Almog Engine

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:

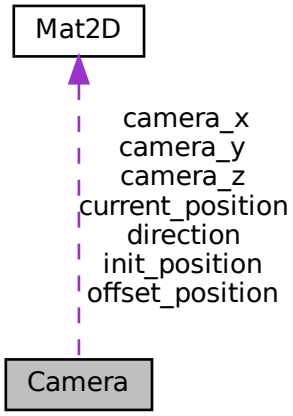
Chapter 3

Class Documentation

3.1 Camera Struct Reference

`#include <Almog_Engine.h>`

Collaboration diagram for Camera:



Public Attributes

- Mat2D init_position
- Mat2D current_position
- Mat2D offset_position
- Mat2D direction
- float z_near
- float z_far
- float fov_deg
- float aspect_ratio

- float roll_offset_deg
- float pitch_offset_deg
- float yaw_offset_deg
- Mat2D camera_x
- Mat2D camera_y
- Mat2D camera_z

3.1.1 Detailed Description

Definition at line 144 of file Almog_Engine.h.

3.1.2 Member Data Documentation

3.1.2.1 aspect_ratio

```
float Camera::aspect_ratio
```

Definition at line 152 of file Almog_Engine.h.

Referenced by ae_camera_init(), ae_scene_init(), check_window_mat_size(), and update().

3.1.2.2 camera_x

```
Mat2D Camera::camera_x
```

Definition at line 156 of file Almog_Engine.h.

Referenced by ae_camera_free(), ae_camera_init(), ae_camera_reset_pos(), and ae_view_mat_set().

3.1.2.3 camera_y

```
Mat2D Camera::camera_y
```

Definition at line 157 of file Almog_Engine.h.

Referenced by ae_camera_free(), ae_camera_init(), ae_camera_reset_pos(), and ae_view_mat_set().

3.1.2.4 camera_z

```
Mat2D Camera::camera_z
```

Definition at line 158 of file Almog_Engine.h.

Referenced by ae_camera_free(), ae_camera_init(), ae_camera_reset_pos(), and ae_view_mat_set().

3.1.2.5 current_position

```
Mat2D Camera::current_position
```

Definition at line 146 of file Almog_Engine.h.

Referenced by ae_camera_free(), ae_camera_init(), ae_camera_reset_pos(), ae_quad_calc_light_intensity(), ae_quad_project_world2screen(), ae_tri_calc_light_intensity(), ae_tri_project_world2screen(), and ae_view_mat_set().

3.1.2.6 direction

```
Mat2D Camera::direction
```

Definition at line 148 of file Almog_Engine.h.

Referenced by ae_camera_free(), ae_camera_init(), and ae_view_mat_set().

3.1.2.7 fov_deg

```
float Camera::fov_deg
```

Definition at line 151 of file Almog_Engine.h.

Referenced by ae_camera_init(), ae_scene_init(), and update().

3.1.2.8 init_position

```
Mat2D Camera::init_position
```

Definition at line 145 of file Almog_Engine.h.

Referenced by ae_camera_free(), ae_camera_init(), and ae_camera_reset_pos().

3.1.2.9 offset_position

```
Mat2D Camera::offset_position
```

Definition at line 147 of file Almog_Engine.h.

Referenced by ae_camera_free(), ae_camera_init(), ae_camera_reset_pos(), ae_view_mat_set(), and process_input_window().

3.1.2.10 pitch_offset_deg

```
float Camera::pitch_offset_deg
```

Definition at line 154 of file Almog_Engine.h.

Referenced by ae_camera_init(), ae_camera_reset_pos(), ae_view_mat_set(), and process_input_window().

3.1.2.11 roll_offset_deg

```
float Camera::roll_offset_deg
```

Definition at line 153 of file Almog_Engine.h.

Referenced by ae_camera_init(), ae_camera_reset_pos(), ae_view_mat_set(), and process_input_window().

3.1.2.12 yaw_offset_deg

```
float Camera::yaw_offset_deg
```

Definition at line 155 of file Almog_Engine.h.

Referenced by ae_camera_init(), ae_camera_reset_pos(), and ae_view_mat_set().

3.1.2.13 z_far

```
float Camera::z_far
```

Definition at line 150 of file Almog_Engine.h.

Referenced by ae_camera_init(), ae_scene_init(), and update().

3.1.2.14 z_near

```
float Camera::z_near
```

Definition at line 149 of file Almog_Engine.h.

Referenced by ae_camera_init(), ae_line_project_world2screen(), ae_quad_project_world2screen(), ae_scene_init(), ae_tri_project_world2screen(), and update().

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

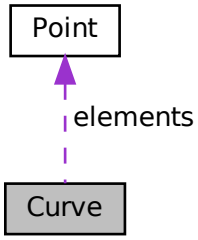
- src/include/Almog_Engine.h

3.2 Curve Struct Reference

Polyline of points with a uniform color.

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

Collaboration diagram for Curve:



Public Attributes

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

3.2.1 Detailed Description

Polyline of points with a uniform color.

Definition at line 94 of file Almog_Draw_Library.h.

3.2.2 Member Data Documentation

3.2.2.1 capacity

```
size_t Curve::capacity
```

Allocated capacity.

Definition at line 97 of file Almog_Draw_Library.h.

3.2.2.2 color

```
uint32_t Curve::color
```

ARGB color (0xAARRGGBB) for the entire curve.

Definition at line 95 of file Almog_Draw_Library.h.

Referenced by adl_curve_add_to_figure(), and adl_curves_plot_on_figure().

3.2.2.3 elements

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

Point array.

Definition at line 98 of file Almog_Draw_Library.h.

Referenced by adl_curves_plot_on_figure(), adl_grid_draw(), ae_curve_copy(), ae_curve_project_world2screen(), ae_print_points(), and ae_tri_mesh_get_from_obj_file().

3.2.2.4 length

```
size_t Curve::length
```

Number of points used.

Definition at line 96 of file Almog_Draw_Library.h.

Referenced by adl_curves_plot_on_figure(), adl_grid_draw(), ae_curve_copy(), ae_curve_project_world2screen(), and ae_print_points().

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

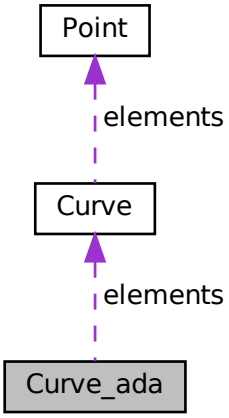
- src/include/Almog_Draw_Library.h

3.3 Curve_ada Struct Reference

Dynamic array of curves (polyline container).

`#include <Almog_Draw_Library.h>`

Collaboration diagram for Curve_ada:



Public Attributes

- size_t length
- size_t capacity
- Curve * elements

3.3.1 Detailed Description

Dynamic array of curves (polyline container).

Definition at line 107 of file Almog_Draw_Library.h.

3.3.2 Member Data Documentation

3.3.2.1 capacity

`size_t Curve_ada::capacity`

Allocated capacity.

Definition at line 109 of file Almog_Draw_Library.h.

3.3.2.2 elements

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

Curves array.

Definition at line 110 of file Almog_Draw_Library.h.

Referenced by adl_curves_plot_on_figure(), adl_grid_draw(), ae_curve_ada_project_world2screen(), and ae_grid_project_world2screen().

3.3.2.3 length

```
size_t Curve_ada::length
```

Number of curves used.

Definition at line 108 of file Almog_Draw_Library.h.

Referenced by adl_curves_plot_on_figure(), adl_grid_draw(), ae_curve_ada_project_world2screen(), and ae_grid_project_world2screen().

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

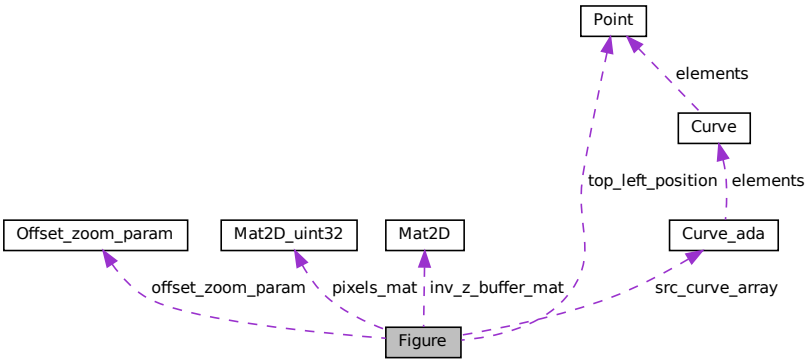
- src/include/Almog_Draw_Library.h

3.4 Figure Struct Reference

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

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

Collaboration diagram for Figure:



Public Attributes

- int min_x_pixel
- int max_x_pixel
- int min_y_pixel
- int max_y_pixel
- float min_x
- float max_x
- float min_y
- float max_y
- int x_axis_head_size
- int y_axis_head_size
- Offset_zoom_param offset_zoom_param
- Curve_ada src_curve_array
- Point top_left_position
- Mat2D_uint32 pixels_mat
- Mat2D inv_z_buffer_mat
- uint32_t background_color
- bool to_draw_axis
- bool to_draw_max_min_values

3.4.1 Detailed Description

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

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

Definition at line 174 of file Almog_Draw_Library.h.

3.4.2 Member Data Documentation

3.4.2.1 background_color

```
uint32_t Figure::background_color
```

Clear color for figure.

Definition at line 195 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), and adl_curves_plot_on_figure().

3.4.2.2 inv_z_buffer_mat

```
Mat2D Figure::inv_z_buffer_mat
```

Owned inverse-Z buffer (double).

Definition at line 193 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_curves_plot_on_figure(), and adl_figure_alloc().

3.4.2.3 max_x

```
float Figure::max_x
```

Max X value in source data.

Definition at line 181 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_curve_add_to_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), and adl_max_min_values_draw_on_figure().

3.4.2.4 max_x_pixel

```
int Figure::max_x_pixel
```

Right bound (pixel space).

Definition at line 176 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_axis_draw_on_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), and adl_max_min_values_draw_on_figure().

3.4.2.5 max_y

```
float Figure::max_y
```

Max Y value in source data.

Definition at line 183 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_curve_add_to_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), and adl_max_min_values_draw_on_figure().

3.4.2.6 max_y_pixel

```
int Figure::max_y_pixel
```

Bottom bound (pixel space).

Definition at line 178 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_axis_draw_on_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), and adl_max_min_values_draw_on_figure().

3.4.2.7 min_x

```
float Figure::min_x
```

Min X value in source data.

Definition at line 180 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_curve_add_to_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), and adl_max_min_values_draw_on_figure().

3.4.2.8 min_x_pixel

```
int Figure::min_x_pixel
```

Left padding (pixel space).

Definition at line 175 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_axis_draw_on_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), and adl_max_min_values_draw_on_figure().

3.4.2.9 min_y

```
float Figure::min_y
```

Min Y value in source data.

Definition at line 182 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_curve_add_to_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), and adl_max_min_values_draw_on_figure().

3.4.2.10 min_y_pixel

```
int Figure::min_y_pixel
```

Top padding (pixel space).

Definition at line 177 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_axis_draw_on_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), and adl_max_min_values_draw_on_figure().

3.4.2.11 offset_zoom_param

```
Offset_zoom_param Figure::offset_zoom_param
```

Pan/zoom parameters.

Definition at line 188 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_axis_draw_on_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), and adl_max_min_values_draw_on_figure().

3.4.2.12 pixels_mat

```
Mat2D_uint32 Figure::pixels_mat
```

Owned ARGB pixel buffer.

Definition at line 192 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_axis_draw_on_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), adl_figure_copy_to_screen(), and adl_max_min_values_draw_on_figure().

3.4.2.13 src_curve_array

```
Curve_ada Figure::src_curve_array
```

Curves to plot.

Definition at line 189 of file Almog_Draw_Library.h.

Referenced by adl_curve_add_to_figure(), adl_curves_plot_on_figure(), and adl_figure_alloc().

3.4.2.14 to_draw_axis

```
bool Figure::to_draw_axis
```

Draw axes when plotting.

Definition at line 196 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), and adl_curves_plot_on_figure().

3.4.2.15 to_draw_max_min_values

```
bool Figure::to_draw_max_min_values
```

Draw min/max labels.

Definition at line 197 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), and adl_curves_plot_on_figure().

3.4.2.16 top_left_position

```
Point Figure::top_left_position
```

On-screen copy position.

Definition at line 190 of file Almog_Draw_Library.h.

Referenced by adl_figure_alloc(), and adl_figure_copy_to_screen().

3.4.2.17 x_axis_head_size

```
int Figure::x_axis_head_size
```

Computed X-axis arrow head size (px).

Definition at line 185 of file Almog_Draw_Library.h.

Referenced by adl_axis_draw_on_figure(), and adl_max_min_values_draw_on_figure().

3.4.2.18 y_axis_head_size

```
int Figure::y_axis_head_size
```

Computed Y-axis arrow head size (px).

Definition at line 186 of file Almog_Draw_Library.h.

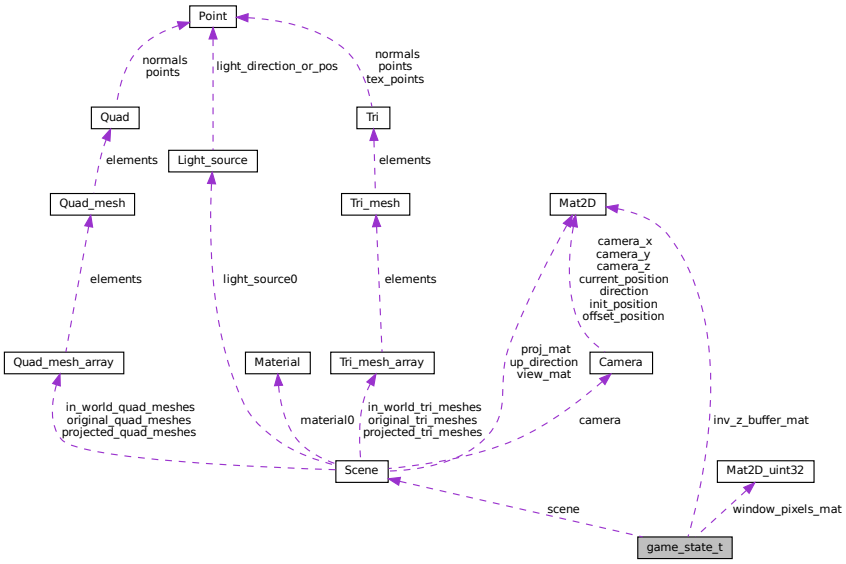
Referenced by adl_axis_draw_on_figure(), and adl_max_min_values_draw_on_figure().

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

- src/include/Almog_Draw_Library.h

3.5 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
- SDL_Surface * window_surface
- SDL_Texture * window_texture
- Mat2D_uint32 window_pixels_mat
- Mat2D inv_z_buffer_mat
- Scene scene

3.5.1 Detailed Description

Definition at line 38 of file display.c.

3.5.2 Member Data Documentation

3.5.2.1 a_was_pressed

```
int game_state_t::a_was_pressed
```

Definition at line 55 of file display.c.

Referenced by main().

3.5.2.2 const_fps

```
float game_state_t::const_fps
```

Definition at line 42 of file display.c.

Referenced by main(), setup(), and update_window().

3.5.2.3 d_was_pressed

```
int game_state_t::d_was_pressed
```

Definition at line 56 of file display.c.

Referenced by main().

3.5.2.4 delta_time

```
float game_state_t::delta_time
```

Definition at line 40 of file display.c.

Referenced by fix_framerate(), main(), and update_window().

3.5.2.5 e_was_pressed

```
int game_state_t::e_was_pressed
```

Definition at line 57 of file display.c.

Referenced by main().

3.5.2.6 elapsed_time

```
float game_state_t::elapsed_time
```

Definition at line 41 of file display.c.

Referenced by main(), and update_window().

3.5.2.7 fps

```
float game_state_t::fps
```

Definition at line 43 of file display.c.

Referenced by main(), and update_window().

3.5.2.8 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().

3.5.2.9 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().

3.5.2.10 inv_z_buffer_mat

```
Mat2D game_state_t::inv_z_buffer_mat
```

Definition at line 69 of file display.c.

Referenced by check_window_mat_size(), destroy_window(), render(), render_window(), and setup_window().

3.5.2.11 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().

3.5.2.12 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().

3.5.2.13 q_was_pressed

```
int game_state_t::q_was_pressed
```

Definition at line 58 of file display.c.

Referenced by main().

3.5.2.14 renderer

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

Definition at line 63 of file display.c.

Referenced by destroy_window(), initialize_window(), and main().

3.5.2.15 s_was_pressed

```
int game_state_t::s_was_pressed
```

Definition at line 54 of file display.c.

Referenced by main().

3.5.2.16 scene

```
Scene game_state_t::scene
```

Definition at line 71 of file display.c.

Referenced by check_window_mat_size(), destroy_window(), process_input_window(), render(), setup(), setup_window(), and update().

3.5.2.17 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().

3.5.2.18 to_clear_renderer

```
int game_state_t::to_clear_renderer
```

Definition at line 50 of file display.c.

Referenced by main(), and render_window().

3.5.2.19 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().

3.5.2.20 to_render

```
int game_state_t::to_render
```

Definition at line 45 of file display.c.

Referenced by main(), and process_input_window().

3.5.2.21 to_update

```
int game_state_t::to_update
```

Definition at line 46 of file display.c.

Referenced by main(), and process_input_window().

3.5.2.22 w_was_pressed

```
int game_state_t::w_was_pressed
```

Definition at line 53 of file display.c.

Referenced by main().

3.5.2.23 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().

3.5.2.24 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(), update(), and update_window().

3.5.2.25 window_pixels_mat

```
Mat2D_uint32 game_state_t::window_pixels_mat
```

Definition at line 68 of file display.c.

Referenced by check_window_mat_size(), copy_mat_to_surface_RGB(), destroy_window(), render(), render_window(), and setup_window().

3.5.2.26 window_surface

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

Definition at line 65 of file display.c.

Referenced by check_window_mat_size(), copy_mat_to_surface_RGB(), destroy_window(), and setup_window().

3.5.2.27 window_texture

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

Definition at line 66 of file display.c.

Referenced by destroy_window().

3.5.2.28 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(), update(), and update_window().

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

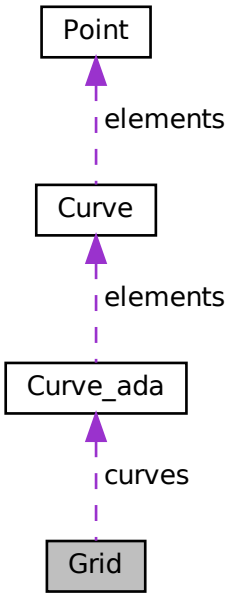
- src/include/display.c

3.6 Grid Struct Reference

Grid definition (as lines) in a chosen plane.

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

Collaboration diagram for Grid:



Public Attributes

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

3.6.1 Detailed Description

Grid definition (as lines) in a chosen plane.

Definition at line 203 of file Almog_Draw_Library.h.

3.6.2 Member Data Documentation

3.6.2.1 curves

```
Curve_ada Grid::curves
```

Line segments implementing the grid.

Definition at line 204 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create(), adl_grid_draw(), and ae_grid_project_world2screen().

3.6.2.2 de1

```
float Grid::de1
```

Step size along axis 1.

Definition at line 213 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.6.2.3 de2

```
float Grid::de2
```

Step size along axis 2.

Definition at line 214 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.6.2.4 max_e1

```
float Grid::max_e1
```

Axis 1 max.

Definition at line 207 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.6.2.5 max_e2

```
float Grid::max_e2
```

Axis 2 max.

Definition at line 209 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.6.2.6 min_e1

```
float Grid::min_e1
```

Axis 1 min.

Definition at line 206 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.6.2.7 min_e2

```
float Grid::min_e2
```

Axis 2 min.

Definition at line 208 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.6.2.8 num_samples_e1

```
int Grid::num_samples_e1
```

Number of divisions along axis 1.

Definition at line 211 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.6.2.9 num_samples_e2

```
int Grid::num_samples_e2
```

Number of divisions along axis 2.

Definition at line 212 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

3.6.2.10 plane

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

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

Definition at line 216 of file Almog_Draw_Library.h.

Referenced by adl_cartesian_grid_create().

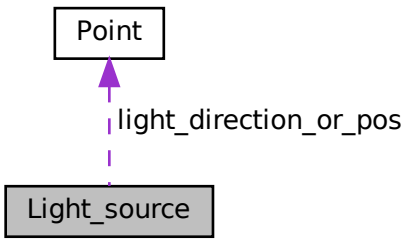
The documentation for this struct was generated from the following file:

- src/include/Almog_Draw_Library.h

3.7 Light_source Struct Reference

```
#include <Almog_Engine.h>
```

Collaboration diagram for Light_source:



Public Attributes

- Point light_direction_or_pos
- float light_intensity

3.7.1 Detailed Description

Definition at line 161 of file Almog_Engine.h.

3.7.2 Member Data Documentation

3.7.2.1 light_direction_or_pos

```
Point Light_source::light_direction_or_pos
```

Definition at line 162 of file Almog_Engine.h.

Referenced by ae_quad_calc_light_intensity(), ae_quad_project_world2screen(), ae_scene_init(), and ae_tri_calc_light_intensity().

3.7.2.2 light_intensity

```
float Light_source::light_intensity
```

Definition at line 163 of file Almog_Engine.h.

Referenced by ae_quad_calc_light_intensity(), ae_scene_init(), and ae_tri_calc_light_intensity().

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

- src/include/Almog_Engine.h

3.8 Mat2D Struct Reference

Dense row-major matrix of doubles.

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

Public Attributes

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

3.8.1 Detailed Description

Dense row-major matrix of doubles.

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

Definition at line 81 of file Matrix2D.h.

3.8.2 Member Data Documentation

3.8.2.1 cols

```
size_t Mat2D::cols
```

Definition at line 83 of file Matrix2D.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), ae_point_to_mat2D(), ae_projection_mat_set(), ae_quad_calc_normal(), ae_tri_calc_normal(), ae_z_buffer_copy_to_screen(), mat2D_add(), mat2D_add_col_to_col(), mat2D_add_row_time_factor_to_row(), mat2D_add_row_to_row(), mat2D_alloc(), mat2D_calc_norma(), mat2D_col_is_all_digit(), mat2D_copy(), mat2D_copy_mat_to_mat_at_window(), mat2D_cross(), mat2D_det(), mat2D_det_2x2_mat(), mat2D_dot(), mat2D_dot_product(), mat2D_fill(), mat2D_fill_sequence(), mat2D_get_col(), mat2D_get_row(), mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_make_identity(), mat2D_mat_is_all_digit(), mat2D_minor_alloc_fill_from_mat(), mat2D_mult(), mat2D_mult_row(), mat2D_offset2d(), mat2D_print(), mat2D_print_as_col(), mat2D_rand(), mat2D_row_is_all_digit(), mat2D_set_identity(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), mat2D_set_rot_mat_z(), mat2D_solve_linear_sys_LUP_decomposition(), mat2D_sub(), mat2D_sub_col_to_col(), mat2D_sub_row_time_factor_to_row(), mat2D_sub_row_to_row(), mat2D_swap_rows(), mat2D_transpose(), mat2D_triangulate(), and render_window().

3.8.2.2 elements

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

Definition at line 85 of file Matrix2D.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), mat2D_alloc(), mat2D_free(), mat2D_print_as_col(), and render_window().

3.8.2.3 rows

```
size_t Mat2D::rows
```

Definition at line 82 of file Matrix2D.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_curves_plot_on_figure(), adl_figure_alloc(), ae_point_to_mat2D(), ae_projection_mat_set(), ae_quad_calc_normal(), ae_tri_calc_normal(), ae_z_buffer_copy_to_screen(), mat2D_add(), mat2D_add_col_to_col(), mat2D_add_row_to_row(), mat2D_alloc(), mat2D_calc_norma(), mat2D_copy(), mat2D_copy_mat_to_mat_at_window(), mat2D_cross(), mat2D_det(), mat2D_det_2x2_mat(), mat2D_dot(), mat2D_dot_product(), mat2D_fill(), mat2D_fill_sequence(), mat2D_get_col(), mat2D_get_row(), mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_make_identity(), mat2D_mat_is_all_digit(), mat2D_minor_alloc_fill_from_mat(), mat2D_mult(), mat2D_offset2d(), mat2D_print(), mat2D_print_as_col(), mat2D_rand(), mat2D_set_identity(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), mat2D_set_rot_mat_z(), mat2D_solve_linear_sys_LUP_decomposition(), mat2D_sub(), mat2D_sub_col_to_col(), mat2D_sub_row_to_row(), mat2D_transpose(), mat2D_triangulate(), and render_window().

3.8.2.4 stride_r

```
size_t Mat2D::stride_r
```

Definition at line 84 of file Matrix2D.h.

Referenced by mat2D_alloc(), and mat2D_offset2d().

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

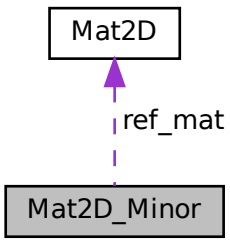
- src/include/Matrix2D.h

3.9 Mat2D_Minor Struct Reference

A minor "view" into a reference matrix.

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

Collaboration diagram for Mat2D_Minor:



Public Attributes

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

3.9.1 Detailed Description

A minor "view" into a reference matrix.

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

Memory ownership:

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

Definition at line 119 of file Matrix2D.h.

3.9.2 Member Data Documentation

3.9.2.1 cols

```
size_t Mat2D_Minor::cols
```

Definition at line 121 of file Matrix2D.h.

Referenced by mat2D_det_2x2_mat_minor(), mat2D_minor_alloc_fill_from_mat(), mat2D_minor_alloc_fill_from_mat_minor(), mat2D_minor_det(), and mat2D_minor_print().

3.9.2.2 cols_list

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

Definition at line 124 of file Matrix2D.h.

Referenced by mat2D_minor_alloc_fill_from_mat(), mat2D_minor_alloc_fill_from_mat_minor(), and mat2D_minor_free().

3.9.2.3 ref_mat

```
Mat2D Mat2D_Minor::ref_mat
```

Definition at line 125 of file Matrix2D.h.

Referenced by mat2D_minor_alloc_fill_from_mat(), and mat2D_minor_alloc_fill_from_mat_minor().

3.9.2.4 rows

```
size_t Mat2D_Minor::rows
```

Definition at line 120 of file Matrix2D.h.

Referenced by mat2D_det_2x2_mat_minor(), mat2D_minor_alloc_fill_from_mat(), mat2D_minor_alloc_fill_from_mat_minor(), mat2D_minor_det(), and mat2D_minor_print().

3.9.2.5 rows_list

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

Definition at line 123 of file Matrix2D.h.

Referenced by mat2D_minor_alloc_fill_from_mat(), mat2D_minor_alloc_fill_from_mat_minor(), and mat2D_minor_free().

3.9.2.6 stride_r

```
size_t Mat2D_Minor::stride_r
```

Definition at line 122 of file Matrix2D.h.

Referenced by mat2D_minor_alloc_fill_from_mat(), and mat2D_minor_alloc_fill_from_mat_minor().

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

- src/include/Matrix2D.h

3.10 Mat2D_uint32 Struct Reference

Dense row-major matrix of uint32_t.

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

Public Attributes

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

3.10.1 Detailed Description

Dense row-major matrix of uint32_t.

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

Definition at line 98 of file Matrix2D.h.

3.10.2 Member Data Documentation

3.10.2.1 cols

```
size_t Mat2D_uint32::cols
```

Definition at line 100 of file Matrix2D.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_axis_draw_on_figure(), adl_figure_alloc(), adl_figure_copy_to_screen(), adl_line_draw(), adl_point_draw(), adl_quad_fill(), adl_quad_fill_interpolate_color_mean_value(), adl_quad_fill_interpolate_normal_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(), and render_window().

3.10.2.2 elements

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

Definition at line 102 of file Matrix2D.h.

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

3.10.2.3 rows

```
size_t Mat2D_uint32::rows
```

Definition at line 99 of file Matrix2D.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_axis_draw_on_figure(), adl_figure_alloc(), adl_figure_copy_to_screen(), adl_line_draw(), adl_max_min_values_draw_on_figure(), adl_point_draw(), adl_quad_fill(), adl_quad_fill_interpolate_color_mean_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(), and render_window().

3.10.2.4 stride_r

```
size_t Mat2D_uint32::stride_r
```

Definition at line 101 of file Matrix2D.h.

Referenced by mat2D_alloc_uint32(), and mat2D_offset2d_uint32().

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

- src/include/Matrix2D.h

3.11 Material Struct Reference

```
#include <Almog_Engine.h>
```

Public Attributes

- float specular_power_alpha
- float c_ambi
- float c_diff
- float c_spec

3.11.1 Detailed Description

Definition at line 166 of file Almog_Engine.h.

3.11.2 Member Data Documentation

3.11.2.1 c_ambi

```
float Material::c_ambi
```

Definition at line 168 of file Almog_Engine.h.

Referenced by ae_quad_calc_light_intensity(), ae_scene_init(), and ae_tri_calc_light_intensity().

3.11.2.2 c_diff

```
float Material::c_diff
```

Definition at line 169 of file Almog_Engine.h.

Referenced by ae_quad_calc_light_intensity(), ae_scene_init(), and ae_tri_calc_light_intensity().

3.11.2.3 c_spec

```
float Material::c_spec
```

Definition at line 170 of file Almog_Engine.h.

Referenced by ae_quad_calc_light_intensity(), ae_scene_init(), and ae_tri_calc_light_intensity().

3.11.2.4 specular_power_alpha

```
float Material::specular_power_alpha
```

Definition at line 167 of file Almog_Engine.h.

Referenced by ae_quad_calc_light_intensity(), ae_scene_init(), and ae_tri_calc_light_intensity().

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

- src/include/Almog_Engine.h

3.12 Offset_zoom_param Struct Reference

Pan/zoom parameters relative to screen center.

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

Public Attributes

- float zoom_multiplier
- float offset_x
- float offset_y
- int mouse_x
- int mouse_y

3.12.1 Detailed Description

Pan/zoom parameters relative to screen center.

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

Definition at line 65 of file Almog_Draw_Library.h.

3.12.2 Member Data Documentation

3.12.2.1 mouse_x

```
int Offset_zoom_param::mouse_x
```

Optional: last mouse x (pixels).

Definition at line 69 of file Almog_Draw_Library.h.

3.12.2.2 mouse_y

```
int Offset_zoom_param::mouse_y
```

Optional: last mouse y (pixels).

Definition at line 70 of file Almog_Draw_Library.h.

3.12.2.3 offset_x

```
float Offset_zoom_param::offset_x
```

Horizontal pan offset (pixels).

Definition at line 67 of file Almog_Draw_Library.h.

Referenced by adl_line_draw(), and adl_point_draw().

3.12.2.4 offset_y

```
float Offset_zoom_param::offset_y
```

Vertical pan offset (pixels).

Definition at line 68 of file Almog_Draw_Library.h.

Referenced by adl_line_draw(), and adl_point_draw().

3.12.2.5 zoom_multiplier

```
float Offset_zoom_param::zoom_multiplier
```

Zoom scale factor (>0).

Definition at line 66 of file Almog_Draw_Library.h.

Referenced by adl_line_draw(), and adl_point_draw().

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

- src/include/Almog_Draw_Library.h

3.13 Point Struct Reference

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

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

Public Attributes

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

3.13.1 Detailed Description

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

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

Definition at line 81 of file Almog_Draw_Library.h.

3.13.2 Member Data Documentation

3.13.2.1 w

`float Point::w`

Homogeneous w.

Definition at line 85 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_cartesian_grid_create(), adl_quad_fill(), adl_quad_fill_interpolate_color_mean_v
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(), ae_point_normalize_xyz(), ae_point_project_view2screen(),
ae_point_project_world2view(), ae_quad_calc_light_intensity(), ae_quad_clip_with_plane(), ae_quad_get_average_normal(),
ae_quad_get_average_point(), ae_quad_transform_to_view(), ae_scene_init(), ae_tri_calc_light_intensity(),
ae_tri_clip_with_plane(), ae_tri_get_average_normal(), ae_tri_get_average_point(), ae_tri_project_world2screen(),
and ae_tri_transform_to_view().

3.13.2.2 x

`float Point::x`

X coordinate (pixels).

Definition at line 82 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_cartesian_grid_create(), adl_curve_add_to_figure(),
adl_curves_plot_on_figure(), adl_figure_copy_to_screen(), adl_quad_fill(), adl_quad_fill_interpolate_color_mean_value(),
adl_quad_fill_interpolate_normal_mean_value(), adl_tan_half_angle(), adl_tri_draw(), adl_tri_fill_Pinedas_rasterizer(),
adl_tri_fill_Pinedas_rasterizer_interpolate_color(), adl_tri_fill_Pinedas_rasterizer_interpolate_normal(), ae_mat2D_to_point(),
ae_point_normalize_xyz(), ae_point_project_view2screen(), ae_point_project_world2view(), ae_point_to_mat2D(),
ae_print_points(), ae_print_tri(), ae_quad_get_average_normal(), ae_quad_get_average_point(), ae_quad_project_world2screen(),
ae_quad_transform_to_view(), ae_scene_init(), ae_signed_dist_point_and_plane(), ae_tri_clip_with_plane(),
ae_tri_get_average_normal(), ae_tri_get_average_point(), ae_tri_mesh_get_from_obj_file(), ae_tri_mesh_get_from_stl_file(),
ae_tri_mesh_normalize(), ae_tri_mesh_rotate_Euler_xyz(), ae_tri_mesh_set_bounding_box(), ae_tri_mesh_translate(),
ae_tri_project_world2screen(), and ae_tri_transform_to_view().

3.13.2.3 y

`float Point::y`

Y coordinate (pixels).

Definition at line 83 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_cartesian_grid_create(), adl_curve_add_to_figure(),
adl_curves_plot_on_figure(), adl_figure_copy_to_screen(), adl_quad_fill(), adl_quad_fill_interpolate_color_mean_value(),
adl_quad_fill_interpolate_normal_mean_value(), adl_tan_half_angle(), adl_tri_draw(), adl_tri_fill_Pinedas_rasterizer(),
adl_tri_fill_Pinedas_rasterizer_interpolate_color(), adl_tri_fill_Pinedas_rasterizer_interpolate_normal(), ae_point_normalize_xyz(),
ae_point_project_view2screen(), ae_point_project_world2view(), ae_point_to_mat2D(), ae_print_points(),
ae_print_tri(), ae_quad_get_average_normal(), ae_quad_get_average_point(), ae_quad_project_world2screen(),
ae_quad_transform_to_view(), ae_scene_init(), ae_signed_dist_point_and_plane(), ae_tri_clip_with_plane(),
ae_tri_get_average_normal(), ae_tri_get_average_point(), ae_tri_mesh_get_from_obj_file(), ae_tri_mesh_get_from_stl_file(),
ae_tri_mesh_normalize(), ae_tri_mesh_rotate_Euler_xyz(), ae_tri_mesh_set_bounding_box(), ae_tri_mesh_translate(),
ae_tri_project_world2screen(), and ae_tri_transform_to_view().

3.13.2.4 z

```
float Point::z
```

Depth value.

Definition at line 84 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_cartesian_grid_create(), adl_quad_fill(), adl_quad_fill_interpolate_color_mean_v
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(), ae_point_normalize_xyz(), ae_point_project_view2screen(),
ae_point_project_world2view(), ae_point_to_mat2D(), ae_print_points(), ae_print_tri(), ae_quad_get_average_normal(),
ae_quad_get_average_point(), ae_quad_project_world2screen(), ae_quad_transform_to_view(), ae_scene_init(),
ae_signed_dist_point_and_plane(), ae_tri_compare(), ae_tri_get_average_normal(), ae_tri_get_average_point(),
ae_tri_mesh_get_from_obj_file(), ae_tri_mesh_get_from_stl_file(), ae_tri_mesh_normalize(), ae_tri_mesh_rotate_Euler_xyz(),
ae_tri_mesh_set_bounding_box(), ae_tri_mesh_translate(), ae_tri_project_world2screen(), and ae_tri_transform_to_view().

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

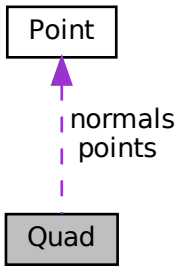
- src/include/Almog_Draw_Library.h

3.14 Quad Struct Reference

Quad primitive with optional per-vertex attributes.

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

Collaboration diagram for Quad:



Public Attributes

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

3.14.1 Detailed Description

Quad primitive with optional per-vertex attributes.

Definition at line 134 of file Almog_Draw_Library.h.

3.14.2 Member Data Documentation

3.14.2.1 colors

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

Optional per-vertex ARGB colors.

Definition at line 137 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_quad2tris(), adl_quad_fill_interpolate_color_mean_value(), ae_quad_clip_with_plane(), and ae_tri_mesh_get_from_quad_mesh().

3.14.2.2 light_intensity

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

Per-vertex light intensity multiplier.

Definition at line 139 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_quad2tris(), adl_quad_fill(), adl_quad_fill_interpolate_color_mean_value(), adl_quad_fill_interpolate_normal_mean_value(), ae_quad_calc_light_intensity(), ae_quad_project_world2screen(), and ae_tri_mesh_get_from_quad_mesh().

3.14.2.3 normals

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

Optional normals (unused here).

Definition at line 136 of file Almog_Draw_Library.h.

Referenced by ae_quad_calc_light_intensity(), ae_quad_get_average_normal(), ae_quad_project_world2screen(), ae_quad_set_normals(), and ae_tri_mesh_get_from_quad_mesh().

3.14.2.4 points

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

Quad vertices (0..3 order).

Definition at line 135 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_quad2tris(), adl_quad_draw(), adl_quad_fill(), adl_quad_fill_interpolate_color_m adl_quad_fill_interpolate_normal_mean_value(), ae_quad_calc_light_intensity(), ae_quad_calc_normal(), ae_quad_clip_with_plane(), ae_quad_get_average_point(), ae_quad_project_world2screen(), ae_quad_set_normals(), ae_quad_transform_to_view(), and ae_tri_mesh_get_from_quad_mesh().

3.14.2.5 to_draw

```
bool Quad::to_draw
```

Whether to include in rendering.

Definition at line 138 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), adl_quad2tris(), adl_quad_mesh_draw(), adl_quad_mesh_fill(), adl_quad_mesh_fill_interpolate_color(), adl_quad_mesh_fill_interpolate_normal(), ae_quad_project_world2screen(), and ae_tri_mesh_get_from_quad_mesh().

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

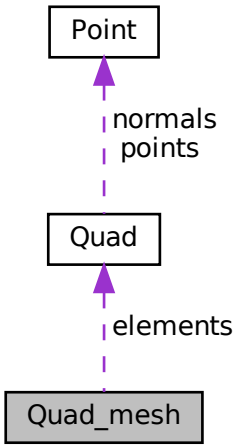
- src/include/Almog_Draw_Library.h

3.15 Quad_mesh Struct Reference

Dynamic array of quads (quad mesh).

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

Collaboration diagram for Quad_mesh:



Public Attributes

- size_t length
- size_t capacity
- Quad * elements

3.15.1 Detailed Description

Dynamic array of quads (quad mesh).

Definition at line 160 of file Almog_Draw_Library.h.

3.15.2 Member Data Documentation

3.15.2.1 capacity

```
size_t Quad_mesh::capacity
```

Allocated capacity.

Definition at line 162 of file Almog_Draw_Library.h.

3.15.2.2 elements

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

Quad array.

Definition at line 163 of file Almog_Draw_Library.h.

Referenced by adl_quad_mesh_draw(), adl_quad_mesh_fill(), adl_quad_mesh_fill_interpolate_color(), adl_quad_mesh_fill_interpolat
ae_quad_mesh_project_world2screen(), ae_quad_project_world2screen(), ae_scene_free(), and ae_tri_mesh_get_from_quad_mesh

3.15.2.3 length

```
size_t Quad_mesh::length
```

Number of quads used.

Definition at line 161 of file Almog_Draw_Library.h.

Referenced by adl_quad_mesh_draw(), adl_quad_mesh_fill(), adl_quad_mesh_fill_interpolate_color(), adl_quad_mesh_fill_interpolat
ae_quad_mesh_project_world2screen(), ae_quad_project_world2screen(), and ae_tri_mesh_get_from_quad_mesh().

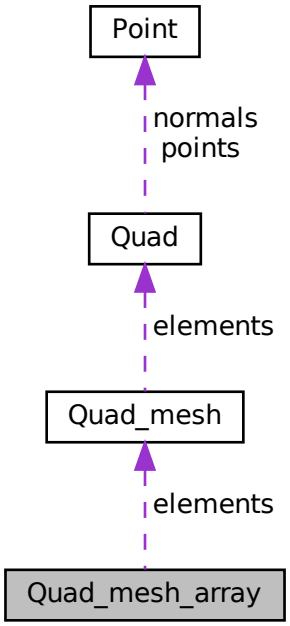
The documentation for this struct was generated from the following file:

- src/include/Almog_Draw_Library.h

3.16 Quad_mesh_array Struct Reference

`#include <Almog_Engine.h>`

Collaboration diagram for Quad_mesh_array:



Public Attributes

- size_t length
- size_t capacity
- Quad_mesh * elements

3.16.1 Detailed Description

Definition at line 137 of file Almog_Engine.h.

3.16.2 Member Data Documentation

3.16.2.1 capacity

```
size_t Quad_mesh_array::capacity
```

Definition at line 139 of file Almog_Engine.h.

3.16.2.2 elements

```
Quad_mesh* Quad_mesh_array::elements
```

Definition at line 140 of file Almog_Engine.h.

Referenced by ae_scene_free().

3.16.2.3 length

```
size_t Quad_mesh_array::length
```

Definition at line 138 of file Almog_Engine.h.

Referenced by ae_scene_free().

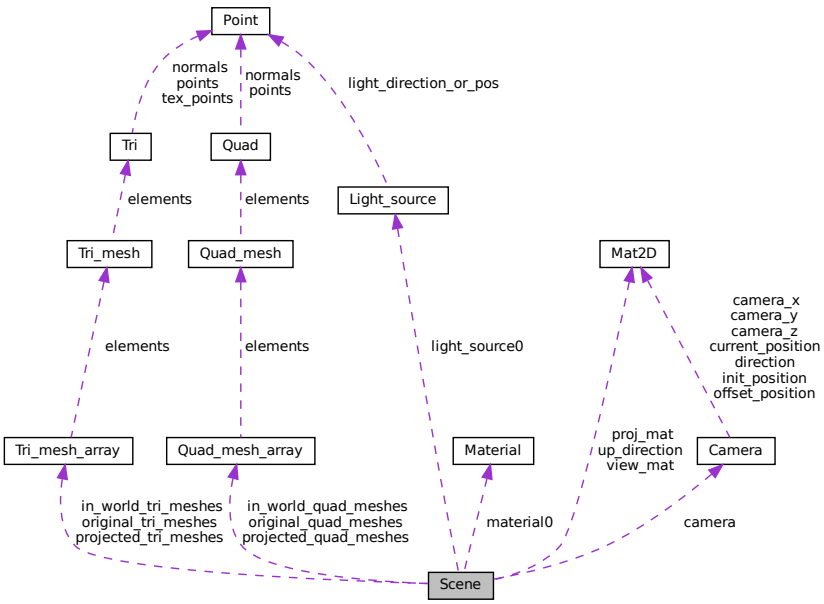
The documentation for this struct was generated from the following file:

- src/include/Almog_Engine.h

3.17 Scene Struct Reference

```
#include <Almog_Engine.h>
```

Collaboration diagram for Scene:



Public Attributes

- Tri_mesh_array in_world_tri_meshes
- Tri_mesh_array projected_tri_meshes
- Tri_mesh_array original_tri_meshes
- Quad_mesh_array in_world_quad_meshes
- Quad_mesh_array projected_quad_meshes
- Quad_mesh_array original_quad_meshes
- Camera camera
- Mat2D up_direction
- Mat2D proj_mat
- Mat2D view_mat
- Light_source light_source0
- Material material0

3.17.1 Detailed Description

Definition at line 173 of file Almog_Engine.h.

3.17.2 Member Data Documentation

3.17.2.1 camera

```
Camera Scene::camera
```

Definition at line 182 of file Almog_Engine.h.

Referenced by ae_camera_free(), ae_camera_init(), ae_camera_reset_pos(), ae_line_project_world2screen(), ae_quad_calc_light_intensity(), ae_quad_project_world2screen(), ae_scene_init(), ae_tri_calc_light_intensity(), ae_tri_project_world2screen(), check_window_mat_size(), process_input_window(), and update().

3.17.2.2 in_world_quad_meshes

```
Quad_mesh_array Scene::in_world_quad_meshes
```

Definition at line 178 of file Almog_Engine.h.

Referenced by ae_scene_free().

3.17.2.3 in_world_tri_meshes

```
Tri_mesh_array Scene::in_world_tri_meshes
```

Definition at line 174 of file Almog_Engine.h.

Referenced by ae_scene_free(), render(), setup(), and update().

3.17.2.4 light_source0

```
Light_source Scene::light_source0
```

Definition at line 187 of file Almog_Engine.h.

Referenced by ae_quad_calc_light_intensity(), ae_quad_project_world2screen(), ae_scene_init(), and ae_tri_calc_light_intensity().

3.17.2.5 material0

```
Material Scene::material0
```

Definition at line 188 of file Almog_Engine.h.

Referenced by ae_quad_calc_light_intensity(), ae_scene_init(), and ae_tri_calc_light_intensity().

3.17.2.6 original_quad_meshes

```
Quad_mesh_array Scene::original_quad_meshes
```

Definition at line 180 of file Almog_Engine.h.

Referenced by ae_scene_free().

3.17.2.7 original_tri_meshes

```
Tri_mesh_array Scene::original_tri_meshes
```

Definition at line 176 of file Almog_Engine.h.

Referenced by ae_scene_free(), and setup().

3.17.2.8 proj_mat

```
Mat2D Scene::proj_mat
```

Definition at line 184 of file Almog_Engine.h.

Referenced by ae_scene_free(), ae_scene_init(), and update().

3.17.2.9 projected_quad_meshes

```
Quad_mesh_array Scene::projected_quad_meshes
```

Definition at line 179 of file Almog_Engine.h.

Referenced by ae_scene_free().

3.17.2.10 projected_tri_meshes

```
Tri_mesh_array Scene::projected_tri_meshes
```

Definition at line 175 of file Almog_Engine.h.

Referenced by ae_scene_free(), render(), setup(), and update().

3.17.2.11 up_direction

```
Mat2D Scene::up_direction
```

Definition at line 183 of file Almog_Engine.h.

Referenced by ae_scene_free(), ae_scene_init(), and update().

3.17.2.12 view_mat

```
Mat2D Scene::view_mat
```

Definition at line 185 of file Almog_Engine.h.

Referenced by ae_scene_free(), ae_scene_init(), and update().

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

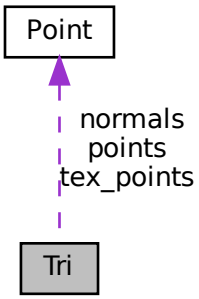
- src/include/Almog_Engine.h

3.18 Tri Struct Reference

Triangle primitive with optional per-vertex attributes.

`#include <Almog_Draw_Library.h>`

Collaboration diagram for Tri:



Public Attributes

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

3.18.1 Detailed Description

Triangle primitive with optional per-vertex attributes.

Definition at line 119 of file Almog_Draw_Library.h.

3.18.2 Member Data Documentation

3.18.2.1 colors

`uint32_t Tri::colors[3]`

Optional per-vertex ARGB colors.

Definition at line 123 of file Almog_Draw_Library.h.

Referenced by adl_quad2tris(), adl_tri_fill_Pinedas_rasterizer_interpolate_color(), ae_tri_clip_with_plane(), ae_tri_mesh_flip_normals(), ae_tri_mesh_get_from_obj_file(), ae_tri_mesh_get_from_quad_mesh(), and ae_tri_mesh_get_from_stl_file().

3.18.2.2 light_intensity

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

Per-vertex light intensity multiplier.

Definition at line 125 of file Almog_Draw_Library.h.

Referenced by adl_quad2tris(), adl_tri_fill_Pinedas_rasterizer(), adl_tri_fill_Pinedas_rasterizer_interpolate_color(), adl_tri_fill_Pinedas_rasterizer_interpolate_normal(), ae_tri_calc_light_intensity(), ae_tri_mesh_flip_normals(), ae_tri_mesh_get_from_obj_file(), ae_tri_mesh_get_from_quad_mesh(), ae_tri_mesh_get_from_stl_file(), and ae_tri_project_world2screen().

3.18.2.3 normals

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

Optional normals (unused here).

Definition at line 122 of file Almog_Draw_Library.h.

Referenced by ae_tri_calc_light_intensity(), ae_tri_get_average_normal(), ae_tri_mesh_flip_normals(), ae_tri_mesh_get_from_quad_
ae_tri_mesh_get_from_stl_file(), ae_tri_project_world2screen(), and ae_tri_set_normals().

3.18.2.4 points

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

Triangle vertices.

Definition at line 120 of file Almog_Draw_Library.h.

Referenced by adl_quad2tris(), adl_tri_draw(), adl_tri_fill_Pinedas_rasterizer(), adl_tri_fill_Pinedas_rasterizer_interpolate_color(), adl_tri_fill_Pinedas_rasterizer_interpolate_normal(), ae_print_tri(), ae_tri_calc_light_intensity(), ae_tri_calc_normal(), ae_tri_clip_with_plane(), ae_tri_compare(), ae_tri_create(), ae_tri_get_average_point(), ae_tri_mesh_flip_normals(), ae_tri_mesh_get_from_obj_file(), ae_tri_mesh_get_from_quad_mesh(), ae_tri_mesh_get_from_stl_file(), ae_tri_mesh_normalize(), ae_tri_mesh_rotate_Euler_xyz(), ae_tri_mesh_set_bounding_box(), ae_tri_mesh_translate(), ae_tri_project_world2screen(), ae_tri_set_normals(), and ae_tri_transform_to_view().

3.18.2.5 tex_points

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

Optional texture coordinates (unused here).

Definition at line 121 of file Almog_Draw_Library.h.

Referenced by ae_tri_clip_with_plane(), ae_tri_mesh_flip_normals(), and ae_tri_project_world2screen().

3.18.2.6 to_draw

```
bool Tri::to_draw
```

Whether to include in rendering.

Definition at line 124 of file Almog_Draw_Library.h.

Referenced by adl_quad2tris(), adl_tri_mesh_draw(), adl_tri_mesh_fill_Pinedas_rasterizer(), adl_tri_mesh_fill_Pinedas_rasterizer_int
adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal(), ae_print_tri(), ae_tri_mesh_flip_normals(), ae_tri_mesh_get_from_obj_file
ae_tri_mesh_get_from_quad_mesh(), ae_tri_mesh_get_from_stl_file(), and ae_tri_project_world2screen().

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

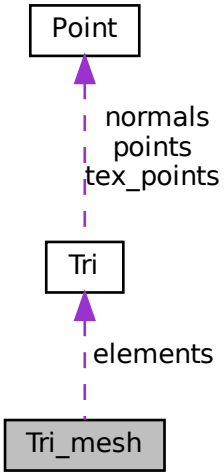
- src/include/Almog_Draw_Library.h

3.19 Tri_mesh Struct Reference

Dynamic array of triangles (triangle mesh).

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

Collaboration diagram for Tri_mesh:



Public Attributes

- size_t length
- size_t capacity
- Tri * elements

3.19.1 Detailed Description

Dynamic array of triangles (triangle mesh).

Definition at line 148 of file Almog_Draw_Library.h.

3.19.2 Member Data Documentation

3.19.2.1 capacity

```
size_t Tri_mesh::capacity
```

Allocated capacity.

Definition at line 150 of file Almog_Draw_Library.h.

3.19.2.2 elements

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

Triangle array.

Definition at line 151 of file Almog_Draw_Library.h.

Referenced by adl_tri_mesh_draw(), adl_tri_mesh_fill_Pinedas_rasterizer(), adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color(), adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal(), ae_print_tri_mesh(), ae_scene_free(), ae_tri_mesh_appand_copy(), ae_tri_mesh_flip_normals(), ae_tri_mesh_normalize(), ae_tri_mesh_project_world2screen(), ae_tri_mesh_rotate_Euler_xyz(), ae_tri_mesh_set_bounding_box(), ae_tri_mesh_set_normals(), ae_tri_mesh_translate(), and ae_tri_project_world2screen().

3.19.2.3 length

```
size_t Tri_mesh::length
```

Number of triangles used.

Definition at line 149 of file Almog_Draw_Library.h.

Referenced by adl_tri_mesh_draw(), adl_tri_mesh_fill_Pinedas_rasterizer(), adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color(), adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal(), ae_print_tri_mesh(), ae_tri_mesh_appand_copy(), ae_tri_mesh_create_copy(), ae_tri_mesh_flip_normals(), ae_tri_mesh_normalize(), ae_tri_mesh_project_world2screen(), ae_tri_mesh_rotate_Euler_xyz(), ae_tri_mesh_set_bounding_box(), ae_tri_mesh_set_normals(), ae_tri_mesh_translate(), ae_tri_project_world2screen(), render(), and setup().

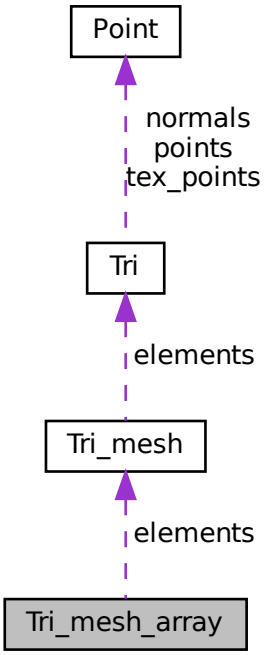
The documentation for this struct was generated from the following file:

- src/include/Almog_Draw_Library.h

3.20 Tri_mesh_array Struct Reference

`#include <Almog_Engine.h>`

Collaboration diagram for Tri_mesh_array:



Public Attributes

- size_t length
- size_t capacity
- Tri_mesh * elements

3.20.1 Detailed Description

Definition at line 128 of file Almog_Engine.h.

3.20.2 Member Data Documentation

3.20.2.1 capacity

```
size_t Tri_mesh_array::capacity
```

Definition at line 130 of file Almog_Engine.h.

3.20.2.2 elements

```
Tri_mesh* Tri_mesh_array::elements
```

Definition at line 131 of file Almog_Engine.h.

Referenced by ae_scene_free(), render(), setup(), and update().

3.20.2.3 length

```
size_t Tri_mesh_array::length
```

Definition at line 129 of file Almog_Engine.h.

Referenced by ae_scene_free(), render(), setup(), and update().

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

- src/include/Almog_Engine.h

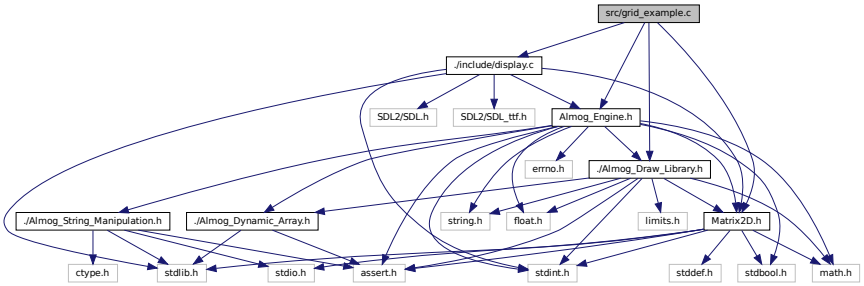
Chapter 4

File Documentation

4.1 src/grid_example.c File Reference

```
#include "./include/display.c"
#include "./include/Matrix2D.h"
#include "./include/Almog_Draw_Library.h"
#include "./include/Almog_Engine.h"
```

Include dependency graph for grid_example.c:



Macros

- #define SETUP
- #define UPDATE
- #define RENDER
- #define MATRIX2D_IMPLEMENTATION
- #define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
- #define ALMOG_ENGINE_IMPLEMENTATION

Functions

- void setup (game_state_t *game_state)
- void update (game_state_t *game_state)
- void render (game_state_t *game_state)

Variables

- Grid grid
- Grid grid_proj

4.1.1 Macro Definition Documentation

4.1.1.1 ALMOG_DRAW_LIBRARY_IMPLEMENTATION

```
#define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
```

Definition at line 7 of file grid_example.c.

4.1.1.2 ALMOG_ENGINE_IMPLEMENTATION

```
#define ALMOG_ENGINE_IMPLEMENTATION
```

Definition at line 9 of file grid_example.c.

4.1.1.3 MATRIX2D_IMPLEMENTATION

```
#define MATRIX2D_IMPLEMENTATION
```

Definition at line 5 of file grid_example.c.

4.1.1.4 RENDER

```
#define RENDER
```

Definition at line 3 of file grid_example.c.

4.1.1.5 SETUP

```
#define SETUP
```

Definition at line 1 of file grid_example.c.

4.1.1.6 UPDATE

```
#define UPDATE
```

Definition at line 2 of file grid_example.c.

4.1.2 Function Documentation

4.1.2.1 render()

```
void render (
            game_state_t * game_state )
```

Definition at line 32 of file grid_example.c.

References ADL_DEFAULT_OFFSET_ZOOM, adl_grid_draw(), grid_proj, and game_state_t::window_pixels_mat.

4.1.2.2 setup()

```
void setup (
            game_state_t * game_state )
```

Definition at line 14 of file grid_example.c.

References adl_cartesian_grid_create(), game_state_t::const_fps, grid, and grid_proj.

4.1.2.3 update()

```
void update (
            game_state_t * game_state )
```

Definition at line 23 of file grid_example.c.

References ae_grid_project_world2screen(), ae_projection_mat_set(), ae_view_mat_set(), Camera::aspect_ratio, Scene::camera, Camera::fov_deg, grid, grid_proj, Scene::proj_mat, game_state_t::scene, Scene::up_direction, Scene::view_mat, game_state_t::window_h, game_state_t::window_w, Camera::z_far, and Camera::z_near.

4.1.3 Variable Documentation

4.1.3.1 grid

```
Grid grid
```

Definition at line 12 of file grid_example.c.

Referenced by adl_cartesian_grid_create(), adl_grid_draw(), setup(), and update().

4.1.3.2 grid_proj

```
Grid grid_proj
```

Definition at line 13 of file grid_example.c.

Referenced by render(), setup(), and update().

4.2 grid_example.c

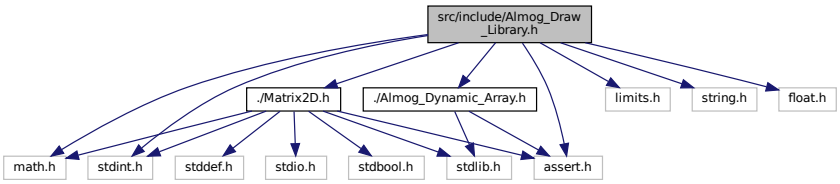
```
00001 #define SETUP
00002 #define UPDATE
00003 #define RENDER
00004 #include "./include/display.c"
00005 #define MATRIX2D_IMPLEMENTATION
00006 #include "./include/Matrix2D.h"
00007 #define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
00008 #include "./include/Almog_Draw_Library.h"
00009 #define ALMOG_ENGINE_IMPLEMENTATION
00010 #include "./include/Almog_Engine.h"
00011
00012 Grid grid;
00013 Grid grid_proj;
00014 void setup(game_state_t *game_state)
00015 {
00016     // game_state->to_limit_fps = 0;
00017     game_state->const_fps = 500;
00018
00019     grid      = adl_cartesian_grid_create(-1, 1, -2, 2, 10, 20, "XZ", 1);
00020     grid_proj = adl_cartesian_grid_create(-1, 1, -2, 2, 10, 20, "XZ", 1);
00021 }
00022
00023 void update(game_state_t *game_state)
00024 {
00025     ae_projection_mat_set(game_state->scene.proj_mat, game_state->scene.camera.aspect_ratio,
      game_state->scene.camera.fov_deg, game_state->scene.camera.z_near, game_state->scene.camera.z_far);
00026     ae_view_mat_set(game_state->scene.view_mat, game_state->scene.camera,
      game_state->scene.up_direction);
00027
00028     ae_grid_project_world2screen(game_state->scene.proj_mat, game_state->scene.view_mat, grid_proj,
      grid, game_state->window_w, game_state->window_h, &(game_state->scene));
00029
00030 }
00031
00032 void render(game_state_t *game_state)
00033 {
00034     adl_grid_draw(game_state->window_pixels_mat, grid_proj, 0xffffffff, ADL_DEFAULT_OFFSET_ZOOM);
00035
00036 }
```

4.3 src/include/Almog_Draw_Library.h File Reference

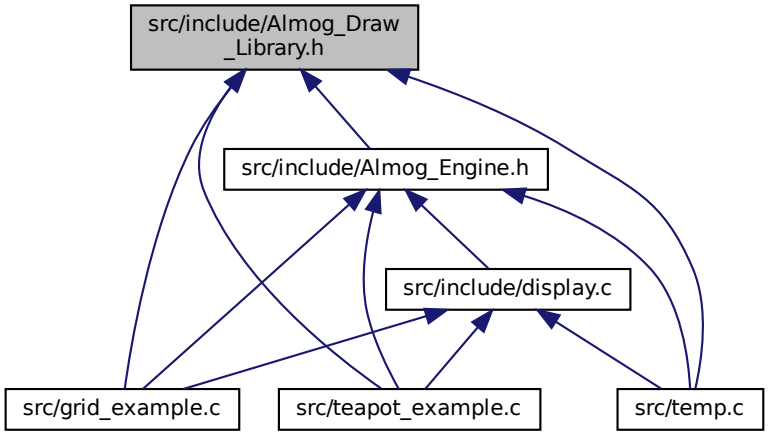
Immediate-mode 2D/3D raster helpers for drawing onto Mat2D_uint32 pixel buffers.

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

Include dependency graph for Almog_Draw_Library.h:



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



Classes

- struct Offset_zoom_param

  Pan/zoom parameters relative to screen center.
- struct Point

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

Polyline of points with a uniform color.

- struct Curve_ada

  Dynamic array of curves (polyline container).

- struct Tri

  Triangle primitive with optional per-vertex attributes.

- struct Quad

  Quad primitive with optional per-vertex attributes.

- struct Tri_mesh

  Dynamic array of triangles (triangle mesh).

- struct Quad_mesh

  Dynamic array of quads (quad mesh).

- struct Figure

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

- struct Grid

  Grid definition (as lines) in a chosen plane.

Macros

- #define ADL_ASSERT assert

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

- #define POINT
- #define CURVE
- #define CURVE_ADA
- #define TRI
- #define QUAD
- #define TRI_MESH
- #define QUAD_MESH
- #define HexARGB_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.

4.3.1 Detailed Description

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

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

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

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

Types Mat2D and Mat2D_uint32 are provided by Matrix2D.h.

Usage:

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

Note

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

Definition in file Almog_Draw_Library.h.

4.3.2 Macro Definition Documentation

4.3.2.1 ADL_ASSERT

```
#define ADL_ASSERT assert
```

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

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

Definition at line 55 of file Almog_Draw_Library.h.

4.3.2.2 adl_assert_point_is_valid

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

Definition at line 310 of file Almog_Draw_Library.h.

4.3.2.3 adl_assert_quad_is_valid

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

Value:

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

Definition at line 314 of file Almog_Draw_Library.h.

4.3.2.4 adl_assert_tri_is_valid

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

Value:

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

Definition at line 311 of file Almog_Draw_Library.h.

4.3.2.5 ADL_DEFAULT_OFFSET_ZOOM

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

Definition at line 331 of file Almog_Draw_Library.h.

4.3.2.6 ADL_FIGURE_AXIS_COLOR

```
#define ADL_FIGURE_AXIS_COLOR 0xff000000
```

Definition at line 324 of file Almog_Draw_Library.h.

4.3.2.7 ADL_FIGURE_HEAD_ANGLE_DEG

```
#define ADL_FIGURE_HEAD_ANGLE_DEG 30
```

Definition at line 323 of file Almog_Draw_Library.h.

4.3.2.8 ADL_FIGURE_PADDING_PRECENTAGE

```
#define ADL_FIGURE_PADDING_PRECENTAGE 20
```

Definition at line 319 of file Almog_Draw_Library.h.

4.3.2.9 ADL_MAX_CHARACTER_OFFSET

```
#define ADL_MAX_CHARACTER_OFFSET 10
```

Definition at line 326 of file Almog_Draw_Library.h.

4.3.2.10 ADL_MAX_FIGURE_PADDING

```
#define ADL_MAX_FIGURE_PADDING 70
```

Definition at line 320 of file Almog_Draw_Library.h.

4.3.2.11 ADL_MAX_HEAD_SIZE

```
#define ADL_MAX_HEAD_SIZE 15
```

Definition at line 322 of file Almog_Draw_Library.h.

4.3.2.12 ADL_MAX_POINT_VAL

```
#define ADL_MAX_POINT_VAL 1e5
```

Definition at line 309 of file Almog_Draw_Library.h.

4.3.2.13 ADL_MAX_SENTENCE_LEN

```
#define ADL_MAX_SENTENCE_LEN 256
```

Definition at line 328 of file Almog_Draw_Library.h.

4.3.2.14 ADL_MAX_ZOOM

```
#define ADL_MAX_ZOOM 1e3
```

Definition at line 329 of file Almog_Draw_Library.h.

4.3.2.15 ADL_MIN_CHARACTER_OFFSET

```
#define ADL_MIN_CHARACTER_OFFSET 5
```

Definition at line 327 of file Almog_Draw_Library.h.

4.3.2.16 ADL_MIN_FIGURE_PADDING

```
#define ADL_MIN_FIGURE_PADDING 20
```

Definition at line 321 of file Almog_Draw_Library.h.

4.3.2.17 adl_offset2d

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

Definition at line 2294 of file Almog_Draw_Library.h.

4.3.2.18 adl_offset_zoom_point

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

Value:

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

Definition at line 332 of file Almog_Draw_Library.h.

4.3.2.19 BLUE_hexARGB

```
#define BLUE_hexARGB 0xFF0000FF
```

Definition at line 299 of file Almog_Draw_Library.h.

4.3.2.20 CURVE

```
#define CURVE
```

Definition at line 90 of file Almog_Draw_Library.h.

4.3.2.21 CURVE_ADA

```
#define CURVE_ADA
```

Definition at line 103 of file Almog_Draw_Library.h.

4.3.2.22 CYAN_hexARGB

```
#define CYAN_hexARGB 0xFF00FFFF
```

Definition at line 301 of file Almog_Draw_Library.h.

4.3.2.23 edge_cross_point

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

Definition at line 304 of file Almog_Draw_Library.h.

4.3.2.24 GREEN_hexARGB

```
#define GREEN_hexARGB 0xFF00FF00
```

Definition at line 298 of file Almog_Draw_Library.h.

4.3.2.25 HexARGB_RGB_VAR

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

Definition at line 224 of file Almog_Draw_Library.h.

4.3.2.26 HexARGB_RGBA

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

Definition at line 221 of file Almog_Draw_Library.h.

4.3.2.27 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 227 of file Almog_Draw_Library.h.

4.3.2.28 is_left_edge

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

Definition at line 306 of file Almog_Draw_Library.h.

4.3.2.29 is_top_edge

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

Definition at line 305 of file Almog_Draw_Library.h.

