Almog Draw Library

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

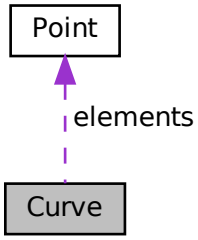
Chapter 3

Class Documentation

3.1 Curve Struct Reference

`#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

Definition at line 60 of file Almog_Draw_Library.h.

3.1.2 Member Data Documentation

3.1.2.1 capacity

```
size_t Curve::capacity
```

Definition at line 63 of file Almog_Draw_Library.h.

3.1.2.2 color

```
uint32_t Curve::color
```

Definition at line 61 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
```

Definition at line 64 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
```

Definition at line 62 of file Almog_Draw_Library.h.

Referenced by adl_curves_plot_on_figure(), adl_grid_draw(), and setup().

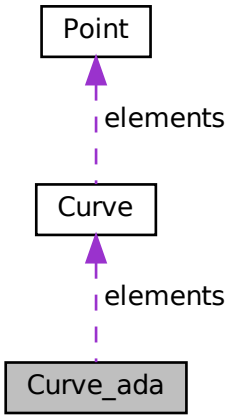
The documentation for this struct was generated from the following file:

- Almog_Draw_Library.h

3.2 Curve_ada Struct Reference

`#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

Definition at line 70 of file Almog_Draw_Library.h.

3.2.2 Member Data Documentation

3.2.2.1 capacity

`size_t Curve_ada::capacity`

Definition at line 72 of file Almog_Draw_Library.h.

3.2.2.2 elements

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

Definition at line 73 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
```

Definition at line 71 of file Almog_Draw_Library.h.

Referenced by adl_curves_plot_on_figure(), and adl_grid_draw().

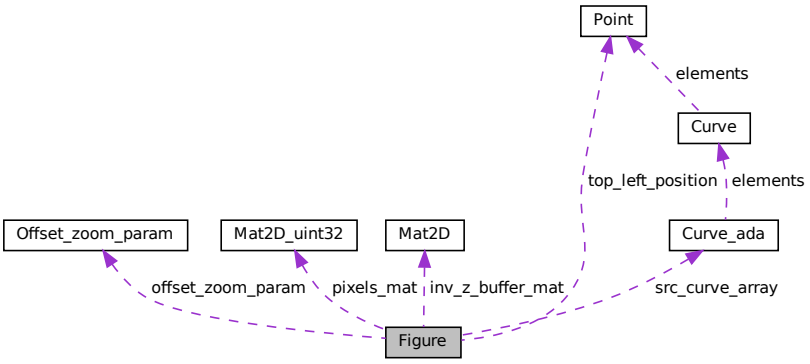
The documentation for this struct was generated from the following file:

- Almog_Draw_Library.h

3.3 Figure Struct Reference

```
#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 inv_z_buffer_mat
- uint32_t background_color
- bool to_draw_axis
- bool to_draw_max_min_values

3.3.1 Detailed Description

Definition at line 118 of file Almog_Draw_Library.h.

3.3.2 Member Data Documentation

3.3.2.1 background_color

```
uint32_t Figure::background_color
```

Definition at line 134 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
```

Definition at line 133 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
```

Definition at line 124 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
```

Definition at line 120 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
```

Definition at line 126 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
```

Definition at line 122 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
```

Definition at line 123 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
```

Definition at line 119 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
```

Definition at line 125 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
```

Definition at line 121 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
```

Definition at line 129 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
```

Definition at line 132 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
```

Definition at line 130 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
```

Definition at line 135 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
```

Definition at line 136 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
```

Definition at line 131 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
```

Definition at line 127 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
```

Definition at line 128 of file Almog_Draw_Library.h.

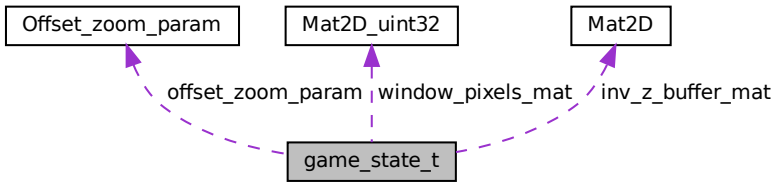
Referenced by adl_axis_draw_on_figure(), and adl_max_min_values_draw_on_figure().

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

```
Offset_zoom_param 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().

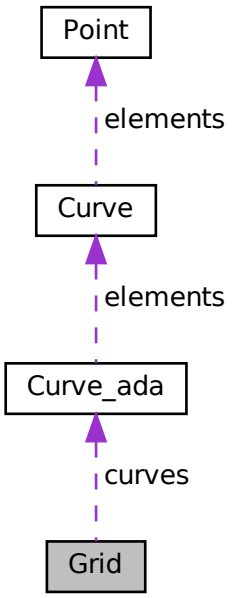
The documentation for this struct was generated from the following file:

- display.c

3.5 Grid Struct Reference

```
#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

Definition at line 139 of file Almog_Draw_Library.h.

3.5.2 Member Data Documentation

3.5.2.1 curves

```
Curve_ada Grid::curves
```

Definition at line 140 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create(), and adl_grid_draw().

3.5.2.2 de1

```
float Grid::de1
```

Definition at line 147 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.5.2.3 de2

```
float Grid::de2
```

Definition at line 148 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.5.2.4 max_e1

```
float Grid::max_e1
```

Definition at line 142 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.5.2.5 max_e2

```
float Grid::max_e2
```

Definition at line 144 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.5.2.6 min_e1

```
float Grid::min_e1
```

Definition at line 141 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.5.2.7 min_e2

```
float Grid::min_e2
```

Definition at line 143 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.5.2.8 num_samples_e1

```
int Grid::num_samples_e1
```

Definition at line 145 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.5.2.9 num_samples_e2

```
int Grid::num_samples_e2
```

Definition at line 146 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.5.2.10 plane

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

Definition at line 149 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

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

Public Attributes

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

3.6.1 Detailed Description

Definition at line 30 of file Matrix2D.h.

3.6.2 Member Data Documentation

3.6.2.1 cols

```
size_t Mat2D::cols
```

Definition at line 32 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_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 34 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 31 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_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 33 of file Matrix2D.h.

Referenced by mat2D_alloc(), and mat2D_offset2d().

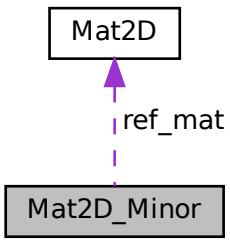
The documentation for this struct was generated from the following file:

- Matrix2D.h

3.7 Mat2D_Minor Struct Reference

```
#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

Definition at line 44 of file Matrix2D.h.

3.7.2 Member Data Documentation

3.7.2.1 cols

```
size_t Mat2D_Minor::cols
```

Definition at line 46 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 49 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 50 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 45 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 48 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 47 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

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

Public Attributes

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

3.8.1 Detailed Description

Definition at line 37 of file Matrix2D.h.

3.8.2 Member Data Documentation

3.8.2.1 cols

```
size_t Mat2D_uint32::cols
```

Definition at line 39 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_m 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(), mat2D_uint32_print(), and render_window().

3.8.2.2 elements

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

Definition at line 41 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 38 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_val 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(), mat2D_uint32_print(), and render_window().

3.8.2.4 stride_r

```
size_t Mat2D_uint32::stride_r
```

Definition at line 40 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

`#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

Definition at line 40 of file Almog_Draw_Library.h.

3.9.2 Member Data Documentation

3.9.2.1 mouse_x

`int Offset_zoom_param::mouse_x`

Definition at line 44 of file Almog_Draw_Library.h.

3.9.2.2 mouse_y

`int Offset_zoom_param::mouse_y`

Definition at line 45 of file Almog_Draw_Library.h.

3.9.2.3 offset_x

`float Offset_zoom_param::offset_x`

Definition at line 42 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
```

Definition at line 43 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
```

Definition at line 41 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

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

Public Attributes

- float x
- float y
- float z
- float w

3.10.1 Detailed Description

Definition at line 50 of file Almog_Draw_Library.h.

3.10.2 Member Data Documentation

3.10.2.1 w

```
float Point::w
```

Definition at line 54 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_v
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
```

Definition at line 51 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
```

Definition at line 52 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
```

Definition at line 53 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_v
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().

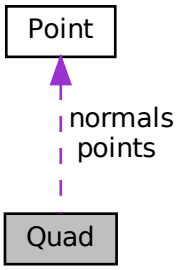
The documentation for this struct was generated from the following file:

- Almog_Draw_Library.h

3.11 Quad Struct Reference

`#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

Definition at line 91 of file Almog_Draw_Library.h.

3.11.2 Member Data Documentation

3.11.2.1 colors

`uint32_t Quad::colors[4]`

Definition at line 94 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]
```

Definition at line 96 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]
```

Definition at line 93 of file Almog_Draw_Library.h.

3.11.2.4 points

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

Definition at line 92 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_me
adl_quad_fill_interpolate_normal_mean_value(), and setup().

3.11.2.5 to_draw

```
bool Quad::to_draw
```

Definition at line 95 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().

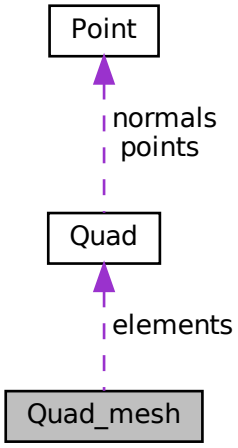
The documentation for this struct was generated from the following file:

- Almog_Draw_Library.h

3.12 Quad_mesh Struct Reference

`#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

Definition at line 111 of file Almog_Draw_Library.h.

3.12.2 Member Data Documentation

3.12.2.1 capacity

`size_t Quad_mesh::capacity`

Definition at line 113 of file Almog_Draw_Library.h.

3.12.2.2 elements

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

Definition at line 114 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
```

Definition at line 112 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().

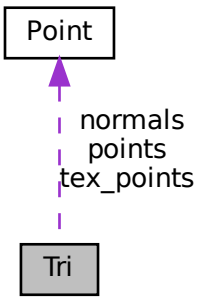
The documentation for this struct was generated from the following file:

- Almog_Draw_Library.h

3.13 Tri Struct Reference

```
#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

Definition at line 79 of file Almog_Draw_Library.h.

3.13.2 Member Data Documentation

3.13.2.1 colors

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

Definition at line 83 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]
```

Definition at line 85 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]
```

Definition at line 82 of file Almog_Draw_Library.h.

3.13.2.4 points

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

Definition at line 80 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]
```

Definition at line 81 of file Almog_Draw_Library.h.

3.13.2.6 to_draw

```
bool Tri::to_draw
```

Definition at line 84 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_int
adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal(), and setup().

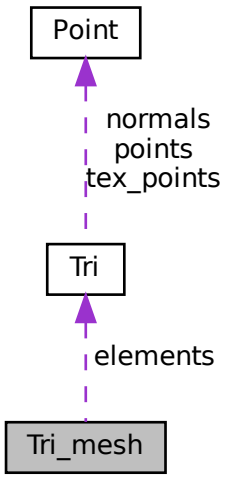
The documentation for this struct was generated from the following file:

- Almog_Draw_Library.h

3.14 Tri_mesh Struct Reference

```
#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

Definition at line 102 of file Almog_Draw_Library.h.

3.14.2 Member Data Documentation

3.14.2.1 capacity

```
size_t Tri_mesh::capacity
```

Definition at line 104 of file Almog_Draw_Library.h.

3.14.2.2 elements

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

Definition at line 105 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
```

Definition at line 103 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

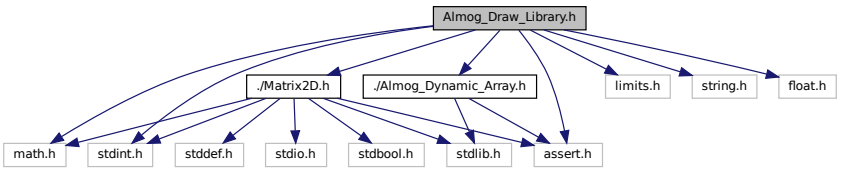
File Documentation

4.1 Almog_Draw_Library.h File Reference

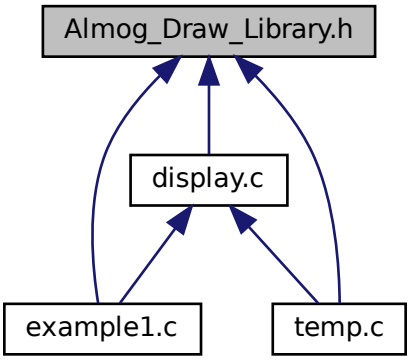
Immediate-mode 2D/3D raster helpers for drawing onto Mat2D_uint32 pixel buffers.

```
#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
- struct Point
- struct Curve
- struct Curve_ada
- struct Tri
- struct Quad
- struct Tri_mesh
- struct Quad_mesh
- struct Figure
- struct Grid

Macros

- #define ADL_ASSERT assert
- #define POINT
- #define CURVE
- #define CURVE_ADA
- #define TRI
- #define QUAD
- #define TRI_MESH
- #define QUAD_MESH
- #define HexARGB_RGBA(x) ((x)>>(8∗2)&0xFF), ((x)>>(8∗1)&0xFF), ((x)>>(8∗0)&0xFF), ((x)>>(8∗3)&0x↩ FF)
- #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)(0x01000000l∗(int)(fminf(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 0xFFFFFF00
- #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_PRECENTAGE 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 hight_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 hight_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_2↩ Dmat, 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 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.

Conventions

- Pixel buffer: Mat2D_uint32 with elements encoded as ARGB 0xAARRGGBB.
- Coordinates: x grows to the right, y grows downward; origin is the top-left corner of the destination buffer.
- Depth: Functions that accept inv_z_buffer perform a depth test using inverse-Z (larger values are closer). The buffer stores doubles.
- Transform: Most drawing functions accept an Offset_zoom_param describing a pan/zoom transform that is applied about the screen center. Use ADL_DEFAULT_OFFSET_ZOOM for identity.
- Colors: Unless noted otherwise, colors are ARGB in 0xAARRGGBB format.
- Alpha: adl_point_draw alpha-blends source over destination and writes an opaque result (A = 255) to the pixel buffer.

This header contains function declarations and optional implementations (guarded by ALMOG_DRAW_LIBRARY↩ _IMPLEMENTATION).

Definition in file Almog_Draw_Library.h.

4.1.2 Macro Definition Documentation

4.1.2.1 ADL_ASSERT

```
#define ADL_ASSERT assert
```

Definition at line 37 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 243 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 247 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 244 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 264 of file Almog_Draw_Library.h.

4.1.2.6 ADL_FIGURE_AXIS_COLOR

```
#define ADL_FIGURE_AXIS_COLOR 0xff000000
```

Definition at line 257 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 256 of file Almog_Draw_Library.h.

4.1.2.8 ADL_FIGURE_PADDING_PRECENTAGE

```
#define ADL_FIGURE_PADDING_PRECENTAGE 20
```

Definition at line 252 of file Almog_Draw_Library.h.

4.1.2.9 ADL_MAX_CHARACTER_OFFSET

```
#define ADL_MAX_CHARACTER_OFFSET 10
```

Definition at line 259 of file Almog_Draw_Library.h.

4.1.2.10 ADL_MAX_FIGURE_PADDING

```
#define ADL_MAX_FIGURE_PADDING 70
```

Definition at line 253 of file Almog_Draw_Library.h.

4.1.2.11 ADL_MAX_HEAD_SIZE

```
#define ADL_MAX_HEAD_SIZE 15
```

Definition at line 255 of file Almog_Draw_Library.h.

4.1.2.12 ADL_MAX_POINT_VAL

```
#define ADL_MAX_POINT_VAL 1e5
```

Definition at line 242 of file Almog_Draw_Library.h.

4.1.2.13 ADL_MAX_SENTENCE_LEN

```
#define ADL_MAX_SENTENCE_LEN 256
```

Definition at line 261 of file Almog_Draw_Library.h.

4.1.2.14 ADL_MAX_ZOOM

```
#define ADL_MAX_ZOOM 1e3
```

Definition at line 262 of file Almog_Draw_Library.h.

4.1.2.15 ADL_MIN_CHARACTER_OFFSET

```
#define ADL_MIN_CHARACTER_OFFSET 5
```

Definition at line 260 of file Almog_Draw_Library.h.

4.1.2.16 ADL_MIN_FIGURE_PADDING

```
#define ADL_MIN_FIGURE_PADDING 20
```

Definition at line 254 of file Almog_Draw_Library.h.

4.1.2.17 adl_offset2d

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

Definition at line 2227 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 265 of file Almog_Draw_Library.h.

4.1.2.19 BLUE_hexARGB

```
#define BLUE_hexARGB 0xFF0000FF
```

Definition at line 232 of file Almog_Draw_Library.h.

4.1.2.20 CURVE

```
#define CURVE
```

Definition at line 59 of file Almog_Draw_Library.h.

4.1.2.21 CURVE_ADA

```
#define CURVE_ADA
```

Definition at line 69 of file Almog_Draw_Library.h.

4.1.2.22 CYAN_hexARGB

```
#define CYAN_hexARGB 0xFF00FFFF
```

Definition at line 234 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 237 of file Almog_Draw_Library.h.

4.1.2.24 GREEN_hexARGB

```
#define GREEN_hexARGB 0xFF00FF00
```

Definition at line 231 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 157 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 154 of file Almog_Draw_Library.h.

4.1.2.27 HexARGB_RGBA_VAR

```
#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)
```

Definition at line 160 of file Almog_Draw_Library.h.

4.1.2.28 is_left_edge

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

Definition at line 239 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 238 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 240 of file Almog_Draw_Library.h.

4.1.2.31 POINT

```
#define POINT
```

Definition at line 49 of file Almog_Draw_Library.h.

4.1.2.32 PURPLE_hexARGB

```
#define PURPLE_hexARGB 0xFFFF00FF
```

Definition at line 233 of file Almog_Draw_Library.h.

4.1.2.33 QUAD

```
#define QUAD
```

Definition at line 90 of file Almog_Draw_Library.h.

4.1.2.34 QUAD_MESH

```
#define QUAD_MESH
```

Definition at line 110 of file Almog_Draw_Library.h.

4.1.2.35 RED_hexARGB

```
#define RED_hexARGB 0xFFFF0000
```

Definition at line 230 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 163 of file Almog_Draw_Library.h.

4.1.2.37 RGBA_hexARGB

```
#define RGBA_hexARGB(
              r,
              g,
              b,
              a ) (int)(0x01000000l*(int)(fminf(a, 255)) + 0x010000*(int)(r) + 0x000100*(int)(g)
+ 0x000001*(int)(b))
```

Definition at line 166 of file Almog_Draw_Library.h.

4.1.2.38 TRI

```
#define TRI
```

Definition at line 78 of file Almog_Draw_Library.h.

4.1.2.39 TRI_MESH

```
#define TRI_MESH
```

Definition at line 101 of file Almog_Draw_Library.h.

4.1.2.40 YELLOW_hexARGB

```
#define YELLOW_hexARGB 0xFFFFFF00
```

Definition at line 235 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

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

Definition at line 2247 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_mea 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

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

Definition at line 451 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

figure	[in,out] Figure to draw onto.

Definition at line 2077 of file Almog_Draw_Library.h.

References adl_arrow_draw(), ADL_FIGURE_AXIS_COLOR, ADL_FIGURE_HEAD_ANGLE_DEG, ADL_FIGURE_PADDING_PREC ADL_MAX_FIGURE_PADDING, ADL_MAX_HEAD_SIZE, ADL_MIN_FIGURE_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

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

Returns

Grid structure containing the generated curves and spacing.

Definition at line 2446 of file Almog_Draw_Library.h.

References ada_append, 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 hight_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

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

Definition at line 519 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

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

Definition at line 1360 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

screen_mat	Destination ARGB pixel buffer.

Parameters

center_x	Circle center X (before pan/zoom).
center_y	Circle center Y (before pan/zoom).
r	Circle radius in pixels.
color	Fill color (0xAARRGGBB).
offset_zoom_param	Pan/zoom transform. Use ADL_DEFAULT_OFFSET_ZOOM for identity.

Definition at line 1382 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

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

Definition at line 2163 of file Almog_Draw_Library.h.

References ada_append, 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

figure	Figure to render into (uses its own pixel buffer).

Definition at line 2198 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

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

Returns

A new Figure with allocated buffers.

Definition at line 2014 of file Almog_Draw_Library.h.

References ada_init_array, ADL_ASSERT, adl_assert_point_is_valid, ADL_DEFAULT_OFFSET_ZOOM, ADL_FIGURE_PADDING_PRECENTAGE, 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

screen_mat	Destination ARGB pixel buffer.
figure	Source figure to copy from.

Definition at line 2057 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

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

Definition at line 2724 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∗num_of_rotations to the second hue, allowing control over the winding direction.

Parameters

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

Definition at line 1986 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

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

Definition at line 316 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

s	Input value.
min_in	Input range minimum.
max_in	Input range maximum.
min_out	Output range minimum.
max_out	Output range maximum.

Returns

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

Definition at line 1798 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

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

Definition at line 1878 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

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

Definition at line 1945 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

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

Definition at line 403 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

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

Definition at line 423 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

figure	Figure whose labels are drawn into its own pixel buffer.

Definition at line 2103 of file Almog_Draw_Library.h.

References ADL_FIGURE_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

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

Definition at line 1913 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

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

Definition at line 1964 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

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

Definition at line 283 of file Almog_Draw_Library.h.

References Mat2D_uint32::cols, HexARGB_RGBA_VAR, MAT2D_AT_UINT32, Offset_zoom_param::offset_x, Offset_zoom_param::offset_y, RGBA_hexARGB, Mat2D_uint32::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

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

Definition at line 1818 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

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

Definition at line 943 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

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

Definition at line 961 of file Almog_Draw_Library.h.

References adl_point_draw(), 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().

4.1.3.27 adl_quad_fill_interpolate_color_mean_value()

```
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).

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

Parameters

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

Definition at line 1149 of file Almog_Draw_Library.h.

References adl_point_draw(), adl_tan_half_angle(), Quad::colors, 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_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

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

Definition at line 1055 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

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

Definition at line 1253 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 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

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

Definition at line 1277 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 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

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

Definition at line 1334 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 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

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

Definition at line 1304 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

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

Definition at line 906 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

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

Definition at line 925 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

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

Definition at line 882 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×b| / (|a||b| + a·b), where a = vi - p and b = vj - p. The lengths li = |a| and lj = |b| are passed in to avoid recomputation.

Parameters

vi	Vertex i.
vj	Vertex j.
p	Pivot point.
li	Precomputed |vi - p|.
lj	Precomputed |vj - p|.

Returns

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

Definition at line 1778 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

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

Definition at line 1402 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

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

Definition at line 1425 of file Almog_Draw_Library.h.

References adl_point_draw(), Mat2D_uint32::cols, edge_cross_point, HexARGB_RGBA_VAR, is_top_left, Tri::light_intensity, MAT2D_AT, MATRIX2D_ASSERT, Tri::points, RGBA_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

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

Definition at line 1506 of file Almog_Draw_Library.h.

References adl_point_draw(), Tri::colors, Mat2D_uint32::cols, edge_cross_point, HexARGB_RGBA_VAR, is_top_left, Tri::light_intensity, MAT2D_AT, MATRIX2D_ASSERT, Tri::points, RGBA_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

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

Definition at line 1597 of file Almog_Draw_Library.h.

