030, 410, 0xFFFFFFFF,
00101         ADL_DEFAULT_OFFSET_ZOOM);
00102     adl_character_draw(game_state->window_pixels_mat, ':', 50, 100, 1085, 410, 0xFFFFFFFF,
00103         ADL_DEFAULT_OFFSET_ZOOM);
00104     adl_character_draw(game_state->window_pixels_mat, '0', 50, 100, 700 , 515, 0xFFFFFFFF,
00105         ADL_DEFAULT_OFFSET_ZOOM);
00106     adl_character_draw(game_state->window_pixels_mat, '1', 50, 100, 755 , 515, 0xFFFFFFFF,
00107         ADL_DEFAULT_OFFSET_ZOOM);
00108     adl_character_draw(game_state->window_pixels_mat, '2', 50, 100, 810 , 515, 0xFFFFFFFF,
00109         ADL_DEFAULT_OFFSET_ZOOM);
00110     adl_character_draw(game_state->window_pixels_mat, '3', 50, 100, 865 , 515, 0xFFFFFFFF,
00111         ADL_DEFAULT_OFFSET_ZOOM);
00112 }
00113

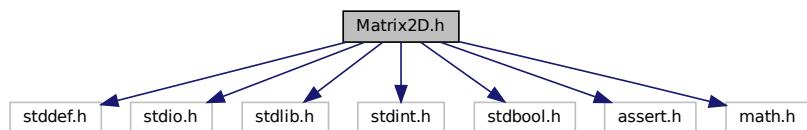
```

4.9 Matrix2D.h File Reference

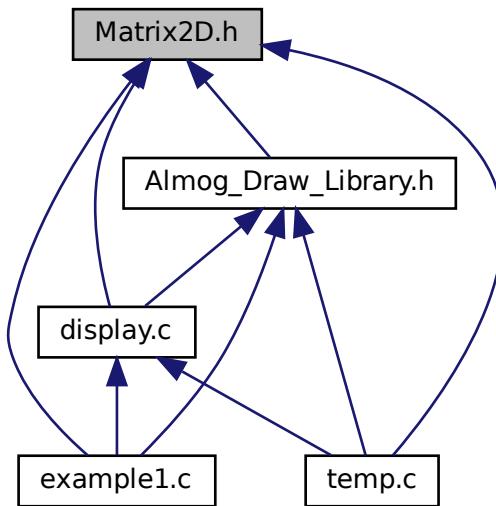
A single-header C library for simple 2D matrix operations on doubles and uint32_t, including allocation, basic arithmetic, linear algebra, and helpers (LUP, inverse, determinant, DCM, etc.).

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <assert.h>
#include <math.h>
```

Include dependency graph for Matrix2D.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [Mat2D](#)
Dense row-major matrix of doubles.
- struct [Mat2D_uint32](#)
Dense row-major matrix of uint32_t.
- struct [Mat2D_Minor](#)
A minor "view" into a reference matrix.

Macros

- #define [MATRIX2D_MALLOC](#) malloc
Allocation function used by the library.
- #define [MATRIX2D_ASSERT](#) assert
Assertion macro used by the library for parameter validation.
- #define [MAT2D_AT](#)(m, i, j) (m).elements[i * m.stride_r + j]
Access element (i, j) of a [Mat2D](#) (0-based).
- #define [MAT2D_AT_UINT32](#)(m, i, j) (m).elements[i * m.stride_r + j]
Access element (i, j) of a [Mat2D_uint32](#) (0-based).
- #define [__USE_MISC](#)
- #define [PI](#) M_PI
- #define [MAT2D_MINOR_AT](#)(mm, i, j) [MAT2D_AT](#)(mm.ref_mat, mm.rows_list[i], mm.cols_list[j])
Access element (i, j) of a [Mat2D_Minor](#) (0-based), dereferencing into the underlying reference matrix.
- #define [MAT2D_PRINT](#)(m) [mat2D_print](#)(m, #m, 0)
Convenience macro to print a matrix with its variable name.
- #define [MAT2D_PRINT_AS_COL](#)(m) [mat2D_print_as_col](#)(m, #m, 0)

- Convenience macro to print a matrix as a single column with its name.
- `#define MAT2D_MINOR_PRINT(mm) mat2D_minor_print(mm, #mm, 0)`
Convenience macro to print a minor with its variable name.
- `#define mat2D_normalize(m) mat2D_mult((m), 1.0 / mat2D_calc_norma((m)))`
In-place normalization of all elements so that the Frobenius norm becomes 1.

Functions

- `double mat2D_rand_double (void)`
Return a pseudo-random double in the range [0, 1].
- `Mat2D mat2D_alloc (size_t rows, size_t cols)`
Allocate a rows x cols matrix of doubles.
- `Mat2D_uint32 mat2D_alloc_uint32 (size_t rows, size_t cols)`
Allocate a rows x cols matrix of uint32_t.
- `void mat2D_free (Mat2D m)`
Free the memory owned by a Mat2D (elements pointer).
- `void mat2D_free_uint32 (Mat2D_uint32 m)`
Free the memory owned by a Mat2D_uint32 (elements pointer).
- `size_t mat2D_offset2d (Mat2D m, size_t i, size_t j)`
Compute the linear offset of element (i, j) in a Mat2D.
- `size_t mat2D_offset2d_uint32 (Mat2D_uint32 m, size_t i, size_t j)`
Compute the linear offset of element (i, j) in a Mat2D_uint32.
- `void mat2D_fill (Mat2D m, double x)`
Fill all elements of a matrix of doubles with a scalar value.
- `void mat2D_fill_sequence (Mat2D m, double start, double step)`
Fill a matrix with an arithmetic sequence laid out in row-major order.
- `void mat2D_fill_uint32 (Mat2D_uint32 m, uint32_t x)`
Fill all elements of a matrix of uint32_t with a scalar value.
- `void mat2D_rand (Mat2D m, double low, double high)`
Fill a matrix with random doubles in [low, high].
- `void mat2D_dot (Mat2D dst, Mat2D a, Mat2D b)`
*Matrix product: dst = a * b.*
- `double mat2D_dot_product (Mat2D a, Mat2D b)`
Dot product between two vectors.
- `void mat2D_cross (Mat2D dst, Mat2D a, Mat2D b)`
3D cross product: dst = a x b for 3x1 vectors.
- `void mat2D_add (Mat2D dst, Mat2D a)`
In-place addition: dst += a.
- `void mat2D_add_row_time_factor_to_row (Mat2D m, size_t des_r, size_t src_r, double factor)`
*Row operation: row(des_r) += factor * row(src_r).*
- `void mat2D_sub (Mat2D dst, Mat2D a)`
In-place subtraction: dst -= a.
- `void mat2D_sub_row_time_factor_to_row (Mat2D m, size_t des_r, size_t src_r, double factor)`
*Row operation: row(des_r) -= factor * row(src_r).*
- `void mat2D_mult (Mat2D m, double factor)`
*In-place scalar multiplication: m *= factor.*
- `void mat2D_mult_row (Mat2D m, size_t r, double factor)`
*In-place row scaling: row(r) *= factor.*
- `void mat2D_print (Mat2D m, const char *name, size_t padding)`
Print a matrix to stdout with a name and indentation padding.

- void `mat2D_print_as_col (Mat2D m, const char *name, size_t padding)`
Print a matrix as a flattened column vector to stdout.
- void `mat2D_set_identity (Mat2D m)`
Set a square matrix to the identity matrix.
- double `mat2D_make_identity (Mat2D m)`
Reduce a matrix to identity via Gauss-Jordan elimination and return the cumulative scaling factor.
- void `mat2D_set_rot_mat_x (Mat2D m, float angle_deg)`
Set a 3x3 rotation matrix for rotation about the X-axis.
- void `mat2D_set_rot_mat_y (Mat2D m, float angle_deg)`
Set a 3x3 rotation matrix for rotation about the Y-axis.
- void `mat2D_set_rot_mat_z (Mat2D m, float angle_deg)`
Set a 3x3 rotation matrix for rotation about the Z-axis.
- void `mat2D_set_DCM_zyx (Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg)`
Build a 3x3 direction cosine matrix (DCM) from Z-Y-X Euler angles.
- void `mat2D_copy (Mat2D des, Mat2D src)`
Copy all elements from src to des.
- void `mat2D_copy_mat_to_mat_at_window (Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)`
Copy a rectangular window from src into des.
- void `mat2D_get_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)`
Copy a column from src into a column of des.
- void `mat2D_add_col_to_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)`
Add a source column into a destination column: $des[:, des_col] += src[:, src_col]$.
- void `mat2D_sub_col_to_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)`
Subtract a source column from a destination column: $des[:, des_col] -= src[:, src_col]$.
- void `mat2D_swap_rows (Mat2D m, size_t r1, size_t r2)`
Swap two rows of a matrix in-place.
- void `mat2D_get_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)`
Copy a row from src into a row of des.
- void `mat2D_add_row_to_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)`
Add a source row into a destination row: $des[des_row, :] += src[src_row, :]$.
- void `mat2D_sub_row_to_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)`
Subtract a source row from a destination row: $des[des_row, :] -= src[src_row, :]$.
- double `mat2D_calc_norma (Mat2D m)`
Compute the Frobenius norm of a matrix, $\sqrt{\sum(m_ij^2)}$.
- bool `mat2D_mat_is_all_digit (Mat2D m, double digit)`
Check if all elements of a matrix equal a given digit.
- bool `mat2D_row_is_all_digit (Mat2D m, double digit, size_t r)`
Check if all elements of a row equal a given digit.
- bool `mat2D_col_is_all_digit (Mat2D m, double digit, size_t c)`
Check if all elements of a column equal a given digit.
- double `mat2D_det_2x2_mat (Mat2D m)`
Determinant of a 2x2 matrix.
- double `mat2D_triangulate (Mat2D m)`
Forward elimination to transform a matrix to upper triangular form.
- double `mat2D_det (Mat2D m)`
Determinant of an NxN matrix via Gaussian elimination.
- void `mat2D_LUP_decomposition_with_swap (Mat2D src, Mat2D l, Mat2D p, Mat2D u)`
Compute LUP decomposition: $P \cdot A = L \cdot U$ with L unit diagonal.
- void `mat2D_transpose (Mat2D des, Mat2D src)`
Transpose a matrix: $des = src^T$.
- void `mat2D_invert (Mat2D des, Mat2D src)`

- Invert a square matrix using Gauss-Jordan elimination.*
- void `mat2D_solve_linear_sys_LUP_decomposition` (`Mat2D A, Mat2D x, Mat2D B)`
Solve the linear system $A x = B$ using LUP decomposition.
 - `Mat2D_Minor mat2D_minor_alloc_fill_from_mat` (`Mat2D ref_mat, size_t i, size_t j)`
Allocate a minor view by excluding row i and column j of `ref_mat`.
 - `Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor` (`Mat2D_Minor ref_mm, size_t i, size_t j)`
Allocate a nested minor view from an existing minor by excluding row i and column j of the minor.
 - void `mat2D_minor_free` (`Mat2D_Minor mm)`
Free the index arrays owned by a minor.
 - void `mat2D_minor_print` (`Mat2D_Minor mm, const char *name, size_t padding)`
Print a minor matrix to stdout with a name and indentation padding.
 - double `mat2D_det_2x2_mat_minor` (`Mat2D_Minor mm)`
Determinant of a 2x2 minor.
 - double `mat2D_minor_det` (`Mat2D_Minor mm)`
Determinant of a minor via recursive expansion by minors.

4.9.1 Detailed Description

A single-header C library for simple 2D matrix operations on doubles and `uint32_t`, including allocation, basic arithmetic, linear algebra, and helpers (LUP, inverse, determinant, DCM, etc.).

- Storage is contiguous row-major (C-style). The element at row i , column j (0-based) is located at `elements[i * stride_r + j]`.
- Dense matrices of `double` are represented by `Mat2D`, and dense matrices of `uint32_t` are represented by `Mat2D_uint32`.
- Some routines assert shape compatibility using `MATRIX2D_ASSERT`.
- Random number generation uses the C library `rand()`; it is not cryptographically secure.
- Inversion is done via Gauss-Jordan elimination with partial pivoting only when a pivot is zero; this can be numerically unstable for ill-conditioned matrices. See notes below.
- To compile the implementation, define `MATRIX2D_IMPLEMENTATION` in exactly one translation unit before including this header.

Example: `#define MATRIX2D_IMPLEMENTATION #include "matrix2d.h"`

Note

This one-file library is heavily inspired by Tsoding's `nn.h` implementation of matrix creation and operations:
<https://github.com/tsoding/nn.h> and the video: <https://youtu.be/L1TbWe8b←VOc?list=PLpM-Dvs8t0VZPZKggcql-MmjBdZKeDMw>

Warning

Numerical stability:

- There is a set of functions for minors that can be used to compute the determinant, but that approach is factorial in complexity and too slow for larger matrices. This library uses Gaussian elimination instead.
- The inversion function can fail or be unstable if pivot values become very small. Consider preconditioning or using a more robust decomposition (e.g., full pivoting, SVD) for ill-conditioned problems.

Definition in file [Matrix2D.h](#).

4.9.2 Macro Definition Documentation

4.9.2.1 __USE_MISC

```
#define __USE_MISC
```

Definition at line 151 of file [Matrix2D.h](#).

4.9.2.2 MAT2D_AT

```
#define MAT2D_AT(  
    m,  
    i,  
    j ) (m).elements[i * m.stride_r + j]
```

Access element (i, j) of a [Mat2D](#) (0-based).

Warning

This macro does not perform bounds checking in the fast configuration. Use carefully.

Definition at line 145 of file [Matrix2D.h](#).

4.9.2.3 MAT2D_AT_UINT32

```
#define MAT2D_AT_UINT32(  
    m,  
    i,  
    j ) (m).elements[i * m.stride_r + j]
```

Access element (i, j) of a [Mat2D_uint32](#) (0-based).

Warning

This macro does not perform bounds checking in the fast configuration. Use carefully.

Definition at line 146 of file [Matrix2D.h](#).

4.9.2.4 MAT2D_MINOR_AT

```
#define MAT2D_MINOR_AT (
    mm,
    i,
    j ) MAT2D_AT (mm.ref_mat, mm.rows_list[i], mm.cols_list[j])
```

Access element (i, j) of a [Mat2D_Minor](#) (0-based), dereferencing into the underlying reference matrix.

Definition at line 162 of file [Matrix2D.h](#).

4.9.2.5 MAT2D_MINOR_PRINT

```
#define MAT2D_MINOR_PRINT (
    mm ) mat2D_minor_print (mm, #mm, 0)
```

Convenience macro to print a minor with its variable name.

Definition at line 177 of file [Matrix2D.h](#).

4.9.2.6 mat2D_normalize

```
#define mat2D_normalize(
    m ) mat2D_mult (m, 1.0 / mat2D_calc_norma (m))
```

In-place normalization of all elements so that the Frobenius norm becomes 1.

Equivalent to: $m *= 1.0 / \text{mat2D_calc_norma}(m)$.

Definition at line 184 of file [Matrix2D.h](#).

4.9.2.7 MAT2D_PRINT

```
#define MAT2D_PRINT (
    m ) mat2D_print (m, #m, 0)
```

Convenience macro to print a matrix with its variable name.

Definition at line 167 of file [Matrix2D.h](#).