4.3.2.30 is_top_left

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

Definition at line 307 of file Almog_Draw_Library.h.

4.3.2.31 POINT

```
#define POINT
```

Definition at line 74 of file Almog_Draw_Library.h.

4.3.2.32 PURPLE_hexARGB

```
#define PURPLE_hexARGB 0xFFFF00FF
```

Definition at line 300 of file Almog_Draw_Library.h.

4.3.2.33 QUAD

```
#define QUAD
```

Definition at line 130 of file Almog_Draw_Library.h.

4.3.2.34 QUAD_MESH

```
#define QUAD_MESH
```

Definition at line 156 of file Almog_Draw_Library.h.

4.3.2.35 RED_hexARGB

```
#define RED_hexARGB 0xFFFF0000
```

Definition at line 297 of file Almog_Draw_Library.h.

4.3.2.36 RGB_hexRGB

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

Definition at line 230 of file Almog_Draw_Library.h.

4.3.2.37 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 233 of file Almog_Draw_Library.h.

4.3.2.38 TRI

```
#define TRI
```

Definition at line 115 of file Almog_Draw_Library.h.

4.3.2.39 TRI_MESH

```
#define TRI_MESH
```

Definition at line 144 of file Almog_Draw_Library.h.

4.3.2.40 YELLOW_hexARGB

```
#define YELLOW_hexARGB 0xFFFFFF00
```

Definition at line 302 of file Almog_Draw_Library.h.

4.3.3 Function Documentation

4.3.3.1 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 2314 of file Almog_Draw_Library.h.

References adl_axis_draw_on_figure(), ADL_DEFAULT_OFFSET_ZOOM, adl_interpolate_ARGBcolor_on_okLch(), adl_linear_map(), adl_max_min_values_draw_on_figure(), adl_offset2d, adl_offset_zoom_point, adl_quad_fill_interpolate_color_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.

4.3.3.2 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 518 of file Almog_Draw_Library.h.

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

Referenced by adl_axis_draw_on_figure().

4.3.3.3 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 2144 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().

4.3.3.4 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 2513 of file Almog_Draw_Library.h.

References ada_append, ada_init_array, Grid::curves, Grid::de1, Grid::de2, grid, 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.

Referenced by setup().

4.3.3.5 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 586 of file Almog_Draw_Library.h.

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

Referenced by adl_sentence_draw().

4.3.3.6 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 1427 of file Almog_Draw_Library.h.

References adl_point_draw().

4.3.3.7 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 1449 of file Almog_Draw_Library.h.

References adl_point_draw().

4.3.3.8 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 2230 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.

4.3.3.9 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 2265 of file Almog_Draw_Library.h.

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

4.3.3.10 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 2081 of file Almog_Draw_Library.h.

References ada_init_array, ADL_ASSERT, adl_assert_point_is_valid, ADL_DEFAULT_OFFSET_ZOOM, ADL_FIGURE_PADDING_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.

4.3.3.11 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 2124 of file Almog_Draw_Library.h.

References adl_point_draw(), Mat2D_uint32::cols, MAT2D_AT_UINT32, Figure::pixels_mat, Mat2D_uint32::rows, Figure::top_left_position, Point::x, and Point::y.

4.3.3.12 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 2791 of file Almog_Draw_Library.h.

References adl_lines_draw(), Grid::curves, Curve::elements, Curve_ada::elements, grid, Curve::length, and Curve_ada::length.

Referenced by render().

4.3.3.13 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 2053 of file Almog_Draw_Library.h.

References adl_linear_sRGB_to_okLch(), and adl_okLch_to_linear_sRGB().

Referenced by adl_2Dscalar_interp_on_figure().

4.3.3.14 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 383 of file Almog_Draw_Library.h.

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

Referenced by adl_arrow_draw(), adl_character_draw(), adl_curves_plot_on_figure(), adl_lines_draw(), adl_lines_loop_draw(), adl_rectangle_draw_min_max(), adl_rectangle_fill_min_max(), and adl_tri_draw().

4.3.3.15 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 1865 of file Almog_Draw_Library.h.

Referenced by adl_2Dscalar_interp_on_figure(), and adl_curves_plot_on_figure().

4.3.3.16 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 1945 of file Almog_Draw_Library.h.

References HexARGB_RGB_VAR.

Referenced by adl_linear_sRGB_to_okLch().

4.3.3.17 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 2012 of file Almog_Draw_Library.h.

References adl_linear_sRGB_to_okLab(), and PI.

Referenced by adl_interpolate_ARGBcolor_on_okLch().

4.3.3.18 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 470 of file Almog_Draw_Library.h.

References adl_line_draw().

Referenced by adl_grid_draw().

4.3.3.19 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 490 of file Almog_Draw_Library.h.

References adl_line_draw().

Referenced by adl_quad_draw().

4.3.3.20 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 2170 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().

4.3.3.21 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 1980 of file Almog_Draw_Library.h.

References RGBA_hexARGB.

Referenced by adl_okLch_to_linear_sRGB().

4.3.3.22 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 2031 of file Almog_Draw_Library.h.

References adl_okLab_to_linear_sRGB(), and PI.

Referenced by adl_interpolate_ARGBcolor_on_okLch().

4.3.3.23 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 350 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().

4.3.3.24 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 1885 of file Almog_Draw_Library.h.

References Tri::colors, Quad::colors, Tri::light_intensity, Quad::light_intensity, Tri::points, Quad::points, Tri::to_draw, and Quad::to_draw.

4.3.3.25 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 1010 of file Almog_Draw_Library.h.

References adl_lines_loop_draw(), and Quad::points.

Referenced by adl_quad_mesh_draw().

4.3.3.26 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 1028 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().

4.3.3.27 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 1216 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(), and adl_quad_mesh_fill_interpolate_color().

4.3.3.28 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 1122 of file Almog_Draw_Library.h.

References adl_point_draw(), adl_tan_half_angle(), Mat2D_uint32::cols, edge_cross_point, HexARGB_RGBA_VAR, Quad::light_intensity, MAT2D_AT, Quad::points, RGBA_hexARGB, Mat2D_uint32::rows, Point::w, Point::x, Point::y, and Point::z.

Referenced by adl_quad_mesh_fill_interpolate_normal().

4.3.3.29 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 1320 of file Almog_Draw_Library.h.

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

4.3.3.30 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 1344 of file Almog_Draw_Library.h.

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

4.3.3.31 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 1401 of file Almog_Draw_Library.h.

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

4.3.3.32 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 1371 of file Almog_Draw_Library.h.

References adl_assert_quad_is_valid, adl_quad_fill_interpolate_normal_mean_value(), Quad_mesh::elements, HexARGB_RGBA_VAR, Quad_mesh::length, and Quad::to_draw.

4.3.3.33 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 973 of file Almog_Draw_Library.h.

References adl_line_draw().

Referenced by adl_character_draw().

4.3.3.34 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 992 of file Almog_Draw_Library.h.

References adl_line_draw().

Referenced by adl_character_draw().

4.3.3.35 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 949 of file Almog_Draw_Library.h.

References adl_character_draw(), ADL_MAX_CHARACTER_OFFSET, and ADL_MIN_CHARACTER_OFFSET.

Referenced by adl_max_min_values_draw_on_figure().

4.3.3.36 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 1845 of file Almog_Draw_Library.h.

References Point::x, and Point::y.

Referenced by adl_quad_fill_interpolate_color_mean_value(), and adl_quad_fill_interpolate_normal_mean_value().

4.3.3.37 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 1469 of file Almog_Draw_Library.h.

References adl_line_draw(), Tri::points, Point::x, and Point::y.

Referenced by adl_tri_mesh_draw().

4.3.3.38 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 1492 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, Point::w, Point::x, Point::y, and Point::z.

Referenced by adl_tri_mesh_fill_Pinedas_rasterizer().

4.3.3.39 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 1573 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, Point::w, Point::x, Point::y, and Point::z.

Referenced by adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color().

4.3.3.40 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 1664 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, Point::w, Point::x, Point::y, and Point::z.

Referenced by adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal().

4.3.3.41 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 1746 of file Almog_Draw_Library.h.

References adl_tri_draw(), Tri_mesh::elements, Tri_mesh::length, and Tri::to_draw.

4.3.3.42 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 1768 of file Almog_Draw_Library.h.

References adl_assert_tri_is_valid, adl_tri_fill_Pinedas_rasterizer(), Tri_mesh::elements, Tri_mesh::length, and Tri::to_draw.

4.3.3.43 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 1792 of file Almog_Draw_Library.h.

References adl_assert_tri_is_valid, adl_tri_fill_Pinedas_rasterizer_interpolate_color(), Tri_mesh::elements, Tri_mesh::length, and Tri::to_draw.

4.3.3.44 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 1817 of file Almog_Draw_Library.h.

References adl_assert_tri_is_valid, adl_tri_fill_Pinedas_rasterizer_interpolate_normal(), Tri_mesh::elements, Tri_mesh::length, and Tri::to_draw.

Referenced by render().

4.4 Almog_Draw_Library.h

```
00001
00034 #ifndef ALMOG_DRAW_LIBRARY_H_
00035 #define ALMOG_DRAW_LIBRARY_H_
00036
00037 #include <math.h>
00038 #include <stdint.h>
00039 #include <limits.h>
00040 #include <string.h>
00041 #include <float.h>
00042
00043 #include "./Matrix2D.h"
00044 #include "./Almog_Dynamic_Array.h"
00045
00053 #ifndef ADL_ASSERT
00054 #include <assert.h>
00055 #define ADL_ASSERT assert
00056 #endif
00057
00065 typedef struct {
00066     float zoom_multiplier;
00067     float offset_x;
00068     float offset_y;
00069     int mouse_x;
00070     int mouse_y;
00071 } Offset_zoom_param;
00072
00073 #ifndef POINT
00074 #define POINT
00081 typedef struct {
00082     float x;
00083     float y;
00084     float z;
00085     float w;
00086 } Point ;
00087 #endif
00088
00089 #ifndef CURVE
00090 #define CURVE
00094 typedef struct {
00095     uint32_t color;
00096     size_t length;
00097     size_t capacity;
00098     Point *elements;
00099 } Curve;
00100 #endif
00101
00102 #ifndef CURVE_ADA
00103 #define CURVE_ADA
00107 typedef struct {
00108     size_t length;
00109     size_t capacity;
00110     Curve *elements;
00111 } Curve_ada;
00112 #endif
00113
00114 #ifndef TRI
00115 #define TRI
00119 typedef struct {
00120     Point points[3];
00121     Point tex_points[3];
00122     Point normals[3];
00123     uint32_t colors[3];
00124     bool to_draw;
00125     float light_intensity[3];
00126 } Tri;
00127 #endif
00128
00129 #ifndef QUAD
00130 #define QUAD
00134 typedef struct {
00135     Point points[4];
00136     Point normals[4];
00137     uint32_t colors[4];
00138     bool to_draw;
00139     float light_intensity[4];
00140 } Quad;
00141 #endif
00142
00143 #ifndef TRI_MESH
```

```
00144 #define TRI_MESH
00148 typedef struct {
00149     size_t length;
00150     size_t capacity;
00151     Tri *elements;
00152 } Tri_mesh; /* Tri ada array */
00153 #endif
00154
00155 #ifndef QUAD_MESH
00156 #define QUAD_MESH
00160 typedef struct {
00161     size_t length;
00162     size_t capacity;
00163     Quad *elements;
00164 } Quad_mesh; /* Quad ada array */
00165 #endif
00166
00174 typedef struct {
00175     int min_x_pixel;
00176     int max_x_pixel;
00177     int min_y_pixel;
00178     int max_y_pixel;
00180     float min_x;
00181     float max_x;
00182     float min_y;
00183     float max_y;
00185     int x_axis_head_size;
00186     int y_axis_head_size;
00188     Offset_zoom_param offset_zoom_param;
00189     Curve_ada src_curve_array;
00190     Point top_left_position;
00192     Mat2D_uint32 pixels_mat;
00193     Mat2D inv_z_buffer_mat;
00195     uint32_t background_color;
00196     bool to_draw_axis;
00197     bool to_draw_max_min_values;
00198 } Figure;
00199
00203 typedef struct {
00204     Curve_ada curves;
00206     float min_e1;
00207     float max_e1;
00208     float min_e2;
00209     float max_e2;
00211     int num_samples_e1;
00212     int num_samples_e2;
00213     float de1;
00214     float de2;
00216     char plane[3];
00217 } Grid; /* direction: e1, e2 */
00218
00219
00220 #ifndef HexARGB_RGBA
00221 #define HexARGB_RGBA(x) ((x)>>(8*2)&0xFF), ((x)>>(8*1)&0xFF), ((x)>>(8*0)&0xFF), ((x)>>(8*3)&0xFF)
00222 #endif
00223 #ifndef HexARGB_RGB_VAR
00224 #define HexARGB_RGB_VAR(x, r, g, b) r = ((x)>>(8*2)&0xFF); g = ((x)>>(8*1)&0xFF); b = ((x)>>(8*0)&0xFF);
00225 #endif
00226 #ifndef HexARGB_RGBA_VAR
00227 #define HexARGB_RGBA_VAR(x, r, g, b, a) r = ((x)>>(8*2)&0xFF); g = ((x)>>(8*1)&0xFF); b =
      ((x)>>(8*0)&0xFF); a = ((x)>>(8*3)&0xFF)
00228 #endif
00229 #ifndef RGB_hexRGB
00230 #define RGB_hexRGB(r, g, b) (int)(0x010000*(int)(r) + 0x000100*(int)(g) + 0x000001*(int)(b))
00231 #endif
00232 #ifndef RGBA_hexARGB
00233 #define RGBA_hexARGB(r, g, b, a) (int)(0x01000000l*(int)(fminf(a, 255)) + 0x010000*(int)(r) +
      0x000100*(int)(g) + 0x000001*(int)(b))
00234 #endif
00235
00236
00237 void    adl_point_draw(Mat2D_uint32 screen_mat, int x, int y, uint32_t color, Offset_zoom_param
      offset_zoom_param);
00238 void    adl_line_draw(Mat2D_uint32 screen_mat, const float x1_input, const float y1_input, const float
      x2_input, const float y2_input, uint32_t color, Offset_zoom_param offset_zoom_param);
00239 void    adl_lines_draw(const Mat2D_uint32 screen_mat, const Point *points, const size_t len, const
      uint32_t color, Offset_zoom_param offset_zoom_param);
00240 void    adl_lines_loop_draw(const Mat2D_uint32 screen_mat, const Point *points, const size_t len,
      const uint32_t color, Offset_zoom_param offset_zoom_param);
00241 void    adl_arrow_draw(Mat2D_uint32 screen_mat, int xs, int ys, int xe, int ye, float head_size, float
      angle_deg, uint32_t color, Offset_zoom_param offset_zoom_param);
00242
00243 void    adl_character_draw(Mat2D_uint32 screen_mat, char c, int width_pixel, int hight_pixel, int
      x_top_left, int y_top_left, uint32_t color, Offset_zoom_param offset_zoom_param);
00244 void    adl_sentence_draw(Mat2D_uint32 screen_mat, const char sentence[], size_t len, const int
      x_top_left, const int y_top_left, const int hight_pixel, const uint32_t color, Offset_zoom_param
      offset_zoom_param);
```

```
00245
00246 void    adl_rectangle_draw_min_max(Mat2D_uint32 screen_mat, int min_x, int max_x, int min_y, int
       max_y, uint32_t color, Offset_zoom_param offset_zoom_param);
00247 void    adl_rectangle_fill_min_max(Mat2D_uint32 screen_mat, int min_x, int max_x, int min_y, int
       max_y, uint32_t color, Offset_zoom_param offset_zoom_param);
00248
00249 void    adl_quad_draw(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
       Offset_zoom_param offset_zoom_param);
00250 void    adl_quad_fill(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
       Offset_zoom_param offset_zoom_param);
00251 void    adl_quad_fill_interpolate_normal_mean_value(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad
       quad, uint32_t color, Offset_zoom_param offset_zoom_param);
00252 void    adl_quad_fill_interpolate_color_mean_value(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad
       quad, Offset_zoom_param offset_zoom_param);
00253
00254 void    adl_quad_mesh_draw(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Quad_mesh mesh, uint32_t
       color, Offset_zoom_param offset_zoom_param);
00255 void    adl_quad_mesh_fill(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Quad_mesh mesh, uint32_t
       color, Offset_zoom_param offset_zoom_param);
00256 void    adl_quad_mesh_fill_interpolate_normal(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat,
       Quad_mesh mesh, uint32_t color, Offset_zoom_param offset_zoom_param);
00257 void    adl_quad_mesh_fill_interpolate_color(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat,
       Quad_mesh mesh, Offset_zoom_param offset_zoom_param);
00258
00259 void    adl_circle_draw(Mat2D_uint32 screen_mat, float center_x, float center_y, float r, uint32_t
       color, Offset_zoom_param offset_zoom_param);
00260 void    adl_circle_fill(Mat2D_uint32 screen_mat, float center_x, float center_y, float r, uint32_t
       color, Offset_zoom_param offset_zoom_param);
00261
00262 void    adl_tri_draw(Mat2D_uint32 screen_mat, Tri tri, uint32_t color, Offset_zoom_param
       offset_zoom_param);
00263 void    adl_tri_fill_Pinedas_rasterizer(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Tri tri, uint32_t
       color, Offset_zoom_param offset_zoom_param);
00264 void    adl_tri_fill_Pinedas_rasterizer_interpolate_color(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer,
       Tri tri, Offset_zoom_param offset_zoom_param);
00265 void    adl_tri_fill_Pinedas_rasterizer_interpolate_normal(Mat2D_uint32 screen_mat, Mat2D
       inv_z_buffer, Tri tri, uint32_t color, Offset_zoom_param offset_zoom_param);
00266
00267 void    adl_tri_mesh_draw(Mat2D_uint32 screen_mat, Tri_mesh mesh, uint32_t color, Offset_zoom_param
       offset_zoom_param);
00268 void    adl_tri_mesh_fill_Pinedas_rasterizer(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Tri_mesh
       mesh, uint32_t color, Offset_zoom_param offset_zoom_param);
00269 void    adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color(Mat2D_uint32 screen_mat, Mat2D
       inv_z_buffer_mat, Tri_mesh mesh, Offset_zoom_param offset_zoom_param);
00270 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);
00271
00272 float   adl_tan_half_angle(Point vi, Point vj, Point p, float li, float lj);
00273 float   adl_linear_map(float s, float min_in, float max_in, float min_out, float max_out);
00274 void    adl_quad2tris(Quad quad, Tri *tri1, Tri *tri2, char split_line[]);
00275 void    adl_linear_sRGB_to_okLab(uint32_t hex_ARGB, float *L, float *a, float *b);
00276 void    adl_okLab_to_linear_sRGB(float L, float a, float b, uint32_t *hex_ARGB);
00277 void    adl_linear_sRGB_to_okLch(uint32_t hex_ARGB, float *L, float *c, float *h_deg);
00278 void    adl_okLch_to_linear_sRGB(float L, float c, float h_deg, uint32_t *hex_ARGB);
00279 void    adl_interpolate_ARGBcolor_on_okLch(uint32_t color1, uint32_t color2, float t, float
       num_of_rotations, uint32_t *color_out);
00280
00281 Figure  adl_figure_alloc(size_t rows, size_t cols, Point top_left_position);
00282 void    adl_figure_copy_to_screen(Mat2D_uint32 screen_mat, Figure figure);
00283 void    adl_axis_draw_on_figure(Figure *figure);
00284 void    adl_max_min_values_draw_on_figure(Figure figure);
00285 void    adl_curve_add_to_figure(Figure *figure, Point *src_points, size_t src_len, uint32_t color);
00286 void    adl_curves_plot_on_figure(Figure figure);
00287 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);
00288
00289 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);
00290 void    adl_grid_draw(Mat2D_uint32 screen_mat, Grid grid, uint32_t color, Offset_zoom_param
       offset_zoom_param);
00291
00292 #endif /*ALMOG_RENDER_SHAPES_H_*/
00293
00294 #ifdef ALMOG_DRAW_LIBRARY_IMPLEMENTATION
00295 #undef ALMOG_DRAW_LIBRARY_IMPLEMENTATION
00296
00297 #define RED_hexARGB    0xFFFF0000
00298 #define GREEN_hexARGB  0xFF00FF00
00299 #define BLUE_hexARGB   0xFF0000FF
00300 #define PURPLE_hexARGB 0xFFFF00FF
00301 #define CYAN_hexARGB   0xFF00FFFF
00302 #define YELLOW_hexARGB 0xFFFFFF00
00303
00304 #define edge_cross_point(a1, b, a2, p) (b.x-a1.x)*(p.y-a2.y)-(b.y-a1.y)*(p.x-a2.x)
00305 #define is_top_edge(x, y) (y == 0 && x > 0)
00306 #define is_left_edge(x, y) (y < 0)
00307 #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))
```

```
00308
00309 #define ADL_MAX_POINT_VAL 1e5
00310 #define adl_assert_point_is_valid(p) ADL_ASSERT(isfinite(p.x) && isfinite(p.y) && isfinite(p.z) &&
       isfinite(p.w))
00311 #define adl_assert_tri_is_valid(tri) adl_assert_point_is_valid(tri.points[0]); \
00312         adl_assert_point_is_valid(tri.points[1]);                                \
00313         adl_assert_point_is_valid(tri.points[2])
00314 #define adl_assert_quad_is_valid(quad) adl_assert_point_is_valid(quad.points[0]);   \
00315         adl_assert_point_is_valid(quad.points[1]);                                  \
00316         adl_assert_point_is_valid(quad.points[2]);                                  \
00317         adl_assert_point_is_valid(quad.points[3])
00318
00319 #define ADL_FIGURE_PADDING_PRECENTAGE 20
00320 #define ADL_MAX_FIGURE_PADDING 70
00321 #define ADL_MIN_FIGURE_PADDING 20
00322 #define ADL_MAX_HEAD_SIZE 15
00323 #define ADL_FIGURE_HEAD_ANGLE_DEG 30
00324 #define ADL_FIGURE_AXIS_COLOR 0xff000000
00325
00326 #define ADL_MAX_CHARACTER_OFFSET 10
00327 #define ADL_MIN_CHARACTER_OFFSET 5
00328 #define ADL_MAX_SENTENCE_LEN 256
00329 #define ADL_MAX_ZOOM 1e3
00330
00331 #define ADL_DEFAULT_OFFSET_ZOOM (Offset_zoom_param){1,0,0,0,0}
00332 #define adl_offset_zoom_point(p, window_w, window_h, offset_zoom_param)
                \
00333     (p).x = ((p).x - (window_w)/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier +
       (window_w)/2; \
00334     (p).y = ((p).y - (window_h)/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier +
       (window_h)/2
00335
00350 void adl_point_draw(Mat2D_uint32 screen_mat, int x, int y, uint32_t color, Offset_zoom_param
       offset_zoom_param)
00351 {
00352     float window_w = (float)screen_mat.cols;
00353     float window_h = (float)screen_mat.rows;
00354
00355     x = (x - window_w/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier +
       window_w/2;
00356     y = (y - window_h/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier +
       window_h/2;
00357
00358     if ((x < (int)screen_mat.cols && y < (int)screen_mat.rows) && (x >= 0 && y >= 0)) { /* point is in
       screen */
00359         uint8_t r_new, g_new, b_new, a_new;
00360         uint8_t r_current, g_current, b_current, a_current;
00361         HexARGB_RGBA_VAR(MAT2D_AT_UINT32(screen_mat, y, x), r_current, g_current, b_current,
       a_current);
00362         HexARGB_RGBA_VAR(color, r_new, g_new, b_new, a_new);
00363         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);
00364         (void)a_current;
00365     }
00366 }
00367
00383 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)
00384 {
00385     /* 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. */
00386
00387     float window_w = (float)screen_mat.cols;
00388     float window_h = (float)screen_mat.rows;
00389
00390     int x1 = (x1_input - window_w/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier
       + window_w/2;
00391     int x2 = (x2_input - window_w/2 + offset_zoom_param.offset_x) * offset_zoom_param.zoom_multiplier
       + window_w/2;
00392     int y1 = (y1_input - window_h/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier
       + window_h/2;
00393     int y2 = (y2_input - window_h/2 + offset_zoom_param.offset_y) * offset_zoom_param.zoom_multiplier
       + window_h/2;
00394
00395     ADL_ASSERT((int)fabsf(fabsf((float)x2) - fabsf((float)x1)) < ADL_MAX_POINT_VAL);
00396     ADL_ASSERT((int)fabsf(fabsf((float)y2) - fabsf((float)y1)) < ADL_MAX_POINT_VAL);
00397
00398     int x = x1;
00399     int y = y1;
00400     int dx, dy;
00401
00402     adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00403
00404     dx = x2 - x1;
00405     dy = y2 - y1;
```

```
00406
00407     ADL_ASSERT(dy > INT_MIN && dy < INT_MAX);
00408     ADL_ASSERT(dx > INT_MIN && dx < INT_MAX);
00409
00410     if (0 == dx && 0 == dy) return;
00411     if (0 == dx) {
00412         while (x != x2 || y != y2) {
00413             if (dy > 0) {
00414                 y++;
00415             }
00416             if (dy < 0) {
00417                 y--;
00418             }
00419             adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00420         }
00421         return;
00422     }
00423     if (0 == dy) {
00424         while (x != x2 || y != y2) {
00425             if (dx > 0) {
00426                 x++;
00427             }
00428             if (dx < 0) {
00429                 x--;
00430             }
00431             adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00432         }
00433         return;
00434     }
00435
00436     /* float m = (float)dy / dx */
00437     int b = y1 - dy * x1 / dx;
00438
00439     if (x1 > x2) {
00440         int temp_x = x1;
00441         x1 = x2;
00442         x2 = temp_x;
00443     }
00444     for (x = x1; x < x2; x++) {
00445         int sy1 = dy * x / dx + b;
00446         int sy2 = dy * (x + 1) / dx + b;
00447         if (sy1 > sy2) {
00448             int temp_y = sy1;
00449             sy1 = sy2;
00450             sy2 = temp_y;
00451         }
00452         for (y = sy1; y <= sy2; y++) {
00453             adl_point_draw(screen_mat, x, y, color, (Offset_zoom_param){1,0,0,0,0});
00454         }
00455     }
00456
00457 }
00458
00470 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)
00471 {
00472     if (len == 0) return;
00473     for (size_t i = 0; i < len-1; i++) {
00474         adl_line_draw(screen_mat, points[i].x, points[i].y, points[i+1].x, points[i+1].y, color,
       offset_zoom_param);
00475     }
00476 }
00477
00490 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)
00491 {
00492     if (len == 0) return;
00493     for (size_t i = 0; i < len-1; i++) {
00494         adl_line_draw(screen_mat, points[i].x, points[i].y, points[i+1].x, points[i+1].y, color,
       offset_zoom_param);
00495     }
00496     adl_line_draw(screen_mat, points[len-1].x, points[len-1].y, points[0].x, points[0].y, color,
       offset_zoom_param);
00497 }
00498
00499
00518 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)
00519 {
00520     Mat2D pe = mat2D_alloc(3, 1);
00521     mat2D_fill(pe, 0);
00522     MAT2D_AT(pe, 0, 0) = xe;
00523     MAT2D_AT(pe, 1, 0) = ye;
00524     Mat2D v1 = mat2D_alloc(3, 1);
00525     mat2D_fill(v1, 0);
00526     Mat2D v2 = mat2D_alloc(3, 1);
00527     mat2D_fill(v2, 0);
```

```
00528     Mat2D temp_v = mat2D_alloc(3, 1);
00529     mat2D_fill(temp_v, 0);
00530     Mat2D DCM_p = mat2D_alloc(3, 3);
00531     mat2D_fill(DCM_p, 0);
00532     mat2D_set_rot_mat_z(DCM_p, angle_deg);
00533     Mat2D DCM_m = mat2D_alloc(3, 3);
00534     mat2D_fill(DCM_m, 0);
00535     mat2D_set_rot_mat_z(DCM_m, -angle_deg);
00536 
00537     int x_center = xs*head_size + xe*(1-head_size);
00538     int y_center = ys*head_size + ye*(1-head_size);
00539 
00540     MAT2D_AT(v1, 0, 0) = x_center;
00541     MAT2D_AT(v1, 1, 0) = y_center;
00542     mat2D_copy(v2, v1);
00543 
00544     /* v1 */
00545     mat2D_copy(temp_v, v1);
00546     mat2D_sub(temp_v, pe);
00547     mat2D_fill(v1, 0);
00548     mat2D_dot(v1, DCM_p, temp_v);
00549     mat2D_add(v1, pe);
00550 
00551     /* v2 */
00552     mat2D_copy(temp_v, v2);
00553     mat2D_sub(temp_v, pe);
00554     mat2D_fill(v2, 0);
00555     mat2D_dot(v2, DCM_m, temp_v);
00556     mat2D_add(v2, pe);
00557 
00558     adl_line_draw(screen_mat, MAT2D_AT(v1, 0, 0), MAT2D_AT(v1, 1, 0), xe, ye, color,
      offset_zoom_param);
00559     adl_line_draw(screen_mat, MAT2D_AT(v2, 0, 0), MAT2D_AT(v2, 1, 0), xe, ye, color,
      offset_zoom_param);
00560     adl_line_draw(screen_mat, xs, ys, xe, ye, color, offset_zoom_param);
00561 
00562     mat2D_free(pe);
00563     mat2D_free(v1);
00564     mat2D_free(v2);
00565     mat2D_free(temp_v);
00566     mat2D_free(DCM_p);
00567     mat2D_free(DCM_m);
00568 }
00569 
00586 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)
00587 {
00588     switch (c)
00589     {
00590     case 'a':
00591     case 'A':
00592         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);
00593         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);
00594         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);
00595         break;
00596     case 'b':
00597     case 'B':
00598         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00599         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);
00600         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);
00601         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);
00602         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);
00603 
00604         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);
00605 
00606         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);
00607         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);
00608         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);
00609         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);
00610         break;
00611     case 'c':
00612     case 'C':
00613         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);
```

```
00614         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);
00615         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);
00616         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);
00617         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);
00618         break;
00619     case 'd':
00620     case 'D':
00621         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);
00622         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);
00623         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);
00624         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);
00625         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);
00626         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left, y_top_left, color,
      offset_zoom_param);
00627         break;
00628     case 'e':
00629     case 'E':
00630         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left, y_top_left, color,
      offset_zoom_param);
00631         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00632         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);
00633
00634         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);
00635         break;
00636     case 'f':
00637     case 'F':
00638         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left, y_top_left, color,
      offset_zoom_param);
00639         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00640
00641         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);
00642         break;
00643     case 'g':
00644     case 'G':
00645         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);
00646         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);
00647         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);
00648         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);
00649         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);
00650         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);
00651         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);
00652         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);
00653         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);
00654         break;
00655     case 'h':
00656     case 'H':
00657         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00658         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);
00659         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);
00660         break;
00661     case 'i':
00662     case 'I':
00663         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
      offset_zoom_param);
00664         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);
00665         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);
00666         break;
00667     case 'j':
00668     case 'J':
```

```
00669         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, 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+2*width_pixel/3,
      y_top_left+5*hight_pixel/6, color, offset_zoom_param);
00671         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);
00672         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);
00673         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);
00674         break;
00675     case 'k':
00676     case 'K':
00677         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00678         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);
00679         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);
00680         break;
00681     case 'l':
00682     case 'L':
00683         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00684         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);
00685         break;
00686     case 'm':
00687     case 'M':
00688         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left, y_top_left, color,
      offset_zoom_param);
00689         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);
00690         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);
00691         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);
00692         break;
00693     case 'n':
00694     case 'N':
00695         adl_line_draw(screen_mat, x_top_left, y_top_left+hight_pixel, x_top_left, y_top_left, color,
      offset_zoom_param);
00696         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);
00697         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);
00698         break;
00699     case 'o':
00700     case 'O':
00701         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);
00702         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);
00703         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);
00704         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);
00705         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);
00706         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);
00707         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);
00708         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);
00709         break;
00710     case 'p':
00711     case 'P':
00712         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00713         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);
00714         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);
00715         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);
00716         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);
00717
00718         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);
00719         break;
00720     case 'q':
00721     case 'Q':
00722         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);
00723         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);
00724         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);
00725         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);
00726         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);
00727         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);
00728         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);
00729         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);
00730
00731         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);
00732         break;
00733      case 'r':
00734      case 'R':
00735         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel, color,
      offset_zoom_param);
00736         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);
00737         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);
00738         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);
00739         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);
00740
00741         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);
00742
00743         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);
00744         break;
00745      case 's':
00746      case 'S':
00747         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);
00748         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);
00749         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);
00750
00751         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);
00752         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);
00753         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);
00754         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);
00755
00756         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);
00757         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);
00758         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);
00759         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);
00760         break;
00761      case 't':
00762      case 'T':
00763         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
      offset_zoom_param);
00764         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);
00765         break;
00766      case 'u':
00767      case 'U':
00768         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel/6, color,
      offset_zoom_param);
00769         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);
00770         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);
00771         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);
00772         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);
00773         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);
00774         break;
00775      case 'v':
00776      case 'V':
```

```
00777         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);
00778         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);
00779         break;
00780     case 'w':
00781     case 'W':
00782         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);
00783         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);
00784         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);
00785         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);
00786         break;
00787     case 'x':
00788     case 'X':
00789         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);
00790         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);
00791         break;
00792     case 'y':
00793     case 'Y':
00794         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);
00795         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);
00796         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);
00797         break;
00798     case 'z':
00799     case 'Z':
00800         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
      offset_zoom_param);
00801         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);
00802         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);
00803         break;
00804     case '.':
00805         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);
00806         break;
00807     case ':':
00808         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);
00809         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);
00810         break;
00811     case '0':
00812         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);
00813         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);
00814         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);
00815         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);
00816         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);
00817         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);
00818         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);
00819         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);
00820
00821         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);
00822         break;
00823     case '1':
00824         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);
00825         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);
00826         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);
00827         break;
00828     case '2':
00829         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);
00830         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);
00831         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);
```

```
00832         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);
00833         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);
00834         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);
00835         break;
00836     case '3':
00837         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);
00838         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);
00839         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);
00840         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);
00841         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);
00842
00843         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);
00844
00845         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);
00846         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);
00847         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);
00848         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);
00849         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);
00850         break;
00851     case '4':
00852         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);
00853         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);
00854         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);
00855         break;
00856     case '5':
00857         adl_line_draw(screen_mat, x_top_left+width_pixel, y_top_left, x_top_left, y_top_left, color,
      offset_zoom_param);
00858         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left, y_top_left+hight_pixel/2, color,
      offset_zoom_param);
00859
00860         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);
00861         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);
00862
00863         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);
00864         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);
00865         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);
00866         break;
00867     case '6':
00868         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);
00869         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);
00870         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);
00871
00872         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);
00873         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);
00874         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);
00875         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);
00876         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);
00877         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);
00878         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);
00879         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);
00880         break;
00881     case '7':
00882         adl_line_draw(screen_mat, x_top_left, y_top_left, x_top_left+width_pixel, y_top_left, color,
      offset_zoom_param);
```

```
00883         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);
00884         break;
00885     case '8':
00886         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);
00887         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);
00888         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);
00889         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);
00890         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);
00891
00892         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);
00893         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);
00894         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);
00895         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);
00896
00897         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);
00898         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);
00899         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);
00900         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);
00901         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);
00902         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);
00903         break;
00904     case '9':
00905         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);
00906         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);
00907         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);
00908         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);
00909         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);
00910         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);
00911         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);
00912         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);
00913         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);
00914         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);
00915         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);
00916         break;
00917     case '-':
00918         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);
00919         break;
00920     case '+':
00921         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);
00922         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);
00923         break;
00924     case ' ':
00925         break;
00926     default:
00927         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);
00928         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);
00929         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);
00930         break;
00931     }
00932 }
00933
00949 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)
```

```
00950 {
00951     int character_width_pixel = hight_pixel/2;
00952     int current_x_top_left = x_top_left;
00953     int character_x_offset = (int)fmaxf(fminf(ADL_MAX_CHARACTER_OFFSET, character_width_pixel / 5),
      ADL_MIN_CHARACTER_OFFSET);
00954 
00955     for (size_t char_index = 0; char_index < len; char_index++) {
00956         adl_character_draw(screen_mat, sentence[char_index], character_width_pixel, hight_pixel,
      current_x_top_left, y_top_left, color, offset_zoom_param);
00957         current_x_top_left += character_width_pixel + character_x_offset;
00958     }
00959 
00960 }
00961 
00973 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)
00974 {
00975     adl_line_draw(screen_mat, min_x, min_y, max_x, min_y, color, offset_zoom_param);
00976     adl_line_draw(screen_mat, min_x, max_y, max_x, max_y, color, offset_zoom_param);
00977     adl_line_draw(screen_mat, min_x, min_y, min_x, max_y, color, offset_zoom_param);
00978     adl_line_draw(screen_mat, max_x, min_y, max_x, max_y, color, offset_zoom_param);
00979 }
00980 
00992 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)
00993 {
00994     for (int y = min_y; y <= max_y; y++) {
00995         adl_line_draw(screen_mat, min_x, y, max_x, y, color, offset_zoom_param);
00996     }
00997 }
00998 
01010 void adl_quad_draw(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
      Offset_zoom_param offset_zoom_param)
01011 {
01012     (void)inv_z_buffer;
01013     adl_lines_loop_draw(screen_mat, quad.points, 4, color, offset_zoom_param);
01014 }
01015 
01028 void adl_quad_fill(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad quad, uint32_t color,
      Offset_zoom_param offset_zoom_param)
01029 {
01030     Point p0 = quad.points[0];
01031     Point p1 = quad.points[1];
01032     Point p2 = quad.points[2];
01033     Point p3 = quad.points[3];
01034 
01035     int x_min = fminf(p0.x, fminf(p1.x, fminf(p2.x, p3.x)));
01036     int x_max = fmaxf(p0.x, fmaxf(p1.x, fmaxf(p2.x, p3.x)));
01037     int y_min = fminf(p0.y, fminf(p1.y, fminf(p2.y, p3.y)));
01038     int y_max = fmaxf(p0.y, fmaxf(p1.y, fmaxf(p2.y, p3.y)));
01039 
01040     if (x_min < 0) x_min = 0;
01041     if (y_min < 0) y_min = 0;
01042     if (x_max >= (int)screen_mat.cols) x_max = (int)screen_mat.cols - 1;
01043     if (y_max >= (int)screen_mat.rows) y_max = (int)screen_mat.rows - 1;
01044 
01045     float w = edge_cross_point(p0, p1, p1, p2) + edge_cross_point(p2, p3, p3, p0);
01046     if (fabs(w) < 1e-6) {
01047         // adl_quad_draw(screen_mat, inv_z_buffer, quad, quad.colors[0], offset_zoom_param);
01048         return;
01049     }
01050 
01051     float size_p3_to_p0 = sqrt((p0.x - p3.x)*(p0.x - p3.x) + (p0.y - p3.y)*(p0.y - p3.y));
01052     float size_p0_to_p1 = sqrt((p1.x - p0.x)*(p1.x - p0.x) + (p1.y - p0.y)*(p1.y - p0.y));
01053     float size_p1_to_p2 = sqrt((p2.x - p1.x)*(p2.x - p1.x) + (p2.y - p1.y)*(p2.y - p1.y));
01054     float size_p2_to_p3 = sqrt((p3.x - p2.x)*(p3.x - p2.x) + (p3.y - p2.y)*(p3.y - p2.y));
01055 
01056     int r, g, b, a;
01057     HexARGB_RGBA_VAR(color, r, g, b, a);
01058     float light_intensity = (quad.light_intensity[0] + quad.light_intensity[1] +
      quad.light_intensity[2] + quad.light_intensity[3]) / 4;
01059     uint8_t base_r = (uint8_t)fmaxf(0, fminf(255, r * light_intensity));
01060     uint8_t base_g = (uint8_t)fmaxf(0, fminf(255, g * light_intensity));
01061     uint8_t base_b = (uint8_t)fmaxf(0, fminf(255, b * light_intensity));
01062 
01063     for (int y = y_min; y <= y_max; y++) {
01064         for (int x = x_min; x <= x_max; x++) {
01065             Point p = {.x = x, .y = y, .z = 0};
01066             bool in_01, in_12, in_23, in_30;
01067 
01068             in_01 = (edge_cross_point(p0, p1, p0, p) >= 0) != (w < 0);
01069             in_12 = (edge_cross_point(p1, p2, p1, p) >= 0) != (w < 0);
01070             in_23 = (edge_cross_point(p2, p3, p2, p) >= 0) != (w < 0);
01071             in_30 = (edge_cross_point(p3, p0, p3, p) >= 0) != (w < 0);
01072 
01073             /* https://www.mn.uio.no/math/english/people/aca/michaelf/papers/mv3d.pdf. */
01074             float size_p_to_p0 = sqrt((p0.x - p.x)*(p0.x - p.x) + (p0.y - p.y)*(p0.y - p.y));
```

```
01075             float size_p_to_p1 = sqrt((p1.x - p.x)*(p1.x - p.x) + (p1.y - p.y)*(p1.y - p.y));
01076             float size_p_to_p2 = sqrt((p2.x - p.x)*(p2.x - p.x) + (p2.y - p.y)*(p2.y - p.y));
01077             float size_p_to_p3 = sqrt((p3.x - p.x)*(p3.x - p.x) + (p3.y - p.y)*(p3.y - p.y));
01078 
01079             /* tangent of half the angle directly using vector math */
01080             float tan_theta_3_over_2 = size_p3_to_p0 / (size_p_to_p3 + size_p_to_p0);
01081             float tan_theta_0_over_2 = size_p0_to_p1 / (size_p_to_p0 + size_p_to_p1);
01082             float tan_theta_1_over_2 = size_p1_to_p2 / (size_p_to_p1 + size_p_to_p2);
01083             float tan_theta_2_over_2 = size_p2_to_p3 / (size_p_to_p2 + size_p_to_p3);
01084             float w0 = (tan_theta_3_over_2 + tan_theta_0_over_2) / size_p_to_p0;
01085             float w1 = (tan_theta_0_over_2 + tan_theta_1_over_2) / size_p_to_p1;
01086             float w2 = (tan_theta_1_over_2 + tan_theta_2_over_2) / size_p_to_p2;
01087             float w3 = (tan_theta_2_over_2 + tan_theta_3_over_2) / size_p_to_p3;
01088 
01089             float inv_w_tot = 1.0f / (w0 + w1 + w2 + w3);
01090             float alpha = w0 * inv_w_tot;
01091             float beta  = w1 * inv_w_tot;
01092             float gamma = w2 * inv_w_tot;
01093             float delta = w3 * inv_w_tot;
01094 
01095             if (in_01 && in_12 && in_23 && in_30) {
01096 
01097                 double inv_w = alpha * (1.0f / p0.w) + beta  * (1.0f / p1.w) + gamma * (1.0f / p2.w) +
      delta * (1.0f / p3.w);
01098                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
      p2.w) + delta * (p3.z / p3.w);
01099                 double inv_z = inv_w / z_over_w;
01100 
01101                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01102                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(base_r, base_g, base_b, a),
      offset_zoom_param);
01103                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01104                 }
01105             }
01106         }
01107     }
01108 }
01109 
01122 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)
01123 {
01124     Point p0 = quad.points[0];
01125     Point p1 = quad.points[1];
01126     Point p2 = quad.points[2];
01127     Point p3 = quad.points[3];
01128 
01129     int x_min = fminf(p0.x, fminf(p1.x, fminf(p2.x, p3.x)));
01130     int x_max = fmaxf(p0.x, fmaxf(p1.x, fmaxf(p2.x, p3.x)));
01131     int y_min = fminf(p0.y, fminf(p1.y, fminf(p2.y, p3.y)));
01132     int y_max = fmaxf(p0.y, fmaxf(p1.y, fmaxf(p2.y, p3.y)));
01133 
01134     if (x_min < 0) x_min = 0;
01135     if (y_min < 0) y_min = 0;
01136     if (x_max >= (int)screen_mat.cols) x_max = (int)screen_mat.cols - 1;
01137     if (y_max >= (int)screen_mat.rows) y_max = (int)screen_mat.rows - 1;
01138 
01139     float w = edge_cross_point(p0, p1, p1, p2) + edge_cross_point(p2, p3, p3, p0);
01140     if (fabs(w) < 1e-6) {
01141         // adl_quad_draw(screen_mat, inv_z_buffer, quad, quad.colors[0], offset_zoom_param);
01142         return;
01143     }
01144 
01145     int r, g, b, a;
01146     HexARGB_RGBA_VAR(color, r, g, b, a);
01147 
01148     for (int y = y_min; y <= y_max; y++) {
01149         for (int x = x_min; x <= x_max; x++) {
01150             Point p = {.x = x, .y = y, .z = 0};
01151             bool in_01, in_12, in_23, in_30;
01152 
01153             in_01 = (edge_cross_point(p0, p1, p0, p) >= 0) != (w < 0);
01154             in_12 = (edge_cross_point(p1, p2, p1, p) >= 0) != (w < 0);
01155             in_23 = (edge_cross_point(p2, p3, p2, p) >= 0) != (w < 0);
01156             in_30 = (edge_cross_point(p3, p0, p3, p) >= 0) != (w < 0);
01157 
01158             /* using 'mean value coordinates'
01159              * https://www.mn.uio.no/math/english/people/aca/michaelf/papers/mv3d.pdf. */
01160             float size_p_to_p0 = sqrt((p0.x - p.x)*(p0.x - p.x) + (p0.y - p.y)*(p0.y - p.y));
01161             float size_p_to_p1 = sqrt((p1.x - p.x)*(p1.x - p.x) + (p1.y - p.y)*(p1.y - p.y));
01162             float size_p_to_p2 = sqrt((p2.x - p.x)*(p2.x - p.x) + (p2.y - p.y)*(p2.y - p.y));
01163             float size_p_to_p3 = sqrt((p3.x - p.x)*(p3.x - p.x) + (p3.y - p.y)*(p3.y - p.y));
01164 
01165             /* calculating the tangent of half the angle directly using vector math */
01166             float t0 = adl_tan_half_angle(p0, p1, p, size_p_to_p0, size_p_to_p1);
01167             float t1 = adl_tan_half_angle(p1, p2, p, size_p_to_p1, size_p_to_p2);
01168             float t2 = adl_tan_half_angle(p2, p3, p, size_p_to_p2, size_p_to_p3);
01169             float t3 = adl_tan_half_angle(p3, p0, p, size_p_to_p3, size_p_to_p0);
```

```
01170
01171             float w0 = (t3 + t0) / size_p_to_p0;
01172             float w1 = (t0 + t1) / size_p_to_p1;
01173             float w2 = (t1 + t2) / size_p_to_p2;
01174             float w3 = (t2 + t3) / size_p_to_p3;
01175
01176             float inv_w_tot = 1.0f / (w0 + w1 + w2 + w3);
01177             float alpha = w0 * inv_w_tot;
01178             float beta  = w1 * inv_w_tot;
01179             float gamma = w2 * inv_w_tot;
01180             float delta = w3 * inv_w_tot;
01181
01182             if (in_01 && in_12 && in_23 && in_30) {
01183                 float light_intensity = quad.light_intensity[0]*alpha + quad.light_intensity[1]*beta +
      quad.light_intensity[2]*gamma + quad.light_intensity[3]*delta;
01184
01185                 float rf = r * light_intensity;
01186                 float gf = g * light_intensity;
01187                 float bf = b * light_intensity;
01188                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01189                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01190                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01191
01192                 double inv_w = alpha * (1.0f / p0.w) + beta  * (1.0f / p1.w) + gamma * (1.0f / p2.w) +
      delta * (1.0f / p3.w);
01193                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
      p2.w) + delta * (p3.z / p3.w);
01194                 double inv_z = inv_w / z_over_w;
01195
01196                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01197                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, a), offset_zoom_param);
01198                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01199                 }
01200             }
01201         }
01202     }
01203 }
01204
01216 void adl_quad_fill_interpolate_color_mean_value(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer, Quad
      quad, Offset_zoom_param offset_zoom_param)
01217 {
01218     Point p0 = quad.points[0];
01219     Point p1 = quad.points[1];
01220     Point p2 = quad.points[2];
01221     Point p3 = quad.points[3];
01222
01223     int x_min = fminf(p0.x, fminf(p1.x, fminf(p2.x, p3.x)));
01224     int x_max = fmaxf(p0.x, fmaxf(p1.x, fmaxf(p2.x, p3.x)));
01225     int y_min = fminf(p0.y, fminf(p1.y, fminf(p2.y, p3.y)));
01226     int y_max = fmaxf(p0.y, fmaxf(p1.y, fmaxf(p2.y, p3.y)));
01227
01228     if (x_min < 0) x_min = 0;
01229     if (y_min < 0) y_min = 0;
01230     if (x_max >= (int)screen_mat.cols) x_max = (int)screen_mat.cols - 1;
01231     if (y_max >= (int)screen_mat.rows) y_max = (int)screen_mat.rows - 1;
01232
01233     float w = edge_cross_point(p0, p1, p1, p2) + edge_cross_point(p2, p3, p3, p0);
01234     if (fabs(w) < 1e-6) {
01235         // adl_quad_draw(screen_mat, inv_z_buffer, quad, quad.colors[0], offset_zoom_param);
01236         return;
01237     }
01238
01239     for (int y = y_min; y <= y_max; y++) {
01240         for (int x = x_min; x <= x_max; x++) {
01241             Point p = {.x = x, .y = y, .z = 0};
01242             bool in_01, in_12, in_23, in_30;
01243
01244             in_01 = (edge_cross_point(p0, p1, p0, p) >= 0) != (w < 0);
01245             in_12 = (edge_cross_point(p1, p2, p1, p) >= 0) != (w < 0);
01246             in_23 = (edge_cross_point(p2, p3, p2, p) >= 0) != (w < 0);
01247             in_30 = (edge_cross_point(p3, p0, p3, p) >= 0) != (w < 0);
01248
01249             /* using 'mean value coordinates'
01250              * https://www.mn.uio.no/math/english/people/aca/michaelf/papers/mv3d.pdf. */
01251             float size_p_to_p0 = sqrt((p0.x - p.x)*(p0.x - p.x) + (p0.y - p.y)*(p0.y - p.y));
01252             float size_p_to_p1 = sqrt((p1.x - p.x)*(p1.x - p.x) + (p1.y - p.y)*(p1.y - p.y));
01253             float size_p_to_p2 = sqrt((p2.x - p.x)*(p2.x - p.x) + (p2.y - p.y)*(p2.y - p.y));
01254             float size_p_to_p3 = sqrt((p3.x - p.x)*(p3.x - p.x) + (p3.y - p.y)*(p3.y - p.y));
01255
01256             /* calculating the tangent of half the angle directly using vector math */
01257             float t0 = adl_tan_half_angle(p0, p1, p, size_p_to_p0, size_p_to_p1);
01258             float t1 = adl_tan_half_angle(p1, p2, p, size_p_to_p1, size_p_to_p2);
01259             float t2 = adl_tan_half_angle(p2, p3, p, size_p_to_p2, size_p_to_p3);
01260             float t3 = adl_tan_half_angle(p3, p0, p, size_p_to_p3, size_p_to_p0);
01261
01262             float w0 = (t3 + t0) / size_p_to_p0;
01263             float w1 = (t0 + t1) / size_p_to_p1;
```

```
01264             float w2 = (t1 + t2) / size_p_to_p2;
01265             float w3 = (t2 + t3) / size_p_to_p3;
01266 
01267             float inv_w_tot = 1.0f / (w0 + w1 + w2 + w3);
01268             float alpha = w0 * inv_w_tot;
01269             float beta  = w1 * inv_w_tot;
01270             float gamma = w2 * inv_w_tot;
01271             float delta = w3 * inv_w_tot;
01272 
01273             if (in_01 && in_12 && in_23 && in_30) {
01274                 int r0, g0, b0, a0;
01275                 int r1, g1, b1, a1;
01276                 int r2, g2, b2, a2;
01277                 int r3, g3, b3, a3;
01278                 HexARGB_RGBA_VAR(quad.colors[0], r0, g0, b0, a0);
01279                 HexARGB_RGBA_VAR(quad.colors[1], r1, g1, b1, a1);
01280                 HexARGB_RGBA_VAR(quad.colors[2], r2, g2, b2, a2);
01281                 HexARGB_RGBA_VAR(quad.colors[3], r3, g3, b3, a3);
01282 
01283                 uint8_t current_r = r0*alpha + r1*beta + r2*gamma + r3*delta;
01284                 uint8_t current_g = g0*alpha + g1*beta + g2*gamma + g3*delta;
01285                 uint8_t current_b = b0*alpha + b1*beta + b2*gamma + b3*delta;
01286                 uint8_t current_a = a0*alpha + a1*beta + a2*gamma + a3*delta;
01287 
01288                 float light_intensity = (quad.light_intensity[0] + quad.light_intensity[1] +
      quad.light_intensity[2] + quad.light_intensity[3]) / 4;
01289                 float rf = current_r * light_intensity;
01290                 float gf = current_g * light_intensity;
01291                 float bf = current_b * light_intensity;
01292                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01293                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01294                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01295 
01296                 double inv_w = alpha * (1.0f / p0.w) + beta  * (1.0f / p1.w) + gamma * (1.0f / p2.w) +
      delta * (1.0f / p3.w);
01297                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
      p2.w) + delta * (p3.z / p3.w);
01298                 double inv_z = inv_w / z_over_w;
01299 
01300                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01301                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, current_a),
      offset_zoom_param);
01302                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01303                 }
01304             }
01305         }
01306     }
01307 }
01308 
01320 void adl_quad_mesh_draw(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Quad_mesh mesh, uint32_t
      color, Offset_zoom_param offset_zoom_param)
01321 {
01322     for (size_t i = 0; i < mesh.length; i++) {
01323         Quad quad = mesh.elements[i];
01324         /* Reject invalid quad */
01325         adl_assert_quad_is_valid(quad);
01326 
01327         if (!quad.to_draw) continue;
01328 
01329         adl_quad_draw(screen_mat, inv_z_buffer_mat, quad, color, offset_zoom_param);
01330     }
01331 }
01332 
01344 void adl_quad_mesh_fill(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Quad_mesh mesh, uint32_t
      color, Offset_zoom_param offset_zoom_param)
01345 {
01346     for (size_t i = 0; i < mesh.length; i++) {
01347         Quad quad = mesh.elements[i];
01348         /* Reject invalid quad */
01349         adl_assert_quad_is_valid(quad);
01350 
01351         if (!quad.to_draw) continue;
01352 
01353         // color = rand_double() * 0xFFFFFFFF;
01354 
01355         adl_quad_fill(screen_mat, inv_z_buffer_mat, quad, color, offset_zoom_param);
01356     }
01357 }
01358 
01371 void adl_quad_mesh_fill_interpolate_normal(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer_mat, Quad_mesh
      mesh, uint32_t color, Offset_zoom_param offset_zoom_param)
01372 {
01373     for (size_t i = 0; i < mesh.length; i++) {
01374         Quad quad = mesh.elements[i];
01375         /* Reject invalid quad */
01376         adl_assert_quad_is_valid(quad);
01377 
```

```
01378         uint8_t a, r, g, b;
01379         HexARGB_RGBA_VAR(color, a, r, g, b);
01380         (void)r;
01381         (void)g;
01382         (void)b;
01383 
01384         if (!quad.to_draw && a == 255) continue;
01385 
01386         adl_quad_fill_interpolate_normal_mean_value(screen_mat, inv_z_buffer_mat, quad, color,
      offset_zoom_param);
01387     }
01388 }
01389 
01401 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)
01402 {
01403     for (size_t i = 0; i < mesh.length; i++) {
01404         Quad quad = mesh.elements[i];
01405         /* Reject invalid quad */
01406         adl_assert_quad_is_valid(quad);
01407 
01408         if (!quad.to_draw) continue;
01409 
01410         adl_quad_fill_interpolate_color_mean_value(screen_mat, inv_z_buffer_mat, quad,
      offset_zoom_param);
01411     }
01412 }
01413 
01427 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)
01428 {
01429     for (int dy = -r; dy <= r; dy++) {
01430         for (int dx = -r; dx <= r; dx ++) {
01431             float diff = dx * dx + dy * dy - r * r;
01432             if (diff < 0 && diff > -r*2) {
01433                 adl_point_draw(screen_mat, center_x + dx, center_y + dy, color, offset_zoom_param);
01434             }
01435         }
01436     }
01437 }
01438 
01449 void adl_circle_fill(Mat2D_uint32 screen_mat, float center_x, float center_y, float r, uint32_t color,
      Offset_zoom_param offset_zoom_param)
01450 {
01451     for (int dy = -r; dy <= r; dy++) {
01452         for (int dx = -r; dx <= r; dx ++) {
01453             float diff = dx * dx + dy * dy - r * r;
01454             if (diff < 0) {
01455                 adl_point_draw(screen_mat, center_x + dx, center_y + dy, color, offset_zoom_param);
01456             }
01457         }
01458     }
01459 }
01460 
01469 void adl_tri_draw(Mat2D_uint32 screen_mat, Tri tri, uint32_t color, Offset_zoom_param
      offset_zoom_param)
01470 {
01471     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);
01472     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);
01473     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);
01474 
01475     // 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);
01476     // 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);
01477     // 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);
01478 }
01479 
01492 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)
01493 {
01494     /* This function follows the rasterizer of 'Pikuma' shown in his YouTube video. You can fine the
      video in this link: https://youtu.be/k5wtuKWmV48. */
01495 
01496     Point p0, p1, p2;
01497     p0 = tri.points[0];
01498     p1 = tri.points[1];
01499     p2 = tri.points[2];
01500 
01501     /* finding bounding box */
01502     int x_min = fmin(p0.x, fmin(p1.x, p2.x));
01503     int x_max = fmax(p0.x, fmax(p1.x, p2.x));
01504     int y_min = fmin(p0.y, fmin(p1.y, p2.y));
```

```
01505     int y_max = fmax(p0.y, fmax(p1.y, p2.y));
01506
01507     /* Clamp to screen bounds */
01508     if (x_min < 0) x_min = 0;
01509     if (y_min < 0) y_min = 0;
01510     if (x_max >= (int)screen_mat.cols) x_max = screen_mat.cols - 1;
01511     if (y_max >= (int)screen_mat.rows) y_max = screen_mat.rows - 1;
01512
01513     /* draw only outline of the tri if there is no area */
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(fabsf(w) > 1e-6 && "triangle must have 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     for (int y = y_min; y <= y_max; y++) {
01527         for (int x = x_min; x <= x_max; x++) {
01528             Point p = {.x = x, .y = y, .z = 0};
01529
01530             float w0 = edge_cross_point(p0, p1, p0, p) + bias0;
01531             float w1 = edge_cross_point(p1, p2, p1, p) + bias1;
01532             float w2 = edge_cross_point(p2, p0, p2, p) + bias2;
01533
01534             float alpha = fabs(w1 / w);
01535             float beta  = fabs(w2 / w);
01536             float gamma = fabs(w0 / w);
01537
01538             if (w0 * w >= 0 && w1 * w >= 0 &&  w2 * w >= 0) {
01539                 int r, b, g, a;
01540                 HexARGB_RGBA_VAR(color, r, g, b, a);
01541                 float light_intensity = (tri.light_intensity[0] + tri.light_intensity[1] +
      tri.light_intensity[2]) / 3;
01542                 float rf = r * light_intensity;
01543                 float gf = g * light_intensity;
01544                 float bf = b * light_intensity;
01545                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01546                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01547                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01548
01549                 double inv_w = alpha * (1.0 / p0.w) + beta  * (1.0 / p1.w) + gamma * (1.0 / p2.w);
01550                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
      p2.w);
01551                 double inv_z = inv_w / z_over_w;
01552
01553                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01554                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, a), offset_zoom_param);
01555                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01556                 }
01557             }
01558         }
01559     }
01560 }
01561
01573 void adl_tri_fill_Pinedas_rasterizer_interpolate_color(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer,
      Tri tri, Offset_zoom_param offset_zoom_param)
01574 {
01575     /* This function follows the rasterizer of 'Pikuma' shown in his YouTube video. You can fine the
      video in this link: https://youtu.be/k5wtuKWmV48. */
01576     Point p0, p1, p2;
01577     p0 = tri.points[0];
01578     p1 = tri.points[1];
01579     p2 = tri.points[2];
01580
01581     float w = edge_cross_point(p0, p1, p1, p2);
01582     if (fabsf(w) < 1e-6) {
01583         // adl_tri_draw(screen_mat, tri, tri.colors[0], offset_zoom_param);
01584         return;
01585     }
01586     MATRIX2D_ASSERT(w != 0 && "triangle has area");
01587
01588     /* fill conventions */
01589     int bias0 = is_top_left(p0, p1) ? 0 : -1;
01590     int bias1 = is_top_left(p1, p2) ? 0 : -1;
01591     int bias2 = is_top_left(p2, p0) ? 0 : -1;
01592
01593     /* finding bounding box */
01594     int x_min = fmin(p0.x, fmin(p1.x, p2.x));
01595     int x_max = fmax(p0.x, fmax(p1.x, p2.x));
01596     int y_min = fmin(p0.y, fmin(p1.y, p2.y));
01597     int y_max = fmax(p0.y, fmax(p1.y, p2.y));
01598     // printf("xmin: %d, xmax: %d || ymin: %d, ymax: %d\n", x_min, x_max, y_min, y_max);
```

```
01599
01600     /* Clamp to screen bounds */
01601     if (x_min < 0) x_min = 0;
01602     if (y_min < 0) y_min = 0;
01603     if (x_max >= (int)screen_mat.cols) x_max = screen_mat.cols - 1;
01604     if (y_max >= (int)screen_mat.rows) y_max = screen_mat.rows - 1;
01605
01606     for (int y = y_min; y <= y_max; y++) {
01607         for (int x = x_min; x <= x_max; x++) {
01608             Point p = {.x = x, .y = y, .z = 0};
01609
01610             float w0 = edge_cross_point(p0, p1, p0, p) + bias0;
01611             float w1 = edge_cross_point(p1, p2, p1, p) + bias1;
01612             float w2 = edge_cross_point(p2, p0, p2, p) + bias2;
01613
01614             float alpha = fabs(w1 / w);
01615             float beta  = fabs(w2 / w);
01616             float gamma = fabs(w0 / w);
01617
01618             if (w0 * w >= 0 && w1 * w >= 0 &&  w2 * w >= 0) {
01619                 int r0, b0, g0, a0;
01620                 int r1, b1, g1, a1;
01621                 int r2, b2, g2, a2;
01622                 HexARGB_RGBA_VAR(tri.colors[0], r0, g0, b0, a0);
01623                 HexARGB_RGBA_VAR(tri.colors[1], r1, g1, b1, a1);
01624                 HexARGB_RGBA_VAR(tri.colors[2], r2, g2, b2, a2);
01625
01626                 uint8_t current_r = r0*alpha + r1*beta + r2*gamma;
01627                 uint8_t current_g = g0*alpha + g1*beta + g2*gamma;
01628                 uint8_t current_b = b0*alpha + b1*beta + b2*gamma;
01629                 uint8_t current_a = a0*alpha + a1*beta + a2*gamma;
01630
01631                 float light_intensity = (tri.light_intensity[0] + tri.light_intensity[1] +
    tri.light_intensity[2]) / 3;
01632                 float rf = current_r * light_intensity;
01633                 float gf = current_g * light_intensity;
01634                 float bf = current_b * light_intensity;
01635                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01636                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01637                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01638
01639                 double inv_w = alpha * (1.0 / p0.w) + beta  * (1.0 / p1.w) + gamma * (1.0 / p2.w);
01640                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
    p2.w);
01641                 double inv_z = inv_w / z_over_w;
01642
01643                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01644                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, current_a),
    offset_zoom_param);
01645                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01646                 }
01647             }
01648         }
01649     }
01650 }
01651
01664 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)
01665 {
01666     /* This function follows the rasterizer of 'Pikuma' shown in his YouTube video. You can fine the
    video in this link: https://youtu.be/k5wtuKWmV48. */
01667     Point p0, p1, p2;
01668     p0 = tri.points[0];
01669     p1 = tri.points[1];
01670     p2 = tri.points[2];
01671
01672     float w = edge_cross_point(p0, p1, p1, p2);
01673     if (fabsf(w) < 1e-6) {
01674         // adl_tri_draw(screen_mat, tri, tri.colors[0], offset_zoom_param);
01675         return;
01676     }
01677     MATRIX2D_ASSERT(w != 0 && "triangle has area");
01678
01679     /* fill conventions */
01680     int bias0 = is_top_left(p0, p1) ? 0 : -1;
01681     int bias1 = is_top_left(p1, p2) ? 0 : -1;
01682     int bias2 = is_top_left(p2, p0) ? 0 : -1;
01683
01684     /* finding bounding box */
01685     int x_min = fmin(p0.x, fmin(p1.x, p2.x));
01686     int x_max = fmax(p0.x, fmax(p1.x, p2.x));
01687     int y_min = fmin(p0.y, fmin(p1.y, p2.y));
01688     int y_max = fmax(p0.y, fmax(p1.y, p2.y));
01689     // printf("xmin: %d, xmax: %d || ymin: %d, ymax: %d\n", x_min, x_max, y_min, y_max);
01690
01691     /* Clamp to screen bounds */
01692     if (x_min < 0) x_min = 0;
```

```
01693     if (y_min < 0) y_min = 0;
01694     if (x_max >= (int)screen_mat.cols) x_max = screen_mat.cols - 1;
01695     if (y_max >= (int)screen_mat.rows) y_max = screen_mat.rows - 1;
01696 
01697     int r, b, g, a;
01698     HexARGB_RGBA_VAR(color, r, g, b, a);
01699 
01700     for (int y = y_min; y <= y_max; y++) {
01701         for (int x = x_min; x <= x_max; x++) {
01702             Point p = {.x = x, .y = y, .z = 0};
01703 
01704             float w0 = edge_cross_point(p0, p1, p0, p) + bias0;
01705             float w1 = edge_cross_point(p1, p2, p1, p) + bias1;
01706             float w2 = edge_cross_point(p2, p0, p2, p) + bias2;
01707 
01708             float alpha = fabs(w1 / w);
01709             float beta  = fabs(w2 / w);
01710             float gamma = fabs(w0 / w);
01711 
01712             if (w0 * w >= 0 && w1 * w >= 0 &&  w2 * w >= 0) {
01713 
01714                 float light_intensity = tri.light_intensity[0]*alpha + tri.light_intensity[1]*beta +
      tri.light_intensity[2]*gamma;
01715 
01716                 float rf = r * light_intensity;
01717                 float gf = g * light_intensity;
01718                 float bf = b * light_intensity;
01719                 uint8_t r8 = (uint8_t)fmaxf(0, fminf(255, rf));
01720                 uint8_t g8 = (uint8_t)fmaxf(0, fminf(255, gf));
01721                 uint8_t b8 = (uint8_t)fmaxf(0, fminf(255, bf));
01722 
01723                 double inv_w = alpha * (1.0 / p0.w) + beta  * (1.0 / p1.w) + gamma * (1.0 / p2.w);
01724                 double z_over_w = alpha * (p0.z / p0.w) + beta  * (p1.z / p1.w) + gamma * (p2.z /
      p2.w);
01725                 double inv_z = inv_w / z_over_w;
01726 
01727                 if (inv_z >= MAT2D_AT(inv_z_buffer, y, x)) {
01728                     adl_point_draw(screen_mat, x, y, RGBA_hexARGB(r8, g8, b8, a), offset_zoom_param);
01729                     MAT2D_AT(inv_z_buffer, y, x) = inv_z;
01730                 }
01731             }
01732         }
01733     }
01734 }
01735 
01746 void adl_tri_mesh_draw(Mat2D_uint32 screen_mat, Tri_mesh mesh, uint32_t color, Offset_zoom_param
      offset_zoom_param)
01747 {
01748     for (size_t i = 0; i < mesh.length; i++) {
01749         Tri tri = mesh.elements[i];
01750         if (tri.to_draw) {
01751             // color = rand_double() * 0xFFFFFFFF;
01752             adl_tri_draw(screen_mat, tri, color, offset_zoom_param);
01753         }
01754     }
01755 }
01756 
01768 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)
01769 {
01770     for (size_t i = 0; i < mesh.length; i++) {
01771         Tri tri = mesh.elements[i];
01772         /* Reject invalid triangles */
01773         adl_assert_tri_is_valid(tri);
01774 
01775         if (!tri.to_draw) continue;
01776 
01777         adl_tri_fill_Pinedas_rasterizer(screen_mat, inv_z_buffer_mat, tri, color, offset_zoom_param);
01778     }
01779 }
01780 
01792 void adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_color(Mat2D_uint32 screen_mat, Mat2D
      inv_z_buffer_mat, Tri_mesh mesh, Offset_zoom_param offset_zoom_param)
01793 {
01794     for (size_t i = 0; i < mesh.length; i++) {
01795         Tri tri = mesh.elements[i];
01796         /* Reject invalid triangles */
01797         adl_assert_tri_is_valid(tri);
01798 
01799         if (!tri.to_draw) continue;
01800 
01801         adl_tri_fill_Pinedas_rasterizer_interpolate_color(screen_mat, inv_z_buffer_mat, tri,
      offset_zoom_param);
01802     }
01803 }
01804 
01817 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)
01818 {
01819     for (size_t i = 0; i < mesh.length; i++) {
01820         Tri tri = mesh.elements[i];
01821         /* Reject invalid triangles */
01822         adl_assert_tri_is_valid(tri);
01823 
01824         if (!tri.to_draw) continue;
01825 
01826         adl_tri_fill_Pinedas_rasterizer_interpolate_normal(screen_mat, inv_z_buffer_mat, tri, color,
      offset_zoom_param);
01827     }
01828 }
01829 
01845 float adl_tan_half_angle(Point vi, Point vj, Point p, float li, float lj)
01846 {
01847     float ax = vi.x - p.x, ay = vi.y - p.y;
01848     float bx = vj.x - p.x, by = vj.y - p.y;
01849     float dot = ax * bx + ay * by;
01850     float cross = ax * by - ay * bx;              // signed 2D cross (scalar)
01851     float denom = dot + li * lj;                   // = |a||b|(1 + cos(alpha))
01852     return fabsf(cross) / fmaxf(1e-20f, denom);    // tan(alpha/2)
01853 }
01854 
01865 float adl_linear_map(float s, float min_in, float max_in, float min_out, float max_out)
01866 {
01867     return (min_out + ((s-min_in)*(max_out-min_out))/(max_in-min_in));
01868 }
01869 
01885 void adl_quad2tris(Quad quad, Tri *tri1, Tri *tri2, char split_line[])
01886 {
01887     if (!strncmp(split_line, "02", 2)) {
01888         tri1->points[0] = quad.points[0];
01889         tri1->points[1] = quad.points[1];
01890         tri1->points[2] = quad.points[2];
01891         tri1->to_draw = quad.to_draw;
01892         tri1->light_intensity[0] = quad.light_intensity[0];
01893         tri1->light_intensity[1] = quad.light_intensity[1];
01894         tri1->light_intensity[2] = quad.light_intensity[2];
01895         tri1->colors[0] = quad.colors[0];
01896         tri1->colors[1] = quad.colors[1];
01897         tri1->colors[2] = quad.colors[2];
01898 
01899         tri2->points[0] = quad.points[2];
01900         tri2->points[1] = quad.points[3];
01901         tri2->points[2] = quad.points[0];
01902         tri2->to_draw = quad.to_draw;
01903         tri1->light_intensity[0] = quad.light_intensity[2];
01904         tri1->light_intensity[1] = quad.light_intensity[3];
01905         tri1->light_intensity[2] = quad.light_intensity[0];
01906         tri2->colors[0] = quad.colors[2];
01907         tri2->colors[1] = quad.colors[3];
01908         tri2->colors[2] = quad.colors[0];
01909     } else if (!strncmp(split_line, "13", 2)) {
01910         tri1->points[0] = quad.points[1];
01911         tri1->points[1] = quad.points[2];
01912         tri1->points[2] = quad.points[3];
01913         tri1->to_draw = quad.to_draw;
01914         tri1->light_intensity[0] = quad.light_intensity[1];
01915         tri1->light_intensity[1] = quad.light_intensity[2];
01916         tri1->light_intensity[2] = quad.light_intensity[3];
01917         tri1->colors[0] = quad.colors[1];
01918         tri1->colors[1] = quad.colors[2];
01919         tri1->colors[2] = quad.colors[3];
01920 
01921         tri2->points[0] = quad.points[3];
01922         tri2->points[1] = quad.points[0];
01923         tri2->points[2] = quad.points[1];
01924         tri2->to_draw = quad.to_draw;
01925         tri1->light_intensity[0] = quad.light_intensity[3];
01926         tri1->light_intensity[1] = quad.light_intensity[0];
01927         tri1->light_intensity[2] = quad.light_intensity[1];
01928         tri2->colors[0] = quad.colors[3];
01929         tri2->colors[1] = quad.colors[0];
01930         tri2->colors[2] = quad.colors[1];
01931     }
01932 }
01933 
01945 void adl_linear_sRGB_to_okLab(uint32_t hex_ARGB, float *L, float *a, float *b)
01946 {
01947     /* https://bottosson.github.io/posts/oklab/
01948        https://en.wikipedia.org/wiki/Oklab_color_space */
01949     int R_255, G_255, B_255;
01950     HexARGB_RGB_VAR(hex_ARGB, R_255, G_255, B_255);
01951 
01952     float R = R_255;
01953     float G = G_255;
```

```
01954     float B = B_255;
01955
01956     float l = 0.4122214705f * R + 0.5363325363f * G + 0.0514459929f * B;
01957     float m = 0.2119034982f * R + 0.6806995451f * G + 0.1073969566f * B;
01958     float s = 0.0883024619f * R + 0.2817188376f * G + 0.6299787005f * B;
01959
01960     float l_ = cbrtf(l);
01961     float m_ = cbrtf(m);
01962     float s_ = cbrtf(s);
01963
01964     *L = 0.2104542553f * l_ + 0.7936177850f * m_ - 0.0040720468f * s_;
01965     *a = 1.9779984951f * l_ - 2.4285922050f * m_ + 0.4505937099f * s_;
01966     *b = 0.0259040371f * l_ + 0.7827717662f * m_ - 0.8086757660f * s_;
01967
01968 }
01969
01980 void adl_okLab_to_linear_sRGB(float L, float a, float b, uint32_t *hex_ARGB)
01981 {
01982     /* https://bottosson.github.io/posts/oklab/
01983        https://en.wikipedia.org/wiki/Oklab_color_space */
01984
01985     float l_ = L + 0.3963377774f * a + 0.2158037573f * b;
01986     float m_ = L - 0.1055613458f * a - 0.0638541728f * b;
01987     float s_ = L - 0.0894841775f * a - 1.2914855480f * b;
01988
01989     float l = l_ * l_ * l_;
01990     float m = m_ * m_ * m_;
01991     float s = s_ * s_ * s_;
01992
01993     float R = + 4.0767416621f * l - 3.3077115913f * m + 0.2309699292f * s;
01994     float G = - 1.2684380046f * l + 2.6097574011f * m - 0.3413193965f * s;
01995     float B = - 0.0041960863f * l - 0.7034186147f * m + 1.7076147010f * s;
01996
01997     R = fmaxf(fminf(R, 255), 0);
01998     G = fmaxf(fminf(G, 255), 0);
01999     B = fmaxf(fminf(B, 255), 0);
02000
02001     *hex_ARGB = RGBA_hexARGB(R, G, B, 0xFF);
02002 }
02003
02012 void adl_linear_sRGB_to_okLch(uint32_t hex_ARGB, float *L, float *c, float *h_deg)
02013 {
02014     float a, b;
02015     adl_linear_sRGB_to_okLab(hex_ARGB, L, &a, &b);
02016
02017     *c = sqrtf(a * a + b * b);
02018     *h_deg = atan2f(b, a) * 180 / PI;
02019 }
02020
02031 void adl_okLch_to_linear_sRGB(float L, float c, float h_deg, uint32_t *hex_ARGB)
02032 {
02033     h_deg = fmodf((h_deg + 360), 360);
02034     float a = c * cosf(h_deg * PI / 180);
02035     float b = c * sinf(h_deg * PI / 180);
02036     adl_okLab_to_linear_sRGB(L, a, b, hex_ARGB);
02037 }
02038
02053 void adl_interpolate_ARGBcolor_on_okLch(uint32_t color1, uint32_t color2, float t, float
       num_of_rotations, uint32_t *color_out)
02054 {
02055     float L_1, c_1, h_1;
02056     float L_2, c_2, h_2;
02057     adl_linear_sRGB_to_okLch(color1, &L_1, &c_1, &h_1);
02058     adl_linear_sRGB_to_okLch(color2, &L_2, &c_2, &h_2);
02059     h_2 = h_2 + 360 * num_of_rotations;
02060
02061     float L, c, h;
02062     L = L_1 * (1 - t) + L_2 * (t);
02063     c = c_1 * (1 - t) + c_2 * (t);
02064     h = h_1 * (1 - t) + h_2 * (t);
02065     adl_okLch_to_linear_sRGB(L, c, h, color_out);
02066 }
02067
02081 Figure adl_figure_alloc(size_t rows, size_t cols, Point top_left_position)
02082 {
02083     ADL_ASSERT(rows && cols);
02084     adl_assert_point_is_valid(top_left_position);
02085
02086     Figure figure = {0};
02087     figure.pixels_mat = mat2D_alloc_uint32(rows, cols);
02088     figure.inv_z_buffer_mat = mat2D_alloc(rows, cols);
02089     memset(figure.inv_z_buffer_mat.elements, 0x0, sizeof(double) * figure.inv_z_buffer_mat.rows *
       figure.inv_z_buffer_mat.cols);
02090     ada_init_array(Curve, figure.src_curve_array);
02091
02092     figure.top_left_position = top_left_position;
02093
```

```
02094     int max_i    = (int)(figure.pixels_mat.rows);
02095     int max_j    = (int)(figure.pixels_mat.cols);
02096     int offset_i = (int)fminf(figure.pixels_mat.rows * ADL_FIGURE_PADDING_PRECENTAGE / 100.0f,
      ADL_MAX_FIGURE_PADDING);
02097     int offset_j = (int)fminf(figure.pixels_mat.cols * ADL_FIGURE_PADDING_PRECENTAGE / 100.0f,
      ADL_MAX_FIGURE_PADDING);
02098
02099     figure.min_x_pixel = offset_j;
02100     figure.max_x_pixel = max_j - offset_j;
02101     figure.min_y_pixel = offset_i;
02102     figure.max_y_pixel = max_i - offset_i;
02103
02104     figure.min_x = + FLT_MAX;
02105     figure.max_x = - FLT_MAX;
02106     figure.min_y = + FLT_MAX;
02107     figure.max_y = - FLT_MAX;
02108
02109     figure.offset_zoom_param = ADL_DEFAULT_OFFSET_ZOOM;
02110
02111     return figure;
02112 }
02113
02124 void adl_figure_copy_to_screen(Mat2D_uint32 screen_mat, Figure figure)
02125 {
02126     for (size_t i = 0; i < figure.pixels_mat.rows; i++) {
02127         for (size_t j = 0; j < figure.pixels_mat.cols; j++) {
02128             int offset_i = figure.top_left_position.y;
02129             int offset_j = figure.top_left_position.x;
02130
02131             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});
02132         }
02133     }
02134 }
02135
02144 void adl_axis_draw_on_figure(Figure *figure)
02145 {
02146     int max_i    = (int)(figure->pixels_mat.rows);
02147     int max_j    = (int)(figure->pixels_mat.cols);
02148     int offset_i = (int)fmaxf(fminf(figure->pixels_mat.rows * ADL_FIGURE_PADDING_PRECENTAGE / 100.0f,
      ADL_MAX_FIGURE_PADDING), ADL_MIN_FIGURE_PADDING);
02149     int offset_j = (int)fmaxf(fminf(figure->pixels_mat.cols * ADL_FIGURE_PADDING_PRECENTAGE / 100.0f,
      ADL_MAX_FIGURE_PADDING), ADL_MIN_FIGURE_PADDING);
02150
02151     int arrow_head_size_x = (int)fminf(ADL_MAX_HEAD_SIZE, ADL_FIGURE_PADDING_PRECENTAGE / 100.0f *
      (max_j - 2 * offset_j));
02152     int arrow_head_size_y = (int)fminf(ADL_MAX_HEAD_SIZE, ADL_FIGURE_PADDING_PRECENTAGE / 100.0f *
      (max_i - 2 * offset_i));
02153
02154     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);
02155     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);
02156     // 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);
02157
02158     figure->x_axis_head_size = arrow_head_size_x;
02159     figure->y_axis_head_size = arrow_head_size_y;
02160 }
02161
02170 void adl_max_min_values_draw_on_figure(Figure figure)
02171 {
02172     char x_min_sentence[256];
02173     char x_max_sentence[256];
02174     snprintf(x_min_sentence, 256, "%g", figure.min_x);
02175     snprintf(x_max_sentence, 256, "%g", figure.max_x);
02176
02177     int x_sentence_hight_pixel = (figure.pixels_mat.rows - figure.max_y_pixel -
      ADL_MIN_CHARACTER_OFFSET * 3);
02178     int x_min_char_width_pixel = x_sentence_hight_pixel / 2;
02179     int x_max_char_width_pixel = x_sentence_hight_pixel / 2;
02180
02181     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));
02182     x_min_char_width_pixel = x_min_sentence_width_pixel / strlen(x_min_sentence) -
      ADL_MIN_CHARACTER_OFFSET;
02183
02184     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;
02185     x_max_char_width_pixel = (x_max_sentence_width_pixel + figure.x_axis_head_size) /
      strlen(x_max_sentence) - ADL_MIN_CHARACTER_OFFSET;
02186
02187     int x_min_sentence_hight_pixel = (int)fminf(x_min_char_width_pixel * 2, x_sentence_hight_pixel);
02188     int x_max_sentence_hight_pixel = (int)fminf(x_max_char_width_pixel * 2, x_sentence_hight_pixel);
```

```
02189
02190     x_min_sentence_hight_pixel = (int)fminf(x_min_sentence_hight_pixel, x_max_sentence_hight_pixel);
02191     x_max_sentence_hight_pixel = x_min_sentence_hight_pixel;
02192
02193     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;
02194
02195     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);
02196     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);
02197
02198     char y_min_sentence[256];
02199     char y_max_sentence[256];
02200     snprintf(y_min_sentence, 256, "%g", figure.min_y);
02201     snprintf(y_max_sentence, 256, "%g", figure.max_y);
02202
02203     int y_sentence_width_pixel = figure.min_x_pixel - ADL_MAX_CHARACTER_OFFSET -
      figure.y_axis_head_size;
02204     int y_max_char_width_pixel = y_sentence_width_pixel;
02205     y_max_char_width_pixel /= strlen(y_max_sentence);
02206     int y_max_sentence_hight_pixel = y_max_char_width_pixel * 2;
02207
02208     int y_min_char_width_pixel = y_sentence_width_pixel;
02209     y_min_char_width_pixel /= strlen(y_min_sentence);
02210     int y_min_sentence_hight_pixel = y_min_char_width_pixel * 2;
02211
02212     y_min_sentence_hight_pixel = (int)fmaxf(fminf(y_min_sentence_hight_pixel,
      y_max_sentence_hight_pixel), 1);
02213     y_max_sentence_hight_pixel = y_min_sentence_hight_pixel;
02214
02215     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);
02216     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);
02217 }
02218
02230 void adl_curve_add_to_figure(Figure *figure, Point *src_points, size_t src_len, uint32_t color)
02231 {
02232     Curve src_points_ada;
02233     ada_init_array(Point, src_points_ada);
02234     src_points_ada.color = color;
02235
02236     for (size_t i = 0; i < src_len; i++) {
02237         Point current_point = src_points[i];
02238         if (current_point.x > figure->max_x) {
02239             figure->max_x = current_point.x;
02240         }
02241         if (current_point.y > figure->max_y) {
02242             figure->max_y = current_point.y;
02243         }
02244         if (current_point.x < figure->min_x) {
02245             figure->min_x = current_point.x;
02246         }
02247         if (current_point.y < figure->min_y) {
02248             figure->min_y = current_point.y;
02249         }
02250         ada_append(Point, src_points_ada, current_point);
02251     }
02252
02253     ada_append(Curve, figure->src_curve_array, src_points_ada);
02254 }
02255
02265 void adl_curves_plot_on_figure(Figure figure)
02266 {
02267     mat2D_fill_uint32(figure.pixels_mat, figure.background_color);
02268     memset(figure.inv_z_buffer_mat.elements, 0x0, sizeof(double) * figure.inv_z_buffer_mat.rows *
      figure.inv_z_buffer_mat.cols);
02269     if (figure.to_draw_axis) adl_axis_draw_on_figure(&figure);
02270
02271     for (size_t curve_index = 0; curve_index < figure.src_curve_array.length; curve_index++) {
02272         size_t src_len = figure.src_curve_array.elements[curve_index].length;
02273         Point *src_points = figure.src_curve_array.elements[curve_index].elements;
02274         for (size_t i = 0; i < src_len-1; i++) {
02275             Point src_start = src_points[i];
02276             Point src_end   = src_points[i+1];
02277             Point des_start = {0};
02278             Point des_end = {0};
02279
02280             des_start.x = adl_linear_map(src_start.x, figure.min_x, figure.max_x, figure.min_x_pixel,
      figure.max_x_pixel);
02281             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));
```

```
02282
02283             des_end.x = adl_linear_map(src_end.x, figure.min_x, figure.max_x, figure.min_x_pixel,
      figure.max_x_pixel);
02284             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));
02285
02286             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);
02287         }
02288     }
02289
02290     if (figure.to_draw_max_min_values) adl_max_min_values_draw_on_figure(figure);
02291 }
02292
02293 /* check offset2D. might convert it to a Mat2D */
02294 #define adl_offset2d(i, j, ni) (j) * (ni) + (i)
02314 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)
02315 {
02316     mat2D_fill_uint32(figure.pixels_mat, figure.background_color);
02317     memset(figure.inv_z_buffer_mat.elements, 0x0, sizeof(double) * figure.inv_z_buffer_mat.rows *
      figure.inv_z_buffer_mat.cols);
02318     if (figure.to_draw_axis) adl_axis_draw_on_figure(&figure);
02319
02320     float min_scalar = FLT_MAX;
02321     float max_scalar = FLT_MIN;
02322     for (int i = 0; i < ni; i++) {
02323         for (int j = 0; j < nj; j++) {
02324             float val = scalar_2Dmat[adl_offset2d(i, j, ni)];
02325             if (val > max_scalar) max_scalar = val;
02326             if (val < min_scalar) min_scalar = val;
02327             float current_x = x_2Dmat[adl_offset2d(i, j, ni)];
02328             float current_y = y_2Dmat[adl_offset2d(i, j, ni)];
02329             if (current_x > figure.max_x) {
02330                 figure.max_x = current_x;
02331             }
02332             if (current_y > figure.max_y) {
02333                 figure.max_y = current_y;
02334             }
02335             if (current_x < figure.min_x) {
02336                 figure.min_x = current_x;
02337             }
02338             if (current_y < figure.min_y) {
02339                 figure.min_y = current_y;
02340             }
02341         }
02342     }
02343
02344     float window_w = (float)figure.pixels_mat.cols;
02345     float window_h = (float)figure.pixels_mat.rows;
02346
02347     for (int i = 0; i < ni-1; i++) {
02348         for (int j = 0; j < nj-1; j++) {
02349             Quad quad = {0};
02350             quad.light_intensity[0] = 1;
02351             quad.light_intensity[1] = 1;
02352             quad.light_intensity[2] = 1;
02353             quad.light_intensity[3] = 1;
02354             quad.to_draw = 1;
02355
02356             quad.points[3].x = x_2Dmat[adl_offset2d(i  , j  , ni)];
02357             quad.points[3].y = y_2Dmat[adl_offset2d(i  , j  , ni)];
02358             quad.points[2].x = x_2Dmat[adl_offset2d(i+1, j  , ni)];
02359             quad.points[2].y = y_2Dmat[adl_offset2d(i+1, j  , ni)];
02360             quad.points[1].x = x_2Dmat[adl_offset2d(i+1, j+1, ni)];
02361             quad.points[1].y = y_2Dmat[adl_offset2d(i+1, j+1, ni)];
02362             quad.points[0].x = x_2Dmat[adl_offset2d(i  , j+1, ni)];
02363             quad.points[0].y = y_2Dmat[adl_offset2d(i  , j+1, ni)];
02364
02365             for (int p_index = 0; p_index < 4; p_index++) {
02366                 quad.points[p_index].z = 1;
02367                 quad.points[p_index].w = 1;
02368                 quad.points[p_index].x = adl_linear_map(quad.points[p_index].x, figure.min_x,
      figure.max_x, figure.min_x_pixel, figure.max_x_pixel);
02369                 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));
02370
02371                 adl_offset_zoom_point(quad.points[p_index], window_w, window_h,
      figure.offset_zoom_param);
02372             }
02373
02374             float t3 = adl_linear_map(scalar_2Dmat[adl_offset2d(i  , j  , ni)], min_scalar,
      max_scalar, 0, 1);
02375             float t2 = adl_linear_map(scalar_2Dmat[adl_offset2d(i+1, j  , ni)], min_scalar,
      max_scalar, 0, 1);
02376             float t1 = adl_linear_map(scalar_2Dmat[adl_offset2d(i+1, j+1, ni)], min_scalar,
```

```
    max_scalar, 0, 1);
02377           float t0 = adl_linear_map(scalar_2Dmat[adl_offset2d(  i, j+1, ni)], min_scalar,
    max_scalar, 0, 1);
02378
02379           /* https://en.wikipedia.org/wiki/Oklab_color_space */
02380           if (!strcmp(color_scale, "b-c")) {
02381               uint32_t color = 0, color1 = BLUE_hexARGB, color2 = CYAN_hexARGB;
02382               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02383               quad.colors[0] = color;
02384
02385               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02386               quad.colors[1] = color;
02387
02388               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02389               quad.colors[2] = color;
02390
02391               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02392               quad.colors[3] = color;
02393           } else if (!strcmp(color_scale, "b-g")) {
02394               uint32_t color = 0, color1 = BLUE_hexARGB, color2 = GREEN_hexARGB;
02395               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02396               quad.colors[0] = color;
02397
02398               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02399               quad.colors[1] = color;
02400
02401               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02402               quad.colors[2] = color;
02403
02404               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02405               quad.colors[3] = color;
02406           } else if (!strcmp(color_scale, "b-r")) {
02407               uint32_t color = 0, color1 = BLUE_hexARGB, color2 = RED_hexARGB;
02408               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02409               quad.colors[0] = color;
02410
02411               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02412               quad.colors[1] = color;
02413
02414               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02415               quad.colors[2] = color;
02416
02417               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02418               quad.colors[3] = color;
02419           } else if (!strcmp(color_scale, "b-y")) {
02420               uint32_t color = 0, color1 = BLUE_hexARGB, color2 = YELLOW_hexARGB;
02421               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02422               quad.colors[0] = color;
02423
02424               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02425               quad.colors[1] = color;
02426
02427               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02428               quad.colors[2] = color;
02429
02430               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02431               quad.colors[3] = color;
02432           } else if (!strcmp(color_scale, "g-y")) {
02433               uint32_t color = 0, color1 = GREEN_hexARGB, color2 = YELLOW_hexARGB;
02434               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02435               quad.colors[0] = color;
02436
02437               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02438               quad.colors[1] = color;
02439
02440               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02441               quad.colors[2] = color;
02442
02443               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02444               quad.colors[3] = color;
02445           } else if (!strcmp(color_scale, "g-p")) {
02446               uint32_t color = 0, color1 = GREEN_hexARGB, color2 = PURPLE_hexARGB;
02447               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02448               quad.colors[0] = color;
02449
02450               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02451               quad.colors[1] = color;
02452
02453               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02454               quad.colors[2] = color;
02455
02456               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02457               quad.colors[3] = color;
02458           } else if (!strcmp(color_scale, "g-r")) {
02459               uint32_t color = 0, color1 = GREEN_hexARGB, color2 = RED_hexARGB;
02460               adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02461               quad.colors[0] = color;
```

```
02462
02463                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02464                 quad.colors[1] = color;
02465
02466                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02467                 quad.colors[2] = color;
02468
02469                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02470                 quad.colors[3] = color;
02471             } else if (!strcmp(color_scale, "r-y")) {
02472                 uint32_t color = 0, color1 = RED_hexARGB, color2 = YELLOW_hexARGB;
02473                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t0, num_of_rotations, &color);
02474                 quad.colors[0] = color;
02475
02476                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t1, num_of_rotations, &color);
02477                 quad.colors[1] = color;
02478
02479                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t2, num_of_rotations, &color);
02480                 quad.colors[2] = color;
02481
02482                 adl_interpolate_ARGBcolor_on_okLch(color1, color2, t3, num_of_rotations, &color);
02483                 quad.colors[3] = color;
02484             }
02485
02486             adl_quad_fill_interpolate_color_mean_value(figure.pixels_mat, figure.inv_z_buffer_mat,
      quad, ADL_DEFAULT_OFFSET_ZOOM);
02487         }
02488     }
02489
02490     if (figure.to_draw_max_min_values) {
02491         adl_max_min_values_draw_on_figure(figure);
02492     }
02493
02494 }
02495
02513 Grid adl_cartesian_grid_create(float min_e1, float max_e1, float min_e2, float max_e2, int
      num_samples_e1, int num_samples_e2, char plane[], float third_direction_position)
02514 {
02515     Grid grid;
02516     ada_init_array(Curve, grid.curves);
02517
02518     grid.min_e1 = min_e1;
02519     grid.max_e1 = max_e1;
02520     grid.min_e2 = min_e2;
02521     grid.max_e2 = max_e2;
02522     grid.num_samples_e1 = num_samples_e1;
02523     grid.num_samples_e2 = num_samples_e2;
02524     strncpy(grid.plane, plane, 2);
02525
02526     float del_e1 = (max_e1 - min_e1) / num_samples_e1;
02527     float del_e2 = (max_e2 - min_e2) / num_samples_e2;
02528
02529     grid.de1 = del_e1;
02530     grid.de2 = del_e2;
02531
02532     if (!strncmp(plane, "XY", 3) || !strncmp(plane, "xy", 3)) {
02533         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02534             Curve curve = {0};
02535             ada_init_array(Point, curve);
02536             Point point_max = {0}, point_min = {0};
02537
02538             point_min.x = min_e1 + e1_index * del_e1;
02539             point_min.y = min_e2;
02540             point_min.z = third_direction_position;
02541             point_min.w = 1;
02542
02543             point_max.x = min_e1 + e1_index * del_e1;
02544             point_max.y = max_e2;
02545             point_max.z = third_direction_position;
02546             point_max.w = 1;
02547
02548             ada_append(Point, curve, point_min);
02549             ada_append(Point, curve, point_max);
02550
02551             ada_append(Curve, grid.curves, curve);
02552         }
02553         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02554             Curve curve = {0};
02555             ada_init_array(Point, curve);
02556             Point point_max = {0}, point_min = {0};
02557
02558             point_min.x = min_e1;
02559             point_min.y = min_e2 + e2_index * del_e2;
02560             point_min.z = third_direction_position;
02561             point_min.w = 1;
02562
02563             point_max.x = max_e1;
```

```
02564             point_max.y = min_e2 + e2_index * del_e2;
02565             point_max.z = third_direction_position;
02566             point_max.w = 1;
02567 
02568             ada_append(Point, curve, point_min);
02569             ada_append(Point, curve, point_max);
02570 
02571             ada_append(Curve, grid.curves, curve);
02572         }
02573     } else if (!strncmp(plane, "XZ", 3) || !strncmp(plane, "xz", 3)) {
02574         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02575             Curve curve = {0};
02576             ada_init_array(Point, curve);
02577             Point point_max = {0}, point_min = {0};
02578 
02579             point_min.x = min_e1 + e1_index * del_e1;
02580             point_min.y = third_direction_position;
02581             point_min.z = min_e2;
02582             point_min.w = 1;
02583 
02584             point_max.x = min_e1 + e1_index * del_e1;
02585             point_max.y = third_direction_position;
02586             point_max.z = max_e2;
02587             point_max.w = 1;
02588 
02589             ada_append(Point, curve, point_min);
02590             ada_append(Point, curve, point_max);
02591 
02592             ada_append(Curve, grid.curves, curve);
02593         }
02594         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02595             Curve curve = {0};
02596             ada_init_array(Point, curve);
02597             Point point_max = {0}, point_min = {0};
02598 
02599             point_min.x = min_e1;
02600             point_min.y = third_direction_position;
02601             point_min.z = min_e2 + e2_index * del_e2;
02602             point_min.w = 1;
02603 
02604             point_max.x = max_e1;
02605             point_max.y = third_direction_position;
02606             point_max.z = min_e2 + e2_index * del_e2;
02607             point_max.w = 1;
02608 
02609             ada_append(Point, curve, point_min);
02610             ada_append(Point, curve, point_max);
02611 
02612             ada_append(Curve, grid.curves, curve);
02613         }
02614     } else if (!strncmp(plane, "YX", 3) || !strncmp(plane, "yx", 3)) {
02615         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02616             Curve curve = {0};
02617             ada_init_array(Point, curve);
02618             Point point_max = {0}, point_min = {0};
02619 
02620             point_min.x = min_e2;
02621             point_min.y = min_e1 + e1_index * del_e1;
02622             point_min.z = third_direction_position;
02623             point_min.w = 1;
02624 
02625             point_max.x = max_e2;
02626             point_max.y = min_e1 + e1_index * del_e1;
02627             point_max.z = third_direction_position;
02628             point_max.w = 1;
02629 
02630             ada_append(Point, curve, point_min);
02631             ada_append(Point, curve, point_max);
02632 
02633             ada_append(Curve, grid.curves, curve);
02634         }
02635         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02636             Curve curve = {0};
02637             ada_init_array(Point, curve);
02638             Point point_max = {0}, point_min = {0};
02639 
02640             point_min.x = min_e2 + e2_index * del_e2;
02641             point_min.y = min_e1;
02642             point_min.z = third_direction_position;
02643             point_min.w = 1;
02644 
02645             point_max.x = min_e2 + e2_index * del_e2;
02646             point_max.y = max_e1;
02647             point_max.z = third_direction_position;
02648             point_max.w = 1;
02649 
02650             ada_append(Point, curve, point_min);
```

```
02651             ada_append(Point, curve, point_max);
02652
02653             ada_append(Curve, grid.curves, curve);
02654         }
02655     } else if (!strncmp(plane, "YZ", 3) || !strncmp(plane, "yz", 3)) {
02656         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02657             Curve curve = {0};
02658             ada_init_array(Point, curve);
02659             Point point_max = {0}, point_min = {0};
02660
02661             point_min.x = third_direction_position;
02662             point_min.y = min_e1 + e1_index * del_e1;
02663             point_min.z = min_e2;
02664             point_min.w = 1;
02665
02666             point_max.x = third_direction_position;
02667             point_max.y = min_e1 + e1_index * del_e1;
02668             point_max.z = max_e2;
02669             point_max.w = 1;
02670
02671             ada_append(Point, curve, point_min);
02672             ada_append(Point, curve, point_max);
02673
02674             ada_append(Curve, grid.curves, curve);
02675         }
02676         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02677             Curve curve = {0};
02678             ada_init_array(Point, curve);
02679             Point point_max = {0}, point_min = {0};
02680
02681             point_min.x = third_direction_position;
02682             point_min.y = min_e1;
02683             point_min.z = min_e2 + e2_index * del_e2;
02684             point_min.w = 1;
02685
02686             point_max.x = third_direction_position;
02687             point_max.y = max_e1;
02688             point_max.z = min_e2 + e2_index * del_e2;
02689             point_max.w = 1;
02690
02691             ada_append(Point, curve, point_min);
02692             ada_append(Point, curve, point_max);
02693
02694             ada_append(Curve, grid.curves, curve);
02695         }
02696     } else if (!strncmp(plane, "ZX", 3) || !strncmp(plane, "zx", 3)) {
02697         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02698             Curve curve = {0};
02699             ada_init_array(Point, curve);
02700             Point point_max = {0}, point_min = {0};
02701
02702             point_min.x = min_e2;
02703             point_min.y = third_direction_position;
02704             point_min.z = min_e1 + e1_index * del_e1;
02705             point_min.w = 1;
02706
02707             point_max.x = max_e2;
02708             point_max.y = third_direction_position;
02709             point_max.z = min_e1 + e1_index * del_e1;
02710             point_max.w = 1;
02711
02712             ada_append(Point, curve, point_min);
02713             ada_append(Point, curve, point_max);
02714
02715             ada_append(Curve, grid.curves, curve);
02716         }
02717         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02718             Curve curve = {0};
02719             ada_init_array(Point, curve);
02720             Point point_max = {0}, point_min = {0};
02721
02722             point_min.x = min_e2 + e2_index * del_e2;
02723             point_min.y = third_direction_position;
02724             point_min.z = min_e1;
02725             point_min.w = 1;
02726
02727             point_max.x = min_e2 + e2_index * del_e2;
02728             point_max.y = third_direction_position;
02729             point_max.z = max_e1;
02730             point_max.w = 1;
02731
02732             ada_append(Point, curve, point_min);
02733             ada_append(Point, curve, point_max);
02734
02735             ada_append(Curve, grid.curves, curve);
02736         }
02737     } else if (!strncmp(plane, "ZY", 3) || !strncmp(plane, "zy", 3)) {
```