References adl_point_draw(), Mat2D_uint32::cols, edge_cross_point, HexARGB_RGBA_VAR, is_top_left, Tri::light_intensity, MAT2D_AT, MATRIX2D_ASSERT, Tri::points, RGBA_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

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

Definition at line 1679 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 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

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

Definition at line 1701 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 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

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

Definition at line 1725 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

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

Definition at line 1750 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
00023 #ifndef ALMOG_DRAW_LIBRARY_H_
00024 #define ALMOG_DRAW_LIBRARY_H_
00025
00026 #include <math.h>
00027 #include <stdint.h>
00028 #include <limits.h>
00029 #include <string.h>
00030 #include <float.h>
00031
00032 #include "./Matrix2D.h"
00033 #include "./Almog_Dynamic_Array.h"
00034
00035 #ifndef ADL_ASSERT
00036 #include <assert.h>
00037 #define ADL_ASSERT assert
00038 #endif
00039
00040 typedef struct {
00041     float zoom_multiplier;
00042     float offset_x;
00043     float offset_y;
00044     int mouse_x;
00045     int mouse_y;
00046 } Offset_zoom_param;
00047
00048 #ifndef POINT
00049 #define POINT
00050 typedef struct {
00051     float x;
00052     float y;
00053     float z;
00054     float w;
00055 } Point ;
00056 #endif
00057
00058 #ifndef CURVE
00059 #define CURVE
00060 typedef struct {
00061     uint32_t color;
00062     size_t length;
00063     size_t capacity;
00064     Point *elements;
00065 } Curve;
00066 #endif
00067
00068 #ifndef CURVE_ADA
00069 #define CURVE_ADA
00070 typedef struct {
00071     size_t length;
00072     size_t capacity;
00073     Curve *elements;
00074 } Curve_ada;
00075 #endif
00076
00077 #ifndef TRI
00078 #define TRI
00079 typedef struct {
00080     Point points[3];
00081     Point tex_points[3];
00082     Point normals[3];
00083     uint32_t colors[3];
00084     bool to_draw;
00085     float light_intensity[3];
00086 } Tri;
00087 #endif
00088
00089 #ifndef QUAD
00090 #define QUAD
00091 typedef struct {
00092     Point points[4];
00093     Point normals[4];
00094     uint32_t colors[4];
00095     bool to_draw;
00096     float light_intensity[4];
00097 } Quad;
00098 #endif
00099
00100 #ifndef TRI_MESH
00101 #define TRI_MESH
00102 typedef struct {
00103     size_t length;
00104     size_t capacity;
00105     Tri *elements;
00106 } Tri_mesh; /* Tri ada array */
```

```
00107 #endif
00108
00109 #ifndef QUAD_MESH
00110 #define QUAD_MESH
00111 typedef struct {
00112     size_t length;
00113     size_t capacity;
00114     Quad *elements;
00115 } Quad_mesh; /* Quad ada array */
00116 #endif
00117
00118 typedef struct {
00119     int min_x_pixel;
00120     int max_x_pixel;
00121     int min_y_pixel;
00122     int max_y_pixel;
00123     float min_x;
00124     float max_x;
00125     float min_y;
00126     float max_y;
00127     int x_axis_head_size;
00128     int y_axis_head_size;
00129     Offset_zoom_param offset_zoom_param;
00130     Curve_ada src_curve_array;
00131     Point top_left_position;
00132     Mat2D_uint32 pixels_mat;
00133     Mat2D inv_z_buffer_mat;
00134     uint32_t background_color;
00135     bool to_draw_axis;
00136     bool to_draw_max_min_values;
00137 } Figure;
00138
00139 typedef struct {
00140     Curve_ada curves;
00141     float min_e1;
00142     float max_e1;
00143     float min_e2;
00144     float max_e2;
00145     int num_samples_e1;
00146     int num_samples_e2;
00147     float de1;
00148     float de2;
00149     char plane[3];
00150 } Grid; /* direction: e1, e2 */
00151
00152
00153 #ifndef HexARGB_RGBA
00154 #define HexARGB_RGBA(x) ((x)»(8*2)&0xFF), ((x)»(8*1)&0xFF), ((x)»(8*0)&0xFF), ((x)»(8*3)&0xFF)
00155 #endif
00156 #ifndef HexARGB_RGB_VAR
00157 #define HexARGB_RGB_VAR(x, r, g, b) r = ((x)»(8*2)&0xFF); g = ((x)»(8*1)&0xFF); b = ((x)»(8*0)&0xFF);
00158 #endif
00159 #ifndef HexARGB_RGBA_VAR
00160 #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)
00161 #endif
00162 #ifndef RGB_hexRGB
00163 #define RGB_hexRGB(r, g, b) (int)(0x010000*(int)(r) + 0x000100*(int)(g) + 0x000001*(int)(b))
00164 #endif
00165 #ifndef RGBA_hexARGB
00166 #define RGBA_hexARGB(r, g, b, a) (int)(0x010000001*(int)(fminf(a, 255)) + 0x010000*(int)(r) +
      0x000100*(int)(g) + 0x000001*(int)(b))
00167 #endif
00168
00169
00170 void    adl_point_draw(Mat2D_uint32 screen_mat, int x, int y, uint32_t color, Offset_zoom_param
      offset_zoom_param);
00171 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);
00172 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);
00173 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);
00174 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);
00175
00176 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);
00177 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);
00178
00179 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);
00180 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);
00181
```

```
00182 void    adl_quad_draw(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
       Offset_zoom_param offset_zoom_param);
00183 void    adl_quad_fill(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
       Offset_zoom_param offset_zoom_param);
00184 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);
00185 void    adl_quad_fill_interpolate_color_mean_value(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad
       quad, Offset_zoom_param offset_zoom_param);
00186
00187 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);
00188 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);
00189 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);
00190 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);
00191
00192 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);
00193 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);
00194
00195 void    adl_tri_draw(Mat2D_uint32 screen_mat, Tri tri, uint32_t color, Offset_zoom_param
       offset_zoom_param);
00196 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);
00197 void    adl_tri_fill_Pinedas_rasterizer_interpolate_color(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer,
       Tri tri, Offset_zoom_param offset_zoom_param);
00198 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);
00199
00200 void    adl_tri_mesh_draw(Mat2D_uint32 screen_mat, Tri_mesh mesh, uint32_t color, Offset_zoom_param
       offset_zoom_param);
00201 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);
00202 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);
00203 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);
00204
00205 float   adl_tan_half_angle(Point vi, Point vj, Point p, float li, float lj);
00206 float   adl_linear_map(float s, float min_in, float max_in, float min_out, float max_out);
00207 void    adl_quad2tris(Quad quad, Tri *tri1, Tri *tri2, char split_line[]);
00208 void    adl_linear_sRGB_to_okLab(uint32_t hex_ARGB, float *L, float *a, float *b);
00209 void    adl_okLab_to_linear_sRGB(float L, float a, float b, uint32_t *hex_ARGB);
00210 void    adl_linear_sRGB_to_okLch(uint32_t hex_ARGB, float *L, float *c, float *h_deg);
00211 void    adl_okLch_to_linear_sRGB(float L, float c, float h_deg, uint32_t *hex_ARGB);
00212 void    adl_interpolate_ARGBcolor_on_okLch(uint32_t color1, uint32_t color2, float t, float
       num_of_rotations, uint32_t *color_out);
00213
00214 Figure  adl_figure_alloc(size_t rows, size_t cols, Point top_left_position);
00215 void    adl_figure_copy_to_screen(Mat2D_uint32 screen_mat, Figure figure);
00216 void    adl_axis_draw_on_figure(Figure *figure);
00217 void    adl_max_min_values_draw_on_figure(Figure figure);
00218 void    adl_curve_add_to_figure(Figure *figure, Point *src_points, size_t src_len, uint32_t color);
00219 void    adl_curves_plot_on_figure(Figure figure);
00220 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);
00221
00222 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);
00223 void    adl_grid_draw(Mat2D_uint32 screen_mat, Grid grid, uint32_t color, Offset_zoom_param
       offset_zoom_param);
00224
00225 #endif /*ALMOG_RENDER_SHAPES_H_*/
00226
00227 #ifdef ALMOG_DRAW_LIBRARY_IMPLEMENTATION
00228 #undef ALMOG_DRAW_LIBRARY_IMPLEMENTATION
00229
00230 #define RED_hexARGB    0xFFFF0000
00231 #define GREEN_hexARGB  0xFF00FF00
00232 #define BLUE_hexARGB   0xFF0000FF
00233 #define PURPLE_hexARGB 0xFFFF00FF
00234 #define CYAN_hexARGB   0xFF00FFFF
00235 #define YELLOW_hexARGB 0xFFFFFF00
00236
00237 #define edge_cross_point(a1, b, a2, p) (b.x-a1.x)*(p.y-a2.y)-(b.y-a1.y)*(p.x-a2.x)
00238 #define is_top_edge(x, y) (y == 0 && x > 0)
00239 #define is_left_edge(x, y) (y < 0)
00240 #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))
00241
00242 #define ADL_MAX_POINT_VAL 1e5
00243 #define adl_assert_point_is_valid(p) ADL_ASSERT(isfinite(p.x) && isfinite(p.y) && isfinite(p.z) &&
       isfinite(p.w))
00244 #define adl_assert_tri_is_valid(tri) adl_assert_point_is_valid(tri.points[0]); \
00245         adl_assert_point_is_valid(tri.points[1]);                              \
```

```
00246         adl_assert_point_is_valid(tri.points[2])
00247 #define adl_assert_quad_is_valid(quad) adl_assert_point_is_valid(quad.points[0]);   \
00248         adl_assert_point_is_valid(quad.points[1]);                                  \
00249         adl_assert_point_is_valid(quad.points[2]);                                  \
00250         adl_assert_point_is_valid(quad.points[3])
00251
00252 #define ADL_FIGURE_PADDING_PRECENTAGE 20
00253 #define ADL_MAX_FIGURE_PADDING 70
00254 #define ADL_MIN_FIGURE_PADDING 20
00255 #define ADL_MAX_HEAD_SIZE 15
00256 #define ADL_FIGURE_HEAD_ANGLE_DEG 30
00257 #define ADL_FIGURE_AXIS_COLOR 0xff000000
00258
00259 #define ADL_MAX_CHARACTER_OFFSET 10
00260 #define ADL_MIN_CHARACTER_OFFSET 5
00261 #define ADL_MAX_SENTENCE_LEN 256
00262 #define ADL_MAX_ZOOM 1e3
00263
00264 #define ADL_DEFAULT_OFFSET_ZOOM (Offset_zoom_param){1,0,0,0,0}
00265 #define adl_offset_zoom_point(p, window_w, window_h, offset_zoom_param)
                \
00266     (p).x = ((p).x - (window_w)/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier +
     (window_w)/2; \
00267     (p).y = ((p).y - (window_h)/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier +
     (window_h)/2
00268
00283 void adl_point_draw(Mat2D_uint32 screen_mat, int x, int y, uint32_t color, Offset_zoom_param
     offset_zoom_param)
00284 {
00285     float window_w = (float)screen_mat.cols;
00286     float window_h = (float)screen_mat.rows;
00287
00288     x = (x - window_w/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier +
     window_w/2;
00289     y = (y - window_h/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier +
     window_h/2;
00290
00291     if ((x < (int)screen_mat.cols && y < (int)screen_mat.rows) && (x >= 0 && y >= 0)) { /* point is in
     screen */
00292         uint8_t r_new, g_new, b_new, a_new;
00293         uint8_t r_current, g_current, b_current, a_current;
00294         HexARGB_RGBA_VAR(MAT2D_AT_UINT32(screen_mat, y, x), r_current, g_current, b_current,
     a_current);
00295         HexARGB_RGBA_VAR(color, r_new, g_new, b_new, a_new);
00296         MAT2D_AT_UINT32(screen_mat, y, x) = RGBA_hexARGB(r_current*(1-a_new/255.0f) +
     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) +
     b_new*a_new/255.0f, 255);
00297         (void)a_current;
00298     }
00299 }
00300
00316 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)
00317 {
00318     /* This function is inspired by the Olive.c function developed by 'Tsoding' on his YouTube
     channel. You can fined the video in this link:
     https://youtu.be/LmQKZmQh1ZQ?list=PLpM-Dvs8t0Va-Gb0Dp4d9t8yvNFHaKH6N&t=4683. */
00319
00320     float window_w = (float)screen_mat.cols;
00321     float window_h = (float)screen_mat.rows;
00322
00323     int x1 = (x1_input - window_w/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier
     + window_w/2;
00324     int x2 = (x2_input - window_w/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier
     + window_w/2;
00325     int y1 = (y1_input - window_h/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier
     + window_h/2;
00326     int y2 = (y2_input - window_h/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier
     + window_h/2;
00327
00328     ADL_ASSERT((int)fabsf(fabsf((float)x2) - fabsf((float)x1)) < ADL_MAX_POINT_VAL);
00329     ADL_ASSERT((int)fabsf(fabsf((float)y2) - fabsf((float)y1)) < ADL_MAX_POINT_VAL);
00330
00331     int x = x1;
00332     int y = y1;
00333     int dx, dy;
00334
00335     adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00336
00337     dx = x2 - x1;
00338     dy = y2 - y1;
00339
00340     ADL_ASSERT(dy > INT_MIN && dy < INT_MAX);
00341     ADL_ASSERT(dx > INT_MIN && dx < INT_MAX);
00342
00343     if (0 == dx && 0 == dy) return;
00344     if (0 == dx) {
```

```
00345         while (x != x2 || y != y2) {
00346             if (dy > 0) {
00347                 y++;
00348             }
00349             if (dy < 0) {
00350                 y--;
00351             }
00352             adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00353         }
00354         return;
00355     }
00356     if (0 == dy) {
00357         while (x != x2 || y != y2) {
00358             if (dx > 0) {
00359                 x++;
00360             }
00361             if (dx < 0) {
00362                 x--;
00363             }
00364             adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00365         }
00366         return;
00367     }
00368
00369     /* float m = (float)dy / dx */
00370     int b = y1 - dy * x1 / dx;
00371
00372     if (x1 > x2) {
00373         int temp_x = x1;
00374         x1 = x2;
00375         x2 = temp_x;
00376     }
00377     for (x = x1; x < x2; x++) {
00378         int sy1 = dy * x / dx + b;
00379         int sy2 = dy * (x + 1) / dx + b;
00380         if (sy1 > sy2) {
00381             int temp_y = sy1;
00382             sy1 = sy2;
00383             sy2 = temp_y;
00384         }
00385         for (y = sy1; y <= sy2; y++) {
00386             adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00387         }
00388     }
00389
00390 }
00391
00403 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)
00404 {
00405     if (len == 0) return;
00406     for (size_t i = 0; i < len-1; i++) {
00407         adl_line_draw(screen_mat, points[i].x, points[i].y, points[i+1].x, points[i+1].y, color,
       offset_zoom_param);
00408     }
00409 }
00410
00423 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)
00424 {
00425     if (len == 0) return;
00426     for (size_t i = 0; i < len-1; i++) {
00427         adl_line_draw(screen_mat, points[i].x, points[i].y, points[i+1].x, points[i+1].y, color,
       offset_zoom_param);
00428     }
00429     adl_line_draw(screen_mat, points[len-1].x, points[len-1].y, points[0].x, points[0].y, color,
       offset_zoom_param);
00430 }
00431
00432
00451 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)
00452 {
00453     Mat2D pe = mat2D_alloc(3, 1);
00454     mat2D_fill(pe, 0);
00455     MAT2D_AT(pe, 0, 0) = xe;
00456     MAT2D_AT(pe, 1, 0) = ye;
00457     Mat2D v1 = mat2D_alloc(3, 1);
00458     mat2D_fill(v1, 0);
00459     Mat2D v2 = mat2D_alloc(3, 1);
00460     mat2D_fill(v2, 0);
00461     Mat2D temp_v = mat2D_alloc(3, 1);
00462     mat2D_fill(temp_v, 0);
00463     Mat2D DCM_p = mat2D_alloc(3, 3);
00464     mat2D_fill(DCM_p, 0);
00465     mat2D_set_rot_mat_z(DCM_p, angle_deg);
00466     Mat2D DCM_m = mat2D_alloc(3, 3);
```

```
00467     mat2D_fill(DCM_m, 0);
00468     mat2D_set_rot_mat_z(DCM_m, -angle_deg);
00469 
00470     int x_center = xs*head_size + xe*(1-head_size);
00471     int y_center = ys*head_size + ye*(1-head_size);
00472 
00473     MAT2D_AT(v1, 0, 0) = x_center;
00474     MAT2D_AT(v1, 1, 0) = y_center;
00475     mat2D_copy(v2, v1);
00476 
00477     /* v1 */
00478     mat2D_copy(temp_v, v1);
00479     mat2D_sub(temp_v, pe);
00480     mat2D_fill(v1, 0);
00481     mat2D_dot(v1, DCM_p, temp_v);
00482     mat2D_add(v1, pe);
00483 
00484     /* v2 */
00485     mat2D_copy(temp_v, v2);
00486     mat2D_sub(temp_v, pe);
00487     mat2D_fill(v2, 0);
00488     mat2D_dot(v2, DCM_m, temp_v);
00489     mat2D_add(v2, pe);
00490 
00491     adl_line_draw(screen_mat, MAT2D_AT(v1, 0, 0), MAT2D_AT(v1, 1, 0), xe, ye, color,
      offset_zoom_param);
00492     adl_line_draw(screen_mat, MAT2D_AT(v2, 0, 0), MAT2D_AT(v2, 1, 0), xe, ye, color,
      offset_zoom_param);
00493     adl_line_draw(screen_mat, xs, ys, xe, ye, color, offset_zoom_param);
00494 
00495     mat2D_free(pe);
00496     mat2D_free(v1);
00497     mat2D_free(v2);
00498     mat2D_free(temp_v);
00499     mat2D_free(DCM_p);
00500     mat2D_free(DCM_m);
00501 }
00502 
00519 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)
00520 {
00521     switch (c)
00522     {
00523     case 'a':
00524     case 'A':
00525         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel/2,
      y_top_left, color, offset_zoom_param);
00526         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00527         adl_line_draw(screen_mat, x_top_left+width_pixel/6, y_top_left+2*hight_pixel/3,
      x_top_left+5*width_pixel/6, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00528         break;
00529     case 'b':
00530     case 'B':
00531         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00532         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+2*width_pixel/3, y_top_left,
      color, offset_zoom_param);
00533         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00534         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+width_pixel, y_top_left+hight_pixel/3, color, offset_zoom_param);
00535         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/3,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00536 
00537         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, x_top_left,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00538 
00539         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00540         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+2*hight_pixel/3,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00541         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00542         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel, x_top_left,
      y_top_left+hight_pixel, color, offset_zoom_param);
00543         break;
00544     case 'c':
00545     case 'C':
00546         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left+width_pixel/3,
      y_top_left, color, offset_zoom_param);
00547         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00548         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00549         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
```

```
00550         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left+hight_pixel, color, offset_zoom_param);
00551         break;
00552     case 'd':
00553     case 'D':
00554         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+2*width_pixel/3, y_top_left,
      color, offset_zoom_param);
00555         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00556         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00557         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00558         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel, x_top_left,
      y_top_left+hight_pixel, color, offset_zoom_param);
00559         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left, y_top_left, color,
      offset_zoom_param);
00560         break;
00561     case 'e':
00562     case 'E':
00563         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left, y_top_left, color,
      offset_zoom_param);
00564         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00565         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00566
00567         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00568         break;
00569     case 'f':
00570     case 'F':
00571         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left, y_top_left, color,
      offset_zoom_param);
00572         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00573
00574         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00575         break;
00576     case 'g':
00577     case 'G':
00578         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00579         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
      y_top_left, color, offset_zoom_param);
00580         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00581         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00582         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00583         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00584         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00585         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel, y_top_left+hight_pixel/2, color, offset_zoom_param);
00586         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/2,
      x_top_left+width_pixel/2, y_top_left+hight_pixel/2, color, offset_zoom_param);
00587         break;
00588     case 'h':
00589     case 'H':
00590         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00591         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00592         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00593         break;
00594     case 'i':
00595     case 'I':
00596         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
      offset_zoom_param);
00597         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00598         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+width_pixel/2,
      y_top_left+hight_pixel, color, offset_zoom_param);
00599         break;
00600     case 'j':
00601     case 'J':
00602         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
      offset_zoom_param);
00603         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+2*width_pixel/3,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00604         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel/2, y_top_left+hight_pixel, color, offset_zoom_param);
```

```
00605         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left+hight_pixel,
      x_top_left+width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00606         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel/6, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00607         break;
00608     case 'k':
00609     case 'K':
00610         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00611         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00612         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
      y_top_left, color, offset_zoom_param);
00613         break;
00614     case 'l':
00615     case 'L':
00616         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00617         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00618         break;
00619     case 'm':
00620     case 'M':
00621         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left, y_top_left, color,
      offset_zoom_param);
00622         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel/2,
      y_top_left+hight_pixel, color, offset_zoom_param);
00623         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left, color, offset_zoom_param);
00624         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00625         break;
00626     case 'n':
00627     case 'N':
00628         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left, y_top_left, color,
      offset_zoom_param);
00629         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00630         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left, color, offset_zoom_param);
00631         break;
00632     case 'o':
00633     case 'O':
00634         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
      y_top_left, color, offset_zoom_param);
00635         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00636         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00637         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00638         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00639         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00640         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel, y_top_left+hight_pixel/6, color, offset_zoom_param);
00641         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00642         break;
00643     case 'p':
00644     case 'P':
00645         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00646         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+2*width_pixel/3, y_top_left,
      color, offset_zoom_param);
00647         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00648         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+width_pixel, y_top_left+hight_pixel/3, color, offset_zoom_param);
00649         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/3,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00650
00651         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, x_top_left,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00652         break;
00653     case 'q':
00654     case 'Q':
00655         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
      y_top_left, color, offset_zoom_param);
00656         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00657         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00658         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00659         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
```

```
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00660         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00661         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel, y_top_left+hight_pixel/6, color, offset_zoom_param);
00662         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00663
00664         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel, y_top_left+hight_pixel, color, offset_zoom_param);
00665         break;
00666     case 'r':
00667     case 'R':
00668         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00669         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+2*width_pixel/3, y_top_left,
      color, offset_zoom_param);
00670         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00671         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+width_pixel, y_top_left+hight_pixel/3, color, offset_zoom_param);
00672         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/3,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00673
00674         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, x_top_left,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00675
00676         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+width_pixel, y_top_left+hight_pixel, color, offset_zoom_param);
00677         break;
00678     case 's':
00679     case 'S':
00680         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00681         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
      y_top_left, color, offset_zoom_param);
00682         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00683
00684         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
      y_top_left+hight_pixel/3, color, offset_zoom_param);
00685         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/3, x_top_left+width_pixel/3,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00686         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00687         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00688
00689         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00690         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00691         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00692         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00693         break;
00694     case 't':
00695     case 'T':
00696         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
      offset_zoom_param);
00697         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+width_pixel/2,
      y_top_left+hight_pixel, color, offset_zoom_param);
00698         break;
00699     case 'u':
00700     case 'U':
00701         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel/6, color,
      offset_zoom_param);
00702         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00703         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00704         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00705         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00706         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel, y_top_left, color, offset_zoom_param);
00707         break;
00708     case 'v':
00709     case 'V':
00710         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel/2,
      y_top_left+hight_pixel, color, offset_zoom_param);
00711         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left, color, offset_zoom_param);
00712         break;
00713     case 'w':
```

```
00714     case 'W':
00715         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00716         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel/2, y_top_left, color, offset_zoom_param);
00717         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+2*width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00718         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left, color, offset_zoom_param);
00719         break;
00720     case 'x':
00721     case 'X':
00722         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00723         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
      y_top_left, color, offset_zoom_param);
00724         break;
00725     case 'y':
00726     case 'Y':
00727         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel/2,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00728         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left+width_pixel/2,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00729         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left+hight_pixel/2,
      x_top_left+width_pixel/2, y_top_left+hight_pixel, color, offset_zoom_param);
00730         break;
00731     case 'z':
00732     case 'Z':
00733         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
      offset_zoom_param);
00734         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00735         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left,
      y_top_left+hight_pixel, color, offset_zoom_param);
00736         break;
00737     case '.':
00738         adl_rectangle_fill_min_max(screen_mat, x_top_left+width_pixel/6, x_top_left+width_pixel/3,
      y_top_left+5*hight_pixel/6, y_top_left+hight_pixel, color, offset_zoom_param);
00739         break;
00740     case ':':
00741         adl_rectangle_fill_min_max(screen_mat, x_top_left+width_pixel/6, x_top_left+width_pixel/3,
      y_top_left+5*hight_pixel/6, y_top_left+hight_pixel, color, offset_zoom_param);
00742         adl_rectangle_fill_min_max(screen_mat, x_top_left+width_pixel/6, x_top_left+width_pixel/3,
      y_top_left, y_top_left+hight_pixel/6, color, offset_zoom_param);
00743         break;
00744     case '0':
00745         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
      y_top_left, color, offset_zoom_param);
00746         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00747         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00748         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00749         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00750         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00751         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel, y_top_left+hight_pixel/6, color, offset_zoom_param);
00752         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00753
00754         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6, x_top_left,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00755         break;
00756     case '1':
00757         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left+width_pixel/2,
      y_top_left, color, offset_zoom_param);
00758         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+width_pixel/2,
      y_top_left+hight_pixel, color, offset_zoom_param);
00759         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00760         break;
00761     case '2':
00762         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left, color, offset_zoom_param);
00763         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left+2*width_pixel/3,
      y_top_left, color, offset_zoom_param);
00764         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00765         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+width_pixel, y_top_left+hight_pixel/3, color, offset_zoom_param);
00766         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/3, x_top_left,
      y_top_left+hight_pixel, color, offset_zoom_param);
00767         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
```