4.9.2.8 MAT2D_PRINT_AS_COL

```
#define MAT2D_PRINT_AS_COL( m ) mat2D_print_as_col(m, #m, 0)
```

Convenience macro to print a matrix as a single column with its name.

Definition at line [172](#) of file [Matrix2D.h](#).

4.9.2.9 MATRIX2D_ASSERT

```
#define MATRIX2D_ASSERT assert
```

Assertion macro used by the library for parameter validation.

Defaults to C `assert`. Override by defining `MATRIX2D_ASSERT` before including this header if you want custom behavior.

Definition at line [68](#) of file [Matrix2D.h](#).

4.9.2.10 MATRIX2D_MALLOC

```
#define MATRIX2D_MALLOC malloc
```

Allocation function used by the library.

Defaults to `malloc`. Override by defining `MATRIX2D_MALLOC` before including this header if you want to use a custom allocator.

Definition at line [56](#) of file [Matrix2D.h](#).

4.9.2.11 PI

```
#define PI M_PI
```

Definition at line [154](#) of file [Matrix2D.h](#).

4.9.3 Function Documentation

4.9.3.1 mat2D_add()

```
void mat2D_add ( Mat2D dst, Mat2D a )
```

In-place addition: `dst += a`.

Parameters

<i>dst</i>	Destination matrix to be incremented.
<i>a</i>	Summand of same shape as dst.

Precondition

Shapes match.

Definition at line 496 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

Referenced by [adl_arrow_draw\(\)](#).

4.9.3.2 mat2D_add_col_to_col()

```
void mat2D_add_col_to_col (
    Mat2D des,
    size_t des_col,
    Mat2D src,
    size_t src_col )
```

Add a source column into a destination column: $\text{des}[:, \text{des_col}] += \text{src}[:, \text{src_col}]$.

Parameters

<i>des</i>	Destination matrix (same row count as src).
<i>des_col</i>	Column index in destination.
<i>src</i>	Source matrix.
<i>src_col</i>	Column index in source.

Definition at line 828 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.3 mat2D_add_row_time_factor_to_row()

```
void mat2D_add_row_time_factor_to_row (
    Mat2D m,
    size_t des_r,
    size_t src_r,
    double factor )
```

Row operation: $\text{row}(\text{des_r}) += \text{factor} * \text{row}(\text{src_r})$.

Parameters

<i>m</i>	Matrix.
<i>des</i> \leftarrow	Destination row index.
<i>_r</i>	
<i>src</i> \leftarrow	Source row index.
<i>_r</i>	
<i>factor</i>	Scalar multiplier.

Definition at line 514 of file [Matrix2D.h](#).

References [Mat2D::cols](#), and [MAT2D_AT](#).

4.9.3.4 mat2D_add_row_to_row()

```
void mat2D_add_row_to_row (
    Mat2D des,
    size_t des_row,
    Mat2D src,
    size_t src_row )
```

Add a source row into a destination row: $des[des_row, :] += src[src_row, :]$.

Parameters

<i>des</i>	Destination matrix (same number of columns as <i>src</i>).
<i>des_row</i>	Row index in destination.
<i>src</i>	Source matrix.
<i>src_row</i>	Row index in source.

Definition at line 897 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.5 mat2D_alloc()

```
Mat2D mat2D_alloc (
    size_t rows,
    size_t cols )
```

Allocate a *rows* x *cols* matrix of doubles.

Parameters

<i>rows</i>	Number of rows (≥ 1).
<i>cols</i>	Number of columns (≥ 1).

Returns

A [Mat2D](#) with contiguous storage; must be freed with [mat2D_free](#).

Postcondition

`m.stride_r == cols.`

Definition at line 278 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [Mat2D::elements](#), [MATRIX2D_ASSERT](#), [MATRIX2D_MALLOC](#), [Mat2D::rows](#), and [Mat2D::stride_r](#).

Referenced by [adl_arrow_draw\(\)](#), [adl_figure_alloc\(\)](#), [check_window_mat_size\(\)](#), [mat2D_det\(\)](#), [mat2D_invert\(\)](#), [mat2D_set_DCM_zyx\(\)](#), [mat2D_solve_linear_sys_LUP_decomposition\(\)](#), and [setup_window\(\)](#).

4.9.3.6 mat2D_alloc_uint32()

```
Mat2D_uint32 mat2D_alloc_uint32 (
    size_t rows,
    size_t cols )
```

Allocate a `rows x cols` matrix of `uint32_t`.

Parameters

<code>rows</code>	Number of rows (≥ 1).
<code>cols</code>	Number of columns (≥ 1).

Returns

A [Mat2D_uint32](#) with contiguous storage; free with [mat2D_free_uint32](#).

Postcondition

`m.stride_r == cols.`

Definition at line 297 of file [Matrix2D.h](#).

References [Mat2D_uint32::cols](#), [Mat2D_uint32::elements](#), [MATRIX2D_ASSERT](#), [MATRIX2D_MALLOC](#), [Mat2D_uint32::rows](#), and [Mat2D_uint32::stride_r](#).

Referenced by [adl_figure_alloc\(\)](#), [check_window_mat_size\(\)](#), and [setup_window\(\)](#).

4.9.3.7 mat2D_calc_norma()

```
double mat2D_calc_norma (
    Mat2D m )
```

Compute the Frobenius norm of a matrix, $\sqrt{\sum(m_{ij})^2}$.

Parameters

<i>m</i>	Matrix.
----------	---------

Returns

Frobenius norm.

Definition at line 931 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), and [Mat2D::rows](#).

4.9.3.8 mat2D_col_is_all_digit()

```
bool mat2D_col_is_all_digit (
    Mat2D m,
    double digit,
    size_t c )
```

Check if all elements of a column equal a given digit.

Parameters

<i>m</i>	Matrix.
<i>digit</i>	Value to compare.
<i>c</i>	Column index.

Returns

true if every element equals digit, false otherwise.

Definition at line 985 of file [Matrix2D.h](#).

References [Mat2D::cols](#), and [MAT2D_AT](#).

Referenced by [mat2D_det\(\)](#).

4.9.3.9 mat2D_copy()

```
void mat2D_copy (
    Mat2D des,
    Mat2D src )
```

Copy all elements from src to des.

Parameters

<i>des</i>	Destination matrix.
<i>src</i>	Source matrix.

Precondition

Shapes match.

Definition at line 768 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

Referenced by [adl_arrow_draw\(\)](#), [mat2D_det\(\)](#), [mat2D_invert\(\)](#), and [mat2D_LUP_decomposition_with_swap\(\)](#).

4.9.3.10 mat2D_copy_mat_to_mat_at_window()

```
void mat2D_copy_mat_to_mat_at_window (
    Mat2D des,
    Mat2D src,
    size_t is,
    size_t js,
    size_t ie,
    size_t je )
```

Copy a rectangular window from src into des.

Parameters

<i>des</i>	Destination matrix. Must have size (ie - is + 1) x (je - js + 1).
<i>src</i>	Source matrix.
<i>is</i>	Start row index in src (inclusive).
<i>js</i>	Start column index in src (inclusive).
<i>ie</i>	End row index in src (inclusive).
<i>je</i>	End column index in src (inclusive).

Precondition

$0 \leq is \leq ie < src.rows$, $0 \leq js \leq je < src.cols$.

Definition at line 790 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.11 mat2D_cross()

```
void mat2D_cross (
    Mat2D dst,
    Mat2D a,
    Mat2D b )
```

3D cross product: $\text{dst} = \mathbf{a} \times \mathbf{b}$ for 3×1 vectors.

Parameters

<i>dst</i>	3x1 destination vector.
<i>a</i>	3x1 input vector.
<i>b</i>	3x1 input vector.

Precondition

All matrices have shape 3×1 .

Definition at line 479 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.12 mat2D_det()

```
double mat2D_det (
    Mat2D m )
```

Determinant of an $N \times N$ matrix via Gaussian elimination.

Parameters

<i>m</i>	Square matrix.
----------	----------------

Returns

$\det(m)$.

Copies *m* internally, triangulates it, and returns the product of diagonal elements (adjusted by any scaling factor as implemented).

Definition at line 1052 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [mat2D_alloc\(\)](#), [MAT2D_AT](#), [mat2D_col_is_all_digit\(\)](#), [mat2D_copy\(\)](#), [mat2D_free\(\)](#), [mat2D_row_is_all_digit\(\)](#), [mat2D_triangulate\(\)](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

Referenced by [mat2D_invert\(\)](#).

4.9.3.13 mat2D_det_2x2_mat()

```
double mat2D_det_2x2_mat (
    Mat2D m )
```

Determinant of a 2x2 matrix.

Parameters

<i>m</i>	Matrix (must be 2x2).
----------	-----------------------

Returns

$\det(m) = a_{11} a_{22} - a_{12} a_{21}$.

Definition at line 1000 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.14 mat2D_det_2x2_mat_minor()

```
double mat2D_det_2x2_mat_minor (
    Mat2D_Minor mm )
```

Determinant of a 2x2 minor.

Parameters

<i>mm</i>	Minor (must be 2x2).
-----------	----------------------

Returns

$\det(mm)$.

Definition at line 1383 of file [Matrix2D.h](#).

References [Mat2D_Minor::cols](#), [MAT2D_MINOR_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D_Minor::rows](#).

Referenced by [mat2D_minor_det\(\)](#).

4.9.3.15 mat2D_dot()

```
void mat2D_dot (
    Mat2D dst,
    Mat2D a,
    Mat2D b )
```

Matrix product: $dst = a * b$.

Parameters

<i>dst</i>	Destination matrix (size a.rows x b.cols).
<i>a</i>	Left matrix (size a.rows x a.cols).
<i>b</i>	Right matrix (size a.cols x b.cols).

Precondition

a.cols == *b*.rows, *dst*.rows == *a*.rows, *dst*.cols == *b*.cols.

Postcondition

dst is overwritten.

Definition at line 424 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

Referenced by [adl_arrow_draw\(\)](#), [mat2D_set_DCM_zyx\(\)](#), and [mat2D_solve_linear_sys_LUP_decomposition\(\)](#).

4.9.3.16 mat2D_dot_product()

```
double mat2D_dot_product (
    Mat2D a,
    Mat2D b )
```

Dot product between two vectors.

Parameters

<i>a</i>	Vector (shape n x 1 or 1 x n).
<i>b</i>	Vector (same shape as <i>a</i>).

Returns

The scalar dot product sum.

Precondition

a.rows == *b*.rows, *a*.cols == *b*.cols, and one dimension equals 1.

Definition at line 450 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.17 mat2D_fill()

```
void mat2D_fill (
    Mat2D m,
    double x )
```

Fill all elements of a matrix of doubles with a scalar value.

Parameters

<i>m</i>	Matrix to fill.
<i>x</i>	Value to assign to every element.

Definition at line 362 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), and [Mat2D::rows](#).

Referenced by [adl_arrow_draw\(\)](#), [mat2D_invert\(\)](#), [mat2D_LUP_decomposition_with_swap\(\)](#), and [mat2D_solve_linear_sys_LUP_deco](#)

4.9.3.18 mat2D_fill_sequence()

```
void mat2D_fill_sequence (
    Mat2D m,
    double start,
    double step )
```

Fill a matrix with an arithmetic sequence laid out in row-major order.

Parameters

<i>m</i>	Matrix to fill.
<i>start</i>	First value in the sequence.
<i>step</i>	Increment between consecutive elements.

Element at linear index k gets value start + step * k.

Definition at line 378 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [mat2D_offset2d\(\)](#), and [Mat2D::rows](#).

4.9.3.19 mat2D_fill_uint32()

```
void mat2D_fill_uint32 (
    Mat2D_uint32 m,
    uint32_t x )
```

Fill all elements of a matrix of `uint32_t` with a scalar value.

Parameters

<i>m</i>	Matrix to fill.
<i>x</i>	Value to assign to every element.

Definition at line 391 of file [Matrix2D.h](#).

References [Mat2D_uint32::cols](#), [MAT2D_AT_UINT32](#), and [Mat2D_uint32::rows](#).

Referenced by [adl_2Dscalar_interp_on_figure\(\)](#), and [adl_curves_plot_on_figure\(\)](#).

4.9.3.20 mat2D_free()

```
void mat2D_free (
    Mat2D m )
```

Free the memory owned by a [Mat2D](#) (elements pointer).

Parameters

<i>m</i>	Matrix whose elements were allocated via MATRIX2D_MALLOC .
----------	--

Note

Safe to call with `m.elements == NULL`.

Definition at line 314 of file [Matrix2D.h](#).

References [Mat2D::elements](#).

Referenced by [adl_arrow_draw\(\)](#), [check_window_mat_size\(\)](#), [mat2D_det\(\)](#), [mat2D_invert\(\)](#), [mat2D_set_DCM_zyx\(\)](#), and [mat2D_solve_linear_sys_LUP_decomposition\(\)](#).

4.9.3.21 mat2D_free_uint32()

```
void mat2D_free_uint32 (
    Mat2D_uint32 m )
```

Free the memory owned by a [Mat2D_uint32](#) (elements pointer).

Parameters

<i>m</i>	Matrix whose elements were allocated via MATRIX2D_MALLOC .
----------	--

Note

Safe to call with `m.elements == NULL`.

Definition at line 324 of file [Matrix2D.h](#).

References [Mat2D::uint32::elements](#).

Referenced by [check_window_mat_size\(\)](#), and [destroy_window\(\)](#).

4.9.3.22 mat2D_get_col()

```
void mat2D_get_col (
    Mat2D des,
    size_t des_col,
    Mat2D src,
    size_t src_col )
```

Copy a column from src into a column of des.

Parameters

<i>des</i>	Destination matrix (same row count as src).
<i>des_col</i>	Column index in destination.
<i>src</i>	Source matrix.
<i>src_col</i>	Column index in source.

Definition at line 810 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.23 mat2D_get_row()

```
void mat2D_get_row (
    Mat2D des,
    size_t des_row,
    Mat2D src,
    size_t src_row )
```

Copy a row from src into a row of des.

Parameters

<i>des</i>	Destination matrix (same number of columns as src).
<i>des_row</i>	Row index in destination.
<i>src</i>	Source matrix.
<i>src_row</i>	Row index in source.

Definition at line 879 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.24 mat2D_invert()

```
void mat2D_invert (
    Mat2D des,
    Mat2D src )
```

Invert a square matrix using Gauss-Jordan elimination.

Parameters

<i>des</i>	Destination matrix (same shape as <i>src</i>).
<i>src</i>	Source square matrix.

Precondition

src is square and nonsingular.

If $\det(\text{src}) == 0$, prints an error and sets *des* to all zeros.

Warning

May be numerically unstable for ill-conditioned matrices.

Definition at line 1169 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [mat2D_alloc\(\)](#), [MAT2D_AT](#), [mat2D_copy\(\)](#), [mat2D_det\(\)](#), [mat2D_fill\(\)](#), [mat2D_free\(\)](#), [mat2D_mult_row\(\)](#), [mat2D_set_identity\(\)](#), [mat2D_sub_row_time_factor_to_row\(\)](#), [mat2D_swap_rows\(\)](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

Referenced by [mat2D_solve_linear_sys_LUP_decomposition\(\)](#).

4.9.3.25 mat2D_LUP_decomposition_with_swap()

```
void mat2D_LUP_decomposition_with_swap (
    Mat2D src,
    Mat2D l,
    Mat2D p,
    Mat2D u )
```

Compute LUP decomposition: $P \cdot A = L \cdot U$ with *L* unit diagonal.