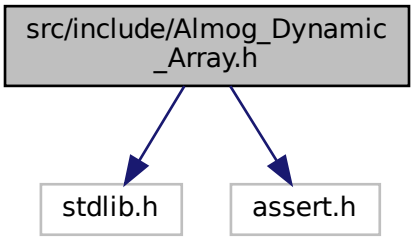
```
02738         for (int e1_index = 0; e1_index <= num_samples_e1; e1_index++) {
02739             Curve curve = {0};
02740             ada_init_array(Point, curve);
02741             Point point_max = {0}, point_min = {0};
02742 
02743             point_min.x = third_direction_position;
02744             point_min.y = min_e2;
02745             point_min.z = min_e1 + e1_index * del_e1;
02746             point_min.w = 1;
02747 
02748             point_max.x = third_direction_position;
02749             point_max.y = max_e2;
02750             point_max.z = min_e1 + e1_index * del_e1;
02751             point_max.w = 1;
02752 
02753             ada_append(Point, curve, point_min);
02754             ada_append(Point, curve, point_max);
02755 
02756             ada_append(Curve, grid.curves, curve);
02757         }
02758         for (int e2_index = 0; e2_index <= num_samples_e2; e2_index++) {
02759             Curve curve = {0};
02760             ada_init_array(Point, curve);
02761             Point point_max = {0}, point_min = {0};
02762 
02763             point_min.x = third_direction_position;
02764             point_min.y = min_e2 + e2_index * del_e2;
02765             point_min.z = min_e1;
02766             point_min.w = 1;
02767 
02768             point_max.x = third_direction_position;
02769             point_max.y = min_e2 + e2_index * del_e2;
02770             point_max.z = max_e1;
02771             point_max.w = 1;
02772 
02773             ada_append(Point, curve, point_min);
02774             ada_append(Point, curve, point_max);
02775 
02776             ada_append(Curve, grid.curves, curve);
02777         }
02778     }
02779 
02780     return grid;
02781 }
02782 
02791 void adl_grid_draw(Mat2D_uint32 screen_mat, Grid grid, uint32_t color, Offset_zoom_param
      offset_zoom_param)
02792 {
02793     for (size_t curve_index = 0; curve_index < grid.curves.length; curve_index++) {
02794         adl_lines_draw(screen_mat, grid.curves.elements[curve_index].elements,
      grid.curves.elements[curve_index].length, color, offset_zoom_param);
02795     }
02796 }
02797 
02798 #endif /*ALMOG_DRAW_LIBRARY_IMPLEMENTATION*/
```

4.5 src/include/Almog_Dynamic_Array.h File Reference

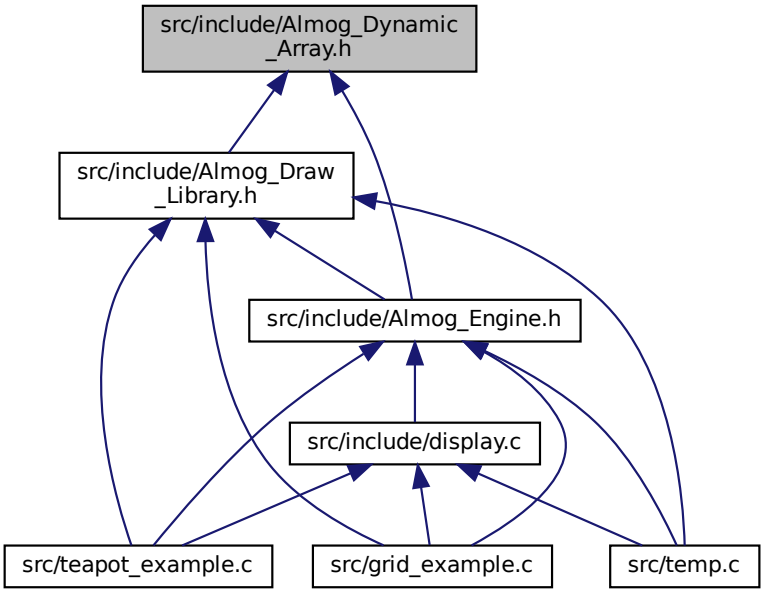
Header-only C macros that implement a simple dynamic array.

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

Include dependency graph for Almog_Dynamic_Array.h:



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



Macros

- #define ADA_INIT_CAPACITY 10

  Default initial capacity used by ada_init_array.
- #define ADA_MALLOC malloc

  Allocation function used by this header (defaults to malloc).
- #define ADA_REALLOC realloc

  Reallocation function used by this header (defaults to realloc).
- #define ADA_ASSERT assert

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

- #define ada_init_array(type, header)

  Initialize an array header and allocate its initial storage.
- #define ada_resize(type, header, new_capacity)

  Resize the underlying storage to hold new_capacity elements.
- #define ada_append(type, header, value)

  Append a value to the end of the array, growing if necessary.
- #define ada_insert(type, header, value, index)

  Insert value at position index, preserving order (O(n)).
- #define ada_insert_unordered(type, header, value, index)

  Insert value at index without preserving order (O(1) amortized).
- #define ada_remove(type, header, index)

  Remove element at index, preserving order (O(n)).
- #define ada_remove_unordered(type, header, index)

  Remove element at index by moving the last element into its place (O(1)); order is not preserved.

4.5.1 Detailed Description

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

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

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

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

Customization:

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

Complexity (n = number of elements):

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

Notes and limitations:

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

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

ada_int_array arr; ada_init_array(int, arr); ada_append(int, arr, 42); ada_insert(int, arr, 7, 0); // requires arr.length > 0 ada_remove(int, arr, 1); free(arr.elements);

Definition in file Almog_Dynamic_Array.h.

4.5.2 Macro Definition Documentation

4.5.2.1 ada_appand

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

Value:

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

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

Parameters

type	Element type stored in the array.
header	Lvalue of the header struct.
value	Value to append.

Postcondition

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

Note

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

Definition at line 169 of file Almog_Dynamic_Array.h.

4.5.2.2 ADA_ASSERT

```
#define ADA_ASSERT assert
```

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

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

Definition at line 96 of file Almog_Dynamic_Array.h.

4.5.2.3 ada_init_array

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

Value:

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

Initialize an array header and allocate its initial storage.

Parameters

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

Precondition

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

Postcondition

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

Note

Allocation uses ADA_MALLOC and is checked via ADA_ASSERT.

Definition at line 120 of file Almog_Dynamic_Array.h.

4.5.2.4 ADA_INIT_CAPACITY

```
#define ADA_INIT_CAPACITY 10
```

Default initial capacity used by ada_init_array.

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

Definition at line 64 of file Almog_Dynamic_Array.h.

4.5.2.5 ada_insert

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

Value:

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

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

Parameters

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

Precondition

0 <= index <= header.length.

header.length > 0 if index == header.length (this macro reads the last element internally). For inserting into an empty array, use ada_append or ada_insert_unordered.

Postcondition

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

Note

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

Definition at line 196 of file Almog_Dynamic_Array.h.

4.5.2.6 ada_insert_unordered

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

Value:

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

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

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

Parameters

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

Precondition

0 <= index <= header.length.

Postcondition

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

Definition at line 222 of file Almog_Dynamic_Array.h.

4.5.2.7 ADA_MALLOC

```
#define ADA_MALLOC malloc
```

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

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

Definition at line 74 of file Almog_Dynamic_Array.h.

4.5.2.8 ADA_REALLOC

```
#define ADA_REALLOC realloc
```

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

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

Definition at line 85 of file Almog_Dynamic_Array.h.

4.5.2.9 ada_remove

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

Value:

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

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

Parameters

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

Precondition

0 <= index < header.length.

Postcondition

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

Definition at line 246 of file Almog_Dynamic_Array.h.

4.5.2.10 ada_remove_unordered

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

Value:

```
    do {          \
    ADA_ASSERT((int)(index) >= 0);                                  \
    ADA_ASSERT((float)(index) - (int)(index) == 0);                 \
    header.elements[index] = header.elements[header.length-1];  \
    header.length--;                                                \
} while (0)
```

Remove element at index by moving the last element into its place (O(1)); order is not preserved.

Parameters

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

Precondition

0 <= index < header.length and header.length > 0.

Postcondition

header.length is decremented by 1; array order is not preserved.

Definition at line 267 of file Almog_Dynamic_Array.h.

4.5.2.11 ada_resize

```
#define ada_resize(
               type,
               header,
               new_capacity )
```

Value:

```
        do {                                                                \
        type *ada_temp_pointer = (type *)ADA_REALLOC((void *)(header.elements), new_capacity*sizeof(type));
       \
        if (ada_temp_pointer == NULL) {
       \
            exit(1);
       \
        }
       \
        header.elements = ada_temp_pointer;
       \
        ADA_ASSERT(header.elements != NULL);
       \
        header.capacity = new_capacity;
       \
    } while (0)
```

Resize the underlying storage to hold new_capacity elements.

Parameters

type	Element type stored in the array.
header	Lvalue of the header struct.
new_capacity	New capacity in number of elements.

Precondition

new_capacity >= header.length (otherwise elements beyond new_capacity are lost and length will not be adjusted).

Postcondition

header.capacity == new_capacity and header.elements points to a block large enough for new_capacity elements.

Warning

On allocation failure, this macro calls exit(1).

Note

Reallocation uses ADA_REALLOC and is also checked via ADA_ASSERT.

Definition at line 143 of file Almog_Dynamic_Array.h.

4.6 Almog_Dynamic_Array.h

```
00001
00051 #ifndef ALMOG_DYNAMIC_ARRAY_H_
00052 #define ALMOG_DYNAMIC_ARRAY_H_
00053
00054 #include <stdlib.h>
00055 #include <assert.h>
00056
00057
00064 #define ADA_INIT_CAPACITY 10
00065
00073 #ifndef ADA_MALLOC
00074 #define ADA_MALLOC malloc
00075 #endif /*ADA_MALLOC*/
00076
00084 #ifndef ADA_REALLOC
00085 #define ADA_REALLOC realloc
00086 #endif /*ADA_REALLOC*/
00087
00095 #ifndef ADA_ASSERT
00096 #define ADA_ASSERT assert
00097 #endif /*ADA_ASSERT*/
00098
00099 /* typedef struct {
00100     size_t length;
00101     size_t capacity;
00102     int* elements;
00103 } ada_int_array; */
00104
00120 #define ada_init_array(type, header) do {                                    \
00121         header.capacity = ADA_INIT_CAPACITY;                                     \
00122         header.length = 0;                                                   \
00123         header.elements = (type *)ADA_MALLOC(sizeof(type) * header.capacity);    \
00124         ADA_ASSERT(header.elements != NULL);                                 \
00125     } while (0)
00126
```

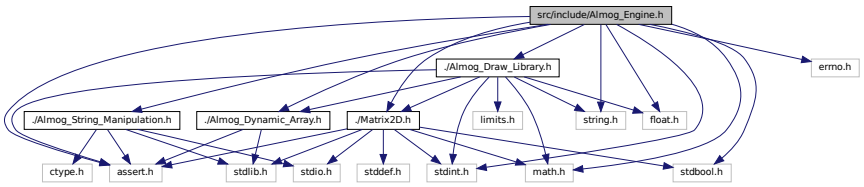
```
00143 #define ada_resize(type, header, new_capacity) do {
        \
00144         type *ada_temp_pointer = (type *)ADA_REALLOC((void *)(header.elements),
      new_capacity*sizeof(type)); \
00145         if (ada_temp_pointer == NULL) {
        \
00146             exit(1);
        \
00147         }
        \
00148         header.elements = ada_temp_pointer;
        \
00149         ADA_ASSERT(header.elements != NULL);
        \
00150         header.capacity = new_capacity;
        \
00151     } while (0)
00152
00169 #define ada_append(type, header, value) do {                          \
00170         if (header.length >= header.capacity) {                       \
00171             ada_resize(type, header, (int)(header.capacity*1.5));     \
00172         }                                                             \
00173         header.elements[header.length] = value;                       \
00174         header.length++;                                              \
00175     } while (0)
00176
00196 #define ada_insert(type, header, value, index) do {
        \
00197     ADA_ASSERT((int)(index) >= 0);
        \
00198     ADA_ASSERT((float)(index) - (int)(index) == 0);
        \
00199     ada_append(type, header, header.elements[header.length-1]);
        \
00200     for (size_t ada_for_loop_index = header.length-2; ada_for_loop_index > (index);
      ada_for_loop_index--) { \
00201         header.elements[ada_for_loop_index] = header.elements [ada_for_loop_index-1];
        \
00202     }
        \
00203     header.elements[(index)] = value;
        \
00204 } while (0)
00205
00206
00222 #define ada_insert_unordered(type, header, value, index) do {    \
00223     ADA_ASSERT((int)(index) >= 0);                               \
00224     ADA_ASSERT((float)(index) - (int)(index) == 0);              \
00225     if ((size_t)(index) == header.length) {                      \
00226         ada_append(type, header, value);                         \
00227     } else {                                                     \
00228         ada_append(type, header, header.elements[(index)]);      \
00229         header.elements[(index)] = value;                        \
00230     }                                                            \
00231 } while (0)
00232
00246 #define ada_remove(type, header, index) do {
        \
00247     ADA_ASSERT((int)(index) >= 0);
        \
00248     ADA_ASSERT((float)(index) - (int)(index) == 0);
        \
00249     for (size_t ada_for_loop_index = (index); ada_for_loop_index < header.length-1;
      ada_for_loop_index++) { \
00250         header.elements[ada_for_loop_index] = header.elements[ada_for_loop_index+1];
        \
00251     }
        \
00252     header.length--;
        \
00253 } while (0)
00254
00267 #define ada_remove_unordered(type, header, index) do {           \
00268     ADA_ASSERT((int)(index) >= 0);                               \
00269     ADA_ASSERT((float)(index) - (int)(index) == 0);              \
00270     header.elements[index] = header.elements[header.length-1];   \
00271     header.length--;                                             \
00272 } while (0)
00273
00274
00275 #endif /*ALMOG_DYNAMIC_ARRAY_H_*/
```

4.7 src/include/Almog_Engine.h File Reference

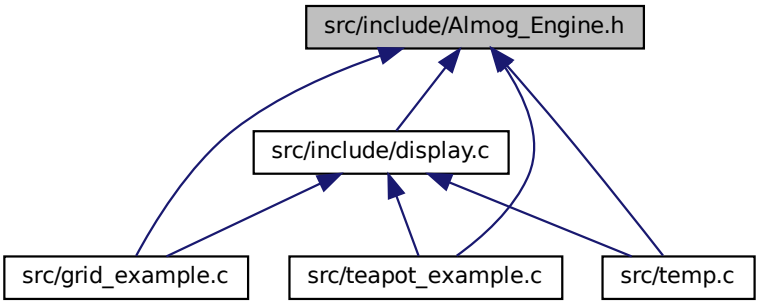
Software 3D rendering and scene utilities for meshes, camera, and projection.

```
#include "./Almog_Dynamic_Array.h"
#include "./Matrix2D.h"
#include "./Almog_Draw_Library.h"
#include "./Almog_String_Manipulation.h"
#include <assert.h>
#include <math.h>
#include <stdbool.h>
#include <float.h>
#include <stdint.h>
#include <errno.h>
#include <string.h>
```

Include dependency graph for Almog_Engine.h:



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



Classes

- struct Tri_mesh_array
- struct Quad_mesh_array
- struct Camera
- struct Light_source
- struct Material
- struct Scene

Macros

- #define AE_ASSERT assert
- #define STL_HEADER_SIZE 80
- #define STL_NUM_SIZE 4
- #define STL_SIZE_FOREACH_TRI 50
- #define STL_ATTRIBUTE_BITS_SIZE 2
- #define ARGB_hexARGB(a, r, g, b) 0x01000000l∗(uint8_t)(a) + 0x00010000∗(uint8_t)(r) + 0x00000100∗(uint8↩ _t)(g) + 0x00000001∗(uint8_t)(b)
- #define AE_MAX_POINT_VAL 1e5
- #define ae_assert_point_is_valid(p)
- #define ae_assert_tri_is_valid(tri)
- #define ae_assert_quad_is_valid(quad)
- #define ae_point_normalize_xyz_norma(p, norma)
- #define ae_point_calc_norma(p) sqrt(((p).x ∗ (p).x) + ((p).y ∗ (p).y) + ((p).z ∗ (p).z))
- #define ae_point_add_point(p, p1, p2)
- #define ae_point_sub_point(p, p1, p2)
- #define ae_point_dot_point(p1, p2) (((p1).x ∗ (p2).x) + ((p1).y ∗ (p2).y) + ((p1).z ∗ (p2).z))
- #define ae_point_mult(p, const)
- #define ae_points_equal(p1, p2) (p1).x == (p2).x && (p1).y == (p2).y && (p1).z == (p2).z
- #define TRI_MESH_ARRAY
- #define QUAD_MESH_ARRAY
- #define AE_PRINT_TRI(tri) ae_print_tri(tri, #tri, 0)
- #define AE_PRINT_MESH(mesh) ae_print_tri_mesh(mesh, #mesh, 0)

Enumerations

- enum Lighting_mode { AE_LIGHTING_FLAT , AE_LIGHTING_SMOOTH , AE_LIGHTING_MODE_LENGTH }

Functions

- Tri ae_tri_create (Point p1, Point p2, Point p3)

  Create a triangle from three points.
- void ae_tri_mesh_create_copy (Tri_mesh ∗des, Tri ∗src_elements, size_t len)

  Append copies of triangles to a destination Tri_mesh (resets destination length first).
- void ae_camera_init (Scene ∗scene, int window_h, int window_w)

  Initialize the camera part of a Scene.
- void ae_camera_free (Scene ∗scene)

  Free camera-related allocations in a Scene.
- Scene ae_scene_init (int window_h, int window_w)

  Create and initialize a Scene.
- void ae_scene_free (Scene ∗scene)

  Free all resources owned by a Scene.
- void ae_camera_reset_pos (Scene ∗scene)

  Reset camera orientation and position to initial state.
- void ae_point_to_mat2D (Point p, Mat2D m)

  Write a Point into a Mat2D vector.
- Point ae_mat2D_to_point (Mat2D m)

  Read a 3x1 Mat2D vector into a Point.
- Tri_mesh ae_tri_mesh_get_from_obj_file (char ∗file_path)

Load a triangle mesh from a Wavefront OBJ file.

- Tri_mesh ae_tri_mesh_get_from_stl_file (char *file_path)

  Load a triangle mesh from a binary STL file.
- Tri_mesh ae_tri_mesh_get_from_file (char *file_path)

  Load a triangle mesh from a file (OBJ or STL).
- void ae_tri_mesh_append_copy (Tri_mesh_array *mesh_array, Tri_mesh mesh)

  Append a copy of a Tri_mesh into a Tri_mesh_array.
- Tri_mesh ae_tri_mesh_get_from_quad_mesh (Quad_mesh q_mesh)

  Convert a Quad_mesh into a Tri_mesh.
- void ae_print_points (Curve p)

  Print a list of points to stdout.
- void ae_print_tri (Tri tri, char *name, size_t padding)

  Print a triangle to stdout.
- void ae_print_tri_mesh (Tri_mesh mesh, char *name, size_t padding)

  Print all triangles in a mesh to stdout.
- Point ae_point_normalize_xyz (Point p)

  Normalize a point's xyz to unit length.
- void ae_tri_set_normals (Tri *tri)

  Compute and set per-vertex normals for a triangle.
- Point ae_tri_get_average_normal (Tri tri)

  Compute the average of the three vertex normals of a triangle.
- Point ae_tri_get_average_point (Tri tri)

  Compute the average of the three vertices of a triangle.
- void ae_tri_calc_normal (Mat2D normal, Tri tri)

  Compute the face normal of a triangle.
- void ae_tri_mesh_translate (Tri_mesh mesh, float x, float y, float z)

  Translate a triangle mesh by (x, y, z).
- void ae_tri_mesh_rotate_Euler_xyz (Tri_mesh mesh, float phi_deg, float theta_deg, float psi_deg)

  Rotate a triangle mesh using XYZ Euler angles (degrees).
- void ae_tri_mesh_set_bounding_box (Tri_mesh mesh, float *x_min, float *x_max, float *y_min, float *y_max, float *z_min, float *z_max)

  Compute the axis-aligned bounding box of a triangle mesh.
- void ae_tri_mesh_normalize (Tri_mesh mesh)

  Normalize mesh coordinates to [-1, 1], centered at origin.
- void ae_tri_mesh_flip_normals (Tri_mesh mesh)

  Flip triangle winding and recompute per-vertex normals.
- void ae_tri_mesh_set_normals (Tri_mesh mesh)

  Recompute per-vertex normals for all triangles in a mesh.
- void ae_quad_set_normals (Quad *quad)

  Compute and set per-vertex normals for a quad.
- Point ae_quad_get_average_normal (Quad quad)

  Compute the average of the four vertex normals of a quad.
- Point ae_quad_get_average_point (Quad quad)

  Compute the average of the four vertices of a quad.
- void ae_quad_calc_normal (Mat2D normal, Quad quad)

  Compute the face normal of a quad using the first three vertices.
- void ae_curve_copy (Curve *des, Curve src)

  Copy a Curve (ADA array of points).
- void ae_tri_calc_light_intensity (Tri *tri, Scene *scene, Lighting_mode lighting_mode)

  Compute per-vertex lighting intensity for a triangle.
- void ae_quad_calc_light_intensity (Quad *quad, Scene *scene, Lighting_mode lighting_mode)

Compute per-vertex lighting intensity for a quad.

- Point ae_line_itersect_plane (Mat2D plane_p, Mat2D plane_n, Mat2D line_start, Mat2D line_end, float *t)

  Intersect a line segment with a plane.
- int ae_line_clip_with_plane (Point start_in, Point end_in, Mat2D plane_p, Mat2D plane_n, Point *start_out, Point *end_out)

  Clip a line segment against a plane.
- float ae_signed_dist_point_and_plane (Point p, Mat2D plane_p, Mat2D plane_n)

  Signed distance from a point to a plane.
- int ae_tri_clip_with_plane (Tri tri_in, Mat2D plane_p, Mat2D plane_n, Tri *tri_out1, Tri *tri_out2)

  Clip a triangle against a plane.
- int ae_quad_clip_with_plane (Quad quad_in, Mat2D plane_p, Mat2D plane_n, Quad *quad_out1, Quad *quad_out2)

  Clip a quad against a plane.
- void ae_projection_mat_set (Mat2D proj_mat, float aspect_ratio, float FOV_deg, float z_near, float z_far)

  Build a perspective projection matrix.
- void ae_view_mat_set (Mat2D view_mat, Camera camera, Mat2D up)

  Build a right-handed view matrix from a Camera and up vector.
- Point ae_point_project_world2screen (Mat2D view_mat, Mat2D proj_mat, Point src, int window_w, int window_h)

  Project a point from world space directly to screen space.
- Point ae_point_project_world2view (Mat2D view_mat, Point src)

  Transform a point from world space to view space.
- Point ae_point_project_view2screen (Mat2D proj_mat, Point src, int window_w, int window_h)

  Project a view-space point to screen space.
- void ae_line_project_world2screen (Mat2D view_mat, Mat2D proj_mat, Point start_src, Point end_src, int window_w, int window_h, Point *start_des, Point *end_des, Scene *scene)

  Project and near-clip a world-space line segment to screen space.
- Tri ae_tri_transform_to_view (Mat2D view_mat, Tri tri)

  Transform a triangle from world space to view space.
- Tri_mesh ae_tri_project_world2screen (Mat2D proj_mat, Mat2D view_mat, Tri tri, int window_w, int window_h, Scene *scene, Lighting_mode lighting_mode)

  Project a single world-space triangle to screen space with clipping.
- void ae_tri_mesh_project_world2screen (Mat2D proj_mat, Mat2D view_mat, Tri_mesh *des, Tri_mesh src, int window_w, int window_h, Scene *scene, Lighting_mode lighting_mode)

  Project a triangle mesh from world to screen space with clipping.
- Quad ae_quad_transform_to_view (Mat2D view_mat, Quad quad)

  Transform a quad from world space to view space.
- Quad_mesh ae_quad_project_world2screen (Mat2D proj_mat, Mat2D view_mat, Quad quad, int window_w, int window_h, Scene *scene, Lighting_mode lighting_mode)

  Project a single world-space quad to screen space with clipping.
- void ae_quad_mesh_project_world2screen (Mat2D proj_mat, Mat2D view_mat, Quad_mesh *des, Quad_mesh src, int window_w, int window_h, Scene *scene, Lighting_mode lighting_mode)

  Project a quad mesh from world to screen space with clipping.
- void ae_curve_project_world2screen (Mat2D proj_mat, Mat2D view_mat, Curve *des, Curve src, int window_w, int window_h, Scene *scene)

  Project and clip a polyline (Curve) from world to screen space.
- void ae_curve_ada_project_world2screen (Mat2D proj_mat, Mat2D view_mat, Curve_ada *des, Curve_ada src, int window_w, int window_h, Scene *scene)

  Project and clip an array of polylines from world to screen space.
- void ae_grid_project_world2screen (Mat2D proj_mat, Mat2D view_mat, Grid des, Grid src, int window_w, int window_h, Scene *scene)

  Project and clip all polylines in a Grid from world to screen.

- void ae_tri_swap (Tri *v, int i, int j)

  Swap two triangles in an array.
- bool ae_tri_compare (Tri t1, Tri t2)

  Compare two triangles for sorting by depth.
- void ae_tri_qsort (Tri *v, int left, int right)

  Quicksort an array of triangles by depth.
- double ae_linear_map (double s, double min_in, double max_in, double min_out, double max_out)

  Linearly map a scalar from one range to another.
- void ae_z_buffer_copy_to_screen (Mat2D_uint32 screen_mat, Mat2D inv_z_buffer)

  Visualize an inverse-z buffer by writing a grayscale image.

4.7.1 Detailed Description

Software 3D rendering and scene utilities for meshes, camera, and projection.

A small, header-driven 3D engine providing:

- Scene and camera setup (projection/view matrices, Euler navigation).
- Triangle and quad mesh loading (OBJ/ASCII+binary STL), normalization, transforms, and per-vertex/face normals.
- Back-face culling, near-plane and screen-space polygon clipping.
- Perspective projection (world->view->screen) and line/grid helpers.
- Basic Phong-like lighting (ambient, diffuse, specular) with flat/smooth modes.
- Simple z-buffer visualization utility.

Inspiration This code is heavily inspired by the 3D engine of 'OneLoneCoder' in C++. You can find the source code in: https://github.com/OneLoneCoder/Javidx9/tree/master/ConsoleGameEngine/↩BiggerProjects/Engine3D . featured in this video of his: https://youtu.be/ih20l3pJoe↩U?si=CzQ8rjk5ZEOlqEHN.

Note

- Depends on Almog_Dynamic_Array.h, Matrix2D.h, Almog_Draw_Library.h, and Almog_String_Manipulation.h for math, containers, and I/O utilities.

- All public functions are prefixed with 'ae_'.

- Define ALMOG_ENGINE_IMPLEMENTATION in exactly one translation unit to compile the function bodies.

Definition in file Almog_Engine.h.

4.7.2 Macro Definition Documentation

4.7.2.1 AE_ASSERT

```
#define AE_ASSERT assert
```

Definition at line 44 of file Almog_Engine.h.

4.7.2.2 ae_assert_point_is_valid

```
#define ae_assert_point_is_valid(
              p )
```

Value:

```
AE_ASSERT(isfinite((p).x) && isfinite((p).y) && isfinite((p).z) && isfinite((p).w));    \
AE_ASSERT((p).x > -AE_MAX_POINT_VAL && (p).x < AE_MAX_POINT_VAL);
                                    \
AE_ASSERT((p).y > -AE_MAX_POINT_VAL && (p).y < AE_MAX_POINT_VAL);
                                    \
AE_ASSERT((p).z > -AE_MAX_POINT_VAL && (p).z < AE_MAX_POINT_VAL);
                                    \
AE_ASSERT((p).w > -AE_MAX_POINT_VAL && (p).w < AE_MAX_POINT_VAL);
```

Definition at line 89 of file Almog_Engine.h.

4.7.2.3 ae_assert_quad_is_valid

```
#define ae_assert_quad_is_valid(
              quad )
```

Value:

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

Definition at line 97 of file Almog_Engine.h.

4.7.2.4 ae_assert_tri_is_valid

```
#define ae_assert_tri_is_valid(
              tri )
```

Value:

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

Definition at line 94 of file Almog_Engine.h.

4.7.2.5 AE_MAX_POINT_VAL

```
#define AE_MAX_POINT_VAL 1e5
```

Definition at line 88 of file Almog_Engine.h.

4.7.2.6 ae_point_add_point

```
#define ae_point_add_point(
              p,
              p1,
              p2 )
```

Value:

```
(p).x = (p1).x + (p2).x;  \
(p).y = (p1).y + (p2).y;                                \
(p).z = (p1).z + (p2).z;                                \
(p).w = (p1).w + (p2).w
```

Definition at line 105 of file Almog_Engine.h.

4.7.2.7 ae_point_calc_norma

```
#define ae_point_calc_norma(
              p ) sqrt(((p).x * (p).x) + ((p).y * (p).y) + ((p).z * (p).z))
```

Definition at line 104 of file Almog_Engine.h.

4.7.2.8 ae_point_dot_point

```
#define ae_point_dot_point(
              p1,
              p2 ) (((p1).x * (p2).x) + ((p1).y * (p2).y) + ((p1).z * (p2).z))
```

Definition at line 113 of file Almog_Engine.h.

4.7.2.9 ae_point_mult

```
#define ae_point_mult(
              p,
              const )
```

Value:

```
(p).x *= const; \
(p).y *= const;                         \
(p).z *= const
```

Definition at line 114 of file Almog_Engine.h.

4.7.2.10 ae_point_normalize_xyz_norma

```
#define ae_point_normalize_xyz_norma(
              p,
              norma )
```

Value:

```
(p).x = (p).x / norma;  \
(p).y = (p).y / norma;                    \
(p).z = (p).z / norma
```

Definition at line 101 of file Almog_Engine.h.

4.7.2.11 ae_point_sub_point

```
#define ae_point_sub_point(
              p,
              p1,
              p2 )
```

Value:

```
(p).x = (p1).x - (p2).x;  \
(p).y = (p1).y - (p2).y;                              \
(p).z = (p1).z - (p2).z;                              \
(p).w = (p1).w - (p2).w
```

Definition at line 109 of file Almog_Engine.h.

4.7.2.12 ae_points_equal

```
#define ae_points_equal(
              p1,
              p2 ) (p1).x == (p2).x && (p1).y == (p2).y && (p1).z == (p2).z
```

Definition at line 117 of file Almog_Engine.h.

4.7.2.13 AE_PRINT_MESH

```
#define AE_PRINT_MESH(
              mesh ) ae_print_tri_mesh(mesh, #mesh, 0)
```

Definition at line 267 of file Almog_Engine.h.

4.7.2.14 AE_PRINT_TRI

```
#define AE_PRINT_TRI(
              tri ) ae_print_tri(tri, #tri, 0)
```

Definition at line 266 of file Almog_Engine.h.

4.7.2.15 ARGB_hexARGB

```
#define ARGB_hexARGB(
              a,
              r,
              g,
              b ) 0x01000000l*(uint8_t)(a) + 0x00010000*(uint8_t)(r) + 0x00000100*(uint8_t)(g)
+ 0x00000001*(uint8_t)(b)
```

Definition at line 83 of file Almog_Engine.h.

4.7.2.16 QUAD_MESH_ARRAY

```
#define QUAD_MESH_ARRAY
```

Definition at line 136 of file Almog_Engine.h.

4.7.2.17 STL_ATTRIBUTE_BITS_SIZE

```
#define STL_ATTRIBUTE_BITS_SIZE 2
```

Definition at line 71 of file Almog_Engine.h.

4.7.2.18 STL_HEADER_SIZE

```
#define STL_HEADER_SIZE 80
```

Definition at line 59 of file Almog_Engine.h.

4.7.2.19 STL_NUM_SIZE

```
#define STL_NUM_SIZE 4
```

Definition at line 63 of file Almog_Engine.h.

4.7.2.20 STL_SIZE_FOREACH_TRI

```
#define STL_SIZE_FOREACH_TRI 50
```

Definition at line 67 of file Almog_Engine.h.

4.7.2.21 TRI_MESH_ARRAY

```
#define TRI_MESH_ARRAY
```

Definition at line 127 of file Almog_Engine.h.

4.7.3 Enumeration Type Documentation

4.7.3.1 Lighting_mode

```
enum Lighting_mode
```

Enumerator

AE_LIGHTING_FLAT	
AE_LIGHTING_SMOOTH	
AE_LIGHTING_MODE_LENGTH	

Definition at line 120 of file Almog_Engine.h.

4.7.4 Function Documentation

4.7.4.1 ae_camera_free()

```
void ae_camera_free (
            Scene * scene )
```

Free camera-related allocations in a Scene.

Frees all Mat2D objects owned by scene->camera (init_position, current_position, offset_position, direction, camera_x/y/z).

Parameters

scene	Scene whose camera resources will be freed.

Definition at line 366 of file Almog_Engine.h.

References Scene::camera, Camera::camera_x, Camera::camera_y, Camera::camera_z, Camera::current_position, Camera::direction, Camera::init_position, mat2D_free(), and Camera::offset_position.