```
00768         break;
00769     case '3':
00770         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left, color, offset_zoom_param);
00771         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left+2*width_pixel/3,
      y_top_left, color, offset_zoom_param);
00772         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00773         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+width_pixel, y_top_left+hight_pixel/3, color, offset_zoom_param);
00774         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/3,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00775
00776         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00777
00778         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00779         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+2*hight_pixel/3,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00780         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00781         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00782         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel, x_top_left,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00783         break;
00784     case '4':
00785         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00786         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left,
      y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00787         adl_line_draw(screen_mat, x_top_left, y_top_left+2*hight_pixel/3, x_top_left+width_pixel,
      y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00788         break;
00789     case '5':
00790         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left, y_top_left, color,
      offset_zoom_param);
00791         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel/2, color,
      offset_zoom_param);
00792
00793         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+2*width_pixel/3,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00794         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00795
00796         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+2*width_pixel/3,
      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,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00798         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00799         break;
00800     case '6':
00801         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00802         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
      y_top_left, color, offset_zoom_param);
00803         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00804
00805         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00806         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00807         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00808         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00809         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00810         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+2*hight_pixel/3,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00811         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00812         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel/2, x_top_left,
      y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00813         break;
00814     case '7':
00815         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
      offset_zoom_param);
00816         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00817         break;
00818     case '8':
00819         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+width_pixel, y_top_left+hight_pixel/3, color, offset_zoom_param);
```

```
00820         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/3,
      x_top_left+width_pixel, y_top_left+hight_pixel/6, color, offset_zoom_param);
00821         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+2*width_pixel/3, y_top_left, color, offset_zoom_param);
00822         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left, x_top_left+width_pixel/3,
      y_top_left, color, offset_zoom_param);
00823         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00824
00825         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
      y_top_left+hight_pixel/3, color, offset_zoom_param);
00826         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/3, x_top_left+width_pixel/3,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00827         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00828         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00829
00830         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel/2, x_top_left,
      y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00831         adl_line_draw(screen_mat, x_top_left, y_top_left+2*hight_pixel/3, x_top_left,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00832         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00833         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00834         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00835         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel, y_top_left+2*hight_pixel/3, color, offset_zoom_param);
00836         break;
00837     case '9':
00838         adl_line_draw(screen_mat, x_top_left, y_top_left+5*hight_pixel/6, x_top_left+width_pixel/3,
      y_top_left+hight_pixel, color, offset_zoom_param);
00839         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left+hight_pixel,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel, color, offset_zoom_param);
00840         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel,
      x_top_left+width_pixel, y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00841         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+5*hight_pixel/6,
      x_top_left+width_pixel, y_top_left+hight_pixel/6, color, offset_zoom_param);
00842         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left+hight_pixel/6,
      x_top_left+2*width_pixel/3, 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/3,
      y_top_left, color, offset_zoom_param);
00844         adl_line_draw(screen_mat, x_top_left+width_pixel/3, y_top_left, x_top_left,
      y_top_left+hight_pixel/6, color, offset_zoom_param);
00845         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/6, x_top_left,
      y_top_left+hight_pixel/3, color, offset_zoom_param);
00846         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/3, 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/3, y_top_left+hight_pixel/2,
      x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2, color, offset_zoom_param);
00848         adl_line_draw(screen_mat, x_top_left+2*width_pixel/3, y_top_left+hight_pixel/2,
      x_top_left+width_pixel, y_top_left+hight_pixel/3, color, offset_zoom_param);
00849         break;
00850     case '-':
00851         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00852         break;
00853     case '+':
00854         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel/2, x_top_left+width_pixel,
      y_top_left+hight_pixel/2, color, offset_zoom_param);
00855         adl_line_draw(screen_mat, x_top_left+width_pixel/2, y_top_left, x_top_left+width_pixel/2,
      y_top_left+hight_pixel, color, offset_zoom_param);
00856         break;
00857     case ' ':
00858         break;
00859     default:
00860         adl_rectangle_draw_min_max(screen_mat, x_top_left, x_top_left+width_pixel, y_top_left,
      y_top_left+hight_pixel, color, offset_zoom_param);
00861         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel,
      y_top_left+hight_pixel, color, offset_zoom_param);
00862         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left+width_pixel,
      y_top_left, color, offset_zoom_param);
00863         break;
00864     }
00865 }
00866
00882 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)
00883 {
00884     int character_width_pixel = hight_pixel/2;
00885     int current_x_top_left = x_top_left;
00886     int character_x_offset = (int)fmaxf(fminf(ADL_MAX_CHARACTER_OFFSET, character_width_pixel / 5),
      ADL_MIN_CHARACTER_OFFSET);
00887
```

```
00888     for (size_t char_index = 0; char_index < len; char_index++) {
00889         adl_character_draw(screen_mat, sentence[char_index], character_width_pixel, hight_pixel,
      current_x_top_left, y_top_left, color, offset_zoom_param);
00890         current_x_top_left += character_width_pixel + character_x_offset;
00891     }
00892
00893 }
00894
00906 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)
00907 {
00908     adl_line_draw(screen_mat, min_x, min_y, max_x, min_y, color, offset_zoom_param);
00909     adl_line_draw(screen_mat, min_x, max_y, max_x, max_y, color, offset_zoom_param);
00910     adl_line_draw(screen_mat, min_x, min_y, min_x, max_y, color, offset_zoom_param);
00911     adl_line_draw(screen_mat, max_x, min_y, max_x, max_y, color, offset_zoom_param);
00912 }
00913
00925 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)
00926 {
00927     for (int y = min_y; y <= max_y; y++) {
00928         adl_line_draw(screen_mat, min_x, y, max_x, y, color, offset_zoom_param);
00929     }
00930 }
00931
00943 void adl_quad_draw(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
      Offset_zoom_param offset_zoom_param)
00944 {
00945     (void)inv_z_buffer;
00946     adl_lines_loop_draw(screen_mat, quad.points, 4, color, offset_zoom_param);
00947 }
00948
00961 void adl_quad_fill(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
      Offset_zoom_param offset_zoom_param)
00962 {
00963     Point p0 = quad.points[0];
00964     Point p1 = quad.points[1];
00965     Point p2 = quad.points[2];
00966     Point p3 = quad.points[3];
00967
00968     int x_min = fminf(p0.x, fminf(p1.x, fminf(p2.x, p3.x)));
00969     int x_max = fmaxf(p0.x, fmaxf(p1.x, fmaxf(p2.x, p3.x)));
00970     int y_min = fminf(p0.y, fminf(p1.y, fminf(p2.y, p3.y)));
00971     int y_max = fmaxf(p0.y, fmaxf(p1.y, fmaxf(p2.y, p3.y)));
00972
00973     if (x_min < 0) x_min = 0;
00974     if (y_min < 0) y_min = 0;
00975     if (x_max >= (int)screen_mat.cols) x_max = (int)screen_mat.cols - 1;
00976     if (y_max >= (int)screen_mat.rows) y_max = (int)screen_mat.rows - 1;
00977
00978     float w = edge_cross_point(p0, p1, p1, p2) + edge_cross_point(p2, p3, p3, p0);
00979     if (fabs(w) < 1e-6) {
00980         // adl_quad_draw(screen_mat, inv_z_buffer, quad, quad.colors[0], offset_zoom_param);
00981         return;
00982     }
00983
00984     float size_p3_to_p0 = sqrt((p0.x - p3.x)*(p0.x - p3.x) + (p0.y - p3.y)*(p0.y - p3.y));
00985     float size_p0_to_p1 = sqrt((p1.x - p0.x)*(p1.x - p0.x) + (p1.y - p0.y)*(p1.y - p0.y));
00986     float size_p1_to_p2 = sqrt((p2.x - p1.x)*(p2.x - p1.x) + (p2.y - p1.y)*(p2.y - p1.y));
00987     float size_p2_to_p3 = sqrt((p3.x - p2.x)*(p3.x - p2.x) + (p3.y - p2.y)*(p3.y - p2.y));
00988
00989     int r, g, b, a;
00990     HexARGB_RGBA_VAR(color, r, g, b, a);
00991     float light_intensity = (quad.light_intensity[0] + quad.light_intensity[1] +
      quad.light_intensity[2] + quad.light_intensity[3]) / 4;
00992     uint8_t base_r = (uint8_t)fmaxf(0, fminf(255, r * light_intensity));
00993     uint8_t base_g = (uint8_t)fmaxf(0, fminf(255, g * light_intensity));
00994     uint8_t base_b = (uint8_t)fmaxf(0, fminf(255, b * light_intensity));
00995
00996     for (int y = y_min; y <= y_max; y++) {
00997         for (int x = x_min; x <= x_max; x++) {
00998             Point p = {.x = x, .y = y, .z = 0};
00999             bool in_01, in_12, in_23, in_30;
01000
01001             in_01 = (edge_cross_point(p0, p1, p0, p) >= 0) != (w < 0);
01002             in_12 = (edge_cross_point(p1, p2, p1, p) >= 0) != (w < 0);
01003             in_23 = (edge_cross_point(p2, p3, p2, p) >= 0) != (w < 0);
01004             in_30 = (edge_cross_point(p3, p0, p3, p) >= 0) != (w < 0);
01005
01006             /* https://www.mn.uio.no/math/english/people/aca/michaelf/papers/mv3d.pdf. */
01007             float size_p_to_p0 = sqrt((p0.x - p.x)*(p0.x - p.x) + (p0.y - p.y)*(p0.y - p.y));
01008             float size_p_to_p1 = sqrt((p1.x - p.x)*(p1.x - p.x) + (p1.y - p.y)*(p1.y - p.y));
01009             float size_p_to_p2 = sqrt((p2.x - p.x)*(p2.x - p.x) + (p2.y - p.y)*(p2.y - p.y));
01010             float size_p_to_p3 = sqrt((p3.x - p.x)*(p3.x - p.x) + (p3.y - p.y)*(p3.y - p.y));
01011
01012             /* tangent of half the angle directly using vector math */
01013             float tan_theta_3_over_2 = size_p3_to_p0 / (size_p_to_p3 + size_p_to_p0);
```

```
01014             float tan_theta_0_over_2 = size_p0_to_p1 / (size_p_to_p0 + size_p_to_p1);
01015             float tan_theta_1_over_2 = size_p1_to_p2 / (size_p_to_p1 + size_p_to_p2);
01016             float tan_theta_2_over_2 = size_p2_to_p3 / (size_p_to_p2 + size_p_to_p3);
01017             float w0 = (tan_theta_3_over_2 + tan_theta_0_over_2) / size_p_to_p0;
01018             float w1 = (tan_theta_0_over_2 + tan_theta_1_over_2) / size_p_to_p1;
01019             float w2 = (tan_theta_1_over_2 + tan_theta_2_over_2) / size_p_to_p2;
01020             float w3 = (tan_theta_2_over_2 + tan_theta_3_over_2) / size_p_to_p3;
01021
01022             float inv_w_tot = 1.0f / (w0 + w1 + w2 + w3);
01023             float alpha = w0 * inv_w_tot;
01024             float beta  = w1 * inv_w_tot;
01025             float gamma = w2 * inv_w_tot;
01026             float delta = w3 * inv_w_tot;
01027
01028             if (in_01 && in_12 && in_23 && in_30) {
01029
01030                 double inv_w = alpha * (1.0f / p0.w) + beta  * (1.0f / p1.w) + gamma * (1.0f / p2.w) +
      delta * (1.0f / p3.w);
01031                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
      p2.w) + delta * (p3.z / p3.w);
01032                 double inv_z = inv_w / z_over_w;
01033
01034                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01035                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(base_r, base_g, base_b, a),
      offset_zoom_param);
01036                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01037                 }
01038             }
01039         }
01040     }
01041 }
01042
01055 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)
01056 {
01057     Point p0 = quad.points[0];
01058     Point p1 = quad.points[1];
01059     Point p2 = quad.points[2];
01060     Point p3 = quad.points[3];
01061
01062     int x_min = fminf(p0.x, fminf(p1.x, fminf(p2.x, p3.x)));
01063     int x_max = fmaxf(p0.x, fmaxf(p1.x, fmaxf(p2.x, p3.x)));
01064     int y_min = fminf(p0.y, fminf(p1.y, fminf(p2.y, p3.y)));
01065     int y_max = fmaxf(p0.y, fmaxf(p1.y, fmaxf(p2.y, p3.y)));
01066
01067     if (x_min < 0) x_min = 0;
01068     if (y_min < 0) y_min = 0;
01069     if (x_max >= (int)screen_mat.cols) x_max = (int)screen_mat.cols - 1;
01070     if (y_max >= (int)screen_mat.rows) y_max = (int)screen_mat.rows - 1;
01071
01072     float w = edge_cross_point(p0, p1, p1, p2) + edge_cross_point(p2, p3, p3, p0);
01073     if (fabs(w) < 1e-6) {
01074         // adl_quad_draw(screen_mat, inv_z_buffer, quad, quad.colors[0], offset_zoom_param);
01075         return;
01076     }
01077
01078     int r, g, b, a;
01079     HexARGB_RGBA_VAR(color, r, g, b, a);
01080
01081     for (int y = y_min; y <= y_max; y++) {
01082         for (int x = x_min; x <= x_max; x++) {
01083             Point p = {.x = x, .y = y, .z = 0};
01084             bool in_01, in_12, in_23, in_30;
01085
01086             in_01 = (edge_cross_point(p0, p1, p0, p) >= 0) != (w < 0);
01087             in_12 = (edge_cross_point(p1, p2, p1, p) >= 0) != (w < 0);
01088             in_23 = (edge_cross_point(p2, p3, p2, p) >= 0) != (w < 0);
01089             in_30 = (edge_cross_point(p3, p0, p3, p) >= 0) != (w < 0);
01090
01091             /* using 'mean value coordinates'
01092              * https://www.mn.uio.no/math/english/people/aca/michaelf/papers/mv3d.pdf. */
01093             float size_p_to_p0 = sqrt((p0.x - p.x)*(p0.x - p.x) + (p0.y - p.y)*(p0.y - p.y));
01094             float size_p_to_p1 = sqrt((p1.x - p.x)*(p1.x - p.x) + (p1.y - p.y)*(p1.y - p.y));
01095             float size_p_to_p2 = sqrt((p2.x - p.x)*(p2.x - p.x) + (p2.y - p.y)*(p2.y - p.y));
01096             float size_p_to_p3 = sqrt((p3.x - p.x)*(p3.x - p.x) + (p3.y - p.y)*(p3.y - p.y));
01097
01098             /* calculating the tangent of half the angle directly using vector math */
01099             float t0 = adl_tan_half_angle(p0, p1, p, size_p_to_p0, size_p_to_p1);
01100             float t1 = adl_tan_half_angle(p1, p2, p, size_p_to_p1, size_p_to_p2);
01101             float t2 = adl_tan_half_angle(p2, p3, p, size_p_to_p2, size_p_to_p3);
01102             float t3 = adl_tan_half_angle(p3, p0, p, size_p_to_p3, size_p_to_p0);
01103
01104             float w0 = (t3 + t0) / size_p_to_p0;
01105             float w1 = (t0 + t1) / size_p_to_p1;
01106             float w2 = (t1 + t2) / size_p_to_p2;
01107             float w3 = (t2 + t3) / size_p_to_p3;
01108
```

```
01109             float inv_w_tot = 1.0f / (w0 + w1 + w2 + w3);
01110             float alpha = w0 * inv_w_tot;
01111             float beta  = w1 * inv_w_tot;
01112             float gamma = w2 * inv_w_tot;
01113             float delta = w3 * inv_w_tot;
01114
01115             if (in_01 && in_12 && in_23 && in_30) {
01116                 float light_intensity = quad.light_intensity[0]*alpha + quad.light_intensity[1]*beta +
      quad.light_intensity[2]*gamma + quad.light_intensity[3]*delta;
01117
01118                 float rf = r * light_intensity;
01119                 float gf = g * light_intensity;
01120                 float bf = b * light_intensity;
01121                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01122                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01123                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01124
01125                 double inv_w = alpha * (1.0f / p0.w) + beta  * (1.0f / p1.w) + gamma * (1.0f / p2.w) +
      delta * (1.0f / p3.w);
01126                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
      p2.w) + delta * (p3.z / p3.w);
01127                 double inv_z = inv_w / z_over_w;
01128
01129                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01130                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, a), offset_zoom_param);
01131                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01132                 }
01133             }
01134         }
01135     }
01136 }
01137
01149 void adl_quad_fill_interpolate_color_mean_value(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad
      quad, Offset_zoom_param offset_zoom_param)
01150 {
01151     Point p0 = quad.points[0];
01152     Point p1 = quad.points[1];
01153     Point p2 = quad.points[2];
01154     Point p3 = quad.points[3];
01155
01156     int x_min = fminf(p0.x, fminf(p1.x, fminf(p2.x, p3.x)));
01157     int x_max = fmaxf(p0.x, fmaxf(p1.x, fmaxf(p2.x, p3.x)));
01158     int y_min = fminf(p0.y, fminf(p1.y, fminf(p2.y, p3.y)));
01159     int y_max = fmaxf(p0.y, fmaxf(p1.y, fmaxf(p2.y, p3.y)));
01160
01161     if (x_min < 0) x_min = 0;
01162     if (y_min < 0) y_min = 0;
01163     if (x_max >= (int)screen_mat.cols) x_max = (int)screen_mat.cols - 1;
01164     if (y_max >= (int)screen_mat.rows) y_max = (int)screen_mat.rows - 1;
01165
01166     float w = edge_cross_point(p0, p1, p1, p2) + edge_cross_point(p2, p3, p3, p0);
01167     if (fabs(w) < 1e-6) {
01168         // adl_quad_draw(screen_mat, inv_z_buffer, quad, quad.colors[0], offset_zoom_param);
01169         return;
01170     }
01171
01172     for (int y = y_min; y <= y_max; y++) {
01173         for (int x = x_min; x <= x_max; x++) {
01174             Point p = {.x = x, .y = y, .z = 0};
01175             bool in_01, in_12, in_23, in_30;
01176
01177             in_01 = (edge_cross_point(p0, p1, p0, p) >= 0) != (w < 0);
01178             in_12 = (edge_cross_point(p1, p2, p1, p) >= 0) != (w < 0);
01179             in_23 = (edge_cross_point(p2, p3, p2, p) >= 0) != (w < 0);
01180             in_30 = (edge_cross_point(p3, p0, p3, p) >= 0) != (w < 0);
01181
01182             /* using 'mean value coordinates'
01183              * https://www.mn.uio.no/math/english/people/aca/michaelf/papers/mv3d.pdf. */
01184             float size_p_to_p0 = sqrt((p0.x - p.x)*(p0.x - p.x) + (p0.y - p.y)*(p0.y - p.y));
01185             float size_p_to_p1 = sqrt((p1.x - p.x)*(p1.x - p.x) + (p1.y - p.y)*(p1.y - p.y));
01186             float size_p_to_p2 = sqrt((p2.x - p.x)*(p2.x - p.x) + (p2.y - p.y)*(p2.y - p.y));
01187             float size_p_to_p3 = sqrt((p3.x - p.x)*(p3.x - p.x) + (p3.y - p.y)*(p3.y - p.y));
01188
01189             /* calculating the tangent of half the angle directly using vector math */
01190             float t0 = adl_tan_half_angle(p0, p1, p, size_p_to_p0, size_p_to_p1);
01191             float t1 = adl_tan_half_angle(p1, p2, p, size_p_to_p1, size_p_to_p2);
01192             float t2 = adl_tan_half_angle(p2, p3, p, size_p_to_p2, size_p_to_p3);
01193             float t3 = adl_tan_half_angle(p3, p0, p, size_p_to_p3, size_p_to_p0);
01194
01195             float w0 = (t3 + t0) / size_p_to_p0;
01196             float w1 = (t0 + t1) / size_p_to_p1;
01197             float w2 = (t1 + t2) / size_p_to_p2;
01198             float w3 = (t2 + t3) / size_p_to_p3;
01199
01200             float inv_w_tot = 1.0f / (w0 + w1 + w2 + w3);
01201             float alpha = w0 * inv_w_tot;
01202             float beta  = w1 * inv_w_tot;
```

```
01203             float gamma = w2 * inv_w_tot;
01204             float delta = w3 * inv_w_tot;
01205 
01206             if (in_01 && in_12 && in_23 && in_30) {
01207                 int r0, g0, b0, a0;
01208                 int r1, g1, b1, a1;
01209                 int r2, g2, b2, a2;
01210                 int r3, g3, b3, a3;
01211                 HexARGB_RGBA_VAR(quad.colors[0], r0, g0, b0, a0);
01212                 HexARGB_RGBA_VAR(quad.colors[1], r1, g1, b1, a1);
01213                 HexARGB_RGBA_VAR(quad.colors[2], r2, g2, b2, a2);
01214                 HexARGB_RGBA_VAR(quad.colors[3], r3, g3, b3, a3);
01215 
01216                 uint8_t current_r = r0*alpha + r1*beta + r2*gamma + r3*delta;
01217                 uint8_t current_g = g0*alpha + g1*beta + g2*gamma + g3*delta;
01218                 uint8_t current_b = b0*alpha + b1*beta + b2*gamma + b3*delta;
01219                 uint8_t current_a = a0*alpha + a1*beta + a2*gamma + a3*delta;
01220 
01221                 float light_intensity = (quad.light_intensity[0] + quad.light_intensity[1] +
      quad.light_intensity[2] + quad.light_intensity[3]) / 4;
01222                 float rf = current_r * light_intensity;
01223                 float gf = current_g * light_intensity;
01224                 float bf = current_b * light_intensity;
01225                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01226                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01227                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01228 
01229                 double inv_w = alpha * (1.0f / p0.w) + beta  * (1.0f / p1.w) + gamma * (1.0f / p2.w) +
      delta * (1.0f / p3.w);
01230                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
      p2.w) + delta * (p3.z / p3.w);
01231                 double inv_z = inv_w / z_over_w;
01232 
01233                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01234                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, current_a),
      offset_zoom_param);
01235                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01236                 }
01237             }
01238         }
01239     }
01240 }
01241 
01253 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)
01254 {
01255     for (size_t i = 0; i < mesh.length; i++) {
01256         Quad quad = mesh.elements[i];
01257         /* Reject invalid quad */
01258         adl_assert_quad_is_valid(quad);
01259 
01260         if (!quad.to_draw) continue;
01261 
01262         adl_quad_draw(screen_mat, inv_z_buffer_mat, quad, color, offset_zoom_param);
01263     }
01264 }
01265 
01277 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)
01278 {
01279     for (size_t i = 0; i < mesh.length; i++) {
01280         Quad quad = mesh.elements[i];
01281         /* Reject invalid quad */
01282         adl_assert_quad_is_valid(quad);
01283 
01284         if (!quad.to_draw) continue;
01285 
01286         // color = rand_double() * 0xFFFFFFFF;
01287 
01288         adl_quad_fill(screen_mat, inv_z_buffer_mat, quad, color, offset_zoom_param);
01289     }
01290 }
01291 
01304 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)
01305 {
01306     for (size_t i = 0; i < mesh.length; i++) {
01307         Quad quad = mesh.elements[i];
01308         /* Reject invalid quad */
01309         adl_assert_quad_is_valid(quad);
01310 
01311         uint8_t a, r, g, b;
01312         HexARGB_RGBA_VAR(color, a, r, g, b);
01313         (void)r;
01314         (void)g;
01315         (void)b;
01316 
```

```
01317         if (!quad.to_draw && a == 255) continue;
01318
01319         adl_quad_fill_interpolate_normal_mean_value(screen_mat, inv_z_buffer_mat, quad, color,
      offset_zoom_param);
01320     }
01321 }
01322
01334 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)
01335 {
01336     for (size_t i = 0; i < mesh.length; i++) {
01337         Quad quad = mesh.elements[i];
01338         /* Reject invalid quad */
01339         adl_assert_quad_is_valid(quad);
01340
01341         if (!quad.to_draw) continue;
01342
01343         adl_quad_fill_interpolate_color_mean_value(screen_mat, inv_z_buffer_mat, quad,
      offset_zoom_param);
01344     }
01345 }
01346
01360 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)
01361 {
01362     for (int dy = -r; dy <= r; dy++) {
01363         for (int dx = -r; dx <= r; dx ++) {
01364             float diff = dx * dx + dy * dy - r * r;
01365             if (diff < 0 && diff > -r*2) {
01366                 adl_point_draw(screen_mat, center_x + dx, center_y + dy, color, offset_zoom_param);
01367             }
01368         }
01369     }
01370 }
01371
01382 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)
01383 {
01384     for (int dy = -r; dy <= r; dy++) {
01385         for (int dx = -r; dx <= r; dx ++) {
01386             float diff = dx * dx + dy * dy - r * r;
01387             if (diff < 0) {
01388                 adl_point_draw(screen_mat, center_x + dx, center_y + dy, color, offset_zoom_param);
01389             }
01390         }
01391     }
01392 }
01393
01402 void adl_tri_draw(Mat2D_uint32 screen_mat, Tri tri, uint32_t color, Offset_zoom_param
      offset_zoom_param)
01403 {
01404     adl_line_draw(screen_mat, tri.points[0].x, tri.points[0].y, tri.points[1].x, tri.points[1].y,
      color, offset_zoom_param);
01405     adl_line_draw(screen_mat, tri.points[1].x, tri.points[1].y, tri.points[2].x, tri.points[2].y,
      color, offset_zoom_param);
01406     adl_line_draw(screen_mat, tri.points[2].x, tri.points[2].y, tri.points[0].x, tri.points[0].y,
      color, offset_zoom_param);
01407
01408     // adl_draw_arrow(screen_mat, tri.points[0].x, tri.points[0].y, tri.points[1].x, tri.points[1].y,
      0.3, 22, color);
01409     // adl_draw_arrow(screen_mat, tri.points[1].x, tri.points[1].y, tri.points[2].x, tri.points[2].y,
      0.3, 22, color);
01410     // adl_draw_arrow(screen_mat, tri.points[2].x, tri.points[2].y, tri.points[0].x, tri.points[0].y,
      0.3, 22, color);
01411 }
01412
01425 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)
01426 {
01427     /* This function follows the rasterizer of 'Pikuma' shown in his YouTube video. You can fine the
      video in this link: https://youtu.be/k5wtuKWmV48. */
01428
01429     Point p0, p1, p2;
01430     p0 = tri.points[0];
01431     p1 = tri.points[1];
01432     p2 = tri.points[2];
01433
01434     /* finding bounding box */
01435     int x_min = fmin(p0.x, fmin(p1.x, p2.x));
01436     int x_max = fmax(p0.x, fmax(p1.x, p2.x));
01437     int y_min = fmin(p0.y, fmin(p1.y, p2.y));
01438     int y_max = fmax(p0.y, fmax(p1.y, p2.y));
01439
01440     /* Clamp to screen bounds */
01441     if (x_min < 0) x_min = 0;
01442     if (y_min < 0) y_min = 0;
01443     if (x_max >= (int)screen_mat.cols) x_max = screen_mat.cols - 1;
```