Parameters

<i>src</i>	Input matrix A (not modified).
<i>l</i>	Lower triangular matrix with unit diagonal (output).
<i>p</i>	Permutation matrix (output).
<i>u</i>	Upper triangular matrix (output).

Precondition

l, *p*, *u* are allocated to match *src* shape; *src* is square.

Definition at line 1107 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [mat2D_copy\(\)](#), [mat2D_fill\(\)](#), [mat2D_set_identity\(\)](#), [mat2D_sub_row_time_factor_to_row\(\)](#), [mat2D_swap_rows\(\)](#), and [Mat2D::rows](#).

Referenced by [mat2D_solve_linear_sys_LUP_decomposition\(\)](#).

4.9.3.26 mat2D_make_identity()

```
double mat2D_make_identity (
    Mat2D m )
```

Reduce a matrix to identity via Gauss-Jordan elimination and return the cumulative scaling factor.

Parameters

<i>m</i>	Matrix reduced in-place to identity (if nonsingular).
----------	---

Returns

The product of row scaling factors applied during elimination.

Note

Intended as a helper for determinant-related operations.

Warning

Not robust to singular or ill-conditioned matrices.

Definition at line 643 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [mat2D_mult_row\(\)](#), [mat2D_sub_row_time_factor_to_row\(\)](#), [mat2D_swap_rows\(\)](#), and [Mat2D::rows](#).

4.9.3.27 mat2D_mat_is_all_digit()

```
bool mat2D_mat_is_all_digit (
    Mat2D m,
    double digit )
```

Check if all elements of a matrix equal a given digit.

Parameters

<i>m</i>	Matrix.
<i>digit</i>	Value to compare.

Returns

true if every element equals digit, false otherwise.

Definition at line 949 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), and [Mat2D::rows](#).

4.9.3.28 mat2D_minor_alloc_fill_from_mat()

```
Mat2D_Minor mat2D_minor_alloc_fill_from_mat (
    Mat2D ref_mat,
    size_t i,
    size_t j )
```

Allocate a minor view by excluding row *i* and column *j* of *ref_mat*.

Parameters

<i>ref_mat</i>	Reference square matrix.
<i>i</i>	Excluded row index in <i>ref_mat</i> .
<i>j</i>	Excluded column index in <i>ref_mat</i> .

Returns

A [Mat2D_Minor](#) that references *ref_mat*.

Note

Free *rows_list* and *cols_list* with *mat2D_minor_free* when done.

Definition at line 1279 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [Mat2D_Minor::cols](#), [Mat2D_Minor::cols_list](#), [MATRIX2D_ASSERT](#), [MATRIX2D_MALLOC](#), [Mat2D_Minor::ref_mat](#), [Mat2D::rows](#), [Mat2D_Minor::rows](#), [Mat2D_Minor::rows_list](#), and [Mat2D_Minor::stride_r](#).

4.9.3.29 mat2D_minor_alloc_fill_from_mat_minor()

```
Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor (
    Mat2D_Minor ref_mm,
    size_t i,
    size_t j )
```

Allocate a nested minor view from an existing minor by excluding row *i* and column *j* of the minor.

Parameters

<i>ref_mm</i>	Reference minor.
<i>i</i>	Excluded row index in the minor.
<i>j</i>	Excluded column index in the minor.

Returns

A new [Mat2D_Minor](#) that references the same underlying matrix.

Note

Free rows_list and cols_list with [mat2D_minor_free](#) when done.

Definition at line [1318](#) of file [Matrix2D.h](#).

References [Mat2D_Minor::cols](#), [Mat2D_Minor::cols_list](#), [MATRIX2D_ASSERT](#), [MATRIX2D_MALLOC](#), [Mat2D_Minor::ref_mat](#), [Mat2D_Minor::rows](#), [Mat2D_Minor::rows_list](#), and [Mat2D_Minor::stride_r](#).

Referenced by [mat2D_minor_det\(\)](#).

4.9.3.30 mat2D_minor_det()

```
double mat2D_minor_det (
    Mat2D_Minor mm )
```

Determinant of a minor via recursive expansion by minors.

Parameters

<i>mm</i>	Square minor.
-----------	---------------

Returns

$\det(\text{mm})$.

Warning

Exponential complexity (factorial). Intended for educational or very small matrices only.

Definition at line 1396 of file [Matrix2D.h](#).

References [Mat2D_Minor::cols](#), [mat2D_det_2x2_mat_minor\(\)](#), [mat2D_minor_alloc_fill_from_mat_minor\(\)](#), [MAT2D_MINOR_AT](#), [mat2D_minor_free\(\)](#), [MATRIX2D_ASSERT](#), and [Mat2D_Minor::rows](#).

4.9.3.31 mat2D_minor_free()

```
void mat2D_minor_free (
    Mat2D_Minor mm )
```

Free the index arrays owned by a minor.

Parameters

<i>mm</i>	Minor to free.
-----------	----------------

Note

After this call, *mm.rows_list* and *mm.cols_list* are invalid.

Definition at line 1353 of file [Matrix2D.h](#).

References [Mat2D_Minor::cols_list](#), and [Mat2D_Minor::rows_list](#).

Referenced by [mat2D_minor_det\(\)](#).

4.9.3.32 mat2D_minor_print()

```
void mat2D_minor_print (
    Mat2D_Minor mm,
    const char * name,
    size_t padding )
```

Print a minor matrix to stdout with a name and indentation padding.

Parameters

<i>mm</i>	Minor to print.
<i>name</i>	Label to print.
<i>padding</i>	Left padding in spaces.

Definition at line 1365 of file [Matrix2D.h](#).

References [Mat2D_Minor::cols](#), [MAT2D_MINOR_AT](#), and [Mat2D_Minor::rows](#).

4.9.3.33 mat2D_mult()

```
void mat2D_mult (
    Mat2D m,
    double factor )
```

In-place scalar multiplication: $m *= \text{factor}$.

Parameters

<i>m</i>	Matrix.
<i>factor</i>	Scalar multiplier.

Definition at line 557 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), and [Mat2D::rows](#).

4.9.3.34 mat2D_mult_row()

```
void mat2D_mult_row (
    Mat2D m,
    size_t r,
    double factor )
```

In-place row scaling: $\text{row}(r) *= \text{factor}$.

Parameters

<i>m</i>	Matrix.
<i>r</i>	Row index.
<i>factor</i>	Scalar multiplier.

Definition at line 572 of file [Matrix2D.h](#).

References [Mat2D::cols](#), and [MAT2D_AT](#).

Referenced by [mat2D_invert\(\)](#), and [mat2D_make_identity\(\)](#).

4.9.3.35 mat2D_offset2d()

```
size_t mat2D_offset2d (
    Mat2D m,
```

```
size_t i,  
size_t j )
```

Compute the linear offset of element (i, j) in a [Mat2D](#).

Parameters

<i>m</i>	Matrix.
<i>i</i>	Row index (0-based).
<i>j</i>	Column index (0-based).

Returns

The linear offset $i * \text{stride_r} + j$.

Precondition

$0 \leq i < \text{rows}$, $0 \leq j < \text{cols}$ (asserted).

Definition at line 337 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MATRIX2D_ASSERT](#), [Mat2D::rows](#), and [Mat2D::stride_r](#).

Referenced by [mat2D_fill_sequence\(\)](#).

4.9.3.36 mat2D_offset2d_uint32()

```
size_t mat2D_offset2d_uint32 (
    Mat2D_uint32 m,
    size_t i,
    size_t j )
```

Compute the linear offset of element (*i*, *j*) in a [Mat2D_uint32](#).

Parameters

<i>m</i>	Matrix.
<i>i</i>	Row index (0-based).
<i>j</i>	Column index (0-based).

Returns

The linear offset $i * \text{stride_r} + j$.

Precondition

$0 \leq i < \text{rows}$, $0 \leq j < \text{cols}$ (asserted).

Definition at line 351 of file [Matrix2D.h](#).

References [Mat2D_uint32::cols](#), [MATRIX2D_ASSERT](#), [Mat2D_uint32::rows](#), and [Mat2D_uint32::stride_r](#).

4.9.3.37 mat2D_print()

```
void mat2D_print (
    Mat2D m,
    const char * name,
    size_t padding )
```

Print a matrix to stdout with a name and indentation padding.

Parameters

<i>m</i>	Matrix to print.
<i>name</i>	Label to print.
<i>padding</i>	Left padding in spaces.

Definition at line 585 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), and [Mat2D::rows](#).

4.9.3.38 mat2D_print_as_col()

```
void mat2D_print_as_col (
    Mat2D m,
    const char * name,
    size_t padding )
```

Print a matrix as a flattened column vector to stdout.

Parameters

<i>m</i>	Matrix to print (flattened in row-major).
<i>name</i>	Label to print.
<i>padding</i>	Left padding in spaces.

Definition at line 604 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [Mat2D::elements](#), and [Mat2D::rows](#).

4.9.3.39 mat2D_rand()

```
void mat2D_rand (
    Mat2D m,
    double low,
    double high )
```

Fill a matrix with random doubles in [low, high).

Parameters

<i>m</i>	Matrix to fill.
<i>low</i>	Lower bound (inclusive).
<i>high</i>	Upper bound (exclusive).

Precondition

high > *low*.

Definition at line 407 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [mat2D_rand_double\(\)](#), and [Mat2D::rows](#).

4.9.3.40 mat2D_rand_double()

```
double mat2D_rand_double (
    void )
```

Return a pseudo-random double in the range [0, 1].

Note

Uses C library rand() and RAND_MAX. Not cryptographically secure.

Definition at line 266 of file [Matrix2D.h](#).

Referenced by [mat2D_rand\(\)](#).

4.9.3.41 mat2D_row_is_all_digit()

```
bool mat2D_row_is_all_digit (
    Mat2D m,
    double digit,
    size_t r )
```

Check if all elements of a row equal a given digit.

Parameters

<i>m</i>	Matrix.
<i>digit</i>	Value to compare.
<i>r</i>	Row index.

Returns

true if every element equals digit, false otherwise.

Definition at line 968 of file [Matrix2D.h](#).

References [Mat2D::cols](#), and [MAT2D_AT](#).

Referenced by [mat2D_det\(\)](#).

4.9.3.42 mat2D_set_DCM_zyx()

```
void mat2D_set_DCM_zyx (
    Mat2D DCM,
    float yaw_deg,
    float pitch_deg,
    float roll_deg )
```

Build a 3x3 direction cosine matrix (DCM) from Z-Y-X Euler angles.

Parameters

<i>DCM</i>	3x3 destination matrix.
<i>yaw_deg</i>	Rotation about Z in degrees.
<i>pitch_deg</i>	Rotation about Y in degrees.
<i>roll_deg</i>	Rotation about X in degrees.

Computes $DCM = R_x(\text{roll}) * R_y(\text{pitch}) * R_z(\text{yaw})$.

Definition at line 743 of file [Matrix2D.h](#).

References [mat2D_alloc\(\)](#), [mat2D_dot\(\)](#), [mat2D_free\(\)](#), [mat2D_set_rot_mat_x\(\)](#), [mat2D_set_rot_mat_y\(\)](#), and [mat2D_set_rot_mat_z\(\)](#).

4.9.3.43 mat2D_set_identity()

```
void mat2D_set_identity (
    Mat2D m )
```

Set a square matrix to the identity matrix.

Parameters

<i>m</i>	Matrix (must be square).
----------	--------------------------

Precondition

```
m.rows == m.cols.
```

Definition at line 619 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

Referenced by [mat2D_invert\(\)](#), [mat2D_LUP_decomposition_with_swap\(\)](#), [mat2D_set_rot_mat_x\(\)](#), [mat2D_set_rot_mat_y\(\)](#), and [mat2D_set_rot_mat_z\(\)](#).

4.9.3.44 mat2D_set_rot_mat_x()

```
void mat2D_set_rot_mat_x (
    Mat2D m,
    float angle_deg )
```

Set a 3x3 rotation matrix for rotation about the X-axis.

Parameters

<i>m</i>	3x3 destination matrix.
<i>angle_deg</i>	Angle in degrees.

Definition at line 689 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [mat2D_set_identity\(\)](#), [MATRIX2D_ASSERT](#), [PI](#), and [Mat2D::rows](#).

Referenced by [mat2D_set_DCM_zyx\(\)](#).

4.9.3.45 mat2D_set_rot_mat_y()

```
void mat2D_set_rot_mat_y (
    Mat2D m,
    float angle_deg )
```

Set a 3x3 rotation matrix for rotation about the Y-axis.

Parameters

<i>m</i>	3x3 destination matrix.
<i>angle_deg</i>	Angle in degrees.

Definition at line 706 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [mat2D_set_identity\(\)](#), [MATRIX2D_ASSERT](#), [PI](#), and [Mat2D::rows](#).

Referenced by [mat2D_set_DCM_zyx\(\)](#).

4.9.3.46 mat2D_set_rot_mat_z()

```
void mat2D_set_rot_mat_z (
    Mat2D m,
    float angle_deg )
```

Set a 3x3 rotation matrix for rotation about the Z-axis.

Parameters

<i>m</i>	3x3 destination matrix.
<i>angle_deg</i>	Angle in degrees.

Definition at line 723 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [mat2D_set_identity\(\)](#), [MATRIX2D_ASSERT](#), [PI](#), and [Mat2D::rows](#).

Referenced by [adl_arrow_draw\(\)](#), and [mat2D_set_DCM_zyx\(\)](#).

4.9.3.47 mat2D_solve_linear_sys_LUP_decomposition()

```
void mat2D_solve_linear_sys_LUP_decomposition (
    Mat2D A,
    Mat2D x,
    Mat2D B )
```

Solve the linear system $A x = B$ using LUP decomposition.

Parameters

<i>A</i>	Coefficient matrix (NxN).
<i>x</i>	Solution vector ($N \times 1$) (output).
<i>B</i>	Right-hand side vector ($N \times 1$).

Internally computes LUP and uses explicit inverses of L and U.

Warning

Forming inverses explicitly can be less stable; a forward/backward substitution would be preferable for production-quality code.

Definition at line 1236 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [mat2D_alloc\(\)](#), [mat2D_dot\(\)](#), [mat2D_fill\(\)](#), [mat2D_free\(\)](#), [mat2D_invert\(\)](#), [mat2D_LUP_decomposition_with_s](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.48 mat2D_sub()

```
void mat2D_sub (
    Mat2D dst,
    Mat2D a )
```

In-place subtraction: $\text{dst} -= \text{a}$.

Parameters

<i>dst</i>	Destination matrix to be decremented.
<i>a</i>	Subtrahend of same shape as <i>dst</i> .

Precondition

Shapes match.

Definition at line 527 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

Referenced by [adl_arrow_draw\(\)](#).

4.9.3.49 mat2D_sub_col_to_col()

```
void mat2D_sub_col_to_col (
    Mat2D des,
    size_t des_col,
    Mat2D src,
    size_t src_col )
```

Subtract a source column from a destination column: $\text{des}[:, \text{des_col}] -= \text{src}[:, \text{src_col}]$.

Parameters

<i>des</i>	Destination matrix (same row count as <i>src</i>).
<i>des_col</i>	Column index in destination.
<i>src</i>	Source matrix.
<i>src_col</i>	Column index in source.