Referenced by ae_scene_free().

4.7.4.2 ae_camera_init()

```
void ae_camera_init (
            Scene * scene,
            int window_h,
            int window_w )
```

Initialize the camera part of a Scene.

Sets perspective parameters (z_near, z_far, fov, aspect_ratio), allocates camera matrices/vectors, and sets initial position and orientation. The aspect ratio is computed as window_h / window_w.

Parameters

scene	Scene whose camera will be initialized.
window_h	Window height in pixels.
window_w	Window width in pixels.

Definition at line 320 of file Almog_Engine.h.

References Camera::aspect_ratio, Scene::camera, Camera::camera_x, Camera::camera_y, Camera::camera_z, Camera::current_position, Camera::direction, Camera::fov_deg, Camera::init_position, mat2D_alloc(), MAT2D_AT, mat2D_copy(), mat2D_fill(), Camera::offset_position, Camera::pitch_offset_deg, Camera::roll_offset_deg, Camera::yaw_offset_deg, Camera::z_far, and Camera::z_near.

Referenced by ae_scene_init().

4.7.4.3 ae_camera_reset_pos()

```
void ae_camera_reset_pos (
            Scene * scene )
```

Reset camera orientation and position to initial state.

Resets roll/pitch/yaw to zero, clears offset_position, restores camera basis vectors to identity, and copies current↩ _position from init_position.

Parameters

scene	Scene containing the camera to reset.

Definition at line 472 of file Almog_Engine.h.

References Scene::camera, Camera::camera_x, Camera::camera_y, Camera::camera_z, Camera::current_position, Camera::init_position, MAT2D_AT, mat2D_copy(), mat2D_fill(), Camera::offset_position, Camera::pitch_offset_deg, Camera::roll_offset_deg, and Camera::yaw_offset_deg.

Referenced by process_input_window().

4.7.4.4 ae_curve_ada_project_world2screen()

```
void ae_curve_ada_project_world2screen (
            Mat2D proj_mat,
            Mat2D view_mat,
            Curve_ada * des,
            Curve_ada src,
            int window_w,
            int window_h,
            Scene * scene )
```

Project and clip an array of polylines from world to screen space.

Applies ae_curve_project_world2screen to each element in src and writes into the corresponding element in des. Arrays must be the same length.

Parameters

proj_mat	Projection matrix (4x4).
view_mat	View matrix (4x4).
des	Output array of curves (each overwritten).
src	Input array of world-space curves.
window↩ _w	Screen width in pixels.
window↩ _h	Screen height in pixels.
scene	Scene (camera for near plane).

Definition at line 3670 of file Almog_Engine.h.

References ae_curve_project_world2screen(), Curve_ada::elements, and Curve_ada::length.

4.7.4.5 ae_curve_copy()

```
void ae_curve_copy (
            Curve * des,
            Curve src )
```

Copy a Curve (ADA array of points).

Clears destination length and appends all points from src.

Parameters

des	Destination curve (modified/grown as needed).
src	Source curve.

Definition at line 1461 of file Almog_Engine.h.

References ada_append, Curve::elements, and Curve::length.

Referenced by ae_curve_project_world2screen().

4.7.4.6 ae_curve_project_world2screen()

```
void ae_curve_project_world2screen (
            Mat2D proj_mat,
            Mat2D view_mat,
            Curve * des,
            Curve src,
            int window_w,
            int window_h,
            Scene * scene )
```

Project and clip a polyline (Curve) from world to screen space.

Projects each segment with near-plane clipping and screen-edge clipping. Segments fully outside are removed. The destination curve is overwritten.

Note

This solution is not prefect. It sometimes delete one more edge then necessary, but I think that it won't brake.

Parameters

proj_mat	Projection matrix (4x4).
view_mat	View matrix (4x4).
des	Output curve (overwritten; ADA array grown as needed).
src	Input world-space curve.
window_w	Screen width in pixels.
window_h	Screen height in pixels.
scene	Scene (camera for near plane).

Definition at line 3558 of file Almog_Engine.h.

References ada_remove, ae_curve_copy(), ae_line_clip_with_plane(), ae_line_project_world2screen(), ae_points_equal, Curve::elements, Curve::length, mat2D_alloc(), MAT2D_AT, mat2D_fill(), and mat2D_free().

Referenced by ae_curve_ada_project_world2screen(), and ae_grid_project_world2screen().

4.7.4.7 ae_grid_project_world2screen()

```
void ae_grid_project_world2screen (
            Mat2D proj_mat,
            Mat2D view_mat,
            Grid des,
            Grid src,
            int window_w,
            int window_h,
            Scene * scene )
```

Project and clip all polylines in a Grid from world to screen.

Applies ae_curve_project_world2screen to each curve in the grid.

Parameters

proj_mat	Projection matrix (4x4).
view_mat	View matrix (4x4).
des	Output grid (curves overwritten).
src	Input world-space grid.
window_w	Screen width in pixels.
window_h	Screen height in pixels.
scene	Scene (camera for near plane).

Definition at line 3691 of file Almog_Engine.h.

References ae_curve_project_world2screen(), Grid::curves, Curve_ada::elements, and Curve_ada::length.

Referenced by update().

4.7.4.8 ae_line_clip_with_plane()

```
int ae_line_clip_with_plane (
            Point start_in,
            Point end_in,
            Mat2D plane_p,
            Mat2D plane_n,
```

```
            Point * start_out,
            Point * end_out )
```

Clip a line segment against a plane.

Returns the portion of the line segment [start_in, end_in] that lies on or inside the plane (signed distance >= 0). plane_n is normalized inside the function.

Parameters

start_in	Input start point (world or view space).
end_in	Input end point.
plane_p	Plane reference point (3x1).
plane_n	Plane normal (3x1).
start_out	Output clipped start point (if visible).
end_out	Output clipped end point (if visible).

Returns

int 0 if fully outside, 1 if fully or partially inside (outputs are valid), -1 on error.

Definition at line 1720 of file Almog_Engine.h.

References ae_assert_point_is_valid, ae_line_itersect_plane(), ae_point_to_mat2D(), ae_signed_dist_point_and_plane(), mat2D_alloc(), mat2D_free(), and mat2D_normalize.

Referenced by ae_curve_project_world2screen(), and ae_line_project_world2screen().

4.7.4.9 ae_line_itersect_plane()

```
Point ae_line_itersect_plane (
            Mat2D plane_p,
            Mat2D plane_n,
            Mat2D line_start,
            Mat2D line_end,
            float * t )
```

Intersect a line segment with a plane.

Computes intersection of segment [line_start, line_end] with the plane defined by point plane_p and normal plane_n. plane_n is normalized inside the function. The parameter t in [0, 1] is returned via out-parameter.

Note

The Mat2D objects line_start and line_end are temporarily modified internally; pass copies if you must preserve their values.

Parameters

plane_p	Plane reference point (3x1).
plane_n	Plane normal (3x1).
line_start	Segment start point (3x1).
line_end	Segment end point (3x1).
t	Output parametric distance along the segment (0=start, 1=end).

Returns

Point Intersection point in 3D.

Definition at line 1680 of file Almog_Engine.h.

References ae_mat2D_to_point(), mat2D_add(), mat2D_alloc(), mat2D_dot_product(), mat2D_fill(), mat2D_free(), mat2D_mult(), mat2D_normalize, and mat2D_sub().

Referenced by ae_line_clip_with_plane(), ae_quad_clip_with_plane(), and ae_tri_clip_with_plane().

4.7.4.10 ae_line_project_world2screen()

```
void ae_line_project_world2screen (
            Mat2D view_mat,
            Mat2D proj_mat,
            Point start_src,
            Point end_src,
            int window_w,
            int window_h,
            Point * start_des,
            Point * end_des,
            Scene * scene )
```

Project and near-clip a world-space line segment to screen space.

Transforms the segment to view space, clips it against the near plane at z = z_near + 0.01, then projects to screen space. If fully clipped, both outputs are set to the sentinel (-1, -1, 1, 1).

Parameters

view_mat	View matrix (4x4).
proj_mat	Projection matrix (4x4).
start_src	World-space start point.
end_src	World-space end point.
window_w	Screen width in pixels.
window_h	Screen height in pixels.
start_des	Output screen-space start point (or sentinel).
end_des	Output screen-space end point (or sentinel).
scene	Scene (used for near plane distance).

Definition at line 2933 of file Almog_Engine.h.

References ae_line_clip_with_plane(), ae_point_project_view2screen(), ae_point_project_world2view(), Scene::camera, mat2D_alloc(), MAT2D_AT, mat2D_fill(), mat2D_free(), and Camera::z_near.

Referenced by ae_curve_project_world2screen().

4.7.4.11 ae_linear_map()

```
double ae_linear_map (
            double s,
            double min_in,
            double max_in,
            double min_out,
            double max_out )
```

Linearly map a scalar from one range to another.

Computes min_out + (s - min_in) * (max_out - min_out) / (max_in - min_in).

Parameters

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

Returns

double Mapped scalar.

Definition at line 3770 of file Almog_Engine.h.

Referenced by ae_z_buffer_copy_to_screen().

4.7.4.12 ae_mat2D_to_point()

```
Point ae_mat2D_to_point (
            Mat2D m )
```

Read a 3x1 Mat2D vector into a Point.

Reads x, y, z from m(0..2,0) and returns a Point with w=1.

Parameters

m	Source matrix (3x1).

Returns

Point The corresponding point with w=1.

Definition at line 522 of file Almog_Engine.h.

References MAT2D_AT, and Point::x.

Referenced by ae_line_itersect_plane(), ae_quad_calc_light_intensity(), ae_quad_set_normals(), ae_tri_calc_light_intensity(), and ae_tri_set_normals().

4.7.4.13 ae_point_normalize_xyz()

```
Point ae_point_normalize_xyz (
            Point p )
```

Normalize a point's xyz to unit length.

Divides x, y, z by their Euclidean norm. w is preserved unchanged.

Parameters

p	Input point.

Returns

Point Unit-length point (xyz), with original w.

Definition at line 976 of file Almog_Engine.h.

References ae_point_calc_norma, Point::w, Point::x, Point::y, and Point::z.

Referenced by ae_quad_calc_light_intensity(), ae_quad_get_average_normal(), ae_scene_init(), ae_tri_calc_light_intensity(), and ae_tri_get_average_normal().

4.7.4.14 ae_point_project_view2screen()

```
Point ae_point_project_view2screen (
            Mat2D proj_mat,
            Point src,
            int window_w,
            int window_h )
```

Project a view-space point to screen space.

Applies the projection matrix, performs perspective divide if $|w| > 1e\text{-}3$, maps normalized device coords to pixel coordinates: x_screen = (x_ndc + 1) $*$ 0.5 $*$ window_w y_screen = (y_ndc + 1) $*$ 0.5 $*$ window_h

z is z_ndc, w is the clip-space w (or 1 if the original w $\sim$ 0).

Parameters

proj_mat	Projection matrix (4x4).
src	View-space point.
window↩ _w	Screen width in pixels.
window↩ _h	Screen height in pixels.

Returns

Point Screen-space point.

Definition at line 2870 of file Almog_Engine.h.

References ae_assert_point_is_valid, mat2D_alloc(), MAT2D_AT, mat2D_dot(), mat2D_free(), Point::w, Point::x, Point::y, and Point::z.

Referenced by ae_line_project_world2screen(), ae_point_project_world2screen(), ae_quad_project_world2screen(), and ae_tri_project_world2screen().

4.7.4.15 ae_point_project_world2screen()

```
Point ae_point_project_world2screen (
            Mat2D view_mat,
            Mat2D proj_mat,
            Point src,
            int window_w,
            int window_h )
```

Project a point from world space directly to screen space.

Combines ae_point_project_world2view and ae_point_project_view2screen.

Parameters

view_mat	View matrix (4x4).
proj_mat	Projection matrix (4x4).
src	World-space point.
window↩ _w	Screen width in pixels.
window↩ _h	Screen height in pixels.

Returns

Point Screen-space point (x,y in pixels). z is post-projection z/w, w is clip-space w.

Definition at line 2806 of file Almog_Engine.h.

References ae_point_project_view2screen(), and ae_point_project_world2view().

4.7.4.16 ae_point_project_world2view()

```
Point ae_point_project_world2view (
            Mat2D view_mat,
            Point src )
```

Transform a point from world space to view space.

Multiplies [x y z 1] by view_mat (row-vector convention in this code). Returns the resulting view-space point; w should be 1.

Parameters

view_mat	View matrix (4x4).
src	World-space point.

Returns

Point View-space point (w=1).

Definition at line 2824 of file Almog_Engine.h.

References AE_ASSERT, ae_assert_point_is_valid, mat2D_alloc(), MAT2D_AT, mat2D_dot(), mat2D_free(), Point::w, Point::x, Point::y, and Point::z.

Referenced by ae_line_project_world2screen(), and ae_point_project_world2screen().

4.7.4.17 ae_point_to_mat2D()

```
void ae_point_to_mat2D (
            Point p,
            Mat2D m )
```

Write a Point into a Mat2D vector.

Writes p into m. m must be either 3x1 or 1x3. Only x, y, z are written.

Parameters

p	Source point (x, y, z used; w ignored).
m	Destination matrix (3x1 or 1x3).

Definition at line 498 of file Almog_Engine.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, Mat2D::rows, Point::x, Point::y, and Point::z.

Referenced by ae_line_clip_with_plane(), ae_quad_calc_normal(), ae_quad_clip_with_plane(), ae_quad_project_world2screen(), ae_quad_set_normals(), ae_tri_calc_normal(), ae_tri_clip_with_plane(), ae_tri_project_world2screen(), and ae_tri_set_normals().

4.7.4.18 ae_print_points()

```
void ae_print_points (
            Curve p )
```

Print a list of points to stdout.

Each point is printed as: "point i: (x, y, z)".

Parameters

p	Curve of points to print.

Definition at line 925 of file Almog_Engine.h.

References Curve::elements, Curve::length, Point::x, Point::y, and Point::z.

4.7.4.19 ae_print_tri()

```
void ae_print_tri (
            Tri tri,
            char * name,
            size_t padding )
```

Print a triangle to stdout.

Prints the triangle's vertices and draw flag, with an optional name and indentation padding (spaces).

Parameters

tri	Triangle to print.
name	Label to print before the triangle.
padding	Number of leading spaces for indentation.

Definition at line 942 of file Almog_Engine.h.

References Tri::points, Tri::to_draw, Point::x, Point::y, and Point::z.

Referenced by ae_print_tri_mesh().

4.7.4.20 ae_print_tri_mesh()

```
void ae_print_tri_mesh (
            Tri_mesh mesh,
            char * name,
            size_t padding )
```

Print all triangles in a mesh to stdout.

Each triangle is printed via ae_print_tri with the given padding.

Parameters

mesh	Triangle mesh to print.
name	Label for the mesh.
padding	Number of leading spaces for indentation.

Definition at line 958 of file Almog_Engine.h.

References ae_print_tri(), Tri_mesh::elements, and Tri_mesh::length.

4.7.4.21 ae_projection_mat_set()

```
void ae_projection_mat_set (
            Mat2D proj_mat,
            float aspect_ratio,
            float FOV_deg,
            float z_near,
            float z_far )
```

Build a perspective projection matrix.

proj_mat must be 4x4. FOV is in degrees. The matrix maps view-space to clip space consistent with the engine's pipeline; z is mapped using z_far/(z_far - z_near).

Parameters

proj_mat	Output 4x4 projection matrix.
aspect_ratio	aspect = window_h / window_w.
FOV_deg	Vertical field of view in degrees (must be $>$ 0).
z_near	Near clipping plane distance ($>$ 0).
z_far	Far clipping plane distance ($>$ z_near).

Definition at line 2682 of file Almog_Engine.h.

References AE_ASSERT, Mat2D::cols, MAT2D_AT, mat2D_fill(), PI, and Mat2D::rows.

Referenced by ae_scene_init(), and update().

4.7.4.22 ae_quad_calc_light_intensity()

```
void ae_quad_calc_light_intensity (
            Quad * quad,
            Scene * scene,
            Lighting_mode lighting_mode )
```

Compute per-vertex lighting intensity for a quad.

Same model as ae_tri_calc_light_intensity, applied to four vertices. Results are clamped to [0, 1].

Parameters

quad	Quad to update (quad->light_intensity[i] is written).
scene	Scene providing light and material parameters.
lighting_mode	Flat or smooth lighting mode.

Definition at line 1580 of file Almog_Engine.h.

References AE_LIGHTING_FLAT, AE_LIGHTING_SMOOTH, ae_mat2D_to_point(), ae_point_add_point, ae_point_dot_point, ae_point_mult, ae_point_normalize_xyz(), ae_point_sub_point, ae_quad_get_average_normal(), ae_quad_get_average_point(), Material::c_ambi, Material::c_diff, Material::c_spec, Scene::camera, Camera::current_position, Light_source::light_direction_or_pos, Quad::light_intensity, Light_source::light_intensity, Scene::light_source0, Scene::material0, Quad::normals, Quad::points, Material::specular_power_alpha, and Point::w.

Referenced by ae_quad_project_world2screen().

4.7.4.23 ae_quad_calc_normal()

```
void ae_quad_calc_normal (
            Mat2D normal,
            Quad quad )
```

Compute the face normal of a quad using the first three vertices.

normal must be a 3x1 vector. The function writes the normalized cross product of (p1 - p0) x (p2 - p0) into normal.

Parameters

normal	Output 3x1 vector for the face normal.
quad	Input quad.

Definition at line 1428 of file Almog_Engine.h.

References AE_ASSERT, ae_assert_quad_is_valid, ae_point_to_mat2D(), Mat2D::cols, mat2D_alloc(), mat2D_calc_norma(), mat2D_cross(), mat2D_free(), mat2D_mult(), mat2D_sub(), Quad::points, and Mat2D::rows.

4.7.4.24 ae_quad_clip_with_plane()

```
int ae_quad_clip_with_plane (
            Quad quad_in,
            Mat2D plane_p,
            Mat2D plane_n,
            Quad * quad_out1,
            Quad * quad_out2 )
```

Clip a quad against a plane.

Splits or discards the quad quad_in against the plane defined by (plane_p, plane_n). plane_n is normalized inside the function.

Parameters

quad_in	Input quad.
plane_p	Plane reference point (3x1).
plane_n	Plane normal (3x1).
quad_out1	First output quad (if any). When output count is 2, this holds one of the resulting polygons (possibly as a quad composed from intersections).
quad_out2	Second output quad (if split).

Returns

int Number of output polygons: 0 (culled), 1, or 2. Returns -1 on error.

Definition at line 2181 of file Almog_Engine.h.

References ae_assert_quad_is_valid, ae_line_itersect_plane(), ae_point_to_mat2D(), ae_signed_dist_point_and_plane(), Quad::colors, mat2D_alloc(), mat2D_free(), mat2D_normalize, Quad::points, and Point::w.

Referenced by ae_quad_mesh_project_world2screen(), and ae_quad_project_world2screen().

4.7.4.25 ae_quad_get_average_normal()

```
Point ae_quad_get_average_normal (
            Quad quad )
```

Compute the average of the four vertex normals of a quad.

Averages the four vertex normals and normalizes the result.

Parameters

quad	Input quad.

Returns

Point The averaged, normalized normal.

Definition at line 1379 of file Almog_Engine.h.

References ae_point_normalize_xyz(), Quad::normals, Point::w, Point::x, Point::y, and Point::z.

Referenced by ae_quad_calc_light_intensity(), and ae_quad_project_world2screen().

4.7.4.26 ae_quad_get_average_point()

```
Point ae_quad_get_average_point (
            Quad quad )
```

Compute the average of the four vertices of a quad.

Parameters

quad	Input quad.

Returns

Point The average point (x, y, z, w are simple averages).

Definition at line 1403 of file Almog_Engine.h.

References Quad::points, Point::w, Point::x, Point::y, and Point::z.

Referenced by ae_quad_calc_light_intensity().

4.7.4.27 ae_quad_mesh_project_world2screen()

```
void ae_quad_mesh_project_world2screen (
            Mat2D proj_mat,
            Mat2D view_mat,
            Quad_mesh * des,
            Quad_mesh src,
            int window_w,
            int window_h,
            Scene * scene,
            Lighting_mode lighting_mode )
```

Project a quad mesh from world to screen space with clipping.

Iterates over all quads, applies near-plane and screen-edge clipping, and writes results into des. Quads can be split by clipping, so des may end up with more elements than src.

Parameters

proj_mat	Projection matrix (4x4).
view_mat	View matrix (4x4).
des	Output mesh (cleared and filled; ADA array grown as needed).
src	Input world-space quad mesh.
window_w	Screen width in pixels.
window_h	Screen height in pixels.
scene	Scene (camera/light/material).
lighting_mode	Flat or smooth lighting mode.

Definition at line 3434 of file Almog_Engine.h.

References ada_append, ada_insert_unordered, ada_remove_unordered, ae_assert_quad_is_valid, ae_quad_clip_with_plane(), ae_quad_project_world2screen(), Quad_mesh::elements, Quad_mesh::length, mat2D_alloc(), MAT2D_AT, mat2D_fill(), and mat2D_free().

4.7.4.28 ae_quad_project_world2screen()

```
Quad_mesh ae_quad_project_world2screen (
            Mat2D proj_mat,
            Mat2D view_mat,
            Quad quad,
            int window_w,
            int window_h,
            Scene * scene,
            Lighting_mode lighting_mode )
```

Project a single world-space quad to screen space with clipping.

Computes lighting and visibility, transforms to view space, clips against near plane, and projects to screen space. A quad may produce one or two quads after clipping.

Parameters

proj_mat	Projection matrix (4x4).
view_mat	View matrix (4x4).
quad	World-space quad.
window_w	Screen width in pixels.
window_h	Screen height in pixels.
scene	Scene (camera/light/material).
lighting_mode	Flat or smooth lighting mode.

Returns

Quad_mesh An ADA array of resulting screen-space quads. Caller must free result.elements.

Definition at line 3321 of file Almog_Engine.h.

References ada_append, ada_init_array, ae_assert_quad_is_valid, ae_point_project_view2screen(), ae_point_to_mat2D(), ae_quad_calc_light_intensity(), ae_quad_clip_with_plane(), ae_quad_get_average_normal(), ae_quad_transform_to_view(), Scene::camera, Camera::current_position, Quad_mesh::elements, Quad_mesh::length, Light_source::light_direction_or_pos, Quad::light_intensity, Scene::light_source0, mat2D_alloc(), MAT2D_AT, mat2D_fill(), mat2D_free(), mat2D_sub(), Quad::normals, Quad::points, Quad::to_draw, Point::x, Point::y, Point::z, and Camera::z_near.

Referenced by ae_quad_mesh_project_world2screen().

4.7.4.29 ae_quad_set_normals()

```
void ae_quad_set_normals (
            Quad * quad )
```

Compute and set per-vertex normals for a quad.

For each vertex, computes the cross product of adjacent edges and normalizes the result. Results are stored in quad->normals[i].

Parameters

quad	Quad whose normals will be computed and written.

Definition at line 1336 of file Almog_Engine.h.

References ae_assert_quad_is_valid, ae_mat2D_to_point(), ae_point_to_mat2D(), mat2D_alloc(), mat2D_copy(), mat2D_cross(), mat2D_free(), mat2D_normalize, mat2D_sub(), Quad::normals, and Quad::points.

4.7.4.30 ae_quad_transform_to_view()

```
Quad ae_quad_transform_to_view (
            Mat2D view_mat,
            Quad quad )
```

Transform a quad from world space to view space.

Applies view_mat to each vertex (homogeneous multiply with w=1). Returns the transformed quad; normals are not changed.

Parameters

view_mat	View matrix (4x4).
quad	World-space quad.

Returns

Quad View-space quad.

Definition at line 3271 of file Almog_Engine.h.

References AE_ASSERT, ae_assert_quad_is_valid, mat2D_alloc(), MAT2D_AT, mat2D_dot(), mat2D_free(), Quad::points, Point::w, Point::x, Point::y, and Point::z.

Referenced by ae_quad_project_world2screen().

4.7.4.31 ae_scene_free()

```
void ae_scene_free (
            Scene * scene )
```

Free all resources owned by a Scene.

Frees camera, matrices, and any allocated meshes in the in_world, projected, and original mesh arrays (triangles and quads). Does not free the Scene struct itself when passed by pointer.

Parameters

scene	Scene to free.

Note

Assumes the game_state was initialized with zeros.

Definition at line 429 of file Almog_Engine.h.

References ae_camera_free(), Tri_mesh::elements, Quad_mesh::elements, Tri_mesh_array::elements, Quad_mesh_array::elements, Scene::in_world_quad_meshes, Scene::in_world_tri_meshes, Tri_mesh_array::length, Quad_mesh_array::length, mat2D_free(), Scene::original_quad_meshes, Scene::original_tri_meshes, Scene::proj_mat, Scene::projected_quad_meshes, Scene::projected_tri_meshes, Scene::up_direction, and Scene::view_mat.

Referenced by destroy_window().

4.7.4.32 ae_scene_init()

```
Scene ae_scene_init (
            int window_h,
            int window_w )
```

Create and initialize a Scene.

Initializes camera, up direction, default light and material, and allocates projection and view matrices. Caller must release resources with ae_scene_free.

Parameters

window_h	Window height in pixels.
window_w	Window width in pixels.

Returns

Scene An initialized scene object.

Definition at line 388 of file Almog_Engine.h.

References ae_camera_init(), ae_point_normalize_xyz(), ae_projection_mat_set(), ae_view_mat_set(), Camera::aspect_ratio, Material::c_ambi, Material::c_diff, Material::c_spec, Scene::camera, Camera::fov_deg, Light_source::light_direction_or_pos, Light_source::light_intensity, Scene::light_source0, mat2D_alloc(), MAT2D_AT, mat2D_fill(), Scene::material0, Scene::proj_mat, Material::specular_power_alpha, Scene::up_direction, Scene::view_mat, Point::w, Point::x, Point::y, Point::z, Camera::z_far, and Camera::z_near.

Referenced by setup_window().

4.7.4.33 ae_signed_dist_point_and_plane()

```
float ae_signed_dist_point_and_plane (
            Point p,
            Mat2D plane_p,
            Mat2D plane_n )
```

Signed distance from a point to a plane.

Computes dot(n, p) - dot(n, plane_p). The normal is not normalized internally; pass a normalized plane_n for distances in consistent units.

Parameters

p	Point to evaluate.
plane_p	Plane reference point (3x1).
plane_n	Plane normal (3x1).

Returns

float Signed distance (>=0 is on the "inside" of the plane).

Definition at line 1807 of file Almog_Engine.h.

References ae_assert_point_is_valid, MAT2D_AT, Point::x, Point::y, and Point::z.

Referenced by ae_line_clip_with_plane(), ae_quad_clip_with_plane(), and ae_tri_clip_with_plane().

4.7.4.34 ae_tri_calc_light_intensity()

```
void ae_tri_calc_light_intensity (
            Tri * tri,
            Scene * scene,
            Lighting_mode lighting_mode )
```

Compute per-vertex lighting intensity for a triangle.

Implements a Phong-like model with ambient, diffuse, and specular terms, using material0 and light_source0 from the scene. When lighting_mode is AE_LIGHTING_FLAT, the average normal and triangle centroid are used for all vertices; when AE_LIGHTING_SMOOTH, each vertex normal and position is used. For directional light, light_↩ direction_or_pos.w == 0; for point light, w != 0. Results are clamped to [0, 1].

Parameters

tri	Triangle to update (tri->light_intensity[i] is written).
scene	Scene providing light and material parameters.
lighting_mode	Flat or smooth lighting mode.

Definition at line 1487 of file Almog_Engine.h.

References AE_LIGHTING_FLAT, AE_LIGHTING_SMOOTH, ae_mat2D_to_point(), ae_point_add_point, ae_point_dot_point, ae_point_mult, ae_point_normalize_xyz(), ae_point_sub_point, ae_tri_get_average_normal(), ae_tri_get_average_point(), Material::c_ambi, Material::c_diff, Material::c_spec, Scene::camera, Camera::current_position, Light_source::light_direction_or_pos, Tri::light_intensity, Light_source::light_intensity, Scene::light_source0, Scene::material0, Tri::normals, Tri::points, Material::specular_power_alpha, and Point::w.

Referenced by ae_tri_project_world2screen().

4.7.4.35 ae_tri_calc_normal()

```
void ae_tri_calc_normal (
            Mat2D normal,
            Tri tri )
```

Compute the face normal of a triangle.

normal must be a 3x1 vector. The function writes the normalized cross product of (p1 - p0) x (p2 - p0) into normal.

Parameters

normal	Output 3x1 vector for the face normal.
tri	Input triangle.

Definition at line 1086 of file Almog_Engine.h.

References AE_ASSERT, ae_assert_tri_is_valid, ae_point_to_mat2D(), Mat2D::cols, mat2D_alloc(), mat2D_calc_norma(), mat2D_cross(), mat2D_free(), mat2D_mult(), mat2D_sub(), Tri::points, and Mat2D::rows.

Referenced by ae_tri_project_world2screen().

4.7.4.36 ae_tri_clip_with_plane()

```
int ae_tri_clip_with_plane (
            Tri tri_in,
            Mat2D plane_p,
            Mat2D plane_n,
            Tri * tri_out1,
            Tri * tri_out2 )
```

Clip a triangle against a plane.

Splits or discards the triangle tri_in against the plane defined by (plane_p, plane_n). plane_n is normalized inside the function.

Parameters

tri_in	Input triangle.
plane_p	Plane reference point (3x1).
plane_n	Plane normal (3x1).
tri_out1	First output triangle (if any).
tri_out2	Second output triangle (if split).

Returns

int Number of output triangles: 0 (culled), 1, or 2. Returns -1 on error.

Definition at line 1838 of file Almog_Engine.h.

References ae_assert_tri_is_valid, ae_line_itersect_plane(), ae_point_to_mat2D(), ae_signed_dist_point_and_plane(), Tri::colors, mat2D_alloc(), mat2D_free(), mat2D_normalize, Tri::points, Tri::tex_points, Point::w, Point::x, and Point::y.

Referenced by ae_tri_mesh_project_world2screen(), and ae_tri_project_world2screen().

4.7.4.37 ae_tri_compare()

```
bool ae_tri_compare (
            Tri t1,
            Tri t2 )
```

Compare two triangles for sorting by depth.

Returns true if t1 should come before t2 when sorting by the maximum z of their vertices (descending order).

Parameters

t1	First triangle.
t2	Second triangle.

Returns

bool true if t1 precedes t2, false otherwise.

Definition at line 3725 of file Almog_Engine.h.

References Tri::points, and Point::z.

Referenced by ae_tri_qsort().

4.7.4.38 ae_tri_create()

```
Tri ae_tri_create (
              Point p1,
              Point p2,
              Point p3 )
```

Create a triangle from three points.

Parameters

p1	First vertex (world space).
p2	Second vertex (world space).
p3	Third vertex (world space).

Returns

Tri The created triangle with vertices set. Other fields are left uninitialized.

Definition at line 278 of file Almog_Engine.h.

References Tri::points.

4.7.4.39 ae_tri_get_average_normal()

```
Point ae_tri_get_average_normal (
              Tri tri )
```

Compute the average of the three vertex normals of a triangle.

Averages the three vertex normals and normalizes the result.

Parameters

tri	Input triangle.

Returns

Point The averaged, normalized normal (w averaged but unused).

Definition at line 1041 of file Almog_Engine.h.

References ae_point_normalize_xyz(), Tri::normals, Point::w, Point::x, Point::y, and Point::z.

Referenced by ae_tri_calc_light_intensity().

4.7.4.40 ae_tri_get_average_point()

```
Point ae_tri_get_average_point (
            Tri tri )
```

Compute the average of the three vertices of a triangle.

Parameters

tri	Input triangle.

Returns

Point The average point (x, y, z, w are simple averages).

Definition at line 1062 of file Almog_Engine.h.

References Tri::points, Point::w, Point::x, Point::y, and Point::z.

Referenced by ae_tri_calc_light_intensity().

4.7.4.41 ae_tri_mesh_append_copy()

```
void ae_tri_mesh_append_copy (
            Tri_mesh_array * mesh_array,
            Tri_mesh mesh )
```

Append a copy of a Tri_mesh into a Tri_mesh_array.

Creates a deep copy of mesh (triangles by value) and appends it to mesh_array (ADA array of meshes).

Parameters

mesh_array	Destination mesh array to append into.
mesh	Source triangle mesh to copy.

Definition at line 851 of file Almog_Engine.h.

References ada_append, ada_init_array, Tri_mesh::elements, and Tri_mesh::length.

Referenced by setup().

4.7.4.42 ae_tri_mesh_create_copy()

```
void ae_tri_mesh_create_copy (
            Tri_mesh * des,
            Tri * src_elements,
            size_t len )
```

Append copies of triangles to a destination Tri_mesh (resets destination length first).

Appends len triangles from src_elements into the destination ADA array pointed by des.

Parameters

des	Destination triangle mesh (ADA array). Will grow as needed.
src_elements	Source array of triangles to copy from.
len	Number of triangles to copy from src_elements.

Definition at line 299 of file Almog_Engine.h.

References ada_append, and Tri_mesh::length.

4.7.4.43 ae_tri_mesh_flip_normals()

```
void ae_tri_mesh_flip_normals (
            Tri_mesh mesh )
```

Flip triangle winding and recompute per-vertex normals.

Swaps vertex order to invert winding, copies attributes accordingly, and recomputes normals.

Parameters

mesh	Mesh to flip (modified in place).

Definition at line 1283 of file Almog_Engine.h.

References ae_tri_set_normals(), Tri::colors, Tri_mesh::elements, Tri_mesh::length, Tri::light_intensity, Tri::normals, Tri::points, Tri::tex_points, and Tri::to_draw.

4.7.4.44 ae_tri_mesh_get_from_file()

```
Tri_mesh ae_tri_mesh_get_from_file (
            char * file_path )
```

Load a triangle mesh from a file (OBJ or STL).

Dispatches to ae_tri_mesh_get_from_obj_file or ae_tri_mesh_get_from_stl_file based on file extension.

Parameters

file_path	Path to the file (.obj, .stl, .STL).

Returns

Tri_mesh The loaded triangle mesh. Caller must free mesh.elements when done.

Definition at line 816 of file Almog_Engine.h.

References ae_tri_mesh_get_from_obj_file(), ae_tri_mesh_get_from_stl_file(), asm_get_word_and_cut(), ASM_MAX_LEN_LINE, and asm_str_in_str().

Referenced by setup().

4.7.4.45 ae_tri_mesh_get_from_obj_file()

```
Tri_mesh ae_tri_mesh_get_from_obj_file (
            char * file_path )
```

Load a triangle mesh from a Wavefront OBJ file.

Supports vertex positions (v). Face lines (f) with 3 or 4 vertices are parsed. Texture coordinates and normals in the file are ignored (a warning is printed once if present). Quads are triangulated as (0,1,2) and (2,3,0). Colors are set to white and to_draw is set to true.

Parameters

file_path	Path to the OBJ file.

Returns

Tri_mesh The loaded triangle mesh. Caller must free mesh.elements when done.

Definition at line 540 of file Almog_Engine.h.

References ada_append, ada_init_array, asm_get_line(), asm_get_next_word_from_line(), asm_get_word_and_cut(), asm_length(), ASM_MAX_LEN_LINE, asm_str_in_str(), Tri::colors, Curve::elements, Tri::light_intensity, Tri::points, Tri::to_draw, Point::x, Point::y, and Point::z.

Referenced by ae_tri_mesh_get_from_file().

4.7.4.46 ae_tri_mesh_get_from_quad_mesh()

```
Tri_mesh ae_tri_mesh_get_from_quad_mesh (
            Quad_mesh q_mesh )
```

Convert a Quad_mesh into a Tri_mesh.

Splits each quad into two triangles: (0,1,2) and (2,3,0), copying per-vertex attributes (points, colors, normals, light intensities).

Parameters

q_mesh	Input quad mesh.

Returns

Tri_mesh Resulting triangle mesh. Caller must free mesh.elements when done.

Definition at line 875 of file Almog_Engine.h.

References ada_append, ada_init_array, Tri::colors, Quad::colors, Quad_mesh::elements, Quad_mesh::length, Tri::light_intensity, Quad::light_intensity, Tri::normals, Quad::normals, Tri::points, Quad::points, Tri::to_draw, and Quad::to_draw.

4.7.4.47 ae_tri_mesh_get_from_stl_file()

```
Tri_mesh ae_tri_mesh_get_from_stl_file (
            char * file_path )
```

Load a triangle mesh from a binary STL file.

Reads binary STL (little-endian). Per-triangle normals from the file are negated to match the engine's convention and copied to each vertex normal. Colors are set to white and to_draw is set to true.

Parameters

file_path	Path to the binary STL file.

Returns

Tri_mesh The loaded triangle mesh. Caller must free mesh.elements when done.

Definition at line 743 of file Almog_Engine.h.

References ada_append, ada_init_array, Tri::colors, Tri::light_intensity, Tri::normals, Tri::points, STL_ATTRIBUTE_BITS_SIZE, STL_HEADER_SIZE, STL_NUM_SIZE, Tri::to_draw, Point::x, Point::y, and Point::z.

Referenced by ae_tri_mesh_get_from_file().

4.7.4.48 ae_tri_mesh_normalize()

```
void ae_tri_mesh_normalize (
            Tri_mesh mesh )
```

Normalize mesh coordinates to [-1, 1], centered at origin.

Uniformly scales and recenters the mesh so that the largest axis fits exactly into [-1, 1]. Other axes are scaled proportionally. Updates all vertices in place.

Parameters

mesh	Triangle mesh to normalize (modified in place).

Definition at line 1244 of file Almog_Engine.h.

References ae_tri_mesh_set_bounding_box(), Tri_mesh::elements, Tri_mesh::length, Tri::points, Point::x, Point::y, and Point::z.

Referenced by setup().

4.7.4.49 ae_tri_mesh_project_world2screen()

```
void ae_tri_mesh_project_world2screen (
            Mat2D proj_mat,
            Mat2D view_mat,
            Tri_mesh * des,
            Tri_mesh src,
            int window_w,
            int window_h,
            Scene * scene,
            Lighting_mode lighting_mode )
```

Project a triangle mesh from world to screen space with clipping.

Iterates over all triangles, applies near-plane and screen-edge clipping (top/right/bottom/left), and writes results into des. Triangles can be split by clipping, so des may end up with more elements than src.

Parameters

proj_mat	Projection matrix (4x4).
view_mat	View matrix (4x4).
des	Output mesh (cleared and filled; ADA array grown as needed).
src	Input world-space triangle mesh.
window_w	Screen width in pixels.
window_h	Screen height in pixels.
scene	Scene (camera/light/material).
lighting_mode	Flat or smooth lighting mode.

Definition at line 3151 of file Almog_Engine.h.

References ada_append, ada_insert_unordered, ada_remove_unordered, ae_assert_tri_is_valid, ae_tri_clip_with_plane(), ae_tri_project_world2screen(), Tri_mesh::elements, Tri_mesh::length, mat2D_alloc(), MAT2D_AT, mat2D_fill(), and mat2D_free().

Referenced by update().

4.7.4.50 ae_tri_mesh_rotate_Euler_xyz()

```
void ae_tri_mesh_rotate_Euler_xyz (
            Tri_mesh mesh,
            float phi_deg,
            float theta_deg,
            float psi_deg )
```

Rotate a triangle mesh using XYZ Euler angles (degrees).

Applies DCM = Cz(psi_deg) ∗ Cy(theta_deg) ∗ Cx(phi_deg) to each vertex. Recomputes per-vertex normals afterward.

Parameters

mesh	Triangle mesh to rotate (modified in place).
phi_deg	Rotation about X axis, degrees.
theta_deg	Rotation about Y axis, degrees.
psi_deg	Rotation about Z axis, degrees.

Definition at line 1143 of file Almog_Engine.h.

References ae_tri_mesh_set_normals(), Tri_mesh::elements, Tri_mesh::length, mat2D_alloc(), MAT2D_AT, mat2D_dot(), mat2D_free(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), mat2D_set_rot_mat_z(), Tri::points, Point::x, Point::y, and Point::z.

Referenced by setup().

4.7.4.51 ae_tri_mesh_set_bounding_box()

```
void ae_tri_mesh_set_bounding_box (
            Tri_mesh mesh,
            float * x_min,
            float * x_max,
            float * y_min,
            float * y_max,
            float * z_min,
            float * z_max )
```

Compute the axis-aligned bounding box of a triangle mesh.

Writes min/max for x, y, z across all vertices in the mesh.

Parameters

mesh	Input triangle mesh.
x_min	Output minimum x.
x_max	Output maximum x.
y_min	Output minimum y.
y_max	Output maximum y.
z_min	Output minimum z.
z_max	Output maximum z.

Definition at line 1206 of file Almog_Engine.h.

References Tri_mesh::elements, Tri_mesh::length, Tri::points, Point::x, Point::y, and Point::z.

Referenced by ae_tri_mesh_normalize().

4.7.4.52 ae_tri_mesh_set_normals()

```
void ae_tri_mesh_set_normals (
            Tri_mesh mesh )
```

Recompute per-vertex normals for all triangles in a mesh.

Calls ae_tri_set_normals on each triangle.

Parameters

mesh	Mesh to update (modified in place).

Definition at line 1321 of file Almog_Engine.h.

References ae_tri_set_normals(), Tri_mesh::elements, and Tri_mesh::length.

Referenced by ae_tri_mesh_rotate_Euler_xyz().

4.7.4.53 ae_tri_mesh_translate()

```
void ae_tri_mesh_translate (
            Tri_mesh mesh,
            float x,
            float y,
            float z )
```

Translate a triangle mesh by (x, y, z).

Adds the given offsets to each vertex in the mesh.

Parameters

mesh	Triangle mesh to translate (modified in place).
x	X-axis offset.
y	Y-axis offset.
z	Z-axis offset.

Definition at line 1121 of file Almog_Engine.h.

References Tri_mesh::elements, Tri_mesh::length, Tri::points, Point::x, Point::y, and Point::z.

4.7.4.54 ae_tri_project_world2screen()

```
Tri_mesh ae_tri_project_world2screen (
            Mat2D proj_mat,
            Mat2D view_mat,
            Tri tri,
            int window_w,
            int window_h,
            Scene * scene,
            Lighting_mode lighting_mode )
```

Project a single world-space triangle to screen space with clipping.

Computes lighting, back-face visibility, transforms to view space, clips against near plane, and projects to screen space. If clipping splits the triangle, multiple triangles may be returned.

Parameters

proj_mat	Projection matrix (4x4).
view_mat	View matrix (4x4).
tri	World-space triangle.
window_w	Screen width in pixels.
window_h	Screen height in pixels.
scene	Scene (camera for near plane, light/material for lighting).
lighting_mode	Flat or smooth lighting mode.

Returns

Tri_mesh An ADA array of resulting screen-space triangles. Caller must free result.elements.

Definition at line 3038 of file Almog_Engine.h.

References ada_append, ada_init_array, ae_assert_tri_is_valid, ae_point_project_view2screen(), ae_point_to_mat2D(), ae_tri_calc_light_intensity(), ae_tri_calc_normal(), ae_tri_clip_with_plane(), ae_tri_transform_to_view(), Scene::camera, Camera::current_position, Tri_mesh::elements, Tri_mesh::length, Tri::light_intensity, mat2D_alloc(), MAT2D_AT, mat2D_fill(), mat2D_free(), mat2D_sub(), mat2D_transpose(), Tri::normals, Tri::points, Tri::tex_points, Tri::to_draw, Point::w, Point::x, Point::y, Point::z, and Camera::z_near.

Referenced by ae_tri_mesh_project_world2screen().

4.7.4.55 ae_tri_qsort()

```
void ae_tri_qsort (
            Tri * v,
            int left,
            int right )
```

Quicksort an array of triangles by depth.

Sorts v[left..right] using ae_tri_compare (descending by max z).

Parameters

v	Array of triangles to sort.
left	Left index (inclusive).
right	Right index (inclusive).

Definition at line 3742 of file Almog_Engine.h.

References ae_tri_compare(), and ae_tri_swap().

4.7.4.56 ae_tri_set_normals()

```
void ae_tri_set_normals (
            Tri * tri )
```

Compute and set per-vertex normals for a triangle.

For each vertex, computes the cross product of the adjacent edges around that vertex and normalizes it. Results are stored in tri->normals[i].

Parameters

tri	Triangle whose normals will be computed and written.

Definition at line 998 of file Almog_Engine.h.

References ae_assert_tri_is_valid, ae_mat2D_to_point(), ae_point_to_mat2D(), mat2D_alloc(), mat2D_copy(), mat2D_cross(), mat2D_free(), mat2D_normalize, mat2D_sub(), Tri::normals, and Tri::points.

Referenced by ae_tri_mesh_flip_normals(), and ae_tri_mesh_set_normals().

4.7.4.57 ae_tri_swap()

```
void ae_tri_swap (
            Tri * v,
            int i,
            int j )
```

Swap two triangles in an array.

Parameters

v	Array of triangles.
i	Index of first element.
j	Index of second element.

Definition at line 3706 of file Almog_Engine.h.

Referenced by ae_tri_qsort().

4.7.4.58 ae_tri_transform_to_view()

```
Tri ae_tri_transform_to_view (
            Mat2D view_mat,
            Tri tri )
```

Transform a triangle from world space to view space.

Applies view_mat to each vertex (homogeneous multiply with w=1). Returns the transformed triangle; normals are not changed.

Parameters

view_mat	View matrix (4x4).
tri	World-space triangle.

Returns

Tri View-space triangle.

Definition at line 2988 of file Almog_Engine.h.

References AE_ASSERT, ae_assert_tri_is_valid, mat2D_alloc(), MAT2D_AT, mat2D_dot(), mat2D_free(), Tri::points, Point::w, Point::x, Point::y, and Point::z.

Referenced by ae_tri_project_world2screen().

4.7.4.59 ae_view_mat_set()

```
void ae_view_mat_set (
            Mat2D view_mat,
            Camera camera,
            Mat2D up )
```

Build a right-handed view matrix from a Camera and up vector.

Computes camera basis (right, up, forward) from yaw/pitch/roll offsets and direction, applies offset_position along those axes to update current_position, then zeroes offset_position. Writes the resulting 4x4 view matrix.

Note

Although camera is passed by value, its Mat2D members (e.g. current_position, offset_position, camera_↩ x/y/z) are modified in place due to internal pointer semantics of Mat2D.

Parameters

view_mat	Output 4x4 view matrix.
camera	Camera state (basis vectors and positions updated).
up	World up direction (3x1).

Definition at line 2716 of file Almog_Engine.h.

References Camera::camera_x, Camera::camera_y, Camera::camera_z, Camera::current_position, Camera::direction, mat2D_add(), mat2D_alloc(), MAT2D_AT, mat2D_calc_norma(), mat2D_copy(), mat2D_cross(), mat2D_dot(), mat2D_dot_product(), mat2D_fill(), mat2D_free(), mat2D_mult(), mat2D_set_DCM_zyx(), mat2D_sub(), mat2D_transpose(), Camera::offset_position, Camera::pitch_offset_deg, Camera::roll_offset_deg, and Camera::yaw_offset_deg.

Referenced by ae_scene_init(), and update().

4.7.4.60 ae_z_buffer_copy_to_screen()

```
void ae_z_buffer_copy_to_screen (
            Mat2D_uint32 screen_mat,
            Mat2D inv_z_buffer )
```

Visualize an inverse-z buffer by writing a grayscale image.

Finds the min positive and max inverse-z in inv_z_buffer, maps the range to [0.1, 1.0], and writes an RGB grayscale value into screen_mat at each pixel. Values <= 0 are clamped to the minimum positive.

Parameters

screen_mat	Output RGB image (Mat2D_uint32) 0xRRGGBB per pixel.
inv_z_buffer	Input inverse-z values (Mat2D of doubles).

Definition at line 3785 of file Almog_Engine.h.

References ae_linear_map(), Mat2D::cols, MAT2D_AT, MAT2D_AT_UINT32, RGB_hexRGB, and Mat2D::rows.