```
01444     if (y_max >= (int)screen_mat.rows) y_max = screen_mat.rows - 1;
01445
01446     /* draw only outline of the tri if there is no area */
01447     float w = edge_cross_point(p0, p1, p1, p2);
01448     if (fabsf(w) < 1e-6) {
01449         // adl_tri_draw(screen_mat, tri, tri.colors[0], offset_zoom_param);
01450         return;
01451     }
01452     MATRIX2D_ASSERT(fabsf(w) > 1e-6 && "triangle must have area");
01453
01454     /* fill conventions */
01455     int bias0 = is_top_left(p0, p1) ? 0 : -1;
01456     int bias1 = is_top_left(p1, p2) ? 0 : -1;
01457     int bias2 = is_top_left(p2, p0) ? 0 : -1;
01458
01459     for (int y = y_min; y <= y_max; y++) {
01460         for (int x = x_min; x <= x_max; x++) {
01461             Point p = {.x = x, .y = y, .z = 0};
01462
01463             float w0 = edge_cross_point(p0, p1, p0, p) + bias0;
01464             float w1 = edge_cross_point(p1, p2, p1, p) + bias1;
01465             float w2 = edge_cross_point(p2, p0, p2, p) + bias2;
01466
01467             float alpha = fabs(w1 / w);
01468             float beta  = fabs(w2 / w);
01469             float gamma = fabs(w0 / w);
01470
01471             if (w0 * w >= 0 && w1 * w >= 0 &&  w2 * w >= 0) {
01472                 int r, b, g, a;
01473                 HexARGB_RGBA_VAR(color, r, g, b, a);
01474                 float light_intensity = (tri.light_intensity[0] + tri.light_intensity[1] +
    tri.light_intensity[2]) / 3;
01475                 float rf = r * light_intensity;
01476                 float gf = g * light_intensity;
01477                 float bf = b * light_intensity;
01478                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01479                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01480                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01481
01482                 double inv_w = alpha * (1.0 / p0.w) + beta  * (1.0 / p1.w) + gamma * (1.0 / p2.w);
01483                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
    p2.w);
01484                 double inv_z = inv_w / z_over_w;
01485
01486                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01487                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, a), offset_zoom_param);
01488                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01489                 }
01490             }
01491         }
01492     }
01493 }
01494
01506 void adl_tri_fill_Pinedas_rasterizer_interpolate_color(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer,
    Tri tri, Offset_zoom_param offset_zoom_param)
01507 {
01508     /* This function follows the rasterizer of 'Pikuma' shown in his YouTube video. You can fine the
    video in this link: https://youtu.be/k5wtuKWmV48. */
01509     Point p0, p1, p2;
01510     p0 = tri.points[0];
01511     p1 = tri.points[1];
01512     p2 = tri.points[2];
01513
01514     float w = edge_cross_point(p0, p1, p1, p2);
01515     if (fabsf(w) < 1e-6) {
01516         // adl_tri_draw(screen_mat, tri, tri.colors[0], offset_zoom_param);
01517         return;
01518     }
01519     MATRIX2D_ASSERT(w != 0 && "triangle has area");
01520
01521     /* fill conventions */
01522     int bias0 = is_top_left(p0, p1) ? 0 : -1;
01523     int bias1 = is_top_left(p1, p2) ? 0 : -1;
01524     int bias2 = is_top_left(p2, p0) ? 0 : -1;
01525
01526     /* finding bounding box */
01527     int x_min = fmin(p0.x, fmin(p1.x, p2.x));
01528     int x_max = fmax(p0.x, fmax(p1.x, p2.x));
01529     int y_min = fmin(p0.y, fmin(p1.y, p2.y));
01530     int y_max = fmax(p0.y, fmax(p1.y, p2.y));
01531     // printf("xmin: %d, xmax: %d || ymin: %d, ymax: %d\n", x_min, x_max, y_min, y_max);
01532
01533     /* Clamp to screen bounds */
01534     if (x_min < 0) x_min = 0;
01535     if (y_min < 0) y_min = 0;
01536     if (x_max >= (int)screen_mat.cols) x_max = screen_mat.cols - 1;
01537     if (y_max >= (int)screen_mat.rows) y_max = screen_mat.rows - 1;
```

```
01538
01539     for (int y = y_min; y <= y_max; y++) {
01540         for (int x = x_min; x <= x_max; x++) {
01541             Point p = {.x = x, .y = y, .z = 0};
01542
01543             float w0 = edge_cross_point(p0, p1, p0, p) + bias0;
01544             float w1 = edge_cross_point(p1, p2, p1, p) + bias1;
01545             float w2 = edge_cross_point(p2, p0, p2, p) + bias2;
01546
01547             float alpha = fabs(w1 / w);
01548             float beta  = fabs(w2 / w);
01549             float gamma = fabs(w0 / w);
01550
01551             if (w0 * w >= 0 && w1 * w >= 0 &&  w2 * w >= 0) {
01552                 int r0, b0, g0, a0;
01553                 int r1, b1, g1, a1;
01554                 int r2, b2, g2, a2;
01555                 HexARGB_RGBA_VAR(tri.colors[0], r0, g0, b0, a0);
01556                 HexARGB_RGBA_VAR(tri.colors[1], r1, g1, b1, a1);
01557                 HexARGB_RGBA_VAR(tri.colors[2], r2, g2, b2, a2);
01558
01559                 uint8_t current_r = r0*alpha + r1*beta + r2*gamma;
01560                 uint8_t current_g = g0*alpha + g1*beta + g2*gamma;
01561                 uint8_t current_b = b0*alpha + b1*beta + b2*gamma;
01562                 uint8_t current_a = a0*alpha + a1*beta + a2*gamma;
01563
01564                 float light_intensity = (tri.light_intensity[0] + tri.light_intensity[1] +
    tri.light_intensity[2]) / 3;
01565                 float rf = current_r * light_intensity;
01566                 float gf = current_g * light_intensity;
01567                 float bf = current_b * light_intensity;
01568                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01569                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01570                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01571
01572                 double inv_w = alpha * (1.0 / p0.w) + beta  * (1.0 / p1.w) + gamma * (1.0 / p2.w);
01573                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
    p2.w);
01574                 double inv_z = inv_w / z_over_w;
01575
01576                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01577                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, current_a),
    offset_zoom_param);
01578                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01579                 }
01580             }
01581         }
01582     }
01583 }
01584
01597 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)
01598 {
01599     /* This function follows the rasterizer of 'Pikuma' shown in his YouTube video. You can fine the
    video in this link: https://youtu.be/k5wtuKWmV48. */
01600     Point p0, p1, p2;
01601     p0 = tri.points[0];
01602     p1 = tri.points[1];
01603     p2 = tri.points[2];
01604
01605     float w = edge_cross_point(p0, p1, p1, p2);
01606     if (fabsf(w) < 1e-6) {
01607         // adl_tri_draw(screen_mat, tri, tri.colors[0], offset_zoom_param);
01608         return;
01609     }
01610     MATRIX2D_ASSERT(w != 0 && "triangle has area");
01611
01612     /* fill conventions */
01613     int bias0 = is_top_left(p0, p1) ? 0 : -1;
01614     int bias1 = is_top_left(p1, p2) ? 0 : -1;
01615     int bias2 = is_top_left(p2, p0) ? 0 : -1;
01616
01617     /* finding bounding box */
01618     int x_min = fmin(p0.x, fmin(p1.x, p2.x));
01619     int x_max = fmax(p0.x, fmax(p1.x, p2.x));
01620     int y_min = fmin(p0.y, fmin(p1.y, p2.y));
01621     int y_max = fmax(p0.y, fmax(p1.y, p2.y));
01622     // printf("xmin: %d, xmax: %d || ymin: %d, ymax: %d\n", x_min, x_max, y_min, y_max);
01623
01624     /* Clamp to screen bounds */
01625     if (x_min < 0) x_min = 0;
01626     if (y_min < 0) y_min = 0;
01627     if (x_max >= (int)screen_mat.cols) x_max = screen_mat.cols - 1;
01628     if (y_max >= (int)screen_mat.rows) y_max = screen_mat.rows - 1;
01629
01630     int r, b, g, a;
01631     HexARGB_RGBA_VAR(color, r, g, b, a);
```

```
01632
01633     for (int y = y_min; y <= y_max; y++) {
01634         for (int x = x_min; x <= x_max; x++) {
01635             Point p = {.x = x, .y = y, .z = 0};
01636
01637             float w0 = edge_cross_point(p0, p1, p0, p) + bias0;
01638             float w1 = edge_cross_point(p1, p2, p1, p) + bias1;
01639             float w2 = edge_cross_point(p2, p0, p2, p) + bias2;
01640
01641             float alpha = fabs(w1 / w);
01642             float beta  = fabs(w2 / w);
01643             float gamma = fabs(w0 / w);
01644
01645             if (w0 * w >= 0 && w1 * w >= 0 &&  w2 * w >= 0) {
01646
01647                 float light_intensity = tri.light_intensity[0]*alpha + tri.light_intensity[1]*beta +
      tri.light_intensity[2]*gamma;
01648
01649                 float rf = r * light_intensity;
01650                 float gf = g * light_intensity;
01651                 float bf = b * light_intensity;
01652                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01653                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01654                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01655
01656                 double inv_w = alpha * (1.0 / p0.w) + beta  * (1.0 / p1.w) + gamma * (1.0 / p2.w);
01657                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
      p2.w);
01658                 double inv_z = inv_w / z_over_w;
01659
01660                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01661                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, a), offset_zoom_param);
01662                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01663                 }
01664             }
01665         }
01666     }
01667 }
01668
01679 void adl_tri_mesh_draw(Mat2D_uint32 screen_mat, Tri_mesh mesh, uint32_t color, Offset_zoom_param
      offset_zoom_param)
01680 {
01681     for (size_t i = 0; i < mesh.length; i++) {
01682         Tri tri = mesh.elements[i];
01683         if (tri.to_draw) {
01684             // color = rand_double() * 0xFFFFFFFF;
01685             adl_tri_draw(screen_mat, tri, color, offset_zoom_param);
01686         }
01687     }
01688 }
01689
01701 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)
01702 {
01703     for (size_t i = 0; i < mesh.length; i++) {
01704         Tri tri = mesh.elements[i];
01705         /* Reject invalid triangles */
01706         adl_assert_tri_is_valid(tri);
01707
01708         if (!tri.to_draw) continue;
01709
01710         adl_tri_fill_Pinedas_rasterizer(screen_mat, inv_z_buffer_mat, tri, color, offset_zoom_param);
01711     }
01712 }
01713
01725 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)
01726 {
01727     for (size_t i = 0; i < mesh.length; i++) {
01728         Tri tri = mesh.elements[i];
01729         /* Reject invalid triangles */
01730         adl_assert_tri_is_valid(tri);
01731
01732         if (!tri.to_draw) continue;
01733
01734         adl_tri_fill_Pinedas_rasterizer_interpolate_color(screen_mat, inv_z_buffer_mat, tri,
      offset_zoom_param);
01735     }
01736 }
01737
01750 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)
01751 {
01752     for (size_t i = 0; i < mesh.length; i++) {
01753         Tri tri = mesh.elements[i];
01754         /* Reject invalid triangles */
01755         adl_assert_tri_is_valid(tri);
```

```
01756
01757         if (!tri.to_draw) continue;
01758
01759         adl_tri_fill_Pinedas_rasterizer_interpolate_normal(screen_mat, inv_z_buffer_mat, tri, color,
      offset_zoom_param);
01760     }
01761 }
01762
01778 float adl_tan_half_angle(Point vi, Point vj, Point p, float li, float lj)
01779 {
01780     float ax = vi.x - p.x, ay = vi.y - p.y;
01781     float bx = vj.x - p.x, by = vj.y - p.y;
01782     float dot = ax * bx + ay * by;
01783     float cross = ax * by - ay * bx;              // signed 2D cross (scalar)
01784     float denom = dot + li * lj;                   // = |a||b|(1 + cos(alpha))
01785     return fabsf(cross) / fmaxf(1e-20f, denom);    // tan(alpha/2)
01786 }
01787
01798 float adl_linear_map(float s, float min_in, float max_in, float min_out, float max_out)
01799 {
01800     return (min_out + ((s-min_in)*(max_out-min_out))/(max_in-min_in));
01801 }
01802
01818 void adl_quad2tris(Quad quad, Tri *tri1, Tri *tri2, char split_line[])
01819 {
01820     if (!strncmp(split_line, "02", 2)) {
01821         tri1->points[0] = quad.points[0];
01822         tri1->points[1] = quad.points[1];
01823         tri1->points[2] = quad.points[2];
01824         tri1->to_draw = quad.to_draw;
01825         tri1->light_intensity[0] = quad.light_intensity[0];
01826         tri1->light_intensity[1] = quad.light_intensity[1];
01827         tri1->light_intensity[2] = quad.light_intensity[2];
01828         tri1->colors[0] = quad.colors[0];
01829         tri1->colors[1] = quad.colors[1];
01830         tri1->colors[2] = quad.colors[2];
01831
01832         tri2->points[0] = quad.points[2];
01833         tri2->points[1] = quad.points[3];
01834         tri2->points[2] = quad.points[0];
01835         tri2->to_draw = quad.to_draw;
01836         tri1->light_intensity[0] = quad.light_intensity[2];
01837         tri1->light_intensity[1] = quad.light_intensity[3];
01838         tri1->light_intensity[2] = quad.light_intensity[0];
01839         tri2->colors[0] = quad.colors[2];
01840         tri2->colors[1] = quad.colors[3];
01841         tri2->colors[2] = quad.colors[0];
01842     } else if (!strncmp(split_line, "13", 2)) {
01843         tri1->points[0] = quad.points[1];
01844         tri1->points[1] = quad.points[2];
01845         tri1->points[2] = quad.points[3];
01846         tri1->to_draw = quad.to_draw;
01847         tri1->light_intensity[0] = quad.light_intensity[1];
01848         tri1->light_intensity[1] = quad.light_intensity[2];
01849         tri1->light_intensity[2] = quad.light_intensity[3];
01850         tri1->colors[0] = quad.colors[1];
01851         tri1->colors[1] = quad.colors[2];
01852         tri1->colors[2] = quad.colors[3];
01853
01854         tri2->points[0] = quad.points[3];
01855         tri2->points[1] = quad.points[0];
01856         tri2->points[2] = quad.points[1];
01857         tri2->to_draw = quad.to_draw;
01858         tri1->light_intensity[0] = quad.light_intensity[3];
01859         tri1->light_intensity[1] = quad.light_intensity[0];
01860         tri1->light_intensity[2] = quad.light_intensity[1];
01861         tri2->colors[0] = quad.colors[3];
01862         tri2->colors[1] = quad.colors[0];
01863         tri2->colors[2] = quad.colors[1];
01864     }
01865 }
01866
01878 void adl_linear_sRGB_to_okLab(uint32_t hex_ARGB, float *L, float *a, float *b)
01879 {
01880     /* https://bottosson.github.io/posts/oklab/
01881        https://en.wikipedia.org/wiki/Oklab_color_space */
01882     int R_255, G_255, B_255;
01883     HexARGB_RGB_VAR(hex_ARGB, R_255, G_255, B_255);
01884
01885     float R = R_255;
01886     float G = G_255;
01887     float B = B_255;
01888
01889     float l = 0.4122214705f * R + 0.5363325363f * G + 0.0514459929f * B;
01890     float m = 0.2119034982f * R + 0.6806995451f * G + 0.1073969566f * B;
01891     float s = 0.0883024619f * R + 0.2817188376f * G + 0.6299787005f * B;
01892
```

```
01893     float l_ = cbrtf(l);
01894     float m_ = cbrtf(m);
01895     float s_ = cbrtf(s);
01896 
01897     *L = 0.2104542553f * l_ + 0.7936177850f * m_ - 0.0040720468f * s_;
01898     *a = 1.9779984951f * l_ - 2.4285922050f * m_ + 0.4505937099f * s_;
01899     *b = 0.0259040371f * l_ + 0.7827717662f * m_ - 0.8086757660f * s_;
01900 
01901 }
01902 
01913 void adl_okLab_to_linear_sRGB(float L, float a, float b, uint32_t *hex_ARGB)
01914 {
01915     /* https://bottosson.github.io/posts/oklab/
01916        https://en.wikipedia.org/wiki/Oklab_color_space */
01917 
01918     float l_ = L + 0.3963377774f * a + 0.2158037573f * b;
01919     float m_ = L - 0.1055613458f * a - 0.0638541728f * b;
01920     float s_ = L - 0.0894841775f * a - 1.2914855480f * b;
01921 
01922     float l = l_ * l_ * l_;
01923     float m = m_ * m_ * m_;
01924     float s = s_ * s_ * s_;
01925 
01926     float R = + 4.0767416621f * l - 3.3077115913f * m + 0.2309699292f * s;
01927     float G = - 1.2684380046f * l + 2.6097574011f * m - 0.3413193965f * s;
01928     float B = - 0.0041960863f * l - 0.7034186147f * m + 1.7076147010f * s;
01929 
01930     R = fmaxf(fminf(R, 255), 0);
01931     G = fmaxf(fminf(G, 255), 0);
01932     B = fmaxf(fminf(B, 255), 0);
01933 
01934     *hex_ARGB = RGBA_hexARGB(R, G, B, 0xFF);
01935 }
01936 
01945 void adl_linear_sRGB_to_okLch(uint32_t hex_ARGB, float *L, float *c, float *h_deg)
01946 {
01947     float a, b;
01948     adl_linear_sRGB_to_okLab(hex_ARGB, L, &a, &b);
01949 
01950     *c = sqrtf(a * a + b * b);
01951     *h_deg = atan2f(b, a) * 180 / PI;
01952 }
01953 
01964 void adl_okLch_to_linear_sRGB(float L, float c, float h_deg, uint32_t *hex_ARGB)
01965 {
01966     h_deg = fmodf((h_deg + 360), 360);
01967     float a = c * cosf(h_deg * PI / 180);
01968     float b = c * sinf(h_deg * PI / 180);
01969     adl_okLab_to_linear_sRGB(L, a, b, hex_ARGB);
01970 }
01971 
01986 void adl_interpolate_ARGBcolor_on_okLch(uint32_t color1, uint32_t color2, float t, float
       num_of_rotations, uint32_t *color_out)
01987 {
01988     float L_1, c_1, h_1;
01989     float L_2, c_2, h_2;
01990     adl_linear_sRGB_to_okLch(color1, &L_1, &c_1, &h_1);
01991     adl_linear_sRGB_to_okLch(color2, &L_2, &c_2, &h_2);
01992     h_2 = h_2 + 360 * num_of_rotations;
01993 
01994     float L, c, h;
01995     L = L_1 * (1 - t) + L_2 * (t);
01996     c = c_1 * (1 - t) + c_2 * (t);
01997     h = h_1 * (1 - t) + h_2 * (t);
01998     adl_okLch_to_linear_sRGB(L, c, h, color_out);
01999 }
02000 
02014 Figure adl_figure_alloc(size_t rows, size_t cols, Point top_left_position)
02015 {
02016     ADL_ASSERT(rows && cols);
02017     adl_assert_point_is_valid(top_left_position);
02018 
02019     Figure figure = {0};
02020     figure.pixels_mat = mat2D_alloc_uint32(rows, cols);
02021     figure.inv_z_buffer_mat = mat2D_alloc(rows, cols);
02022     memset(figure.inv_z_buffer_mat.elements, 0x0, sizeof(double) * figure.inv_z_buffer_mat.rows *
       figure.inv_z_buffer_mat.cols);
02023     ada_init_array(Curve, figure.src_curve_array);
02024 
02025     figure.top_left_position = top_left_position;
02026 
02027     int max_i    = (int)(figure.pixels_mat.rows);
02028     int max_j    = (int)(figure.pixels_mat.cols);
02029     int offset_i = (int)fminf(figure.pixels_mat.rows * ADL_FIGURE_PADDING_PRECENTAGE / 100.0f,
       ADL_MAX_FIGURE_PADDING);
02030     int offset_j = (int)fminf(figure.pixels_mat.cols * ADL_FIGURE_PADDING_PRECENTAGE / 100.0f,
       ADL_MAX_FIGURE_PADDING);
```

```
02031
02032     figure.min_x_pixel = offset_j;
02033     figure.max_x_pixel = max_j - offset_j;
02034     figure.min_y_pixel = offset_i;
02035     figure.max_y_pixel = max_i - offset_i;
02036
02037     figure.min_x = + FLT_MAX;
02038     figure.max_x = - FLT_MAX;
02039     figure.min_y = + FLT_MAX;
02040     figure.max_y = - FLT_MAX;
02041
02042     figure.offset_zoom_param = ADL_DEFAULT_OFFSET_ZOOM;
02043
02044     return figure;
02045 }
02046
02057 void adl_figure_copy_to_screen(Mat2D_uint32 screen_mat, Figure figure)
02058 {
02059     for (size_t i = 0; i < figure.pixels_mat.rows; i++) {
02060         for (size_t j = 0; j < figure.pixels_mat.cols; j++) {
02061             int offset_i = figure.top_left_position.y;
02062             int offset_j = figure.top_left_position.x;
02063
02064             adl_point_draw(screen_mat, offset_j+j, offset_i+i, MAT2D_AT_UINT32(figure.pixels_mat, i,
      j), (Offset_zoom_param){1,0,0,0,0});
02065         }
02066     }
02067 }
02068
02077 void adl_axis_draw_on_figure(Figure *figure)
02078 {
02079     int max_i    = (int)(figure->pixels_mat.rows);
02080     int max_j    = (int)(figure->pixels_mat.cols);
02081     int offset_i = (int)fmaxf(fminf(figure->pixels_mat.rows * ADL_FIGURE_PADDING_PRECENTAGE / 100.0f,
      ADL_MAX_FIGURE_PADDING), ADL_MIN_FIGURE_PADDING);
02082     int offset_j = (int)fmaxf(fminf(figure->pixels_mat.cols * ADL_FIGURE_PADDING_PRECENTAGE / 100.0f,
      ADL_MAX_FIGURE_PADDING), ADL_MIN_FIGURE_PADDING);
02083
02084     int arrow_head_size_x = (int)fminf(ADL_MAX_HEAD_SIZE, ADL_FIGURE_PADDING_PRECENTAGE / 100.0f *
      (max_j - 2 * offset_j));
02085     int arrow_head_size_y = (int)fminf(ADL_MAX_HEAD_SIZE, ADL_FIGURE_PADDING_PRECENTAGE / 100.0f *
      (max_i - 2 * offset_i));
02086
02087     adl_arrow_draw(figure->pixels_mat, figure->min_x_pixel, figure->max_y_pixel, figure->max_x_pixel,
      figure->max_y_pixel, (float)arrow_head_size_x / (max_j-2*offset_j), ADL_FIGURE_HEAD_ANGLE_DEG,
      ADL_FIGURE_AXIS_COLOR, figure->offset_zoom_param);
02088     adl_arrow_draw(figure->pixels_mat, figure->min_x_pixel, figure->max_y_pixel, figure->min_x_pixel,
      figure->min_y_pixel, (float)arrow_head_size_y / (max_i-2*offset_i), ADL_FIGURE_HEAD_ANGLE_DEG,
      ADL_FIGURE_AXIS_COLOR, figure->offset_zoom_param);
02089     // adl_draw_rectangle_min_max(figure->pixels_mat, figure->min_x_pixel, figure->max_x_pixel,
      figure->min_y_pixel, figure->max_y_pixel, 0);
02090
02091     figure->x_axis_head_size = arrow_head_size_x;
02092     figure->y_axis_head_size = arrow_head_size_y;
02093 }
02094
02103 void adl_max_min_values_draw_on_figure(Figure figure)
02104 {
02105     char x_min_sentence[256];
02106     char x_max_sentence[256];
02107     snprintf(x_min_sentence, 256, "%g", figure.min_x);
02108     snprintf(x_max_sentence, 256, "%g", figure.max_x);
02109
02110     int x_sentence_hight_pixel = (figure.pixels_mat.rows - figure.max_y_pixel -
      ADL_MIN_CHARACTER_OFFSET * 3);
02111     int x_min_char_width_pixel = x_sentence_hight_pixel / 2;
02112     int x_max_char_width_pixel = x_sentence_hight_pixel / 2;
02113
02114     int x_min_sentence_width_pixel = (int)fminf((figure.max_x_pixel - figure.min_x_pixel)/2,
      (x_min_char_width_pixel + ADL_MAX_CHARACTER_OFFSET)*strlen(x_min_sentence));
02115     x_min_char_width_pixel = x_min_sentence_width_pixel / strlen(x_min_sentence) -
      ADL_MIN_CHARACTER_OFFSET;
02116
02117     int x_max_sentence_width_pixel = (int)fminf((figure.max_x_pixel - figure.min_x_pixel)/2,
      (x_max_char_width_pixel + ADL_MAX_CHARACTER_OFFSET)*strlen(x_max_sentence)) -
      figure.x_axis_head_size;
02118     x_max_char_width_pixel = (x_max_sentence_width_pixel + figure.x_axis_head_size) /
      strlen(x_max_sentence) - ADL_MIN_CHARACTER_OFFSET;
02119
02120     int x_min_sentence_hight_pixel = (int)fminf(x_min_char_width_pixel * 2, x_sentence_hight_pixel);
02121     int x_max_sentence_hight_pixel = (int)fminf(x_max_char_width_pixel * 2, x_sentence_hight_pixel);
02122
02123     x_min_sentence_hight_pixel = (int)fminf(x_min_sentence_hight_pixel, x_max_sentence_hight_pixel);
02124     x_max_sentence_hight_pixel = x_min_sentence_hight_pixel;
02125
02126     int x_max_x_top_left = figure.max_x_pixel - strlen(x_max_sentence) * (x_max_sentence_hight_pixel /
      2 + ADL_MIN_CHARACTER_OFFSET) - figure.x_axis_head_size;
```