Definition at line 846 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.50 mat2D_sub_row_time_factor_to_row()

```
void mat2D_sub_row_time_factor_to_row (
    Mat2D m,
```

```
size_t des_r,
size_t src_r,
double factor )
```

Row operation: `row(des_r) -= factor * row(src_r)`.

Parameters

<i>m</i>	Matrix.
<i>des_r</i>	Destination row index.
<i>src_r</i>	Source row index.
<i>factor</i>	Scalar multiplier.

Definition at line 545 of file [Matrix2D.h](#).

References [Mat2D::cols](#), and [MAT2D_AT](#).

Referenced by [mat2D_invert\(\)](#), [mat2D_LUP_decomposition_with_swap\(\)](#), [mat2D_make_identity\(\)](#), and [mat2D_triangulate\(\)](#).

4.9.3.51 mat2D_sub_row_to_row()

```
void mat2D_sub_row_to_row (
    Mat2D des,
    size_t des_row,
    Mat2D src,
    size_t src_row )
```

Subtract a source row from a destination row: `des[des_row, :] -= src[src_row, :]`.

Parameters

<i>des</i>	Destination matrix (same number of columns as <i>src</i>).
<i>des_{row}</i>	Row index in destination.
<i>src</i>	Source matrix.
<i>src_{row}</i>	Row index in source.

Definition at line 915 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.52 mat2D_swap_rows()

```
void mat2D_swap_rows (
    Mat2D m,
    size_t r1,
    size_t r2 )
```

Swap two rows of a matrix in-place.

Parameters

<i>m</i>	Matrix.
<i>r1</i>	First row index.
<i>r2</i>	Second row index.

Definition at line 863 of file [Matrix2D.h](#).

References [Mat2D::cols](#), and [MAT2D_AT](#).

Referenced by [mat2D_invert\(\)](#), [mat2D_LUP_decomposition_with_swap\(\)](#), [mat2D_make_identity\(\)](#), and [mat2D_triangulate\(\)](#).

4.9.3.53 mat2D_transpose()

```
void mat2D_transpose (
    Mat2D des,
    Mat2D src )
```

Transpose a matrix: $\text{des} = \text{src}^T$.

Parameters

<i>des</i>	Destination matrix (shape $\text{src}.cols \times \text{src}.rows$).
<i>src</i>	Source matrix.

Definition at line 1149 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [MATRIX2D_ASSERT](#), and [Mat2D::rows](#).

4.9.3.54 mat2D_triangulate()

```
double mat2D_triangulate (
    Mat2D m )
```

Forward elimination to transform a matrix to upper triangular form.

Parameters

<i>m</i>	Matrix transformed in-place.
----------	------------------------------

Returns

Product of row scaling factors (currently 1 in this implementation).

Note

Used as part of determinant computation via triangularization.

Warning

Not robust for linearly dependent rows or tiny pivots.

Definition at line 1013 of file [Matrix2D.h](#).

References [Mat2D::cols](#), [MAT2D_AT](#), [mat2D_sub_row_time_factor_to_row\(\)](#), [mat2D_swap_rows\(\)](#), and [Mat2D::rows](#).

Referenced by [mat2D_det\(\)](#).

4.10 Matrix2D.h

```

00001
00039 #ifndef MATRIX2D_H_
00040 #define MATRIX2D_H_
00041
00042 #include <stddef.h>
00043 #include <stdio.h>
00044 #include <stdlib.h>
00045 #include <stdint.h>
00046 #include <stdbool.h>
00047
00055 #ifndef MATRIX2D_MALLOC
00056 #define MATRIX2D_MALLOC malloc
00057 #endif //MATRIX2D_MALLOC
00058
00066 #ifndef MATRIX2D_ASSERT
00067 #include <assert.h>
00068 #define MATRIX2D_ASSERT assert
00069 #endif //MATRIX2D_ASSERT
00070
00081 typedef struct {
00082     size_t rows;
00083     size_t cols;
00084     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00085     double *elements;
00086 } Mat2D;
00087
00098 typedef struct {
00099     size_t rows;
00100     size_t cols;
00101     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00102     uint32_t *elements;
00103 } Mat2D_uint32;
00104
00119 typedef struct {
00120     size_t rows;
00121     size_t cols;
00122     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00123     size_t *rows_list;
00124     size_t *cols_list;
00125     Mat2D ref_mat;
00126 } Mat2D_Minor;
00127
00127 #if 0
00142 #define MAT2D_AT(m, i, j) (m).elements[mat2D_offset2d((m), (i), (j))]
00143 #define MAT2D_AT_UINT32(m, i, j) (m).elements[mat2D_offset2d_uint32((m), (i), (j))]
00144 #else /* use this macro for batter performance but no assertion */
00145 #define MAT2D_AT(m, i, j) (m).elements[i * m.stride_r + j]
00146 #define MAT2D_AT_UINT32(m, i, j) (m).elements[i * m.stride_r + j]
00147 #endif
00148
00149 #ifndef PI
00150     #ifndef __USE_MISC
00151         #define __USE_MISC
00152     #endif
00153     #include <math.h>
00154     #define PI M_PI
00155 #endif
00156

```

```

00162 #define MAT2D_MINOR_AT(mm, i, j) MAT2D_AT(mm.ref_mat, mm.rows_list[i], mm.cols_list[j])
00167 #define MAT2D_PRINT(m) mat2D_print(m, #m, 0)
00172 #define MAT2D_PRINT_AS_COL(m) mat2D_print_as_col(m, #m, 0)
00177 #define MAT2D_MINOR_PRINT(mm) mat2D_minor_print(mm, #mm, 0)
00184 #define mat2D_normalize(m) mat2D_mult((m), 1.0 / mat2D_calc_norma((m)))
00185
00186 double mat2D_rand_double(void);
00187
00188 Mat2D mat2D_alloc(size_t rows, size_t cols);
00189 Mat2D_uint32 mat2D_alloc_uint32(size_t rows, size_t cols);
00190 void mat2D_free(Mat2D m);
00191 void mat2D_free_uint32(Mat2D_uint32 m);
00192 size_t mat2D_offset2d(Mat2D m, size_t i, size_t j);
00193 size_t mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j);
00194
00195 void mat2D_fill(Mat2D m, double x);
00196 void mat2D_fill_sequence(Mat2D m, double start, double step);
00197 void mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x);
00198 void mat2D_rand(Mat2D m, double low, double high);
00199
00200 void mat2D_dot(Mat2D dst, Mat2D a, Mat2D b);
00201 double mat2D_dot_product(Mat2D a, Mat2D b);
00202 void mat2D_cross(Mat2D dst, Mat2D a, Mat2D b);
00203
00204 void mat2D_add(Mat2D dst, Mat2D a);
00205 void mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00206
00207 void mat2D_sub(Mat2D dst, Mat2D a);
00208 void mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00209
00210 void mat2D_mult(Mat2D m, double factor);
00211 void mat2D_mult_row(Mat2D m, size_t r, double factor);
00212
00213 void mat2D_print(Mat2D m, const char *name, size_t padding);
00214 void mat2D_print_as_col(Mat2D m, const char *name, size_t padding);
00215
00216 void mat2D_set_identity(Mat2D m);
00217 double mat2D_make_identity(Mat2D m);
00218 void mat2D_set_rot_mat_x(Mat2D m, float angle_deg);
00219 void mat2D_set_rot_mat_y(Mat2D m, float angle_deg);
00220 void mat2D_set_rot_mat_z(Mat2D m, float angle_deg);
00221 void mat2D_set_DCM_zyx(Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg);
00222
00223 void mat2D_copy(Mat2D des, Mat2D src);
00224 void mat2D_copy_mat_to_mat_at_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je);
00225
00226 void mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00227 void mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00228 void mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00229
00230 void mat2D_swap_rows(Mat2D m, size_t r1, size_t r2);
00231 void mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00232 void mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00233 void mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00234
00235 double mat2D_calc_norma(Mat2D m);
00236
00237 bool mat2D_mat_is_all_digit(Mat2D m, double digit);
00238 bool mat2D_row_is_all_digit(Mat2D m, double digit, size_t r);
00239 bool mat2D_col_is_all_digit(Mat2D m, double digit, size_t c);
00240
00241 double mat2D_det_2x2_mat(Mat2D m);
00242 double mat2D_triangulate(Mat2D m);
00243 double mat2D_det(Mat2D m);
00244 void mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u);
00245 void mat2D_transpose(Mat2D des, Mat2D src);
00246 void mat2D_invert(Mat2D des, Mat2D src);
00247 void mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B);
00248
00249 Mat2D_Minor mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j);
00250 Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j);
00251 void mat2D_minor_free(Mat2D_Minor mm);
00252 void mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding);
00253 double mat2D_det_2x2_mat_minor(Mat2D_Minor mm);
00254 double mat2D_minor_det(Mat2D_Minor mm);
00255
00256 #endif // MATRIX2D_H_
00257
00258 #ifdef MATRIX2D_IMPLEMENTATION
00259 #undef MATRIX2D_IMPLEMENTATION
00260
00261
00262 double mat2D_rand_double(void)
00263 {
00264     return (double) rand() / (double) RAND_MAX;
00265 }

```

```

00270
00278 Mat2D mat2D_alloc(size_t rows, size_t cols)
00279 {
00280     Mat2D m;
00281     m.rows = rows;
00282     m.cols = cols;
00283     m.stride_r = cols;
00284     m.elements = (double*)MATRIX2D_MALLOC(sizeof(double)*rows*cols);
00285     MATRIX2D_ASSERT(m.elements != NULL);
00286
00287     return m;
00288 }
00289
00297 Mat2D_uint32 mat2D_alloc_uint32(size_t rows, size_t cols)
00298 {
00299     Mat2D_uint32 m;
00300     m.rows = rows;
00301     m.cols = cols;
00302     m.stride_r = cols;
00303     m.elements = (uint32_t*)MATRIX2D_MALLOC(sizeof(uint32_t)*rows*cols);
00304     MATRIX2D_ASSERT(m.elements != NULL);
00305
00306     return m;
00307 }
00308
00314 void mat2D_free(Mat2D m)
00315 {
00316     free(m.elements);
00317 }
00318
00324 void mat2D_free_uint32(Mat2D_uint32 m)
00325 {
00326     free(m.elements);
00327 }
00328
00337 size_t mat2D_offset2d(Mat2D m, size_t i, size_t j)
00338 {
00339     MATRIX2D_ASSERT(i < m.rows && j < m.cols);
00340     return i * m.stride_r + j;
00341 }
00342
00351 size_t mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j)
00352 {
00353     MATRIX2D_ASSERT(i < m.rows && j < m.cols);
00354     return i * m.stride_r + j;
00355 }
00356
00362 void mat2D_fill(Mat2D m, double x)
00363 {
00364     for (size_t i = 0; i < m.rows; ++i) {
00365         for (size_t j = 0; j < m.cols; ++j) {
00366             MAT2D_AT(m, i, j) = x;
00367         }
00368     }
00369 }
00370
00378 void mat2D_fill_sequence(Mat2D m, double start, double step) {
00379     for (size_t i = 0; i < m.rows; i++) {
00380         for (size_t j = 0; j < m.cols; j++) {
00381             MAT2D_AT(m, i, j) = start + step * mat2D_offset2d(m, i, j);
00382         }
00383     }
00384 }
00385
00391 void mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x)
00392 {
00393     for (size_t i = 0; i < m.rows; ++i) {
00394         for (size_t j = 0; j < m.cols; ++j) {
00395             MAT2D_AT_UINT32(m, i, j) = x;
00396         }
00397     }
00398 }
00399
00407 void mat2D_rand(Mat2D m, double low, double high)
00408 {
00409     for (size_t i = 0; i < m.rows; ++i) {
00410         for (size_t j = 0; j < m.cols; ++j) {
00411             MAT2D_AT(m, i, j) = mat2D_rand_double()*(high - low) + low;
00412         }
00413     }
00414 }
00415
00424 void mat2D_dot(Mat2D dst, Mat2D a, Mat2D b)
00425 {
00426     MATRIX2D_ASSERT(a.cols == b.rows);
00427     MATRIX2D_ASSERT(a.rows == dst.rows);
00428     MATRIX2D_ASSERT(b.cols == dst.cols);

```

```

00429
00430     size_t i, j, k;
00431
00432     for (i = 0; i < dst.rows; i++) {
00433         for (j = 0; j < dst.cols; j++) {
00434             MAT2D_AT(dst, i, j) = 0;
00435             for (k = 0; k < a.cols; k++) {
00436                 MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, k)*MAT2D_AT(b, k, j);
00437             }
00438         }
00439     }
00440 }
00441
00450 double mat2D_dot_product(Mat2D a, Mat2D b)
00451 {
00452     MATRIX2D_ASSERT(a.rows == b.rows);
00453     MATRIX2D_ASSERT(a.cols == b.cols);
00454     MATRIX2D_ASSERT((1 == a.cols && 1 == b.cols) || (1 == a.rows && 1 == b.rows));
00455
00456     double dot_product = 0;
00457
00458     if (1 == a.cols) {
00459         for (size_t i = 0; i < a.rows; i++) {
00460             dot_product += MAT2D_AT(a, i, 0) * MAT2D_AT(b, i, 0);
00461         }
00462     } else {
00463         for (size_t j = 0; j < a.cols; j++) {
00464             dot_product += MAT2D_AT(a, 0, j) * MAT2D_AT(b, 0, j);
00465         }
00466     }
00467
00468     return dot_product;
00469 }
00471
00479 void mat2D_cross(Mat2D dst, Mat2D a, Mat2D b)
00480 {
00481     MATRIX2D_ASSERT(3 == dst.rows && 1 == dst.cols);
00482     MATRIX2D_ASSERT(3 == a.rows && 1 == a.cols);
00483     MATRIX2D_ASSERT(3 == b.rows && 1 == b.cols);
00484
00485     MAT2D_AT(dst, 0, 0) = MAT2D_AT(a, 1, 0) * MAT2D_AT(b, 2, 0) - MAT2D_AT(a, 2, 0) * MAT2D_AT(b, 1, 0);
00486     MAT2D_AT(dst, 1, 0) = MAT2D_AT(a, 2, 0) * MAT2D_AT(b, 0, 0) - MAT2D_AT(a, 0, 0) * MAT2D_AT(b, 2, 0);
00487     MAT2D_AT(dst, 2, 0) = MAT2D_AT(a, 0, 0) * MAT2D_AT(b, 1, 0) - MAT2D_AT(a, 1, 0) * MAT2D_AT(b, 0, 0);
00488 }
00489
00496 void mat2D_add(Mat2D dst, Mat2D a)
00497 {
00498     MATRIX2D_ASSERT(dst.rows == a.rows);
00499     MATRIX2D_ASSERT(dst.cols == a.cols);
00500     for (size_t i = 0; i < dst.rows; ++i) {
00501         for (size_t j = 0; j < dst.cols; ++j) {
00502             MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, j);
00503         }
00504     }
00505 }
00506
00514 void mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
00515 {
00516     for (size_t j = 0; j < m.cols; ++j) {
00517         MAT2D_AT(m, des_r, j) += factor * MAT2D_AT(m, src_r, j);
00518     }
00519 }
00520
00527 void mat2D_sub(Mat2D dst, Mat2D a)
00528 {
00529     MATRIX2D_ASSERT(dst.rows == a.rows);
00530     MATRIX2D_ASSERT(dst.cols == a.cols);
00531     for (size_t i = 0; i < dst.rows; ++i) {
00532         for (size_t j = 0; j < dst.cols; ++j) {
00533             MAT2D_AT(dst, i, j) -= MAT2D_AT(a, i, j);
00534         }
00535     }
00536 }
00537
00545 void mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
00546 {
00547     for (size_t j = 0; j < m.cols; ++j) {
00548         MAT2D_AT(m, des_r, j) -= factor * MAT2D_AT(m, src_r, j);
00549     }
00550 }
00551
00557 void mat2D_mult(Mat2D m, double factor)