4.8 Almog_Engine.h

```
00001
00030 #ifndef ALMOG_ENGINE_H_
00031 #define ALMOG_ENGINE_H_
00032
00033 #include "./Almog_Dynamic_Array.h"
00034 #include "./Matrix2D.h"
00035 #include "./Almog_Draw_Library.h"
00036
00037 #ifndef ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00038 #define ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00039 #endif
00040 #include "./Almog_String_Manipulation.h"
00041
00042 #ifndef AE_ASSERT
00043 #include <assert.h>
00044 #define AE_ASSERT assert
00045 #endif
00046
00047 #include <math.h>
00048 #include <stdbool.h>
00049 #include <float.h>
00050 #include <stdint.h>
00051 #include <errno.h>
00052 #include <string.h>
00053
00054 #ifndef PI
00055 #define PI M_PI
00056 #endif
00057
00058 #ifndef STL_HEADER_SIZE
00059 #define STL_HEADER_SIZE 80
00060 #endif
00061
00062 #ifndef STL_NUM_SIZE
00063 #define STL_NUM_SIZE 4
00064 #endif
00065
00066 #ifndef STL_SIZE_FOREACH_TRI
00067 #define STL_SIZE_FOREACH_TRI 50
00068 #endif
00069
00070 #ifndef STL_ATTRIBUTE_BITS_SIZE
00071 #define STL_ATTRIBUTE_BITS_SIZE 2
00072 #endif
00073
00074 #ifndef HexARGB_RGBA
00075 #define HexARGB_RGBA(x) ((x)»(8*2)&0xFF), ((x)»(8*1)&0xFF), ((x)»(8*0)&0xFF), ((x)»(8*3)&0xFF)
00076 #endif
00077 #ifndef HexARGB_RGB_VAR
00078 #define HexARGB_RGB_VAR(x, r, g, b) r = ((x)»(8*2)&0xFF); g = ((x)»(8*1)&0xFF); b = ((x)»(8*0)&0xFF);
00079 #endif
00080 #ifndef HexARGB_RGBA_VAR
00081 #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)
00082 #endif
00083 #define ARGB_hexARGB(a, r, g, b) 0x010000001*(uint8_t)(a) + 0x00010000*(uint8_t)(r) +
       0x00000100*(uint8_t)(g) + 0x00000001*(uint8_t)(b)
00084 #ifndef RGB_hexRGB
00085 #define RGB_hexRGB(r, g, b) (int)(0x010000*(int)(r) + 0x000100*(int)(g) + 0x000001*(int)(b))
00086 #endif
00087
00088 #define AE_MAX_POINT_VAL 1e5
00089 #define ae_assert_point_is_valid(p) AE_ASSERT(isfinite((p).x) && isfinite((p).y) && isfinite((p).z) &&
       isfinite((p).w));    \
00090         AE_ASSERT((p).x > -AE_MAX_POINT_VAL && (p).x < AE_MAX_POINT_VAL);
                                                \
```

```
00091         AE_ASSERT((p).y > -AE_MAX_POINT_VAL && (p).y < AE_MAX_POINT_VAL);
                                    \
00092         AE_ASSERT((p).z > -AE_MAX_POINT_VAL && (p).z < AE_MAX_POINT_VAL);
                                    \
00093         AE_ASSERT((p).w > -AE_MAX_POINT_VAL && (p).w < AE_MAX_POINT_VAL);
00094 #define ae_assert_tri_is_valid(tri) ae_assert_point_is_valid((tri).points[0]);  \
00095         ae_assert_point_is_valid((tri).points[1]);                                \
00096         ae_assert_point_is_valid((tri).points[2])
00097 #define ae_assert_quad_is_valid(quad) ae_assert_point_is_valid((quad).points[0]);   \
00098         ae_assert_point_is_valid((quad).points[1]);                                 \
00099         ae_assert_point_is_valid((quad).points[2]);                                 \
00100         ae_assert_point_is_valid((quad).points[3])
00101 #define ae_point_normalize_xyz_norma(p, norma) (p).x = (p).x / norma;  \
00102         (p).y = (p).y / norma;                          \
00103         (p).z = (p).z / norma
00104 #define ae_point_calc_norma(p) sqrt(((p).x * (p).x) + ((p).y * (p).y) + ((p).z * (p).z))
00105 #define ae_point_add_point(p, p1, p2) (p).x = (p1).x + (p2).x;  \
00106         (p).y = (p1).y + (p2).y;                                  \
00107         (p).z = (p1).z + (p2).z;                                  \
00108         (p).w = (p1).w + (p2).w
00109 #define ae_point_sub_point(p, p1, p2) (p).x = (p1).x - (p2).x;  \
00110         (p).y = (p1).y - (p2).y;                                  \
00111         (p).z = (p1).z - (p2).z;                                  \
00112         (p).w = (p1).w - (p2).w
00113 #define ae_point_dot_point(p1, p2) (((p1).x * (p2).x) + ((p1).y * (p2).y) + ((p1).z * (p2).z))
00114 #define ae_point_mult(p, const) (p).x *= const; \
00115         (p).y *= const;                           \
00116         (p).z *= const
00117 #define ae_points_equal(p1, p2) (p1).x == (p2).x && (p1).y == (p2).y && (p1).z == (p2).z
00118
00119
00120 typedef enum {
00121     AE_LIGHTING_FLAT,
00122     AE_LIGHTING_SMOOTH,
00123     AE_LIGHTING_MODE_LENGTH
00124 } Lighting_mode;
00125
00126 #ifndef TRI_MESH_ARRAY
00127 #define TRI_MESH_ARRAY
00128 typedef struct {
00129     size_t length;
00130     size_t capacity;
00131     Tri_mesh *elements;
00132 } Tri_mesh_array; /* Tri_mesh ada array */
00133 #endif
00134
00135 #ifndef QUAD_MESH_ARRAY
00136 #define QUAD_MESH_ARRAY
00137 typedef struct {
00138     size_t length;
00139     size_t capacity;
00140     Quad_mesh *elements;
00141 } Quad_mesh_array; /* Quad_mesh ada array */
00142 #endif
00143
00144 typedef struct {
00145     Mat2D init_position;
00146     Mat2D current_position;
00147     Mat2D offset_position;
00148     Mat2D direction;
00149     float z_near;
00150     float z_far;
00151     float fov_deg;
00152     float aspect_ratio;
00153     float roll_offset_deg;
00154     float pitch_offset_deg;
00155     float yaw_offset_deg;
00156     Mat2D camera_x;
00157     Mat2D camera_y;
00158     Mat2D camera_z;
00159 } Camera;
00160
00161 typedef struct {
00162     Point light_direction_or_pos;
00163     float light_intensity;
00164 } Light_source;
00165
00166 typedef struct {
00167     float specular_power_alpha;
00168     float c_ambi;
00169     float c_diff;
00170     float c_spec;
00171 } Material;
00172
00173 typedef struct {
00174     Tri_mesh_array in_world_tri_meshes;
00175     Tri_mesh_array projected_tri_meshes;
```

```
00176     Tri_mesh_array original_tri_meshes;
00177
00178     Quad_mesh_array in_world_quad_meshes;
00179     Quad_mesh_array projected_quad_meshes;
00180     Quad_mesh_array original_quad_meshes;
00181
00182     Camera camera;
00183     Mat2D up_direction;
00184     Mat2D proj_mat;
00185     Mat2D view_mat;
00186
00187     Light_source light_source0;
00188     Material material0;
00189 } Scene;
00190
00191 Tri         ae_tri_create(Point p1, Point p2, Point p3);
00192 void        ae_tri_mesh_create_copy(Tri_mesh *des, Tri *src_elements, size_t len);
00193
00194 void        ae_camera_init(Scene *scene, int window_h, int window_w);
00195 void        ae_camera_free(Scene *scene);
00196 Scene       ae_scene_init(int window_h, int window_w);
00197 void        ae_scene_free(Scene *scene);
00198 void        ae_camera_reset_pos(Scene *scene);
00199
00200 void        ae_point_to_mat2D(Point p, Mat2D m);
00201 Point       ae_mat2D_to_point(Mat2D m);
00202
00203 Tri_mesh    ae_tri_mesh_get_from_obj_file(char *file_path);
00204 Tri_mesh    ae_tri_mesh_get_from_stl_file(char *file_path);
00205 Tri_mesh    ae_tri_mesh_get_from_file(char *file_path);
00206 void        ae_tri_mesh_append_copy(Tri_mesh_array *mesh_array, Tri_mesh mesh);
00207 Tri_mesh    ae_tri_mesh_get_from_quad_mesh(Quad_mesh q_mesh);
00208
00209 void        ae_print_points(Curve p);
00210 void        ae_print_tri(Tri tri, char *name, size_t padding);
00211 void        ae_print_tri_mesh(Tri_mesh mesh, char *name, size_t padding);
00212
00213 Point       ae_point_normalize_xyz(Point p);
00214 void        ae_tri_set_normals(Tri *tri);
00215 Point       ae_tri_get_average_normal(Tri tri);
00216 Point       ae_tri_get_average_point(Tri tri);
00217 void        ae_tri_calc_normal(Mat2D normal, Tri tri);
00218 void        ae_tri_mesh_translate(Tri_mesh mesh, float x, float y, float z);
00219 void        ae_tri_mesh_rotate_Euler_xyz(Tri_mesh mesh, float phi_deg, float theta_deg, float
       psi_deg);
00220 void        ae_tri_mesh_set_bounding_box(Tri_mesh mesh, float *x_min, float *x_max, float *y_min,
       float *y_max, float *z_min, float *z_max);
00221 void        ae_tri_mesh_normalize(Tri_mesh mesh);
00222 void        ae_tri_mesh_flip_normals(Tri_mesh mesh);
00223 void        ae_tri_mesh_set_normals(Tri_mesh mesh);
00224 void        ae_quad_set_normals(Quad *quad);
00225 Point       ae_quad_get_average_normal(Quad quad);
00226 Point       ae_quad_get_average_point(Quad quad);
00227 void        ae_quad_calc_normal(Mat2D normal, Quad quad);
00228 void        ae_curve_copy(Curve *des, Curve src);
00229
00230 void        ae_tri_calc_light_intensity(Tri *tri, Scene *scene, Lighting_mode lighting_mode);
00231 void        ae_quad_calc_light_intensity(Quad *quad, Scene *scene, Lighting_mode lighting_mode);
00232
00233 Point       ae_line_itersect_plane(Mat2D plane_p, Mat2D plane_n, Mat2D line_start, Mat2D line_end,
       float *t);
00234 int         ae_line_clip_with_plane(Point start_in, Point end_in, Mat2D plane_p, Mat2D plane_n, Point
       *start_out, Point *end_out);
00235 float       ae_signed_dist_point_and_plane(Point p, Mat2D plane_p, Mat2D plane_n);
00236 int         ae_tri_clip_with_plane(Tri tri_in, Mat2D plane_p, Mat2D plane_n, Tri *tri_out1, Tri
       *tri_out2);
00237 int         ae_quad_clip_with_plane(Quad quad_in, Mat2D plane_p, Mat2D plane_n, Quad *quad_out1, Quad
       *quad_out2);
00238
00239 void        ae_projection_mat_set(Mat2D proj_mat,float aspect_ratio, float FOV_deg, float z_near,
       float z_far);
00240 void        ae_view_mat_set(Mat2D view_mat, Camera camera, Mat2D up);
00241 Point       ae_point_project_world2screen(Mat2D view_mat, Mat2D proj_mat, Point src, int window_w, int
       window_h);
00242 Point       ae_point_project_world2view(Mat2D view_mat, Point src);
00243 Point       ae_point_project_view2screen(Mat2D proj_mat, Point src, int window_w, int window_h);
00244 void        ae_line_project_world2screen(Mat2D view_mat, Mat2D proj_mat, Point start_src, Point
       end_src, int window_w, int window_h, Point *start_des, Point *end_des, Scene *scene);
00245 Tri         ae_tri_transform_to_view(Mat2D view_mat, Tri tri);
00246 Tri_mesh    ae_tri_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Tri tri, int window_w, int
       window_h, Scene *scene, Lighting_mode lighting_mode);
00247 void        ae_tri_mesh_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Tri_mesh *des, Tri_mesh
       src, int window_w, int window_h, Scene *scene, Lighting_mode lighting_mode);
00248 Quad        ae_quad_transform_to_view(Mat2D view_mat, Quad quad);
00249 Quad_mesh   ae_quad_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Quad quad, int window_w, int
       window_h, Scene *scene, Lighting_mode lighting_mode);
00250 void        ae_quad_mesh_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Quad_mesh *des,
```

```
      Quad_mesh src, int window_w, int window_h, Scene *scene, Lighting_mode lighting_mode);
00251 void        ae_curve_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Curve *des, Curve src, int
      window_w, int window_h, Scene *scene);
00252 void        ae_curve_ada_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Curve_ada *des,
      Curve_ada src, int window_w, int window_h, Scene *scene);
00253 void        ae_grid_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Grid des, Grid src, int
      window_w, int window_h, Scene *scene);
00254
00255 void        ae_tri_swap(Tri *v, int i, int j);
00256 bool        ae_tri_compare(Tri t1, Tri t2);
00257 void        ae_tri_qsort(Tri *v, int left, int right);
00258 double      ae_linear_map(double s, double min_in, double max_in, double min_out, double max_out);
00259 void        ae_z_buffer_copy_to_screen(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer);
00260
00261 #endif /* ALMOG_ENGINE_H_ */
00262
00263 #ifdef ALMOG_ENGINE_IMPLEMENTATION
00264 #undef ALMOG_ENGINE_IMPLEMENTATION
00265
00266 #define AE_PRINT_TRI(tri) ae_print_tri(tri, #tri, 0)
00267 #define AE_PRINT_MESH(mesh) ae_print_tri_mesh(mesh, #mesh, 0)
00268
00278 Tri ae_tri_create(Point p1, Point p2, Point p3)
00279 {
00280     Tri tri = {0};
00281
00282     tri.points[0] = p1;
00283     tri.points[1] = p2;
00284     tri.points[2] = p3;
00285
00286     return tri;
00287 }
00288
00299 void ae_tri_mesh_create_copy(Tri_mesh *des, Tri *src_elements, size_t len)
00300 {
00301     Tri_mesh temp_des = *des;
00302     temp_des.length = 0;
00303     for (size_t i = 0; i < len; i++) {
00304         ada_append(Tri, temp_des, src_elements[i]);
00305     }
00306     *des = temp_des;
00307 }
00308
00320 void ae_camera_init(Scene *scene, int window_h, int window_w)
00321 {
00322     scene->camera.z_near       = 0.1;
00323     scene->camera.z_far        = 1000;
00324     scene->camera.fov_deg      = 60;
00325     scene->camera.aspect_ratio = (float)window_h / (float)window_w;
00326
00327     scene->camera.init_position = mat2D_alloc(3, 1);
00328     mat2D_fill(scene->camera.init_position, 0);
00329     MAT2D_AT(scene->camera.init_position, 2, 0) = -4;
00330
00331     scene->camera.current_position = mat2D_alloc(3, 1);
00332     mat2D_copy(scene->camera.current_position, scene->camera.init_position);
00333
00334     scene->camera.offset_position = mat2D_alloc(3, 1);
00335     mat2D_fill(scene->camera.offset_position, 0);
00336
00337     scene->camera.roll_offset_deg = 0;
00338     scene->camera.pitch_offset_deg = 0;
00339     scene->camera.yaw_offset_deg = 0;
00340
00341     scene->camera.direction = mat2D_alloc(3, 1);
00342     mat2D_fill(scene->camera.direction, 0);
00343     MAT2D_AT(scene->camera.direction, 2, 0) = 1;
00344
00345     scene->camera.camera_x = mat2D_alloc(3, 1);
00346     mat2D_fill(scene->camera.camera_x, 0);
00347     MAT2D_AT(scene->camera.camera_x, 0, 0) = 1;
00348
00349     scene->camera.camera_y = mat2D_alloc(3, 1);
00350     mat2D_fill(scene->camera.camera_y, 0);
00351     MAT2D_AT(scene->camera.camera_y, 1, 0) = 1;
00352
00353     scene->camera.camera_z = mat2D_alloc(3, 1);
00354     mat2D_fill(scene->camera.camera_z, 0);
00355     MAT2D_AT(scene->camera.camera_z, 2, 0) = 1;
00356 }
00357
00366 void ae_camera_free(Scene *scene)
00367 {
00368     mat2D_free(scene->camera.init_position);
00369     mat2D_free(scene->camera.current_position);
00370     mat2D_free(scene->camera.offset_position);
00371     mat2D_free(scene->camera.direction);
```

```
00372     mat2D_free(scene->camera.camera_x);
00373     mat2D_free(scene->camera.camera_y);
00374     mat2D_free(scene->camera.camera_z);
00375 }
00376
00388 Scene ae_scene_init(int window_h, int window_w)
00389 {
00390     Scene scene = {0};
00391     ae_camera_init(&(scene), window_h, window_w);
00392
00393     scene.up_direction = mat2D_alloc(3, 1);
00394     mat2D_fill(scene.up_direction, 0);
00395     MAT2D_AT(scene.up_direction, 1, 0) = 1;
00396
00397     scene.light_source0.light_direction_or_pos.x = 0.5;
00398     scene.light_source0.light_direction_or_pos.y = 1;
00399     scene.light_source0.light_direction_or_pos.z = 1;
00400     scene.light_source0.light_direction_or_pos.w = 0;
00401     scene.light_source0.light_direction_or_pos =
      ae_point_normalize_xyz(scene.light_source0.light_direction_or_pos);
00402     scene.light_source0.light_intensity = 1;
00403
00404     scene.material0.specular_power_alpha = 1;
00405     scene.material0.c_ambi = 0.2;
00406     scene.material0.c_diff = 0.6;
00407     scene.material0.c_spec = 0.2;
00408
00409     scene.proj_mat = mat2D_alloc(4, 4);
00410     ae_projection_mat_set(scene.proj_mat, scene.camera.aspect_ratio, scene.camera.fov_deg,
      scene.camera.z_near, scene.camera.z_far);
00411
00412     scene.view_mat = mat2D_alloc(4, 4);
00413     ae_view_mat_set(scene.view_mat, scene.camera, scene.up_direction);
00414
00415     return scene;
00416 }
00417
00429 void ae_scene_free(Scene *scene)
00430 {
00431     ae_camera_free(scene);
00432     mat2D_free(scene->up_direction);
00433     mat2D_free(scene->proj_mat);
00434     mat2D_free(scene->view_mat);
00435
00436     for (size_t i = 0; i < scene->in_world_tri_meshes.length; i++) {
00437         free(scene->in_world_tri_meshes.elements[i].elements);
00438     }
00439     for (size_t i = 0; i < scene->projected_tri_meshes.length; i++) {
00440         free(scene->projected_tri_meshes.elements[i].elements);
00441     }
00442     for (size_t i = 0; i < scene->original_tri_meshes.length; i++) {
00443         free(scene->original_tri_meshes.elements[i].elements);
00444     }
00445     if (scene->in_world_tri_meshes.elements) free(scene->in_world_tri_meshes.elements);
00446     if (scene->projected_tri_meshes.elements) free(scene->projected_tri_meshes.elements);
00447     if (scene->original_tri_meshes.elements) free(scene->original_tri_meshes.elements);
00448
00449     for (size_t i = 0; i < scene->in_world_quad_meshes.length; i++) {
00450         free(scene->in_world_quad_meshes.elements[i].elements);
00451     }
00452     for (size_t i = 0; i < scene->projected_quad_meshes.length; i++) {
00453         free(scene->projected_quad_meshes.elements[i].elements);
00454     }
00455     for (size_t i = 0; i < scene->original_quad_meshes.length; i++) {
00456         free(scene->original_quad_meshes.elements[i].elements);
00457     }
00458     if (scene->in_world_quad_meshes.elements) free(scene->in_world_quad_meshes.elements);
00459     if (scene->projected_quad_meshes.elements) free(scene->projected_quad_meshes.elements);
00460     if (scene->original_quad_meshes.elements) free(scene->original_quad_meshes.elements);
00461 }
00462
00472 void ae_camera_reset_pos(Scene *scene)
00473 {
00474     scene->camera.roll_offset_deg = 0;
00475     scene->camera.pitch_offset_deg = 0;
00476     scene->camera.yaw_offset_deg = 0;
00477
00478     mat2D_fill(scene->camera.offset_position, 0);
00479
00480     mat2D_fill(scene->camera.camera_x, 0);
00481     MAT2D_AT(scene->camera.camera_x, 0, 0) = 1;
00482     mat2D_fill(scene->camera.camera_y, 0);
00483     MAT2D_AT(scene->camera.camera_y, 1, 0) = 1;
00484     mat2D_fill(scene->camera.camera_z, 0);
00485     MAT2D_AT(scene->camera.camera_z, 2, 0) = 1;
00486
00487     mat2D_copy(scene->camera.current_position, scene->camera.init_position);
```

```
00488 }
00489
00498 void ae_point_to_mat2D(Point p, Mat2D m)
00499 {
00500     MATRIX2D_ASSERT((3 == m.rows && 1 == m.cols) || (1 == m.rows && 3 == m.cols));
00501
00502     if (3 == m.rows) {
00503         MAT2D_AT(m, 0, 0) = p.x;
00504         MAT2D_AT(m, 1, 0) = p.y;
00505         MAT2D_AT(m, 2, 0) = p.z;
00506     }
00507     if (3 == m.cols) {
00508         MAT2D_AT(m, 0, 0) = p.x;
00509         MAT2D_AT(m, 0, 1) = p.y;
00510         MAT2D_AT(m, 0, 2) = p.z;
00511     }
00512 }
00513
00522 Point ae_mat2D_to_point(Mat2D m)
00523 {
00524     Point res = {.x = MAT2D_AT(m, 0, 0), .y = MAT2D_AT(m, 1, 0), .z = MAT2D_AT(m, 2, 0), .w = 1};
00525     return res;
00526 }
00527
00540 Tri_mesh ae_tri_mesh_get_from_obj_file(char *file_path)
00541 {
00542     char current_line[ASM_MAX_LEN_LINE], current_word[ASM_MAX_LEN_LINE],
     current_num_str[ASM_MAX_LEN_LINE];
00543     char file_name[ASM_MAX_LEN_LINE], file_extention[ASM_MAX_LEN_LINE], mesh_name[ASM_MAX_LEN_LINE];
00544     int texture_warning_was_printed = 0;
00545
00546     strncpy(file_name, file_path, ASM_MAX_LEN_LINE);
00547     strncpy(file_extention, file_name, ASM_MAX_LEN_LINE);
00548
00549     /* check if file is an obj file*/
00550     asm_get_word_and_cut(file_name, file_extention, '.');
00551     asm_get_word_and_cut(file_name, file_extention, '.');
00552     if (strncmp(file_extention, ".obj", ASM_MAX_LEN_LINE)) {
00553         fprintf(stderr, "%s:%s:%d:\n[Error] unsupported file format: '%s'\n\n", __FILE__, __func__,
     __LINE__, file_name);
00554         exit(1);
00555     }
00556
00557     strncpy(mesh_name, file_name, ASM_MAX_LEN_LINE);
00558     while(asm_length(mesh_name)) {
00559         asm_get_word_and_cut(current_word, mesh_name, '/');
00560     }
00561
00562     strncpy(mesh_name, current_word, ASM_MAX_LEN_LINE);
00563
00564     strncpy(current_word, ".", ASM_MAX_LEN_LINE);
00565     strncat(file_name, ".obj", ASM_MAX_LEN_LINE/2);
00566     strncat(current_word, file_name, ASM_MAX_LEN_LINE/2);
00567
00568     FILE *fp_input = fopen(current_word, "rt");
00569     if (fp_input == NULL) {
00570         fprintf(stderr, "%s:%s:%d:\n[Error] failed to open input file: '%s', %s\n\n", __FILE__,
     __func__, __LINE__, current_word, strerror(errno));
00571         exit(1);
00572     }
00573
00574     // strncpy(output_file_name, "./build/", ASM_MAX_LEN_LINE);
00575     // strncat(output_file_name, mesh_name, ASM_MAX_LEN_LINE/2);
00576     // strncat(output_file_name, ".c", ASM_MAX_LEN_LINE/2);
00577     // FILE *fp_output = fopen(output_file_name, "wt");
00578     // if (fp_input == NULL) {
00579     //     fprintf(stderr, "%s:%s:%d:\n[Error] failed to open output file: '%s'. %s\n\n", __FILE__,
     __func__, __LINE__, output_file_name, strerror(errno));
00580     //     exit(1);
00581     // }
00582
00583     /* parsing data from file */
00584     Curve points = {0};
00585     ada_init_array(Point, points);
00586     Tri_mesh mesh = {0};
00587     ada_init_array(Tri, mesh);
00588
00589     int line_len;
00590
00591     while ((line_len = asm_get_line(fp_input, current_line)) != -1) {
00592         asm_get_next_word_from_line(current_word, current_line, ' ');
00593         if (!strncmp(current_word, "v", 1)) {
00594             Point p;
00595             asm_get_word_and_cut(current_word, current_line, ' ');
00596             asm_get_word_and_cut(current_word, current_line, ' ');
00597             p.x = atof(current_word);
00598             asm_get_word_and_cut(current_word, current_line, ' ');
```

```
00599             p.y = atof(current_word);
00600             asm_get_word_and_cut(current_word, current_line, ' ');
00601             p.z = atof(current_word);
00602             // printf("current word: %s\n", current_word);
00603             ada_append(Point, points, p);
00604             // break;
00605         }
00606         if (!strncmp(current_word, "f", 1)) {
00607             Tri tri1 = {0}, tri2 = {0};
00608 
00609             // printf("line: %s\nword: %s, %d\n", current_line, current_word, atoi(current_word));
00610             asm_get_word_and_cut(current_word, current_line, ' ');
00611             // printf("line: %s\nword: %s, %d\n", current_line, current_word, atoi(current_word));
00612 
00613             int number_of_spaces = asm_str_in_str(current_line, " ");
00614             // printf("%d\n", number_of_spaces);
00615             // exit(1);
00616             if (!(number_of_spaces == 3 || number_of_spaces == 4 || number_of_spaces == 5)) {
00617                 fprintf(stderr, "%s:%s:%d:\n[Error] there is unsupported number of vertices for a
    face: %d\n\n", __FILE__, __func__, __LINE__, number_of_spaces);
00618                 exit(1);
00619             }
00620             if (number_of_spaces == 3) {
00621                 /* there are 3 vertices for the face. */
00622                 asm_get_word_and_cut(current_word, current_line, ' ');
00623                 // printf("line: %s\nword: %s, %d\n", current_line, current_word, atoi(current_word));
00624                 int number_of_backslash = asm_str_in_str(current_word, "/");
00625                 if (number_of_backslash == 0) {
00626                     tri1.points[0] = points.elements[atoi(current_word)-1];
00627                     asm_get_word_and_cut(current_word, current_line, ' ');
00628                     tri1.points[1] = points.elements[atoi(current_word)-1];
00629                     asm_get_word_and_cut(current_word, current_line, ' ');
00630                     tri1.points[2] = points.elements[atoi(current_word)-1];
00631                 }
00632                 if (number_of_backslash == 2) {
00633                     if (!texture_warning_was_printed) {
00634                         fprintf(stderr, "%s:%s:%d\n[Warning] texture and normals data ignored of file
    at - '%s'\n\n", __FILE__, __func__, __LINE__, file_path);
00635                         texture_warning_was_printed = 1;
00636                     }
00637 
00638                     asm_get_word_and_cut(current_num_str, current_word, '/');
00639                     // printf("line: %s\nword: %s\nnum str: %s, %d\n", current_line, current_word,
    current_num_str, atoi(current_num_str));
00640                     tri1.points[0] = points.elements[atoi(current_num_str)-1];
00641 
00642                     asm_get_word_and_cut(current_word, current_line, ' ');
00643                     asm_get_word_and_cut(current_num_str, current_word, '/');
00644                     // printf("line: %s\nword: %s\nnum str: %s, %d\n", current_line, current_word,
    current_num_str, atoi(current_num_str));
00645                     tri1.points[1] = points.elements[atoi(current_num_str)-1];
00646 
00647                     asm_get_word_and_cut(current_word, current_line, ' ');
00648                     asm_get_word_and_cut(current_num_str, current_word, '/');
00649                     // printf("line: %s\nword: %s\nnum str: %s, %d\n", current_line, current_word,
    current_num_str, atoi(current_num_str));
00650                     tri1.points[2] = points.elements[atoi(current_num_str)-1];
00651                 }
00652 
00653                 tri1.to_draw = true;
00654                 tri1.light_intensity[0] = 1;
00655                 tri1.light_intensity[1] = 1;
00656                 tri1.light_intensity[2] = 1;
00657                 tri1.colors[0] = 0xFFFFFFFF;
00658                 tri1.colors[1] = 0xFFFFFFFF;
00659                 tri1.colors[2] = 0xFFFFFFFF;
00660 
00661                 ada_append(Tri, mesh, tri1);
00662                 // AE_PRINT_TRI(tri1);
00663             } else if (number_of_spaces == 5 || number_of_spaces == 4) {
00664                 /* there are 4 vertices for the face. */
00665                 /* sometimes there is a space in the end */
00666                 asm_get_word_and_cut(current_word, current_line, ' ');
00667                 // printf("line: %s\nword: %s, %d\n", current_line, current_word, atoi(current_word));
00668                 int number_of_backslash = asm_str_in_str(current_word, "/");
00669                 if (number_of_backslash == 0) {
00670                     tri1.points[0] = points.elements[atoi(current_word)-1];
00671                     asm_get_word_and_cut(current_word, current_line, ' ');
00672                     tri1.points[1] = points.elements[atoi(current_word)-1];
00673                     asm_get_word_and_cut(current_word, current_line, ' ');
00674                     tri1.points[2] = points.elements[atoi(current_word)-1];
00675                 }
00676                 if (number_of_backslash == 2 || number_of_backslash == 1) {
00677                     if (!texture_warning_was_printed) {
00678                         fprintf(stderr, "%s:%s:%d\n[Warning] texture and normals data ignored of file
    at - '%s'\n\n", __FILE__, __func__, __LINE__, file_path);
00679                         texture_warning_was_printed = 1;
```

```
00680                     }
00681
00682                     asm_get_word_and_cut(current_num_str, current_word, '/');
00683                     // printf("line: %s\nword: %s\nnum str: %s, %d\n", current_line, current_word,
      current_num_str, atoi(current_num_str));
00684                     tri1.points[0] = points.elements[atoi(current_num_str)-1];
00685                     tri2.points[2] = points.elements[atoi(current_num_str)-1];
00686
00687                     asm_get_word_and_cut(current_word, current_line, ' ');
00688                     asm_get_word_and_cut(current_num_str, current_word, '/');
00689                     // printf("line: %s\nword: %s\nnum str: %s, %d\n", current_line, current_word,
      current_num_str, atoi(current_num_str));
00690                     tri1.points[1] = points.elements[atoi(current_num_str)-1];
00691
00692                     asm_get_word_and_cut(current_word, current_line, ' ');
00693                     asm_get_word_and_cut(current_num_str, current_word, '/');
00694                     // printf("line: %s\nword: %s\nnum str: %s, %d\n", current_line, current_word,
      current_num_str, atoi(current_num_str));
00695                     tri1.points[2] = points.elements[atoi(current_num_str)-1];
00696                     tri2.points[0] = points.elements[atoi(current_num_str)-1];
00697
00698                     asm_get_word_and_cut(current_word, current_line, ' ');
00699                     asm_get_word_and_cut(current_num_str, current_word, '/');
00700                     // printf("line: %s\nword: %s\nnum str: %s, %d\n", current_line, current_word,
      current_num_str, atoi(current_num_str));
00701                     tri2.points[1] = points.elements[atoi(current_num_str)-1];
00702                 }
00703
00704                 tri1.to_draw = true;
00705                 tri1.light_intensity[0] = 1;
00706                 tri1.light_intensity[1] = 1;
00707                 tri1.light_intensity[2] = 1;
00708                 tri1.colors[0] = 0xFFFFFFFF;
00709                 tri1.colors[1] = 0xFFFFFFFF;
00710                 tri1.colors[2] = 0xFFFFFFFF;
00711
00712                 tri2.to_draw = true;
00713                 tri2.light_intensity[0] = 1;
00714                 tri2.light_intensity[1] = 1;
00715                 tri2.light_intensity[2] = 1;
00716                 tri2.colors[0] = 0xFFFFFFFF;
00717                 tri2.colors[1] = 0xFFFFFFFF;
00718                 tri2.colors[2] = 0xFFFFFFFF;
00719
00720                 ada_append(Tri, mesh, tri1);
00721                 ada_append(Tri, mesh, tri2);
00722                 // AE_PRINT_TRI(tri1);
00723                 // AE_PRINT_TRI(tri2);
00724             }
00725             // exit(2);
00726         }
00727     }
00728
00729     return mesh;
00730 }
00731
00743 Tri_mesh ae_tri_mesh_get_from_stl_file(char *file_path)
00744 {
00745     FILE *file;
00746     file = fopen(file_path, "rb");
00747     if (file == NULL) {
00748         fprintf(stderr, "%s:%s:%d:\n[Error] failed to open input file: '%s', %s\n\n", __FILE__,
      __func__, __LINE__, file_path, strerror(errno));
00749         exit(1);
00750     }
00751
00752     char header[STL_HEADER_SIZE];
00753     fread(header, STL_HEADER_SIZE, 1, file);
00754     // dprintSTRING(header);
00755
00756     uint32_t num_of_tri;
00757     fread(&num_of_tri, STL_NUM_SIZE, 1, file);
00758     // dprintINT(num_of_tri);
00759
00760     Tri_mesh mesh;
00761     ada_init_array(Tri, mesh);
00762     for (size_t i = 0; i < num_of_tri; i++) {
00763         Tri temp_tri = {0};
00764
00765         fread(&(temp_tri.normals[0].x), STL_NUM_SIZE, 1, file);
00766         fread(&(temp_tri.normals[0].y), STL_NUM_SIZE, 1, file);
00767         fread(&(temp_tri.normals[0].z), STL_NUM_SIZE, 1, file);
00768
00769         temp_tri.normals[0].x = - temp_tri.normals[0].x;
00770         temp_tri.normals[0].y = - temp_tri.normals[0].y;
00771         temp_tri.normals[0].z = - temp_tri.normals[0].z;
00772
```

```
00773         temp_tri.normals[1] = temp_tri.normals[0];
00774         temp_tri.normals[2] = temp_tri.normals[0];
00775 
00776         fread(&(temp_tri.points[0].x), STL_NUM_SIZE, 1, file);
00777         fread(&(temp_tri.points[0].y), STL_NUM_SIZE, 1, file);
00778         fread(&(temp_tri.points[0].z), STL_NUM_SIZE, 1, file);
00779 
00780         fread(&(temp_tri.points[1].x), STL_NUM_SIZE, 1, file);
00781         fread(&(temp_tri.points[1].y), STL_NUM_SIZE, 1, file);
00782         fread(&(temp_tri.points[1].z), STL_NUM_SIZE, 1, file);
00783 
00784         fread(&(temp_tri.points[2].x), STL_NUM_SIZE, 1, file);
00785         fread(&(temp_tri.points[2].y), STL_NUM_SIZE, 1, file);
00786         fread(&(temp_tri.points[2].z), STL_NUM_SIZE, 1, file);
00787 
00788         fseek(file, STL_ATTRIBUTE_BITS_SIZE, SEEK_CUR);
00789 
00790         temp_tri.to_draw = true;
00791         temp_tri.light_intensity[0] = 1;
00792         temp_tri.light_intensity[1] = 1;
00793         temp_tri.light_intensity[2] = 1;
00794         temp_tri.colors[0] = 0xFFFFFFFF;
00795         temp_tri.colors[1] = 0xFFFFFFFF;
00796         temp_tri.colors[2] = 0xFFFFFFFF;
00797 
00798         // ae_tri_set_normals(&temp_tri);
00799 
00800         ada_append(Tri, mesh, temp_tri);
00801     }
00802 
00803     return mesh;
00804 }
00805 
00816 Tri_mesh ae_tri_mesh_get_from_file(char *file_path)
00817 {
00818     char file_extention[ASM_MAX_LEN_LINE], temp_word[ASM_MAX_LEN_LINE];
00819 
00820     strncpy(file_extention, file_path, ASM_MAX_LEN_LINE);
00821 
00822     int num_of_dots;
00823     while ((num_of_dots = asm_str_in_str(file_extention, ".")) >= 1) {
00824         asm_get_word_and_cut(temp_word, file_extention, '.');
00825     }
00826 
00827     if (!(!strncmp(file_extention, "obj", 3) || !strncmp(file_extention, "STL", 3) ||
       !strncmp(file_extention, "stl", 3))) {
00828         fprintf(stderr, "%s:%s:%d:\n[Error] unsupported file format: '%s'\n\n", __FILE__, __func__,
      __LINE__, file_path);
00829         exit(1);
00830     }
00831 
00832     if (!strncmp(file_extention, "STL", 3) || !strncmp(file_extention, "stl", 3)) {
00833         return ae_tri_mesh_get_from_stl_file(file_path);
00834     } else if (!strncmp(file_extention, "obj", 3)) {
00835         return ae_tri_mesh_get_from_obj_file(file_path);
00836     }
00837 
00838     Tri_mesh null_mesh = {0};
00839     return null_mesh;
00840 }
00841 
00851 void ae_tri_mesh_append_copy(Tri_mesh_array *mesh_array, Tri_mesh mesh)
00852 {
00853     Tri_mesh_array temp_mesh_array = *mesh_array;
00854     Tri_mesh temp_mesh;
00855     ada_init_array(Tri, temp_mesh);
00856     for (size_t i = 0; i < mesh.length; i++) {
00857         ada_append(Tri, temp_mesh, mesh.elements[i]);
00858     }
00859     ada_append(Tri_mesh, temp_mesh_array, temp_mesh);
00860 
00861 
00862     *mesh_array = temp_mesh_array;
00863 }
00864 
00875 Tri_mesh ae_tri_mesh_get_from_quad_mesh(Quad_mesh q_mesh)
00876 {
00877     Tri_mesh t_mesh;
00878     ada_init_array(Tri, t_mesh);
00879 
00880     for (size_t q_index = 0; q_index < q_mesh.length; q_index++) {
00881         Quad current_q = q_mesh.elements[q_index];
00882         Tri temp_t = {.to_draw = current_q.to_draw};
00883 
00884         temp_t.points[0] = current_q.points[0];
00885         temp_t.colors[0] = current_q.colors[0];
00886         temp_t.normals[0] = current_q.normals[0];
```

```
00887         temp_t.light_intensity[0] = current_q.light_intensity[0];
00888         temp_t.points[1] = current_q.points[1];
00889         temp_t.colors[1] = current_q.colors[1];
00890         temp_t.normals[1] = current_q.normals[1];
00891         temp_t.light_intensity[1] = current_q.light_intensity[1];
00892         temp_t.points[2] = current_q.points[2];
00893         temp_t.colors[2] = current_q.colors[2];
00894         temp_t.normals[2] = current_q.normals[2];
00895         temp_t.light_intensity[2] = current_q.light_intensity[2];
00896 
00897         ada_append(Tri, t_mesh, temp_t);
00898 
00899         temp_t.points[0] = current_q.points[2];
00900         temp_t.colors[0] = current_q.colors[2];
00901         temp_t.normals[0] = current_q.normals[2];
00902         temp_t.light_intensity[0] = current_q.light_intensity[2];
00903         temp_t.points[1] = current_q.points[3];
00904         temp_t.colors[1] = current_q.colors[3];
00905         temp_t.normals[1] = current_q.normals[3];
00906         temp_t.light_intensity[1] = current_q.light_intensity[3];
00907         temp_t.points[2] = current_q.points[0];
00908         temp_t.colors[2] = current_q.colors[0];
00909         temp_t.normals[2] = current_q.normals[0];
00910         temp_t.light_intensity[2] = current_q.light_intensity[0];
00911 
00912         ada_append(Tri, t_mesh, temp_t);
00913     }
00914 
00915     return t_mesh;
00916 }
00917 
00925 void ae_print_points(Curve p)
00926 {
00927     for (size_t i = 0; i < p.length; i++) {
00928         printf("point %3zu: (%5f, %5f, %5f)\n", i, p.elements[i].x, p.elements[i].y, p.elements[i].z);
00929     }
00930 }
00931 
00942 void ae_print_tri(Tri tri, char *name, size_t padding)
00943 {
00944     printf("%*s%s:\n", (int) padding, "", name);
00945     printf("%*s    (%f, %f, %f)\n%*s    (%f, %f, %f)\n%*s    (%f, %f, %f)\n", (int) padding, "",
       tri.points[0].x, tri.points[0].y, tri.points[0].z, (int) padding, "", tri.points[1].x,
       tri.points[1].y, tri.points[1].z, (int) padding, "", tri.points[2].x, tri.points[2].y,
       tri.points[2].z);
00946     printf("%*s    draw? %d\n", (int)padding, "", tri.to_draw);
00947 }
00948 
00958 void ae_print_tri_mesh(Tri_mesh mesh, char *name, size_t padding)
00959 {
00960     char tri_name[256];
00961     printf("%*s%s:\n", (int) padding, "", name);
00962     for (size_t i = 0; i < mesh.length; i++) {
00963         snprintf(tri_name, 256, "tri %zu", i);
00964         ae_print_tri(mesh.elements[i], tri_name, 4);
00965     }
00966 }
00967 
00976 Point ae_point_normalize_xyz(Point p)
00977 {
00978     Point res = {0};
00979 
00980     float norma = ae_point_calc_norma(p);
00981 
00982     res.x = p.x / norma;
00983     res.y = p.y / norma;
00984     res.z = p.z / norma;
00985     res.w = p.w;
00986 
00987     return res;
00988 }
00989 
00998 void ae_tri_set_normals(Tri *tri)
00999 {
01000     ae_assert_tri_is_valid(*tri);
01001 
01002     Mat2D point  = mat2D_alloc(3, 1);
01003     Mat2D to_p   = mat2D_alloc(3, 1);
01004     Mat2D from_p = mat2D_alloc(3, 1);
01005     Mat2D normal = mat2D_alloc(3, 1);
01006 
01007     for (int i = 0; i < 3; i++) {
01008         int current_index  = i;
01009         int next_index     = (i + 1) % 3;
01010         int previous_index = (i - 1 + 3) % 3;
01011         ae_point_to_mat2D(tri->points[current_index], point);
01012         ae_point_to_mat2D(tri->points[next_index], from_p);
```

```
01013         ae_point_to_mat2D(tri->points[previous_index], to_p);
01014
01015         mat2D_sub(from_p, point);
01016         mat2D_sub(point, to_p);
01017
01018         mat2D_copy(to_p, point);
01019
01020         mat2D_cross(normal, to_p, from_p);
01021         // mat2D_cross(normal, from_p, to_p);
01022         mat2D_normalize(normal);
01023
01024         tri->normals[current_index] = ae_mat2D_to_point(normal);
01025     }
01026
01027     mat2D_free(point);
01028     mat2D_free(to_p);
01029     mat2D_free(from_p);
01030     mat2D_free(normal);
01031 }
01032
01041 Point ae_tri_get_average_normal(Tri tri)
01042 {
01043     Point normal0 = tri.normals[0];
01044     Point normal1 = tri.normals[1];
01045     Point normal2 = tri.normals[2];
01046
01047     Point res;
01048     res.x = (normal0.x + normal1.x + normal2.x) / 3;
01049     res.y = (normal0.y + normal1.y + normal2.y) / 3;
01050     res.z = (normal0.z + normal1.z + normal2.z) / 3;
01051     res.w = (normal0.w + normal1.w + normal2.w) / 3;
01052
01053     return ae_point_normalize_xyz(res);
01054 }
01055
01062 Point ae_tri_get_average_point(Tri tri)
01063 {
01064     Point point0 = tri.points[0];
01065     Point point1 = tri.points[1];
01066     Point point2 = tri.points[2];
01067
01068     Point res;
01069     res.x = (point0.x + point1.x + point2.x) / 3;
01070     res.y = (point0.y + point1.y + point2.y) / 3;
01071     res.z = (point0.z + point1.z + point2.z) / 3;
01072     res.w = (point0.w + point1.w + point2.w) / 3;
01073
01074     return res;
01075 }
01076
01086 void ae_tri_calc_normal(Mat2D normal, Tri tri)
01087 {
01088     AE_ASSERT(3 == normal.rows && 1 == normal.cols);
01089     ae_assert_tri_is_valid(tri);
01090
01091     Mat2D a = mat2D_alloc(3, 1);
01092     Mat2D b = mat2D_alloc(3, 1);
01093     Mat2D c = mat2D_alloc(3, 1);
01094
01095     ae_point_to_mat2D(tri.points[0], a);
01096     ae_point_to_mat2D(tri.points[1], b);
01097     ae_point_to_mat2D(tri.points[2], c);
01098
01099     mat2D_sub(b, a);
01100     mat2D_sub(c, a);
01101
01102     mat2D_cross(normal, b, c);
01103
01104     mat2D_mult(normal, 1/mat2D_calc_norma(normal));
01105
01106     mat2D_free(a);
01107     mat2D_free(b);
01108     mat2D_free(c);
01109 }
01110
01121 void ae_tri_mesh_translate(Tri_mesh mesh, float x, float y, float z)
01122 {
01123     for (size_t i = 0; i < mesh.length; i++) {
01124         for (int j = 0; j < 3; j++) {
01125             mesh.elements[i].points[j].x += x;
01126             mesh.elements[i].points[j].y += y;
01127             mesh.elements[i].points[j].z += z;
01128         }
01129     }
01130 }
01131
01143 void ae_tri_mesh_rotate_Euler_xyz(Tri_mesh mesh, float phi_deg, float theta_deg, float psi_deg)
```

```
01144 {
01145     Mat2D RotZ = mat2D_alloc(3,3);
01146     mat2D_set_rot_mat_z(RotZ, psi_deg);
01147     Mat2D RotY = mat2D_alloc(3,3);
01148     mat2D_set_rot_mat_y(RotY, theta_deg);
01149     Mat2D RotX = mat2D_alloc(3,3);
01150     mat2D_set_rot_mat_x(RotX, phi_deg);
01151     Mat2D DCM = mat2D_alloc(3,3);
01152     // mat2D_fill(DCM,0);
01153     Mat2D temp = mat2D_alloc(3,3);
01154     // mat2D_fill(temp,0);
01155     mat2D_dot(temp, RotY, RotZ);
01156     mat2D_dot(DCM, RotX, temp); /* I have a DCM */
01157
01158     Mat2D src_point_mat = mat2D_alloc(3,1);
01159     Mat2D des_point_mat = mat2D_alloc(3,1);
01160
01161     for (size_t i = 0; i < mesh.length; i++) {
01162         for (int j = 0; j < 3; j++) {
01163             // mat2D_fill(src_point_mat, 0);
01164             // mat2D_fill(des_point_mat, 0);
01165             Point des = {0};
01166             Point src = mesh.elements[i].points[j];
01167
01168             MAT2D_AT(src_point_mat, 0, 0) = src.x;
01169             MAT2D_AT(src_point_mat, 1, 0) = src.y;
01170             MAT2D_AT(src_point_mat, 2, 0) = src.z;
01171
01172             mat2D_dot(des_point_mat, DCM, src_point_mat);
01173
01174             des.x = MAT2D_AT(des_point_mat, 0, 0);
01175             des.y = MAT2D_AT(des_point_mat, 1, 0);
01176             des.z = MAT2D_AT(des_point_mat, 2, 0);
01177
01178             mesh.elements[i].points[j] = des;
01179         }
01180     }
01181
01182     ae_tri_mesh_set_normals(mesh);
01183
01184     mat2D_free(RotZ);
01185     mat2D_free(RotY);
01186     mat2D_free(RotX);
01187     mat2D_free(DCM);
01188     mat2D_free(temp);
01189     mat2D_free(src_point_mat);
01190     mat2D_free(des_point_mat);
01191 }
01192
01206 void ae_tri_mesh_set_bounding_box(Tri_mesh mesh, float *x_min, float *x_max, float *y_min, float
      *y_max, float *z_min, float *z_max)
01207 {
01208     float xmin = FLT_MAX, xmax = FLT_MIN;
01209     float ymin = FLT_MAX, ymax = FLT_MIN;
01210     float zmin = FLT_MAX, zmax = FLT_MIN;
01211
01212     float x, y, z;
01213
01214     for (size_t t = 0; t < mesh.length; t++) {
01215         for (size_t p = 0; p < 3; p++) {
01216             x = mesh.elements[t].points[p].x;
01217             y = mesh.elements[t].points[p].y;
01218             z = mesh.elements[t].points[p].z;
01219             if (x > xmax) xmax = x;
01220             if (x < xmin) xmin = x;
01221             if (y > ymax) ymax = y;
01222             if (y < ymin) ymin = y;
01223             if (z > zmax) zmax = z;
01224             if (z < zmin) zmin = z;
01225         }
01226     }
01227     *x_min = xmin;
01228     *x_max = xmax;
01229     *y_min = ymin;
01230     *y_max = ymax;
01231     *z_min = zmin;
01232     *z_max = zmax;
01233 }
01234
01244 void ae_tri_mesh_normalize(Tri_mesh mesh)
01245 {
01246     float xmax, xmin, ymax, ymin, zmax, zmin;
01247     ae_tri_mesh_set_bounding_box(mesh, &xmin, &xmax, &ymin, &ymax, &zmin, &zmax);
01248
01249     for (size_t t = 0; t < mesh.length; t++) {
01250         for (size_t p = 0; p < 3; p++) {
01251             float x, y, z;
```

```
01252             x = mesh.elements[t].points[p].x;
01253             y = mesh.elements[t].points[p].y;
01254             z = mesh.elements[t].points[p].z;
01255 
01256             float xdiff = xmax-xmin;
01257             float ydiff = ymax-ymin;
01258             float zdiff = zmax-zmin;
01259             float max_diff = fmax(xdiff, fmax(ydiff, zdiff));
01260             float xfactor = xdiff/max_diff;
01261             float yfactor = ydiff/max_diff;
01262             float zfactor = zdiff/max_diff;
01263 
01264             x = (((x - xmin) / (xdiff)) * 2 - 1) * xfactor;
01265             y = (((y - ymin) / (ydiff)) * 2 - 1) * yfactor;
01266             z = (((z - zmin) / (zdiff)) * 2 - 1) * zfactor;
01267 
01268             mesh.elements[t].points[p].x = x;
01269             mesh.elements[t].points[p].y = y;
01270             mesh.elements[t].points[p].z = z;
01271         }
01272     }
01273 }
01274 
01283 void ae_tri_mesh_flip_normals(Tri_mesh mesh)
01284 {
01285     for (size_t i = 0; i < mesh.length; i++) {
01286         Tri res_tri, tri = mesh.elements[i];
01287 
01288         res_tri.to_draw = tri.to_draw;
01289 
01290         res_tri.colors[0]          = tri.colors[2];
01291         res_tri.light_intensity[0] = tri.light_intensity[2];
01292         res_tri.normals[0]         = tri.normals[2];
01293         res_tri.points[0]          = tri.points[2];
01294         res_tri.tex_points[0]      = tri.tex_points[2];
01295 
01296         res_tri.colors[1]          = tri.colors[1];
01297         res_tri.light_intensity[1] = tri.light_intensity[1];
01298         res_tri.normals[1]         = tri.normals[1];
01299         res_tri.points[1]          = tri.points[1];
01300         res_tri.tex_points[1]      = tri.tex_points[1];
01301 
01302         res_tri.colors[2]          = tri.colors[0];
01303         res_tri.light_intensity[2] = tri.light_intensity[0];
01304         res_tri.normals[2]         = tri.normals[0];
01305         res_tri.points[2]          = tri.points[0];
01306         res_tri.tex_points[2]      = tri.tex_points[0];
01307 
01308         ae_tri_set_normals(&res_tri);
01309 
01310         mesh.elements[i] = res_tri;
01311     }
01312 }
01313 
01321 void ae_tri_mesh_set_normals(Tri_mesh mesh)
01322 {
01323     for (size_t i = 0; i < mesh.length; i++) {
01324         ae_tri_set_normals(&(mesh.elements[i]));
01325     }
01326 }
01327 
01336 void ae_quad_set_normals(Quad *quad)
01337 {
01338     ae_assert_quad_is_valid(*quad);
01339 
01340     Mat2D point  = mat2D_alloc(3, 1);
01341     Mat2D to_p   = mat2D_alloc(3, 1);
01342     Mat2D from_p = mat2D_alloc(3, 1);
01343     Mat2D normal = mat2D_alloc(3, 1);
01344 
01345     for (int i = 0; i < 4; i++) {
01346         int current_index  = i;
01347         int next_index     = (i + 1) % 4;
01348         int previous_index = (i - 1 + 4) % 4;
01349         ae_point_to_mat2D(quad->points[current_index], point);
01350         ae_point_to_mat2D(quad->points[next_index], from_p);
01351         ae_point_to_mat2D(quad->points[previous_index], to_p);
01352 
01353         mat2D_sub(from_p, point);
01354         mat2D_sub(point, to_p);
01355 
01356         mat2D_copy(to_p, point);
01357 
01358         mat2D_cross(normal, to_p, from_p);
01359         mat2D_normalize(normal);
01360 
01361         quad->normals[current_index] = ae_mat2D_to_point(normal);
```

```
01362     }
01363
01364     mat2D_free(point);
01365     mat2D_free(to_p);
01366     mat2D_free(from_p);
01367     mat2D_free(normal);
01368
01369 }
01370
01379 Point ae_quad_get_average_normal(Quad quad)
01380 {
01381     Point normal0 = quad.normals[0];
01382     Point normal1 = quad.normals[1];
01383     Point normal2 = quad.normals[2];
01384     Point normal3 = quad.normals[3];
01385
01386     Point res;
01387     res.x = (normal0.x + normal1.x + normal2.x + normal3.x) / 4;
01388     res.y = (normal0.y + normal1.y + normal2.y + normal3.y) / 4;
01389     res.z = (normal0.z + normal1.z + normal2.z + normal3.z) / 4;
01390     res.w = (normal0.w + normal1.w + normal2.w + normal3.w) / 4;
01391
01392     res = ae_point_normalize_xyz(res);
01393
01394     return res;
01395 }
01396
01403 Point ae_quad_get_average_point(Quad quad)
01404 {
01405     Point point0 = quad.points[0];
01406     Point point1 = quad.points[1];
01407     Point point2 = quad.points[2];
01408     Point point3 = quad.points[3];
01409
01410     Point res;
01411     res.x = (point0.x + point1.x + point2.x + point3.x) / 4;
01412     res.y = (point0.y + point1.y + point2.y + point3.y) / 4;
01413     res.z = (point0.z + point1.z + point2.z + point3.z) / 4;
01414     res.w = (point0.w + point1.w + point2.w + point3.w) / 4;
01415
01416     return res;
01417 }
01418
01428 void ae_quad_calc_normal(Mat2D normal, Quad quad)
01429 {
01430     AE_ASSERT(3 == normal.rows && 1 == normal.cols);
01431     ae_assert_quad_is_valid(quad);
01432
01433     Mat2D a = mat2D_alloc(3, 1);
01434     Mat2D b = mat2D_alloc(3, 1);
01435     Mat2D c = mat2D_alloc(3, 1);
01436
01437     ae_point_to_mat2D(quad.points[0], a);
01438     ae_point_to_mat2D(quad.points[1], b);
01439     ae_point_to_mat2D(quad.points[2], c);
01440
01441     mat2D_sub(b, a);
01442     mat2D_sub(c, a);
01443
01444     mat2D_cross(normal, b, c);
01445
01446     mat2D_mult(normal, 1/mat2D_calc_norma(normal));
01447
01448     mat2D_free(a);
01449     mat2D_free(b);
01450     mat2D_free(c);
01451 }
01452
01461 void ae_curve_copy(Curve *des, Curve src)
01462 {
01463     Curve temp_des = *des;
01464     temp_des.length = 0;
01465
01466     for (size_t i = 0; i < src.length; i++) {
01467         ada_append(Point, temp_des, src.elements[i]);
01468     }
01469
01470     *des = temp_des;
01471 }
01472
01487 void ae_tri_calc_light_intensity(Tri *tri, Scene *scene, Lighting_mode lighting_mode)
01488 {
01489     /* based on the lighting model described in: 'Alexandru C. Telea-Data Visualization_ Principles
      and Practice-A K Peters_CRC Press (2014)' Pg.29 */
01490     Point L = {0};
01491     Point r = {0};
01492     Point v = {0};
```

```
01493     Point mL = {0};
01494     Point pml = {0};
01495     Point mLn2n = {0};
01496     Point ave_norm = ae_tri_get_average_normal(*tri);
01497     Point camera_pos = ae_mat2D_to_point(scene->camera.current_position);
01498 
01499     float c_ambi = scene->material0.c_ambi;
01500     float c_diff = scene->material0.c_diff;
01501     float c_spec = scene->material0.c_spec;
01502     float alpha  = scene->material0.specular_power_alpha;
01503 
01504     switch (lighting_mode) {
01505     case AE_LIGHTING_FLAT:
01506         for (int i = 0; i < 3; i++) {
01507             if (scene->light_source0.light_direction_or_pos.w == 0) {
01508                 L = scene->light_source0.light_direction_or_pos;
01509                 L = ae_point_normalize_xyz(L);
01510                 mL = L;
01511                 ae_point_mult(mL, -1);
01512             } else {
01513                 Point l = scene->light_source0.light_direction_or_pos;
01514                 Point p = tri->points[i];
01515                 ae_point_sub_point(pml, p, l);
01516                 pml = ae_point_normalize_xyz(pml);
01517                 L = pml;
01518                 L.w = 0;
01519                 mL = L;
01520                 ae_point_mult(mL, -1);
01521             }
01522 
01523             ae_point_sub_point(v, camera_pos, ae_tri_get_average_point(*tri));
01524             float mL_dot_norm = ae_point_dot_point(mL, ave_norm);
01525             mLn2n = ave_norm;
01526             ae_point_mult(mLn2n, 2 * mL_dot_norm);
01527             ae_point_add_point(r, L, mLn2n);
01528 
01529             tri->light_intensity[i] = c_ambi + scene->light_source0.light_intensity * (c_diff *
      fmaxf(mL_dot_norm, 0) + c_spec * powf(fmaxf(ae_point_dot_point(r, v), 0), alpha));
01530         }
01531         break;
01532     case AE_LIGHTING_SMOOTH:
01533         for (int i = 0; i < 3; i++) {
01534             if (scene->light_source0.light_direction_or_pos.w == 0) {
01535                 L = scene->light_source0.light_direction_or_pos;
01536                 L = ae_point_normalize_xyz(L);
01537                 mL = L;
01538                 ae_point_mult(mL, -1);
01539             } else {
01540                 Point l = scene->light_source0.light_direction_or_pos;
01541                 Point p = tri->points[i];
01542                 ae_point_sub_point(pml, p, l);
01543                 pml = ae_point_normalize_xyz(pml);
01544                 L = pml;
01545                 L.w = 0;
01546                 mL = L;
01547                 ae_point_mult(mL, -1);
01548             }
01549             ae_point_sub_point(v, camera_pos, tri->points[i]);
01550             float mL_dot_norm = ae_point_dot_point(mL, tri->normals[i]);
01551             mLn2n = tri->normals[i];
01552             ae_point_mult(mLn2n, 2 * mL_dot_norm);
01553             ae_point_add_point(r, L, mLn2n);
01554 
01555             tri->light_intensity[i] = c_ambi + scene->light_source0.light_intensity * (c_diff *
      fmaxf(mL_dot_norm, 0) + c_spec * powf(fmaxf(ae_point_dot_point(r, v), 0), alpha));
01556         }
01557         break;
01558     default:
01559         for (int i = 0; i < 3; i++) {
01560             tri->light_intensity[i] = 1;
01561         }
01562         break;
01563     }
01564 
01565     for (int i = 0; i < 3; i++) {
01566         tri->light_intensity[i] = fminf(1, fmaxf(0, tri->light_intensity[i]));
01567     }
01568 }
01569 
01580 void ae_quad_calc_light_intensity(Quad *quad, Scene *scene, Lighting_mode lighting_mode)
01581 {
01582     /* based on the lighting model described in: 'Alexandru C. Telea-Data Visualization_ Principles
      and Practice-A K Peters_CRC Press (2014)' Pg.29 */
01583     Point L = {0};
01584     Point r = {0};
01585     Point v = {0};
01586     Point mL = {0};
```

```
01587     Point pml = {0};
01588     Point mLn2n = {0};
01589     Point ave_norm = ae_quad_get_average_normal(*quad);
01590     Point camera_pos = ae_mat2D_to_point(scene->camera.current_position);
01591 
01592     float c_ambi = scene->material0.c_ambi;
01593     float c_diff = scene->material0.c_diff;
01594     float c_spec = scene->material0.c_spec;
01595     float alpha  = scene->material0.specular_power_alpha;
01596 
01597     switch (lighting_mode) {
01598     case AE_LIGHTING_FLAT:
01599         for (int i = 0; i < 4; i++) {
01600             if (scene->light_source0.light_direction_or_pos.w == 0) {
01601                 L = scene->light_source0.light_direction_or_pos;
01602                 L = ae_point_normalize_xyz(L);
01603                 mL = L;
01604                 ae_point_mult(mL, -1);
01605             } else {
01606                 Point l = scene->light_source0.light_direction_or_pos;
01607                 Point p = quad->points[i];
01608                 ae_point_sub_point(pml, p, l);
01609                 pml = ae_point_normalize_xyz(pml);
01610                 L = pml;
01611                 L.w = 0;
01612                 mL = L;
01613                 ae_point_mult(mL, -1);
01614             }
01615 
01616             ae_point_sub_point(v, camera_pos, ae_quad_get_average_point(*quad));
01617             float mL_dot_norm = ae_point_dot_point(mL, ave_norm);
01618             mLn2n = ave_norm;
01619             ae_point_mult(mLn2n, 2 * mL_dot_norm);
01620             ae_point_add_point(r, L, mLn2n);
01621 
01622             quad->light_intensity[i] = c_ambi + scene->light_source0.light_intensity * (c_diff *
      fmaxf(mL_dot_norm, 0) + c_spec * powf(fmaxf(ae_point_dot_point(r, v), 0), alpha));
01623         }
01624         break;
01625     case AE_LIGHTING_SMOOTH:
01626         for (int i = 0; i < 4; i++) {
01627             if (scene->light_source0.light_direction_or_pos.w == 0) {
01628                 L = scene->light_source0.light_direction_or_pos;
01629                 L = ae_point_normalize_xyz(L);
01630                 mL = L;
01631                 ae_point_mult(mL, -1);
01632             } else {
01633                 Point l = scene->light_source0.light_direction_or_pos;
01634                 Point p = quad->points[i];
01635                 ae_point_sub_point(pml, p, l);
01636                 pml = ae_point_normalize_xyz(pml);
01637                 L = pml;
01638                 L.w = 0;
01639                 mL = L;
01640                 ae_point_mult(mL, -1);
01641             }
01642             ae_point_sub_point(v, camera_pos, quad->points[i]);
01643             float mL_dot_norm = ae_point_dot_point(mL, quad->normals[i]);
01644             mLn2n = quad->normals[i];
01645             ae_point_mult(mLn2n, 2 * mL_dot_norm);
01646             ae_point_add_point(r, L, mLn2n);
01647 
01648             quad->light_intensity[i] = c_ambi + scene->light_source0.light_intensity * (c_diff *
      fmaxf(mL_dot_norm, 0) + c_spec * powf(fmaxf(ae_point_dot_point(r, v), 0), alpha));
01649         }
01650         break;
01651     default:
01652         for (int i = 0; i < 4; i++) {
01653             quad->light_intensity[i] = 1;
01654         }
01655         break;
01656     }
01657 
01658     for (int i = 0; i < 4; i++) {
01659         quad->light_intensity[i] = fminf(1, fmaxf(0, quad->light_intensity[i]));
01660     }
01661 }
01662 
01680 Point ae_line_itersect_plane(Mat2D plane_p, Mat2D plane_n, Mat2D line_start, Mat2D line_end, float *t)
01681 {
01682     mat2D_normalize(plane_n);
01683     float plane_d = - mat2D_dot_product(plane_n, plane_p);
01684     float ad = mat2D_dot_product(line_start, plane_n);
01685     float bd = mat2D_dot_product(line_end, plane_n);
01686     *t  = (- plane_d - ad) / (bd - ad);
01687     mat2D_sub(line_end, line_start);
01688     Mat2D line_start_to_end = line_end;
```

```
01689     mat2D_mult(line_start_to_end, *t);
01690     Mat2D line_to_intersection = line_start_to_end;
01691 
01692     Mat2D intersection_p = mat2D_alloc(3, 1);
01693     mat2D_fill(intersection_p, 0);
01694     mat2D_add(intersection_p, line_start);
01695     mat2D_add(intersection_p, line_to_intersection);
01696 
01697     Point ans_p = ae_mat2D_to_point(intersection_p);
01698 
01699     mat2D_free(intersection_p);
01700 
01701     return ans_p;
01702 }
01703 
01720 int ae_line_clip_with_plane(Point start_in, Point end_in, Mat2D plane_p, Mat2D plane_n, Point
       *start_out, Point *end_out)
01721 {
01722     ae_assert_point_is_valid(start_in);
01723     ae_assert_point_is_valid(end_in);
01724 
01725     mat2D_normalize(plane_n);
01726 
01727     /* if the signed distance is positive, the point lies on the "inside" of the plane */
01728     Point inside_points[2];
01729     Point outside_points[2];
01730     int inside_points_count = 0;
01731     int outside_points_count = 0;
01732 
01733     /* calc signed distance of each point of tri_in */
01734     float d0 = ae_signed_dist_point_and_plane(start_in, plane_p, plane_n);
01735     float d1 = ae_signed_dist_point_and_plane(end_in, plane_p, plane_n);
01736     float t;
01737 
01738     // float epsilon = 1e-3;
01739     float epsilon = 0;
01740     if (d0 >= epsilon) {
01741         inside_points[inside_points_count++] = start_in;
01742     } else {
01743         outside_points[outside_points_count++] = start_in;
01744     }
01745     if (d1 >= epsilon) {
01746         inside_points[inside_points_count++] = end_in;
01747     } else {
01748         outside_points[outside_points_count++] = end_in;
01749     }
01750 
01751     /* classifying the triangle points */
01752     if (outside_points_count == 2) {
01753         return 0;
01754     } else if (inside_points_count == 2) {
01755         *start_out = start_in;
01756         *end_out = end_in;
01757         return 1;
01758     } else if (d0 >= epsilon && d1 < epsilon) {
01759         Mat2D line_start = mat2D_alloc(3, 1);
01760         Mat2D line_end   = mat2D_alloc(3, 1);
01761 
01762         *start_out = inside_points[0];
01763 
01764         ae_point_to_mat2D(inside_points[0], line_start);
01765         ae_point_to_mat2D(outside_points[0], line_end);
01766         *end_out = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
01767 
01768         mat2D_free(line_start);
01769         mat2D_free(line_end);
01770 
01771         ae_assert_point_is_valid(*start_out);
01772         ae_assert_point_is_valid(*end_out);
01773 
01774         return 1;
01775     } else if (d1 >= epsilon && d0 < epsilon) {
01776         Mat2D line_start = mat2D_alloc(3, 1);
01777         Mat2D line_end   = mat2D_alloc(3, 1);
01778 
01779         *end_out = inside_points[0];
01780 
01781         ae_point_to_mat2D(inside_points[0], line_start);
01782         ae_point_to_mat2D(outside_points[0], line_end);
01783         *start_out = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
01784 
01785         mat2D_free(line_start);
01786         mat2D_free(line_end);
01787 
01788         ae_assert_point_is_valid(*start_out);
01789         ae_assert_point_is_valid(*end_out);
01790 
```

```
01791         return 1;
01792     }
01793     return -1;
01794 }
01795
01807 float ae_signed_dist_point_and_plane(Point p, Mat2D plane_p, Mat2D plane_n)
01808 {
01809     ae_assert_point_is_valid(p);
01810
01811     // mat2D_normalize(plane_n);
01812     // Mat2D p_mat2D = mat2D_alloc(3, 1);
01813     // ae_point_to_mat2D(p, p_mat2D);
01814
01815     // float res = mat2D_dot_product(plane_n, p_mat2D) - mat2D_dot_product(plane_n, plane_p);
01816
01817     float res = MAT2D_AT(plane_n, 0, 0) * p.x + MAT2D_AT(plane_n, 1, 0) * p.y + MAT2D_AT(plane_n, 2,
       0) * p.z - (MAT2D_AT(plane_n, 0, 0) * MAT2D_AT(plane_p, 0, 0) + MAT2D_AT(plane_n, 1, 0) *
      MAT2D_AT(plane_p, 1, 0) + MAT2D_AT(plane_n, 2, 0) * MAT2D_AT(plane_p, 2, 0));
01818
01819     // mat2D_free(p_mat2D);
01820
01821     return res;
01822 }
01823
01838 int ae_tri_clip_with_plane(Tri tri_in, Mat2D plane_p, Mat2D plane_n, Tri *tri_out1, Tri *tri_out2)
01839 {
01840     ae_assert_tri_is_valid(tri_in);
01841
01842     mat2D_normalize(plane_n);
01843
01844     /* if the signed distance is positive, the point lies on the "inside" of the plane */
01845     Point inside_points[3];
01846     Point outside_points[3];
01847     int inside_points_count = 0;
01848     int outside_points_count = 0;
01849     Point tex_inside_points[3];
01850     Point tex_outside_points[3];
01851     int tex_inside_points_count = 0;
01852     int tex_outside_points_count = 0;
01853
01854     /* calc signed distance of each point of tri_in */
01855     float d0 = ae_signed_dist_point_and_plane(tri_in.points[0], plane_p, plane_n);
01856     float d1 = ae_signed_dist_point_and_plane(tri_in.points[1], plane_p, plane_n);
01857     float d2 = ae_signed_dist_point_and_plane(tri_in.points[2], plane_p, plane_n);
01858     float t;
01859
01860     // float epsilon = 1e-3;
01861     float epsilon = 0;
01862     if (d0 >= epsilon) {
01863         inside_points[inside_points_count++] = tri_in.points[0];
01864         tex_inside_points[tex_inside_points_count++] = tri_in.tex_points[0];
01865     } else {
01866         outside_points[outside_points_count++] = tri_in.points[0];
01867         tex_outside_points[tex_outside_points_count++] = tri_in.tex_points[0];
01868     }
01869     if (d1 >= epsilon) {
01870         inside_points[inside_points_count++] = tri_in.points[1];
01871         tex_inside_points[tex_inside_points_count++] = tri_in.tex_points[1];
01872     } else {
01873         outside_points[outside_points_count++] = tri_in.points[1];
01874         tex_outside_points[tex_outside_points_count++] = tri_in.tex_points[1];
01875     }
01876     if (d2 >= epsilon) {
01877         inside_points[inside_points_count++] = tri_in.points[2];
01878         tex_inside_points[tex_inside_points_count++] = tri_in.tex_points[2];
01879     } else {
01880         outside_points[outside_points_count++] = tri_in.points[2];
01881         tex_outside_points[tex_outside_points_count++] = tri_in.tex_points[2];
01882     }
01883
01884     /* classifying the triangle points */
01885     if (inside_points_count == 0) {
01886         return 0;
01887     } else if (inside_points_count == 3) {
01888         *tri_out1 = tri_in;
01889         return 1;
01890     } else if (inside_points_count == 1 && outside_points_count == 2 && d2 >= epsilon) {
01891         Mat2D line_start = mat2D_alloc(3, 1);
01892         Mat2D line_end   = mat2D_alloc(3, 1);
01893
01894         *tri_out1 = tri_in;
01895         // tri_out1->colors[0] = 0xFF0000;
01896         // tri_out1->colors[1] = 0xFF0000;
01897         // tri_out1->colors[2] = 0xFF0000;
01898
01899         (*tri_out1).points[0] = inside_points[0];
01900         (*tri_out1).tex_points[0] = tex_inside_points[0];
```

```
01901
01902         ae_point_to_mat2D(inside_points[0], line_start);
01903         ae_point_to_mat2D(outside_points[0], line_end);
01904         (*tri_out1).points[1] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
01905         (*tri_out1).points[1].w = t * (outside_points[0].w - inside_points[0].w) + inside_points[0].w;
01906         (*tri_out1).tex_points[1].x = t * (tex_outside_points[0].x - tex_inside_points[0].x) +
      tex_inside_points[0].x;
01907         (*tri_out1).tex_points[1].y = t * (tex_outside_points[0].y - tex_inside_points[0].y) +
      tex_inside_points[0].y;
01908
01909         ae_point_to_mat2D(inside_points[0], line_start);
01910         ae_point_to_mat2D(outside_points[1], line_end);
01911         (*tri_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
01912         (*tri_out1).points[2].w = t * (outside_points[1].w - inside_points[0].w) + inside_points[0].w;
01913         (*tri_out1).tex_points[2].x = t * (tex_outside_points[1].x - tex_inside_points[0].x) +
      tex_inside_points[0].x;
01914         (*tri_out1).tex_points[2].y = t * (tex_outside_points[1].y - tex_inside_points[0].y) +
      tex_inside_points[0].y;
01915
01916         mat2D_free(line_start);
01917         mat2D_free(line_end);
01918
01919         /* fixing color ordering */
01920         uint32_t temp_color = tri_out1->colors[2];
01921         tri_out1->colors[2] = tri_out1->colors[1];
01922         tri_out1->colors[1] = tri_out1->colors[0];
01923         tri_out1->colors[0] = temp_color;
01924
01925         ae_assert_tri_is_valid(*tri_out1);
01926
01927         return 1;
01928     } else if (inside_points_count == 1 && outside_points_count == 2 && d1 >= epsilon) {
01929         Mat2D line_start = mat2D_alloc(3, 1);
01930         Mat2D line_end   = mat2D_alloc(3, 1);
01931
01932         *tri_out1 = tri_in;
01933         // tri_out1->colors[0] = 0xFF0000;
01934         // tri_out1->colors[1] = 0xFF0000;
01935         // tri_out1->colors[2] = 0xFF0000;
01936
01937         (*tri_out1).points[0] = inside_points[0];
01938         (*tri_out1).tex_points[0] = tex_inside_points[0];
01939
01940         ae_point_to_mat2D(inside_points[0], line_start);
01941         ae_point_to_mat2D(outside_points[0], line_end);
01942         (*tri_out1).points[1] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
01943         (*tri_out1).points[1].w = t * (outside_points[0].w - inside_points[0].w) + inside_points[0].w;
01944         (*tri_out1).tex_points[1].x = t * (tex_outside_points[0].x - tex_inside_points[0].x) +
      tex_inside_points[0].x;
01945         (*tri_out1).tex_points[1].y = t * (tex_outside_points[0].y - tex_inside_points[0].y) +
      tex_inside_points[0].y;
01946
01947         ae_point_to_mat2D(inside_points[0], line_start);
01948         ae_point_to_mat2D(outside_points[1], line_end);
01949         (*tri_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
01950         (*tri_out1).points[2].w = t * (outside_points[1].w - inside_points[0].w) + inside_points[0].w;
01951         (*tri_out1).tex_points[2].x = t * (tex_outside_points[1].x - tex_inside_points[0].x) +
      tex_inside_points[0].x;
01952         (*tri_out1).tex_points[2].y = t * (tex_outside_points[1].y - tex_inside_points[0].y) +
      tex_inside_points[0].y;
01953
01954         mat2D_free(line_start);
01955         mat2D_free(line_end);
01956
01957         /* fixing color ordering */
01958         uint32_t temp_color = tri_out1->colors[2];
01959         tri_out1->colors[2] = tri_out1->colors[1];
01960         tri_out1->colors[1] = tri_out1->colors[0];
01961         tri_out1->colors[0] = temp_color;
01962
01963         temp_color = tri_out1->colors[2];
01964         tri_out1->colors[2] = tri_out1->colors[0];
01965         tri_out1->colors[0] = temp_color;
01966
01967         ae_assert_tri_is_valid(*tri_out1);
01968
01969         return 1;
01970     } else if (inside_points_count == 1 && outside_points_count == 2 && d0 >= epsilon) {
01971         Mat2D line_start = mat2D_alloc(3, 1);
01972         Mat2D line_end   = mat2D_alloc(3, 1);
01973
01974         *tri_out1 = tri_in;
01975         // tri_out1->colors[0] = 0xFF0000;
01976         // tri_out1->colors[1] = 0xFF0000;
01977         // tri_out1->colors[2] = 0xFF0000;
01978
01979         (*tri_out1).points[0] = inside_points[0];
```

```
01980         (*tri_out1).tex_points[0] = tex_inside_points[0];
01981
01982         ae_point_to_mat2D(inside_points[0], line_start);
01983         ae_point_to_mat2D(outside_points[0], line_end);
01984         (*tri_out1).points[1] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
01985         (*tri_out1).points[1].w = t * (outside_points[0].w - inside_points[0].w) + inside_points[0].w;
01986         (*tri_out1).tex_points[1].x = t * (tex_outside_points[0].x - tex_inside_points[0].x) +
      tex_inside_points[0].x;
01987         (*tri_out1).tex_points[1].y = t * (tex_outside_points[0].y - tex_inside_points[0].y) +
      tex_inside_points[0].y;
01988
01989         ae_point_to_mat2D(inside_points[0], line_start);
01990         ae_point_to_mat2D(outside_points[1], line_end);
01991         (*tri_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
01992         (*tri_out1).points[2].w = t * (outside_points[1].w - inside_points[0].w) + inside_points[0].w;
01993         (*tri_out1).tex_points[2].x = t * (tex_outside_points[1].x - tex_inside_points[0].x) +
      tex_inside_points[0].x;
01994         (*tri_out1).tex_points[2].y = t * (tex_outside_points[1].y - tex_inside_points[0].y) +
      tex_inside_points[0].y;
01995
01996         mat2D_free(line_start);
01997         mat2D_free(line_end);
01998
01999         ae_assert_tri_is_valid(*tri_out1);
02000
02001         return 1;
02002     } else if (inside_points_count == 2 && outside_points_count == 1 && d2 < epsilon) {
02003         Mat2D line_start = mat2D_alloc(3, 1);
02004         Mat2D line_end   = mat2D_alloc(3, 1);
02005
02006         *tri_out1 = tri_in;
02007         // tri_out1->colors[0] = 0x00FF00;
02008         // tri_out1->colors[1] = 0x00FF00;
02009         // tri_out1->colors[2] = 0x00FF00;
02010
02011         *tri_out2 = tri_in;
02012         // tri_out2->colors[0] = 0x0000FF;
02013         // tri_out2->colors[1] = 0x0000FF;
02014         // tri_out2->colors[2] = 0x0000FF;
02015
02016         (*tri_out1).points[0] = inside_points[0];
02017         (*tri_out1).tex_points[0] = tex_inside_points[0];
02018         (*tri_out1).points[1] = inside_points[1];
02019         (*tri_out1).tex_points[1] = tex_inside_points[1];
02020         ae_point_to_mat2D(inside_points[0], line_start);
02021         ae_point_to_mat2D(outside_points[0], line_end);
02022         (*tri_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02023         (*tri_out1).points[2].w = t * (outside_points[0].w - inside_points[0].w) + inside_points[0].w;
02024         (*tri_out1).tex_points[2].x = t * (tex_outside_points[0].x - tex_inside_points[0].x) +
      tex_inside_points[0].x;
02025         (*tri_out1).tex_points[2].y = t * (tex_outside_points[0].y - tex_inside_points[0].y) +
      tex_inside_points[0].y;
02026
02027         (*tri_out2).points[0] = inside_points[1];
02028         (*tri_out2).tex_points[0] = tex_inside_points[1];
02029         ae_point_to_mat2D(inside_points[1], line_start);
02030         ae_point_to_mat2D(outside_points[0], line_end);
02031         (*tri_out2).points[1] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02032         (*tri_out2).points[1].w = t * (outside_points[0].w - inside_points[1].w) + inside_points[1].w;
02033         (*tri_out2).tex_points[1].x = t * (tex_outside_points[0].x - tex_inside_points[1].x) +
      tex_inside_points[1].x;
02034         (*tri_out2).tex_points[1].y = t * (tex_outside_points[0].y - tex_inside_points[1].y) +
      tex_inside_points[1].y;
02035         (*tri_out2).points[2] = (*tri_out1).points[2];
02036         (*tri_out2).tex_points[2] = (*tri_out1).tex_points[2];
02037
02038         mat2D_free(line_start);
02039         mat2D_free(line_end);
02040
02041         /* fixing color ordering */
02042         uint32_t temp_color = tri_out2->colors[2];
02043         tri_out2->colors[2] = tri_out2->colors[0];
02044         tri_out2->colors[0] = tri_out2->colors[1];
02045         tri_out2->colors[1] = temp_color;
02046
02047         ae_assert_tri_is_valid(*tri_out1);
02048         ae_assert_tri_is_valid(*tri_out2);
02049
02050         return 2;
02051     } else if (inside_points_count == 2 && outside_points_count == 1 && d1 < epsilon) {
02052         Mat2D line_start = mat2D_alloc(3, 1);
02053         Mat2D line_end   = mat2D_alloc(3, 1);
02054
02055         *tri_out1 = tri_in;
02056         // tri_out1->colors[0] = 0x00FF00;
02057         // tri_out1->colors[1] = 0x00FF00;
02058         // tri_out1->colors[2] = 0x00FF00;
```

```
02059
02060         *tri_out2 = tri_in;
02061         // tri_out2->colors[0] = 0x0000FF;
02062         // tri_out2->colors[1] = 0x0000FF;
02063         // tri_out2->colors[2] = 0x0000FF;
02064
02065         (*tri_out1).points[0] = inside_points[0];
02066         (*tri_out1).tex_points[0] = tex_inside_points[0];
02067         (*tri_out1).points[1] = inside_points[1];
02068         (*tri_out1).tex_points[1] = tex_inside_points[1];
02069         ae_point_to_mat2D(inside_points[0], line_start);
02070         ae_point_to_mat2D(outside_points[0], line_end);
02071         (*tri_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02072         (*tri_out1).points[2].w = t * (outside_points[0].w - inside_points[0].w) + inside_points[0].w;
02073         (*tri_out1).tex_points[2].x = t * (tex_outside_points[0].x - tex_inside_points[0].x) +
      tex_inside_points[0].x;
02074         (*tri_out1).tex_points[2].y = t * (tex_outside_points[0].y - tex_inside_points[0].y) +
      tex_inside_points[0].y;
02075
02076         (*tri_out2).points[0] = inside_points[1];
02077         (*tri_out2).tex_points[0] = tex_inside_points[1];
02078         ae_point_to_mat2D(inside_points[1], line_start);
02079         ae_point_to_mat2D(outside_points[0], line_end);
02080         (*tri_out2).points[1] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02081         (*tri_out2).points[1].w = t * (outside_points[0].w - inside_points[1].w) + inside_points[1].w;
02082         (*tri_out2).tex_points[1].x = t * (tex_outside_points[0].x - tex_inside_points[1].x) +
      tex_inside_points[1].x;
02083         (*tri_out2).tex_points[1].y = t * (tex_outside_points[0].y - tex_inside_points[1].y) +
      tex_inside_points[1].y;
02084         (*tri_out2).points[2] = (*tri_out1).points[2];
02085         (*tri_out2).tex_points[2] = (*tri_out1).tex_points[2];
02086
02087         mat2D_free(line_start);
02088         mat2D_free(line_end);
02089
02090         /* fixing color ordering */
02091         uint32_t temp_color = tri_out1->colors[2];
02092         tri_out1->colors[2] = tri_out1->colors[1];
02093         tri_out1->colors[1] = temp_color;
02094
02095         temp_color = tri_out2->colors[2];
02096         tri_out2->colors[2] = tri_out2->colors[0];
02097         tri_out2->colors[0] = tri_out2->colors[1];
02098         tri_out2->colors[1] = temp_color;
02099         temp_color = tri_out2->colors[1];
02100         tri_out2->colors[1] = tri_out2->colors[0];
02101         tri_out2->colors[0] = temp_color;
02102
02103         ae_assert_tri_is_valid(*tri_out1);
02104         ae_assert_tri_is_valid(*tri_out2);
02105
02106         return 2;
02107     } else if (inside_points_count == 2 && outside_points_count == 1 && d0 < epsilon) {
02108         Mat2D line_start = mat2D_alloc(3, 1);
02109         Mat2D line_end   = mat2D_alloc(3, 1);
02110
02111         *tri_out1 = tri_in;
02112         // tri_out1->colors[0] = 0x00FF00;
02113         // tri_out1->colors[1] = 0x00FF00;
02114         // tri_out1->colors[2] = 0x00FF00;
02115
02116         *tri_out2 = tri_in;
02117         // tri_out2->colors[0] = 0x0000FF;
02118         // tri_out2->colors[1] = 0x0000FF;
02119         // tri_out2->colors[2] = 0x0000FF;
02120
02121         (*tri_out1).points[0] = inside_points[0];
02122         (*tri_out1).tex_points[0] = tex_inside_points[0];
02123         (*tri_out1).points[1] = inside_points[1];
02124         (*tri_out1).tex_points[1] = tex_inside_points[1];
02125         ae_point_to_mat2D(inside_points[0], line_start);
02126         ae_point_to_mat2D(outside_points[0], line_end);
02127         (*tri_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02128         (*tri_out1).points[2].w = t * (outside_points[0].w - inside_points[0].w) + inside_points[0].w;
02129         (*tri_out1).tex_points[2].x = t * (tex_outside_points[0].x - tex_inside_points[0].x) +
      tex_inside_points[0].x;
02130         (*tri_out1).tex_points[2].y = t * (tex_outside_points[0].y - tex_inside_points[0].y) +
      tex_inside_points[0].y;
02131
02132         (*tri_out2).points[0] = inside_points[1];
02133         (*tri_out2).tex_points[0] = tex_inside_points[1];
02134         ae_point_to_mat2D(inside_points[1], line_start);
02135         ae_point_to_mat2D(outside_points[0], line_end);
02136         (*tri_out2).points[1] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02137         (*tri_out2).points[1].w = t * (outside_points[0].w - inside_points[1].w) + inside_points[1].w;
02138         (*tri_out2).tex_points[1].x = t * (tex_outside_points[0].x - tex_inside_points[1].x) +
      tex_inside_points[1].x;
```

```
02139         (*tri_out2).tex_points[1].y = t * (tex_outside_points[0].y - tex_inside_points[1].y) +
      tex_inside_points[1].y;
02140         (*tri_out2).points[2] = (*tri_out1).points[2];
02141         (*tri_out2).tex_points[2] = (*tri_out1).tex_points[2];
02142 
02143         mat2D_free(line_start);
02144         mat2D_free(line_end);
02145 
02146         /* fixing color ordering */
02147         uint32_t temp_color = tri_out1->colors[2];
02148         tri_out1->colors[2] = tri_out1->colors[0];
02149         tri_out1->colors[0] = tri_out1->colors[1];
02150         tri_out1->colors[1] = temp_color;
02151 
02152         temp_color = tri_out2->colors[2];
02153         tri_out2->colors[2] = tri_out2->colors[1];
02154         tri_out2->colors[1] = tri_out2->colors[0];
02155         tri_out2->colors[0] = temp_color;
02156 
02157         ae_assert_tri_is_valid(*tri_out1);
02158         ae_assert_tri_is_valid(*tri_out2);
02159 
02160         return 2;
02161     }
02162     return -1;
02163 }
02164 
02181 int ae_quad_clip_with_plane(Quad quad_in, Mat2D plane_p, Mat2D plane_n, Quad *quad_out1, Quad
      *quad_out2)
02182 {
02183     ae_assert_quad_is_valid(quad_in);
02184 
02185     mat2D_normalize(plane_n);
02186 
02187     /* if the signed distance is positive, the point lies on the "inside" of the plane */
02188     Point inside_points[4];
02189     Point outside_points[4];
02190     int inside_points_count = 0;
02191     int outside_points_count = 0;
02192 
02193     /* calc signed distance of each point of tri_in */
02194     float d0 = ae_signed_dist_point_and_plane(quad_in.points[0], plane_p, plane_n);
02195     float d1 = ae_signed_dist_point_and_plane(quad_in.points[1], plane_p, plane_n);
02196     float d2 = ae_signed_dist_point_and_plane(quad_in.points[2], plane_p, plane_n);
02197     float d3 = ae_signed_dist_point_and_plane(quad_in.points[3], plane_p, plane_n);
02198     float t;
02199 
02200     // float epsilon = 1e-3;
02201     float epsilon = 0;
02202     if (d0 >= epsilon) {
02203         inside_points[inside_points_count++] = quad_in.points[0];
02204     } else {
02205         outside_points[outside_points_count++] = quad_in.points[0];
02206     }
02207     if (d1 >= epsilon) {
02208         inside_points[inside_points_count++] = quad_in.points[1];
02209     } else {
02210         outside_points[outside_points_count++] = quad_in.points[1];
02211     }
02212     if (d2 >= epsilon) {
02213         inside_points[inside_points_count++] = quad_in.points[2];
02214     } else {
02215         outside_points[outside_points_count++] = quad_in.points[2];
02216     }
02217     if (d3 >= epsilon) {
02218         inside_points[inside_points_count++] = quad_in.points[3];
02219     } else {
02220         outside_points[outside_points_count++] = quad_in.points[3];
02221     }
02222 
02223     /* classifying the triangle points */
02224     if (inside_points_count == 0) {
02225         return 0;
02226     } else if (inside_points_count == 4) {
02227         *quad_out1 = quad_in;
02228         return 1;
02229     } else if (inside_points_count == 1 && outside_points_count == 3 && d1 >= epsilon) {
02230         Mat2D line_start = mat2D_alloc(3, 1);
02231         Mat2D line_end   = mat2D_alloc(3, 1);
02232 
02233         *quad_out1 = quad_in;
02234         *quad_out2 = quad_in;
02235 
02236         (*quad_out1).points[1] = quad_in.points[1];
02237 
02238         ae_point_to_mat2D(quad_in.points[1], line_start);
02239         ae_point_to_mat2D(quad_in.points[2], line_end);
```

```
02240         (*quad_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02241         (*quad_out1).points[2].w = t * (quad_in.points[2].w - quad_in.points[1].w) +
      quad_in.points[1].w;
02242         (*quad_out1).colors[2] = quad_in.colors[2];
02243 
02244         ae_point_to_mat2D(quad_in.points[1], line_start);
02245         ae_point_to_mat2D(quad_in.points[0], line_end);
02246         (*quad_out1).points[0] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02247         (*quad_out1).points[0].w = t * (quad_in.points[0].w - quad_in.points[1].w) +
      quad_in.points[1].w;
02248         (*quad_out1).colors[0] = quad_in.colors[0];
02249 
02250         (*quad_out1).points[3].x = ((*quad_out1).points[0].x + (*quad_out1).points[2].x) / 2;
02251         (*quad_out1).points[3].y = ((*quad_out1).points[0].y + (*quad_out1).points[2].y) / 2;
02252         (*quad_out1).points[3].z = ((*quad_out1).points[0].z + (*quad_out1).points[2].z) / 2;
02253         (*quad_out1).points[3].w = ((*quad_out1).points[0].w + (*quad_out1).points[2].w) / 2;
02254         (*quad_out1).colors[3] = quad_in.colors[3];
02255 
02256         mat2D_free(line_start);
02257         mat2D_free(line_end);
02258 
02259         ae_assert_quad_is_valid(*quad_out1);
02260 
02261         return 1;
02262     } else if (inside_points_count == 1 && outside_points_count == 3 && d2 >= epsilon) {
02263         Mat2D line_start = mat2D_alloc(3, 1);
02264         Mat2D line_end   = mat2D_alloc(3, 1);
02265 
02266         *quad_out1 = quad_in;
02267         *quad_out2 = quad_in;
02268 
02269         (*quad_out1).points[2] = quad_in.points[2];
02270 
02271         ae_point_to_mat2D(quad_in.points[2], line_start);
02272         ae_point_to_mat2D(quad_in.points[3], line_end);
02273         (*quad_out1).points[3] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02274         (*quad_out1).points[3].w = t * (quad_in.points[3].w - quad_in.points[2].w) +
      quad_in.points[2].w;
02275         (*quad_out1).colors[3] = quad_in.colors[3];
02276 
02277         ae_point_to_mat2D(quad_in.points[2], line_start);
02278         ae_point_to_mat2D(quad_in.points[1], line_end);
02279         (*quad_out1).points[1] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02280         (*quad_out1).points[1].w = t * (quad_in.points[1].w - quad_in.points[2].w) +
      quad_in.points[2].w;
02281         (*quad_out1).colors[1] = quad_in.colors[1];
02282 
02283         (*quad_out1).points[0].x = ((*quad_out1).points[3].x + (*quad_out1).points[1].x) / 2;
02284         (*quad_out1).points[0].y = ((*quad_out1).points[3].y + (*quad_out1).points[1].y) / 2;
02285         (*quad_out1).points[0].z = ((*quad_out1).points[3].z + (*quad_out1).points[1].z) / 2;
02286         (*quad_out1).points[0].w = ((*quad_out1).points[3].w + (*quad_out1).points[1].w) / 2;
02287         (*quad_out1).colors[0] = quad_in.colors[0];
02288 
02289         mat2D_free(line_start);
02290         mat2D_free(line_end);
02291 
02292         ae_assert_quad_is_valid(*quad_out1);
02293 
02294         return 1;
02295     } else if (inside_points_count == 1 && outside_points_count == 3 && d3 >= epsilon) {
02296         Mat2D line_start = mat2D_alloc(3, 1);
02297         Mat2D line_end   = mat2D_alloc(3, 1);
02298 
02299         *quad_out1 = quad_in;
02300         *quad_out2 = quad_in;
02301 
02302         (*quad_out1).points[3] = quad_in.points[3];
02303 
02304         ae_point_to_mat2D(quad_in.points[3], line_start);
02305         ae_point_to_mat2D(quad_in.points[0], line_end);
02306         (*quad_out1).points[0] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02307         (*quad_out1).points[0].w = t * (quad_in.points[0].w - quad_in.points[3].w) +
      quad_in.points[3].w;
02308         (*quad_out1).colors[0] = quad_in.colors[0];
02309 
02310         ae_point_to_mat2D(quad_in.points[3], line_start);
02311         ae_point_to_mat2D(quad_in.points[2], line_end);
02312         (*quad_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02313         (*quad_out1).points[2].w = t * (quad_in.points[2].w - quad_in.points[3].w) +
      quad_in.points[3].w;
02314         (*quad_out1).colors[2] = quad_in.colors[2];
02315 
02316         (*quad_out1).points[1].x = ((*quad_out1).points[2].x + (*quad_out1).points[0].x) / 2;
02317         (*quad_out1).points[1].y = ((*quad_out1).points[2].y + (*quad_out1).points[0].y) / 2;
02318         (*quad_out1).points[1].z = ((*quad_out1).points[2].z + (*quad_out1).points[0].z) / 2;
02319         (*quad_out1).points[1].w = ((*quad_out1).points[2].w + (*quad_out1).points[0].w) / 2;
02320         (*quad_out1).colors[1] = quad_in.colors[1];
```

```
02321
02322         mat2D_free(line_start);
02323         mat2D_free(line_end);
02324
02325         ae_assert_quad_is_valid(*quad_out1);
02326
02327         return 1;
02328     } else if (inside_points_count == 1 && outside_points_count == 3) {
02329         Mat2D line_start = mat2D_alloc(3, 1);
02330         Mat2D line_end   = mat2D_alloc(3, 1);
02331
02332         *quad_out1 = quad_in;
02333         *quad_out2 = quad_in;
02334
02335         (*quad_out1).points[0] = inside_points[0];
02336
02337         ae_point_to_mat2D(inside_points[0], line_start);
02338         ae_point_to_mat2D(outside_points[0], line_end);
02339         (*quad_out1).points[1] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02340         (*quad_out1).points[1].w = t * (outside_points[0].w - inside_points[0].w) +
      inside_points[0].w;
02341
02342         ae_point_to_mat2D(inside_points[0], line_start);
02343         ae_point_to_mat2D(outside_points[1], line_end);
02344         (*quad_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02345         (*quad_out1).points[2].w = t * (outside_points[1].w - inside_points[0].w) +
      inside_points[0].w;
02346
02347         ae_point_to_mat2D(inside_points[0], line_start);
02348         ae_point_to_mat2D(outside_points[2], line_end);
02349         (*quad_out1).points[3] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02350         (*quad_out1).points[3].w = t * (outside_points[2].w - inside_points[0].w) +
      inside_points[0].w;
02351
02352         mat2D_free(line_start);
02353         mat2D_free(line_end);
02354
02355         ae_assert_quad_is_valid(*quad_out1);
02356
02357         return 1;
02358     } else if (inside_points_count == 2 && outside_points_count == 2 && d2 < epsilon && d1 < epsilon)
      {
02359         Mat2D line_start = mat2D_alloc(3, 1);
02360         Mat2D line_end   = mat2D_alloc(3, 1);
02361
02362         *quad_out1 = quad_in;
02363
02364         (*quad_out1).points[0] = quad_in.points[3];
02365         (*quad_out1).colors[0] = quad_in.colors[3];
02366         (*quad_out1).points[1] = quad_in.points[0];
02367         (*quad_out1).colors[1] = quad_in.colors[0];
02368
02369         ae_point_to_mat2D(quad_in.points[0], line_start);
02370         ae_point_to_mat2D(quad_in.points[1], line_end);
02371         (*quad_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02372         (*quad_out1).points[2].w = t * (quad_in.points[1].w - quad_in.points[0].w) +
      quad_in.points[0].w;
02373         (*quad_out1).colors[2] = quad_in.colors[1];
02374
02375         ae_point_to_mat2D(quad_in.points[3], line_start);
02376         ae_point_to_mat2D(quad_in.points[2], line_end);
02377         (*quad_out1).points[3] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02378         (*quad_out1).points[3].w = t * (quad_in.points[2].w - quad_in.points[3].w) +
      quad_in.points[3].w;
02379         (*quad_out1).colors[3] = quad_in.colors[2];
02380
02381         mat2D_free(line_start);
02382         mat2D_free(line_end);
02383
02384         ae_assert_quad_is_valid(*quad_out1);
02385
02386         return 2;
02387     } else if (inside_points_count == 2 && outside_points_count == 2 && d0 < epsilon && d1 < epsilon)
      {
02388         Mat2D line_start = mat2D_alloc(3, 1);
02389         Mat2D line_end   = mat2D_alloc(3, 1);
02390
02391         *quad_out1 = quad_in;
02392
02393         (*quad_out1).points[0] = quad_in.points[2];
02394         (*quad_out1).colors[0] = quad_in.colors[2];
02395         (*quad_out1).points[1] = quad_in.points[3];
02396         (*quad_out1).colors[1] = quad_in.colors[3];
02397
02398         ae_point_to_mat2D(quad_in.points[2], line_start);
02399         ae_point_to_mat2D(quad_in.points[1], line_end);
02400         (*quad_out1).points[3] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
```

```
02401         (*quad_out1).points[3].w = t * (quad_in.points[1].w - quad_in.points[2].w) +
      quad_in.points[2].w;
02402         (*quad_out1).colors[3] = quad_in.colors[1];
02403 
02404         ae_point_to_mat2D(quad_in.points[3], line_start);
02405         ae_point_to_mat2D(quad_in.points[0], line_end);
02406         (*quad_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02407         (*quad_out1).points[2].w = t * (quad_in.points[0].w - quad_in.points[3].w) +
      quad_in.points[3].w;
02408         (*quad_out1).colors[2] = quad_in.colors[0];
02409 
02410         mat2D_free(line_start);
02411         mat2D_free(line_end);
02412 
02413         ae_assert_quad_is_valid(*quad_out1);
02414 
02415         return 2;
02416     } else if (inside_points_count == 2 && outside_points_count == 2 && d0 < epsilon && d3 < epsilon)
      {
02417         Mat2D line_start = mat2D_alloc(3, 1);
02418         Mat2D line_end   = mat2D_alloc(3, 1);
02419 
02420         *quad_out1 = quad_in;
02421 
02422         (*quad_out1).points[0] = quad_in.points[1];
02423         (*quad_out1).colors[0] = quad_in.colors[1];
02424         (*quad_out1).points[1] = quad_in.points[2];
02425         (*quad_out1).colors[1] = quad_in.colors[2];
02426 
02427         ae_point_to_mat2D(quad_in.points[2], line_start);
02428         ae_point_to_mat2D(quad_in.points[3], line_end);
02429         (*quad_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02430         (*quad_out1).points[2].w = t * (quad_in.points[3].w - quad_in.points[2].w) +
      quad_in.points[2].w;
02431         (*quad_out1).colors[2] = quad_in.colors[3];
02432 
02433         ae_point_to_mat2D(quad_in.points[1], line_start);
02434         ae_point_to_mat2D(quad_in.points[0], line_end);
02435         (*quad_out1).points[3] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02436         (*quad_out1).points[3].w = t * (quad_in.points[1].w - quad_in.points[3].w) +
      quad_in.points[3].w;
02437         (*quad_out1).colors[3] = quad_in.colors[0];
02438 
02439         mat2D_free(line_start);
02440         mat2D_free(line_end);
02441 
02442         ae_assert_quad_is_valid(*quad_out1);
02443 
02444         return 1;
02445     } else if (inside_points_count == 2 && outside_points_count == 2) {
02446         Mat2D line_start = mat2D_alloc(3, 1);
02447         Mat2D line_end   = mat2D_alloc(3, 1);
02448 
02449         *quad_out1 = quad_in;
02450 
02451         (*quad_out1).points[0] = inside_points[0];
02452         (*quad_out1).points[1] = inside_points[1];
02453         ae_point_to_mat2D(inside_points[1], line_start);
02454         ae_point_to_mat2D(outside_points[0], line_end);
02455         (*quad_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02456         (*quad_out1).points[2].w = t * (outside_points[0].w - inside_points[1].w) +
      inside_points[1].w;
02457 
02458         ae_point_to_mat2D(inside_points[0], line_start);
02459         ae_point_to_mat2D(outside_points[1], line_end);
02460         (*quad_out1).points[3] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02461         (*quad_out1).points[3].w = t * (outside_points[1].w - inside_points[0].w) +
      inside_points[0].w;
02462 
02463         mat2D_free(line_start);
02464         mat2D_free(line_end);
02465 
02466         ae_assert_quad_is_valid(*quad_out1);
02467 
02468         return 1;
02469     } else if (inside_points_count == 3 && outside_points_count == 1 && d0 < epsilon) {
02470         Mat2D line_start = mat2D_alloc(3, 1);
02471         Mat2D line_end   = mat2D_alloc(3, 1);
02472 
02473         *quad_out1 = quad_in;
02474         *quad_out2 = quad_in;
02475 
02476         ae_point_to_mat2D(quad_in.points[3], line_start);
02477         ae_point_to_mat2D(quad_in.points[0], line_end);
02478         (*quad_out1).points[0] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02479         (*quad_out1).points[0].w = t * (quad_in.points[0].w - quad_in.points[3].w) +
      quad_in.points[3].w;
```

```
02480         // adl_interpolate_ARGBcolor_on_RGB((quad_in).colors[3], (quad_in).colors[0], t,
      &((*quad_out1).colors[0]));
02481
02482         (*quad_out2).points[1] = quad_in.points[1];
02483         // (*quad_out2).colors[1] = quad_in.colors[1];
02484
02485         ae_point_to_mat2D(quad_in.points[1], line_start);
02486         ae_point_to_mat2D(quad_in.points[0], line_end);
02487         (*quad_out2).points[0] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02488         (*quad_out2).points[0].w = t * (quad_in.points[0].w - quad_in.points[1].w) +
      quad_in.points[1].w;
02489         // adl_interpolate_ARGBcolor_on_RGB((quad_in).colors[1], (quad_in).colors[0], t,
      &((*quad_out2).colors[0]));
02490
02491         (*quad_out2).points[2] = (*quad_out1).points[0];
02492         // (*quad_out2).colors[2] = (*quad_out1).colors[0];
02493
02494         (*quad_out2).points[3].x = ((*quad_out2).points[2].x + (*quad_out2).points[0].x) / 2;
02495         (*quad_out2).points[3].y = ((*quad_out2).points[2].y + (*quad_out2).points[0].y) / 2;
02496         (*quad_out2).points[3].z = ((*quad_out2).points[2].z + (*quad_out2).points[0].z) / 2;
02497         (*quad_out2).points[3].w = ((*quad_out2).points[2].w + (*quad_out2).points[0].w) / 2;
02498         // adl_interpolate_ARGBcolor_on_RGB((*quad_out2).colors[2], (*quad_out2).colors[0], 0.5f,
      &((*quad_out2).colors[3]));
02499
02500
02501         mat2D_free(line_start);
02502         mat2D_free(line_end);
02503
02504         ae_assert_quad_is_valid(*quad_out1);
02505
02506         return 2;
02507     } else if (inside_points_count == 3 && outside_points_count == 1 && d1 < epsilon) {
02508         Mat2D line_start = mat2D_alloc(3, 1);
02509         Mat2D line_end   = mat2D_alloc(3, 1);
02510
02511         *quad_out1 = quad_in;
02512         *quad_out2 = quad_in;
02513
02514         (*quad_out1).points[0] = quad_in.points[0];
02515         (*quad_out1).colors[0] = quad_in.colors[0];
02516         (*quad_out1).points[2] = quad_in.points[2];
02517         (*quad_out1).colors[2] = quad_in.colors[2];
02518         (*quad_out1).points[3] = quad_in.points[3];
02519         (*quad_out1).colors[3] = quad_in.colors[3];
02520
02521         ae_point_to_mat2D(quad_in.points[2], line_start);
02522         ae_point_to_mat2D(quad_in.points[1], line_end);
02523         (*quad_out1).points[1] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02524         (*quad_out1).points[1].w = t * (quad_in.points[1].w - quad_in.points[2].w) +
      quad_in.points[2].w;
02525         (*quad_out1).colors[1] = (*quad_out1).colors[1];
02526         // adl_interpolate_ARGBcolor_on_RGB((quad_in).colors[2], (quad_in).colors[1], t,
      &((*quad_out1).colors[1]));
02527
02528         (*quad_out2).points[0] = quad_in.points[0];
02529         (*quad_out2).colors[0] = quad_in.colors[0];
02530         (*quad_out2).points[3] = (*quad_out1).points[1];
02531         (*quad_out2).colors[3] = (*quad_out1).colors[3];
02532
02533         ae_point_to_mat2D(quad_in.points[0], line_start);
02534         ae_point_to_mat2D(quad_in.points[1], line_end);
02535         (*quad_out2).points[1] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02536         (*quad_out2).points[1].w = t * (quad_in.points[1].w - quad_in.points[0].w) +
      quad_in.points[0].w;
02537         (*quad_out2).colors[1] = quad_in.colors[1];
02538         // adl_interpolate_ARGBcolor_on_RGB((quad_in).colors[0], (quad_in).colors[1], t,
      &((*quad_out2).colors[1]));
02539
02540         (*quad_out2).points[2].x = ((*quad_out2).points[1].x + (*quad_out2).points[3].x) / 2;
02541         (*quad_out2).points[2].y = ((*quad_out2).points[1].y + (*quad_out2).points[3].y) / 2;
02542         (*quad_out2).points[2].z = ((*quad_out2).points[1].z + (*quad_out2).points[3].z) / 2;
02543         (*quad_out2).points[2].w = ((*quad_out2).points[1].w + (*quad_out2).points[3].w) / 2;
02544         // adl_interpolate_ARGBcolor_on_RGB((*quad_out2).colors[1], (*quad_out2).colors[3], 0.5f,
      &((*quad_out2).colors[2]));
02545
02546
02547         mat2D_free(line_start);
02548         mat2D_free(line_end);
02549
02550         ae_assert_quad_is_valid(*quad_out1);
02551
02552         return 2;
02553     } else if (inside_points_count == 3 && outside_points_count == 1 && d2 < epsilon) {
02554         Mat2D line_start = mat2D_alloc(3, 1);
02555         Mat2D line_end   = mat2D_alloc(3, 1);
02556
02557         *quad_out1 = quad_in;
```

```
02558         *quad_out2 = quad_in;
02559
02560         (*quad_out1).points[0] = quad_in.points[0];
02561         (*quad_out1).colors[0] = quad_in.colors[0];
02562         (*quad_out1).points[1] = quad_in.points[1];
02563         (*quad_out1).colors[1] = quad_in.colors[1];
02564         (*quad_out1).points[3] = quad_in.points[3];
02565         (*quad_out1).colors[3] = quad_in.colors[3];
02566
02567         ae_point_to_mat2D(quad_in.points[1], line_start);
02568         ae_point_to_mat2D(quad_in.points[2], line_end);
02569         (*quad_out1).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02570         (*quad_out1).points[2].w = t * (quad_in.points[2].w - quad_in.points[1].w) +
      quad_in.points[1].w;
02571         (*quad_out1).colors[2] = (*quad_out1).colors[2];
02572         // adl_interpolate_ARGBcolor_on_RGB((quad_in).colors[2], (quad_in).colors[1], t,
      &((*quad_out1).colors[1]));
02573
02574         (*quad_out2).points[3] = quad_in.points[3];
02575         (*quad_out2).colors[3] = quad_in.colors[3];
02576         (*quad_out2).points[0] = (*quad_out1).points[2];
02577         (*quad_out2).colors[0] = (*quad_out1).colors[0];
02578
02579         ae_point_to_mat2D(quad_in.points[3], line_start);
02580         ae_point_to_mat2D(quad_in.points[2], line_end);
02581         (*quad_out2).points[2] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02582         (*quad_out2).points[2].w = t * (quad_in.points[2].w - quad_in.points[3].w) +
      quad_in.points[3].w;
02583         (*quad_out2).colors[2] = quad_in.colors[2];
02584         // adl_interpolate_ARGBcolor_on_RGB((quad_in).colors[0], (quad_in).colors[1], t,
      &((*quad_out2).colors[1]));
02585
02586         (*quad_out2).points[1].x = ((*quad_out2).points[2].x + (*quad_out2).points[0].x) / 2;
02587         (*quad_out2).points[1].y = ((*quad_out2).points[2].y + (*quad_out2).points[0].y) / 2;
02588         (*quad_out2).points[1].z = ((*quad_out2).points[2].z + (*quad_out2).points[0].z) / 2;
02589         (*quad_out2).points[1].w = ((*quad_out2).points[2].w + (*quad_out2).points[0].w) / 2;
02590         // adl_interpolate_ARGBcolor_on_RGB((*quad_out2).colors[1], (*quad_out2).colors[3], 0.5f,
      &((*quad_out2).colors[2]));
02591
02592
02593         mat2D_free(line_start);
02594         mat2D_free(line_end);
02595
02596         ae_assert_quad_is_valid(*quad_out1);
02597
02598         return 2;
02599     } else if (inside_points_count == 3 && outside_points_count == 1 && d3 < epsilon) {
02600         Mat2D line_start = mat2D_alloc(3, 1);
02601         Mat2D line_end   = mat2D_alloc(3, 1);
02602
02603         *quad_out1 = quad_in;
02604         *quad_out2 = quad_in;
02605
02606         (*quad_out1).points[0] = quad_in.points[0];
02607         (*quad_out1).colors[0] = quad_in.colors[0];
02608         (*quad_out1).points[1] = quad_in.points[1];
02609         (*quad_out1).colors[1] = quad_in.colors[1];
02610         (*quad_out1).points[2] = quad_in.points[2];
02611         (*quad_out1).colors[2] = quad_in.colors[2];
02612
02613         ae_point_to_mat2D(quad_in.points[0], line_start);
02614         ae_point_to_mat2D(quad_in.points[3], line_end);
02615         (*quad_out1).points[3] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02616         (*quad_out1).points[3].w = t * (quad_in.points[3].w - quad_in.points[0].w) +
      quad_in.points[0].w;
02617         (*quad_out1).colors[3] = (*quad_out1).colors[3];
02618         // adl_interpolate_ARGBcolor_on_RGB((quad_in).colors[2], (quad_in).colors[1], t,
      &((*quad_out1).colors[1]));
02619
02620         (*quad_out2).points[2] = quad_in.points[2];
02621         (*quad_out2).colors[2] = quad_in.colors[2];
02622         (*quad_out2).points[1] = (*quad_out1).points[3];
02623         (*quad_out2).colors[1] = (*quad_out1).colors[1];
02624
02625         ae_point_to_mat2D(quad_in.points[2], line_start);
02626         ae_point_to_mat2D(quad_in.points[3], line_end);
02627         (*quad_out2).points[3] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02628         (*quad_out2).points[3].w = t * (quad_in.points[3].w - quad_in.points[2].w) +
      quad_in.points[2].w;
02629         (*quad_out2).colors[3] = quad_in.colors[3];
02630         // adl_interpolate_ARGBcolor_on_RGB((quad_in).colors[0], (quad_in).colors[1], t,
      &((*quad_out2).colors[1]));
02631
02632         (*quad_out2).points[0].x = ((*quad_out2).points[3].x + (*quad_out2).points[1].x) / 2;
02633         (*quad_out2).points[0].y = ((*quad_out2).points[3].y + (*quad_out2).points[1].y) / 2;
02634         (*quad_out2).points[0].z = ((*quad_out2).points[3].z + (*quad_out2).points[1].z) / 2;
02635         (*quad_out2).points[0].w = ((*quad_out2).points[3].w + (*quad_out2).points[1].w) / 2;
```

```
02636         // adl_interpolate_ARGBcolor_on_RGB((*quad_out2).colors[1], (*quad_out2).colors[3], 0.5f,
      &((*quad_out2).colors[2]));
02637
02638
02639         mat2D_free(line_start);
02640         mat2D_free(line_end);
02641
02642         ae_assert_quad_is_valid(*quad_out1);
02643
02644         return 2;
02645     } else if (inside_points_count == 3 && outside_points_count == 1) {
02646         Mat2D line_start = mat2D_alloc(3, 1);
02647         Mat2D line_end   = mat2D_alloc(3, 1);
02648
02649         *quad_out1 = quad_in;
02650
02651         (*quad_out1).points[0] = inside_points[0];
02652         (*quad_out1).points[1] = inside_points[1];
02653         (*quad_out1).points[2] = inside_points[2];
02654         ae_point_to_mat2D(inside_points[2], line_start);
02655         ae_point_to_mat2D(outside_points[0], line_end);
02656         (*quad_out1).points[3] = ae_line_itersect_plane(plane_p, plane_n, line_start, line_end, &t);
02657         (*quad_out1).points[3].w = t * (outside_points[0].w - inside_points[2].w) +
     inside_points[2].w;
02658
02659         mat2D_free(line_start);
02660         mat2D_free(line_end);
02661
02662         ae_assert_quad_is_valid(*quad_out1);
02663
02664         return 1;
02665     }
02666     return -1;
02667 }
02668
02682 void ae_projection_mat_set(Mat2D proj_mat,float aspect_ratio, float FOV_deg, float z_near, float
      z_far)
02683 {
02684     AE_ASSERT(4 == proj_mat.cols);
02685     AE_ASSERT(4 == proj_mat.rows);
02686     AE_ASSERT(FOV_deg && "FOV needs to bee bigger then zero");
02687
02688     mat2D_fill(proj_mat, 0);
02689
02690     float field_of_view = 1.0f / tanf(0.5f * FOV_deg * PI / 180);
02691     float z_normalization = z_far / (z_far - z_near);
02692
02693     MAT2D_AT(proj_mat, 0, 0) = aspect_ratio * field_of_view;
02694     MAT2D_AT(proj_mat, 1, 1) = field_of_view;
02695     MAT2D_AT(proj_mat, 2, 2) = z_normalization;
02696     MAT2D_AT(proj_mat, 2, 3) = 1;
02697     MAT2D_AT(proj_mat, 3, 2) = - z_normalization * z_near;
02698 }
02699
02716 void ae_view_mat_set(Mat2D view_mat, Camera camera, Mat2D up)
02717 {
02718     Mat2D DCM = mat2D_alloc(3,3);
02719     Mat2D DCM_trans = mat2D_alloc(3,3);
02720     mat2D_set_DCM_zyx(DCM, camera.yaw_offset_deg, camera.pitch_offset_deg, camera.roll_offset_deg);
02721     mat2D_transpose(DCM_trans, DCM);
02722
02723     Mat2D temp_vec = mat2D_alloc(3, 1);
02724     Mat2D camera_direction = mat2D_alloc(3, 1);
02725
02726     /* rotating camera_direction */
02727     mat2D_dot(camera_direction, DCM_trans, camera.direction);
02728
02729     /* calc new forward direction */
02730     Mat2D new_forward = mat2D_alloc(3, 1);
02731     mat2D_copy(new_forward, camera_direction);
02732     mat2D_mult(new_forward, 1.0 / mat2D_calc_norma(new_forward));
02733
02734     /* calc new up direction */
02735     mat2D_copy(temp_vec, new_forward);
02736     mat2D_mult(temp_vec, mat2D_dot_product(up, new_forward));
02737     Mat2D new_up = mat2D_alloc(3, 1);
02738     mat2D_copy(new_up, up);
02739     mat2D_sub(new_up, temp_vec);
02740     mat2D_mult(new_up, 1.0 / mat2D_calc_norma(new_up));
02741
02742     /* calc new right direction */
02743     Mat2D new_right = mat2D_alloc(3, 1);
02744     mat2D_cross(new_right, new_up, new_forward);
02745     mat2D_mult(new_right, 1.0 / mat2D_calc_norma(new_right));
02746
02747     mat2D_copy(camera.camera_x, new_right);
02748     mat2D_copy(camera.camera_y, new_up);
```

```
02749     mat2D_copy(camera.camera_z, new_forward);
02750
02751     /* adding offset to init_position */
02752     // mat2D_add(camera_pos, camera.offset_position);
02753
02754     mat2D_copy(temp_vec, camera.camera_x);
02755     mat2D_mult(temp_vec, MAT2D_AT(camera.offset_position, 0, 0));
02756     mat2D_add(camera.current_position, temp_vec);
02757     mat2D_copy(temp_vec, camera.camera_y);
02758     mat2D_mult(temp_vec, MAT2D_AT(camera.offset_position, 1, 0));
02759     mat2D_add(camera.current_position, temp_vec);
02760     mat2D_copy(temp_vec, camera.camera_z);
02761     mat2D_mult(temp_vec, MAT2D_AT(camera.offset_position, 2, 0));
02762     mat2D_add(camera.current_position, temp_vec);
02763
02764     mat2D_fill(camera.offset_position, 0);
02765
02766     MAT2D_AT(view_mat, 0, 0) = MAT2D_AT(new_right, 0, 0);
02767     MAT2D_AT(view_mat, 0, 1) = MAT2D_AT(new_up, 0, 0);
02768     MAT2D_AT(view_mat, 0, 2) = MAT2D_AT(new_forward, 0, 0);
02769     MAT2D_AT(view_mat, 0, 3) = 0;
02770     MAT2D_AT(view_mat, 1, 0) = MAT2D_AT(new_right, 1, 0);
02771     MAT2D_AT(view_mat, 1, 1) = MAT2D_AT(new_up, 1, 0);
02772     MAT2D_AT(view_mat, 1, 2) = MAT2D_AT(new_forward, 1, 0);
02773     MAT2D_AT(view_mat, 1, 3) = 0;
02774     MAT2D_AT(view_mat, 2, 0) = MAT2D_AT(new_right, 2, 0);
02775     MAT2D_AT(view_mat, 2, 1) = MAT2D_AT(new_up, 2, 0);
02776     MAT2D_AT(view_mat, 2, 2) = MAT2D_AT(new_forward, 2, 0);
02777     MAT2D_AT(view_mat, 2, 3) = 0;
02778     MAT2D_AT(view_mat, 3, 0) = - mat2D_dot_product(camera.current_position, new_right);
02779     MAT2D_AT(view_mat, 3, 1) = - mat2D_dot_product(camera.current_position, new_up);
02780     MAT2D_AT(view_mat, 3, 2) = - mat2D_dot_product(camera.current_position, new_forward);
02781     MAT2D_AT(view_mat, 3, 3) = 1;
02782
02783
02784     mat2D_free(temp_vec);
02785     mat2D_free(new_forward);
02786     mat2D_free(new_up);
02787     mat2D_free(new_right);
02788     mat2D_free(DCM);
02789     mat2D_free(DCM_trans);
02790     mat2D_free(camera_direction);
02791 }
02792
02806 Point ae_point_project_world2screen(Mat2D view_mat, Mat2D proj_mat, Point src, int window_w, int
       window_h)
02807 {
02808     Point view_point = ae_point_project_world2view(view_mat, src);
02809     Point screen_point = ae_point_project_view2screen(proj_mat, view_point, window_w, window_h);
02810
02811     return screen_point;
02812 }
02813
02824 Point ae_point_project_world2view(Mat2D view_mat, Point src)
02825 {
02826     ae_assert_point_is_valid(src);
02827
02828     Mat2D src_point_mat = mat2D_alloc(1,4);
02829     Mat2D des_point_mat = mat2D_alloc(1,4);
02830
02831     Point des_point = {0};
02832
02833     MAT2D_AT(src_point_mat, 0, 0) = src.x;
02834     MAT2D_AT(src_point_mat, 0, 1) = src.y;
02835     MAT2D_AT(src_point_mat, 0, 2) = src.z;
02836     MAT2D_AT(src_point_mat, 0, 3) = 1;
02837
02838     mat2D_dot(des_point_mat, src_point_mat, view_mat);
02839
02840     double w = MAT2D_AT(des_point_mat, 0, 3);
02841     AE_ASSERT(w == 1);
02842     des_point.x = MAT2D_AT(des_point_mat, 0, 0) / w;
02843     des_point.y = MAT2D_AT(des_point_mat, 0, 1) / w;
02844     des_point.z = MAT2D_AT(des_point_mat, 0, 2) / w;
02845     des_point.w = w;
02846
02847     mat2D_free(src_point_mat);
02848     mat2D_free(des_point_mat);
02849
02850     return des_point;
02851
02852 }
02853
02870 Point ae_point_project_view2screen(Mat2D proj_mat, Point src, int window_w, int window_h)
02871 {
02872     ae_assert_point_is_valid(src);
02873
```

```
02874     Mat2D src_point_mat = mat2D_alloc(1,4);
02875     Mat2D des_point_mat = mat2D_alloc(1,4);
02876     Point des;
02877 
02878     MAT2D_AT(src_point_mat, 0, 0) = src.x;
02879     MAT2D_AT(src_point_mat, 0, 1) = src.y;
02880     MAT2D_AT(src_point_mat, 0, 2) = src.z;
02881     MAT2D_AT(src_point_mat, 0, 3) = 1;
02882 
02883     mat2D_dot(des_point_mat, src_point_mat, proj_mat);
02884 
02885     double w = MAT2D_AT(des_point_mat, 0, 3);
02886     if (fabs(w) > 1e-3) {
02887         des.x = MAT2D_AT(des_point_mat, 0, 0) / w;
02888         des.y = MAT2D_AT(des_point_mat, 0, 1) / w;
02889         des.z = MAT2D_AT(des_point_mat, 0, 2) / w;
02890         des.w = w;
02891     } else {
02892         // des.x = MAT2D_AT(des_point_mat, 0, 0);
02893         // des.y = MAT2D_AT(des_point_mat, 0, 1);
02894         // des.z = MAT2D_AT(des_point_mat, 0, 2);
02895         // des.w = 1;
02896 
02897         des.x = 0;
02898         des.y = 0;
02899         des.z = 0;
02900         des.w = 1;
02901     }
02902 
02903     mat2D_free(src_point_mat);
02904     mat2D_free(des_point_mat);
02905 
02906     /* scale into view */
02907     des.x += 1;
02908     des.y += 1;
02909 
02910     des.x *= 0.5f * window_w;
02911     des.y *= 0.5f * window_h;
02912 
02913     return des;
02914 }
02915 
02933 void ae_line_project_world2screen(Mat2D view_mat, Mat2D proj_mat, Point start_src, Point end_src, int
       window_w, int window_h, Point *start_des, Point *end_des, Scene *scene)
02934 {
02935     Point start_view_point = ae_point_project_world2view(view_mat, start_src);
02936     Point end_view_point = ae_point_project_world2view(view_mat, end_src);
02937 
02938     Mat2D z_p = mat2D_alloc(3, 1);
02939     Mat2D z_n = mat2D_alloc(3, 1);
02940     mat2D_fill(z_p, 0);
02941     mat2D_fill(z_n, 0);
02942     MAT2D_AT(z_p, 2, 0) = scene->camera.z_near+0.01;
02943     MAT2D_AT(z_n, 2, 0) = 1;
02944 
02945     Point clipped_start_view_point = {0}, clipped_end_view_point = {0};
02946     int rc = ae_line_clip_with_plane(start_view_point, end_view_point, z_p, z_n,
       &clipped_start_view_point, &clipped_end_view_point);
02947 
02948     if (rc == -1) {
02949         fprintf(stderr, "%s:%s:%d:\n[error] problem with clipping lines\n\n", __FILE__, __func__,
       __LINE__);
02950         exit(1);
02951     } else if (rc == 0) {
02952         clipped_start_view_point = (Point){-1,-1,1,1};
02953         clipped_end_view_point = (Point){-1,-1,1,1};
02954         start_view_point = clipped_start_view_point;
02955         end_view_point = clipped_end_view_point;
02956 
02957         *start_des = start_view_point;
02958         *end_des = end_view_point;
02959         return;
02960 
02961     } else if (rc == 1) {
02962         start_view_point = clipped_start_view_point;
02963         end_view_point = clipped_end_view_point;
02964     }
02965 
02966 
02967     Point start_screen_point = ae_point_project_view2screen(proj_mat, start_view_point, window_w,
       window_h);
02968     Point end_screen_point = ae_point_project_view2screen(proj_mat, end_view_point, window_w,
       window_h);
02969 
02970     mat2D_free(z_p);
02971     mat2D_free(z_n);
02972 
```

```
02973     *start_des = start_screen_point;
02974     *end_des = end_screen_point;
02975
02976 }
02977
02988 Tri ae_tri_transform_to_view(Mat2D view_mat, Tri tri)
02989 {
02990     ae_assert_tri_is_valid(tri);
02991
02992     Mat2D src_point_mat = mat2D_alloc(1,4);
02993     Mat2D des_point_mat = mat2D_alloc(1,4);
02994
02995     Tri des_tri = tri;
02996
02997     for (int i = 0; i < 3; i++) {
02998         MAT2D_AT(src_point_mat, 0, 0) = tri.points[i].x;
02999         MAT2D_AT(src_point_mat, 0, 1) = tri.points[i].y;
03000         MAT2D_AT(src_point_mat, 0, 2) = tri.points[i].z;
03001         MAT2D_AT(src_point_mat, 0, 3) = 1;
03002
03003         mat2D_dot(des_point_mat, src_point_mat, view_mat);
03004
03005         double w = MAT2D_AT(des_point_mat, 0, 3);
03006         AE_ASSERT(w == 1);
03007         des_tri.points[i].x = MAT2D_AT(des_point_mat, 0, 0) / w;
03008         des_tri.points[i].y = MAT2D_AT(des_point_mat, 0, 1) / w;
03009         des_tri.points[i].z = MAT2D_AT(des_point_mat, 0, 2) / w;
03010         des_tri.points[i].w = w;
03011     }
03012
03013     mat2D_free(src_point_mat);
03014     mat2D_free(des_point_mat);
03015
03016     ae_assert_tri_is_valid(des_tri);
03017
03018     return des_tri;
03019 }
03020
03038 Tri_mesh ae_tri_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Tri tri, int window_w, int
       window_h, Scene *scene, Lighting_mode lighting_mode)
03039 {
03040     ae_assert_tri_is_valid(tri);
03041
03042     Mat2D tri_normal = mat2D_alloc(3, 1);
03043     Mat2D temp_camera2tri = mat2D_alloc(3, 1);
03044     Mat2D camera2tri = mat2D_alloc(1, 3);
03045     Mat2D dot_product = mat2D_alloc(1, 1);
03046     Tri des_tri = tri;
03047
03048     ae_point_to_mat2D(tri.points[0], temp_camera2tri);
03049     mat2D_sub(temp_camera2tri, scene->camera.current_position);
03050     mat2D_transpose(camera2tri, temp_camera2tri);
03051
03052     /* calc lighting intensity of tri */
03053     #if 1
03054         ae_tri_calc_light_intensity(&des_tri, scene, lighting_mode);
03055     #else
03056     for (int i = 0; i < 3; i++) {
03057         ae_point_to_mat2D(tri.normals[i], tri_normal);
03058         MAT2D_AT(dot_product, 0, 0) = MAT2D_AT(light_direction, 0, 0) * MAT2D_AT(tri_normal, 0, 0) +
       MAT2D_AT(light_direction, 1, 0) * MAT2D_AT(tri_normal, 1, 0) + MAT2D_AT(light_direction, 2, 0) *
       MAT2D_AT(tri_normal, 2, 0);
03059         des_tri.light_intensity[i] = fmaxf(0.2, fminf(1, MAT2D_AT(dot_product, 0, 0)));
03060     }
03061     #endif
03062
03063     /* calc if tri is visible to the camera */
03064     ae_tri_calc_normal(tri_normal, tri);
03065     // ae_point_to_mat2D(tri.normals[0], tri_normal);
03066     MAT2D_AT(dot_product, 0, 0) = MAT2D_AT(camera2tri, 0, 0) * MAT2D_AT(tri_normal, 0, 0) +
       MAT2D_AT(camera2tri, 0, 1) * MAT2D_AT(tri_normal, 1, 0) + MAT2D_AT(camera2tri, 0, 2) *
       MAT2D_AT(tri_normal, 2, 0);
03067     if (MAT2D_AT(dot_product, 0, 0) < 0) {
03068         des_tri.to_draw = true && des_tri.to_draw;
03069     } else {
03070         des_tri.to_draw = false && des_tri.to_draw;
03071     }
03072
03073     /* transform tri to camera view */
03074     tri = ae_tri_transform_to_view(view_mat, tri);
03075
03076     // Tri_mesh temp_tri_array;
03077     // ada_init_array(Tri, temp_tri_array);
03078     // ada_append(Tri, temp_tri_array, tri);
03079     /* clip tir */
03080     Tri clipped_tri1 = {0};
03081     Tri clipped_tri2 = {0};
```

```
03082     Mat2D z_plane_p = mat2D_alloc(3, 1);
03083     Mat2D z_plane_n = mat2D_alloc(3, 1);
03084     mat2D_fill(z_plane_p, 0);
03085     mat2D_fill(z_plane_n, 0);
03086     MAT2D_AT(z_plane_p, 2, 0) = scene->camera.z_near+0.01;
03087     MAT2D_AT(z_plane_n, 2, 0) = 1;
03088 
03089     int num_clipped_tri = ae_tri_clip_with_plane(tri, z_plane_p, z_plane_n, &clipped_tri1,
      &clipped_tri2);
03090     Tri_mesh temp_tri_array;
03091     ada_init_array(Tri, temp_tri_array);
03092     if (num_clipped_tri == -1) {
03093         fprintf(stderr, "%s:%s:%d:\n[error] problem with clipping triangles\n\n", __FILE__, __func__,
      __LINE__);
03094         exit(1);
03095     } else if (num_clipped_tri == 0) {
03096         ;
03097     } else if (num_clipped_tri == 1) {
03098         ae_assert_tri_is_valid(clipped_tri1);
03099         ada_append(Tri, temp_tri_array, clipped_tri1);
03100     } else if (num_clipped_tri == 2) {
03101         ae_assert_tri_is_valid(clipped_tri1);
03102         ae_assert_tri_is_valid(clipped_tri2);
03103         ada_append(Tri, temp_tri_array, clipped_tri1);
03104         ada_append(Tri, temp_tri_array, clipped_tri2);
03105     }
03106     mat2D_free(z_plane_p);
03107     mat2D_free(z_plane_n);
03108 
03109     for (size_t temp_tri_index = 0; temp_tri_index < temp_tri_array.length; temp_tri_index++) {
03110         /* project tri to screen */
03111         for (int i = 0; i < 3; i++) {
03112             des_tri.points[i] = ae_point_project_view2screen(proj_mat,
      temp_tri_array.elements[temp_tri_index].points[i], window_w, window_h);
03113 
03114             if (des_tri.points[i].w) {
03115                 des_tri.tex_points[i].x /= des_tri.points[i].w;
03116                 des_tri.tex_points[i].y /= des_tri.points[i].w;
03117                 des_tri.tex_points[i].z /= des_tri.points[i].w;
03118                 des_tri.tex_points[i].w  = des_tri.points[i].w;
03119             }
03120 
03121         }
03122         ae_assert_tri_is_valid(des_tri);
03123         temp_tri_array.elements[temp_tri_index] = des_tri;
03124     }
03125 
03126 
03127     mat2D_free(tri_normal);
03128     mat2D_free(temp_camera2tri);
03129     mat2D_free(camera2tri);
03130     mat2D_free(dot_product);
03131 
03132     return temp_tri_array;
03133 }
03134 
03151 void ae_tri_mesh_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Tri_mesh *des, Tri_mesh src, int
       window_w, int window_h, Scene *scene, Lighting_mode lighting_mode)
03152 {
03153     Tri_mesh temp_des = *des;
03154     temp_des.length = 0;
03155 
03156     size_t i;
03157     for (i = 0; i < src.length; i++) {
03158         Tri_mesh temp_tri_array = ae_tri_project_world2screen(proj_mat, view_mat, src.elements[i],
      window_w, window_h, scene, lighting_mode);
03159 
03160         for (size_t tri_index = 0; tri_index < temp_tri_array.length; tri_index++) {
03161             Tri temp_tri = temp_tri_array.elements[tri_index];
03162             ada_append(Tri, temp_des, temp_tri);
03163         }
03164 
03165         free(temp_tri_array.elements);
03166     }
03167 
03168     /* clip tir */
03169     int offset = 0;
03170     Mat2D top_p = mat2D_alloc(3, 1);
03171     Mat2D top_n = mat2D_alloc(3, 1);
03172     mat2D_fill(top_p, 0);
03173     mat2D_fill(top_n, 0);
03174     MAT2D_AT(top_p, 1, 0) = 0 + offset;
03175     MAT2D_AT(top_n, 1, 0) = 1;
03176 
03177     Mat2D bottom_p = mat2D_alloc(3, 1);
03178     Mat2D bottom_n = mat2D_alloc(3, 1);
03179     mat2D_fill(bottom_p, 0);
```

```
03180     mat2D_fill(bottom_n, 0);
03181     MAT2D_AT(bottom_p, 1, 0) = window_h - offset;
03182     MAT2D_AT(bottom_n, 1, 0) = -1;
03183 
03184     Mat2D left_p = mat2D_alloc(3, 1);
03185     Mat2D left_n = mat2D_alloc(3, 1);
03186     mat2D_fill(left_p, 0);
03187     mat2D_fill(left_n, 0);
03188     MAT2D_AT(left_p, 0, 0) = 0 + offset;
03189     MAT2D_AT(left_n, 0, 0) = 1;
03190 
03191     Mat2D right_p = mat2D_alloc(3, 1);
03192     Mat2D right_n = mat2D_alloc(3, 1);
03193     mat2D_fill(right_p, 0);
03194     mat2D_fill(right_n, 0);
03195     MAT2D_AT(right_p, 0, 0) = window_w - offset;
03196     MAT2D_AT(right_n, 0, 0) = -1;
03197 
03198     for (int plane_number = 0; plane_number < 4; plane_number++) {
03199         for (int tri_index = 0; tri_index < (int)(temp_des.length); tri_index++) {
03200             if (temp_des.length == 0) {
03201                 break;
03202             }
03203             // if (temp_des.elements[tri_index].to_draw == false) {
03204             //     ada_remove_unordered(Tri, temp_des, tri_index);
03205             //     tri_index--;
03206             //     tri_index = (int)fmaxf((float)tri_index, 0.0f);
03207             //     continue;
03208             // }
03209             Tri clipped_tri1 = {0};
03210             Tri clipped_tri2 = {0};
03211             int num_clipped_tri;
03212             switch (plane_number) {
03213                 case 0:
03214                     num_clipped_tri = ae_tri_clip_with_plane(temp_des.elements[tri_index], top_p,
    top_n, &clipped_tri1, &clipped_tri2);
03215                 break;
03216                 case 1:
03217                     num_clipped_tri = ae_tri_clip_with_plane(temp_des.elements[tri_index], right_p,
    right_n, &clipped_tri1, &clipped_tri2);
03218                 break;
03219                 case 2:
03220                     num_clipped_tri = ae_tri_clip_with_plane(temp_des.elements[tri_index], bottom_p,
    bottom_n, &clipped_tri1, &clipped_tri2);
03221                 break;
03222                 case 3:
03223                     num_clipped_tri = ae_tri_clip_with_plane(temp_des.elements[tri_index], left_p,
    left_n, &clipped_tri1, &clipped_tri2);
03224                 break;
03225             }
03226             if (num_clipped_tri == -1) {
03227                 fprintf(stderr, "%s:%s:%d:\n[error] problem with clipping triangles\n\n", __FILE__,
    __func__, __LINE__);
03228                 exit(1);
03229             } else if (num_clipped_tri == 0) {
03230                 ada_remove_unordered(Tri, temp_des, tri_index);
03231                 tri_index--;
03232                 tri_index = (int)fmaxf((float)tri_index, 0.0f);
03233             } else if (num_clipped_tri == 1) {
03234                 ae_assert_tri_is_valid(clipped_tri1);
03235                 temp_des.elements[tri_index] = clipped_tri1;
03236             } else if (num_clipped_tri == 2) {
03237                 ae_assert_tri_is_valid(clipped_tri1);
03238                 ae_assert_tri_is_valid(clipped_tri2);
03239                 temp_des.elements[tri_index] = clipped_tri1;
03240                 ada_insert_unordered(Tri, temp_des, clipped_tri2, tri_index+1);
03241             }
03242         }
03243     }
03244 
03245     // if (temp_des.length > 2) {
03246     //     ae_qsort_tri(temp_des.elements, 0, temp_des.length-1);
03247     // }
03248 
03249     mat2D_free(top_p);
03250     mat2D_free(top_n);
03251     mat2D_free(bottom_p);
03252     mat2D_free(bottom_n);
03253     mat2D_free(left_p);
03254     mat2D_free(left_n);
03255     mat2D_free(right_p);
03256     mat2D_free(right_n);
03257 
03258     *des = temp_des;
03259 }
03260 
03271 Quad ae_quad_transform_to_view(Mat2D view_mat, Quad quad)
```

```
03272 {
03273     ae_assert_quad_is_valid(quad);
03274 
03275     Mat2D src_point_mat = mat2D_alloc(1,4);
03276     Mat2D des_point_mat = mat2D_alloc(1,4);
03277 
03278     Quad des_quad = quad;
03279 
03280     for (int i = 0; i < 4; i++) {
03281         MAT2D_AT(src_point_mat, 0, 0) = quad.points[i].x;
03282         MAT2D_AT(src_point_mat, 0, 1) = quad.points[i].y;
03283         MAT2D_AT(src_point_mat, 0, 2) = quad.points[i].z;
03284         MAT2D_AT(src_point_mat, 0, 3) = 1;
03285 
03286         mat2D_dot(des_point_mat, src_point_mat, view_mat);
03287 
03288         double w = MAT2D_AT(des_point_mat, 0, 3);
03289         AE_ASSERT(w == 1);
03290         des_quad.points[i].x = MAT2D_AT(des_point_mat, 0, 0) / w;
03291         des_quad.points[i].y = MAT2D_AT(des_point_mat, 0, 1) / w;
03292         des_quad.points[i].z = MAT2D_AT(des_point_mat, 0, 2) / w;
03293         des_quad.points[i].w = w;
03294     }
03295 
03296     mat2D_free(src_point_mat);
03297     mat2D_free(des_point_mat);
03298 
03299     ae_assert_quad_is_valid(des_quad);
03300 
03301     return des_quad;
03302 }
03303 
03321 Quad_mesh ae_quad_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Quad quad, int window_w, int
       window_h, Scene *scene, Lighting_mode lighting_mode)
03322 {
03323     ae_assert_quad_is_valid(quad);
03324 
03325     Mat2D quad_normal = mat2D_alloc(3, 1);
03326     Mat2D camera2quad = mat2D_alloc(3, 1);
03327     Mat2D dot_product = mat2D_alloc(1, 1);
03328     Quad des_quad = quad;
03329 
03330     /* calc lighting intensity of tri */
03331     #if 1
03332         ae_quad_calc_light_intensity(&des_quad, scene, lighting_mode);
03333     #else
03334     for (int i = 0; i < 4; i++) {
03335         ae_point_to_mat2D(quad.normals[i], quad_normal);
03336         MAT2D_AT(dot_product, 0, 0) = scene->light_source0.light_direction_or_pos.x *
       MAT2D_AT(quad_normal, 0, 0) + scene->light_source0.light_direction_or_pos.y * MAT2D_AT(quad_normal,
       1, 0) + scene->light_source0.light_direction_or_pos.z * MAT2D_AT(quad_normal, 2, 0);
03337         des_quad.light_intensity[i] = fmaxf(0.2, fminf(1, MAT2D_AT(dot_product, 0, 0)));
03338     }
03339     #endif
03340 
03341     /* calc if quad is visible to the camera */
03342     bool visible = 0;
03343     #if 1
03344     for (int i = 0; i < 4; i++) {
03345         ae_point_to_mat2D(quad.points[i], camera2quad);
03346         mat2D_sub(camera2quad, scene->camera.current_position);
03347 
03348         ae_point_to_mat2D(quad.normals[i], quad_normal);
03349         MAT2D_AT(dot_product, 0, 0) = MAT2D_AT(camera2quad, 0, 0) * MAT2D_AT(quad_normal, 0, 0) +
       MAT2D_AT(camera2quad, 1, 0) * MAT2D_AT(quad_normal, 1, 0) + MAT2D_AT(camera2quad, 2, 0) *
       MAT2D_AT(quad_normal, 2, 0);
03350         visible = visible || (MAT2D_AT(dot_product, 0, 0) < 0);
03351     }
03352     #else
03353     ae_point_to_mat2D(quad.points[0], camera2quad);
03354     mat2D_sub(camera2quad, scene->camera.current_position);
03355 
03356     Point ave_norm = ae_quad_get_average_normal(quad);
03357     ae_point_to_mat2D(ave_norm, quad_normal);
03358     MAT2D_AT(dot_product, 0, 0) = MAT2D_AT(camera2quad, 0, 0) * MAT2D_AT(quad_normal, 0, 0) +
       MAT2D_AT(camera2quad, 1, 0) * MAT2D_AT(quad_normal, 1, 0) + MAT2D_AT(camera2quad, 2, 0) *
       MAT2D_AT(quad_normal, 2, 0);
03359     visible = MAT2D_AT(dot_product, 0, 0) < 0;
03360     #endif
03361 
03362     if (visible) {
03363         des_quad.to_draw = true && des_quad.to_draw;
03364     } else {
03365         des_quad.to_draw = false && des_quad.to_draw;
03366     }
03367 
03368     /* transform quad to camera view */
```

```
03369     quad = ae_quad_transform_to_view(view_mat, quad);
03370
03371     /* clip quad */
03372     Quad  clipped_quad1 = {0}, clipped_quad2 = {0};
03373     Mat2D z_plane_p = mat2D_alloc(3, 1);
03374     Mat2D z_plane_n = mat2D_alloc(3, 1);
03375     mat2D_fill(z_plane_p, 0);
03376     mat2D_fill(z_plane_n, 0);
03377     MAT2D_AT(z_plane_p, 2, 0) = scene->camera.z_near+0.01;
03378     MAT2D_AT(z_plane_n, 2, 0) = 1;
03379
03380     int num_clipped_quad = ae_quad_clip_with_plane(quad, z_plane_p, z_plane_n, &clipped_quad1,
      &clipped_quad2);
03381     Quad_mesh temp_quad_array;
03382     ada_init_array(Quad, temp_quad_array);
03383     if (num_clipped_quad == -1) {
03384         fprintf(stderr, "%s:%s:%d:\n[error] problem with clipping quad\n\n", __FILE__, __func__,
      __LINE__);
03385         exit(1);
03386     } else if (num_clipped_quad == 0) {
03387         ;
03388     } else if (num_clipped_quad == 1) {
03389         ae_assert_quad_is_valid(clipped_quad1);
03390         ada_append(Quad, temp_quad_array, clipped_quad1);
03391     } else if (num_clipped_quad == 2) {
03392         ae_assert_quad_is_valid(clipped_quad1);
03393         ae_assert_quad_is_valid(clipped_quad2);
03394         ada_append(Quad, temp_quad_array, clipped_quad1);
03395         ada_append(Quad, temp_quad_array, clipped_quad2);
03396     }
03397     mat2D_free(z_plane_p);
03398     mat2D_free(z_plane_n);
03399
03400     for (size_t temp_quad_index = 0; temp_quad_index < temp_quad_array.length; temp_quad_index++) {
03401         /* project quad to screen */
03402         for (int i = 0; i < 4; i++) {
03403             des_quad.points[i] = ae_point_project_view2screen(proj_mat,
      temp_quad_array.elements[temp_quad_index].points[i], window_w, window_h);
03404
03405         }
03406         ae_assert_quad_is_valid(des_quad);
03407         temp_quad_array.elements[temp_quad_index] = des_quad;
03408     }
03409
03410
03411     mat2D_free(quad_normal);
03412     mat2D_free(camera2quad);
03413     mat2D_free(dot_product);
03414
03415     return temp_quad_array;
03416 }
03417
03434 void ae_quad_mesh_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Quad_mesh *des, Quad_mesh src,
       int window_w, int window_h, Scene *scene, Lighting_mode lighting_mode)
03435 {
03436     Quad_mesh temp_des = *des;
03437     temp_des.length = 0;
03438
03439     size_t i;
03440     for (i = 0; i < src.length; i++) {
03441         Quad_mesh temp_quad_array = ae_quad_project_world2screen(proj_mat, view_mat, src.elements[i],
      window_w, window_h, scene, lighting_mode);
03442
03443         for (size_t quad_index = 0; quad_index < temp_quad_array.length; quad_index++) {
03444             Quad temp_quad = temp_quad_array.elements[quad_index];
03445             ada_append(Quad, temp_des, temp_quad);
03446         }
03447
03448         free(temp_quad_array.elements);
03449     }
03450
03451
03452     /* clip quad */
03453     int offset = 0;
03454     Mat2D top_p = mat2D_alloc(3, 1);
03455     Mat2D top_n = mat2D_alloc(3, 1);
03456     mat2D_fill(top_p, 0);
03457     mat2D_fill(top_n, 0);
03458     MAT2D_AT(top_p, 1, 0) = 0 + offset;
03459     MAT2D_AT(top_n, 1, 0) = 1;
03460
03461     Mat2D bottom_p = mat2D_alloc(3, 1);
03462     Mat2D bottom_n = mat2D_alloc(3, 1);
03463     mat2D_fill(bottom_p, 0);
03464     mat2D_fill(bottom_n, 0);
03465     MAT2D_AT(bottom_p, 1, 0) = window_h - offset;
03466     MAT2D_AT(bottom_n, 1, 0) = -1;
```

```
03467
03468     Mat2D left_p = mat2D_alloc(3, 1);
03469     Mat2D left_n = mat2D_alloc(3, 1);
03470     mat2D_fill(left_p, 0);
03471     mat2D_fill(left_n, 0);
03472     MAT2D_AT(left_p, 0, 0) = 0 + offset;
03473     MAT2D_AT(left_n, 0, 0) = 1;
03474
03475     Mat2D right_p = mat2D_alloc(3, 1);
03476     Mat2D right_n = mat2D_alloc(3, 1);
03477     mat2D_fill(right_p, 0);
03478     mat2D_fill(right_n, 0);
03479     MAT2D_AT(right_p, 0, 0) = window_w - offset;
03480     MAT2D_AT(right_n, 0, 0) = -1;
03481
03482     for (int plane_number = 0; plane_number < 4; plane_number++) {
03483         for (int quad_index = 0; quad_index < (int)(temp_des.length); quad_index++) {
03484             if (temp_des.length == 0) {
03485                 break;
03486             }
03487             // if (temp_des.elements[quad_index].to_draw == false) {
03488             //     ada_remove_unordered(Quad, temp_des, quad_index);
03489             //     quad_index--;
03490             //     quad_index = (int)fmaxf((float)quad_index, 0.0f);
03491             //     continue;
03492             // }
03493             Quad clipped_quad1 = {0}, clipped_quad2 = {0};
03494             int num_clipped_quad;
03495             switch (plane_number) {
03496                 case 0:
03497                     num_clipped_quad = ae_quad_clip_with_plane(temp_des.elements[quad_index], top_p,
    top_n, &clipped_quad1, &clipped_quad2);
03498                 break;
03499                 case 1:
03500                     num_clipped_quad = ae_quad_clip_with_plane(temp_des.elements[quad_index], right_p,
    right_n, &clipped_quad1, &clipped_quad2);
03501                 break;
03502                 case 2:
03503                     num_clipped_quad = ae_quad_clip_with_plane(temp_des.elements[quad_index],
    bottom_p, bottom_n, &clipped_quad1, &clipped_quad2);
03504                 break;
03505                 case 3:
03506                     num_clipped_quad = ae_quad_clip_with_plane(temp_des.elements[quad_index], left_p,
    left_n, &clipped_quad1, &clipped_quad2);
03507                 break;
03508             }
03509             if (num_clipped_quad == -1) {
03510                 fprintf(stderr, "%s:%s:%d:\n[error] problem with clipping quads\n\n", __FILE__,
    __func__, __LINE__);
03511                 exit(1);
03512             } else if (num_clipped_quad == 0) {
03513                 ada_remove_unordered(Quad, temp_des, quad_index);
03514                 quad_index--;
03515                 quad_index = (int)fmaxf((float)quad_index, 0.0f);
03516             } else if (num_clipped_quad == 1) {
03517                 ae_assert_quad_is_valid(clipped_quad1);
03518                 temp_des.elements[quad_index] = clipped_quad1;
03519             } else if (num_clipped_quad == 2) {
03520                 ae_assert_quad_is_valid(clipped_quad1);
03521                 ae_assert_quad_is_valid(clipped_quad2);
03522                 temp_des.elements[quad_index] = clipped_quad1;
03523                 ada_insert_unordered(Quad, temp_des, clipped_quad2, quad_index+1);
03524                 quad_index++;
03525             }
03526         }
03527     }
03528
03529     mat2D_free(top_p);
03530     mat2D_free(top_n);
03531     mat2D_free(bottom_p);
03532     mat2D_free(bottom_n);
03533     mat2D_free(left_p);
03534     mat2D_free(left_n);
03535     mat2D_free(right_p);
03536     mat2D_free(right_n);
03537
03538     *des = temp_des;
03539 }
03540
03541
03558 void ae_curve_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Curve *des, Curve src, int
    window_w, int window_h, Scene *scene)
03559 {
03560     ae_curve_copy(des, src);
03561     Curve temp_des = *des;
03562     /* set planes */
03563     int offset = 50;
```

```
03564     Mat2D top_p = mat2D_alloc(3, 1);
03565     Mat2D top_n = mat2D_alloc(3, 1);
03566     mat2D_fill(top_p, 0);
03567     mat2D_fill(top_n, 0);
03568     MAT2D_AT(top_p, 1, 0) = 0 + offset;
03569     MAT2D_AT(top_n, 1, 0) = 1;
03570     Mat2D bottom_p = mat2D_alloc(3, 1);
03571     Mat2D bottom_n = mat2D_alloc(3, 1);
03572     mat2D_fill(bottom_p, 0);
03573     mat2D_fill(bottom_n, 0);
03574     MAT2D_AT(bottom_p, 1, 0) = window_h - offset;
03575     MAT2D_AT(bottom_n, 1, 0) = -1;
03576     Mat2D left_p = mat2D_alloc(3, 1);
03577     Mat2D left_n = mat2D_alloc(3, 1);
03578     mat2D_fill(left_p, 0);
03579     mat2D_fill(left_n, 0);
03580     MAT2D_AT(left_p, 0, 0) = 0 + offset;
03581     MAT2D_AT(left_n, 0, 0) = 1;
03582     Mat2D right_p = mat2D_alloc(3, 1);
03583     Mat2D right_n = mat2D_alloc(3, 1);
03584     mat2D_fill(right_p, 0);
03585     mat2D_fill(right_n, 0);
03586     MAT2D_AT(right_p, 0, 0) = window_w - offset;
03587     MAT2D_AT(right_n, 0, 0) = -1;
03588 
03589     for (size_t point_index = 0; point_index < temp_des.length-1; point_index++) {
03590         Point start_src_point = src.elements[point_index];
03591         Point end_src_point = src.elements[point_index+1];
03592 
03593         Point start_des_point = {0}, end_des_point = {0};
03594 
03595         ae_line_project_world2screen(view_mat, proj_mat, start_src_point, end_src_point, window_w,
    window_h, &start_des_point, &end_des_point, scene);
03596 
03597 
03598         Point clipped_start_des_point = {0}, clipped_end_des_point = {0};
03599 
03600         for (int plane_number = 0; plane_number < 4; plane_number++) {
03601             int rc;
03602             switch (plane_number) {
03603                 case 0:
03604                     rc = ae_line_clip_with_plane(start_des_point, end_des_point, top_p, top_n,
    &clipped_start_des_point, &clipped_end_des_point);
03605                 break;
03606                 case 1:
03607                     rc = ae_line_clip_with_plane(start_des_point, end_des_point, right_p, right_n,
    &clipped_start_des_point, &clipped_end_des_point);
03608                 break;
03609                 case 2:
03610                     rc = ae_line_clip_with_plane(start_des_point, end_des_point, bottom_p, bottom_n,
    &clipped_start_des_point, &clipped_end_des_point);
03611                 break;
03612                 case 3:
03613                     rc = ae_line_clip_with_plane(start_des_point, end_des_point, left_p, left_n,
    &clipped_start_des_point, &clipped_end_des_point);
03614                 break;
03615             }
03616             if (rc == -1) {
03617                 fprintf(stderr, "%s:%s:%d:\n[error] problem with clipping lines\n\n", __FILE__,
    __func__, __LINE__);
03618                 exit(1);
03619             } else if (rc == 0) {
03620                 clipped_start_des_point = (Point){-1,-1,1,1};
03621                 clipped_end_des_point = (Point){-1,-1,1,1};
03622                 start_des_point = clipped_start_des_point;
03623                 end_des_point = clipped_end_des_point;
03624                 temp_des.elements[point_index] = start_des_point;
03625                 temp_des.elements[point_index+1] = end_des_point;
03626             } else if (rc == 1) {
03627                 start_des_point = clipped_start_des_point;
03628                 end_des_point = clipped_end_des_point;
03629                 temp_des.elements[point_index] = start_des_point;
03630                 temp_des.elements[point_index+1] = end_des_point;
03631             }
03632         }
03633 
03634     }
03635 
03636     Point default_point = (Point){-1,-1,1,1};
03637     for (int i = 0; i < (int)temp_des.length; i++) {
03638         if (ae_points_equal(temp_des.elements[i], default_point)) {
03639             ada_remove(Point, temp_des, i);
03640             i--;
03641         }
03642     }
03643 
03644     *des = temp_des;
```

```
03645
03646     mat2D_free(top_p);
03647     mat2D_free(top_n);
03648     mat2D_free(bottom_p);
03649     mat2D_free(bottom_n);
03650     mat2D_free(left_p);
03651     mat2D_free(left_n);
03652     mat2D_free(right_p);
03653     mat2D_free(right_n);
03654 }
03655
03670 void ae_curve_ada_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Curve_ada *des, Curve_ada src,
       int window_w, int window_h, Scene *scene)
03671 {
03672     if (src.length == 0) return;
03673     for (size_t curve_index = 0; curve_index < src.length; curve_index++) {
03674         ae_curve_project_world2screen(proj_mat, view_mat, &(des->elements[curve_index]),
       src.elements[curve_index], window_w, window_h, scene);
03675     }
03676 }
03677
03691 void ae_grid_project_world2screen(Mat2D proj_mat, Mat2D view_mat, Grid des, Grid src, int window_w,
       int window_h, Scene *scene)
03692 {
03693     if (src.curves.length == 0) return;
03694     for (size_t curve_index = 0; curve_index < src.curves.length; curve_index++) {
03695         ae_curve_project_world2screen(proj_mat, view_mat, &(des.curves.elements[curve_index]),
       src.curves.elements[curve_index], window_w, window_h, scene);
03696     }
03697 }
03698
03706 void ae_tri_swap(Tri *v, int i, int j)
03707 {
03708     Tri temp;
03709
03710     temp = v[i];
03711     v[i] = v[j];
03712     v[j] = temp;
03713 }
03714
03725 bool ae_tri_compare(Tri t1, Tri t2)
03726 {
03727     float t1_z_max = fmaxf(t1.points[0].z, fmaxf(t1.points[1].z, t1.points[2].z));
03728     float t2_z_max = fmaxf(t2.points[0].z, fmaxf(t2.points[1].z, t2.points[2].z));
03729
03730     return t1_z_max > t2_z_max;
03731 }
03732
03742 void ae_tri_qsort(Tri *v, int left, int right)
03743 {
03744     int i, last;
03745
03746     if (left >= right)                  /* do nothing if array contains */
03747         return;                         /* fewer than two elements */
03748     ae_tri_swap(v, left, (left + right) / 2);  /* move partition elem */
03749     last = left;                        /* to v[0] */
03750     for (i = left + 1; i <= right; i++) /* partition */
03751         if (ae_tri_compare(v[i], v[left]))
03752             ae_tri_swap(v, ++last, i);
03753     ae_tri_swap(v, left, last); /* restore partition elem */
03754     ae_tri_qsort(v, left, last - 1);
03755     ae_tri_qsort(v, last + 1, right);
03756 }
03757
03770 double ae_linear_map(double s, double min_in, double max_in, double min_out, double max_out)
03771 {
03772     return (min_out + ((s-min_in)*(max_out-min_out))/(max_in-min_in));
03773 }
03774
03785 void ae_z_buffer_copy_to_screen(Mat2D_uint32 screen_mat, Mat2D inv_z_buffer)
03786 {
03787     double max_inv_z = 0;
03788     double min_inv_z = DBL_MAX;
03789     for (size_t i = 0; i < inv_z_buffer.rows; i++) {
03790         for (size_t j = 0; j < inv_z_buffer.cols; j++) {
03791             if (MAT2D_AT(inv_z_buffer, i, j) > max_inv_z) {
03792                 max_inv_z = MAT2D_AT(inv_z_buffer, i, j);
03793             }
03794             if (MAT2D_AT(inv_z_buffer, i, j) < min_inv_z && MAT2D_AT(inv_z_buffer, i, j) > 0) {
03795                 min_inv_z = MAT2D_AT(inv_z_buffer, i, j);
03796             }
03797         }
03798     }
03799     for (size_t i = 0; i < inv_z_buffer.rows; i++) {
03800         for (size_t j = 0; j < inv_z_buffer.cols; j++) {
03801             double z_fraq = MAT2D_AT(inv_z_buffer, i, j);
03802             z_fraq = fmax(z_fraq, min_inv_z);
```