```
02127
02128     adl_sentence_draw(figure.pixels_mat, x_min_sentence, strlen(x_min_sentence), figure.min_x_pixel,
      figure.max_y_pixel+ADL_MIN_CHARACTER_OFFSET*2, x_min_sentence_hight_pixel, ADL_FIGURE_AXIS_COLOR,
      figure.offset_zoom_param);
02129     adl_sentence_draw(figure.pixels_mat, x_max_sentence, strlen(x_max_sentence), x_max_x_top_left,
      figure.max_y_pixel+ADL_MIN_CHARACTER_OFFSET*2, x_max_sentence_hight_pixel, ADL_FIGURE_AXIS_COLOR,
      figure.offset_zoom_param);
02130
02131     char y_min_sentence[256];
02132     char y_max_sentence[256];
02133     snprintf(y_min_sentence, 256, "%g", figure.min_y);
02134     snprintf(y_max_sentence, 256, "%g", figure.max_y);
02135
02136     int y_sentence_width_pixel = figure.min_x_pixel - ADL_MAX_CHARACTER_OFFSET -
      figure.y_axis_head_size;
02137     int y_max_char_width_pixel = y_sentence_width_pixel;
02138     y_max_char_width_pixel /= strlen(y_max_sentence);
02139     int y_max_sentence_hight_pixel = y_max_char_width_pixel * 2;
02140
02141     int y_min_char_width_pixel = y_sentence_width_pixel;
02142     y_min_char_width_pixel /= strlen(y_min_sentence);
02143     int y_min_sentence_hight_pixel = y_min_char_width_pixel * 2;
02144
02145     y_min_sentence_hight_pixel = (int)fmaxf(fminf(y_min_sentence_hight_pixel,
      y_max_sentence_hight_pixel), 1);
02146     y_max_sentence_hight_pixel = y_min_sentence_hight_pixel;
02147
02148     adl_sentence_draw(figure.pixels_mat, y_max_sentence, strlen(y_max_sentence),
      ADL_MAX_CHARACTER_OFFSET/2, figure.min_y_pixel, y_max_sentence_hight_pixel, ADL_FIGURE_AXIS_COLOR,
      figure.offset_zoom_param);
02149     adl_sentence_draw(figure.pixels_mat, y_min_sentence, strlen(y_min_sentence),
      ADL_MAX_CHARACTER_OFFSET/2, figure.max_y_pixel-y_min_sentence_hight_pixel,
      y_min_sentence_hight_pixel, ADL_FIGURE_AXIS_COLOR, figure.offset_zoom_param);
02150 }
02151
02163 void adl_curve_add_to_figure(Figure *figure, Point *src_points, size_t src_len, uint32_t color)
02164 {
02165     Curve src_points_ada;
02166     ada_init_array(Point, src_points_ada);
02167     src_points_ada.color = color;
02168
02169     for (size_t i = 0; i < src_len; i++) {
02170         Point current_point = src_points[i];
02171         if (current_point.x > figure->max_x) {
02172             figure->max_x = current_point.x;
02173         }
02174         if (current_point.y > figure->max_y) {
02175             figure->max_y = current_point.y;
02176         }
02177         if (current_point.x < figure->min_x) {
02178             figure->min_x = current_point.x;
02179         }
02180         if (current_point.y < figure->min_y) {
02181             figure->min_y = current_point.y;
02182         }
02183         ada_append(Point, src_points_ada, current_point);
02184     }
02185
02186     ada_append(Curve, figure->src_curve_array, src_points_ada);
02187 }
02188
02198 void adl_curves_plot_on_figure(Figure figure)
02199 {
02200     mat2D_fill_uint32(figure.pixels_mat, figure.background_color);
02201     memset(figure.inv_z_buffer_mat.elements, 0x0, sizeof(double) * figure.inv_z_buffer_mat.rows *
      figure.inv_z_buffer_mat.cols);
02202     if (figure.to_draw_axis) adl_axis_draw_on_figure(&figure);
02203
02204     for (size_t curve_index = 0; curve_index < figure.src_curve_array.length; curve_index++) {
02205         size_t src_len = figure.src_curve_array.elements[curve_index].length;
02206         Point *src_points = figure.src_curve_array.elements[curve_index].elements;
02207         for (size_t i = 0; i < src_len-1; i++) {
02208             Point src_start = src_points[i];
02209             Point src_end   = src_points[i+1];
02210             Point des_start = {0};
02211             Point des_end = {0};
02212
02213             des_start.x = adl_linear_map(src_start.x, figure.min_x, figure.max_x, figure.min_x_pixel,
      figure.max_x_pixel);
02214             des_start.y = ((figure.max_y_pixel + figure.min_y_pixel) - adl_linear_map(src_start.y,
      figure.min_y, figure.max_y, figure.min_y_pixel, figure.max_y_pixel));
02215
02216             des_end.x = adl_linear_map(src_end.x, figure.min_x, figure.max_x, figure.min_x_pixel,
      figure.max_x_pixel);
02217             des_end.y = ((figure.max_y_pixel + figure.min_y_pixel) - adl_linear_map(src_end.y,
      figure.min_y, figure.max_y, figure.min_y_pixel, figure.max_y_pixel));
02218
```

```
02219             adl_line_draw(figure.pixels_mat, des_start.x, des_start.y, des_end.x, des_end.y,
      figure.src_curve_array.elements[curve_index].color, figure.offset_zoom_param);
02220         }
02221     }
02222
02223     if (figure.to_draw_max_min_values) adl_max_min_values_draw_on_figure(figure);
02224 }
02225
02226 /* check offset2D. might convert it to a Mat2D */
02227 #define adl_offset2d(i, j, ni) (j) * (ni) + (i)
02247 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)
02248 {
02249     mat2D_fill_uint32(figure.pixels_mat, figure.background_color);
02250     memset(figure.inv_z_buffer_mat.elements, 0x0, sizeof(double) * figure.inv_z_buffer_mat.rows *
      figure.inv_z_buffer_mat.cols);
02251     if (figure.to_draw_axis) adl_axis_draw_on_figure(&figure);
02252
02253     float min_scalar = FLT_MAX;
02254     float max_scalar = FLT_MIN;
02255     for (int i = 0; i < ni; i++) {
02256         for (int j = 0; j < nj; j++) {
02257             float val = scalar_2Dmat[adl_offset2d(i, j, ni)];
02258             if (val > max_scalar) max_scalar = val;
02259             if (val < min_scalar) min_scalar = val;
02260             float current_x = x_2Dmat[adl_offset2d(i, j, ni)];
02261             float current_y = y_2Dmat[adl_offset2d(i, j, ni)];
02262             if (current_x > figure.max_x) {
02263                 figure.max_x = current_x;
02264             }
02265             if (current_y > figure.max_y) {
02266                 figure.max_y = current_y;
02267             }
02268             if (current_x < figure.min_x) {
02269                 figure.min_x = current_x;
02270             }
02271             if (current_y < figure.min_y) {
02272                 figure.min_y = current_y;
02273             }
02274         }
02275     }
02276
02277     float window_w = (float)figure.pixels_mat.cols;
02278     float window_h = (float)figure.pixels_mat.rows;
02279
02280     for (int i = 0; i < ni-1; i++) {
02281         for (int j = 0; j < nj-1; j++) {
02282             Quad quad = {0};
02283             quad.light_intensity[0] = 1;
02284             quad.light_intensity[1] = 1;
02285             quad.light_intensity[2] = 1;
02286             quad.light_intensity[3] = 1;
02287             quad.to_draw = 1;
02288
02289             quad.points[3].x = x_2Dmat[adl_offset2d(i  , j  , ni)];
02290             quad.points[3].y = y_2Dmat[adl_offset2d(i  , j  , ni)];
02291             quad.points[2].x = x_2Dmat[adl_offset2d(i+1, j  , ni)];
02292             quad.points[2].y = y_2Dmat[adl_offset2d(i+1, j  , ni)];
02293             quad.points[1].x = x_2Dmat[adl_offset2d(i+1, j+1, ni)];
02294             quad.points[1].y = y_2Dmat[adl_offset2d(i+1, j+1, ni)];
02295             quad.points[0].x = x_2Dmat[adl_offset2d(i  , j+1, ni)];
02296             quad.points[0].y = y_2Dmat[adl_offset2d(i  , j+1, ni)];
02297
02298             for (int p_index = 0; p_index < 4; p_index++) {
02299                 quad.points[p_index].z = 1;
02300                 quad.points[p_index].w = 1;
02301                 quad.points[p_index].x = adl_linear_map(quad.points[p_index].x, figure.min_x,
      figure.max_x, figure.min_x_pixel, figure.max_x_pixel);
02302                 quad.points[p_index].y = ((figure.max_y_pixel + figure.min_y_pixel) -
      adl_linear_map(quad.points[p_index].y, figure.min_y, figure.max_y, figure.min_y_pixel,
      figure.max_y_pixel));
02303
02304                 adl_offset_zoom_point(quad.points[p_index], window_w, window_h,
      figure.offset_zoom_param);
02305             }
02306
02307             float t3 = adl_linear_map(scalar_2Dmat[adl_offset2d(i  , j  , ni)], min_scalar,
      max_scalar, 0, 1);
02308             float t2 = adl_linear_map(scalar_2Dmat[adl_offset2d(i+1, j  , ni)], min_scalar,
      max_scalar, 0, 1);
02309             float t1 = adl_linear_map(scalar_2Dmat[adl_offset2d(i+1, j+1, ni)], min_scalar,
      max_scalar, 0, 1);
02310             float t0 = adl_linear_map(scalar_2Dmat[adl_offset2d(  i, j+1, ni)], min_scalar,
      max_scalar, 0, 1);
02311
02312             /* https://en.wikipedia.org/wiki/Oklab_color_space */
02313             if (!strcmp(color_scale, "b-c")) {
```

```
02314                uint32_t color = 0, color1 = BLUE_hexARGB, color2 = CYAN_hexARGB;
02315                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02316                quad.colors[0] = color;
02317
02318                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02319                quad.colors[1] = color;
02320
02321                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02322                quad.colors[2] = color;
02323
02324                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02325                quad.colors[3] = color;
02326            } else if (!strcmp(color_scale, "b-g")) {
02327                uint32_t color = 0, color1 = BLUE_hexARGB, color2 = GREEN_hexARGB;
02328                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02329                quad.colors[0] = color;
02330
02331                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02332                quad.colors[1] = color;
02333
02334                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02335                quad.colors[2] = color;
02336
02337                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02338                quad.colors[3] = color;
02339            } else if (!strcmp(color_scale, "b-r")) {
02340                uint32_t color = 0, color1 = BLUE_hexARGB, color2 = RED_hexARGB;
02341                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02342                quad.colors[0] = color;
02343
02344                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02345                quad.colors[1] = color;
02346
02347                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02348                quad.colors[2] = color;
02349
02350                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02351                quad.colors[3] = color;
02352            } else if (!strcmp(color_scale, "b-y")) {
02353                uint32_t color = 0, color1 = BLUE_hexARGB, color2 = YELLOW_hexARGB;
02354                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02355                quad.colors[0] = color;
02356
02357                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02358                quad.colors[1] = color;
02359
02360                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02361                quad.colors[2] = color;
02362
02363                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02364                quad.colors[3] = color;
02365            } else if (!strcmp(color_scale, "g-y")) {
02366                uint32_t color = 0, color1 = GREEN_hexARGB, color2 = YELLOW_hexARGB;
02367                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02368                quad.colors[0] = color;
02369
02370                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02371                quad.colors[1] = color;
02372
02373                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02374                quad.colors[2] = color;
02375
02376                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02377                quad.colors[3] = color;
02378            } else if (!strcmp(color_scale, "g-p")) {
02379                uint32_t color = 0, color1 = GREEN_hexARGB, color2 = PURPLE_hexARGB;
02380                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02381                quad.colors[0] = color;
02382
02383                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02384                quad.colors[1] = color;
02385
02386                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02387                quad.colors[2] = color;
02388
02389                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02390                quad.colors[3] = color;
02391            } else if (!strcmp(color_scale, "g-r")) {
02392                uint32_t color = 0, color1 = GREEN_hexARGB, color2 = RED_hexARGB;
02393                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02394                quad.colors[0] = color;
02395
02396                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02397                quad.colors[1] = color;
02398
02399                adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02400                quad.colors[2] = color;
```

```
02401
02402                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02403                 quad.colors[3] = color;
02404             } else if (!strcmp(color_scale, "r-y")) {
02405                 uint32_t color = 0, color1 = RED_hexARGB, color2 = YELLOW_hexARGB;
02406                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02407                 quad.colors[0] = color;
02408
02409                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02410                 quad.colors[1] = color;
02411
02412                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02413                 quad.colors[2] = color;
02414
02415                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02416                 quad.colors[3] = color;
02417             }
02418
02419             adl_quad_fill_interpolate_color_mean_value(figure.pixels_mat, figure.inv_z_buffer_mat,
      quad, ADL_DEFAULT_OFFSET_ZOOM);
02420         }
02421     }
02422
02423     if (figure.to_draw_max_min_values) {
02424         adl_max_min_values_draw_on_figure(figure);
02425     }
02426
02427 }
02428
02446 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)
02447 {
02448     Grid grid;
02449     ada_init_array(Curve, grid.curves);
02450
02451     grid.min_e1 = min_e1;
02452     grid.max_e1 = max_e1;
02453     grid.min_e2 = min_e2;
02454     grid.max_e2 = max_e2;
02455     grid.num_samples_e1 = num_samples_e1;
02456     grid.num_samples_e2 = num_samples_e2;
02457     strncpy(grid.plane, plane, 2);
02458
02459     float del_e1 = (max_e1 - min_e1) / num_samples_e1;
02460     float del_e2 = (max_e2 - min_e2) / num_samples_e2;
02461
02462     grid.de1 = del_e1;
02463     grid.de2 = del_e2;
02464
02465     if (!strncmp(plane, "XY", 3) || !strncmp(plane, "xy", 3)) {
02466         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02467             Curve curve;
02468             ada_init_array(Point, curve);
02469             Point point_max = {0}, point_min = {0};
02470
02471             point_min.x = min_e1 + e1_index * del_e1;
02472             point_min.y = min_e2;
02473             point_min.z = third_direction_position;
02474             point_min.w = 1;
02475
02476             point_max.x = min_e1 + e1_index * del_e1;
02477             point_max.y = max_e2;
02478             point_max.z = third_direction_position;
02479             point_max.w = 1;
02480
02481             ada_append(Point, curve, point_min);
02482             ada_append(Point, curve, point_max);
02483
02484             ada_append(Curve, grid.curves, curve);
02485         }
02486         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02487             Curve curve;
02488             ada_init_array(Point, curve);
02489             Point point_max = {0}, point_min = {0};
02490
02491             point_min.x = min_e1;
02492             point_min.y = min_e2 + e2_index * del_e2;
02493             point_min.z = third_direction_position;
02494             point_min.w = 1;
02495
02496             point_max.x = max_e1;
02497             point_max.y = min_e2 + e2_index * del_e2;
02498             point_max.z = third_direction_position;
02499             point_max.w = 1;
02500
02501             ada_append(Point, curve, point_min);
02502             ada_append(Point, curve, point_max);
```

```
02503
02504             ada_append(Curve, grid.curves, curve);
02505         }
02506     } else if (!strncmp(plane, "XZ", 3) || !strncmp(plane, "xz", 3)) {
02507         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02508             Curve curve;
02509             ada_init_array(Point, curve);
02510             Point point_max = {0}, point_min = {0};
02511
02512             point_min.x = min_e1 + e1_index * del_e1;
02513             point_min.y = third_direction_position;
02514             point_min.z = min_e2;
02515             point_min.w = 1;
02516
02517             point_max.x = min_e1 + e1_index * del_e1;
02518             point_max.y = third_direction_position;
02519             point_max.z = max_e2;
02520             point_max.w = 1;
02521
02522             ada_append(Point, curve, point_min);
02523             ada_append(Point, curve, point_max);
02524
02525             ada_append(Curve, grid.curves, curve);
02526         }
02527         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02528             Curve curve;
02529             ada_init_array(Point, curve);
02530             Point point_max = {0}, point_min = {0};
02531
02532             point_min.x = min_e1;
02533             point_min.y = third_direction_position;
02534             point_min.z = min_e2 + e2_index * del_e2;
02535             point_min.w = 1;
02536
02537             point_max.x = max_e1;
02538             point_max.y = third_direction_position;
02539             point_max.z = min_e2 + e2_index * del_e2;
02540             point_max.w = 1;
02541
02542             ada_append(Point, curve, point_min);
02543             ada_append(Point, curve, point_max);
02544
02545             ada_append(Curve, grid.curves, curve);
02546         }
02547     } else if (!strncmp(plane, "YX", 3) || !strncmp(plane, "yx", 3)) {
02548         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02549             Curve curve;
02550             ada_init_array(Point, curve);
02551             Point point_max = {0}, point_min = {0};
02552
02553             point_min.x = min_e2;
02554             point_min.y = min_e1 + e1_index * del_e1;
02555             point_min.z = third_direction_position;
02556             point_min.w = 1;
02557
02558             point_max.x = max_e2;
02559             point_max.y = min_e1 + e1_index * del_e1;
02560             point_max.z = third_direction_position;
02561             point_max.w = 1;
02562
02563             ada_append(Point, curve, point_min);
02564             ada_append(Point, curve, point_max);
02565
02566             ada_append(Curve, grid.curves, curve);
02567         }
02568         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02569             Curve curve;
02570             ada_init_array(Point, curve);
02571             Point point_max = {0}, point_min = {0};
02572
02573             point_min.x = min_e2 + e2_index * del_e2;
02574             point_min.y = min_e1;
02575             point_min.z = third_direction_position;
02576             point_min.w = 1;
02577
02578             point_max.x = min_e2 + e2_index * del_e2;
02579             point_max.y = max_e1;
02580             point_max.z = third_direction_position;
02581             point_max.w = 1;
02582
02583             ada_append(Point, curve, point_min);
02584             ada_append(Point, curve, point_max);
02585
02586             ada_append(Curve, grid.curves, curve);
02587         }
02588     } else if (!strncmp(plane, "YZ", 3) || !strncmp(plane, "yz", 3)) {
02589         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
```

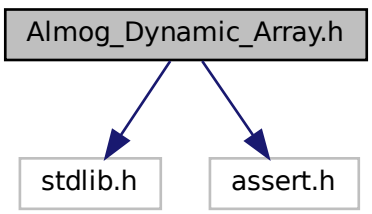
```
02590             Curve curve;
02591             ada_init_array(Point, curve);
02592             Point point_max = {0}, point_min = {0};
02593 
02594             point_min.x = third_direction_position;
02595             point_min.y = min_e1 + e1_index * del_e1;
02596             point_min.z = min_e2;
02597             point_min.w = 1;
02598 
02599             point_max.x = third_direction_position;
02600             point_max.y = min_e1 + e1_index * del_e1;
02601             point_max.z = max_e2;
02602             point_max.w = 1;
02603 
02604             ada_append(Point, curve, point_min);
02605             ada_append(Point, curve, point_max);
02606 
02607             ada_append(Curve, grid.curves, curve);
02608         }
02609         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02610             Curve curve;
02611             ada_init_array(Point, curve);
02612             Point point_max = {0}, point_min = {0};
02613 
02614             point_min.x = third_direction_position;
02615             point_min.y = min_e1;
02616             point_min.z = min_e2 + e2_index * del_e2;
02617             point_min.w = 1;
02618 
02619             point_max.x = third_direction_position;
02620             point_max.y = max_e1;
02621             point_max.z = min_e2 + e2_index * del_e2;
02622             point_max.w = 1;
02623 
02624             ada_append(Point, curve, point_min);
02625             ada_append(Point, curve, point_max);
02626 
02627             ada_append(Curve, grid.curves, curve);
02628         }
02629     } else if (!strncmp(plane, "ZX", 3) || !strncmp(plane, "zx", 3)) {
02630         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02631             Curve curve;
02632             ada_init_array(Point, curve);
02633             Point point_max = {0}, point_min = {0};
02634 
02635             point_min.x = min_e2;
02636             point_min.y = third_direction_position;
02637             point_min.z = min_e1 + e1_index * del_e1;
02638             point_min.w = 1;
02639 
02640             point_max.x = max_e2;
02641             point_max.y = third_direction_position;
02642             point_max.z = min_e1 + e1_index * del_e1;
02643             point_max.w = 1;
02644 
02645             ada_append(Point, curve, point_min);
02646             ada_append(Point, curve, point_max);
02647 
02648             ada_append(Curve, grid.curves, curve);
02649         }
02650         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02651             Curve curve;
02652             ada_init_array(Point, curve);
02653             Point point_max = {0}, point_min = {0};
02654 
02655             point_min.x = min_e2 + e2_index * del_e2;
02656             point_min.y = third_direction_position;
02657             point_min.z = min_e1;
02658             point_min.w = 1;
02659 
02660             point_max.x = min_e2 + e2_index * del_e2;
02661             point_max.y = third_direction_position;
02662             point_max.z = max_e1;
02663             point_max.w = 1;
02664 
02665             ada_append(Point, curve, point_min);
02666             ada_append(Point, curve, point_max);
02667 
02668             ada_append(Curve, grid.curves, curve);
02669         }
02670     } else if (!strncmp(plane, "ZY", 3) || !strncmp(plane, "zy", 3)) {
02671         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02672             Curve curve;
02673             ada_init_array(Point, curve);
02674             Point point_max = {0}, point_min = {0};
02675 
02676             point_min.x = third_direction_position;
```

```
02677             point_min.y = min_e2;
02678             point_min.z = min_e1 + e1_index * del_e1;
02679             point_min.w = 1;
02680 
02681             point_max.x = third_direction_position;
02682             point_max.y = max_e2;
02683             point_max.z = min_e1 + e1_index * del_e1;
02684             point_max.w = 1;
02685 
02686             ada_append(Point, curve, point_min);
02687             ada_append(Point, curve, point_max);
02688 
02689             ada_append(Curve, grid.curves, curve);
02690         }
02691         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02692             Curve curve;
02693             ada_init_array(Point, curve);
02694             Point point_max = {0}, point_min = {0};
02695 
02696             point_min.x = third_direction_position;
02697             point_min.y = min_e2 + e2_index * del_e2;
02698             point_min.z = min_e1;
02699             point_min.w = 1;
02700 
02701             point_max.x = third_direction_position;
02702             point_max.y = min_e2 + e2_index * del_e2;
02703             point_max.z = max_e1;
02704             point_max.w = 1;
02705 
02706             ada_append(Point, curve, point_min);
02707             ada_append(Point, curve, point_max);
02708 
02709             ada_append(Curve, grid.curves, curve);
02710         }
02711     }
02712 
02713     return grid;
02714 }
02715 
02724 void adl_grid_draw(Mat2D_uint32 screen_mat, Grid grid, uint32_t color, Offset_zoom_param
       offset_zoom_param)
02725 {
02726     for (size_t curve_index = 0; curve_index < grid.curves.length; curve_index++) {
02727         adl_lines_draw(screen_mat, grid.curves.elements[curve_index].elements,
       grid.curves.elements[curve_index].length, color, offset_zoom_param);
02728     }
02729 }
02730 
02731 #endif /*ALMOG_DRAW_LIBRARY_IMPLEMENTATION*/
```

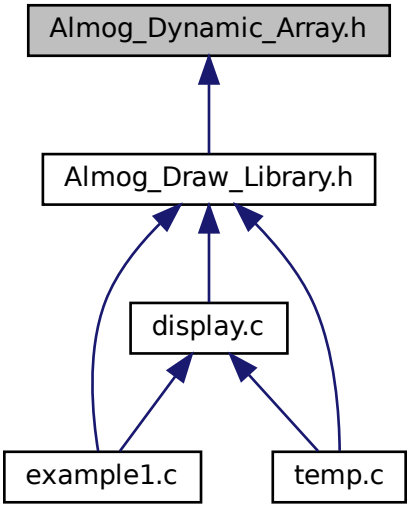
4.3 Almog_Dynamic_Array.h File Reference

```
#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 INIT_CAPACITY 10
- #define ADA_MALLOC malloc
- #define ADA_REALLOC realloc
- #define ADA_ASSERT assert
- #define ada_init_array(type, header)
- #define ada_resize(type, header, new_capacity)
- #define ada_append(type, header, value)
- #define ada_insert(type, header, value, index)
- #define ada_insert_unordered(type, header, value, index)
- #define ada_remove(type, header, index)
- #define ada_remove_unordered(type, header, index)

4.3.1 Macro Definition Documentation

4.3.1.1 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)                                                     \
```

Definition at line 44 of file Almog_Dynamic_Array.h.

4.3.1.2 ADA_ASSERT

```
#define ADA_ASSERT assert
```

Definition at line 18 of file Almog_Dynamic_Array.h.

4.3.1.3 ada_init_array

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

Value:

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

Definition at line 27 of file Almog_Dynamic_Array.h.

4.3.1.4 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)
        \
```

Definition at line 52 of file Almog_Dynamic_Array.h.