```

```

00558 {
00559     for (size_t i = 0; i < m.rows; ++i) {
00560         for (size_t j = 0; j < m.cols; ++j) {
00561             MAT2D_AT(m, i, j) *= factor;
00562         }
00563     }
00564 }
00565
00572 void mat2D_mult_row(Mat2D m, size_t r, double factor)
00573 {
00574     for (size_t j = 0; j < m.cols; ++j) {
00575         MAT2D_AT(m, r, j) *= factor;
00576     }
00577 }
00578
00585 void mat2D_print(Mat2D m, const char *name, size_t padding)
00586 {
00587     printf("%*s% = [\n", (int) padding, "", name);
00588     for (size_t i = 0; i < m.rows; ++i) {
00589         printf("%*s ", (int) padding, "");
00590         for (size_t j = 0; j < m.cols; ++j) {
00591             printf("%9.6f ", MAT2D_AT(m, i, j));
00592         }
00593         printf("\n");
00594     }
00595     printf("%*s]\n", (int) padding, "");
00596 }
00597
00604 void mat2D_print_as_col(Mat2D m, const char *name, size_t padding)
00605 {
00606     printf("%*s% = [\n", (int) padding, "", name);
00607     for (size_t i = 0; i < m.rows*m.cols; ++i) {
00608         printf("%*s ", (int) padding, "");
00609         printf("%f\n", m.elements[i]);
00610     }
00611     printf("%*s]\n", (int) padding, "");
00612 }
00613
00619 void mat2D_set_identity(Mat2D m)
00620 {
00621     MATRIX2D_ASSERT(m.cols == m.rows);
00622     for (size_t i = 0; i < m.rows; ++i) {
00623         for (size_t j = 0; j < m.cols; ++j) {
00624             MAT2D_AT(m, i, j) = i == j ? 1 : 0;
00625             // if (i == j) {
00626             //     MAT2D_AT(m, i, j) = 1;
00627             // }
00628             // else {
00629             //     MAT2D_AT(m, i, j) = 0;
00630             // }
00631         }
00632     }
00633 }
00634
00643 double mat2D_make_identity(Mat2D m)
00644 {
00645     /* make identity matrix using Gauss elimination */
00646     /* performing Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
00647     /* returns the factor multiplying the determinant */
00648
00649     double factor_to_return = 1;
00650
00651     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
00652         /* check if it is the biggest first number (absolute value) */
00653         size_t biggest_r = i;
00654         for (size_t index = i; index < m.rows; index++) {
00655             if (fabs(MAT2D_AT(m, index, index)) > fabs(MAT2D_AT(m, biggest_r, 0))) {
00656                 biggest_r = index;
00657             }
00658         }
00659         if (i != biggest_r) {
00660             mat2D_swap_rows(m, i, biggest_r);
00661             factor_to_return *= -1;
00662         }
00663         for (size_t j = i+1; j < m.cols; j++) {
00664             double factor = 1 / MAT2D_AT(m, i, i);
00665             mat2D_sub_row_time_factor_to_row(m, j, i, MAT2D_AT(m, j, i) * factor);
00666             mat2D_mult_row(m, i, factor);
00667             factor_to_return *= factor;
00668         }
00669     }
00670     double factor = 1 / MAT2D_AT(m, m.rows-1, m.cols-1);
00671     mat2D_mult_row(m, m.rows-1, factor);
00672     factor_to_return *= factor;
00673     for (size_t c = m.cols-1; c > 0; c--) {
00674         for (int r = c-1; r >= 0; r--) {
00675             double factor = 1 / MAT2D_AT(m, c, c);
00676         }
00677     }
}

```

```

00676         mat2D_sub_row_time_factor_to_row(m, r, c, MAT2D_AT(m, r, c) * factor);
00677     }
00678 }
00679
00680
00681     return factor_to_return;
00682 }
00683
00684 void mat2D_set_rot_mat_x(Mat2D m, float angle_deg)
00685 {
00686     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00687
00688     float angle_rad = angle_deg * PI / 180;
00689     mat2D_set_identity(m);
00690     MAT2D_AT(m, 1, 1) = cos(angle_rad);
00691     MAT2D_AT(m, 1, 2) = sin(angle_rad);
00692     MAT2D_AT(m, 2, 1) = -sin(angle_rad);
00693     MAT2D_AT(m, 2, 2) = cos(angle_rad);
00694 }
00695
00696 void mat2D_set_rot_mat_y(Mat2D m, float angle_deg)
00697 {
00698     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00699
00700     float angle_rad = angle_deg * PI / 180;
00701     mat2D_set_identity(m);
00702     MAT2D_AT(m, 0, 0) = cos(angle_rad);
00703     MAT2D_AT(m, 0, 2) = -sin(angle_rad);
00704     MAT2D_AT(m, 2, 0) = sin(angle_rad);
00705     MAT2D_AT(m, 2, 2) = cos(angle_rad);
00706 }
00707
00708 void mat2D_set_rot_mat_z(Mat2D m, float angle_deg)
00709 {
00710     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00711
00712     float angle_rad = angle_deg * PI / 180;
00713     mat2D_set_identity(m);
00714     MAT2D_AT(m, 0, 0) = cos(angle_rad);
00715     MAT2D_AT(m, 0, 1) = sin(angle_rad);
00716     MAT2D_AT(m, 1, 0) = -sin(angle_rad);
00717     MAT2D_AT(m, 1, 1) = cos(angle_rad);
00718 }
00719
00720 void mat2D_set_DCM_zyx(Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg)
00721 {
00722     Mat2D RotZ = mat2D_alloc(3,3);
00723     mat2D_set_rot_mat_z(RotZ, yaw_deg);
00724     Mat2D RotY = mat2D_alloc(3,3);
00725     mat2D_set_rot_mat_y(RotY, pitch_deg);
00726     Mat2D RotX = mat2D_alloc(3,3);
00727     mat2D_set_rot_mat_x(RotX, roll_deg);
00728     Mat2D temp = mat2D_alloc(3,3);
00729
00730     mat2D_dot(temp, RotY, RotZ);
00731     mat2D_dot(DCM, RotX, temp); /* I have a DCM */
00732
00733 }
00734
00735 mat2D_free(RotZ);
00736 mat2D_free(RotY);
00737 mat2D_free(RotX);
00738 mat2D_free(temp);
00739
00740 }
00741
00742 void mat2D_copy(Mat2D des, Mat2D src)
00743 {
00744     MATRIX2D_ASSERT(des.cols == src.cols);
00745     MATRIX2D_ASSERT(des.rows == src.rows);
00746
00747     for (size_t i = 0; i < des.rows; ++i) {
00748         for (size_t j = 0; j < des.cols; ++j) {
00749             MAT2D_AT(des, i, j) = MAT2D_AT(src, i, j);
00750         }
00751     }
00752 }
00753
00754 void mat2D_copy_mat_to_mat_at_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)
00755 {
00756     MATRIX2D_ASSERT(je > js && ie > is);
00757     MATRIX2D_ASSERT(je-js+1 == des.cols);
00758     MATRIX2D_ASSERT(ie-is+1 == des.rows);
00759
00760     for (size_t index = 0; index < des.rows; ++index) {
00761         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
00762             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, is+index, js+jndex);
00763         }
00764     }
00765 }
```

```
00802
00810 void mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00811 {
00812     MATRIX2D_ASSERT(src_col < src.cols);
00813     MATRIX2D_ASSERT(des.rows == src.rows);
00814     MATRIX2D_ASSERT(des_col < des.cols);
00815
00816     for (size_t i = 0; i < des.rows; i++) {
00817         MAT2D_AT(des, i, des_col) = MAT2D_AT(src, i, src_col);
00818     }
00819 }
00820
00828 void mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00829 {
00830     MATRIX2D_ASSERT(src_col < src.cols);
00831     MATRIX2D_ASSERT(des.rows == src.rows);
00832     MATRIX2D_ASSERT(des_col < des.cols);
00833
00834     for (size_t i = 0; i < des.rows; i++) {
00835         MAT2D_AT(des, i, des_col) += MAT2D_AT(src, i, src_col);
00836     }
00837 }
00838
00846 void mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00847 {
00848     MATRIX2D_ASSERT(src_col < src.cols);
00849     MATRIX2D_ASSERT(des.rows == src.rows);
00850     MATRIX2D_ASSERT(des_col < des.cols);
00851
00852     for (size_t i = 0; i < des.rows; i++) {
00853         MAT2D_AT(des, i, des_col) -= MAT2D_AT(src, i, src_col);
00854     }
00855 }
00856
00863 void mat2D_swap_rows(Mat2D m, size_t r1, size_t r2)
00864 {
00865     for (size_t j = 0; j < m.cols; j++) {
00866         double temp = MAT2D_AT(m, r1, j);
00867         MAT2D_AT(m, r1, j) = MAT2D_AT(m, r2, j);
00868         MAT2D_AT(m, r2, j) = temp;
00869     }
00870 }
00871
00879 void mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00880 {
00881     MATRIX2D_ASSERT(src_row < src.rows);
00882     MATRIX2D_ASSERT(des.cols == src.cols);
00883     MATRIX2D_ASSERT(des_row < des.rows);
00884
00885     for (size_t j = 0; j < des.cols; j++) {
00886         MAT2D_AT(des, des_row, j) = MAT2D_AT(src, src_row, j);
00887     }
00888 }
00889
00897 void mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00898 {
00899     MATRIX2D_ASSERT(src_row < src.rows);
00900     MATRIX2D_ASSERT(des.cols == src.cols);
00901     MATRIX2D_ASSERT(des_row < des.rows);
00902
00903     for (size_t j = 0; j < des.cols; j++) {
00904         MAT2D_AT(des, des_row, j) += MAT2D_AT(src, src_row, j);
00905     }
00906 }
00907
00915 void mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00916 {
00917     MATRIX2D_ASSERT(src_row < src.rows);
00918     MATRIX2D_ASSERT(des.cols == src.cols);
00919     MATRIX2D_ASSERT(des_row < des.rows);
00920
00921     for (size_t j = 0; j < des.cols; j++) {
00922         MAT2D_AT(des, des_row, j) -= MAT2D_AT(src, src_row, j);
00923     }
00924 }
00925
00931 double mat2D_calc_norma(Mat2D m)
00932 {
00933     double sum = 0;
00934
00935     for (size_t i = 0; i < m.rows; ++i) {
00936         for (size_t j = 0; j < m.cols; ++j) {
00937             sum += MAT2D_AT(m, i, j) * MAT2D_AT(m, i, j);
00938         }
00939     }
00940     return sqrt(sum);
00941 }
```

```

00942
00949 bool mat2D_mat_is_all_digit(Mat2D m, double digit)
00950 {
00951     for (size_t i = 0; i < m.rows; ++i) {
00952         for (size_t j = 0; j < m.cols; ++j) {
00953             if (MAT2D_AT(m, i, j) != digit) {
00954                 return false;
00955             }
00956         }
00957     }
00958     return true;
00959 }
00960
00968 bool mat2D_row_is_all_digit(Mat2D m, double digit, size_t r)
00969 {
00970     for (size_t j = 0; j < m.cols; ++j) {
00971         if (MAT2D_AT(m, r, j) != digit) {
00972             return false;
00973         }
00974     }
00975     return true;
00976 }
00977
00985 bool mat2D_col_is_all_digit(Mat2D m, double digit, size_t c)
00986 {
00987     for (size_t i = 0; i < m.cols; ++i) {
00988         if (MAT2D_AT(m, i, c) != digit) {
00989             return false;
00990         }
00991     }
00992     return true;
00993 }
00994
01000 double mat2D_det_2x2_mat(Mat2D m)
01001 {
01002     MATRIX2D_ASSERT(2 == m.cols && 2 == m.rows && "Not a 2x2 matrix");
01003     return MAT2D_AT(m, 0, 0) * MAT2D_AT(m, 1, 1) - MAT2D_AT(m, 0, 1) * MAT2D_AT(m, 1, 0);
01004 }
01005
01013 double mat2D_triangulate(Mat2D m)
01014 {
01015     /* preforming Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
01016     /* returns the factor multiplying the determinant */
01017
01018     double factor_to_return = 1;
01019
01020     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
01021         if (!MAT2D_AT(m, i, i)) { /* swapping only if it is zero */
01022             /* finding biggest first number (absolute value) */
01023             size_t biggest_r = i;
01024             for (size_t index = i; index < m.rows; index++) {
01025                 if (fabs(MAT2D_AT(m, index, i)) > fabs(MAT2D_AT(m, biggest_r, i))) {
01026                     biggest_r = index;
01027                 }
01028             }
01029             if (i != biggest_r) {
01030                 mat2D_swap_rows(m, i, biggest_r);
01031             }
01032         }
01033         for (size_t j = i+1; j < m.cols; j++) {
01034             double factor = 1 / MAT2D_AT(m, i, i);
01035             if (!isfinite(factor)) {
01036                 printf("%s:%d: [Error] unable to transfrom into uperr triangular matrix. Probably some
of the rows are not independent.\n", __FILE__, __LINE__);
01037             }
01038             double mat_value = MAT2D_AT(m, j, i);
01039             mat2D_sub_row_time_factor_to_row(m, j, i, mat_value * factor);
01040         }
01041     }
01042     return factor_to_return;
01043 }
01044
01052 double mat2D_det(Mat2D m)
01053 {
01054     MATRIX2D_ASSERT(m.cols == m.rows && "should be a square matrix");
01055
01056     /* checking if there is a row or column with all zeros */
01057     /* checking rows */
01058     for (size_t i = 0; i < m.rows; i++) {
01059         if (mat2D_row_is_all_digit(m, 0, i)) {
01060             return 0;
01061         }
01062     }
01063     /* checking cols */
01064     for (size_t j = 0; j < m.rows; j++) {
01065         if (mat2D_col_is_all_digit(m, 0, j)) {
01066             return 0;
01067         }
01068     }
01069 }
```

```

01067         }
01068     }
01069
01070     /* This is an implementation of naive determinant calculation using minors. This is too slow */
01071
01072     // double det = 0;
01073     // /* TODO: finding beast row or col? */
01074     // for (size_t i = 0, j = 0; i < m.rows; i++) { /* first column */
01075     //     if (MAT2D_AT(m, i, j) < 1e-10) continue;
01076     //     Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat(m, i, j);
01077     //     int factor = (i+j)%2 ? -1 : 1;
01078     //     if (sub_mm.cols != 2) {
01079     //         MATRIX2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
01080     //         det += MAT2D_AT(m, i, j) * (factor) * mat2D_minor_det(sub_mm);
01081     //     } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
01082     //         det += MAT2D_AT(m, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);
01083     //     }
01084     //     mat2D_minor_free(sub_mm);
01085     // }
01086
01087     Mat2D temp_m = mat2D_alloc(m.rows, m.cols);
01088     mat2D_copy(temp_m, m);
01089     double factor = mat2D_triangulate(temp_m);
01090     double diag_mul = 1;
01091     for (size_t i = 0; i < temp_m.rows; i++) {
01092         diag_mul *= MAT2D_AT(temp_m, i, i);
01093     }
01094     mat2D_free(temp_m);
01095
01096     return diag_mul / factor;
01097 }
01098
01099 void mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u)
01100 {
01101     /* performing LU decomposition Following the Wikipedia page:
01102      https://en.wikipedia.org/wiki/LU_decomposition */
01103
01104     mat2D_copy(u, src);
01105     mat2D_set_identity(p);
01106     mat2D_fill(l, 0);
01107
01108     for (size_t i = 0; i < (size_t)fmin(u.rows-1, u.cols); i++) {
01109         if (!MAT2D_AT(u, i, i)) { /* swapping only if it is zero */
01110             /* finding biggest first number (absolute value) */
01111             size_t biggest_r = i;
01112             for (size_t index = i; index < u.rows; index++) {
01113                 if (fabs(MAT2D_AT(u, index, i)) > fabs(MAT2D_AT(u, biggest_r, i))) {
01114                     biggest_r = index;
01115                 }
01116             }
01117             if (i != biggest_r) {
01118                 mat2D_swap_rows(u, i, biggest_r);
01119                 mat2D_swap_rows(p, i, biggest_r);
01120                 mat2D_swap_rows(l, i, biggest_r);
01121             }
01122         }
01123     }
01124     for (size_t j = i+1; j < u.cols; j++) {
01125         double factor = 1 / MAT2D_AT(u, i, i);
01126         if (!isfinite(factor)) {
01127             printf("%s:%d: [Error] unable to transform into upper triangular matrix. Probably some
01128 of the rows are not independent.\n", __FILE__, __LINE__);
01129         }
01130         double mat_value = MAT2D_AT(u, j, i);
01131         mat2D_sub_row_time_factor_to_row(u, j, i, mat_value * factor);
01132         MAT2D_AT(l, j, i) = mat_value * factor;
01133     }
01134     MAT2D_AT(l, i, i) = 1;
01135 }
01136
01137     MAT2D_AT(l, 1.rows-1, 1.cols-1) = 1;
01138 }
01139
01140 void mat2D_transpose(Mat2D des, Mat2D src)
01141 {
01142     MATRIX2D_ASSERT(des.cols == src.rows);
01143     MATRIX2D_ASSERT(des.rows == src.cols);
01144
01145     for (size_t index = 0; index < des.rows; ++index) {
01146         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
01147             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, jndex, index);
01148         }
01149     }
01150 }
01151
01152 void mat2D_invert(Mat2D des, Mat2D src)
01153 {
01154     MATRIX2D_ASSERT(src.cols == src.rows && "should be an NxN matrix");
01155     MATRIX2D_ASSERT(des.cols == src.cols && des.rows == des.cols);