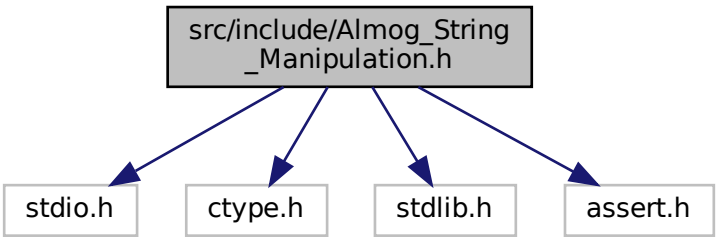
```
03803             z_fraq = ae_linear_map(z_fraq, min_inv_z, max_inv_z, 0.1, 1);
03804             uint32_t color = RGB_hexRGB(0xFF*z_fraq, 0xFF*z_fraq, 0xFF*z_fraq);
03805             MAT2D_AT_UINT32(screen_mat, i, j) = color;
03806         }
03807     }
03808 }
03809
03810 #endif /* ALMOG_ENGINE_IMPLEMENTATION */
```

4.9 src/include/Almog_String_Manipulation.h File Reference

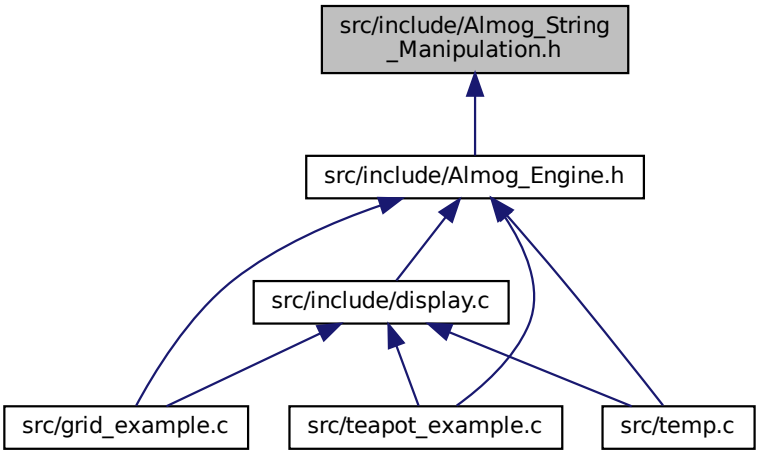
Lightweight string and line manipulation helpers.