4.3.1.5 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_append(type, header, value);                            \
    } else {                                                        \
        ada_append(type, header, header.elements[(index)]);         \
        header.elements[(index)] = value;                           \
    }                                                               \
} while (0)
```

Definition at line 62 of file Almog_Dynamic_Array.h.

4.3.1.6 ADA_MALLOC

```
#define ADA_MALLOC malloc
```

Definition at line 10 of file Almog_Dynamic_Array.h.

4.3.1.7 ADA_REALLOC

```
#define ADA_REALLOC realloc
```

Definition at line 14 of file Almog_Dynamic_Array.h.

4.3.1.8 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)
```

Definition at line 73 of file Almog_Dynamic_Array.h.

4.3.1.9 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)
```

Definition at line 82 of file Almog_Dynamic_Array.h.

4.3.1.10 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)
       \
```

Definition at line 34 of file Almog_Dynamic_Array.h.

4.3.1.11 INIT_CAPACITY

```
#define INIT_CAPACITY 10
```

Definition at line 7 of file Almog_Dynamic_Array.h.

4.4 Almog_Dynamic_Array.h

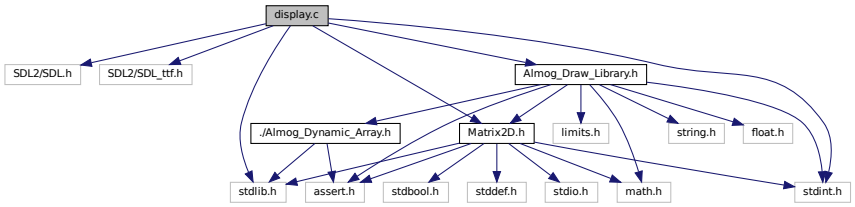
```
00001 #ifndef ALMOG_DYNAMIC_ARRAY_H_
00002 #define ALMOG_DYNAMIC_ARRAY_H_
00003
00004 #include <stdlib.h>
00005 #include <assert.h>
00006
00007 #define INIT_CAPACITY 10
00008
00009 #ifndef ADA_MALLOC
00010 #define ADA_MALLOC malloc
00011 #endif /*ADA_MALLOC*/
00012
00013 #ifndef ADA_REALLOC
00014 #define ADA_REALLOC realloc
00015 #endif /*ADA_REALLOC*/
00016
00017 #ifndef ADA_ASSERT
00018 #define ADA_ASSERT assert
00019 #endif /*ADA_ASSERT*/
00020
00021 /* typedef struct {
00022     size_t length;
00023     size_t capacity;
00024     int* elements;
00025 } ada_int_array; */
00026
00027 #define ada_init_array(type, header) do {                                    \
00028         header.capacity = INIT_CAPACITY;                                     \
00029         header.length = 0;                                                   \
00030         header.elements = (type *)ADA_MALLOC(sizeof(type) * header.capacity);   \
00031         ADA_ASSERT(header.elements != NULL);                                 \
00032     } while (0)                                                              \
00033
00034 #define ada_resize(type, header, new_capacity) do {
        \
00035         type *ada_temp_pointer = (type *)ADA_REALLOC((void *)(header.elements),
       new_capacity*sizeof(type)); \
00036         if (ada_temp_pointer == NULL) {
        \
00037             exit(1);
        \
00038         }
        \
00039         header.elements = ada_temp_pointer;
        \
00040         ADA_ASSERT(header.elements != NULL);
        \
00041         header.capacity = new_capacity;
        \
00042     } while (0)
        \
00043
00044 #define ada_append(type, header, value) do {                         \
00045         if (header.length >= header.capacity) {                      \
00046             ada_resize(type, header, (int)(header.capacity*1.5));    \
00047         }                                                            \
00048         header.elements[header.length] = value;                      \
00049         header.length++;                                             \
00050     } while (0)                                                      \
00051
00052 #define ada_insert(type, header, value, index) do {
        \
00053     ADA_ASSERT((int)(index) >= 0);
        \
00054     ADA_ASSERT((float)(index) - (int)(index) == 0);
        \
00055     ada_append(type, header, header.elements[header.length-1]);
        \
00056     for (size_t ada_for_loop_index = header.length-2; ada_for_loop_index > (index);
       ada_for_loop_index--) { \
00057         header.elements[ada_for_loop_index] = header.elements [ada_for_loop_index-1];
        \
00058     }
        \
00059     header.elements[(index)] = value;
        \
00060 } while (0)
        \
00061
00062 #define ada_insert_unordered(type, header, value, index) do {    \
00063     ADA_ASSERT((int)(index) >= 0);                               \
00064     ADA_ASSERT((float)(index) - (int)(index) == 0);              \
00065     if ((size_t)(index) == header.length) {                      \
00066         ada_append(type, header, value);                         \
00067     } else {                                                     \
```

```
00068         ada_append(type, header, header.elements[(index)]);      \
00069         header.elements[(index)] = value;                        \
00070     }                                                            \
00071 } while (0)
00072
00073 #define ada_remove(type, header, index) do {
        \
00074     ADA_ASSERT((int)(index) >= 0);
        \
00075     ADA_ASSERT((float)(index) - (int)(index) == 0);
        \
00076     for (size_t ada_for_loop_index = (index); ada_for_loop_index < header.length-1;
      ada_for_loop_index++) { \
00077         header.elements[ada_for_loop_index] = header.elements[ada_for_loop_index+1];
        \
00078     }
        \
00079     header.length--;
        \
00080 } while (0)
00081
00082 #define ada_remove_unordered(type, header, index) do {           \
00083     ADA_ASSERT((int)(index) >= 0);                               \
00084     ADA_ASSERT((float)(index) - (int)(index) == 0);              \
00085     header.elements[index] = header.elements[header.length-1];   \
00086     header.length--;                                             \
00087 } while (0)
00088
00089 #endif /*ALMOG_DYNAMIC_ARRAY_H_*/
```

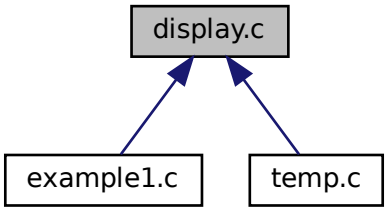
4.5 display.c File Reference

```
#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_multipl

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         if (game_state.to_render) {
00132             render_window(&game_state);
00133         }
00134 
00135     }
00136     destroy_window(&game_state);
00137 
00138     return 0;
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;
```

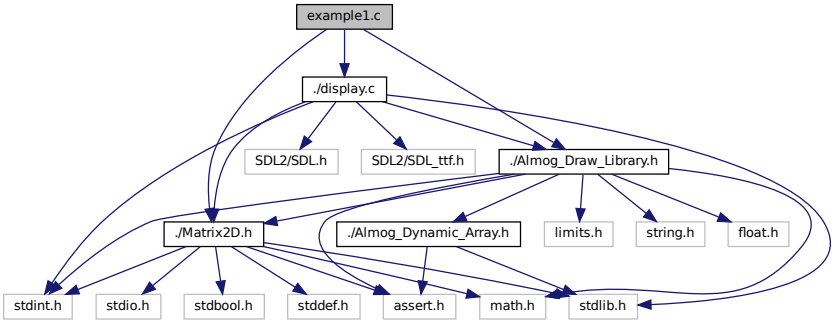
```
00218                         game_state->previous_frame_time = SDL_GetTicks();
00219                         game_state->space_bar_was_pressed = 0;
00220                         break;
00221                     }
00222                 }
00223                 if (event.key.keysym.sym == SDLK_w) {
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 =
      fminf(game_state->offset_zoom_param.zoom_multiplier, ADL_MAX_ZOOM);
00238                 }
00239                 if (event.key.keysym.sym == SDLK_q) {
00240                     game_state->offset_zoom_param.zoom_multiplier -=
      0.1*game_state->offset_zoom_param.zoom_multiplier;
00241                     game_state->offset_zoom_param.zoom_multiplier =
      fminf(game_state->offset_zoom_param.zoom_multiplier, ADL_MAX_ZOOM);
00242                 }
00243                 if (event.key.keysym.sym == SDLK_r) {
00244                     game_state->offset_zoom_param.zoom_multiplier = 1;
00245                     game_state->offset_zoom_param.offset_x = 0;
00246                     game_state->offset_zoom_param.offset_y = 0;
00247                 }
00248                 break;
00249             case SDL_MOUSEBUTTONDOWN:
00250                 if (event.button.button == SDL_BUTTON_LEFT) {
00251                     game_state->left_button_pressed = 1;
00252                 }
00253                 break;
00254             case SDL_MOUSEBUTTONUP:
00255                 if (event.button.button == SDL_BUTTON_LEFT) {
00256                     game_state->left_button_pressed = 0;
00257                 }
00258                 break;
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) *
      game_state->window_pixels_mat.rows * game_state->window_pixels_mat.cols);
00298         /* not using mat2D_fill but using memset because it is way faster, so the buffer needs to be
      of 1/z */
00299         memset(game_state->inv_z_buffer_mat.elements, 0x0, sizeof(double) *
      game_state->inv_z_buffer_mat.rows * game_state->inv_z_buffer_mat.cols);
00300     }
00301     /*------------------------------------------------------------------------*/
00302
00303     render(game_state);
00304
00305     /*------------------------------------------------------------------------*/
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 !=
      (int)game_state->window_pixels_mat.cols) {
00358         mat2D_free_uint32(game_state->window_pixels_mat);
00359         mat2D_free(game_state->inv_z_buffer_mat);
00360         SDL_FreeSurface(game_state->window_surface);
00361
00362         game_state->window_pixels_mat = mat2D_alloc_uint32(game_state->window_h,
      game_state->window_w);
00363         game_state->inv_z_buffer_mat = mat2D_alloc(game_state->window_h, game_state->window_w);
00364
00365         game_state->window_surface = SDL_GetWindowSurface(game_state->window);
00366     }
00367 }
00368
00369 void copy_mat_to_surface_RGB(game_state_t *game_state)
00370 {
00371     SDL_LockSurface(game_state->window_surface);
00372
00373     memcpy(game_state->window_surface->pixels, game_state->window_pixels_mat.elements,
      sizeof(uint32_t) * game_state->window_pixels_mat.rows * game_state->window_pixels_mat.cols);
00374
00375     SDL_UnlockSurface(game_state->window_surface);
00376 }
```

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_append, 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_append(Point, points, temp_point);
00027     ada_append(Point, points1, temp_point);
00028     temp_point = (Point){2,2,0,0};
00029     ada_append(Point, points, temp_point);
00030     ada_append(Point, points1, temp_point);
00031     temp_point = (Point){3,1,0,0};
00032     ada_append(Point, points, temp_point);
00033     ada_append(Point, points1, temp_point);
00034     temp_point = (Point){4,10,0,0};
00035     ada_append(Point, points, temp_point);
```

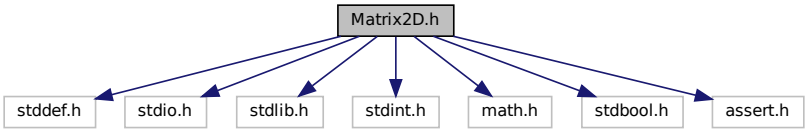
```
00036     temp_point = (Point){5,-10,0,0};
00037     ada_append(Point, points, temp_point);
00038     temp_point = (Point){3,-20,0,0};
00039     ada_append(Point, points, temp_point);
00040 
00041     temp_point = (Point){3.5,-10,0,0};
00042     ada_append(Point, points1, temp_point);
00043 
00044     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 
00073     adl_character_draw(game_state->window_pixels_mat, 'A', 50, 100, 700 , 200, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00074     adl_character_draw(game_state->window_pixels_mat, 'B', 50, 100, 755 , 200, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00075     adl_character_draw(game_state->window_pixels_mat, 'C', 50, 100, 810 , 200, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00076     adl_character_draw(game_state->window_pixels_mat, 'D', 50, 100, 865 , 200, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00077     adl_character_draw(game_state->window_pixels_mat, 'E', 50, 100, 920 , 200, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00078     adl_character_draw(game_state->window_pixels_mat, 'F', 50, 100, 975 , 200, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00079     adl_character_draw(game_state->window_pixels_mat, 'G', 50, 100, 1030, 200, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00080     adl_character_draw(game_state->window_pixels_mat, 'H', 50, 100, 1085, 200, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00081     adl_character_draw(game_state->window_pixels_mat, 'I', 50, 100, 1140, 200, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00082     adl_character_draw(game_state->window_pixels_mat, 'J', 50, 100, 1195, 200, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00083     adl_character_draw(game_state->window_pixels_mat, 'K', 50, 100, 700 , 305, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00084     adl_character_draw(game_state->window_pixels_mat, 'L', 50, 100, 755 , 305, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00085     adl_character_draw(game_state->window_pixels_mat, 'M', 50, 100, 810 , 305, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00086     adl_character_draw(game_state->window_pixels_mat, 'N', 50, 100, 865 , 305, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00087     adl_character_draw(game_state->window_pixels_mat, 'O', 50, 100, 920 , 305, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00088     adl_character_draw(game_state->window_pixels_mat, 'P', 50, 100, 975 , 305, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00089     adl_character_draw(game_state->window_pixels_mat, 'Q', 50, 100, 1030, 305, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00090     adl_character_draw(game_state->window_pixels_mat, 'R', 50, 100, 1085, 305, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00091     adl_character_draw(game_state->window_pixels_mat, 'S', 50, 100, 1140, 305, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00092     adl_character_draw(game_state->window_pixels_mat, 'T', 50, 100, 1195, 305, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00093     adl_character_draw(game_state->window_pixels_mat, 'U', 50, 100, 700 , 410, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00094     adl_character_draw(game_state->window_pixels_mat, 'V', 50, 100, 755 , 410, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00095     adl_character_draw(game_state->window_pixels_mat, 'W', 50, 100, 810 , 410, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00096     adl_character_draw(game_state->window_pixels_mat, 'X', 50, 100, 865 , 410, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00097     adl_character_draw(game_state->window_pixels_mat, 'Y', 50, 100, 920 , 410, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
```

```
00098     adl_character_draw(game_state->window_pixels_mat, 'Z', 50, 100, 975 , 410, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00099     adl_character_draw(game_state->window_pixels_mat, '.', 50, 100, 1030, 410, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00100     adl_character_draw(game_state->window_pixels_mat, ':', 50, 100, 1085, 410, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00101     adl_character_draw(game_state->window_pixels_mat, '0', 50, 100, 700 , 515, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00102     adl_character_draw(game_state->window_pixels_mat, '1', 50, 100, 755 , 515, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00103     adl_character_draw(game_state->window_pixels_mat, '2', 50, 100, 810 , 515, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00104     adl_character_draw(game_state->window_pixels_mat, '3', 50, 100, 865 , 515, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00105     adl_character_draw(game_state->window_pixels_mat, '4', 50, 100, 920 , 515, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00106     adl_character_draw(game_state->window_pixels_mat, '5', 50, 100, 975 , 515, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00107     adl_character_draw(game_state->window_pixels_mat, '6', 50, 100, 1030, 515, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00108     adl_character_draw(game_state->window_pixels_mat, '7', 50, 100, 1085, 515, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00109     adl_character_draw(game_state->window_pixels_mat, '8', 50, 100, 1140, 515, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00110     adl_character_draw(game_state->window_pixels_mat, '9', 50, 100, 1195, 515, 0xFFFFFFFF,
      ADL_DEFAULT_OFFSET_ZOOM);
00111
00112 }
00113
```

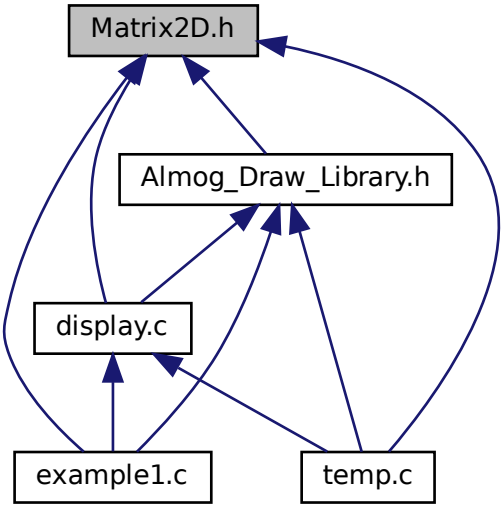
4.9 Matrix2D.h File Reference

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>
#include <stdbool.h>
#include <assert.h>
```

Include dependency graph for Matrix2D.h:



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



Classes

- struct Mat2D
- struct Mat2D_uint32
- struct Mat2D_Minor

Macros

- #define MATRIX2D_MALLOC malloc
- #define MATRIX2D_ASSERT assert
- #define MAT2D_AT(m, i, j) (m).elements[mat2D_offset2d((m), (i), (j))]
- #define MAT2D_AT_UINT32(m, i, j) (m).elements[mat2D_offset2d_uint32((m), (i), (j))]
- #define PI M_PI
- #define MAT2D_MINOR_AT(mm, i, j) MAT2D_AT(mm.ref_mat, mm.rows_list[i], mm.cols_list[j])
- #define MAT2D_PRINT(m) mat2D_print(m, #m, 0)
- #define MAT2D_UINT32_PRINT(m) mat2D_uint32_print(m, #m, 0)
- #define MAT2D_PRINT_AS_COL(m) mat2D_print_as_col(m, #m, 0)
- #define MAT2D_MINOR_PRINT(mm) mat2D_minor_print(mm, #mm, 0)

Functions

- double rand_double (void)
- Mat2D mat2D_alloc (size_t rows, size_t cols)
- Mat2D_uint32 mat2D_alloc_uint32 (size_t rows, size_t cols)
- void mat2D_free (Mat2D m)
- void mat2D_free_uint32 (Mat2D_uint32 m)
- size_t mat2D_offset2d (Mat2D m, size_t i, size_t j)

- size_t mat2D_offset2d_uint32 (Mat2D_uint32 m, size_t i, size_t j)
- void mat2D_fill (Mat2D m, double x)
- void mat2D_fill_uint32 (Mat2D_uint32 m, uint32_t x)
- void mat2D_fill_sequence (Mat2D m, double start, double step)
- void mat2D_rand (Mat2D m, double low, double high)
- void mat2D_dot (Mat2D dst, Mat2D a, Mat2D b)
- void mat2D_cross (Mat2D dst, Mat2D a, Mat2D b)
- void mat2D_add (Mat2D dst, Mat2D a)
- void mat2D_add_row_time_factor_to_row (Mat2D m, size_t des_r, size_t src_r, double factor)
- void mat2D_sub (Mat2D dst, Mat2D a)
- void mat2D_sub_row_time_factor_to_row (Mat2D m, size_t des_r, size_t src_r, double factor)
- void mat2D_mult (Mat2D m, double factor)
- void mat2D_mult_row (Mat2D m, size_t r, double factor)
- void mat2D_print (Mat2D m, const char *name, size_t padding)
- void mat2D_uint32_print (Mat2D_uint32 m, const char *name, size_t padding)
- void mat2D_print_as_col (Mat2D m, const char *name, size_t padding)
- void mat2D_set_identity (Mat2D m)
- double mat2D_make_identity (Mat2D m)
- void mat2D_set_rot_mat_x (Mat2D m, float angle_deg)
- void mat2D_set_rot_mat_y (Mat2D m, float angle_deg)
- void mat2D_set_rot_mat_z (Mat2D m, float angle_deg)
- void mat2D_copy (Mat2D des, Mat2D src)
- void mat2D_copy_mat_to_mat_at_window (Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)
- void mat2D_get_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)
- void mat2D_add_col_to_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)
- void mat2D_sub_col_to_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)
- void mat2D_swap_rows (Mat2D m, size_t r1, size_t r2)
- void mat2D_get_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)
- void mat2D_add_row_to_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)
- void mat2D_sub_row_to_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)
- double mat2D_calc_norma (Mat2D m)
- bool mat2D_mat_is_all_digit (Mat2D m, double digit)
- bool mat2D_row_is_all_digit (Mat2D m, double digit, size_t r)
- bool mat2D_col_is_all_digit (Mat2D m, double digit, size_t c)
- double mat2D_det_2x2_mat (Mat2D m)
- double mat2D_triangulate (Mat2D m)
- double mat2D_det (Mat2D m)
- void mat2D_LUP_decomposition_with_swap (Mat2D src, Mat2D l, Mat2D p, Mat2D u)
- void mat2D_transpose (Mat2D des, Mat2D src)
- void mat2D_invert (Mat2D des, Mat2D src)
- void mat2D_solve_linear_sys_LUP_decomposition (Mat2D A, Mat2D x, Mat2D B)
- Mat2D_Minor mat2D_minor_alloc_fill_from_mat (Mat2D ref_mat, size_t i, size_t j)
- Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor (Mat2D_Minor ref_mm, size_t i, size_t j)
- void mat2D_minor_free (Mat2D_Minor mm)
- void mat2D_minor_print (Mat2D_Minor mm, const char *name, size_t padding)
- double mat2D_det_2x2_mat_minor (Mat2D_Minor mm)
- double mat2D_minor_det (Mat2D_Minor mm)

4.9.1 Macro Definition Documentation

4.9.1.1 MAT2D_AT

```
#define MAT2D_AT(
              m,
              i,
              j ) (m).elements[mat2D_offset2d((m), (i), (j))]
```

Definition at line 54 of file Matrix2D.h.

4.9.1.2 MAT2D_AT_UINT32

```
#define MAT2D_AT_UINT32(
              m,
              i,
              j ) (m).elements[mat2D_offset2d_uint32((m), (i), (j))]
```

Definition at line 55 of file Matrix2D.h.

4.9.1.3 MAT2D_MINOR_AT

```
#define MAT2D_MINOR_AT(
              mm,
              i,
              j ) MAT2D_AT(mm.ref_mat, mm.rows_list[i], mm.cols_list[j])
```

Definition at line 65 of file Matrix2D.h.

4.9.1.4 MAT2D_MINOR_PRINT

```
#define MAT2D_MINOR_PRINT(
              mm ) mat2D_minor_print(mm, #mm, 0)
```

Definition at line 69 of file Matrix2D.h.

4.9.1.5 MAT2D_PRINT

```
#define MAT2D_PRINT(
              m ) mat2D_print(m, #m, 0)
```

Definition at line 66 of file Matrix2D.h.

4.9.1.6 MAT2D_PRINT_AS_COL

```
#define MAT2D_PRINT_AS_COL(
              m ) mat2D_print_as_col(m, #m, 0)
```

Definition at line 68 of file Matrix2D.h.