```

```

01173
01174     Mat2D m = mat2D_alloc(src.rows, src.cols);
01175     mat2D_copy(m, src);
01176
01177     mat2D_set_identity(des);
01178
01179     if (!mat2D_det(m)) {
01180         mat2D_fill(des, 0);
01181         printf("%s:%d: [Error] Can't invert the matrix. Determinant is zero! Set the inverse matrix to
01182             all zeros\n", __FILE__, __LINE__);
01183         return;
01184     }
01185
01186     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
01187         if (!MAT2D_AT(m, i, i)) { /* swapping only if it is zero */
01188             /* finding biggest first number (absolute value) */
01189             size_t biggest_r = i;
01190             for (size_t index = i; index < m.rows; index++) {
01191                 if (fabs(MAT2D_AT(m, index, i)) > fabs(MAT2D_AT(m, biggest_r, i))) {
01192                     biggest_r = index;
01193                 }
01194             }
01195             if (i != biggest_r) {
01196                 mat2D_swap_rows(m, i, biggest_r);
01197                 mat2D_swap_rows(des, i, biggest_r);
01198                 printf("%s:%d: [INFO] swapping row %zu with row %zu.\n", __FILE__, __LINE__, i,
01199                     biggest_r);
01200             } else {
01201                 MATRIX2D_ASSERT(0 && "can't inverse");
01202             }
01203         }
01204         for (size_t j = i+1; j < m.cols; j++) {
01205             double factor = 1 / MAT2D_AT(m, i, i);
01206             double mat_value = MAT2D_AT(m, j, i);
01207             mat2D_sub_row_time_factor_to_row(m, j, i, mat_value * factor);
01208             mat2D_mult_row(m, i, factor);
01209
01210             mat2D_sub_row_time_factor_to_row(des, j, i, mat_value * factor);
01211             mat2D_mult_row(des, i, factor);
01212         }
01213         double factor = 1 / MAT2D_AT(m, m.rows-1, m.cols-1);
01214         mat2D_mult_row(m, m.rows-1, factor);
01215         mat2D_mult_row(des, des.rows-1, factor);
01216         for (size_t c = m.cols-1; c > 0; c--) {
01217             for (int r = c-1; r >= 0; r--) {
01218                 double factor = 1 / MAT2D_AT(m, c, c);
01219                 double mat_value = MAT2D_AT(m, r, c);
01220                 mat2D_sub_row_time_factor_to_row(m, r, c, mat_value * factor);
01221                 mat2D_sub_row_time_factor_to_row(des, r, c, mat_value * factor);
01222             }
01223         }
01224         mat2D_free(m);
01225     }
01226
01227 void mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B)
01228 {
01229     MATRIX2D_ASSERT(A.cols == x.rows);
01230     MATRIX2D_ASSERT(1 == x.cols);
01231     MATRIX2D_ASSERT(A.rows == B.rows);
01232     MATRIX2D_ASSERT(1 == B.cols);
01233
01234     Mat2D y      = mat2D_alloc(x.rows, x.cols);
01235     Mat2D l      = mat2D_alloc(A.rows, A.cols);
01236     Mat2D p      = mat2D_alloc(A.rows, A.cols);
01237     Mat2D u      = mat2D_alloc(A.rows, A.cols);
01238     Mat2D inv_l  = mat2D_alloc(l.rows, l.cols);
01239     Mat2D inv_u  = mat2D_alloc(u.rows, u.cols);
01240
01241     mat2D_LUP_decomposition_with_swap(A, l, p, u);
01242
01243     mat2D_invert(inv_l, l);
01244     mat2D_invert(inv_u, u);
01245
01246     mat2D_fill(x, 0); /* x here is only a temp mat*/
01247     mat2D_fill(y, 0);
01248     mat2D_dot(x, p, B);
01249     mat2D_dot(y, inv_l, x);
01250
01251     mat2D_fill(x, 0);
01252     mat2D_dot(x, inv_u, y);
01253
01254     mat2D_free(y);
01255     mat2D_free(l);
01256     mat2D_free(p);
01257     mat2D_free(u);
01258 }
```

```

01267     mat2D_free(inv_l);
01268     mat2D_free(inv_u);
01269 }
01270
01279 Mat2D_Minor mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j)
01280 {
01281     MATRIX2D_ASSERT(ref_mat.cols == ref_mat.rows && "minor is defined only for square matrix");
01282
01283     Mat2D_Minor mm;
01284     mm.cols = ref_mat.cols-1;
01285     mm.rows = ref_mat.rows-1;
01286     mm.stride_r = ref_mat.cols-1;
01287     mm.cols_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mat.cols-1));
01288     mm.rows_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mat.rows-1));
01289     mm.ref_mat = ref_mat;
01290
01291     MATRIX2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
01292
01293     for (size_t index = 0, temp_index = 0; index < ref_mat.rows; index++) {
01294         if (index != i) {
01295             mm.rows_list[temp_index] = index;
01296             temp_index++;
01297         }
01298     }
01299     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mat.rows; jndex++) {
01300         if (jndex != j) {
01301             mm.cols_list[temp_jndex] = jndex;
01302             temp_jndex++;
01303         }
01304     }
01305
01306     return mm;
01307 }
01308
01318 Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j)
01319 {
01320     MATRIX2D_ASSERT(ref_mm.cols == ref_mm.rows && "minor is defined only for square matrix");
01321
01322     Mat2D_Minor mm;
01323     mm.cols = ref_mm.cols-1;
01324     mm.rows = ref_mm.rows-1;
01325     mm.stride_r = ref_mm.cols-1;
01326     mm.cols_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mm.cols-1));
01327     mm.rows_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mm.rows-1));
01328     mm.ref_mat = ref_mm.ref_mat;
01329
01330     MATRIX2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
01331
01332     for (size_t index = 0, temp_index = 0; index < ref_mm.rows; index++) {
01333         if (index != i) {
01334             mm.rows_list[temp_index] = ref_mm.rows_list[index];
01335             temp_index++;
01336         }
01337     }
01338     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mm.rows; jndex++) {
01339         if (jndex != j) {
01340             mm.cols_list[temp_jndex] = ref_mm.cols_list[jndex];
01341             temp_jndex++;
01342         }
01343     }
01344
01345     return mm;
01346 }
01347
01353 void mat2D_minor_free(Mat2D_Minor mm)
01354 {
01355     free(mm.cols_list);
01356     free(mm.rows_list);
01357 }
01358
01365 void mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding)
01366 {
01367     printf("%*s%s = [\n", (int) padding, "", name);
01368     for (size_t i = 0; i < mm.rows; ++i) {
01369         printf("%*s    ", (int) padding, "");
01370         for (size_t j = 0; j < mm.cols; ++j) {
01371             printf("%f ", MAT2D_MINOR_AT(mm, i, j));
01372         }
01373         printf("\n");
01374     }
01375     printf("%*s]\n", (int) padding, "");
01376 }
01377
01383 double mat2D_det_2x2_mat_minor(Mat2D_Minor mm)
01384 {
01385     MATRIX2D_ASSERT(2 == mm.cols && 2 == mm.rows && "Not a 2x2 matrix");
01386     return MAT2D_MINOR_AT(mm, 0, 0) * MAT2D_MINOR_AT(mm, 1, 1) - MAT2D_MINOR_AT(mm, 0, 1) *

```

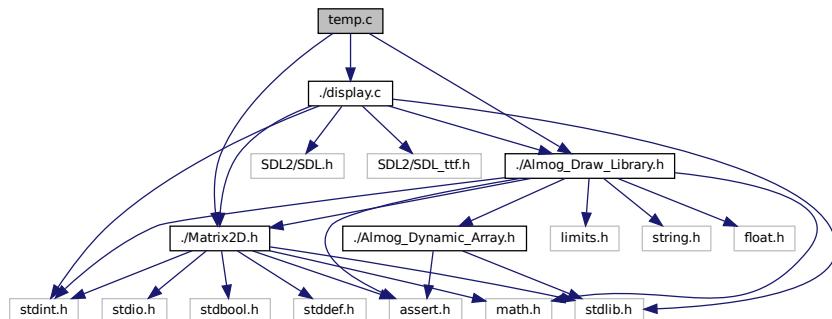
```

    MAT2D_MINOR_AT(mm, 1, 0);
01387 }
01388
01396 double mat2D_minor_det(Mat2D_Minor mm)
01397 {
01398     MATRIX2D_ASSERT(mm.cols == mm.rows && "should be a square matrix");
01399
01400     double det = 0;
01401     /* TODO: finding beast row or col? */
01402     for (size_t i = 0, j = 0; i < mm.rows; i++) { /* first column */
01403         if (MAT2D_MINOR_AT(mm, i, j) < 1e-10) continue;
01404         Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat_minor(mm, i, j);
01405         int factor = (i+j)%2 ? -1 : 1;
01406         if (sub_mm.cols != 2) {
01407             MATRIX2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
01408             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_minor_det(sub_mm);
01409         } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
01410             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);
01411         }
01412         mat2D_minor_free(sub_mm);
01413     }
01414     return det;
01415 }
01416
01417
01418 #endif // MATRIX2D_IMPLEMENTATION

```

4.11 temp.c File Reference

```
#include "./Almog_Draw_Library.h"
#include "./display.c"
#include "./Matrix2D.h"
Include dependency graph for temp.c:
```



Macros

- #define SETUP
- #define UPDATE
- #define RENDER
- #define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
- #define MATRIX2D_IMPLEMENTATION

Functions

- void setup (game_state_t *game_state)
- void update (game_state_t *game_state)
- void render (game_state_t *game_state)

Variables

- Quad quad1
- Tri tri

4.11.1 Macro Definition Documentation

4.11.1.1 ALMOG_DRAW_LIBRARY_IMPLEMENTATION

```
#define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
```

Definition at line [4](#) of file [temp.c](#).

4.11.1.2 MATRIX2D_IMPLEMENTATION

```
#define MATRIX2D_IMPLEMENTATION
```

Definition at line [7](#) of file [temp.c](#).

4.11.1.3 RENDER

```
#define RENDER
```

Definition at line [3](#) of file [temp.c](#).

4.11.1.4 SETUP

```
#define SETUP
```

Definition at line [1](#) of file [temp.c](#).

4.11.1.5 UPDATE

```
#define UPDATE
```

Definition at line [2](#) of file [temp.c](#).

4.11.2 Function Documentation

4.11.2.1 render()

```
void render (
    game_state_t * game_state )
```

Definition at line 50 of file [temp.c](#).

References [adl_linear_map\(\)](#), [adl_quad_draw\(\)](#), [adl_quad_fill_interpolate_color_mean_value\(\)](#), [adl_tri_draw\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), [Mat2D::cols](#), [game_state_t::inv_z_buffer_mat](#), [MAT2D_AT](#), [MAT2D_AT_UINT32](#), [game_state_t::offset_zoom_param](#), [quad1](#), [RGB_hexRGB](#), [Mat2D::rows](#), [tri](#), and [game_state_t::window_pixels_mat](#).

4.11.2.2 setup()

```
void setup (
    game_state_t * game_state )
```

Definition at line 13 of file [temp.c](#).

References [Tri::colors](#), [Quad::colors](#), [Tri::light_intensity](#), [Quad::light_intensity](#), [Tri::points](#), [Quad::points](#), [quad1](#), [Tri::to_draw](#), [Quad::to_draw](#), [game_state_t::to_limit_fps](#), and [tri](#).

4.11.2.3 update()

```
void update (
    game_state_t * game_state )
```

Definition at line 45 of file [temp.c](#).

4.11.3 Variable Documentation

4.11.3.1 quad1

[Quad](#) quad1

Definition at line 10 of file [temp.c](#).

Referenced by [render\(\)](#), and [setup\(\)](#).

4.11.3.2 tri

Tri tri

Definition at line 11 of file [temp.c](#).

Referenced by [adl_tri_draw\(\)](#), [adl_tri_fill_Pinedas_rasterizer\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_color\(\)](#), [adl_tri_fill_Pinedas_rasterizer_interpolate_normal\(\)](#), [adl_tri_mesh_draw\(\)](#), [adl_tri_mesh_fill_Pinedas_rasterizer\(\)](#), [adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color\(\)](#), [adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal\(\)](#), [render\(\)](#), and [setup\(\)](#).