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

Include dependency graph for Almog_String_Manipulation.h:



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



Macros

- #define ASM_MAXDIR 100

  Generic maximum directory length constant (not used by the functions in this header but available to callers).
- #define ASM_MAX_LEN_LINE (int)1e3

  Maximum number of characters read by asm_get_line (excluding the terminating null).
- #define asm_dprintSTRING(expr) printf(#expr " = %s\n", expr)

  Debug print a C string expression as "expr = value\n".
- #define asm_dprintCHAR(expr) printf(#expr " = %c\n", expr)

  Debug print a character expression as "expr = c\n".
- #define asm_dprintINT(expr) printf(#expr " = %d\n", expr)

  Debug print an integer expression as "expr = n\n".
- #define asm_dprintSIZE_T(expr) printf(#expr " = %zu\n", expr)

  Debug print a size_t expression as "expr = n\n".

Functions

- int asm_get_line (FILE ∗fp, char ∗dst)

  Read a single line from a stream into a buffer.
- int asm_length (char ∗str)

  Compute the length of a null-terminated C string.
- int asm_get_next_word_from_line (char ∗dst, char ∗src, char seperator)

  Extract the next word from a line without modifying the source.
- void asm_copy_array_by_indesies (char ∗target, int start, int end, char ∗src)

  Copy a substring [start, end) from src into target and null-terminate.
- int asm_get_word_and_cut (char ∗dst, char ∗src, char seperator)

  Get the next word and cut the source string at that point.
- int asm_str_in_str (char ∗src, char ∗word2search)

  Count occurrences of a substring within a string.
- int asm_strncmp (const char ∗s1, const char ∗s2, const int N)

  Compare up to N characters for equality (boolean result).

4.9.1 Detailed Description

Lightweight string and line manipulation helpers.

This single-header module provides small utilities for working with C strings:

- Reading a single line from a FILE stream
- Measuring string length
- Extracting the next "word" (token) from a line using a separator
- Cutting the extracted word from the source buffer
- Copying a substring by indices
- Counting occurrences of a substring
- A boolean-style strncmp (returns 1 on equality, 0 otherwise)

Usage

- In exactly one translation unit, define ALMOG_STRING_MANIPULATION_IMPLEMENTATION before including this header to compile the implementation.
- In all other files, include the header without the macro to get declarations only.

Notes and limitations

- All destination buffers must be large enough; functions do not grow or allocate buffers.
- asm_get_line enforces MAX_LEN_LINE characters (not counting the terminating '\0'). Longer lines cause a fatal error via exit(1).
- asm_strncmp differs from the standard C strncmp: this version returns 1 if equal and 0 otherwise.
- These functions are not locale-aware unless otherwise noted (isspace is used for whitespace handling).

Definition in file Almog_String_Manipulation.h.

4.9.2 Macro Definition Documentation

4.9.2.1 asm_dprintCHAR

```
#define asm_dprintCHAR(
              expr ) printf(#expr " = %c\n", expr)
```

Debug print a character expression as "expr = c\n".

Parameters

expr	An expression that yields a character promoted to int.

Definition at line 72 of file Almog_String_Manipulation.h.

4.9.2.2 asm_dprintINT

```
#define asm_dprintINT(
              expr ) printf(#expr " = %d\n", expr)
```

Debug print an integer expression as "expr = n\n".

Parameters

expr	An expression that yields an int.

Definition at line 79 of file Almog_String_Manipulation.h.

4.9.2.3 asm_dprintSIZE_T

```
#define asm_dprintSIZE_T(
              expr ) printf(#expr " = %zu\n", expr)
```

Debug print a size_t expression as "expr = n\n".

Parameters

expr	An expression that yields a size_t.

Definition at line 86 of file Almog_String_Manipulation.h.

4.9.2.4 asm_dprintSTRING

```
#define asm_dprintSTRING(
              expr ) printf(#expr " = %s\n", expr)
```

Debug print a C string expression as "expr = value\n".

Parameters

expr	An expression that yields a pointer to char (const or non-const).

Definition at line 65 of file Almog_String_Manipulation.h.

4.9.2.5 ASM_MAX_LEN_LINE

```
#define ASM_MAX_LEN_LINE (int)1e3
```

Maximum number of characters read by asm_get_line (excluding the terminating null).

If an input line exceeds this value before encountering '
' or EOF, asm_get_line prints an error to stderr and terminates the process with exit(1).

Definition at line 58 of file Almog_String_Manipulation.h.

4.9.2.6 ASM_MAXDIR

```
#define ASM_MAXDIR 100
```

Generic maximum directory length constant (not used by the functions in this header but available to callers).

Definition at line 47 of file Almog_String_Manipulation.h.

4.9.3 Function Documentation

4.9.3.1 asm_copy_array_by_indesies()

```
void asm_copy_array_by_indesies (
            char * target,
            int start,
            int end,
            char * src )
```

Copy a substring [start, end) from src into target and null-terminate.

Copies characters with indices i = start, start+1, ..., end-1 from src into target, then writes a terminating '\0'.

Parameters

target	Destination buffer. Must be large enough to hold (end - start) characters plus the null terminator.
start	Inclusive start index within src (0-based).
end	Exclusive end index within src (must satisfy end >= start).
src	Source string buffer.

Warning

No bounds checking is performed. The caller must ensure valid indices and sufficient target capacity.

Note

This routine supports in-place "left-shift" usage where target == src and start > 0 (used by asm_get_word_↩ and_cut).

Definition at line 232 of file Almog_String_Manipulation.h.

Referenced by asm_get_word_and_cut().

4.9.3.2 asm_get_line()

```
int asm_get_line (
            FILE * fp,
            char * dst )
```

Read a single line from a stream into a buffer.

Reads characters from the FILE stream until a newline ('
') or EOF is encountered. The newline, if present, is not copied. The result is always null-terminated.

Parameters

fp	Input stream (must be non-NULL).
dst	Destination buffer. Must have capacity of at least MAX_LEN_LINE + 1 bytes.

Returns

Number of characters stored in dst (excluding the terminating null).

Return values

-1	EOF was encountered before any character was read.

Note

If the line exceeds MAX_LEN_LINE characters before a newline or EOF, the function prints an error and calls exit(1).

An empty line returns 0 (not -1).

Definition at line 119 of file Almog_String_Manipulation.h.

References ASM_MAX_LEN_LINE.

Referenced by ae_tri_mesh_get_from_obj_file().

4.9.3.3 asm_get_next_word_from_line()

```
int asm_get_next_word_from_line (
            char * dst,
            char * src,
            char seperator )
```

Extract the next word from a line without modifying the source.

Skips leading whitespace in src (as determined by isspace), then copies characters into dst until one of the following is seen: the separator, a newline ('
'), or the string terminator ('\0'). The copied word in dst is null-terminated and is never empty on success.

Special case:

- If the very first character in src (at index 0, without leading whitespace) is the separator, '
', or '\0', that single character is returned as a one-character "word".

Parameters

dst	Destination buffer for the extracted word. Must be large enough to hold the token plus the null terminator.
src	Source C string to parse (not modified by this function).
seperator	Separator character to stop at (spelling as in the API).

Returns

The number of characters consumed from src (i.e., the index of the first unconsumed character).

Return values

-1	No word was found (e.g., only whitespace before a delimiter or end-of-string).

Note

The source buffer is not altered. To both extract and advance/cut the source, see asm_get_word_and_cut.

Definition at line 182 of file Almog_String_Manipulation.h.

Referenced by ae_tri_mesh_get_from_obj_file(), and asm_get_word_and_cut().

4.9.3.4 asm_get_word_and_cut()

```
int asm_get_word_and_cut (
              char * dst,
              char * src,
              char seperator )
```

Get the next word and cut the source string at that point.

Extracts the next word from src (per asm_get_next_word_from_line semantics) into dst. On success, src is modified in-place to remove the consumed prefix. The new src begins at the stopping character (the separator, newline, or terminator).

Example: For src = "abc,def", separator = ','

- dst becomes "abc"
- src becomes ",def" (note the leading separator remains)

Parameters

dst	Destination buffer for the extracted word (large enough for the token and terminating null).
src	Source buffer. Modified in-place if a word is found.
seperator	Separator character to stop at (spelling as in the API).

Returns

1 if a word was extracted and src adjusted, 0 otherwise.

Definition at line 260 of file Almog_String_Manipulation.h.

References asm_copy_array_by_indesies(), asm_get_next_word_from_line(), and asm_length().

Referenced by ae_tri_mesh_get_from_file(), and ae_tri_mesh_get_from_obj_file().

4.9.3.5 asm_length()

```
int asm_length (
            char * str )
```

Compute the length of a null-terminated C string.

Parameters

str	Null-terminated string (must be non-NULL).

Returns

The number of characters before the terminating null byte.

Definition at line 146 of file Almog_String_Manipulation.h.

Referenced by ae_tri_mesh_get_from_obj_file(), asm_get_word_and_cut(), and asm_str_in_str().

4.9.3.6 asm_str_in_str()

```
int asm_str_in_str (
            char * src,
            char * word2search )
```

Count occurrences of a substring within a string.

Counts how many times word2search appears in src. Occurrences may overlap.

Parameters

src	The string to search in (must be null-terminated).
word2search	The substring to find (must be null-terminated).

Returns

The number of (possibly overlapping) occurrences found.

Definition at line 285 of file Almog_String_Manipulation.h.

References asm_length(), and asm_strncmp().

Referenced by ae_tri_mesh_get_from_file(), and ae_tri_mesh_get_from_obj_file().

4.9.3.7 asm_strncmp()

```
int asm_strncmp (
            const char * s1,
            const char * s2,
            const int N )
```

Compare up to N characters for equality (boolean result).

Returns 1 if the first N characters of s1 and s2 are all equal; otherwise returns 0. Unlike the standard C strncmp, which returns 0 on equality and a non-zero value on inequality/order, this function returns a boolean-like result (1 == equal, 0 == different).

Parameters

s1	First string (may be shorter than N).
s2	Second string (may be shorter than N).
N	Number of characters to compare.

Returns

1 if equal for the first N characters, 0 otherwise.

Definition at line 310 of file Almog_String_Manipulation.h.

Referenced by asm_str_in_str().

4.10 Almog_String_Manipulation.h

```
00001
00034 #ifndef ALMOG_STRING_MANIPULATION_H_
00035 #define ALMOG_STRING_MANIPULATION_H_
00036
00037 #include <stdio.h>
00038 #include <ctype.h>
00039 #include <stdlib.h>
00040 #include <assert.h>
00041
00047 #define ASM_MAXDIR 100
00048
00058 #define ASM_MAX_LEN_LINE (int)1e3
00059
00065 #define asm_dprintSTRING(expr) printf(#expr " = %s\n", expr)
00066
00072 #define asm_dprintCHAR(expr) printf(#expr " = %c\n", expr)
```

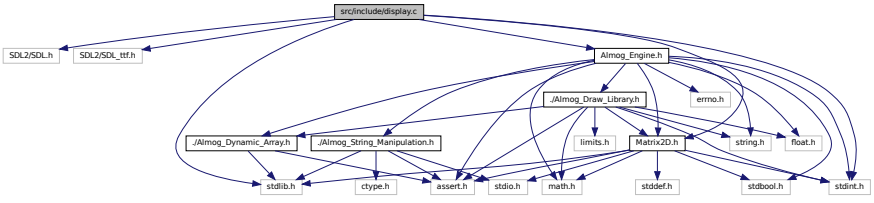
```
00073
00079 #define asm_dprintINT(expr) printf(#expr " = %d\n", expr)
00080
00086 #define asm_dprintSIZE_T(expr) printf(#expr " = %zu\n", expr)
00087
00088 int asm_get_line(FILE *fp, char *dst);
00089 int asm_length(char *str);
00090 int asm_get_next_word_from_line(char *dst, char *src, char seperator);
00091 void asm_copy_array_by_indesies(char *target, int start, int end, char *src);
00092 int asm_get_word_and_cut(char *dst, char *src, char seperator);
00093 int asm_str_in_str(char *src, char *word2search);
00094 int asm_strncmp(const char *s1, const char *s2, const int N);
00095
00096 #endif /*ALMOG_STRING_MANIPULATION_H_*/
00097
00098 #ifdef ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00099 #undef ALMOG_STRING_MANIPULATION_IMPLEMENTATION
00100
00101
00119 int asm_get_line(FILE *fp, char *dst)
00120 {
00121     int i = 0;
00122     char c;
00123
00124     while ((c = fgetc(fp)) != '\n' && c != EOF) {
00125         dst[i] = c;
00126         i++;
00127         if (i >= ASM_MAX_LEN_LINE) {
00128             fprintf(stderr, "ERROR: line too long\n");
00129             exit(1);
00130         }
00131     }
00132     dst[i] = '\0';
00133     if (c == EOF && i == 0) {
00134         return -1;
00135     }
00136     return i;
00137 }
00138
00146 int asm_length(char *str)
00147 {
00148     char c;
00149     int i = 0;
00150
00151     while ((c = str[i]) != '\0') {
00152         i++;
00153     }
00154     return i++;
00155 }
00156
00182 int asm_get_next_word_from_line(char *dst, char *src, char seperator)
00183 {
00184     int i = 0, j = 0;
00185     char c;
00186
00187     while (isspace((c = src[i]))) {
00188         i++;
00189     }
00190
00191     while ((c = src[i]) != seperator &&
00192                       c != '\n'&&
00193                       c != '\0') {
00194                         dst[j] = src[i];
00195                         i++;
00196                         j++;
00197     }
00198
00199     if ((c == seperator ||
00200          c == '\n'||
00201          c == '\0') && i == 0) {
00202             dst[j++] = c;
00203             i++;
00204     }
00205
00206     dst[j] = '\0';
00207
00208     if (j == 0) {
00209         return -1;
00210     }
00211     return i;
00212
00213 }
00214
00232 void asm_copy_array_by_indesies(char *target, int start, int end, char *src)
00233 {
00234     int j = 0;
00235     for (int i = start; i < end; i++) {
```

```
00236         target[j] = src[i];
00237         j++;
00238     }
00239     target[j] = '\0';
00240 }
00241
00260 int asm_get_word_and_cut(char *dst, char *src, char seperator)
00261 {
00262     int last_pos;
00263
00264     if (src[0] == '\0') {
00265         return 0;
00266     }
00267     last_pos = asm_get_next_word_from_line(dst, src, seperator);
00268     if (last_pos == -1) {
00269         return 0;
00270     }
00271     asm_copy_array_by_indesies(src, last_pos, asm_length(src), src);
00272     return 1;
00273 }
00274
00285 int asm_str_in_str(char *src, char *word2search)
00286 {
00287     int i = 0, num_of_accur = 0;
00288     while (src[i] != '\0') {
00289         if (asm_strncmp(src+i, word2search, asm_length(word2search))) {
00290             num_of_accur++;
00291         }
00292         i++;
00293     }
00294     return num_of_accur;
00295 }
00296
00310 int asm_strncmp(const char *s1, const char *s2, const int N)
00311 {
00312     int i = 0;
00313     while (i < N) {
00314         if (s1[i] == '\0' && s2[i] == '\0') {
00315             break;
00316         }
00317         if (s1[i] != s2[i] || (s1[i] == '\0') || (s2[i] == '\0')) {
00318             return 0;
00319         }
00320         i++;
00321     }
00322     return 1;
00323 }
00324
00325
00326 #endif /*ALMOG_STRING_MANIPULATION_IMPLEMENTATION*/
00327
```

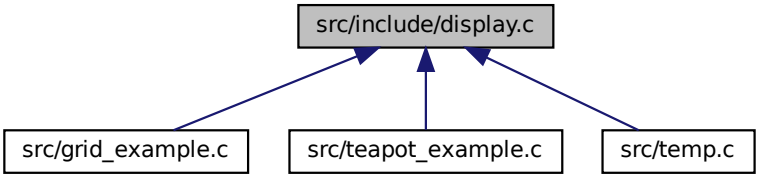
4.11 src/include/display.c File Reference

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include "Matrix2D.h"
#include <stdlib.h>
#include <stdint.h>
#include "Almog_Engine.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)

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 ()

4.11.1 Macro Definition Documentation

4.11.1.1 dprintCHAR

```
#define dprintCHAR(
              expr ) printf(#expr " = %c\n", expr)
```

Definition at line 25 of file display.c.

4.11.1.2 dprintD

```
#define dprintD(
              expr ) printf(#expr " = %g\n", expr)
```

Definition at line 27 of file display.c.

4.11.1.3 dprintINT

```
#define dprintINT(
              expr ) printf(#expr " = %d\n", expr)
```

Definition at line 26 of file display.c.

4.11.1.4 dprintSIZE_T

```
#define dprintSIZE_T(
              expr ) printf(#expr " = %zu\n", expr)
```

Definition at line 28 of file display.c.

4.11.1.5 dprintSTRING

```
#define dprintSTRING(
              expr ) printf(#expr " = %s\n", expr)
```

Definition at line 24 of file display.c.

4.11.1.6 FPS

```
#define FPS 100
```

Definition at line 17 of file display.c.

4.11.1.7 FRAME_TARGET_TIME

```
#define FRAME_TARGET_TIME (1000 / FPS)
```

Definition at line 21 of file display.c.

4.11.1.8 WINDOW_HEIGHT

```
#define WINDOW_HEIGHT (9 * 80)
```

Definition at line 13 of file display.c.

4.11.1.9 WINDOW_WIDTH

```
#define WINDOW_WIDTH (16 * 80)
```

Definition at line 9 of file display.c.

4.11.2 Function Documentation

4.11.2.1 check_window_mat_size()

```
void check_window_mat_size (
            game_state_t * game_state )
```

Definition at line 361 of file display.c.

References Camera::aspect_ratio, Scene::camera, 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::scene, 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().

4.11.2.2 copy_mat_to_surface_RGB()

```
void copy_mat_to_surface_RGB (
            game_state_t * game_state )
```

Definition at line 376 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().

4.11.2.3 destroy_window()

```
void destroy_window (
            game_state_t * game_state )
```

Definition at line 316 of file display.c.

References ae_scene_free(), game_state_t::inv_z_buffer_mat, mat2D_free(), mat2D_free_uint32(), game_state_t::renderer, game_state_t::scene, game_state_t::window, game_state_t::window_pixels_mat, game_state_t::window_surface, and game_state_t::window_texture.

Referenced by main().

4.11.2.4 fix_framerate()

```
void fix_framerate (
            game_state_t * game_state )
```

Definition at line 333 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().

4.11.2.5 initialize_window()

```
int initialize_window (
            game_state_t * game_state )
```

Definition at line 137 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().

4.11.2.6 main()

```
int main ( )
```

Definition at line 88 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, 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::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, and WINDOW_WIDTH.

4.11.2.7 process_input_window()

```
void process_input_window (
            game_state_t * game_state )
```

Definition at line 186 of file display.c.

References ae_camera_reset_pos(), Scene::camera, game_state_t::game_is_running, game_state_t::left_button_pressed, MAT2D_AT, Camera::offset_position, Camera::pitch_offset_deg, game_state_t::previous_frame_time, Camera::roll_offset_deg, game_state_t::scene, game_state_t::space_bar_was_pressed, game_state_t::to_render, and game_state_t::to_update.

Referenced by main().

4.11.2.8 render()

```
void render (
            game_state_t * game_state )
```

Definition at line 32 of file grid_example.c.

References ADL_DEFAULT_OFFSET_ZOOM, adl_grid_draw(), adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal(), Tri_mesh_array::elements, grid_proj, Scene::in_world_tri_meshes, game_state_t::inv_z_buffer_mat, Tri_mesh::length, Tri_mesh_array::length, Scene::projected_tri_meshes, game_state_t::scene, and game_state_t::window_pixels_mat.

Referenced by render_window().

4.11.2.9 render_window()

```
void render_window (
            game_state_t * game_state )
```

Definition at line 295 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().

4.11.2.10 setup()

```
void setup (
            game_state_t * game_state )
```

Definition at line 14 of file grid_example.c.

References ada_append, ada_init_array, adl_cartesian_grid_create(), ae_tri_mesh_append_copy(), ae_tri_mesh_get_from_file(), ae_tri_mesh_normalize(), ae_tri_mesh_rotate_Euler_xyz(), ASM_MAX_LEN_LINE, game_state_t::const_fps, Tri_mesh_array::elements, grid, grid_proj, Scene::in_world_tri_meshes, Tri_mesh::length, Tri_mesh_array::length, Scene::original_tri_meshes, Scene::projected_tri_meshes, game_state_t::scene, and game_state_t::to_limit_fps.

Referenced by setup_window().

4.11.2.11 setup_window()

```
void setup_window (
            game_state_t * game_state )
```

Definition at line 170 of file display.c.

References ae_scene_init(), game_state_t::inv_z_buffer_mat, mat2D_alloc(), mat2D_alloc_uint32(), game_state_t::scene, 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().

4.11.2.12 update()

```
void update (
            game_state_t * game_state )
```

Definition at line 23 of file grid_example.c.

References ae_grid_project_world2screen(), AE_LIGHTING_FLAT, ae_projection_mat_set(), ae_tri_mesh_project_world2screen(), ae_view_mat_set(), Camera::aspect_ratio, Scene::camera, Tri_mesh_array::elements, Camera::fov_deg, grid, grid_proj, Scene::in_world_tri_meshes, Tri_mesh_array::length, Scene::proj_mat, Scene::projected_tri_meshes, game_state_t::scene, Scene::up_direction, Scene::view_mat, game_state_t::window_h, game_state_t::window_w, Camera::z_far, and Camera::z_near.

Referenced by update_window().

4.11.2.13 update_window()

```
void update_window (
            game_state_t * game_state )
```

Definition at line 267 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().

4.12 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_Engine.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
00065     SDL_Surface *window_surface;
00066     SDL_Texture *window_texture;
00067
00068     Mat2D_uint32 window_pixels_mat;
00069     Mat2D inv_z_buffer_mat;
00070
00071     Scene scene;
00072 } game_state_t;
00073
00074 int initialize_window(game_state_t *game_state);
00075 void setup_window(game_state_t *game_state);
00076 void process_input_window(game_state_t *game_state);
00077 void update_window(game_state_t *game_state);
00078 void render_window(game_state_t *game_state);
00079 void destroy_window(game_state_t *game_state);
00080 void fix_framerate(game_state_t *game_state);
00081 void setup(game_state_t *game_state);
00082 void update(game_state_t *game_state);
00083 void render(game_state_t *game_state);
00084
00085 void check_window_mat_size(game_state_t *game_state);
```

```
00086 void copy_mat_to_surface_RGB(game_state_t *game_state);
00087
00088 int main()
00089 {
00090     game_state_t game_state = {0};
00091
00092     game_state.game_is_running = 0;
00093     game_state.delta_time = 0;
00094     game_state.elapsed_time = 0;
00095     game_state.const_fps = FPS;
00096     game_state.fps = 0;
00097     game_state.frame_target_time = FRAME_TARGET_TIME;
00098
00099     game_state.space_bar_was_pressed = 0;
00100     game_state.w_was_pressed = 0;
00101     game_state.s_was_pressed = 0;
00102     game_state.a_was_pressed = 0;
00103     game_state.d_was_pressed = 0;
00104     game_state.e_was_pressed = 0;
00105     game_state.q_was_pressed = 0;
00106
00107     game_state.to_render = 1;
00108     game_state.to_update = 1;
00109     game_state.previous_frame_time = 0;
00110     game_state.left_button_pressed = 0;
00111     game_state.to_limit_fps = 1;
00112     game_state.to_clear_renderer = 1;
00113     game_state.window = NULL;
00114     game_state.window_w = WINDOW_WIDTH;
00115     game_state.window_h = WINDOW_HEIGHT;
00116     game_state.renderer = NULL;
00117
00118     game_state.game_is_running = !initialize_window(&game_state);
00119
00120     setup_window(&game_state);
00121
00122     while (game_state.game_is_running) {
00123         process_input_window(&game_state);
00124         if (game_state.to_update) {
00125             update_window(&game_state);
00126         }
00127         if (game_state.to_render) {
00128             render_window(&game_state);
00129         }
00130
00131     }
00132     destroy_window(&game_state);
00133
00134     return 0;
00135 }
00136
00137 int initialize_window(game_state_t *game_state)
00138 {
00139     if (SDL_Init(SDL_INIT_EVERYTHING) != 0) {
00140         fprintf(stderr, "%s:%d: [Error] initializing SDL.\n", __FILE__, __LINE__);
00141         return -1;
00142     }
00143
00144     game_state->window = SDL_CreateWindow(NULL,
00145                                 SDL_WINDOWPOS_CENTERED,
00146                                 SDL_WINDOWPOS_CENTERED,
00147                                 game_state->window_w,
00148                                 game_state->window_h,
00149                                 SDL_WINDOW_RESIZABLE
00150                                 );
00151     if (!game_state->window) {
00152         fprintf(stderr, "%s:%d: [Error] creating SDL window.\n", __FILE__, __LINE__);
00153         return -1;
00154     }
00155
00156     game_state->renderer = SDL_CreateRenderer(game_state->window, -1, 0);
00157     if (!game_state->renderer) {
00158         fprintf(stderr, "%s:%d: [Error] creating SDL renderer.\n", __FILE__, __LINE__);
00159         return -1;
00160     }
00161
00162     if (TTF_Init() == -1) {
00163         fprintf(stderr, "%s:%d: [Error] initializing SDL_ttf.\n", __FILE__, __LINE__);
00164         return -1;
00165     }
00166
00167     return 0;
00168 }
00169
00170 void setup_window(game_state_t *game_state)
00171 {
00172
```

```
00173     game_state->window_surface = SDL_GetWindowSurface(game_state->window);
00174
00175     game_state->window_pixels_mat = mat2D_alloc_uint32(game_state->window_h, game_state->window_w);
00176     game_state->inv_z_buffer_mat = mat2D_alloc(game_state->window_h, game_state->window_w);
00177
00178     game_state->scene = ae_scene_init(game_state->window_h, game_state->window_w);
00179
00180     /*-----------------------------------*/
00181
00182     setup(game_state);
00183
00184 }
00185
00186 void process_input_window(game_state_t *game_state)
00187 {
00188     SDL_Event event;
00189     while (SDL_PollEvent(&event)) {
00190         switch (event.type) {
00191             case SDL_QUIT:
00192                 game_state->game_is_running = 0;
00193                 break;
00194             case SDL_KEYDOWN:
00195                 if (event.key.keysym.sym == SDLK_ESCAPE) {
00196                     game_state->game_is_running = 0;
00197                 }
00198                 if (event.key.keysym.sym == SDLK_SPACE) {
00199                     if (!game_state->space_bar_was_pressed) {
00200                         game_state->to_render = 0;
00201                         game_state->to_update = 0;
00202                         game_state->space_bar_was_pressed = 1;
00203                         break;
00204                     }
00205                     if (game_state->space_bar_was_pressed) {
00206                         game_state->to_render = 1;
00207                         game_state->to_update = 1;
00208                         game_state->previous_frame_time = SDL_GetTicks();
00209                         game_state->space_bar_was_pressed = 0;
00210                         break;
00211                     }
00212                 }
00213                 if (event.key.keysym.sym == SDLK_w) {
00214                     MAT2D_AT(game_state->scene.camera.offset_position, 1, 0) -= 0.05;
00215                 }
00216                 if (event.key.keysym.sym == SDLK_s) {
00217                     MAT2D_AT(game_state->scene.camera.offset_position, 1, 0) += 0.05;
00218                 }
00219                 if (event.key.keysym.sym == SDLK_d) {
00220                     MAT2D_AT(game_state->scene.camera.offset_position, 0, 0) += 0.05;
00221                 }
00222                 if (event.key.keysym.sym == SDLK_a) {
00223                     MAT2D_AT(game_state->scene.camera.offset_position, 0, 0) -= 0.05;
00224                 }
00225                 if (event.key.keysym.sym == SDLK_e) {
00226                     MAT2D_AT(game_state->scene.camera.offset_position, 2, 0) += 0.05;
00227                 }
00228                 if (event.key.keysym.sym == SDLK_q) {
00229                     MAT2D_AT(game_state->scene.camera.offset_position, 2, 0) -= 0.05;
00230                 }
00231                 if (event.key.keysym.sym == SDLK_LEFT) {
00232                     game_state->scene.camera.pitch_offset_deg -= 1;
00233                 }
00234                 if (event.key.keysym.sym == SDLK_RIGHT) {
00235                     game_state->scene.camera.pitch_offset_deg += 1;
00236                 }
00237                 if (event.key.keysym.sym == SDLK_UP) {
00238                     game_state->scene.camera.roll_offset_deg += 1;
00239                     if (game_state->scene.camera.roll_offset_deg > 89) {
00240                         game_state->scene.camera.roll_offset_deg = 89;
00241                     }
00242                 }
00243                 if (event.key.keysym.sym == SDLK_DOWN) {
00244                     game_state->scene.camera.roll_offset_deg -= 1;
00245                     if (game_state->scene.camera.roll_offset_deg < -89) {
00246                         game_state->scene.camera.roll_offset_deg = -89;
00247                     }
00248                 }
00249                 if (event.key.keysym.sym == SDLK_r) {
00250                     ae_camera_reset_pos(&(game_state->scene));
00251                 }
00252                 break;
00253             case SDL_MOUSEBUTTONDOWN:
00254                 if (event.button.button == SDL_BUTTON_LEFT) {
00255                     game_state->left_button_pressed = 1;
00256                 }
00257                 break;
00258             case SDL_MOUSEBUTTONUP:
00259                 if (event.button.button == SDL_BUTTON_LEFT) {
```

```
00260                     game_state->left_button_pressed = 0;
00261                 }
00262                 break;
00263         }
00264     }
00265 }
00266
00267 void update_window(game_state_t *game_state)
00268 {
00269     SDL_GetWindowSize(game_state->window, &(game_state->window_w), &(game_state->window_h));
00270
00271     fix_framerate(game_state);
00272     game_state->elapsed_time += game_state->delta_time;
00273     game_state->fps = 1.0f / game_state->delta_time;
00274     game_state->frame_target_time = 1000/game_state->const_fps;
00275
00276     char fps_count[100];
00277     if (!game_state->to_limit_fps) {
00278         sprintf(fps_count, "dt = %5.02f [ms]", game_state->delta_time*1000);
00279     } else {
00280         sprintf(fps_count, "FPS = %5.2f", game_state->fps);
00281     }
00282
00283     if (game_state->elapsed_time*10-(int)(game_state->elapsed_time*10) < 0.1) {
00284         SDL_SetWindowTitle(game_state->window, fps_count);
00285     }
00286
00287     check_window_mat_size(game_state);
00288
00289     /*-------------------------------------------------------------------------*/
00290
00291     update(game_state);
00292
00293 }
00294
00295 void render_window(game_state_t *game_state)
00296 {
00297     if (game_state->to_clear_renderer) {
00298         // SDL_SetRenderDrawColor(game_state->renderer, HexARGB_RGBA(0xFF181818));
00299         // SDL_RenderClear(game_state->renderer);
00300         // mat2D_fill(game_state->window_pixels_mat, 0x181818);
00301         memset(game_state->window_pixels_mat.elements, 0x20, sizeof(uint32_t) *
      game_state->window_pixels_mat.rows * game_state->window_pixels_mat.cols);
00302         /* not using mat2D_fill but using memset because it is way faster, so the buffer needs to be
      of 1/z */
00303         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);
00304     }
00305     /*-------------------------------------------------------------------------*/
00306
00307     render(game_state);
00308
00309     /*-------------------------------------------------------------------------*/
00310
00311     copy_mat_to_surface_RGB(game_state);
00312     SDL_UpdateWindowSurface(game_state->window);
00313
00314 }
00315
00316 void destroy_window(game_state_t *game_state)
00317 {
00318     mat2D_free_uint32(game_state->window_pixels_mat);
00319     mat2D_free(game_state->inv_z_buffer_mat);
00320     ae_scene_free(&(game_state->scene));
00321
00322     if (game_state->window_surface) SDL_FreeSurface(game_state->window_surface);
00323     if (game_state->window_texture) SDL_DestroyTexture(game_state->window_texture);
00324
00325     SDL_DestroyRenderer(game_state->renderer);
00326     SDL_DestroyWindow(game_state->window);
00327
00328     SDL_Quit();
00329
00330     (void)game_state;
00331 }
00332
00333 void fix_framerate(game_state_t *game_state)
00334 {
00335     int time_ellapsed = SDL_GetTicks() - game_state->previous_frame_time;
00336     int time_to_wait = game_state->frame_target_time - time_ellapsed;
00337     if (time_to_wait > 0 && time_to_wait < game_state->frame_target_time) {
00338         if (game_state->to_limit_fps) {
00339             SDL_Delay(time_to_wait);
00340         }
00341     }
00342     game_state->delta_time = (SDL_GetTicks() - game_state->previous_frame_time) / 1000.0f;
00343     game_state->previous_frame_time = SDL_GetTicks();
```

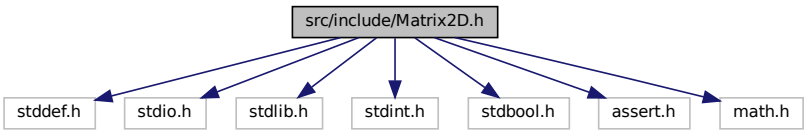
```
00344 }
00345
00346 #ifndef SETUP
00347 #define SETUP
00348 void setup(game_state_t *game_state) { (void)game_state; }
00349 #endif
00350
00351 #ifndef UPDATE
00352 #define UPDATE
00353 void update(game_state_t *game_state) { (void)game_state; }
00354 #endif
00355
00356 #ifndef RENDER
00357 #define RENDER
00358 void render(game_state_t *game_state) { (void)game_state; }
00359 #endif
00360
00361 void check_window_mat_size(game_state_t *game_state)
00362 {
00363     if (game_state->window_h != (int)game_state->window_pixels_mat.rows || game_state->window_w !=
      (int)game_state->window_pixels_mat.cols) {
00364         mat2D_free_uint32(game_state->window_pixels_mat);
00365         mat2D_free(game_state->inv_z_buffer_mat);
00366         SDL_FreeSurface(game_state->window_surface);
00367
00368         game_state->window_pixels_mat = mat2D_alloc_uint32(game_state->window_h,
      game_state->window_w);
00369         game_state->inv_z_buffer_mat = mat2D_alloc(game_state->window_h, game_state->window_w);
00370         game_state->scene.camera.aspect_ratio = (float)(game_state->window_h) /
      (float)(game_state->window_w);
00371
00372         game_state->window_surface = SDL_GetWindowSurface(game_state->window);
00373     }
00374 }
00375
00376 void copy_mat_to_surface_RGB(game_state_t *game_state)
00377 {
00378     SDL_LockSurface(game_state->window_surface);
00379
00380     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);
00381
00382     SDL_UnlockSurface(game_state->window_surface);
00383 }
00384
00385
```

4.13 src/include/Matrix2D.h File Reference

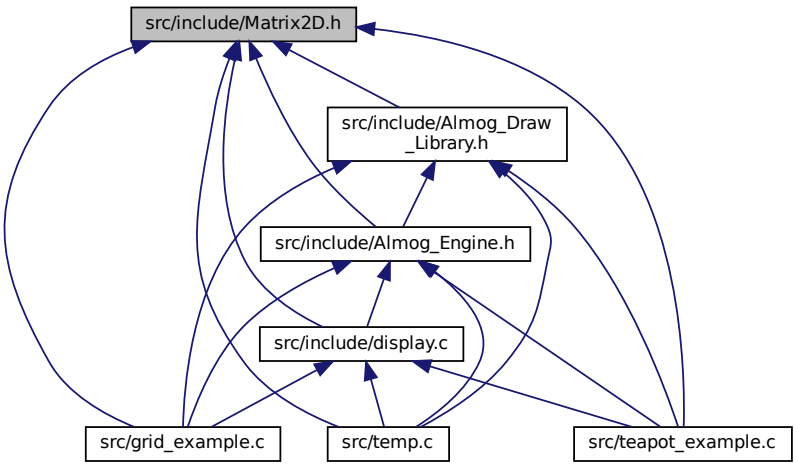
A single-header C library for simple 2D matrix operations on doubles and uint32_t, including allocation, basic arithmetic, linear algebra, and helpers (LUP, inverse, determinant, DCM, etc.).

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

Include dependency graph for Matrix2D.h:



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



Classes

- struct Mat2D

  Dense row-major matrix of doubles.
- struct Mat2D_uint32

  Dense row-major matrix of uint32_t.
- struct Mat2D_Minor

  A minor "view" into a reference matrix.

Macros

- #define MATRIX2D_MALLOC malloc

  Allocation function used by the library.
- #define MATRIX2D_ASSERT assert

  Assertion macro used by the library for parameter validation.
- #define MAT2D_AT(m, i, j) (m).elements[i ∗ m.stride_r + j]

  Access element (i, j) of a Mat2D (0-based).
- #define MAT2D_AT_UINT32(m, i, j) (m).elements[i ∗ m.stride_r + j]

  Access element (i, j) of a Mat2D_uint32 (0-based).
- #define __USE_MISC
- #define PI M_PI
- #define MAT2D_MINOR_AT(mm, i, j) MAT2D_AT(mm.ref_mat, mm.rows_list[i], mm.cols_list[j])

  Access element (i, j) of a Mat2D_Minor (0-based), dereferencing into the underlying reference matrix.
- #define MAT2D_PRINT(m) mat2D_print(m, #m, 0)

  Convenience macro to print a matrix with its variable name.
- #define MAT2D_PRINT_AS_COL(m) mat2D_print_as_col(m, #m, 0)

  Convenience macro to print a matrix as a single column with its name.
- #define MAT2D_MINOR_PRINT(mm) mat2D_minor_print(mm, #mm, 0)

  Convenience macro to print a minor with its variable name.
- #define mat2D_normalize(m) mat2D_mult((m), 1.0 / mat2D_calc_norma((m)))

  In-place normalization of all elements so that the Frobenius norm becomes 1.

Functions

- double mat2D_rand_double (void)

  Return a pseudo-random double in the range [0, 1].
- Mat2D mat2D_alloc (size_t rows, size_t cols)

  Allocate a rows x cols matrix of doubles.
- Mat2D_uint32 mat2D_alloc_uint32 (size_t rows, size_t cols)

  Allocate a rows x cols matrix of uint32_t.
- void mat2D_free (Mat2D m)

  Free the memory owned by a Mat2D (elements pointer).
- void mat2D_free_uint32 (Mat2D_uint32 m)

  Free the memory owned by a Mat2D_uint32 (elements pointer).
- size_t mat2D_offset2d (Mat2D m, size_t i, size_t j)

  Compute the linear offset of element (i, j) in a Mat2D.
- size_t mat2D_offset2d_uint32 (Mat2D_uint32 m, size_t i, size_t j)

  Compute the linear offset of element (i, j) in a Mat2D_uint32.
- void mat2D_fill (Mat2D m, double x)

  Fill all elements of a matrix of doubles with a scalar value.
- void mat2D_fill_sequence (Mat2D m, double start, double step)

  Fill a matrix with an arithmetic sequence laid out in row-major order.
- void mat2D_fill_uint32 (Mat2D_uint32 m, uint32_t x)

  Fill all elements of a matrix of uint32_t with a scalar value.
- void mat2D_rand (Mat2D m, double low, double high)

  Fill a matrix with random doubles in [low, high).
- void mat2D_dot (Mat2D dst, Mat2D a, Mat2D b)

  *Matrix product: dst = a * b.*
- double mat2D_dot_product (Mat2D a, Mat2D b)

  Dot product between two vectors.
- void mat2D_cross (Mat2D dst, Mat2D a, Mat2D b)

  3D cross product: dst = a x b for 3x1 vectors.
- void mat2D_add (Mat2D dst, Mat2D a)

  In-place addition: dst += a.
- void mat2D_add_row_time_factor_to_row (Mat2D m, size_t des_r, size_t src_r, double factor)

  *Row operation: row(des_r) += factor * row(src_r).*
- void mat2D_sub (Mat2D dst, Mat2D a)

  In-place subtraction: dst -= a.
- void mat2D_sub_row_time_factor_to_row (Mat2D m, size_t des_r, size_t src_r, double factor)

  *Row operation: row(des_r) -= factor * row(src_r).*
- void mat2D_mult (Mat2D m, double factor)

  *In-place scalar multiplication: m *= factor.*
- void mat2D_mult_row (Mat2D m, size_t r, double factor)

  *In-place row scaling: row(r) *= factor.*
- void mat2D_print (Mat2D m, const char *name, size_t padding)

  Print a matrix to stdout with a name and indentation padding.
- void mat2D_print_as_col (Mat2D m, const char *name, size_t padding)

  Print a matrix as a flattened column vector to stdout.
- void mat2D_set_identity (Mat2D m)

  Set a square matrix to the identity matrix.
- double mat2D_make_identity (Mat2D m)

  Reduce a matrix to identity via Gauss-Jordan elimination and return the cumulative scaling factor.
- void mat2D_set_rot_mat_x (Mat2D m, float angle_deg)

Set a 3x3 rotation matrix for rotation about the X-axis.

- void mat2D_set_rot_mat_y (Mat2D m, float angle_deg)

  Set a 3x3 rotation matrix for rotation about the Y-axis.
- void mat2D_set_rot_mat_z (Mat2D m, float angle_deg)

  Set a 3x3 rotation matrix for rotation about the Z-axis.
- void mat2D_set_DCM_zyx (Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg)

  Build a 3x3 direction cosine matrix (DCM) from Z-Y-X Euler angles.
- void mat2D_copy (Mat2D des, Mat2D src)

  Copy all elements from src to des.
- void mat2D_copy_mat_to_mat_at_window (Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)

  Copy a rectangular window from src into des.
- void mat2D_get_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)

  Copy a column from src into a column of des.
- void mat2D_add_col_to_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)

  Add a source column into a destination column: des[:, des_col] += src[:, src_col].
- void mat2D_sub_col_to_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)

  Subtract a source column from a destination column: des[:, des_col] -= src[:, src_col].
- void mat2D_swap_rows (Mat2D m, size_t r1, size_t r2)

  Swap two rows of a matrix in-place.
- void mat2D_get_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)

  Copy a row from src into a row of des.
- void mat2D_add_row_to_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)

  Add a source row into a destination row: des[des_row, :] += src[src_row, :].
- void mat2D_sub_row_to_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)

  Subtract a source row from a destination row: des[des_row, :] -= src[src_row, :].
- double mat2D_calc_norma (Mat2D m)

  Compute the Frobenius norm of a matrix, sqrt(sum(m_ij^2)).
- bool mat2D_mat_is_all_digit (Mat2D m, double digit)

  Check if all elements of a matrix equal a given digit.
- bool mat2D_row_is_all_digit (Mat2D m, double digit, size_t r)

  Check if all elements of a row equal a given digit.
- bool mat2D_col_is_all_digit (Mat2D m, double digit, size_t c)

  Check if all elements of a column equal a given digit.
- double mat2D_det_2x2_mat (Mat2D m)

  Determinant of a 2x2 matrix.
- double mat2D_triangulate (Mat2D m)

  Forward elimination to transform a matrix to upper triangular form.
- double mat2D_det (Mat2D m)

  Determinant of an NxN matrix via Gaussian elimination.
- void mat2D_LUP_decomposition_with_swap (Mat2D src, Mat2D l, Mat2D p, Mat2D u)

  Compute LUP decomposition: P∗A = L∗U with L unit diagonal.
- void mat2D_transpose (Mat2D des, Mat2D src)

  Transpose a matrix: des = src^T.
- void mat2D_invert (Mat2D des, Mat2D src)

  Invert a square matrix using Gauss-Jordan elimination.
- void mat2D_solve_linear_sys_LUP_decomposition (Mat2D A, Mat2D x, Mat2D B)

  Solve the linear system A x = B using LUP decomposition.
- Mat2D_Minor mat2D_minor_alloc_fill_from_mat (Mat2D ref_mat, size_t i, size_t j)

  Allocate a minor view by excluding row i and column j of ref_mat.
- Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor (Mat2D_Minor ref_mm, size_t i, size_t j)

  Allocate a nested minor view from an existing minor by excluding row i and column j of the minor.

- void mat2D_minor_free (Mat2D_Minor mm)

  Free the index arrays owned by a minor.
- void mat2D_minor_print (Mat2D_Minor mm, const char *name, size_t padding)

  Print a minor matrix to stdout with a name and indentation padding.
- double mat2D_det_2x2_mat_minor (Mat2D_Minor mm)

  Determinant of a 2x2 minor.
- double mat2D_minor_det (Mat2D_Minor mm)

  Determinant of a minor via recursive expansion by minors.

4.13.1 Detailed Description

A single-header C library for simple 2D matrix operations on doubles and uint32_t, including allocation, basic arithmetic, linear algebra, and helpers (LUP, inverse, determinant, DCM, etc.).

- Storage is contiguous row-major (C-style). The element at row i, column j (0-based) is located at `elements[i * stride_r + j]`.
- Dense matrices of `double` are represented by Mat2D, and dense matrices of `uint32_t` are represented by Mat2D_uint32.
- Some routines assert shape compatibility using MATRIX2D_ASSERT.
- Random number generation uses the C library `rand()`; it is not cryptographically secure.
- Inversion is done via Gauss-Jordan elimination with partial pivoting only when a pivot is zero; this can be numerically unstable for ill-conditioned matrices. See notes below.
- To compile the implementation, define MATRIX2D_IMPLEMENTATION in exactly one translation unit before including this header.

Example: #define MATRIX2D_IMPLEMENTATION #include "matrix2d.h"

Note

This one-file library is heavily inspired by Tsoding's nn.h implementation of matrix creation and operations: `https://github.com/tsoding/nn.h` and the video: `https://youtu.be/L1TbWe8b↩VOc?list=PLpM-Dvs8t0VZPZKggcql-MmjaBdZKeDMw`

Warning

Numerical stability:

- There is a set of functions for minors that can be used to compute the determinant, but that approach is factorial in complexity and too slow for larger matrices. This library uses Gaussian elimination instead.
- The inversion function can fail or be unstable if pivot values become very small. Consider preconditioning or using a more robust decomposition (e.g., full pivoting, SVD) for ill-conditioned problems.

Definition in file Matrix2D.h.

4.13.2 Macro Definition Documentation

4.13.2.1 __USE_MISC

```
#define __USE_MISC
```

Definition at line 151 of file Matrix2D.h.

4.13.2.2 MAT2D_AT

```
#define MAT2D_AT(
              m,
              i,
              j ) (m).elements[i * m.stride_r + j]
```

Access element (i, j) of a Mat2D (0-based).

Warning

This macro does not perform bounds checking in the fast configuration. Use carefully.

Definition at line 145 of file Matrix2D.h.

4.13.2.3 MAT2D_AT_UINT32

```
#define MAT2D_AT_UINT32(
              m,
              i,
              j ) (m).elements[i * m.stride_r + j]
```

Access element (i, j) of a Mat2D_uint32 (0-based).

Warning

This macro does not perform bounds checking in the fast configuration. Use carefully.

Definition at line 146 of file Matrix2D.h.

4.13.2.4 MAT2D_MINOR_AT

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

Access element (i, j) of a Mat2D_Minor (0-based), dereferencing into the underlying reference matrix.

Definition at line 162 of file Matrix2D.h.

4.13.2.5 MAT2D_MINOR_PRINT

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

Convenience macro to print a minor with its variable name.

Definition at line 177 of file Matrix2D.h.

4.13.2.6 mat2D_normalize

```
#define mat2D_normalize(
              m ) mat2D_mult((m), 1.0 / mat2D_calc_norma((m)))
```

In-place normalization of all elements so that the Frobenius norm becomes 1.

Equivalent to: m *= 1.0 / mat2D_calc_norma(m).

Definition at line 184 of file Matrix2D.h.

4.13.2.7 MAT2D_PRINT

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

Convenience macro to print a matrix with its variable name.

Definition at line 167 of file Matrix2D.h.

4.13.2.8 MAT2D_PRINT_AS_COL

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

Convenience macro to print a matrix as a single column with its name.

Definition at line 172 of file Matrix2D.h.

4.13.2.9 MATRIX2D_ASSERT

```
#define MATRIX2D_ASSERT assert
```

Assertion macro used by the library for parameter validation.

Defaults to C `assert`. Override by defining MATRIX2D_ASSERT before including this header if you want custom behavior.

Definition at line 68 of file Matrix2D.h.

4.13.2.10 MATRIX2D_MALLOC

```
#define MATRIX2D_MALLOC malloc
```

Allocation function used by the library.

Defaults to `malloc`. Override by defining MATRIX2D_MALLOC before including this header if you want to use a custom allocator.

Definition at line 56 of file Matrix2D.h.

4.13.2.11 PI

```
#define PI M_PI
```

Definition at line 154 of file Matrix2D.h.

4.13.3 Function Documentation

4.13.3.1 mat2D_add()

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

In-place addition: dst += a.

Parameters

dst	Destination matrix to be incremented.
a	Summand of same shape as dst.

Precondition

Shapes match.

Definition at line 496 of file Matrix2D.h.

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

Referenced by adl_arrow_draw(), ae_line_itersect_plane(), and ae_view_mat_set().

4.13.3.2 mat2D_add_col_to_col()

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

Add a source column into a destination column: des[:, des_col] += src[:, src_col].

Parameters

des	Destination matrix (same row count as src).
des_col	Column index in destination.
src	Source matrix.
src_col	Column index in source.

Definition at line 828 of file Matrix2D.h.

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

4.13.3.3 mat2D_add_row_time_factor_to_row()

```
void mat2D_add_row_time_factor_to_row (
            Mat2D m,
            size_t des_r,
            size_t src_r,
            double factor )
```

Row operation: row(des_r) += factor * row(src_r).

Parameters

m	Matrix.
des_r	Destination row index.
src_r	Source row index.
factor	Scalar multiplier.

Definition at line 514 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

4.13.3.4 mat2D_add_row_to_row()

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

Add a source row into a destination row: des[des_row, :] += src[src_row, :].

Parameters

des	Destination matrix (same number of columns as src).
des_row	Row index in destination.
src	Source matrix.
src_row	Row index in source.

Definition at line 897 of file Matrix2D.h.

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

4.13.3.5 mat2D_alloc()

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

Allocate a rows x cols matrix of doubles.

Parameters

rows	Number of rows (>= 1).
cols	Number of columns (>= 1).

Returns

A Mat2D with contiguous storage; must be freed with mat2D_free.

Postcondition

m.stride_r == cols.

Definition at line 278 of file Matrix2D.h.

References Mat2D::cols, Mat2D::elements, MATRIX2D_ASSERT, MATRIX2D_MALLOC, Mat2D::rows, and Mat2D::stride_r.

Referenced by adl_arrow_draw(), adl_figure_alloc(), ae_camera_init(), ae_curve_project_world2screen(), ae_line_clip_with_plane(), ae_line_itersect_plane(), ae_line_project_world2screen(), ae_point_project_view2screen(), ae_point_project_world2view(), ae_quad_calc_normal(), ae_quad_clip_with_plane(), ae_quad_mesh_project_world2screen(), ae_quad_project_world2screen(), ae_quad_set_normals(), ae_quad_transform_to_view(), ae_scene_init(), ae_tri_calc_normal(), ae_tri_clip_with_plane(), ae_tri_mesh_project_world2screen(), ae_tri_mesh_rotate_Euler_xyz(), ae_tri_project_world2screen(), ae_tri_set_normals(), ae_tri_transform_to_view(), ae_view_mat_set(), check_window_mat_size(), mat2D_det(), mat2D_invert(), mat2D_set_DCM_zyx(), mat2D_solve_linear_sys_LUP_decomposition(), and setup_window().

4.13.3.6 mat2D_alloc_uint32()

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

Allocate a rows x cols matrix of uint32_t.

Parameters

rows	Number of rows (>= 1).
cols	Number of columns (>= 1).

Returns

A Mat2D_uint32 with contiguous storage; free with mat2D_free_uint32.

Postcondition

m.stride_r == cols.

Definition at line 297 of file Matrix2D.h.

References Mat2D_uint32::cols, Mat2D_uint32::elements, MATRIX2D_ASSERT, MATRIX2D_MALLOC, Mat2D_uint32::rows, and Mat2D_uint32::stride_r.

Referenced by adl_figure_alloc(), check_window_mat_size(), and setup_window().

4.13.3.7 mat2D_calc_norma()

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

Compute the Frobenius norm of a matrix, sqrt(sum(m_ij^2)).

Parameters

m	Matrix.

Returns

Frobenius norm.

Definition at line 931 of file Matrix2D.h.

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

Referenced by ae_quad_calc_normal(), ae_tri_calc_normal(), and ae_view_mat_set().

4.13.3.8 mat2D_col_is_all_digit()

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

Check if all elements of a column equal a given digit.

Parameters

m	Matrix.
digit	Value to compare.
c	Column index.

Returns

true if every element equals digit, false otherwise.

Definition at line 985 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_det().

4.13.3.9 mat2D_copy()

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

Copy all elements from src to des.

Parameters

des	Destination matrix.
src	Source matrix.

Precondition

Shapes match.

Definition at line 768 of file Matrix2D.h.

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

Referenced by adl_arrow_draw(), ae_camera_init(), ae_camera_reset_pos(), ae_quad_set_normals(), ae_tri_set_normals(), ae_view_mat_set(), mat2D_det(), mat2D_invert(), and mat2D_LUP_decomposition_with_swap().

4.13.3.10 mat2D_copy_mat_to_mat_at_window()

```
void mat2D_copy_mat_to_mat_at_window (
             Mat2D des,
             Mat2D src,
             size_t is,
             size_t js,
             size_t ie,
             size_t je )
```

Copy a rectangular window from src into des.

Parameters

des	Destination matrix. Must have size (ie - is + 1) x (je - js + 1).
src	Source matrix.
is	Start row index in src (inclusive).
js	Start column index in src (inclusive).
ie	End row index in src (inclusive).
je	End column index in src (inclusive).

Precondition

0 <= is <= ie < src.rows, 0 <= js <= je < src.cols.

Definition at line 790 of file Matrix2D.h.

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

4.13.3.11 mat2D_cross()

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

3D cross product: dst = a x b for 3x1 vectors.

Parameters

dst	3x1 destination vector.
a	3x1 input vector.
b	3x1 input vector.

Precondition

All matrices have shape 3x1.

Definition at line 479 of file Matrix2D.h.

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

Referenced by ae_quad_calc_normal(), ae_quad_set_normals(), ae_tri_calc_normal(), ae_tri_set_normals(), and ae_view_mat_set().

4.13.3.12 mat2D_det()

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

Determinant of an NxN matrix via Gaussian elimination.

Parameters

m	Square matrix.

Returns

det(m).

Copies m internally, triangulates it, and returns the product of diagonal elements (adjusted by any scaling factor as implemented).

Definition at line 1052 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), MAT2D_AT, mat2D_col_is_all_digit(), mat2D_copy(), mat2D_free(), mat2D_row_is_all_digit(), mat2D_triangulate(), MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by mat2D_invert().

4.13.3.13 mat2D_det_2x2_mat()

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

Determinant of a 2x2 matrix.

Parameters

m	Matrix (must be 2x2).

Returns

det(m) = a11 a22 - a12 a21.

Definition at line 1000 of file Matrix2D.h.

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

4.13.3.14 mat2D_det_2x2_mat_minor()

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

Determinant of a 2x2 minor.

Parameters

mm	Minor (must be 2x2).

Returns

det(mm).

Definition at line 1383 of file Matrix2D.h.

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

Referenced by mat2D_minor_det().

4.13.3.15 mat2D_dot()

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

Matrix product: dst = a * b.

Parameters

dst	Destination matrix (size a.rows x b.cols).
a	Left matrix (size a.rows x a.cols).
b	Right matrix (size a.cols x b.cols).

Precondition

a.cols == b.rows, dst.rows == a.rows, dst.cols == b.cols.

Postcondition

dst is overwritten.

Definition at line 424 of file Matrix2D.h.

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

Referenced by adl_arrow_draw(), ae_point_project_view2screen(), ae_point_project_world2view(), ae_quad_transform_to_view(), ae_tri_mesh_rotate_Euler_xyz(), ae_tri_transform_to_view(), ae_view_mat_set(), mat2D_set_DCM_zyx(), and mat2D_solve_linear_sys_LUP_decomposition().

4.13.3.16 mat2D_dot_product()

```
double mat2D_dot_product (
            Mat2D a,
            Mat2D b )
```

Dot product between two vectors.

Parameters

a	Vector (shape n x 1 or 1 x n).
b	Vector (same shape as a).

Returns

The scalar dot product sum.

Precondition

a.rows == b.rows, a.cols == b.cols, and one dimension equals 1.

Definition at line 450 of file Matrix2D.h.

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

Referenced by ae_line_itersect_plane(), and ae_view_mat_set().

4.13.3.17 mat2D_fill()

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

Fill all elements of a matrix of doubles with a scalar value.

Parameters

m	Matrix to fill.
x	Value to assign to every element.

Definition at line 362 of file Matrix2D.h.

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

Referenced by adl_arrow_draw(), ae_camera_init(), ae_camera_reset_pos(), ae_curve_project_world2screen(), ae_line_itersect_plane(), ae_line_project_world2screen(), ae_projection_mat_set(), ae_quad_mesh_project_world2screen(), ae_quad_project_world2screen(), ae_scene_init(), ae_tri_mesh_project_world2screen(), ae_tri_project_world2screen(), ae_view_mat_set(), mat2D_invert(), mat2D_LUP_decomposition_with_swap(), and mat2D_solve_linear_sys_LUP_decomposition().

4.13.3.18 mat2D_fill_sequence()

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

Fill a matrix with an arithmetic sequence laid out in row-major order.

Parameters

m	Matrix to fill.
start	First value in the sequence.
step	Increment between consecutive elements.

Element at linear index k gets value start + step ∗ k.

Definition at line 378 of file Matrix2D.h.

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

4.13.3.19 mat2D_fill_uint32()

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

Fill all elements of a matrix of uint32_t with a scalar value.

Parameters

m	Matrix to fill.
x	Value to assign to every element.

Definition at line 391 of file Matrix2D.h.

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

Referenced by adl_2Dscalar_interp_on_figure(), and adl_curves_plot_on_figure().

4.13.3.20 mat2D_free()

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

Free the memory owned by a Mat2D (elements pointer).

Parameters

m	Matrix whose elements were allocated via MATRIX2D_MALLOC.

Note

Safe to call with m.elements == NULL.

Definition at line 314 of file Matrix2D.h.

References Mat2D::elements.

Referenced by adl_arrow_draw(), ae_camera_free(), ae_curve_project_world2screen(), ae_line_clip_with_plane(), ae_line_itersect_plane(), ae_line_project_world2screen(), ae_point_project_view2screen(), ae_point_project_world2view(), ae_quad_calc_normal(), ae_quad_clip_with_plane(), ae_quad_mesh_project_world2screen(), ae_quad_project_world2screen(), ae_quad_set_normals(), ae_quad_transform_to_view(), ae_scene_free(), ae_tri_calc_normal(), ae_tri_clip_with_plane(), ae_tri_mesh_project_world2screen(), ae_tri_mesh_rotate_Euler_xyz(), ae_tri_project_world2screen(), ae_tri_set_normals(), ae_tri_transform_to_view(), ae_view_mat_set(), check_window_mat_size(), destroy_window(), mat2D_det(), mat2D_invert(), mat2D_set_DCM_zyx(), and mat2D_solve_linear_sys_LUP_decomposition().

4.13.3.21 mat2D_free_uint32()

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

Free the memory owned by a Mat2D_uint32 (elements pointer).

Parameters

m	Matrix whose elements were allocated via MATRIX2D_MALLOC.

Note

Safe to call with m.elements == NULL.

Definition at line 324 of file Matrix2D.h.

References Mat2D_uint32::elements.

Referenced by check_window_mat_size(), and destroy_window().

4.13.3.22 mat2D_get_col()

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

Copy a column from src into a column of des.

Parameters

des	Destination matrix (same row count as src).
des_col	Column index in destination.
src	Source matrix.
src_col	Column index in source.

Definition at line 810 of file Matrix2D.h.

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

4.13.3.23 mat2D_get_row()

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

Copy a row from src into a row of des.

Parameters

des	Destination matrix (same number of columns as src).
des_row	Row index in destination.
src	Source matrix.
src_row	Row index in source.

Definition at line 879 of file Matrix2D.h.

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

4.13.3.24 mat2D_invert()

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

Invert a square matrix using Gauss-Jordan elimination.

Parameters

des	Destination matrix (same shape as src).
src	Source square matrix.

Precondition

src is square and nonsingular.

If det(src) == 0, prints an error and sets des to all zeros.

Warning

May be numerically unstable for ill-conditioned matrices.

Definition at line 1169 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), MAT2D_AT, mat2D_copy(), mat2D_det(), mat2D_fill(), mat2D_free(), mat2D_mult_row(), mat2D_set_identity(), mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by mat2D_solve_linear_sys_LUP_decomposition().

4.13.3.25 mat2D_LUP_decomposition_with_swap()

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

Compute LUP decomposition: P∗A = L∗U with L unit diagonal.

Parameters

src	Input matrix A (not modified).
l	Lower triangular matrix with unit diagonal (output).
p	Permutation matrix (output).
u	Upper triangular matrix (output).

Precondition

l, p, u are allocated to match src shape; src is square.

Definition at line 1107 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_copy(), mat2D_fill(), mat2D_set_identity(), mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), and Mat2D::rows.

Referenced by mat2D_solve_linear_sys_LUP_decomposition().

4.13.3.26 mat2D_make_identity()

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

Reduce a matrix to identity via Gauss-Jordan elimination and return the cumulative scaling factor.

Parameters

m	Matrix reduced in-place to identity (if nonsingular).

Returns

The product of row scaling factors applied during elimination.

Note

Intended as a helper for determinant-related operations.

Warning

Not robust to singular or ill-conditioned matrices.

Definition at line 643 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_mult_row(), mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), and Mat2D::rows.

4.13.3.27 mat2D_mat_is_all_digit()

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

Check if all elements of a matrix equal a given digit.

Parameters

m	Matrix.
digit	Value to compare.

Returns

true if every element equals digit, false otherwise.

Definition at line 949 of file Matrix2D.h.

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

4.13.3.28 mat2D_minor_alloc_fill_from_mat()

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

Allocate a minor view by excluding row i and column j of ref_mat.

Parameters

ref_mat	Reference square matrix.
i	Excluded row index in ref_mat.
j	Excluded column index in ref_mat.

Returns

A Mat2D_Minor that references ref_mat.

Note

Free rows_list and cols_list with mat2D_minor_free when done.

Definition at line 1279 of file Matrix2D.h.

References Mat2D::cols, Mat2D_Minor::cols, Mat2D_Minor::cols_list, MATRIX2D_ASSERT, MATRIX2D_MALLOC, Mat2D_Minor::ref_mat, Mat2D::rows, Mat2D_Minor::rows, Mat2D_Minor::rows_list, and Mat2D_Minor::stride_r.

4.13.3.29 mat2D_minor_alloc_fill_from_mat_minor()