4.9.1.7 MAT2D_UINT32_PRINT

```
#define MAT2D_UINT32_PRINT(
              m ) mat2D_uint32_print(m, #m, 0)
```

Definition at line 67 of file Matrix2D.h.

4.9.1.8 MATRIX2D_ASSERT

```
#define MATRIX2D_ASSERT assert
```

Definition at line 27 of file Matrix2D.h.

4.9.1.9 MATRIX2D_MALLOC

```
#define MATRIX2D_MALLOC malloc
```

Definition at line 22 of file Matrix2D.h.

4.9.1.10 PI

```
#define PI M_PI
```

Definition at line 62 of file Matrix2D.h.

4.9.2 Function Documentation

4.9.2.1 mat2D_add()

```
void mat2D_add (
            Mat2D dst,
            Mat2D a )
```

Definition at line 258 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by adl_arrow_draw().

4.9.2.2 mat2D_add_col_to_col()

```
void mat2D_add_col_to_col (
            Mat2D des,
            size_t des_col,
            Mat2D src,
            size_t src_col )
```

Definition at line 479 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.9.2.3 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 )
```

Definition at line 269 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

4.9.2.4 mat2D_add_row_to_row()

```
void mat2D_add_row_to_row (
            Mat2D des,
            size_t des_row,
            Mat2D src,
            size_t src_row )
```

Definition at line 521 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.9.2.5 mat2D_alloc()

```
Mat2D mat2D_alloc (
            size_t rows,
            size_t cols )
```

Definition at line 150 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_solve_linear_sys_LUP_decomposition(), and setup_window().

4.9.2.6 mat2D_alloc_uint32()

```
Mat2D_uint32 mat2D_alloc_uint32 (
            size_t rows,
            size_t cols )
```

Definition at line 162 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().

4.9.2.7 mat2D_calc_norma()

```
double mat2D_calc_norma (
            Mat2D m )
```

Definition at line 543 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, and Mat2D::rows.

4.9.2.8 mat2D_col_is_all_digit()

```
bool mat2D_col_is_all_digit (
            Mat2D m,
            double digit,
            size_t c )
```

Definition at line 577 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_det().

4.9.2.9 mat2D_copy()

```
void mat2D_copy (
            Mat2D des,
            Mat2D src )
```

Definition at line 443 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().

4.9.2.10 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 )
```

Definition at line 455 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.9.2.11 mat2D_cross()

```
void mat2D_cross (
            Mat2D dst,
            Mat2D a,
            Mat2D b )
```

Definition at line 247 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.9.2.12 mat2D_det()

```
double mat2D_det (
            Mat2D m )
```

Definition at line 625 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().

4.9.2.13 mat2D_det_2x2_mat()

```
double mat2D_det_2x2_mat (
            Mat2D m )
```

Definition at line 587 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.9.2.14 mat2D_det_2x2_mat_minor()

```
double mat2D_det_2x2_mat_minor (
            Mat2D_Minor mm )
```

Definition at line 893 of file Matrix2D.h.

References Mat2D_Minor::cols, MAT2D_MINOR_AT, MATRIX2D_ASSERT, and Mat2D_Minor::rows.

Referenced by mat2D_minor_det().

4.9.2.15 mat2D_dot()

```
void mat2D_dot (
            Mat2D dst,
            Mat2D a,
            Mat2D b )
```

Definition at line 230 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by adl_arrow_draw(), and mat2D_solve_linear_sys_LUP_decomposition().

4.9.2.16 mat2D_fill()

```
void mat2D_fill (
            Mat2D m,
            double x )
```

Definition at line 196 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

4.9.2.17 mat2D_fill_sequence()

```
void mat2D_fill_sequence (
            Mat2D m,
            double start,
            double step )
```

Definition at line 213 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_offset2d(), and Mat2D::rows.

4.9.2.18 mat2D_fill_uint32()

```
void mat2D_fill_uint32 (
            Mat2D_uint32 m,
            uint32_t x )
```

Definition at line 205 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().

4.9.2.19 mat2D_free()

```
void mat2D_free (
            Mat2D m )
```

Definition at line 174 of file Matrix2D.h.

References Mat2D::elements.

Referenced by adl_arrow_draw(), check_window_mat_size(), mat2D_det(), mat2D_invert(), and mat2D_solve_linear_sys_LUP_decor

4.9.2.20 mat2D_free_uint32()

```
void mat2D_free_uint32 (
            Mat2D_uint32 m )
```

Definition at line 179 of file Matrix2D.h.

References Mat2D_uint32::elements.

Referenced by check_window_mat_size(), and destroy_window().

4.9.2.21 mat2D_get_col()

```
void mat2D_get_col (
            Mat2D des,
            size_t des_col,
            Mat2D src,
            size_t src_col )
```

Definition at line 468 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.9.2.22 mat2D_get_row()

```
void mat2D_get_row (
            Mat2D des,
            size_t des_row,
            Mat2D src,
            size_t src_row )
```

Definition at line 510 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.9.2.23 mat2D_invert()

```
void mat2D_invert (
            Mat2D des,
            Mat2D src )
```

Definition at line 721 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().

4.9.2.24 mat2D_LUP_decomposition_with_swap()

```
void mat2D_LUP_decomposition_with_swap (
            Mat2D src,
            Mat2D l,
            Mat2D p,
            Mat2D u )
```

Definition at line 672 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().

4.9.2.25 mat2D_make_identity()

```
double mat2D_make_identity (
            Mat2D m )
```

Definition at line 366 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.

4.9.2.26 mat2D_mat_is_all_digit()

```
bool mat2D_mat_is_all_digit (
            Mat2D m,
            double digit )
```

Definition at line 555 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, and Mat2D::rows.

4.9.2.27 mat2D_minor_alloc_fill_from_mat()

```
Mat2D_Minor mat2D_minor_alloc_fill_from_mat (
            Mat2D ref_mat,
            size_t i,
            size_t j )
```

Definition at line 814 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.

4.9.2.28 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 )
```

Definition at line 844 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().

4.9.2.29 mat2D_minor_det()

```
double mat2D_minor_det (
            Mat2D_Minor mm )
```

Definition at line 899 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.

4.9.2.30 mat2D_minor_free()

```
void mat2D_minor_free (
            Mat2D_Minor mm )
```

Definition at line 874 of file Matrix2D.h.

References Mat2D_Minor::cols_list, and Mat2D_Minor::rows_list.

Referenced by mat2D_minor_det().

4.9.2.31 mat2D_minor_print()

```
void mat2D_minor_print (
            Mat2D_Minor mm,
            const char * name,
            size_t padding )
```

Definition at line 880 of file Matrix2D.h.

References Mat2D_Minor::cols, MAT2D_MINOR_AT, and Mat2D_Minor::rows.

4.9.2.32 mat2D_mult()

```
void mat2D_mult (
            Mat2D m,
            double factor )
```

Definition at line 294 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, and Mat2D::rows.

4.9.2.33 mat2D_mult_row()

```
void mat2D_mult_row (
            Mat2D m,
            size_t r,
            double factor )
```

Definition at line 303 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_invert(), and mat2D_make_identity().

4.9.2.34 mat2D_offset2d()

```
size_t mat2D_offset2d (
            Mat2D m,
            size_t i,
            size_t j )
```

Definition at line 184 of file Matrix2D.h.

References Mat2D::cols, MATRIX2D_ASSERT, Mat2D::rows, and Mat2D::stride_r.

Referenced by mat2D_fill_sequence().

4.9.2.35 mat2D_offset2d_uint32()

```
size_t mat2D_offset2d_uint32 (
            Mat2D_uint32 m,
            size_t i,
            size_t j )
```

Definition at line 190 of file Matrix2D.h.

References Mat2D_uint32::cols, MATRIX2D_ASSERT, Mat2D_uint32::rows, and Mat2D_uint32::stride_r.

4.9.2.36 mat2D_print()

```
void mat2D_print (
            Mat2D m,
            const char * name,
            size_t padding )
```

Definition at line 310 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, and Mat2D::rows.

4.9.2.37 mat2D_print_as_col()

```
void mat2D_print_as_col (
            Mat2D m,
            const char * name,
            size_t padding )
```

Definition at line 340 of file Matrix2D.h.

References Mat2D::cols, Mat2D::elements, and Mat2D::rows.

4.9.2.38 mat2D_rand()

```
void mat2D_rand (
            Mat2D m,
            double low,
            double high )
```

Definition at line 221 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, rand_double(), and Mat2D::rows.

4.9.2.39 mat2D_row_is_all_digit()

```
bool mat2D_row_is_all_digit (
            Mat2D m,
            double digit,
            size_t r )
```

Definition at line 567 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_det().

4.9.2.40 mat2D_set_identity()

```
void mat2D_set_identity (
            Mat2D m )
```

Definition at line 350 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().

4.9.2.41 mat2D_set_rot_mat_x()

```
void mat2D_set_rot_mat_x (
            Mat2D m,
            float angle_deg )
```

Definition at line 407 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_set_identity(), MATRIX2D_ASSERT, PI, and Mat2D::rows.

4.9.2.42 mat2D_set_rot_mat_y()

```
void mat2D_set_rot_mat_y (
            Mat2D m,
            float angle_deg )
```

Definition at line 419 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_set_identity(), MATRIX2D_ASSERT, PI, and Mat2D::rows.

4.9.2.43 mat2D_set_rot_mat_z()

```
void mat2D_set_rot_mat_z (
            Mat2D m,
            float angle_deg )
```

Definition at line 431 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_set_identity(), MATRIX2D_ASSERT, PI, and Mat2D::rows.

Referenced by adl_arrow_draw().

4.9.2.44 mat2D_solve_linear_sys_LUP_decomposition()

```
void mat2D_solve_linear_sys_LUP_decomposition (
            Mat2D A,
            Mat2D x,
            Mat2D B )
```

Definition at line 779 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.

4.9.2.45 mat2D_sub()

```
void mat2D_sub (
            Mat2D dst,
            Mat2D a )
```

Definition at line 276 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by adl_arrow_draw().

4.9.2.46 mat2D_sub_col_to_col()

```
void mat2D_sub_col_to_col (
            Mat2D des,
            size_t des_col,
            Mat2D src,
            size_t src_col )
```

Definition at line 490 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.9.2.47 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 )
```

Definition at line 287 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().

4.9.2.48 mat2D_sub_row_to_row()

```
void mat2D_sub_row_to_row (
            Mat2D des,
            size_t des_row,
            Mat2D src,
            size_t src_row )
```

Definition at line 532 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.9.2.49 mat2D_swap_rows()

```
void mat2D_swap_rows (
            Mat2D m,
            size_t r1,
            size_t r2 )
```

Definition at line 501 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().

4.9.2.50 mat2D_transpose()

```
void mat2D_transpose (
            Mat2D des,
            Mat2D src )
```

Definition at line 709 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.9.2.51 mat2D_triangulate()

```
double mat2D_triangulate (
            Mat2D m )
```

Definition at line 593 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().

4.9.2.52 mat2D_uint32_print()

```
void mat2D_uint32_print (
            Mat2D_uint32 m,
            const char * name,
            size_t padding )
```

Definition at line 323 of file Matrix2D.h.

References Mat2D_uint32::cols, MAT2D_AT_UINT32, and Mat2D_uint32::rows.

4.9.2.53 rand_double()

```
double rand_double (
            void  )
```

Definition at line 145 of file Matrix2D.h.

Referenced by mat2D_rand().