4.12 temp.c

```

00001 #define SETUP
00002 #define UPDATE
00003 #define RENDER
00004 #define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
00005 #include "./Almog_Draw_Library.h"
00006 #include "./display.c"
00007 #define MATRIX2D_IMPLEMENTATION
00008 #include "./Matrix2D.h"
00009
00010 Quad quad1;
00011 Tri tri;
00012
00013 void setup(game_state_t *game_state)
00014 {
00015     // game_state->const_fps = 30;
00016     game_state->to_limit_fps = 0;
00017
00018     quad1.points[3] = (Point){200, 100, 1, 1};
00019     quad1.points[2] = (Point){600, 50, 1, 1};
00020     quad1.points[1] = (Point){200, 700, 1, 1};
00021     quad1.points[0] = (Point){100, 300, 1, 1};
00022     quad1.to_draw = true;
00023     quad1.light_intensity[0] = 1;
00024     quad1.light_intensity[1] = 1;
00025     quad1.light_intensity[2] = 1;
00026     quad1.light_intensity[3] = 1;
00027     quad1.colors[0] = 0xFFFFFFFF;
00028     quad1.colors[1] = 0xFF0000FF;
00029     quad1.colors[2] = 0xFF00FF00;
00030     quad1.colors[3] = 0xFFFF0000;
00031
00032     tri.points[2] = (Point){750, 100, 1, 1};
00033     tri.points[1] = (Point){1250, 700, 1, 1};
00034     tri.points[0] = (Point){650, 500, 1, 1};
00035     tri.to_draw = true;
00036     tri.light_intensity[0] = 1;
00037     tri.light_intensity[1] = 1;
00038     tri.light_intensity[2] = 1;
00039     tri.colors[0] = 0xFFFFFFFF;
00040     tri.colors[1] = 0xFF0000FF;
00041     tri.colors[2] = 0xFF00FF00;
00042
00043 }
00044
00045 void update(game_state_t *game_state)
00046 {
00047     SDL_Delay(1);
00048 }
00049
00050 void render(game_state_t *game_state)
00051 {
00052     adl_quad_fill_interpolate_color_mean_value(game_state->window_pixels_mat,
00053     game_state->inv_z_buffer_mat, quad1, game_state->offset_zoom_param);
00054     adl_quad_draw(game_state->window_pixels_mat, game_state->inv_z_buffer_mat, quad1, 0xFF000000,
00055     game_state->offset_zoom_param);
00056
00057     adl_tri_fill_Pinedas_rasterizer_interpolate_color(game_state->window_pixels_mat,
00058     game_state->inv_z_buffer_mat, tri, game_state->offset_zoom_param);
00059     adl_tri_draw(game_state->window_pixels_mat, tri, 0xff000000, game_state->offset_zoom_param);
00060
00061     #if 0
00062     Mat2D inv_z_buffer = game_state->inv_z_buffer_mat;
00063     double max_inv_z = 0;
00064     double min_inv_z = DBL_MAX;
00065     for (size_t i = 0; i < inv_z_buffer.rows; i++) {

```

```
00063     for (size_t j = 0; j < inv_z_buffer.cols; j++) {
00064         if (MAT2D_AT(inv_z_buffer, i, j) > max_inv_z) {
00065             max_inv_z = MAT2D_AT(inv_z_buffer, i, j);
00066         }
00067         if (MAT2D_AT(inv_z_buffer, i, j) < min_inv_z && MAT2D_AT(inv_z_buffer, i, j) > 0) {
00068             min_inv_z = MAT2D_AT(inv_z_buffer, i, j);
00069         }
00070     }
00071 }
00072 for (size_t i = 0; i < inv_z_buffer.rows; i++) {
00073     for (size_t j = 0; j < inv_z_buffer.cols; j++) {
00074         double z_fraq = MAT2D_AT(inv_z_buffer, i, j);
00075         z_fraq = fmax(z_fraq, min_inv_z);
00076         z_fraq = adl_linear_map(z_fraq, min_inv_z, max_inv_z, 0.1, 1);
00077         uint32_t color = RGB_hexRGB(0xFF*z_fraq, 0xFF*z_fraq, 0xFF*z_fraq);
00078         MAT2D_AT_UINT32(game_state->window_pixels_mat, i, j) = color;
00079     }
00080 }
00081 #endif
00082 }
00083
```

Index

__USE_MISC
Matrix2D.h, 143

a_was_pressed
game_state_t, 15

ada_appand
Almog_Dynamic_Array.h, 113

ADA_ASSERT
Almog_Dynamic_Array.h, 114

ada_init_array
Almog_Dynamic_Array.h, 114

ADA_INIT_CAPACITY
Almog_Dynamic_Array.h, 115

ada_insert
Almog_Dynamic_Array.h, 115

ada_insert_unordered
Almog_Dynamic_Array.h, 116

ADA_MALLOC
Almog_Dynamic_Array.h, 117

ADA_REALLOC
Almog_Dynamic_Array.h, 117

ada_remove
Almog_Dynamic_Array.h, 117

ada_remove_unordered
Almog_Dynamic_Array.h, 118

ada_resize
Almog_Dynamic_Array.h, 118

adl_2Dscalar_interp_on_figure
Almog_Draw_Library.h, 56

adl_arrow_draw
Almog_Draw_Library.h, 57

ADL_ASSERT
Almog_Draw_Library.h, 48

adl_assert_point_is_valid
Almog_Draw_Library.h, 49

adl_assert_quad_is_valid
Almog_Draw_Library.h, 49

adl_assert_tri_is_valid
Almog_Draw_Library.h, 49

adl_axis_draw_on_figure
Almog_Draw_Library.h, 57

adl_cartesian_grid_create
Almog_Draw_Library.h, 58

adl_character_draw
Almog_Draw_Library.h, 59

adl_circle_draw
Almog_Draw_Library.h, 59

adl_circle_fill
Almog_Draw_Library.h, 60

adl_curve_add_to_figure
Almog_Draw_Library.h, 60

adl_curves_plot_on_figure
Almog_Draw_Library.h, 61

ADL_DEFAULT_OFFSET_ZOOM
Almog_Draw_Library.h, 49

adl_figure_alloc
Almog_Draw_Library.h, 62

ADL FIGURE_AXIS_COLOR
Almog_Draw_Library.h, 50

adl_figure_copy_to_screen
Almog_Draw_Library.h, 62

ADL FIGURE_HEAD_ANGLE_DEG
Almog_Draw_Library.h, 50

ADL FIGURE_PADDING_PERCENTAGE
Almog_Draw_Library.h, 50

adl_grid_draw
Almog_Draw_Library.h, 63

adl_interpolate_ARGBcolor_on_okLch
Almog_Draw_Library.h, 63

adl_line_draw
Almog_Draw_Library.h, 64

adl_linear_map
Almog_Draw_Library.h, 64

adl_linear_sRGB_to_okLab
Almog_Draw_Library.h, 65

adl_linear_sRGB_to_okLch
Almog_Draw_Library.h, 65

adl_lines_draw
Almog_Draw_Library.h, 66

adl_lines_loop_draw
Almog_Draw_Library.h, 66

ADL_MAX_CHARACTER_OFFSET
Almog_Draw_Library.h, 50

ADL_MAX FIGURE_PADDING
Almog_Draw_Library.h, 50

ADL_MAX_HEAD_SIZE
Almog_Draw_Library.h, 50

adl_max_min_values_draw_on_figure
Almog_Draw_Library.h, 67

ADL_MAX_POINT_VAL
Almog_Draw_Library.h, 51

ADL_MAX_SENTENCE_LEN
Almog_Draw_Library.h, 51

ADL_MAX_ZOOM
Almog_Draw_Library.h, 51

ADL_MIN_CHARACTER_OFFSET
Almog_Draw_Library.h, 51

ADL_MIN FIGURE_PADDING
Almog_Draw_Library.h, 51

adl_offset2d
 Almog_Draw_Library.h, 51
 adl_offset_zoom_point
 Almog_Draw_Library.h, 52
 adl_okLab_to_linear_sRGB
 Almog_Draw_Library.h, 67
 adl_okLch_to_linear_sRGB
 Almog_Draw_Library.h, 68
 adl_point_draw
 Almog_Draw_Library.h, 68
 adl_quad2tris
 Almog_Draw_Library.h, 69
 adl_quad_draw
 Almog_Draw_Library.h, 70
 adl_quad_fill
 Almog_Draw_Library.h, 70
 adl_quad_fill_interpolate_color_mean_value
 Almog_Draw_Library.h, 71
 adl_quad_fill_interpolate_normal_mean_value
 Almog_Draw_Library.h, 71
 adl_quad_mesh_draw
 Almog_Draw_Library.h, 72
 adl_quad_mesh_fill
 Almog_Draw_Library.h, 73
 adl_quad_mesh_fill_interpolate_color
 Almog_Draw_Library.h, 73
 adl_quad_mesh_fill_interpolate_normal
 Almog_Draw_Library.h, 74
 adl_rectangle_draw_min_max
 Almog_Draw_Library.h, 74
 adl_rectangle_fill_min_max
 Almog_Draw_Library.h, 75
 adl_sentence_draw
 Almog_Draw_Library.h, 75
 adl_tan_half_angle
 Almog_Draw_Library.h, 76
 adl_tri_draw
 Almog_Draw_Library.h, 77
 adl_tri_fill_Pinedas_rasterizer
 Almog_Draw_Library.h, 77
 adl_tri_fill_Pinedas_rasterizer_interpolate_color
 Almog_Draw_Library.h, 78
 adl_tri_fill_Pinedas_rasterizer_interpolate_normal
 Almog_Draw_Library.h, 78
 adl_tri_mesh_draw
 Almog_Draw_Library.h, 79
 adl_tri_mesh_fill_Pinedas_rasterizer
 Almog_Draw_Library.h, 79
 adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color
 Almog_Draw_Library.h, 80
 adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal
 Almog_Draw_Library.h, 81
 Almog_Draw_Library.h, 43
 adl_2Dscalar_interp_on_figure, 56
 adl_arrow_draw, 57
 ADL_ASSERT, 48
 adl_assert_point_is_valid, 49
 adl_assert_quad_is_valid, 49
 adl_assert_tri_is_valid, 49
 adl_axis_draw_on_figure, 57
 adl_cartesian_grid_create, 58
 adl_character_draw, 59
 adl_circle_draw, 59
 adl_circle_fill, 60
 adl_curve_add_to_figure, 60
 adl_curves_plot_on_figure, 61
 ADL_DEFAULT_OFFSET_ZOOM, 49
 adl_figure_alloc, 62
 ADL FIGURE_AXIS_COLOR, 50
 adl_figure_copy_to_screen, 62
 ADL FIGURE_HEAD_ANGLE_DEG, 50
 ADL FIGURE_PADDING_PERCENTAGE, 50
 adl_grid_draw, 63
 adl_interpolate_ARGBcolor_on_okLch, 63
 adl_line_draw, 64
 adl_linear_map, 64
 adl_linear_sRGB_to_okLab, 65
 adl_linear_sRGB_to_okLch, 65
 adl_lines_draw, 66
 adl_lines_loop_draw, 66
 ADL_MAX_CHARACTER_OFFSET, 50
 ADL_MAX FIGURE_PADDING, 50
 ADL_MAX_HEAD_SIZE, 50
 adl_max_min_values_draw_on_figure, 67
 ADL_MAX_POINT_VAL, 51
 ADL_MAX_SENTENCE_LEN, 51
 ADL_MAX_ZOOM, 51
 ADL_MIN_CHARACTER_OFFSET, 51
 ADL_MIN FIGURE_PADDING, 51
 adl_offset2d, 51
 adl_offset_zoom_point, 52
 adl_okLab_to_linear_sRGB, 67
 adl_okLch_to_linear_sRGB, 68
 adl_point_draw, 68
 adl_quad2tris, 69
 adl_quad_draw, 70
 adl_quad_fill, 70
 adl_quad_fill_interpolate_color_mean_value, 71
 adl_quad_fill_interpolate_normal_mean_value, 71
 adl_quad_mesh_draw, 72
 adl_quad_mesh_fill, 73
 adl_quad_mesh_fill_interpolate_color, 73
 adl_quad_mesh_fill_interpolate_normal, 74
 adl_rectangle_draw_min_max, 74
 adl_rectangle_fill_min_max, 75
 adl_sentence_draw, 75
 adl_tan_half_angle, 76
 adl_tri_draw, 77
 adl_tri_fill_Pinedas_rasterizer, 77
 adl_tri_fill_Pinedas_rasterizer_interpolate_color,
 78
 adl_tri_fill_Pinedas_rasterizer_interpolate_normal,
 78
 adl_tri_mesh_draw, 79
 adl_tri_mesh_fill_Pinedas_rasterizer, 79

adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_colorops
 80
adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal,
 81
 Mat2D, 25
 Mat2D_Minor, 27
 Mat2D_uint32, 29
 cols_list
 Mat2D_Minor, 27
const_fps
 game_state_t, 15
copy_mat_to_surface_RGB
 display.c, 125
CURVE
 Almog_Draw_Library.h, 52
Curve, 5
 capacity, 6
 color, 6
 elements, 6
 length, 6
CURVE_ADA
 Almog_Draw_Library.h, 52
Curve_ada, 7
 capacity, 7
 elements, 7
 length, 8
curves
 Grid, 22
CYAN_hexARGB
 Almog_Draw_Library.h, 52
d_was_pressed
 game_state_t, 15
de1
 Grid, 22
de2
 Grid, 22
delta_time
 game_state_t, 15
destroy_window
 display.c, 125
display.c, 121
 check_window_mat_size, 125
 copy_mat_to_surface_RGB, 125
 destroy_window, 125
 dprintCHAR, 123
 dprintD, 123
 dprintINT, 123
 dprintSIZE_T, 123
 dprintSTRING, 123
 fix_framerate, 125
 FPS, 123
 FRAME_TARGET_TIME, 124
 initialize_window, 126
 main, 126
 process_input_window, 126
 RENDER, 124
 render, 126
 render_window, 127
 SETUP, 124
 setup, 127
 setup_window, 127
background_color
 Figure, 9
BLUE_hexARGB
 Almog_Draw_Library.h, 52
capacity
 Curve, 6
 Curve_ada, 7
 Quad_mesh, 36
 Tri_mesh, 40
check_window_mat_size
 display.c, 125
color
 Curve, 6
colors
 Quad, 34
 Tri, 38
 Mat2D, 25
 Mat2D_Minor, 27
 Mat2D_uint32, 29
 cols_list
 Mat2D_Minor, 27
const_fps
 game_state_t, 15
copy_mat_to_surface_RGB
 display.c, 125
CURVE
 Almog_Draw_Library.h, 52
Curve, 5
 capacity, 6
 color, 6
 elements, 6
 length, 6
CURVE_ADA
 Almog_Draw_Library.h, 52
Curve_ada, 7
 capacity, 7
 elements, 7
 length, 8
curves
 Grid, 22
CYAN_hexARGB
 Almog_Draw_Library.h, 52
d_was_pressed
 game_state_t, 15
de1
 Grid, 22
de2
 Grid, 22
delta_time
 game_state_t, 15
destroy_window
 display.c, 125
display.c, 121
 check_window_mat_size, 125
 copy_mat_to_surface_RGB, 125
 destroy_window, 125
 dprintCHAR, 123
 dprintD, 123
 dprintINT, 123
 dprintSIZE_T, 123
 dprintSTRING, 123
 fix_framerate, 125
 FPS, 123
 FRAME_TARGET_TIME, 124
 initialize_window, 126
 main, 126
 process_input_window, 126
 RENDER, 124
 render, 126
 render_window, 127
 SETUP, 124
 setup, 127
 setup_window, 127