```
Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor (
            Mat2D_Minor ref_mm,
            size_t i,
            size_t j )
```

Allocate a nested minor view from an existing minor by excluding row i and column j of the minor.

Parameters

ref_mm	Reference minor.
i	Excluded row index in the minor.
j	Excluded column index in the minor.

Returns

A new Mat2D_Minor that references the same underlying matrix.

Note

Free rows_list and cols_list with mat2D_minor_free when done.

Definition at line 1318 of file Matrix2D.h.

References Mat2D_Minor::cols, Mat2D_Minor::cols_list, MATRIX2D_ASSERT, MATRIX2D_MALLOC, Mat2D_Minor::ref_mat, Mat2D_Minor::rows, Mat2D_Minor::rows_list, and Mat2D_Minor::stride_r.

Referenced by mat2D_minor_det().

4.13.3.30 mat2D_minor_det()

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

Determinant of a minor via recursive expansion by minors.

Parameters

mm	Square minor.

Returns

det(mm).

Warning

Exponential complexity (factorial). Intended for educational or very small matrices only.

Definition at line 1396 of file Matrix2D.h.

References Mat2D_Minor::cols, mat2D_det_2x2_mat_minor(), mat2D_minor_alloc_fill_from_mat_minor(), MAT2D_MINOR_AT, mat2D_minor_free(), MATRIX2D_ASSERT, and Mat2D_Minor::rows.

4.13.3.31 mat2D_minor_free()

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

Free the index arrays owned by a minor.

Parameters

mm	Minor to free.

Note

After this call, mm.rows_list and mm.cols_list are invalid.

Definition at line 1353 of file Matrix2D.h.

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

Referenced by mat2D_minor_det().

4.13.3.32 mat2D_minor_print()

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

Print a minor matrix to stdout with a name and indentation padding.

Parameters

mm	Minor to print.
name	Label to print.
padding	Left padding in spaces.

Definition at line 1365 of file Matrix2D.h.

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

4.13.3.33 mat2D_mult()

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

In-place scalar multiplication: m *= factor.

Parameters

m	Matrix.
factor	Scalar multiplier.

Definition at line 557 of file Matrix2D.h.

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

Referenced by ae_line_itersect_plane(), ae_quad_calc_normal(), ae_tri_calc_normal(), and ae_view_mat_set().

4.13.3.34 mat2D_mult_row()

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

In-place row scaling: row(r) *= factor.

Parameters

m	Matrix.
r	Row index.
factor	Scalar multiplier.

Definition at line 572 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_invert(), and mat2D_make_identity().

4.13.3.35 mat2D_offset2d()

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

Compute the linear offset of element (i, j) in a Mat2D.

Parameters

m	Matrix.
i	Row index (0-based).
j	Column index (0-based).

Returns

The linear offset i ∗ stride_r + j.

Precondition

0 <= i < rows, 0 <= j < cols (asserted).

Definition at line 337 of file Matrix2D.h.

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

Referenced by mat2D_fill_sequence().

4.13.3.36 mat2D_offset2d_uint32()

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

Compute the linear offset of element (i, j) in a Mat2D_uint32.

Parameters

m	Matrix.
i	Row index (0-based).
j	Column index (0-based).

Returns

The linear offset i ∗ stride_r + j.

Precondition

0 <= i < rows, 0 <= j < cols (asserted).

Definition at line 351 of file Matrix2D.h.

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

4.13.3.37 mat2D_print()

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

Print a matrix to stdout with a name and indentation padding.

Parameters

m	Matrix to print.
name	Label to print.
padding	Left padding in spaces.

Definition at line 585 of file Matrix2D.h.

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

4.13.3.38 mat2D_print_as_col()

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

Print a matrix as a flattened column vector to stdout.

Parameters

m	Matrix to print (flattened in row-major).
name	Label to print.
padding	Left padding in spaces.

Definition at line 604 of file Matrix2D.h.

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

4.13.3.39 mat2D_rand()

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

Fill a matrix with random doubles in [low, high).

Parameters

m	Matrix to fill.
low	Lower bound (inclusive).
high	Upper bound (exclusive).

Precondition

high $>$ low.

Definition at line 407 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_rand_double(), and Mat2D::rows.

4.13.3.40 mat2D_rand_double()

```
double mat2D_rand_double (
            void  )
```

Return a pseudo-random double in the range [0, 1].

Note

Uses C library rand() and RAND_MAX. Not cryptographically secure.

Definition at line 266 of file Matrix2D.h.

Referenced by mat2D_rand().

4.13.3.41 mat2D_row_is_all_digit()

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

Check if all elements of a row equal a given digit.

Parameters

m	Matrix.
digit	Value to compare.
r	Row index.

Returns

true if every element equals digit, false otherwise.

Definition at line 968 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_det().

4.13.3.42 mat2D_set_DCM_zyx()

```
void mat2D_set_DCM_zyx (
            Mat2D DCM,
            float yaw_deg,
            float pitch_deg,
            float roll_deg )
```

Build a 3x3 direction cosine matrix (DCM) from Z-Y-X Euler angles.

Parameters

DCM	3x3 destination matrix.
yaw_deg	Rotation about Z in degrees.
pitch_deg	Rotation about Y in degrees.
roll_deg	Rotation about X in degrees.

Computes DCM = R_x(roll) ∗ R_y(pitch) ∗ R_z(yaw).

Definition at line 743 of file Matrix2D.h.

References mat2D_alloc(), mat2D_dot(), mat2D_free(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), and mat2D_set_rot_mat_z().

Referenced by ae_view_mat_set().

4.13.3.43 mat2D_set_identity()

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

Set a square matrix to the identity matrix.

Parameters

m	Matrix (must be square).

Precondition

m.rows == m.cols.

Definition at line 619 of file Matrix2D.h.

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

Referenced by mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), and mat2D_set_rot_mat_z().

4.13.3.44 mat2D_set_rot_mat_x()

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

Set a 3x3 rotation matrix for rotation about the X-axis.

Parameters

m	3x3 destination matrix.
angle_deg	Angle in degrees.

Definition at line 689 of file Matrix2D.h.

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

Referenced by ae_tri_mesh_rotate_Euler_xyz(), and mat2D_set_DCM_zyx().

4.13.3.45 mat2D_set_rot_mat_y()

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

Set a 3x3 rotation matrix for rotation about the Y-axis.

Parameters

m	3x3 destination matrix.
angle_deg	Angle in degrees.

Definition at line 706 of file Matrix2D.h.

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

Referenced by ae_tri_mesh_rotate_Euler_xyz(), and mat2D_set_DCM_zyx().

4.13.3.46 mat2D_set_rot_mat_z()

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

Set a 3x3 rotation matrix for rotation about the Z-axis.

Parameters

m	3x3 destination matrix.
angle_deg	Angle in degrees.

Definition at line 723 of file Matrix2D.h.

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

Referenced by adl_arrow_draw(), ae_tri_mesh_rotate_Euler_xyz(), and mat2D_set_DCM_zyx().

4.13.3.47 mat2D_solve_linear_sys_LUP_decomposition()

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

Solve the linear system A x = B using LUP decomposition.

Parameters

A	Coefficient matrix (NxN).
x	Solution vector (N x 1) (output).
B	Right-hand side vector (N x 1).

Internally computes LUP and uses explicit inverses of L and U.

Warning

Forming inverses explicitly can be less stable; a forward/backward substitution would be preferable for production-quality code.

Definition at line 1236 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), mat2D_dot(), mat2D_fill(), mat2D_free(), mat2D_invert(), mat2D_LUP_decomposition_with_s
MATRIX2D_ASSERT, and Mat2D::rows.

4.13.3.48 mat2D_sub()

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

In-place subtraction: dst -= a.

Parameters

dst	Destination matrix to be decremented.
a	Subtrahend of same shape as dst.

Precondition

Shapes match.

Definition at line 527 of file Matrix2D.h.

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

Referenced by adl_arrow_draw(), ae_line_itersect_plane(), ae_quad_calc_normal(), ae_quad_project_world2screen(), ae_quad_set_normals(), ae_tri_calc_normal(), ae_tri_project_world2screen(), ae_tri_set_normals(), and ae_view_mat_set().

4.13.3.49 mat2D_sub_col_to_col()

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

Subtract a source column from a destination column: des[:, des_col] -= src[:, src_col].

Parameters

des	Destination matrix (same row count as src).
des_col	Column index in destination.
src	Source matrix.
src_col	Column index in source.

Definition at line 846 of file Matrix2D.h.

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

4.13.3.50 mat2D_sub_row_time_factor_to_row()

```
void mat2D_sub_row_time_factor_to_row (
            Mat2D m,
            size_t des_r,
            size_t src_r,
            double factor )
```

Row operation: row(des_r) -= factor * row(src_r).

Parameters

m	Matrix.
des_r	Destination row index.
src_r	Source row index.
factor	Scalar multiplier.

Definition at line 545 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_make_identity(), and mat2D_triangulate().

4.13.3.51 mat2D_sub_row_to_row()

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

Subtract a source row from a destination row: des[des_row, :] -= src[src_row, :].

Parameters

des	Destination matrix (same number of columns as src).
des_row	Row index in destination.
src	Source matrix.
src_row	Row index in source.

Definition at line 915 of file Matrix2D.h.

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

4.13.3.52 mat2D_swap_rows()

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

Swap two rows of a matrix in-place.

Parameters

m	Matrix.
r1	First row index.
r2	Second row index.

Definition at line 863 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_make_identity(), and mat2D_triangulate().

4.13.3.53 mat2D_transpose()

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

Transpose a matrix: des = src^T.

Parameters

des	Destination matrix (shape src.cols x src.rows).
src	Source matrix.

Definition at line 1149 of file Matrix2D.h.

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

Referenced by ae_tri_project_world2screen(), and ae_view_mat_set().

4.13.3.54 mat2D_triangulate()

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

Forward elimination to transform a matrix to upper triangular form.

Parameters

m	Matrix transformed in-place.

Returns

Product of row scaling factors (currently 1 in this implementation).

Note

Used as part of determinant computation via triangularization.

Warning

Not robust for linearly dependent rows or tiny pivots.

Definition at line 1013 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), and Mat2D::rows.

Referenced by mat2D_det().

4.14 Matrix2D.h

```
00001
00039 #ifndef MATRIX2D_H_
00040 #define MATRIX2D_H_
00041
00042 #include <stddef.h>
00043 #include <stdio.h>
00044 #include <stdlib.h>
00045 #include <stdint.h>
00046 #include <stdbool.h>
00047
00055 #ifndef MATRIX2D_MALLOC
00056 #define MATRIX2D_MALLOC malloc
00057 #endif //MATRIX2D_MALLOC
00058
00066 #ifndef MATRIX2D_ASSERT
00067 #include <assert.h>
00068 #define MATRIX2D_ASSERT assert
00069 #endif //MATRIX2D_ASSERT
00070
00081 typedef struct {
00082     size_t rows;
00083     size_t cols;
00084     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00085     double *elements;
00086 } Mat2D;
00087
00098 typedef struct {
00099     size_t rows;
00100     size_t cols;
00101     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00102     uint32_t *elements;
00103 } Mat2D_uint32;
00104
00119 typedef struct {
00120     size_t rows;
00121     size_t cols;
00122     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00123     size_t *rows_list;
00124     size_t *cols_list;
00125     Mat2D ref_mat;
00126 } Mat2D_Minor;
00127
```

```
00141 #if 0
00142 #define MAT2D_AT(m, i, j) (m).elements[mat2D_offset2d((m), (i), (j))]
00143 #define MAT2D_AT_UINT32(m, i, j) (m).elements[mat2D_offset2d_uint32((m), (i), (j))]
00144 #else /* use this macro for batter performance but no assertion */
00145 #define MAT2D_AT(m, i, j) (m).elements[i * m.stride_r + j]
00146 #define MAT2D_AT_UINT32(m, i, j) (m).elements[i * m.stride_r + j]
00147 #endif
00148
00149 #ifndef PI
00150     #ifndef __USE_MISC
00151     #define __USE_MISC
00152     #endif
00153     #include <math.h>
00154     #define PI M_PI
00155 #endif
00156
00162 #define MAT2D_MINOR_AT(mm, i, j) MAT2D_AT(mm.ref_mat, mm.rows_list[i], mm.cols_list[j])
00167 #define MAT2D_PRINT(m) mat2D_print(m, #m, 0)
00172 #define MAT2D_PRINT_AS_COL(m) mat2D_print_as_col(m, #m, 0)
00177 #define MAT2D_MINOR_PRINT(mm) mat2D_minor_print(mm, #mm, 0)
00184 #define mat2D_normalize(m) mat2D_mult((m), 1.0 / mat2D_calc_norma((m)))
00185
00186 double mat2D_rand_double(void);
00187
00188 Mat2D mat2D_alloc(size_t rows, size_t cols);
00189 Mat2D_uint32 mat2D_alloc_uint32(size_t rows, size_t cols);
00190 void mat2D_free(Mat2D m);
00191 void mat2D_free_uint32(Mat2D_uint32 m);
00192 size_t mat2D_offset2d(Mat2D m, size_t i, size_t j);
00193 size_t mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j);
00194
00195 void mat2D_fill(Mat2D m, double x);
00196 void mat2D_fill_sequence(Mat2D m, double start, double step);
00197 void mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x);
00198 void mat2D_rand(Mat2D m, double low, double high);
00199
00200 void mat2D_dot(Mat2D dst, Mat2D a, Mat2D b);
00201 double mat2D_dot_product(Mat2D a, Mat2D b);
00202 void mat2D_cross(Mat2D dst, Mat2D a, Mat2D b);
00203
00204 void mat2D_add(Mat2D dst, Mat2D a);
00205 void mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00206
00207 void mat2D_sub(Mat2D dst, Mat2D a);
00208 void mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00209
00210 void mat2D_mult(Mat2D m, double factor);
00211 void mat2D_mult_row(Mat2D m, size_t r, double factor);
00212
00213 void mat2D_print(Mat2D m, const char *name, size_t padding);
00214 void mat2D_print_as_col(Mat2D m, const char *name, size_t padding);
00215
00216 void mat2D_set_identity(Mat2D m);
00217 double mat2D_make_identity(Mat2D m);
00218 void mat2D_set_rot_mat_x(Mat2D m, float angle_deg);
00219 void mat2D_set_rot_mat_y(Mat2D m, float angle_deg);
00220 void mat2D_set_rot_mat_z(Mat2D m, float angle_deg);
00221 void mat2D_set_DCM_zyx(Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg);
00222
00223 void mat2D_copy(Mat2D des, Mat2D src);
00224 void mat2D_copy_mat_to_mat_at_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t
      je);
00225
00226 void mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00227 void mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00228 void mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00229
00230 void mat2D_swap_rows(Mat2D m, size_t r1, size_t r2);
00231 void mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00232 void mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00233 void mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00234
00235 double mat2D_calc_norma(Mat2D m);
00236
00237 bool mat2D_mat_is_all_digit(Mat2D m, double digit);
00238 bool mat2D_row_is_all_digit(Mat2D m, double digit, size_t r);
00239 bool mat2D_col_is_all_digit(Mat2D m, double digit, size_t c);
00240
00241 double mat2D_det_2x2_mat(Mat2D m);
00242 double mat2D_triangulate(Mat2D m);
00243 double mat2D_det(Mat2D m);
00244 void mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u);
00245 void mat2D_transpose(Mat2D des, Mat2D src);
00246 void mat2D_invert(Mat2D des, Mat2D src);
00247 void mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B);
00248
00249 Mat2D_Minor mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j);
```

```
00250 Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j);
00251 void mat2D_minor_free(Mat2D_Minor mm);
00252 void mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding);
00253 double mat2D_det_2x2_mat_minor(Mat2D_Minor mm);
00254 double mat2D_minor_det(Mat2D_Minor mm);
00255
00256 #endif // MATRIX2D_H_
00257
00258 #ifdef MATRIX2D_IMPLEMENTATION
00259 #undef MATRIX2D_IMPLEMENTATION
00260
00261
00266 double mat2D_rand_double(void)
00267 {
00268     return (double) rand() / (double) RAND_MAX;
00269 }
00270
00278 Mat2D mat2D_alloc(size_t rows, size_t cols)
00279 {
00280     Mat2D m;
00281     m.rows = rows;
00282     m.cols = cols;
00283     m.stride_r = cols;
00284     m.elements = (double*)MATRIX2D_MALLOC(sizeof(double)*rows*cols);
00285     MATRIX2D_ASSERT(m.elements != NULL);
00286
00287     return m;
00288 }
00289
00297 Mat2D_uint32 mat2D_alloc_uint32(size_t rows, size_t cols)
00298 {
00299     Mat2D_uint32 m;
00300     m.rows = rows;
00301     m.cols = cols;
00302     m.stride_r = cols;
00303     m.elements = (uint32_t*)MATRIX2D_MALLOC(sizeof(uint32_t)*rows*cols);
00304     MATRIX2D_ASSERT(m.elements != NULL);
00305
00306     return m;
00307 }
00308
00314 void mat2D_free(Mat2D m)
00315 {
00316     free(m.elements);
00317 }
00318
00324 void mat2D_free_uint32(Mat2D_uint32 m)
00325 {
00326     free(m.elements);
00327 }
00328
00337 size_t mat2D_offset2d(Mat2D m, size_t i, size_t j)
00338 {
00339     MATRIX2D_ASSERT(i < m.rows && j < m.cols);
00340     return i * m.stride_r + j;
00341 }
00342
00351 size_t mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j)
00352 {
00353     MATRIX2D_ASSERT(i < m.rows && j < m.cols);
00354     return i * m.stride_r + j;
00355 }
00356
00362 void mat2D_fill(Mat2D m, double x)
00363 {
00364     for (size_t i = 0; i < m.rows; ++i) {
00365         for (size_t j = 0; j < m.cols; ++j) {
00366             MAT2D_AT(m, i, j) = x;
00367         }
00368     }
00369 }
00370
00378 void mat2D_fill_sequence(Mat2D m, double start, double step) {
00379     for (size_t i = 0; i < m.rows; i++) {
00380         for (size_t j = 0; j < m.cols; j++) {
00381             MAT2D_AT(m, i, j) = start + step * mat2D_offset2d(m, i, j);
00382         }
00383     }
00384 }
00385
00391 void mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x)
00392 {
00393     for (size_t i = 0; i < m.rows; ++i) {
00394         for (size_t j = 0; j < m.cols; ++j) {
00395             MAT2D_AT_UINT32(m, i, j) = x;
00396         }
00397     }
```

```
00398 }
00399
00407 void mat2D_rand(Mat2D m, double low, double high)
00408 {
00409     for (size_t i = 0; i < m.rows; ++i) {
00410         for (size_t j = 0; j < m.cols; ++j) {
00411             MAT2D_AT(m, i, j) = mat2D_rand_double()*(high - low) + low;
00412         }
00413     }
00414 }
00415
00424 void mat2D_dot(Mat2D dst, Mat2D a, Mat2D b)
00425 {
00426     MATRIX2D_ASSERT(a.cols == b.rows);
00427     MATRIX2D_ASSERT(a.rows == dst.rows);
00428     MATRIX2D_ASSERT(b.cols == dst.cols);
00429
00430     size_t i, j, k;
00431
00432     for (i = 0; i < dst.rows; i++) {
00433         for (j = 0; j < dst.cols; j++) {
00434             MAT2D_AT(dst, i, j) = 0;
00435             for (k = 0; k < a.cols; k++) {
00436                 MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, k)*MAT2D_AT(b, k, j);
00437             }
00438         }
00439     }
00440
00441 }
00442
00450 double mat2D_dot_product(Mat2D a, Mat2D b)
00451 {
00452     MATRIX2D_ASSERT(a.rows == b.rows);
00453     MATRIX2D_ASSERT(a.cols == b.cols);
00454     MATRIX2D_ASSERT((1 == a.cols && 1 == b.cols) || (1 == a.rows && 1 == b.rows));
00455
00456     double dot_product = 0;
00457
00458     if (1 == a.cols) {
00459         for (size_t i = 0; i < a.rows; i++) {
00460             dot_product += MAT2D_AT(a, i, 0) * MAT2D_AT(b, i, 0);
00461         }
00462     } else {
00463         for (size_t j = 0; j < a.cols; j++) {
00464             dot_product += MAT2D_AT(a, 0, j) * MAT2D_AT(b, 0, j);
00465         }
00466     }
00467
00468     return dot_product;
00469
00470 }
00471
00479 void mat2D_cross(Mat2D dst, Mat2D a, Mat2D b)
00480 {
00481     MATRIX2D_ASSERT(3 == dst.rows && 1 == dst.cols);
00482     MATRIX2D_ASSERT(3 == a.rows && 1 == a.cols);
00483     MATRIX2D_ASSERT(3 == b.rows && 1 == b.cols);
00484
00485     MAT2D_AT(dst, 0, 0) = MAT2D_AT(a, 1, 0) * MAT2D_AT(b, 2, 0) - MAT2D_AT(a, 2, 0) * MAT2D_AT(b, 1,
      0);
00486     MAT2D_AT(dst, 1, 0) = MAT2D_AT(a, 2, 0) * MAT2D_AT(b, 0, 0) - MAT2D_AT(a, 0, 0) * MAT2D_AT(b, 2,
      0);
00487     MAT2D_AT(dst, 2, 0) = MAT2D_AT(a, 0, 0) * MAT2D_AT(b, 1, 0) - MAT2D_AT(a, 1, 0) * MAT2D_AT(b, 0,
      0);
00488 }
00489
00496 void mat2D_add(Mat2D dst, Mat2D a)
00497 {
00498     MATRIX2D_ASSERT(dst.rows == a.rows);
00499     MATRIX2D_ASSERT(dst.cols == a.cols);
00500     for (size_t i = 0; i < dst.rows; ++i) {
00501         for (size_t j = 0; j < dst.cols; ++j) {
00502             MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, j);
00503         }
00504     }
00505 }
00506
00514 void mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
00515 {
00516     for (size_t j = 0; j < m.cols; ++j) {
00517         MAT2D_AT(m, des_r, j) += factor * MAT2D_AT(m, src_r, j);
00518     }
00519 }
00520
00527 void mat2D_sub(Mat2D dst, Mat2D a)
00528 {
00529     MATRIX2D_ASSERT(dst.rows == a.rows);
```

```
00530     MATRIX2D_ASSERT(dst.cols == a.cols);
00531     for (size_t i = 0; i < dst.rows; ++i) {
00532         for (size_t j = 0; j < dst.cols; ++j) {
00533             MAT2D_AT(dst, i, j) -= MAT2D_AT(a, i, j);
00534         }
00535     }
00536 }
00537
00545 void mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
00546 {
00547     for (size_t j = 0; j < m.cols; ++j) {
00548         MAT2D_AT(m, des_r, j) -= factor * MAT2D_AT(m, src_r, j);
00549     }
00550 }
00551
00557 void mat2D_mult(Mat2D m, double factor)
00558 {
00559     for (size_t i = 0; i < m.rows; ++i) {
00560         for (size_t j = 0; j < m.cols; ++j) {
00561             MAT2D_AT(m, i, j) *= factor;
00562         }
00563     }
00564 }
00565
00572 void mat2D_mult_row(Mat2D m, size_t r, double factor)
00573 {
00574     for (size_t j = 0; j < m.cols; ++j) {
00575         MAT2D_AT(m, r, j) *= factor;
00576     }
00577 }
00578
00585 void mat2D_print(Mat2D m, const char *name, size_t padding)
00586 {
00587     printf("%*s%s = [\n", (int) padding, "", name);
00588     for (size_t i = 0; i < m.rows; ++i) {
00589         printf("%*s    ", (int) padding, "");
00590         for (size_t j = 0; j < m.cols; ++j) {
00591             printf("%9.6f ", MAT2D_AT(m, i, j));
00592         }
00593         printf("\n");
00594     }
00595     printf("%*s]\n", (int) padding, "");
00596 }
00597
00604 void mat2D_print_as_col(Mat2D m, const char *name, size_t padding)
00605 {
00606     printf("%*s%s = [\n", (int) padding, "", name);
00607     for (size_t i = 0; i < m.rows*m.cols; ++i) {
00608             printf("%*s    ", (int) padding, "");
00609             printf("%f\n", m.elements[i]);
00610     }
00611     printf("%*s]\n", (int) padding, "");
00612 }
00613
00619 void mat2D_set_identity(Mat2D m)
00620 {
00621     MATRIX2D_ASSERT(m.cols == m.rows);
00622     for (size_t i = 0; i < m.rows; ++i) {
00623         for (size_t j = 0; j < m.cols; ++j) {
00624             MAT2D_AT(m, i, j) = i == j ? 1 : 0;
00625             // if (i == j) {
00626             //     MAT2D_AT(m, i, j) = 1;
00627             // }
00628             // else {
00629             //     MAT2D_AT(m, i, j) = 0;
00630             // }
00631         }
00632     }
00633 }
00634
00643 double mat2D_make_identity(Mat2D m)
00644 {
00645     /* make identity matrix using Gauss elimination */
00646     /* preforming Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
00647     /* returns the factor multiplying the determinant */
00648
00649     double factor_to_return = 1;
00650
00651     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
00652         /* check if it is the biggest first number (absolute value) */
00653         size_t biggest_r = i;
00654         for (size_t index = i; index < m.rows; index++) {
00655             if (fabs(MAT2D_AT(m, index, index)) > fabs(MAT2D_AT(m, biggest_r, 0))) {
00656                 biggest_r = index;
00657             }
00658         }
00659         if (i != biggest_r) {
```

```
00660             mat2D_swap_rows(m, i, biggest_r);
00661             factor_to_return *= -1;
00662         }
00663         for (size_t j = i+1; j < m.cols; j++) {
00664             double factor = 1 / MAT2D_AT(m, i, i);
00665             mat2D_sub_row_time_factor_to_row(m, j, i, MAT2D_AT(m, j, i) * factor);
00666             mat2D_mult_row(m, i, factor);
00667             factor_to_return *= factor;
00668         }
00669     }
00670     double factor = 1 / MAT2D_AT(m, m.rows-1, m.cols-1);
00671     mat2D_mult_row(m, m.rows-1, factor);
00672     factor_to_return *= factor;
00673     for (size_t c = m.cols-1; c > 0; c--) {
00674         for (int r = c-1; r >= 0; r--) {
00675             double factor = 1 / MAT2D_AT(m, c, c);
00676             mat2D_sub_row_time_factor_to_row(m, r, c, MAT2D_AT(m, r, c) * factor);
00677         }
00678     }
00679
00680
00681     return factor_to_return;
00682 }
00683
00689 void mat2D_set_rot_mat_x(Mat2D m, float angle_deg)
00690 {
00691     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00692
00693     float angle_rad = angle_deg * PI / 180;
00694     mat2D_set_identity(m);
00695     MAT2D_AT(m, 1, 1) =  cos(angle_rad);
00696     MAT2D_AT(m, 1, 2) =  sin(angle_rad);
00697     MAT2D_AT(m, 2, 1) = -sin(angle_rad);
00698     MAT2D_AT(m, 2, 2) =  cos(angle_rad);
00699 }
00700
00706 void mat2D_set_rot_mat_y(Mat2D m, float angle_deg)
00707 {
00708     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00709
00710     float angle_rad = angle_deg * PI / 180;
00711     mat2D_set_identity(m);
00712     MAT2D_AT(m, 0, 0) =  cos(angle_rad);
00713     MAT2D_AT(m, 0, 2) = -sin(angle_rad);
00714     MAT2D_AT(m, 2, 0) =  sin(angle_rad);
00715     MAT2D_AT(m, 2, 2) =  cos(angle_rad);
00716 }
00717
00723 void mat2D_set_rot_mat_z(Mat2D m, float angle_deg)
00724 {
00725     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00726
00727     float angle_rad = angle_deg * PI / 180;
00728     mat2D_set_identity(m);
00729     MAT2D_AT(m, 0, 0) =  cos(angle_rad);
00730     MAT2D_AT(m, 0, 1) =  sin(angle_rad);
00731     MAT2D_AT(m, 1, 0) = -sin(angle_rad);
00732     MAT2D_AT(m, 1, 1) =  cos(angle_rad);
00733 }
00734
00743 void mat2D_set_DCM_zyx(Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg)
00744 {
00745     Mat2D RotZ = mat2D_alloc(3,3);
00746     mat2D_set_rot_mat_z(RotZ, yaw_deg);
00747     Mat2D RotY = mat2D_alloc(3,3);
00748     mat2D_set_rot_mat_y(RotY, pitch_deg);
00749     Mat2D RotX = mat2D_alloc(3,3);
00750     mat2D_set_rot_mat_x(RotX, roll_deg);
00751     Mat2D temp = mat2D_alloc(3,3);
00752
00753     mat2D_dot(temp, RotY, RotZ);
00754     mat2D_dot(DCM, RotX, temp); /* I have a DCM */
00755
00756     mat2D_free(RotZ);
00757     mat2D_free(RotY);
00758     mat2D_free(RotX);
00759     mat2D_free(temp);
00760 }
00761
00768 void mat2D_copy(Mat2D des, Mat2D src)
00769 {
00770     MATRIX2D_ASSERT(des.cols == src.cols);
00771     MATRIX2D_ASSERT(des.rows == src.rows);
00772
00773     for (size_t i = 0; i < des.rows; ++i) {
00774         for (size_t j = 0; j < des.cols; ++j) {
00775             MAT2D_AT(des, i, j) = MAT2D_AT(src, i, j);
```

```
00776         }
00777     }
00778 }
00779
00790 void mat2D_copy_mat_to_mat_at_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)
00791 {
00792     MATRIX2D_ASSERT(je > js && ie > is);
00793     MATRIX2D_ASSERT(je-js+1 == des.cols);
00794     MATRIX2D_ASSERT(ie-is+1 == des.rows);
00795
00796     for (size_t index = 0; index < des.rows; ++index) {
00797         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
00798             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, is+index, js+jndex);
00799         }
00800     }
00801 }
00802
00810 void mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00811 {
00812     MATRIX2D_ASSERT(src_col < src.cols);
00813     MATRIX2D_ASSERT(des.rows == src.rows);
00814     MATRIX2D_ASSERT(des_col < des.cols);
00815
00816     for (size_t i = 0; i < des.rows; i++) {
00817         MAT2D_AT(des, i, des_col) = MAT2D_AT(src, i, src_col);
00818     }
00819 }
00820
00828 void mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00829 {
00830     MATRIX2D_ASSERT(src_col < src.cols);
00831     MATRIX2D_ASSERT(des.rows == src.rows);
00832     MATRIX2D_ASSERT(des_col < des.cols);
00833
00834     for (size_t i = 0; i < des.rows; i++) {
00835         MAT2D_AT(des, i, des_col) += MAT2D_AT(src, i, src_col);
00836     }
00837 }
00838
00846 void mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00847 {
00848     MATRIX2D_ASSERT(src_col < src.cols);
00849     MATRIX2D_ASSERT(des.rows == src.rows);
00850     MATRIX2D_ASSERT(des_col < des.cols);
00851
00852     for (size_t i = 0; i < des.rows; i++) {
00853         MAT2D_AT(des, i, des_col) -= MAT2D_AT(src, i, src_col);
00854     }
00855 }
00856
00863 void mat2D_swap_rows(Mat2D m, size_t r1, size_t r2)
00864 {
00865     for (size_t j = 0; j < m.cols; j++) {
00866         double temp = MAT2D_AT(m, r1, j);
00867         MAT2D_AT(m, r1, j) = MAT2D_AT(m, r2, j);
00868         MAT2D_AT(m, r2, j) = temp;
00869     }
00870 }
00871
00879 void mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00880 {
00881     MATRIX2D_ASSERT(src_row < src.rows);
00882     MATRIX2D_ASSERT(des.cols == src.cols);
00883     MATRIX2D_ASSERT(des_row < des.rows);
00884
00885     for (size_t j = 0; j < des.cols; j++) {
00886         MAT2D_AT(des, des_row, j) = MAT2D_AT(src, src_row, j);
00887     }
00888 }
00889
00897 void mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00898 {
00899     MATRIX2D_ASSERT(src_row < src.rows);
00900     MATRIX2D_ASSERT(des.cols == src.cols);
00901     MATRIX2D_ASSERT(des_row < des.rows);
00902
00903     for (size_t j = 0; j < des.cols; j++) {
00904         MAT2D_AT(des, des_row, j) += MAT2D_AT(src, src_row, j);
00905     }
00906 }
00907
00915 void mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00916 {
00917     MATRIX2D_ASSERT(src_row < src.rows);
00918     MATRIX2D_ASSERT(des.cols == src.cols);
00919     MATRIX2D_ASSERT(des_row < des.rows);
00920
```

```
00921     for (size_t j = 0; j < des.cols; j++) {
00922         MAT2D_AT(des, des_row, j) -= MAT2D_AT(src, src_row, j);
00923     }
00924 }
00925
00931 double mat2D_calc_norma(Mat2D m)
00932 {
00933     double sum = 0;
00934
00935     for (size_t i = 0; i < m.rows; ++i) {
00936         for (size_t j = 0; j < m.cols; ++j) {
00937             sum += MAT2D_AT(m, i, j) * MAT2D_AT(m, i, j);
00938         }
00939     }
00940     return sqrt(sum);
00941 }
00942
00949 bool mat2D_mat_is_all_digit(Mat2D m, double digit)
00950 {
00951     for (size_t i = 0; i < m.rows; ++i) {
00952         for (size_t j = 0; j < m.cols; ++j) {
00953             if (MAT2D_AT(m, i, j) != digit) {
00954                 return false;
00955             }
00956         }
00957     }
00958     return true;
00959 }
00960
00968 bool mat2D_row_is_all_digit(Mat2D m, double digit, size_t r)
00969 {
00970     for (size_t j = 0; j < m.cols; ++j) {
00971         if (MAT2D_AT(m, r, j) != digit) {
00972             return false;
00973         }
00974     }
00975     return true;
00976 }
00977
00985 bool mat2D_col_is_all_digit(Mat2D m, double digit, size_t c)
00986 {
00987     for (size_t i = 0; i < m.cols; ++i) {
00988         if (MAT2D_AT(m, i, c) != digit) {
00989             return false;
00990         }
00991     }
00992     return true;
00993 }
00994
01000 double mat2D_det_2x2_mat(Mat2D m)
01001 {
01002     MATRIX2D_ASSERT(2 == m.cols && 2 == m.rows && "Not a 2x2 matrix");
01003     return MAT2D_AT(m, 0, 0) * MAT2D_AT(m, 1, 1) - MAT2D_AT(m, 0, 1) * MAT2D_AT(m, 1, 0);
01004 }
01005
01013 double mat2D_triangulate(Mat2D m)
01014 {
01015     /* preforming Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
01016     /* returns the factor multiplying the determinant */
01017
01018     double factor_to_return = 1;
01019
01020     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
01021         if (!MAT2D_AT(m, i, i)) {   /* swapping only if it is zero */
01022             /* finding biggest first number (absolute value) */
01023             size_t biggest_r = i;
01024             for (size_t index = i; index < m.rows; index++) {
01025                 if (fabs(MAT2D_AT(m, index, i)) > fabs(MAT2D_AT(m, biggest_r, i))) {
01026                     biggest_r = index;
01027                 }
01028             }
01029             if (i != biggest_r) {
01030                 mat2D_swap_rows(m, i, biggest_r);
01031             }
01032         }
01033         for (size_t j = i+1; j < m.cols; j++) {
01034             double factor = 1 / MAT2D_AT(m, i, i);
01035             if (!isfinite(factor)) {
01036                 printf("%s:%d: [Error] unable to transfrom into uperr triangular matrix. Probably some
      of the rows are not independent.\n", __FILE__, __LINE__);
01037             }
01038             double mat_value = MAT2D_AT(m, j, i);
01039             mat2D_sub_row_time_factor_to_row(m, j, i, mat_value * factor);
01040         }
01041     }
01042     return factor_to_return;
01043 }
```

```
01044
01052 double mat2D_det(Mat2D m)
01053 {
01054     MATRIX2D_ASSERT(m.cols == m.rows && "should be a square matrix");
01055
01056     /* checking if there is a row or column with all zeros */
01057     /* checking rows */
01058     for (size_t i = 0; i < m.rows; i++) {
01059         if (mat2D_row_is_all_digit(m, 0, i)) {
01060             return 0;
01061         }
01062     }
01063     /* checking cols */
01064     for (size_t j = 0; j < m.rows; j++) {
01065         if (mat2D_col_is_all_digit(m, 0, j)) {
01066             return 0;
01067         }
01068     }
01069
01070     /* This is an implementation of naive determinant calculation using minors. This is too slow */
01071
01072     // double det = 0;
01073     // /* TODO: finding beast row or col? */
01074     // for (size_t i = 0, j = 0; i < m.rows; i++) { /* first column */
01075     //     if (MAT2D_AT(m, i, j) < 1e-10) continue;
01076     //     Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat(m, i, j);
01077     //     int factor = (i+j)%2 ? -1 : 1;
01078     //     if (sub_mm.cols != 2) {
01079     //         MATRIX2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
01080     //         det += MAT2D_AT(m, i, j) * (factor) * mat2D_minor_det(sub_mm);
01081     //     } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
01082     //         det += MAT2D_AT(m, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);;
01083     //     }
01084     //     mat2D_minor_free(sub_mm);
01085     // }
01086
01087     Mat2D temp_m = mat2D_alloc(m.rows, m.cols);
01088     mat2D_copy(temp_m, m);
01089     double factor = mat2D_triangulate(temp_m);
01090     double diag_mul = 1;
01091     for (size_t i = 0; i < temp_m.rows; i++) {
01092         diag_mul *= MAT2D_AT(temp_m, i, i);
01093     }
01094     mat2D_free(temp_m);
01095
01096     return diag_mul / factor;
01097 }
01098
01107 void mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u)
01108 {
01109     /* performing LU decomposition Following the Wikipedia page:
      https://en.wikipedia.org/wiki/LU_decomposition */
01110
01111     mat2D_copy(u, src);
01112     mat2D_set_identity(p);
01113     mat2D_fill(l, 0);
01114
01115     for (size_t i = 0; i < (size_t)fmin(u.rows-1, u.cols); i++) {
01116         if (!MAT2D_AT(u, i, i)) {   /* swapping only if it is zero */
01117             /* finding biggest first number (absolute value) */
01118             size_t biggest_r = i;
01119             for (size_t index = i; index < u.rows; index++) {
01120                 if (fabs(MAT2D_AT(u, index, i)) > fabs(MAT2D_AT(u, biggest_r, i))) {
01121                     biggest_r = index;
01122                 }
01123             }
01124             if (i != biggest_r) {
01125                 mat2D_swap_rows(u, i, biggest_r);
01126                 mat2D_swap_rows(p, i, biggest_r);
01127                 mat2D_swap_rows(l, i, biggest_r);
01128             }
01129         }
01130         for (size_t j = i+1; j < u.cols; j++) {
01131             double factor = 1 / MAT2D_AT(u, i, i);
01132             if (!isfinite(factor)) {
01133                 printf("%s:%d: [Error] unable to transfrom into uper triangular matrix. Probably some
      of the rows are not independent.\n", __FILE__, __LINE__);
01134             }
01135             double mat_value = MAT2D_AT(u, j, i);
01136             mat2D_sub_row_time_factor_to_row(u, j, i, mat_value * factor);
01137             MAT2D_AT(l, j, i) = mat_value * factor;
01138         }
01139         MAT2D_AT(l, i, i) = 1;
01140     }
01141     MAT2D_AT(l, l.rows-1, l.cols-1) = 1;
01142 }
01143
```

```
01149 void mat2D_transpose(Mat2D des, Mat2D src)
01150 {
01151     MATRIX2D_ASSERT(des.cols == src.rows);
01152     MATRIX2D_ASSERT(des.rows == src.cols);
01153 
01154     for (size_t index = 0; index < des.rows; ++index) {
01155         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
01156             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, jndex, index);
01157         }
01158     }
01159 }
01160 
01169 void mat2D_invert(Mat2D des, Mat2D src)
01170 {
01171     MATRIX2D_ASSERT(src.cols == src.rows && "should be an NxN matrix");
01172     MATRIX2D_ASSERT(des.cols == src.cols && des.rows == des.cols);
01173 
01174     Mat2D m = mat2D_alloc(src.rows, src.cols);
01175     mat2D_copy(m, src);
01176 
01177     mat2D_set_identity(des);
01178 
01179     if (!mat2D_det(m)) {
01180         mat2D_fill(des, 0);
01181         printf("%s:%d: [Error] Can't invert the matrix. Determinant is zero! Set the inverse matrix to
      all zeros\n", __FILE__, __LINE__);
01182         return;
01183     }
01184 
01185     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
01186         if (!MAT2D_AT(m, i, i)) {   /* swapping only if it is zero */
01187             /* finding biggest first number (absolute value) */
01188             size_t biggest_r = i;
01189             for (size_t index = i; index < m.rows; index++) {
01190                 if (fabs(MAT2D_AT(m, index, i)) > fabs(MAT2D_AT(m, biggest_r, i))) {
01191                     biggest_r = index;
01192                 }
01193             }
01194             if (i != biggest_r) {
01195                 mat2D_swap_rows(m, i, biggest_r);
01196                 mat2D_swap_rows(des, i, biggest_r);
01197                 printf("%s:%d: [INFO] swapping row %zu with row %zu.\n", __FILE__, __LINE__, i,
      biggest_r);
01198             } else {
01199                 MATRIX2D_ASSERT(0 && "can't inverse");
01200             }
01201         }
01202         for (size_t j = i+1; j < m.cols; j++) {
01203             double factor = 1 / MAT2D_AT(m, i, i);
01204             double mat_value = MAT2D_AT(m, j, i);
01205             mat2D_sub_row_time_factor_to_row(m, j, i, mat_value * factor);
01206             mat2D_mult_row(m, i, factor);
01207 
01208             mat2D_sub_row_time_factor_to_row(des, j, i, mat_value * factor);
01209             mat2D_mult_row(des, i, factor);
01210         }
01211     }
01212     double factor = 1 / MAT2D_AT(m, m.rows-1, m.cols-1);
01213     mat2D_mult_row(m, m.rows-1, factor);
01214     mat2D_mult_row(des, des.rows-1, factor);
01215     for (size_t c = m.cols-1; c > 0; c--) {
01216         for (int r = c-1; r >= 0; r--) {
01217             double factor = 1 / MAT2D_AT(m, c, c);
01218             double mat_value = MAT2D_AT(m, r, c);
01219             mat2D_sub_row_time_factor_to_row(m, r, c, mat_value * factor);
01220             mat2D_sub_row_time_factor_to_row(des, r, c, mat_value * factor);
01221         }
01222     }
01223 
01224     mat2D_free(m);
01225 }
01226 
01236 void mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B)
01237 {
01238     MATRIX2D_ASSERT(A.cols == x.rows);
01239     MATRIX2D_ASSERT(1 == x.cols);
01240     MATRIX2D_ASSERT(A.rows == B.rows);
01241     MATRIX2D_ASSERT(1 == B.cols);
01242 
01243     Mat2D y     = mat2D_alloc(x.rows, x.cols);
01244     Mat2D l     = mat2D_alloc(A.rows, A.cols);
01245     Mat2D p     = mat2D_alloc(A.rows, A.cols);
01246     Mat2D u     = mat2D_alloc(A.rows, A.cols);
01247     Mat2D inv_l = mat2D_alloc(l.rows, l.cols);
01248     Mat2D inv_u = mat2D_alloc(u.rows, u.cols);
01249 
01250     mat2D_LUP_decomposition_with_swap(A, l, p, u);
```

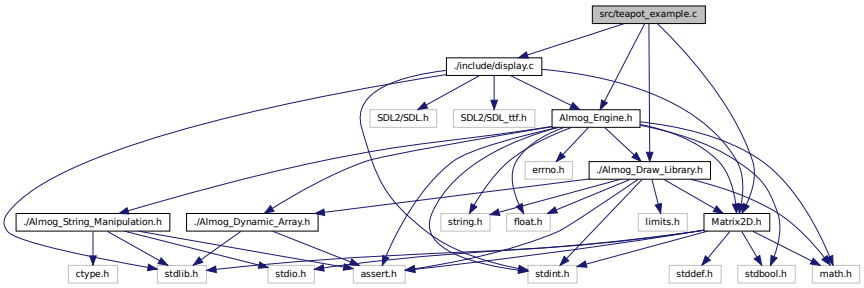
```
01251
01252     mat2D_invert(inv_l, l);
01253     mat2D_invert(inv_u, u);
01254
01255     mat2D_fill(x, 0);    /* x here is only a temp mat*/
01256     mat2D_fill(y, 0);
01257     mat2D_dot(x, p, B);
01258     mat2D_dot(y, inv_l, x);
01259
01260     mat2D_fill(x, 0);
01261     mat2D_dot(x, inv_u, y);
01262
01263     mat2D_free(y);
01264     mat2D_free(l);
01265     mat2D_free(p);
01266     mat2D_free(u);
01267     mat2D_free(inv_l);
01268     mat2D_free(inv_u);
01269 }
01270
01279 Mat2D_Minor mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j)
01280 {
01281     MATRIX2D_ASSERT(ref_mat.cols == ref_mat.rows && "minor is defined only for square matrix");
01282
01283     Mat2D_Minor mm;
01284     mm.cols = ref_mat.cols-1;
01285     mm.rows = ref_mat.rows-1;
01286     mm.stride_r = ref_mat.cols-1;
01287     mm.cols_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mat.cols-1));
01288     mm.rows_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mat.rows-1));
01289     mm.ref_mat = ref_mat;
01290
01291     MATRIX2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
01292
01293     for (size_t index = 0, temp_index = 0; index < ref_mat.rows; index++) {
01294         if (index != i) {
01295             mm.rows_list[temp_index] = index;
01296             temp_index++;
01297         }
01298     }
01299     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mat.rows; jndex++) {
01300         if (jndex != j) {
01301             mm.cols_list[temp_jndex] = jndex;
01302             temp_jndex++;
01303         }
01304     }
01305
01306     return mm;
01307 }
01308
01318 Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j)
01319 {
01320     MATRIX2D_ASSERT(ref_mm.cols == ref_mm.rows && "minor is defined only for square matrix");
01321
01322     Mat2D_Minor mm;
01323     mm.cols = ref_mm.cols-1;
01324     mm.rows = ref_mm.rows-1;
01325     mm.stride_r = ref_mm.cols-1;
01326     mm.cols_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mm.cols-1));
01327     mm.rows_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mm.rows-1));
01328     mm.ref_mat = ref_mm.ref_mat;
01329
01330     MATRIX2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
01331
01332     for (size_t index = 0, temp_index = 0; index < ref_mm.rows; index++) {
01333         if (index != i) {
01334             mm.rows_list[temp_index] = ref_mm.rows_list[index];
01335             temp_index++;
01336         }
01337     }
01338     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mm.rows; jndex++) {
01339         if (jndex != j) {
01340             mm.cols_list[temp_jndex] = ref_mm.cols_list[jndex];
01341             temp_jndex++;
01342         }
01343     }
01344
01345     return mm;
01346 }
01347
01353 void mat2D_minor_free(Mat2D_Minor mm)
01354 {
01355     free(mm.cols_list);
01356     free(mm.rows_list);
01357 }
01358
01365 void mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding)
```

```
01366 {
01367     printf("%*s%s = [\n", (int) padding, "", name);
01368     for (size_t i = 0; i < mm.rows; ++i) {
01369         printf("%*s    ", (int) padding, "");
01370         for (size_t j = 0; j < mm.cols; ++j) {
01371             printf("%f ", MAT2D_MINOR_AT(mm, i, j));
01372         }
01373         printf("\n");
01374     }
01375     printf("%*s]\n", (int) padding, "");
01376 }
01377
01383 double mat2D_det_2x2_mat_minor(Mat2D_Minor mm)
01384 {
01385     MATRIX2D_ASSERT(2 == mm.cols && 2 == mm.rows && "Not a 2x2 matrix");
01386     return MAT2D_MINOR_AT(mm, 0, 0) * MAT2D_MINOR_AT(mm, 1, 1) - MAT2D_MINOR_AT(mm, 0, 1) *
       MAT2D_MINOR_AT(mm, 1, 0);
01387 }
01388
01396 double mat2D_minor_det(Mat2D_Minor mm)
01397 {
01398     MATRIX2D_ASSERT(mm.cols == mm.rows && "should be a square matrix");
01399
01400     double det = 0;
01401     /* TODO: finding beast row or col? */
01402     for (size_t i = 0, j = 0; i < mm.rows; i++) { /* first column */
01403         if (MAT2D_MINOR_AT(mm, i, j) < 1e-10) continue;
01404         Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat_minor(mm, i, j);
01405         int factor = (i+j)%2 ? -1 : 1;
01406         if (sub_mm.cols != 2) {
01407             MATRIX2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
01408             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_minor_det(sub_mm);
01409         } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
01410             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);;
01411         }
01412         mat2D_minor_free(sub_mm);
01413     }
01414     return det;
01415 }
01416
01417
01418 #endif // MATRIX2D_IMPLEMENTATION
```

4.15 src/teapot_example.c File Reference

```
#include "./include/display.c"
#include "./include/Matrix2D.h"
#include "./include/Almog_Draw_Library.h"
#include "./include/Almog_Engine.h"
```

Include dependency graph for teapot_example.c:



Macros

- #define SETUP
- #define UPDATE

- #define RENDER
- #define MATRIX2D_IMPLEMENTATION
- #define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
- #define ALMOG_ENGINE_IMPLEMENTATION

Functions

- void setup (game_state_t *game_state)
- void update (game_state_t *game_state)
- void render (game_state_t *game_state)

4.15.1 Macro Definition Documentation

4.15.1.1 ALMOG_DRAW_LIBRARY_IMPLEMENTATION

```
#define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
```

Definition at line 7 of file teapot_example.c.

4.15.1.2 ALMOG_ENGINE_IMPLEMENTATION

```
#define ALMOG_ENGINE_IMPLEMENTATION
```

Definition at line 9 of file teapot_example.c.

4.15.1.3 MATRIX2D_IMPLEMENTATION

```
#define MATRIX2D_IMPLEMENTATION
```

Definition at line 5 of file teapot_example.c.

4.15.1.4 RENDER

```
#define RENDER
```

Definition at line 3 of file teapot_example.c.

4.15.1.5 SETUP

```
#define SETUP
```

Definition at line 1 of file teapot_example.c.

4.15.1.6 UPDATE

```
#define UPDATE
```

Definition at line 2 of file teapot_example.c.

4.15.2 Function Documentation

4.15.2.1 render()

```
void render (
            game_state_t * game_state )
```

Definition at line 61 of file teapot_example.c.

References ADL_DEFAULT_OFFSET_ZOOM, adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal(), Tri_mesh_array::elements, Scene::in_world_tri_meshes, game_state_t::inv_z_buffer_mat, Tri_mesh::length, Tri_mesh_array::length, Scene::projected_tri_meshes, game_state_t::scene, and game_state_t::window_pixels_mat.

4.15.2.2 setup()

```
void setup (
            game_state_t * game_state )
```

Definition at line 12 of file teapot_example.c.

References ada_append, ada_init_array, ae_tri_mesh_append_copy(), ae_tri_mesh_get_from_file(), ae_tri_mesh_normalize(), ae_tri_mesh_rotate_Euler_xyz(), Tri_mesh_array::elements, Scene::in_world_tri_meshes, Tri_mesh::length, Tri_mesh_array::length, Scene::original_tri_meshes, Scene::projected_tri_meshes, game_state_t::scene, and game_state_t::to_limit_fps.

4.15.2.3 update()

```
void update (
            game_state_t * game_state )
```

Definition at line 50 of file teapot_example.c.

References AE_LIGHTING_FLAT, ae_projection_mat_set(), ae_tri_mesh_project_world2screen(), ae_view_mat_set(), Camera::aspect_ratio, Scene::camera, Tri_mesh_array::elements, Camera::fov_deg, Scene::in_world_tri_meshes, Tri_mesh_array::length, Scene::proj_mat, Scene::projected_tri_meshes, game_state_t::scene, Scene::up_direction, Scene::view_mat, game_state_t::window_h, game_state_t::window_w, Camera::z_far, and Camera::z_near.

4.16 teapot_example.c

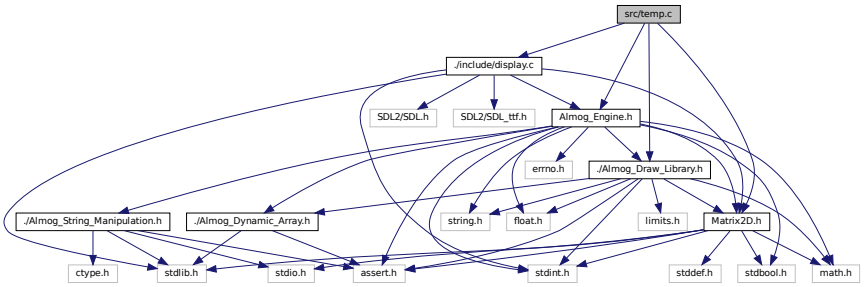
```
00001 #define SETUP
00002 #define UPDATE
00003 #define RENDER
00004 #include "./include/display.c"
00005 #define MATRIX2D_IMPLEMENTATION
00006 #include "./include/Matrix2D.h"
00007 #define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
00008 #include "./include/Almog_Draw_Library.h"
00009 #define ALMOG_ENGINE_IMPLEMENTATION
00010 #include "./include/Almog_Engine.h"
00011
00012 void setup(game_state_t *game_state)
00013 {
00014     game_state->to_limit_fps = 0;
00015
00016     ada_init_array(Tri_mesh, game_state->scene.original_tri_meshes);
00017     ada_init_array(Tri_mesh, game_state->scene.in_world_tri_meshes);
00018     ada_init_array(Tri_mesh, game_state->scene.projected_tri_meshes);
00019
00020     char file_path[MAX_LEN_LINE];
00021     strncpy(file_path, "./teapot.stl", MAX_LEN_LINE);
00022
00023     Tri_mesh teapot_mesh = ae_tri_mesh_get_from_file(file_path);
00024     // ae_tri_mesh_flip_normals(teapot_mesh);
00025     ada_append(Tri_mesh, game_state->scene.original_tri_meshes, teapot_mesh);
00026
00027     printf("[INFO] number of meshes: %zu\n", game_state->scene.original_tri_meshes.length);
00028     size_t sum = 0;
00029     for (size_t i = 0; i < game_state->scene.original_tri_meshes.length; i++) {
00030         printf("[INFO] mesh number %zu: %zu\n", i,
       game_state->scene.original_tri_meshes.elements[i].length);
00031         sum += game_state->scene.original_tri_meshes.elements[i].length;
00032     }
00033     printf("[INFO] total number of triangles: %zu\n", sum);
00034
00035
00036     for (size_t i = 0; i < game_state->scene.original_tri_meshes.length; i++) {
00037         ae_tri_mesh_normalize(game_state->scene.original_tri_meshes.elements[i]);
00038     }
00039     for (size_t i = 0; i < game_state->scene.original_tri_meshes.length; i++) {
00040         ae_tri_mesh_append_copy(&(game_state->scene.in_world_tri_meshes),
       game_state->scene.original_tri_meshes.elements[i]);
00041         ae_tri_mesh_append_copy(&(game_state->scene.projected_tri_meshes),
       game_state->scene.original_tri_meshes.elements[i]);
00042         game_state->scene.projected_tri_meshes.elements[i].length = 0;
00043     }
00044
00045     ae_tri_mesh_rotate_Euler_xyz(game_state->scene.in_world_tri_meshes.elements[0], -90, 0, 180);
00046
00047     // ae_translate_mesh(game_state->scene.in_world_tri_meshes.elements[0], 0, 0, 2);
00048 }
00049
00050 void update(game_state_t *game_state)
00051 {
00052     ae_projection_mat_set(game_state->scene.proj_mat, game_state->scene.camera.aspect_ratio,
       game_state->scene.camera.fov_deg, game_state->scene.camera.z_near, game_state->scene.camera.z_far);
00053     ae_view_mat_set(game_state->scene.view_mat, game_state->scene.camera,
       game_state->scene.up_direction);
00054
00055     for (size_t i = 0; i < game_state->scene.in_world_tri_meshes.length; i++) {
00056         ae_tri_mesh_project_world2screen(game_state->scene.proj_mat, game_state->scene.view_mat,
       &(game_state->scene.projected_tri_meshes.elements[i]),
       game_state->scene.in_world_tri_meshes.elements[i], game_state->window_w, game_state->window_h,
       &(game_state->scene), AE_LIGHTING_FLAT);
```

```
00057     }
00058
00059 }
00060
00061 void render(game_state_t *game_state)
00062 {
00063     for (size_t i = 0; i < game_state->scene.projected_tri_meshes.length; i++) {
00064         adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal(game_state->window_pixels_mat,
      game_state->inv_z_buffer_mat, game_state->scene.projected_tri_meshes.elements[i], 0xffffffff,
      ADL_DEFAULT_OFFSET_ZOOM);
00065     }
00066
00067     for (size_t i = 0; i < game_state->scene.in_world_tri_meshes.length; i++) {
00068         game_state->scene.projected_tri_meshes.elements[i].length = 0;
00069     }
00070 }
```

4.17 src/temp.c File Reference

```
#include "./include/display.c"
#include "./include/Matrix2D.h"
#include "./include/Almog_Draw_Library.h"
#include "./include/Almog_Engine.h"
```

Include dependency graph for temp.c:



Macros

- #define SETUP
- #define UPDATE
- #define RENDER
- #define MATRIX2D_IMPLEMENTATION
- #define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
- #define ALMOG_ENGINE_IMPLEMENTATION

Functions

- void setup (game_state_t *game_state)
- void update (game_state_t *game_state)
- void render (game_state_t *game_state)

4.17.1 Macro Definition Documentation

4.17.1.1 ALMOG_DRAW_LIBRARY_IMPLEMENTATION

```
#define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
```

Definition at line 7 of file temp.c.

4.17.1.2 ALMOG_ENGINE_IMPLEMENTATION

```
#define ALMOG_ENGINE_IMPLEMENTATION
```

Definition at line 9 of file temp.c.

4.17.1.3 MATRIX2D_IMPLEMENTATION

```
#define MATRIX2D_IMPLEMENTATION
```

Definition at line 5 of file temp.c.

4.17.1.4 RENDER

```
#define RENDER
```

Definition at line 3 of file temp.c.

4.17.1.5 SETUP

```
#define SETUP
```

Definition at line 1 of file temp.c.

4.17.1.6 UPDATE

```
#define UPDATE
```

Definition at line 2 of file temp.c.

4.17.2 Function Documentation

4.17.2.1 render()

```
void render (
            game_state_t * game_state )
```

Definition at line 63 of file temp.c.

References ADL_DEFAULT_OFFSET_ZOOM, adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal(), Tri_mesh_array::elements, game_state_t::inv_z_buffer_mat, Tri_mesh_array::length, Scene::projected_tri_meshes, game_state_t::scene, and game_state_t::window_pixels_mat.

Referenced by render_window().

4.17.2.2 setup()

```
void setup (
            game_state_t * game_state )
```

Definition at line 12 of file temp.c.

References ada_append, ada_init_array, ae_tri_mesh_append_copy(), ae_tri_mesh_get_from_file(), ae_tri_mesh_normalize(), ae_tri_mesh_rotate_Euler_xyz(), ASM_MAX_LEN_LINE, Tri_mesh_array::elements, Scene::in_world_tri_meshes, Tri_mesh::length, Tri_mesh_array::length, Scene::original_tri_meshes, Scene::projected_tri_meshes, game_state_t::scene, and game_state_t::to_limit_fps.

Referenced by setup_window().

4.17.2.3 update()

```
void update (
            game_state_t * game_state )
```

Definition at line 49 of file temp.c.

References AE_LIGHTING_FLAT, ae_projection_mat_set(), ae_tri_mesh_project_world2screen(), ae_view_mat_set(), Camera::aspect_ratio, Scene::camera, Tri_mesh_array::elements, Camera::fov_deg, Scene::in_world_tri_meshes, Tri_mesh_array::length, Scene::proj_mat, Scene::projected_tri_meshes, game_state_t::scene, Scene::up_direction, Scene::view_mat, game_state_t::window_h, game_state_t::window_w, Camera::z_far, and Camera::z_near.

Referenced by update_window().

4.18 temp.c

```
00001 #define SETUP
00002 #define UPDATE
00003 #define RENDER
00004 #include "./include/display.c"
00005 #define MATRIX2D_IMPLEMENTATION
00006 #include "./include/Matrix2D.h"
00007 #define ALMOG_DRAW_LIBRARY_IMPLEMENTATION
00008 #include "./include/Almog_Draw_Library.h"
00009 #define ALMOG_ENGINE_IMPLEMENTATION
00010 #include "./include/Almog_Engine.h"
00011
00012 void setup(game_state_t *game_state)
00013 {
00014     game_state->to_limit_fps = 0;
00015
00016     ada_init_array(Tri_mesh, game_state->scene.original_tri_meshes);
00017     ada_init_array(Tri_mesh, game_state->scene.in_world_tri_meshes);
00018     ada_init_array(Tri_mesh, game_state->scene.projected_tri_meshes);
00019
00020     char file_path[ASM_MAX_LEN_LINE];
00021     strncpy(file_path, "./teapot.stl", ASM_MAX_LEN_LINE);
00022
00023     Tri_mesh tri_mesh = ae_tri_mesh_get_from_file(file_path);
00024
00025     ada_append(Tri_mesh, game_state->scene.original_tri_meshes, tri_mesh);
00026
00027     printf("[INFO] number of meshes: %zu\n", game_state->scene.original_tri_meshes.length);
00028     size_t sum = 0;
00029     for (size_t i = 0; i < game_state->scene.original_tri_meshes.length; i++) {
00030         printf("[INFO] mesh number %zu: %zu\n", i,
      game_state->scene.original_tri_meshes.elements[i].length);
00031         sum += game_state->scene.original_tri_meshes.elements[i].length;
00032     }
00033     printf("[INFO] total number of triangles: %zu\n", sum);
00034
00035
00036     for (size_t i = 0; i < game_state->scene.original_tri_meshes.length; i++) {
00037         ae_tri_mesh_normalize(game_state->scene.original_tri_meshes.elements[i]);
00038     }
00039     for (size_t i = 0; i < game_state->scene.original_tri_meshes.length; i++) {
00040         ae_tri_mesh_append_copy(&(game_state->scene.in_world_tri_meshes),
      game_state->scene.original_tri_meshes.elements[i]);
00041         ae_tri_mesh_append_copy(&(game_state->scene.projected_tri_meshes),
      game_state->scene.original_tri_meshes.elements[i]);
00042         game_state->scene.projected_tri_meshes.elements[i].length = 0;
00043     }
00044
00045     ae_tri_mesh_rotate_Euler_xyz(game_state->scene.in_world_tri_meshes.elements[0], -90, 0, 180);
00046
00047 }
00048
00049 void update(game_state_t *game_state)
00050 {
00051     // MAT2D_PRINT(game_state->scene.camera.current_position);
00052     // MAT2D_PRINT(game_state->scene.light_direction);
00053
00054     ae_projection_mat_set(game_state->scene.proj_mat, game_state->scene.camera.aspect_ratio,
      game_state->scene.camera.fov_deg, game_state->scene.camera.z_near, game_state->scene.camera.z_far);
00055     ae_view_mat_set(game_state->scene.view_mat, game_state->scene.camera,
      game_state->scene.up_direction);
00056
00057     for (size_t i = 0; i < game_state->scene.in_world_tri_meshes.length; i++) {
00058         ae_tri_mesh_project_world2screen(game_state->scene.proj_mat, game_state->scene.view_mat,
      &(game_state->scene.projected_tri_meshes.elements[i]),
      game_state->scene.in_world_tri_meshes.elements[i], game_state->window_w, game_state->window_h,
      &(game_state->scene), AE_LIGHTING_FLAT);
00059     }
00060
00061 }
00062
00063 void render(game_state_t *game_state)
00064 {
00065     for (size_t i = 0; i < game_state->scene.projected_tri_meshes.length; i++) {
00066         adl_tri_mesh_fill_Pinedas_rasterizer_interpolate_normal(game_state->window_pixels_mat,
      game_state->inv_z_buffer_mat, game_state->scene.projected_tri_meshes.elements[i], 0xffffffff,
      ADL_DEFAULT_OFFSET_ZOOM);
00067     }
00068 }
```

Index