4.10 Matrix2D.h

```
00001 /* This one-file library is heavily inspired by Tsoding's nn.h implementation of matrix
00002 creation and operation. you can find the source code in:
00003 https://github.com/tsoding/nn.h .
00004 featured in this video of his:
00005 https://youtu.be/L1TbWe8bVOc?list=PLpM-Dvs8t0VZPZKggcql-MmjaBdZKeDMw .*/
00006
00007 /* NOTES:
00008  * There is a hole set of function for deling with the minors of a matrix because I tried to calculate
       the determinant of a matrix with them but it terns out to be TO SLOW. Insted I use Gauss elimination.
00009  * There are some stability problems in the inversion function. When the values of the matrix becomes
       too small, the inversion fails. Currently the only fix I can think of is to use pre-conditioners.
       However, it means to creat a function to solve the hole problem 'Ax=B', which wasn't my intention in
       the first place. I might do it but I am not sure. */
00010
00011 #ifndef MATRIX2D_H_
00012 #define MATRIX2D_H_
00013
00014 #include <stddef.h>
00015 #include <stdio.h>
00016 #include <stdlib.h>
00017 #include <stdint.h>
00018 #include <math.h>
00019 #include <stdbool.h>
00020
00021 #ifndef MATRIX2D_MALLOC
00022 #define MATRIX2D_MALLOC malloc
00023 #endif //MATRIX2D_MALLOC
00024
00025 #ifndef MATRIX2D_ASSERT
00026 #include <assert.h>
00027 #define MATRIX2D_ASSERT assert
00028 #endif //MATRIX2D_ASSERT
00029
00030 typedef struct {
00031     size_t rows;
00032     size_t cols;
00033     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00034     double *elements;
00035 } Mat2D;
00036
00037 typedef struct {
00038     size_t rows;
00039     size_t cols;
00040     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00041     uint32_t *elements;
00042 } Mat2D_uint32;
00043
00044 typedef struct {
00045     size_t rows;
00046     size_t cols;
00047     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00048     size_t *rows_list;
00049     size_t *cols_list;
00050     Mat2D ref_mat;
00051 } Mat2D_Minor;
00052
00053 #if 1
00054 #define MAT2D_AT(m, i, j) (m).elements[mat2D_offset2d((m), (i), (j))]
00055 #define MAT2D_AT_UINT32(m, i, j) (m).elements[mat2D_offset2d_uint32((m), (i), (j))]
00056 #else /* use this macro for batter performance but no assertion */
00057 #define MAT2D_AT(m, i, j) (m).elements[i * m.stride_r + j]
00058 #define MAT2D_AT_UINT32(m, i, j) (m).elements[i * m.stride_r + j]
00059 #endif
00060
00061 #ifndef PI
00062 #define PI M_PI
00063 #endif
00064
```

```
00065 #define MAT2D_MINOR_AT(mm, i, j) MAT2D_AT(mm.ref_mat, mm.rows_list[i], mm.cols_list[j])
00066 #define MAT2D_PRINT(m) mat2D_print(m, #m, 0)
00067 #define MAT2D_UINT32_PRINT(m) mat2D_uint32_print(m, #m, 0)
00068 #define MAT2D_PRINT_AS_COL(m) mat2D_print_as_col(m, #m, 0)
00069 #define MAT2D_MINOR_PRINT(mm) mat2D_minor_print(mm, #mm, 0)
00070
00071 double rand_double(void);
00072
00073 Mat2D mat2D_alloc(size_t rows, size_t cols);
00074 Mat2D_uint32 mat2D_alloc_uint32(size_t rows, size_t cols);
00075 void mat2D_free(Mat2D m);
00076 void mat2D_free_uint32(Mat2D_uint32 m);
00077 size_t mat2D_offset2d(Mat2D m, size_t i, size_t j);
00078 size_t mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j);
00079
00080 void mat2D_fill(Mat2D m, double x);
00081 void mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x);
00082 void mat2D_fill_sequence(Mat2D m, double start, double step);
00083 void mat2D_rand(Mat2D m, double low, double high);
00084
00085 void mat2D_dot(Mat2D dst, Mat2D a, Mat2D b);
00086 void mat2D_cross(Mat2D dst, Mat2D a, Mat2D b);
00087
00088 void mat2D_add(Mat2D dst, Mat2D a);
00089 void mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00090
00091 void mat2D_sub(Mat2D dst, Mat2D a);
00092 void mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00093
00094 void mat2D_mult(Mat2D m, double factor);
00095 void mat2D_mult_row(Mat2D m, size_t r, double factor);
00096
00097 void mat2D_print(Mat2D m, const char *name, size_t padding);
00098 void mat2D_uint32_print(Mat2D_uint32 m, const char *name, size_t padding);
00099 void mat2D_print_as_col(Mat2D m, const char *name, size_t padding);
00100
00101 void mat2D_set_identity(Mat2D m);
00102 double mat2D_make_identity(Mat2D m);
00103 void mat2D_set_rot_mat_x(Mat2D m, float angle_deg);
00104 void mat2D_set_rot_mat_y(Mat2D m, float angle_deg);
00105 void mat2D_set_rot_mat_z(Mat2D m, float angle_deg);
00106
00107 void mat2D_copy(Mat2D des, Mat2D src);
00108 void mat2D_copy_mat_to_mat_at_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t
       je);
00109
00110 void mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00111 void mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00112 void mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00113
00114 void mat2D_swap_rows(Mat2D m, size_t r1, size_t r2);
00115 void mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00116 void mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00117 void mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00118
00119 double mat2D_calc_norma(Mat2D m);
00120
00121 bool mat2D_mat_is_all_digit(Mat2D m, double digit);
00122 bool mat2D_row_is_all_digit(Mat2D m, double digit, size_t r);
00123 bool mat2D_col_is_all_digit(Mat2D m, double digit, size_t c);
00124
00125 double mat2D_det_2x2_mat(Mat2D m);
00126 double mat2D_triangulate(Mat2D m);
00127 double mat2D_det(Mat2D m);
00128 void mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u);
00129 void mat2D_transpose(Mat2D des, Mat2D src);
00130 void mat2D_invert(Mat2D des, Mat2D src);
00131 void mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B);
00132
00133 Mat2D_Minor mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j);
00134 Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j);
00135 void mat2D_minor_free(Mat2D_Minor mm);
00136 void mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding);
00137 double mat2D_det_2x2_mat_minor(Mat2D_Minor mm);
00138 double mat2D_minor_det(Mat2D_Minor mm);
00139
00140 #endif // MATRIX2D_H_
00141
00142 #ifdef MATRIX2D_IMPLEMENTATION
00143 #undef MATRIX2D_IMPLEMENTATION
00144
00145 double rand_double(void)
00146 {
00147     return (double) rand() / (double) RAND_MAX;
00148 }
00149
00150 Mat2D mat2D_alloc(size_t rows, size_t cols)
```

```
00151 {
00152     Mat2D m;
00153     m.rows = rows;
00154     m.cols = cols;
00155     m.stride_r = cols;
00156     m.elements = (double*)MATRIX2D_MALLOC(sizeof(double)*rows*cols);
00157     MATRIX2D_ASSERT(m.elements != NULL);
00158 
00159     return m;
00160 }
00161 
00162 Mat2D_uint32 mat2D_alloc_uint32(size_t rows, size_t cols)
00163 {
00164     Mat2D_uint32 m;
00165     m.rows = rows;
00166     m.cols = cols;
00167     m.stride_r = cols;
00168     m.elements = (uint32_t*)MATRIX2D_MALLOC(sizeof(uint32_t)*rows*cols);
00169     MATRIX2D_ASSERT(m.elements != NULL);
00170 
00171     return m;
00172 }
00173 
00174 void mat2D_free(Mat2D m)
00175 {
00176     free(m.elements);
00177 }
00178 
00179 void mat2D_free_uint32(Mat2D_uint32 m)
00180 {
00181     free(m.elements);
00182 }
00183 
00184 size_t mat2D_offset2d(Mat2D m, size_t i, size_t j)
00185 {
00186     MATRIX2D_ASSERT(i < m.rows && j < m.cols);
00187     return i * m.stride_r + j;
00188 }
00189 
00190 size_t mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j)
00191 {
00192     MATRIX2D_ASSERT(i < m.rows && j < m.cols);
00193     return i * m.stride_r + j;
00194 }
00195 
00196 void mat2D_fill(Mat2D m, double x)
00197 {
00198     for (size_t i = 0; i < m.rows; ++i) {
00199         for (size_t j = 0; j < m.cols; ++j) {
00200             MAT2D_AT(m, i, j) = x;
00201         }
00202     }
00203 }
00204 
00205 void mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x)
00206 {
00207     for (size_t i = 0; i < m.rows; ++i) {
00208         for (size_t j = 0; j < m.cols; ++j) {
00209             MAT2D_AT_UINT32(m, i, j) = x;
00210         }
00211     }
00212 }
00213 void mat2D_fill_sequence(Mat2D m, double start, double step) {
00214     for (size_t i = 0; i < m.rows; i++) {
00215         for (size_t j = 0; j < m.cols; j++) {
00216             MAT2D_AT(m, i, j) = start + step * mat2D_offset2d(m, i, j);
00217         }
00218     }
00219 }
00220 
00221 void mat2D_rand(Mat2D m, double low, double high)
00222 {
00223     for (size_t i = 0; i < m.rows; ++i) {
00224         for (size_t j = 0; j < m.cols; ++j) {
00225             MAT2D_AT(m, i, j) = rand_double()*(high - low) + low;
00226         }
00227     }
00228 }
00229 
00230 void mat2D_dot(Mat2D dst, Mat2D a, Mat2D b)
00231 {
00232     MATRIX2D_ASSERT(a.cols == b.rows);
00233     MATRIX2D_ASSERT(a.rows == dst.rows);
00234     MATRIX2D_ASSERT(b.cols == dst.cols);
00235 
00236     for (size_t i = 0; i < dst.rows; i++) {
00237         for (size_t j = 0; j < dst.cols; j++) {
```

```
00238             MAT2D_AT(dst, i, j) = 0;
00239             for (size_t k = 0; k < a.cols; k++) {
00240                 MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, k)*MAT2D_AT(b, k, j);
00241             }
00242         }
00243     }
00244
00245 }
00246
00247 void mat2D_cross(Mat2D dst, Mat2D a, Mat2D b)
00248 {
00249     MATRIX2D_ASSERT(3 == dst.rows && 1 == dst.cols);
00250     MATRIX2D_ASSERT(3 == a.rows && 1 == a.cols);
00251     MATRIX2D_ASSERT(3 == b.rows && 1 == b.cols);
00252
00253     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);
00254     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);
00255     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);
00256 }
00257
00258 void mat2D_add(Mat2D dst, Mat2D a)
00259 {
00260     MATRIX2D_ASSERT(dst.rows == a.rows);
00261     MATRIX2D_ASSERT(dst.cols == a.cols);
00262     for (size_t i = 0; i < dst.rows; ++i) {
00263         for (size_t j = 0; j < dst.cols; ++j) {
00264             MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, j);
00265         }
00266     }
00267 }
00268
00269 void mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
00270 {
00271     for (size_t j = 0; j < m.cols; ++j) {
00272         MAT2D_AT(m, des_r, j) += factor * MAT2D_AT(m, src_r, j);
00273     }
00274 }
00275
00276 void mat2D_sub(Mat2D dst, Mat2D a)
00277 {
00278     MATRIX2D_ASSERT(dst.rows == a.rows);
00279     MATRIX2D_ASSERT(dst.cols == a.cols);
00280     for (size_t i = 0; i < dst.rows; ++i) {
00281         for (size_t j = 0; j < dst.cols; ++j) {
00282             MAT2D_AT(dst, i, j) -= MAT2D_AT(a, i, j);
00283         }
00284     }
00285 }
00286
00287 void mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
00288 {
00289     for (size_t j = 0; j < m.cols; ++j) {
00290         MAT2D_AT(m, des_r, j) -= factor * MAT2D_AT(m, src_r, j);
00291     }
00292 }
00293
00294 void mat2D_mult(Mat2D m, double factor)
00295 {
00296     for (size_t i = 0; i < m.rows; ++i) {
00297         for (size_t j = 0; j < m.cols; ++j) {
00298             MAT2D_AT(m, i, j) *= factor;
00299         }
00300     }
00301 }
00302
00303 void mat2D_mult_row(Mat2D m, size_t r, double factor)
00304 {
00305     for (size_t j = 0; j < m.cols; ++j) {
00306         MAT2D_AT(m, r, j) *= factor;
00307     }
00308 }
00309
00310 void mat2D_print(Mat2D m, const char *name, size_t padding)
00311 {
00312     printf("%*s%s = [\n", (int) padding, "", name);
00313     for (size_t i = 0; i < m.rows; ++i) {
00314         printf("%*s    ", (int) padding, "");
00315         for (size_t j = 0; j < m.cols; ++j) {
00316             printf("%9.6f ", MAT2D_AT(m, i, j));
00317         }
00318         printf("\n");
00319     }
00320     printf("%*s]\n", (int) padding, "");
00321 }
```

```
00322
00323 void mat2D_uint32_print(Mat2D_uint32 m, const char *name, size_t padding)
00324 {
00325     printf("%*s%s = [\n", (int) padding, "", name);
00326     for (size_t i = 0; i < m.rows; ++i) {
00327         printf("%*s    ", (int) padding, "");
00328         for (size_t j = 0; j < m.cols; ++j) {
00329             if (MAT2D_AT_UINT32(m, i, j)) {
00330                 printf("%u ", MAT2D_AT_UINT32(m, i, j));
00331             } else {
00332                 printf("  ");
00333             }
00334         }
00335         printf("\n");
00336     }
00337     printf("%*s]\n", (int) padding, "");
00338 }
00339
00340 void mat2D_print_as_col(Mat2D m, const char *name, size_t padding)
00341 {
00342     printf("%*s%s = [\n", (int) padding, "", name);
00343     for (size_t i = 0; i < m.rows*m.cols; ++i) {
00344             printf("%*s    ", (int) padding, "");
00345             printf("%f\n", m.elements[i]);
00346     }
00347     printf("%*s]\n", (int) padding, "");
00348 }
00349
00350 void mat2D_set_identity(Mat2D m)
00351 {
00352     MATRIX2D_ASSERT(m.cols == m.rows);
00353     for (size_t i = 0; i < m.rows; ++i) {
00354         for (size_t j = 0; j < m.cols; ++j) {
00355             MAT2D_AT(m, i, j) = i == j ? 1 : 0;
00356             // if (i == j) {
00357             //     MAT2D_AT(m, i, j) = 1;
00358             // }
00359             // else {
00360             //     MAT2D_AT(m, i, j) = 0;
00361             // }
00362         }
00363     }
00364 }
00365
00366 double mat2D_make_identity(Mat2D m)
00367 {
00368     /* make identity matrix using Gauss elimination */
00369     /* preforming Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
00370     /* returns the factor multiplying the determinant */
00371
00372     double factor_to_return = 1;
00373
00374     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
00375         /* check if it is the biggest first number (absolute value) */
00376         size_t biggest_r = i;
00377         for (size_t index = i; index < m.rows; index++) {
00378             if (fabs(MAT2D_AT(m, index, index)) > fabs(MAT2D_AT(m, biggest_r, 0))) {
00379                 biggest_r = index;
00380             }
00381         }
00382         if (i != biggest_r) {
00383             mat2D_swap_rows(m, i, biggest_r);
00384             factor_to_return *= -1;
00385         }
00386         for (size_t j = i+1; j < m.cols; j++) {
00387             double factor = 1 / MAT2D_AT(m, i, i);
00388             mat2D_sub_row_time_factor_to_row(m, j, i, MAT2D_AT(m, j, i) * factor);
00389             mat2D_mult_row(m, i, factor);
00390             factor_to_return *= factor;
00391         }
00392     }
00393     double factor = 1 / MAT2D_AT(m, m.rows-1, m.cols-1);
00394     mat2D_mult_row(m, m.rows-1, factor);
00395     factor_to_return *= factor;
00396     for (size_t c = m.cols-1; c > 0; c--) {
00397         for (int r = c-1; r >= 0; r--) {
00398             double factor = 1 / MAT2D_AT(m, c, c);
00399             mat2D_sub_row_time_factor_to_row(m, r, c, MAT2D_AT(m, r, c) * factor);
00400         }
00401     }
00402
00403
00404     return factor_to_return;
00405 }
00406
00407 void mat2D_set_rot_mat_x(Mat2D m, float angle_deg)
00408 {
```

```
00409     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00410
00411     float angle_rad = angle_deg * PI / 180;
00412     mat2D_set_identity(m);
00413     MAT2D_AT(m, 1, 1) =  cos(angle_rad);
00414     MAT2D_AT(m, 1, 2) =  sin(angle_rad);
00415     MAT2D_AT(m, 2, 1) = -sin(angle_rad);
00416     MAT2D_AT(m, 2, 2) =  cos(angle_rad);
00417 }
00418
00419 void mat2D_set_rot_mat_y(Mat2D m, float angle_deg)
00420 {
00421     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00422
00423     float angle_rad = angle_deg * PI / 180;
00424     mat2D_set_identity(m);
00425     MAT2D_AT(m, 0, 0) =  cos(angle_rad);
00426     MAT2D_AT(m, 0, 2) = -sin(angle_rad);
00427     MAT2D_AT(m, 2, 0) =  sin(angle_rad);
00428     MAT2D_AT(m, 2, 2) =  cos(angle_rad);
00429 }
00430
00431 void mat2D_set_rot_mat_z(Mat2D m, float angle_deg)
00432 {
00433     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00434
00435     float angle_rad = angle_deg * PI / 180;
00436     mat2D_set_identity(m);
00437     MAT2D_AT(m, 0, 0) =  cos(angle_rad);
00438     MAT2D_AT(m, 0, 1) =  sin(angle_rad);
00439     MAT2D_AT(m, 1, 0) = -sin(angle_rad);
00440     MAT2D_AT(m, 1, 1) =  cos(angle_rad);
00441 }
00442
00443 void mat2D_copy(Mat2D des, Mat2D src)
00444 {
00445     MATRIX2D_ASSERT(des.cols == src.cols);
00446     MATRIX2D_ASSERT(des.rows == src.rows);
00447
00448     for (size_t i = 0; i < des.rows; ++i) {
00449         for (size_t j = 0; j < des.cols; ++j) {
00450             MAT2D_AT(des, i, j) = MAT2D_AT(src, i, j);
00451         }
00452     }
00453 }
00454
00455 void mat2D_copy_mat_to_mat_at_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)
00456 {
00457     MATRIX2D_ASSERT(je > js && ie > is);
00458     MATRIX2D_ASSERT(je-js+1 == des.cols);
00459     MATRIX2D_ASSERT(ie-is+1 == des.rows);
00460
00461     for (size_t index = 0; index < des.rows; ++index) {
00462         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
00463             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, is+index, js+jndex);
00464         }
00465     }
00466 }
00467
00468 void mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00469 {
00470     MATRIX2D_ASSERT(src_col < src.cols);
00471     MATRIX2D_ASSERT(des.rows == src.rows);
00472     MATRIX2D_ASSERT(des_col < des.cols);
00473
00474     for (size_t i = 0; i < des.rows; i++) {
00475         MAT2D_AT(des, i, des_col) = MAT2D_AT(src, i, src_col);
00476     }
00477 }
00478
00479 void mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00480 {
00481     MATRIX2D_ASSERT(src_col < src.cols);
00482     MATRIX2D_ASSERT(des.rows == src.rows);
00483     MATRIX2D_ASSERT(des_col < des.cols);
00484
00485     for (size_t i = 0; i < des.rows; i++) {
00486         MAT2D_AT(des, i, des_col) += MAT2D_AT(src, i, src_col);
00487     }
00488 }
00489
00490 void mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00491 {
00492     MATRIX2D_ASSERT(src_col < src.cols);
00493     MATRIX2D_ASSERT(des.rows == src.rows);
00494     MATRIX2D_ASSERT(des_col < des.cols);
00495
```

```
00496     for (size_t i = 0; i < des.rows; i++) {
00497         MAT2D_AT(des, i, des_col) -= MAT2D_AT(src, i, src_col);
00498     }
00499 }
00500
00501 void mat2D_swap_rows(Mat2D m, size_t r1, size_t r2)
00502 {
00503     for (size_t j = 0; j < m.cols; j++) {
00504         double temp = MAT2D_AT(m, r1, j);
00505         MAT2D_AT(m, r1, j) = MAT2D_AT(m, r2, j);
00506         MAT2D_AT(m, r2, j) = temp;
00507     }
00508 }
00509
00510 void mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00511 {
00512     MATRIX2D_ASSERT(src_row < src.rows);
00513     MATRIX2D_ASSERT(des.cols == src.cols);
00514     MATRIX2D_ASSERT(des_row < des.rows);
00515
00516     for (size_t j = 0; j < des.cols; j++) {
00517         MAT2D_AT(des, des_row, j) = MAT2D_AT(src, src_row, j);
00518     }
00519 }
00520
00521 void mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00522 {
00523     MATRIX2D_ASSERT(src_row < src.rows);
00524     MATRIX2D_ASSERT(des.cols == src.cols);
00525     MATRIX2D_ASSERT(des_row < des.rows);
00526
00527     for (size_t j = 0; j < des.cols; j++) {
00528         MAT2D_AT(des, des_row, j) += MAT2D_AT(src, src_row, j);
00529     }
00530 }
00531
00532 void mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00533 {
00534     MATRIX2D_ASSERT(src_row < src.rows);
00535     MATRIX2D_ASSERT(des.cols == src.cols);
00536     MATRIX2D_ASSERT(des_row < des.rows);
00537
00538     for (size_t j = 0; j < des.cols; j++) {
00539         MAT2D_AT(des, des_row, j) -= MAT2D_AT(src, src_row, j);
00540     }
00541 }
00542
00543 double mat2D_calc_norma(Mat2D m)
00544 {
00545     double sum = 0;
00546
00547     for (size_t i = 0; i < m.rows; ++i) {
00548         for (size_t j = 0; j < m.cols; ++j) {
00549             sum += MAT2D_AT(m, i, j) * MAT2D_AT(m, i, j);
00550         }
00551     }
00552     return sqrt(sum);
00553 }
00554
00555 bool mat2D_mat_is_all_digit(Mat2D m, double digit)
00556 {
00557     for (size_t i = 0; i < m.rows; ++i) {
00558         for (size_t j = 0; j < m.cols; ++j) {
00559             if (MAT2D_AT(m, i, j) != digit) {
00560                 return false;
00561             }
00562         }
00563     }
00564     return true;
00565 }
00566
00567 bool mat2D_row_is_all_digit(Mat2D m, double digit, size_t r)
00568 {
00569     for (size_t j = 0; j < m.cols; ++j) {
00570         if (MAT2D_AT(m, r, j) != digit) {
00571             return false;
00572         }
00573     }
00574     return true;
00575 }
00576
00577 bool mat2D_col_is_all_digit(Mat2D m, double digit, size_t c)
00578 {
00579     for (size_t i = 0; i < m.cols; ++i) {
00580         if (MAT2D_AT(m, i, c) != digit) {
00581             return false;
00582         }
```

```
00583     }
00584     return true;
00585 }
00586
00587 double mat2D_det_2x2_mat(Mat2D m)
00588 {
00589     MATRIX2D_ASSERT(2 == m.cols && 2 == m.rows && "Not a 2x2 matrix");
00590     return MAT2D_AT(m, 0, 0) * MAT2D_AT(m, 1, 1) - MAT2D_AT(m, 0, 1) * MAT2D_AT(m, 1, 0);
00591 }
00592
00593 double mat2D_triangulate(Mat2D m)
00594 {
00595     /* preforming Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
00596     /* returns the factor multiplying the determinant */
00597
00598     double factor_to_return = 1;
00599
00600     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
00601         if (!MAT2D_AT(m, i, i)) {   /* swapping only if it is zero */
00602             /* finding biggest first number (absolute value) */
00603             size_t biggest_r = i;
00604             for (size_t index = i; index < m.rows; index++) {
00605                 if (fabs(MAT2D_AT(m, index, i)) > fabs(MAT2D_AT(m, biggest_r, i))) {
00606                     biggest_r = index;
00607                 }
00608             }
00609             if (i != biggest_r) {
00610                 mat2D_swap_rows(m, i, biggest_r);
00611             }
00612         }
00613         for (size_t j = i+1; j < m.cols; j++) {
00614             double factor = 1 / MAT2D_AT(m, i, i);
00615             if (!isfinite(factor)) {
00616                 printf("%s:%d: [Error] unable to transfrom into uperr triangular matrix. Probably some
      of the rows are not independent.\n", __FILE__, __LINE__);
00617             }
00618             double mat_value = MAT2D_AT(m, j, i);
00619             mat2D_sub_row_time_factor_to_row(m, j, i, mat_value * factor);
00620         }
00621     }
00622     return factor_to_return;
00623 }
00624
00625 double mat2D_det(Mat2D m)
00626 {
00627     MATRIX2D_ASSERT(m.cols == m.rows && "should be a square matrix");
00628
00629     /* checking if there is a row or column with all zeros */
00630     /* checking rows */
00631     for (size_t i = 0; i < m.rows; i++) {
00632         if (mat2D_row_is_all_digit(m, 0, i)) {
00633             return 0;
00634         }
00635     }
00636     /* checking cols */
00637     for (size_t j = 0; j < m.rows; j++) {
00638         if (mat2D_col_is_all_digit(m, 0, j)) {
00639             return 0;
00640         }
00641     }
00642
00643     /* This is an implementation of naive determinant calculation using minors. This is too slow */
00644
00645     // double det = 0;
00646     // /* TODO: finding beast row or col? */
00647     // for (size_t i = 0, j = 0; i < m.rows; i++) { /* first column */
00648     //     if (MAT2D_AT(m, i, j) < 1e-10) continue;
00649     //     Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat(m, i, j);
00650     //     int factor = (i+j)%2 ? -1 : 1;
00651     //     if (sub_mm.cols != 2) {
00652     //         MATRIX2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
00653     //         det += MAT2D_AT(m, i, j) * (factor) * mat2D_minor_det(sub_mm);
00654     //     } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
00655     //         det += MAT2D_AT(m, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);;
00656     //     }
00657     //     mat2D_minor_free(sub_mm);
00658     // }
00659
00660     Mat2D temp_m = mat2D_alloc(m.rows, m.cols);
00661     mat2D_copy(temp_m, m);
00662     double factor = mat2D_triangulate(temp_m);
00663     double diag_mul = 1;
00664     for (size_t i = 0; i < temp_m.rows; i++) {
00665         diag_mul *= MAT2D_AT(temp_m, i, i);
00666     }
00667     mat2D_free(temp_m);
00668
```

```
00669     return diag_mul / factor;
00670 }
00671
00672 void mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u)
00673 {
00674     /* performing LU decomposition Following the Wikipedia page:
      https://en.wikipedia.org/wiki/LU_decomposition */
00675
00676     mat2D_copy(u, src);
00677     mat2D_set_identity(p);
00678     mat2D_fill(l, 0);
00679
00680     for (size_t i = 0; i < (size_t)fmin(u.rows-1, u.cols); i++) {
00681         if (!MAT2D_AT(u, i, i)) {   /* swapping only if it is zero */
00682             /* finding biggest first number (absolute value) */
00683             size_t biggest_r = i;
00684             for (size_t index = i; index < u.rows; index++) {
00685                 if (fabs(MAT2D_AT(u, index, i)) > fabs(MAT2D_AT(u, biggest_r, i))) {
00686                     biggest_r = index;
00687                 }
00688             }
00689             if (i != biggest_r) {
00690                 mat2D_swap_rows(u, i, biggest_r);
00691                 mat2D_swap_rows(p, i, biggest_r);
00692                 mat2D_swap_rows(l, i, biggest_r);
00693             }
00694         }
00695         for (size_t j = i+1; j < u.cols; j++) {
00696             double factor = 1 / MAT2D_AT(u, i, i);
00697             if (!isfinite(factor)) {
00698                 printf("%s:%d: [Error] unable to transfrom into uper triangular matrix. Probably some
      of the rows are not independent.\n", __FILE__, __LINE__);
00699             }
00700             double mat_value = MAT2D_AT(u, j, i);
00701             mat2D_sub_row_time_factor_to_row(u, j, i, mat_value * factor);
00702             MAT2D_AT(l, j, i) = mat_value * factor;
00703         }
00704         MAT2D_AT(l, i, i) = 1;
00705     }
00706     MAT2D_AT(l, l.rows-1, l.cols-1) = 1;
00707 }
00708
00709 void mat2D_transpose(Mat2D des, Mat2D src)
00710 {
00711     MATRIX2D_ASSERT(des.cols == src.rows);
00712     MATRIX2D_ASSERT(des.rows == src.cols);
00713
00714     for (size_t index = 0; index < des.rows; ++index) {
00715         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
00716             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, jndex, index);
00717         }
00718     }
00719 }
00720
00721 void mat2D_invert(Mat2D des, Mat2D src)
00722 {
00723     MATRIX2D_ASSERT(src.cols == src.rows && "should be an NxN matrix");
00724     MATRIX2D_ASSERT(des.cols == src.cols && des.rows == des.cols);
00725
00726     Mat2D m = mat2D_alloc(src.rows, src.cols);
00727     mat2D_copy(m, src);
00728
00729     mat2D_set_identity(des);
00730
00731     if (!mat2D_det(m)) {
00732         mat2D_fill(des, 0);
00733         printf("%s:%d: [Error] Can't invert the matrix. Determinant is zero! Set the inverse matrix to
      all zeros\n", __FILE__, __LINE__);
00734         return;
00735     }
00736
00737     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
00738         if (!MAT2D_AT(m, i, i)) {   /* swapping only if it is zero */
00739             /* finding biggest first number (absolute value) */
00740             size_t biggest_r = i;
00741             for (size_t index = i; index < m.rows; index++) {
00742                 if (fabs(MAT2D_AT(m, index, i)) > fabs(MAT2D_AT(m, biggest_r, i))) {
00743                     biggest_r = index;
00744                 }
00745             }
00746             if (i != biggest_r) {
00747                 mat2D_swap_rows(m, i, biggest_r);
00748                 mat2D_swap_rows(des, i, biggest_r);
00749                 printf("%s:%d: [INFO] swapping row %zu with row %zu.\n", __FILE__, __LINE__, i,
      biggest_r);
00750             } else {
00751                 MATRIX2D_ASSERT(0 && "can't inverse");
```

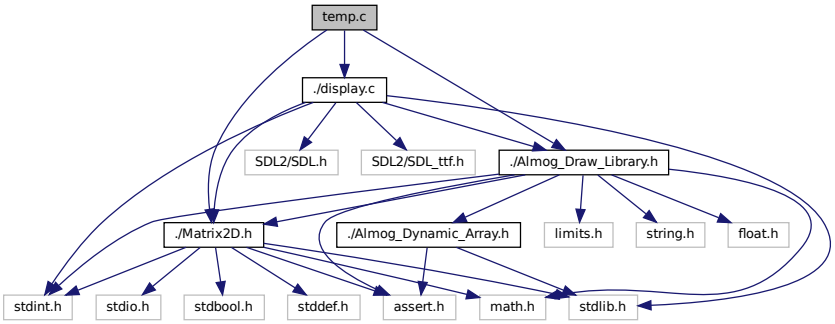
```
00752             }
00753         }
00754         for (size_t j = i+1; j < m.cols; j++) {
00755             double factor = 1 / MAT2D_AT(m, i, i);
00756             double mat_value = MAT2D_AT(m, j, i);
00757             mat2D_sub_row_time_factor_to_row(m, j, i, mat_value * factor);
00758             mat2D_mult_row(m, i, factor);
00759 
00760             mat2D_sub_row_time_factor_to_row(des, j, i, mat_value * factor);
00761             mat2D_mult_row(des, i, factor);
00762         }
00763     }
00764     double factor = 1 / MAT2D_AT(m, m.rows-1, m.cols-1);
00765     mat2D_mult_row(m, m.rows-1, factor);
00766     mat2D_mult_row(des, des.rows-1, factor);
00767     for (size_t c = m.cols-1; c > 0; c--) {
00768         for (int r = c-1; r >= 0; r--) {
00769             double factor = 1 / MAT2D_AT(m, c, c);
00770             double mat_value = MAT2D_AT(m, r, c);
00771             mat2D_sub_row_time_factor_to_row(m, r, c, mat_value * factor);
00772             mat2D_sub_row_time_factor_to_row(des, r, c, mat_value * factor);
00773         }
00774     }
00775 
00776     mat2D_free(m);
00777 }
00778 
00779 void mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B)
00780 {
00781     MATRIX2D_ASSERT(A.cols == x.rows);
00782     MATRIX2D_ASSERT(1 == x.cols);
00783     MATRIX2D_ASSERT(A.rows == B.rows);
00784     MATRIX2D_ASSERT(1 == B.cols);
00785 
00786     Mat2D y     = mat2D_alloc(x.rows, x.cols);
00787     Mat2D l     = mat2D_alloc(A.rows, A.cols);
00788     Mat2D p     = mat2D_alloc(A.rows, A.cols);
00789     Mat2D u     = mat2D_alloc(A.rows, A.cols);
00790     Mat2D inv_l = mat2D_alloc(l.rows, l.cols);
00791     Mat2D inv_u = mat2D_alloc(u.rows, u.cols);
00792 
00793     mat2D_LUP_decomposition_with_swap(A, l, p, u);
00794 
00795     mat2D_invert(inv_l, l);
00796     mat2D_invert(inv_u, u);
00797 
00798     mat2D_fill(x, 0);   /* x here is only a temp mat*/
00799     mat2D_fill(y, 0);
00800     mat2D_dot(x, p, B);
00801     mat2D_dot(y, inv_l, x);
00802 
00803     mat2D_fill(x, 0);
00804     mat2D_dot(x, inv_u, y);
00805 
00806     mat2D_free(y);
00807     mat2D_free(l);
00808     mat2D_free(p);
00809     mat2D_free(u);
00810     mat2D_free(inv_l);
00811     mat2D_free(inv_u);
00812 }
00813 
00814 Mat2D_Minor mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j)
00815 {
00816     MATRIX2D_ASSERT(ref_mat.cols == ref_mat.rows && "minor is defined only for square matrix");
00817 
00818     Mat2D_Minor mm;
00819     mm.cols = ref_mat.cols-1;
00820     mm.rows = ref_mat.rows-1;
00821     mm.stride_r = ref_mat.cols-1;
00822     mm.cols_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mat.cols-1));
00823     mm.rows_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mat.rows-1));
00824     mm.ref_mat = ref_mat;
00825 
00826     MATRIX2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
00827 
00828     for (size_t index = 0, temp_index = 0; index < ref_mat.rows; index++) {
00829         if (index != i) {
00830             mm.rows_list[temp_index] = index;
00831             temp_index++;
00832         }
00833     }
00834     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mat.rows; jndex++) {
00835         if (jndex != j) {
00836             mm.cols_list[temp_jndex] = jndex;
00837             temp_jndex++;
00838         }
```

```
00839     }
00840
00841     return mm;
00842 }
00843
00844 Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j)
00845 {
00846     MATRIX2D_ASSERT(ref_mm.cols == ref_mm.rows && "minor is defined only for square matrix");
00847
00848     Mat2D_Minor mm;
00849     mm.cols = ref_mm.cols-1;
00850     mm.rows = ref_mm.rows-1;
00851     mm.stride_r = ref_mm.cols-1;
00852     mm.cols_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mm.cols-1));
00853     mm.rows_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mm.rows-1));
00854     mm.ref_mat = ref_mm.ref_mat;
00855
00856     MATRIX2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
00857
00858     for (size_t index = 0, temp_index = 0; index < ref_mm.rows; index++) {
00859         if (index != i) {
00860             mm.rows_list[temp_index] = ref_mm.rows_list[index];
00861             temp_index++;
00862         }
00863     }
00864     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mm.rows; jndex++) {
00865         if (jndex != j) {
00866             mm.cols_list[temp_jndex] = ref_mm.cols_list[jndex];
00867             temp_jndex++;
00868         }
00869     }
00870
00871     return mm;
00872 }
00873
00874 void mat2D_minor_free(Mat2D_Minor mm)
00875 {
00876     free(mm.cols_list);
00877     free(mm.rows_list);
00878 }
00879
00880 void mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding)
00881 {
00882     printf("%*s%s = [\n", (int) padding, "", name);
00883     for (size_t i = 0; i < mm.rows; ++i) {
00884         printf("%*s    ", (int) padding, "");
00885         for (size_t j = 0; j < mm.cols; ++j) {
00886             printf("%f ", MAT2D_MINOR_AT(mm, i, j));
00887         }
00888         printf("\n");
00889     }
00890     printf("%*s]\n", (int) padding, "");
00891 }
00892
00893 double mat2D_det_2x2_mat_minor(Mat2D_Minor mm)
00894 {
00895     MATRIX2D_ASSERT(2 == mm.cols && 2 == mm.rows && "Not a 2x2 matrix");
00896     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);
00897 }
00898
00899 double mat2D_minor_det(Mat2D_Minor mm)
00900 {
00901     MATRIX2D_ASSERT(mm.cols == mm.rows && "should be a square matrix");
00902
00903     double det = 0;
00904     /* TODO: finding beast row or col? */
00905     for (size_t i = 0, j = 0; i < mm.rows; i++) { /* first column */
00906         if (MAT2D_MINOR_AT(mm, i, j) < 1e-10) continue;
00907         Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat_minor(mm, i, j);
00908         int factor = (i+j)%2 ? -1 : 1;
00909         if (sub_mm.cols != 2) {
00910             MATRIX2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
00911             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_minor_det(sub_mm);
00912         } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
00913             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);;
00914         }
00915         mat2D_minor_free(sub_mm);
00916     }
00917     return det;
00918 }
00919
00920
00921 #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,
      game_state->inv_z_buffer_mat, quad1, game_state->offset_zoom_param);
00053     adl_quad_draw(game_state->window_pixels_mat, game_state->inv_z_buffer_mat, quad1, 0xFF000000,
      game_state->offset_zoom_param);
00054
00055     adl_tri_fill_Pinedas_rasterizer_interpolate_color(game_state->window_pixels_mat,
      game_state->inv_z_buffer_mat, tri, game_state->offset_zoom_param);
00056     adl_tri_draw(game_state->window_pixels_mat, tri, 0xff000000, game_state->offset_zoom_param);
00057
00058     #if 0
00059     Mat2D inv_z_buffer = game_state->inv_z_buffer_mat;
00060     double max_inv_z = 0;
00061     double min_inv_z = DBL_MAX;
00062     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