UPDATE, 124
 update, 127
 update_window, 128
 WINDOW_HEIGHT, 124
 WINDOW_WIDTH, 124
 dprintCHAR
 display.c, 123
 dprintD
 display.c, 123
 dprintINT
 display.c, 123
 dprintSIZE_T
 display.c, 123
 dprintSTRING
 display.c, 123

 e_was_pressed
 game_state_t, 16
 edge_cross_point
 Almog_Draw_Library.h, 53
 elapsed_time
 game_state_t, 16
 elements
 Curve, 6
 Curve_ada, 7
 Mat2D, 25
 Mat2D_uint32, 29
 Quad_mesh, 36
 Tri_mesh, 41
 example1.c, 133
 ALMOG_DRAW_LIBRARY_IMPLEMENTATION,
 133
 figure1, 135
 figure2, 135
 MATRIX2D_IMPLEMENTATION, 134
 points, 136
 points1, 136
 RENDER, 134
 render, 134
 SETUP, 134
 setup, 135
 UPDATE, 134
 update, 135

 Figure, 8
 background_color, 9
 inv_z_buffer_mat, 9
 max_x, 10
 max_x_pixel, 10
 max_y, 10
 max_y_pixel, 10
 min_x, 11
 min_x_pixel, 11
 min_y, 11
 min_y_pixel, 11
 offset_zoom_param, 12
 pixels_mat, 12
 src_curve_array, 12
 to_draw_axis, 12

 to_draw_max_min_values, 13
 top_left_position, 13
 x_axis_head_size, 13
 y_axis_head_size, 13
 figure1
 example1.c, 135
 figure2
 example1.c, 135
 fix_framerate
 display.c, 125
 font
 game_state_t, 16
 FPS
 display.c, 123
 fps
 game_state_t, 16
 FRAME_TARGET_TIME
 display.c, 124
 frame_target_time
 game_state_t, 16

 game_is_running
 game_state_t, 17
 game_state_t, 14
 a_was_pressed, 15
 const_fps, 15
 d_was_pressed, 15
 delta_time, 15
 e_was_pressed, 16
 elapsed_time, 16
 font, 16
 fps, 16
 frame_target_time, 16
 game_is_running, 17
 inv_z_buffer_mat, 17
 left_button_pressed, 17
 offset_zoom_param, 17
 previous_frame_time, 17
 q_was_pressed, 18
 renderer, 18
 s_was_pressed, 18
 space_bar_was_pressed, 18
 to_clear_renderer, 18
 to_limit_fps, 19
 to_render, 19
 to_update, 19
 w_was_pressed, 19
 window, 19
 window_h, 20
 window_pixels_mat, 20
 window_surface, 20
 window_texture, 20
 window_w, 20
 GREEN_hexRGB
 Almog_Draw_Library.h, 53
 Grid, 21
 curves, 22
 de1, 22
 de2, 22

max_e1, 22
max_e2, 23
min_e1, 23
min_e2, 23
num_samples_e1, 23
num_samples_e2, 24
plane, 24

HexARGB_RGB_VAR
 Almog_Draw_Library.h, 53

HexARGB_RGBA
 Almog_Draw_Library.h, 53

HexARGB_RGBA_VAR
 Almog_Draw_Library.h, 53

initialize_window
 display.c, 126

inv_z_buffer_mat
 Figure, 9
 game_state_t, 17

is_left_edge
 Almog_Draw_Library.h, 54

is_top_edge
 Almog_Draw_Library.h, 54

is_top_left
 Almog_Draw_Library.h, 54

left_button_pressed
 game_state_t, 17

length
 Curve, 6
 Curve_ada, 8
 Quad_mesh, 37
 Tri_mesh, 41

light_intensity
 Quad, 34
 Tri, 38

main
 display.c, 126

Mat2D, 24
 cols, 25
 elements, 25
 rows, 25
 stride_r, 26

mat2D_add
 Matrix2D.h, 145

mat2D_add_col_to_col
 Matrix2D.h, 146

mat2D_add_row_time_factor_to_row
 Matrix2D.h, 146

mat2D_add_row_to_row
 Matrix2D.h, 147

mat2D_alloc
 Matrix2D.h, 147

mat2D_alloc_uint32
 Matrix2D.h, 148

MAT2D_AT
 Matrix2D.h, 143

 MAT2D_AT_UINT32
 Matrix2D.h, 143

 mat2D_calc_norma
 Matrix2D.h, 148

 mat2D_col_is_all_digit
 Matrix2D.h, 149

 mat2D_copy
 Matrix2D.h, 149

 mat2D_copy_mat_to_mat_at_window
 Matrix2D.h, 150

 mat2D_cross
 Matrix2D.h, 150

 mat2D_det
 Matrix2D.h, 151

 mat2D_det_2x2_mat
 Matrix2D.h, 151

 mat2D_det_2x2_mat_minor
 Matrix2D.h, 152

 mat2D_dot
 Matrix2D.h, 152

 mat2D_dot_product
 Matrix2D.h, 153

 mat2D_fill
 Matrix2D.h, 153

 mat2D_fill_sequence
 Matrix2D.h, 154

 mat2D_fill_uint32
 Matrix2D.h, 154

 mat2D_free
 Matrix2D.h, 155

 mat2D_free_uint32
 Matrix2D.h, 155

 mat2D_get_col
 Matrix2D.h, 156

 mat2D_get_row
 Matrix2D.h, 156

 mat2D_invert
 Matrix2D.h, 157

 mat2D_LUP_decomposition_with_swap
 Matrix2D.h, 157

 mat2D_make_identity
 Matrix2D.h, 158

 mat2D_mat_is_all_digit
 Matrix2D.h, 158

 Mat2D_Minor, 26
 cols, 27
 cols_list, 27
 ref_mat, 27
 rows, 28
 rows_list, 28
 stride_r, 28

 mat2D_minor_alloc_fill_from_mat
 Matrix2D.h, 159

 mat2D_minor_alloc_fill_from_mat_minor
 Matrix2D.h, 159

 MAT2D_MINOR_AT
 Matrix2D.h, 143

 mat2D_minor_det

Matrix2D.h, 160
 mat2D_minor_free
 Matrix2D.h, 161
 MAT2D_MINOR_PRINT
 Matrix2D.h, 144
 mat2D_minor_print
 Matrix2D.h, 161
 mat2D_mult
 Matrix2D.h, 162
 mat2D_mult_row
 Matrix2D.h, 162
 mat2D_normalize
 Matrix2D.h, 144
 mat2D_offset2d
 Matrix2D.h, 162
 mat2D_offset2d_uint32
 Matrix2D.h, 164
 MAT2D_PRINT
 Matrix2D.h, 144
 mat2D_print
 Matrix2D.h, 164
 MAT2D_PRINT_AS_COL
 Matrix2D.h, 144
 mat2D_print_as_col
 Matrix2D.h, 165
 mat2D_rand
 Matrix2D.h, 165
 mat2D_rand_double
 Matrix2D.h, 166
 mat2D_row_is_all_digit
 Matrix2D.h, 166
 mat2D_set_DCM_zyx
 Matrix2D.h, 167
 mat2D_set_identity
 Matrix2D.h, 167
 mat2D_set_rot_mat_x
 Matrix2D.h, 168
 mat2D_set_rot_mat_y
 Matrix2D.h, 168
 mat2D_set_rot_mat_z
 Matrix2D.h, 168
 mat2D_solve_linear_sys_LUP_decomposition
 Matrix2D.h, 169
 mat2D_sub
 Matrix2D.h, 169
 mat2D_sub_col_to_col
 Matrix2D.h, 170
 mat2D_sub_row_time_factor_to_row
 Matrix2D.h, 170
 mat2D_sub_row_to_row
 Matrix2D.h, 171
 mat2D_swap_rows
 Matrix2D.h, 171
 mat2D_transpose
 Matrix2D.h, 172
 mat2D_triangulate
 Matrix2D.h, 172
 Mat2D_uint32, 28
 cols, 29
 elements, 29
 rows, 29
 stride_r, 30
 Matrix2D.h, 138
 __USE_MISC, 143
 mat2D_add, 145
 mat2D_add_col_to_col, 146
 mat2D_add_row_time_factor_to_row, 146
 mat2D_add_row_to_row, 147
 mat2D_alloc, 147
 mat2D_alloc_uint32, 148
 MAT2D_AT, 143
 MAT2D_AT_UINT32, 143
 mat2D_calc_norma, 148
 mat2D_col_is_all_digit, 149
 mat2D_copy, 149
 mat2D_copy_mat_to_mat_at_window, 150
 mat2D_cross, 150
 mat2D_det, 151
 mat2D_det_2x2_mat, 151
 mat2D_det_2x2_mat_minor, 152
 mat2D_dot, 152
 mat2D_dot_product, 153
 mat2D_fill, 153
 mat2D_fill_sequence, 154
 mat2D_fill_uint32, 154
 mat2D_free, 155
 mat2D_free_uint32, 155
 mat2D_get_col, 156
 mat2D_get_row, 156
 mat2D_invert, 157
 mat2D_LUP_decomposition_with_swap, 157
 mat2D_make_identity, 158
 mat2D_mat_is_all_digit, 158
 mat2D_minor_alloc_fill_from_mat, 159
 mat2D_minor_alloc_fill_from_mat_minor, 159
 MAT2D_MINOR_AT, 143
 mat2D_minor_det, 160
 mat2D_minor_free, 161
 MAT2D_MINOR_PRINT, 144
 mat2D_minor_print, 161
 mat2D_mult, 162
 mat2D_mult_row, 162
 mat2D_normalize, 144
 mat2D_offset2d, 162
 mat2D_offset2d_uint32, 164
 MAT2D_PRINT, 144
 mat2D_print, 164
 MAT2D_PRINT_AS_COL, 144
 mat2D_print_as_col, 165
 mat2D_rand, 165
 mat2D_rand_double, 166
 mat2D_row_is_all_digit, 166
 mat2D_set_DCM_zyx, 167
 mat2D_set_identity, 167
 mat2D_set_rot_mat_x, 168
 mat2D_set_rot_mat_y, 168

mat2D_set_rot_mat_z, 168
mat2D_solve_linear_sys_LUP_decomposition, 169
mat2D_sub, 169
mat2D_sub_col_to_col, 170
mat2D_sub_row_time_factor_to_row, 170
mat2D_sub_row_to_row, 171
mat2D_swap_rows, 171
mat2D_transpose, 172
mat2D_triangulate, 172
MATRIX2D_ASSERT, 145
MATRIX2D_MALLOC, 145
PI, 145
MATRIX2D_ASSERT
 Matrix2D.h, 145
MATRIX2D_IMPLEMENTATION
 example1.c, 134
 temp.c, 185
MATRIX2D_MALLOC
 Matrix2D.h, 145
max_e1
 Grid, 22
max_e2
 Grid, 23
max_x
 Figure, 10
max_x_pixel
 Figure, 10
max_y
 Figure, 10
max_y_pixel
 Figure, 10
min_e1
 Grid, 23
min_e2
 Grid, 23
min_x
 Figure, 11
min_x_pixel
 Figure, 11
min_y
 Figure, 11
min_y_pixel
 Figure, 11
mouse_x
 Offset_zoom_param, 31
mouse_y
 Offset_zoom_param, 31
normals
 Quad, 35
 Tri, 38
num_samples_e1
 Grid, 23
num_samples_e2
 Grid, 24
offset_x
 Offset_zoom_param, 31
offset_y
 Offset_zoom_param, 30
 mouse_x, 31
 mouse_y, 31
 offset_x, 31
 offset_y, 31
 zoom_multiplier, 31
offset_zoom_param
 Figure, 12
 game_state_t, 17
PI
 Matrix2D.h, 145
pixels_mat
 Figure, 12
plane
 Grid, 24
POINT
 Almog_Draw_Library.h, 54
Point, 32
 w, 32
 x, 32
 y, 33
 z, 33
points
 example1.c, 136
 Quad, 35
 Tri, 38
points1
 example1.c, 136
previous_frame_time
 game_state_t, 17
process_input_window
 display.c, 126
PURPLE_hexRGB
 Almog_Draw_Library.h, 54
q_was_pressed
 game_state_t, 18
QUAD
 Almog_Draw_Library.h, 55
Quad, 34
 colors, 34
 light_intensity, 34
 normals, 35
 points, 35
 to_draw, 35
quad1
 temp.c, 186
QUAD_MESH
 Almog_Draw_Library.h, 55
Quad_mesh, 36
 capacity, 36
 elements, 36
 length, 37
RED_hexRGB
 Almog_Draw_Library.h, 55

ref_mat
 Mat2D_Minor, 27
 RENDER
 display.c, 124
 example1.c, 134
 temp.c, 185
 render
 display.c, 126
 example1.c, 134
 temp.c, 186
 render_window
 display.c, 127
 renderer
 game_state_t, 18
 RGB_hexRGB
 Almog_Draw_Library.h, 55
 RGBA_hexARGB
 Almog_Draw_Library.h, 55
 rows
 Mat2D, 25
 Mat2D_Minor, 28
 Mat2D_uint32, 29
 rows_list
 Mat2D_Minor, 28

 s_was_pressed
 game_state_t, 18
 SETUP
 display.c, 124
 example1.c, 134
 temp.c, 185
 setup
 display.c, 127
 example1.c, 135
 temp.c, 186
 setup_window
 display.c, 127
 space_bar_was_pressed
 game_state_t, 18
 src_curve_array
 Figure, 12
 stride_r
 Mat2D, 26
 Mat2D_Minor, 28
 Mat2D_uint32, 30

 temp.c, 184
 ALMOG_DRAW_LIBRARY_IMPLEMENTATION, 185
 MATRIX2D_IMPLEMENTATION, 185
 quad1, 186
 RENDER, 185
 render, 186
 SETUP, 185
 setup, 186
 tri, 186
 UPDATE, 185
 update, 186
 tex_points

 Tri, 39
 to_clear_renderer
 game_state_t, 18
 to_draw
 Quad, 35
 Tri, 39
 to_draw_axis
 Figure, 12
 to_draw_max_min_values
 Figure, 13
 to_limit_fps
 game_state_t, 19
 to_render
 game_state_t, 19
 to_update
 game_state_t, 19
 top_left_position
 Figure, 13
 TRI
 Almog_Draw_Library.h, 55
 Tri, 37
 colors, 38
 light_intensity, 38
 normals, 38
 points, 38
 tex_points, 39
 to_draw, 39
 tri
 temp.c, 186
 TRI_MESH
 Almog_Draw_Library.h, 56
 Tri_mesh, 40
 capacity, 40
 elements, 41
 length, 41

 UPDATE
 display.c, 124
 example1.c, 134
 temp.c, 185
 update
 display.c, 127
 example1.c, 135
 temp.c, 186
 update_window
 display.c, 128

 w
 Point, 32
 w_was_pressed
 game_state_t, 19
 window
 game_state_t, 19
 window_h
 game_state_t, 20
 WINDOW_HEIGHT
 display.c, 124
 window_pixels_mat
 game_state_t, 20

window_surface
 game_state_t, 20
window_texture
 game_state_t, 20
window_w
 game_state_t, 20
WINDOW_WIDTH
 display.c, 124

x
 Point, 32
x_axis_head_size
 Figure, 13

y
 Point, 33
y_axis_head_size
 Figure, 13

YELLOW_hexARGB
 Almog_Draw_Library.h, 56

z
 Point, 33
zoom_multiplier
 Offset_zoom_param, 31